Greedy Algorithms

T. M. Murali

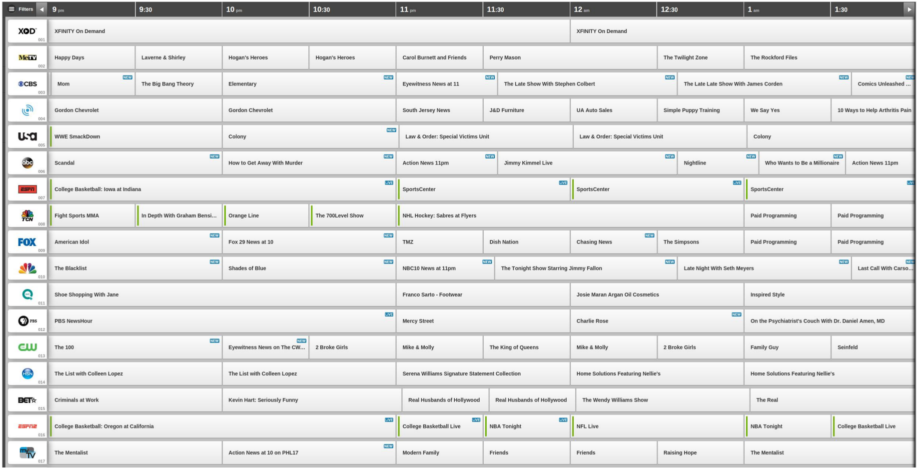
February 11, 16, 2016

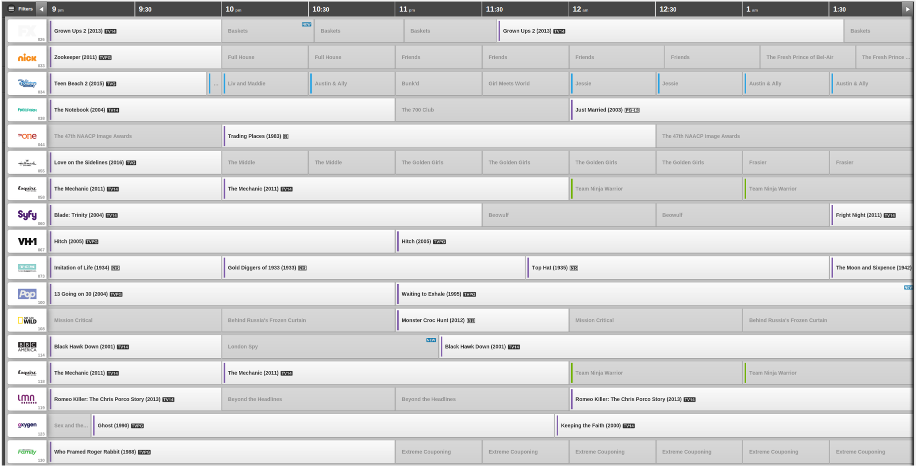
Algorithm Design

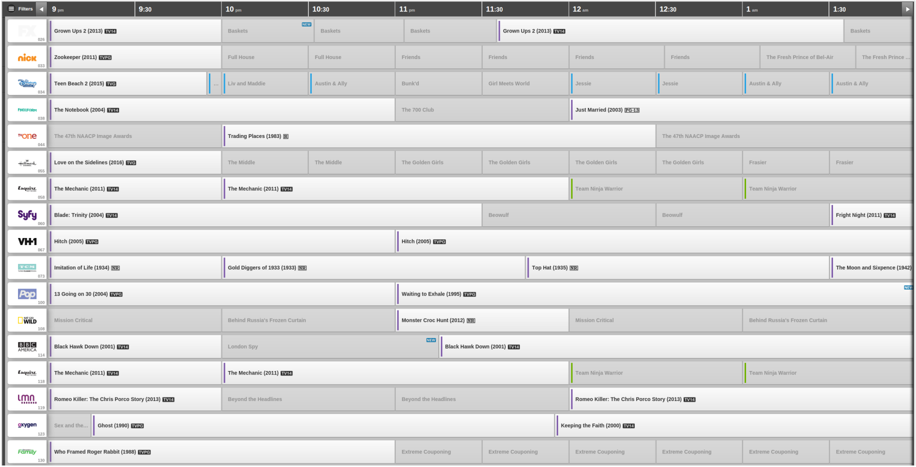
- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.

