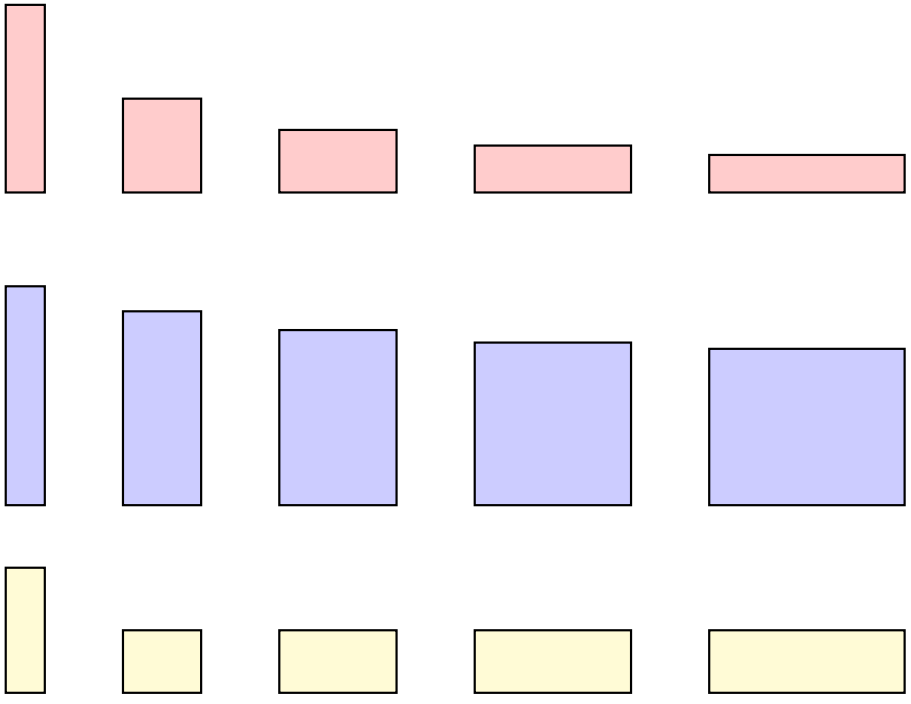
Scheduling Monotone Moldable Jobs in Linear Time

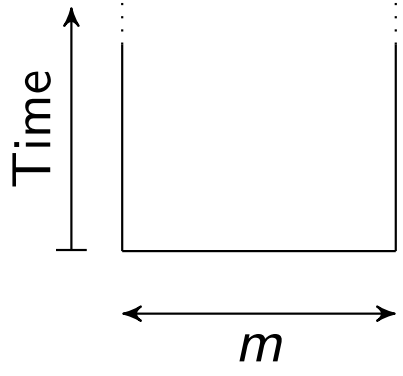
K. Grage *K. Jansen* F. Land F. Ohnesorge

Kiel University
Faculty of Engineering
Department of Computer Science

Moldable Job Scheduling

- n jobs
- m machines
- job j has processing time $t_j(k)$ when processed on k machines
  work $w_j(k) = k \times t_j(k)$

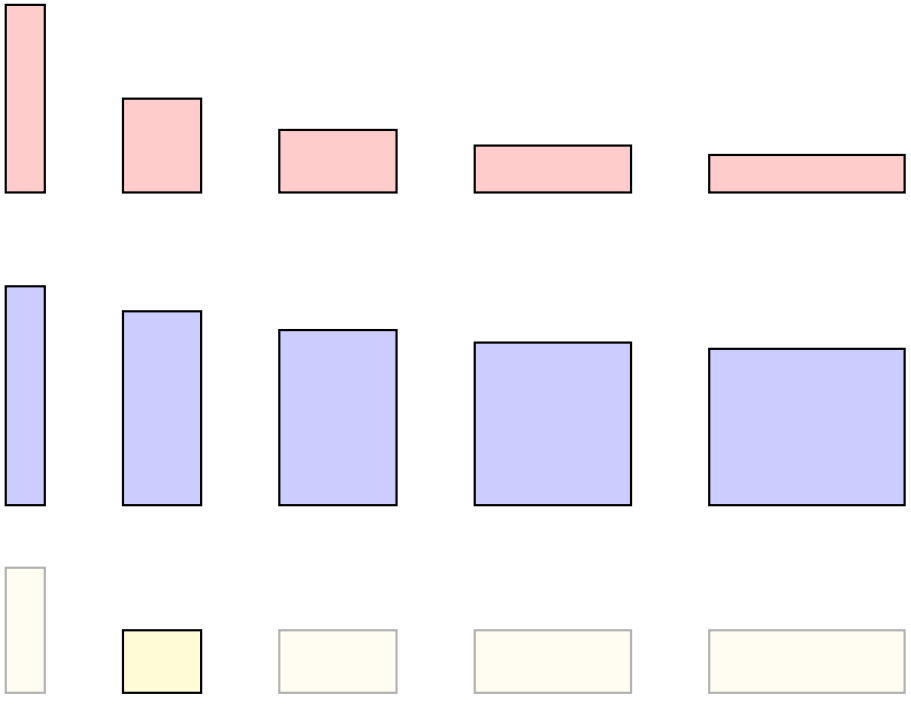