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.


Filters
9 pm
9:30
10 pm
10:30
11 pm
11:30
12 am
12:30
1 am
1:30
XFINITY On Demand
XFINITY On Demand
Happy Days
Laverne & Shirley
Hogan's Heroes
Hogan's Heroes
Carol Burnett and Friends
Perry Mason
The Twilight Zone
The Rockford Files
Mom
The Big Bang Theory
Elementary
Eyewitness News at 11
The Late Show With Stephen Colbert
The Late Late Show With James Corden
Comics Unleashed ...
Gordon Chevrolet
Gordon Chevrolet
South Jersey News
J&D Furniture
UA Auto Sales
Simple Puppy Training
We Say Yes
10 Ways to Help Arthritis Pain
WWE SmackDown
Colony
Law & Order: Special Victims Unit
Law & Order: Special Victims Unit
Colony
Scandal
How to Get Away With Murder
Action News 11pm
Jimmy Kimmel Live
Nightline
Who Wants to Be a Millionaire
Action News 11pm
College Basketball: Iowa at Indiana
SportsCenter
SportsCenter
SportsCenter
Fight Sports MMA
In Depth With Graham Bensi...
Orange Line
The 700Level Show
NHL Hockey: Sabres at Flyers
Paid Programming
Paid Programming
American Idol
Fox 29 News at 10
TMZ
Dish Nation
Chasing News
The Simpsons
Paid Programming
Paid Programming
The Blacklist
Shades of Blue
NBC10 News at 11pm
The Tonight Show Starring Jimmy Fallon
Late Night With Seth Meyers
Last Call With Carso...
Shoe Shopping With Jane
Franco Sarto - Footwear
Josie Maran Argan Oil Cosmetics
Inspired Style
PBS NewsHour
Mercy Street
Charlie Rose
On the Psychiatrist's Couch With Dr. Daniel Amen, MD
The 100
Eyewitness News on The CW...
2 Broke Girls
Mike & Molly
The King of Queens
Mike & Molly
2 Broke Girls
Family Guy
Seinfeld
The List with Colleen Lopez
The List with Colleen Lopez
Serena Williams Signature Statement Collection
Home Solutions Featuring Nellie's
Home Solutions Featuring Nellie's
Criminals at Work
Kevin Hart: Seriously Funny
Real Husbands of Hollywood
Real Husbands of Hollywood
The Wendy Williams Show
The Real
College Basketball: Oregon at California
College Basketball Live
NBA Tonight
NFL Live
NBA Tonight
College Basketball Live
The Mentalist
Action News at 10 on PHL17
Modern Family
Friends
Friends
Raising Hope
The Mentalist


Filters
9 pm
9:30
10 pm
10:30
11 pm
11:30
12 am
12:30
1 am
1:30
FX 026
Grown Ups 2 (2013) TV14
Baskets
NEW
Baskets
Baskets
Grown Ups 2 (2013) TV14
Baskets
nick 033
Zookeeper (2011) TVPG
Full House
Full House
Friends
Friends
Friends
Friends
The Fresh Prince of Bel-Air
The Fresh Prince ...
034
Teen Beach 2 (2015) TVG
...
Liv and Maddie
Austin & Ally
Bunk'd
Girl Meets World
Jessie
Jessie
Austin & Ally
Austin & Ally
FREEFORM 038
The Notebook (2004) TV14
The 700 Club
Just Married (2003) PG-13
TVone 044
The 47th NAACP Image Awards
Trading Places (1983) R
The 47th NAACP Image Awards
055
Love on the Sidelines (2016) TVG
The Middle
The Middle
The Golden Girls
The Golden Girls
The Golden Girls
The Golden Girls
Frasier
Frasier
Esquire 058
The Mechanic (2011) TV14
The Mechanic (2011) TV14
Team Ninja Warrior
Team Ninja Warrior
Syfy 060
Blade: Trinity (2004) TV14
Beowulf
Beowulf
Fright Night (2011) TV14
VH1 067
Hitch (2005) TVPG
Hitch (2005) TVPG
TCM 073
Imitation of Life (1934) NR
Gold Diggers of 1933 (1933) NR
Top Hat (1935) NR
The Moon and Sixpence (1942)
Pop 100
13 Going on 30 (2004) TVPG
Waiting to Exhale (1995) TVPG
NEW
WILD 108
Mission Critical
Behind Russia's Frozen Curtain
Monster Croc Hunt (2012) NR
Mission Critical
Behind Russia's Frozen Curtain
BBC AMERICA 114
Black Hawk Down (2001) TV14
London Spy
NEW
Black Hawk Down (2001) TV14
Esquire 118
The Mechanic (2011) TV14
The Mechanic (2011) TV14
Team Ninja Warrior
Team Ninja Warrior
lmn 119
Romeo Killer: The Chris Porco Story (2013) TV14
Beyond the Headlines
Beyond the Headlines
Romeo Killer: The Chris Porco Story (2013) TV14
oxygen 123
Sex and the...
Ghost (1990) TVPG
Keeping the Faith (2000) TV14
Family 130
Who Framed Roger Rabbit (1988) TVPG
Extreme Couponing
Extreme Couponing
Extreme Couponing
Extreme Couponing
Extreme Couponing
Extreme Couponing



- Input: Start and end time of each movie.
- Constraint: Only one TV $\Rightarrow$ cannot watch two overlapping classes at the same time.
- Output: Compute the largest number of movies we can watch.

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

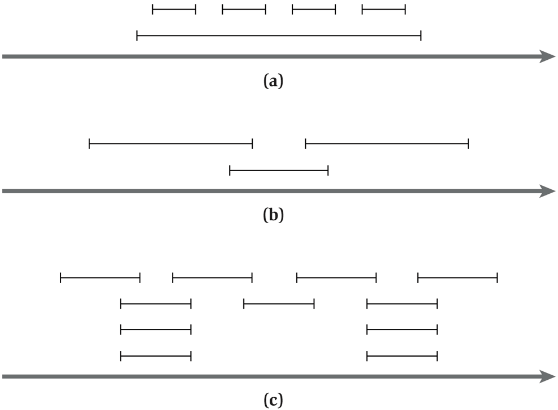


Figure 4.1 Some instances of the Interval Scheduling Problem on which natural greedy algorithms fail to find the optimal solution. In (a), it does not work to select the interval that starts earliest; in (b), it does not work to select the shortest interval; and in (c), it does not work to select the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

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 jobs. Proof follows by construction, i.e., the algorithm computes a compatible set of 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.

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 jobs in any solution in increasing order of their finishing time.
  - Finishing time of job number r selected by $A \leq$ finishing time of job number r selected by any other algorithm.

Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs 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 jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs 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 jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs 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.

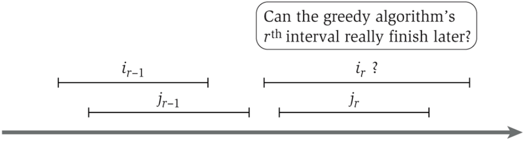


Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs 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.

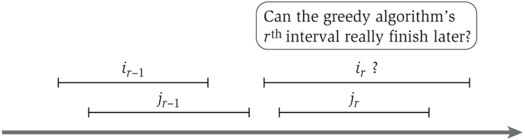


Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

- Claim: $m = k$.

Analysing the EFT Algorithm

- Let O be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs 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.

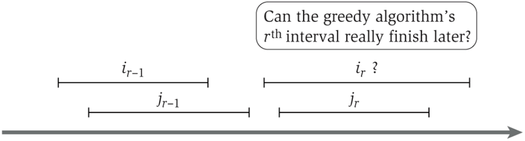


Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

- 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. $k = 1$.
4. While $k \leq |S|$,
   4.1 Output job k.
   4.2 Let finish time of job k be f.
   4.3 Iterate over S from index k onwards to find the first index i such that $S[i] \geq f$.
   4.4 $k = i$

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. $k = 1$.
4. While $k \leq |S|$,
   4.1 Output job k.
   4.2 Let finish time of job k be f.
   4.3 Iterate over S from index k onwards to find the first index i such that $S[i] \geq f$.
   4.4 $k = i$

- Must be careful to iterate over S such that we never scan same index more than once.
- Running time is $O(n \log n)$, dominated by sorting. advance from the

12528	CS-3604	Professionalism in Computing	L	3	Full -5	54	DR Dunlap	T R	11:00AM	12:15PM	PAM 32	11T
	Comments for CRN 12528:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
20110	CS-3634	Comp Sci Foundations for CMDA	L	3	Full -8	30	T Warburton	T R	5:00PM	6:15PM	SAUND 408	17T
	Comments for CRN 20110:	Not for CS major credit. To submit a force/add request, send e-mail to Ms. Nora Sullivan: nora84@vt.edu										
12529	CS-3654	Intro Data Analytics & Visual	L	3	2	60	CL North	M W	4:00PM	5:15PM	WLH 350	16M
	Comments for CRN 12529:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
19527	CS-3704	Intermed Software Des	L	3	33	75	FJ Servant Cortes	M W	2:30PM	3:45PM	WLH 350	14M
	Comments for CRN 19527:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12530	CS-3714	Mobile Software Development	L	3	Full -12	65	DS McCrickard	T R	3:30PM	4:45PM	MCB 113	15T
	Comments for CRN 12530:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12531	CS-3724	Human-Computr Intrctn	L	3	Full -24	50	SR Harrison	T R	12:30PM	1:45PM	SURGE 109	12T
	Comments for CRN 12531:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12533	CS-3744	Intro GUI Programming/Graphics	L	3	Full -9	75	K Luther	T R	5:00PM	6:15PM	MCB 129	17T
	Comments for CRN 12533:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12534	CS-4104	Data and Algorithm Analysis	L	3	Full -1	65	TM Murali	T R	2:00PM	3:15PM	GYM 124	14T
	Comments for CRN 12534:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12535	CS-4114	Formal Languages	L	3	Full -16	50	L Zhang	T R	3:30PM	4:45PM	TORG 1060	15T
	Comments for CRN 12535:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12537	CS-4284	Systems & Networking Capstone	L	3	5	29	AR Butt	M W	2:30PM	3:45PM	RAND 222	14M
	Comments for CRN 12537:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
19512	CS-4504	Computer Organization	L	3	2	36	W Feng	T R	12:30PM	1:45PM	WHIT 257	12T
	Comments for CRN 19512:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12540	CS-4604	Int Data Base Mgt Sys	L	3	7	60	BA Prakash	M W	4:00PM	5:15PM	WLH 340	16M
	Comments for CRN 12540:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12541	CS-4624	Multimedia/Hypertext	L	3	Full -11	45	EA Fox	T R	9:30AM	10:45AM	WLH 340	09T
	Comments for CRN 12541:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
20098	CS-4644	Creative Computing Studio	L	3	Full -10	20	A Kelliher	T R	11:00AM	12:15PM	MCB 238	11T

12528	CS-3604	Professionalism in Computing	L	3	Full -5	54	DR Dunlap	T R	11:00AM	12:15PM	PAM 32	11T
	Comments for CRN 12528:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
20110	CS-3634	Comp Sci Foundations for CMDA	L	3	Full -8	30	T Warburton	T R	5:00PM	6:15PM	SAUND 408	17T
	Comments for CRN 20110:	Not for CS major credit. To submit a force/add request, send e-mail to Ms. Nora Sullivan: nora84@vt.edu										
12529	CS-3654	Intro Data Analytics & Visual	L	3	2	60	CL North	M W	4:00PM	5:15PM	WLH 350	16M
	Comments for CRN 12529:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
19527	CS-3704	Intermed Software Des	L	3	33	75	FJ Servant Cortes	M W	2:30PM	3:45PM	WLH 350	14M
	Comments for CRN 19527:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12530	CS-3714	Mobile Software Development	L	3	Full -12	65	DS McCrickard	T R	3:30PM	4:45PM	MCB 113	15T
	Comments for CRN 12530:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12531	CS-3724	Human-Computr Intrctn	L	3	Full -24	50	SR Harrison	T R	12:30PM	1:45PM	SURGE 109	12T
	Comments for CRN 12531:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12533	CS-3744	Intro GUI Programming/Graphics	L	3	Full -9	75	K Luther	T R	5:00PM	6:15PM	MCB 129	17T
	Comments for CRN 12533:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12534	CS-4104	Data and Algorithm Analysis	L	3	Full -1	65	TM Murali	T R	2:00PM	3:15PM	GYM 124	14T
	Comments for CRN 12534:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12535	CS-4114	Formal Languages	L	3	Full -16	50	L Zhang	T R	3:30PM	4:45PM	TORG 1060	15T
	Comments for CRN 12535:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12537	CS-4284	Systems & Networking Capstone	L	3	5	29	AR Butt	M W	2:30PM	3:45PM	RAND 222	14M
	Comments for CRN 12537:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
19512	CS-4504	Computer Organization	L	3	2	36	W Feng	T R	12:30PM	1:45PM	WHIT 257	12T
	Comments for CRN 19512:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12540	CS-4604	Int Data Base Mgt Sys	L	3	7	60	BA Prakash	M W	4:00PM	5:15PM	WLH 340	16M
	Comments for CRN 12540:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
12541	CS-4624	Multimedia/Hypertext	L	3	Full -11	45	EA Fox	T R	9:30AM	10:45AM	WLH 340	09T
	Comments for CRN 12541:	Force/add subject to availability; for information, see www.cs.vt.edu/undergraduate/survey.										
20098	CS-4644	Creative Computing Studio	L	3	Full -10	20	A Kelliher	T R	11:00AM	12:15PM	MCB 238	11T

- Input: Start and end time of each class.
- Constraint: Cannot schedule two overlapping classes to the same room.
- Output: Assign each class to a room and use smallest number of rooms possible.

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

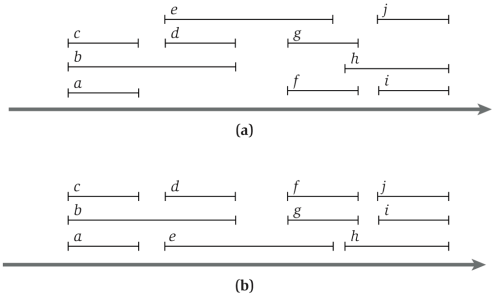


Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (*a* through *j*). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

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

Depth of Intervals

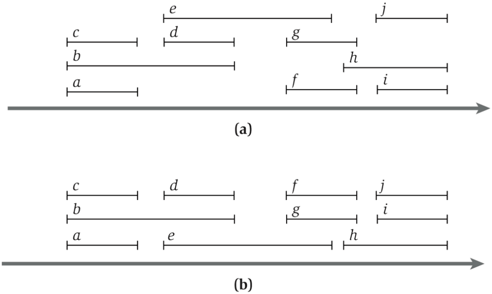


Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (*a* through *j*). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

- 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

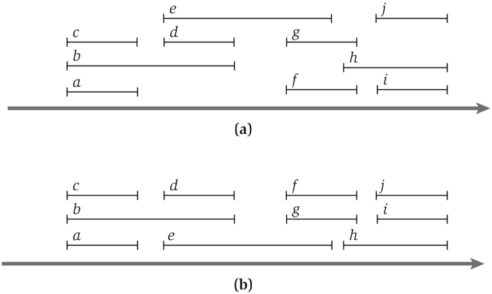


Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (*a* through *j*). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

- 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 the depth 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 $O(n \log n)$.

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 $O(n \log n)$. Can modify algorithm for computing depth to maintain set of available labels and to assign them efficiently.

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all n 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.

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all n 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

$$\max_{1 \leq i \leq n} \Big(\max \big(0, f(i) - d(i)\big)\Big).$$

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all n 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

$$\max_{1 \leq i \leq n} \big(\max \left(0, f(i) - d(i)\right)\big).$$

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 $\max_{1 \leq i \leq n} \big(\max \left(0, s(i) + t(i) - d(i)\right)\big)$ is as small as possible.

Scheduling to Minimise Lateness

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 $\max_{1 \leq i \leq n} \big(\max \big(0, s(i) + t(i) - d(i)\big)\big)$ is as small as possible.

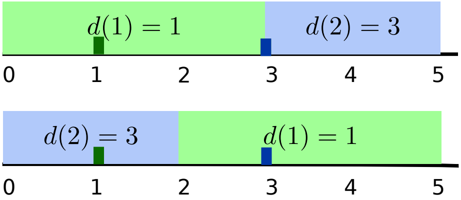
Scheduling to Minimise Lateness

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 $\max_{1 \leq i \leq n} \big(\max \big(0, s(i) + t(i) - d(i)\big)\big)$ is as small as possible.

i	1	2
$t(i)$	3	2
$d(i)$	1	3



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 $d(i)$.

Template for Greedy Algorithm

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

  Shortest length Increasing order of length $t(i)$. Ignores deadlines completely! Shortest job may have a very late deadline.

i	1	2
$t(i)$	1	10
$d(i)$	100	10

  Shortest slack time Increasing order of $d(i) - t(i)$.

  Earliest deadline Increasing order of deadline $d(i)$.

Template for Greedy Algorithm

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

  Shortest length Increasing order of length $t(i)$. Ignores deadlines completely! Shortest job may have a very late deadline.

i	1	2
$t(i)$	1	10
$d(i)$	100	10

  Shortest slack time Increasing order of $d(i) - t(i)$. Job with smallest slack may take a long time.

i	1	2
$t(i)$	1	10
$d(i)$	2	10

  Earliest deadline Increasing order of deadline $d(i)$.

Template for Greedy Algorithm

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

  Shortest length Increasing order of length $t(i)$. Ignores deadlines completely! Shortest job may have a very late deadline.

i	1	2
$t(i)$	1	10
$d(i)$	100	10

  Shortest slack time Increasing order of $d(i) - t(i)$. Job with smallest slack may take a long time.

i	1	2
$t(i)$	1	10
$d(i)$	2	10

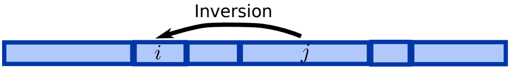
  Earliest deadline Increasing order of deadline $d(i)$. Correct? Does it make sense to tackle jobs with earliest deadlines first?

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
```

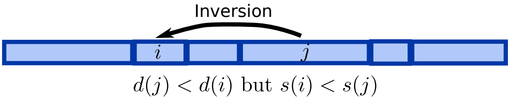
- Proof of correctness is more complex.
- We will use an exchange argument: gradually modify the optimal schedule O till it is the same as the schedule A computed by the algorithm.

Properties of Schedules



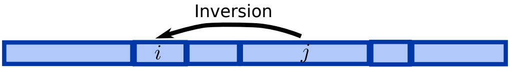
- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.

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.

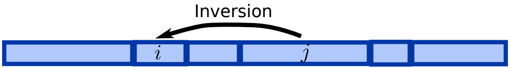
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.

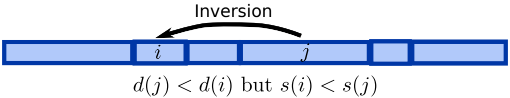
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

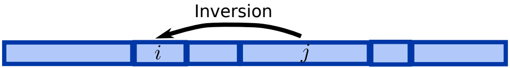
- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle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

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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.

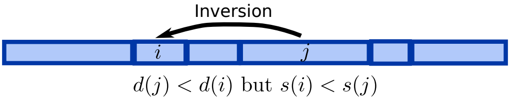
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

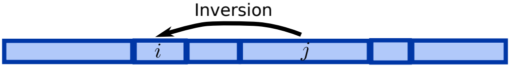
- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle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

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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

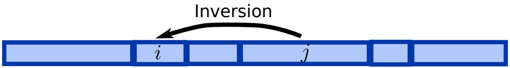
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
- Claim 3: There is an optimal schedule with no idle time.

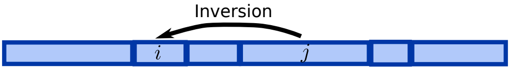
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
- 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.

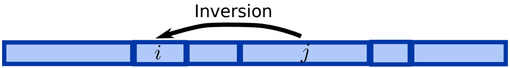
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
- 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.?!

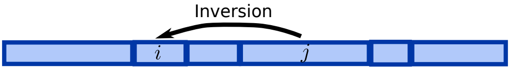
Properties of Schedules



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
- 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



$d(j) < d(i)$ but $s(i) < s(j)$

- 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.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
- 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 lateness of O' is no larger than the 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 lateness is no larger than that of O.

Swapping Inverted Jobs

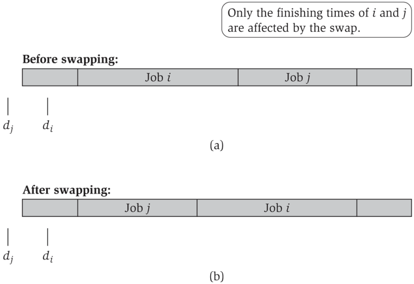


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

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

Swapping Inverted Jobs

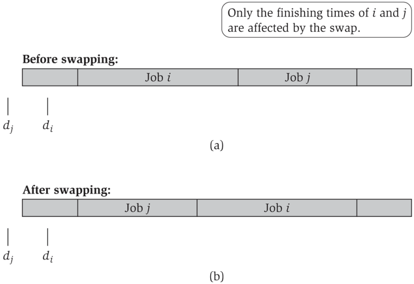


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

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

Swapping Inverted Jobs

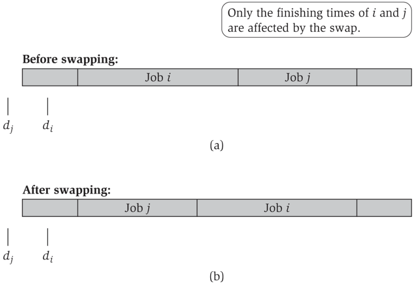


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

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

Swapping Inverted Jobs

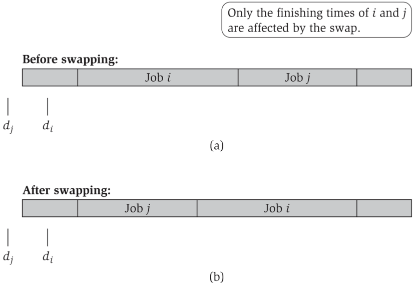


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

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

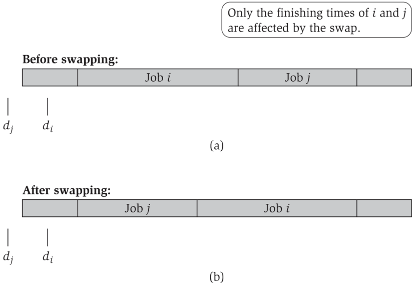


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each job r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of job 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 $l'(i) = f(j) - d_i \leq f(j) - d_j = l(j)$.

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie?

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie?

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie?

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$
5. Find an inversion in X and then isolate the inversion to be between consecutive jobs in X.
6. Swap the jobs to get a new schedule X_{i-1}. Where does X_{i-1} lie?

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$
5. Find an inversion in X and then isolate the inversion to be between consecutive jobs in X.
6. Swap the jobs to get a new schedule X_{i-1}. Where does X_{i-1} lie? X_{i-1} has one fewer inversion! Lateness cannot increase! So X_{i-1} is at $(i-1, l_X)$ or "below."
7. Repeat until we have X_1 with one inversion at

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$
5. Find an inversion in X and then isolate the inversion to be between consecutive jobs in X.
6. Swap the jobs to get a new schedule X_{i-1}. Where does X_{i-1} lie? X_{i-1} has one fewer inversion! Lateness cannot increase! So X_{i-1} is at $(i-1, l_X)$ or "below."
7. Repeat until we have X_1 with one inversion at $(1, l_X)$ or "below", where $l_X < l_A$.
8. Repeat one more step: X_0 has no inversions. What is X_0's location?

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$
5. Find an inversion in X and then isolate the inversion to be between consecutive jobs in X.
6. Swap the jobs to get a new schedule X_{i-1}. Where does X_{i-1} lie? X_{i-1} has one fewer inversion! Lateness cannot increase! So X_{i-1} is at $(i-1, l_X)$ or "below."
7. Repeat until we have X_1 with one inversion at $(1, l_X)$ or "below", where $l_X < l_A$.
8. Repeat one more step: X_0 has no inversions. What is X_0's location? $(0, l_X)$ or "below" because of #7 and $(0, l_A)$ because of #3.

Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule A produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule B with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let X be any schedule that is supposed to be optimal (and better than A). Where does X lie? At some point (i, l_X), where $i > 0$ and l_X are the number of inversions in and lateness of X, respectively. $l_X < l_A$
5. Find an inversion in X and then isolate the inversion to be between consecutive jobs in X.
6. Swap the jobs to get a new schedule X_{i-1}. Where does X_{i-1} lie? X_{i-1} has one fewer inversion! Lateness cannot increase! So X_{i-1} is at $(i-1, l_X)$ or "below."
7. Repeat until we have X_1 with one inversion at $(1, l_X)$ or "below", where $l_X < l_A$.
8. Repeat one more step: X_0 has no inversions. What is X_0's location? $(0, l_X)$ or "below" because of #7 and $(0, l_A)$ because of #3.
9. We have a contradiction!
10. Lateness of A cannot be larger than that of O!

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 is essentially the same as the one the algorithm produces without increasing the lateness.

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 is essentially the same as the one the algorithm produces without increasing the lateness.
- 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.

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.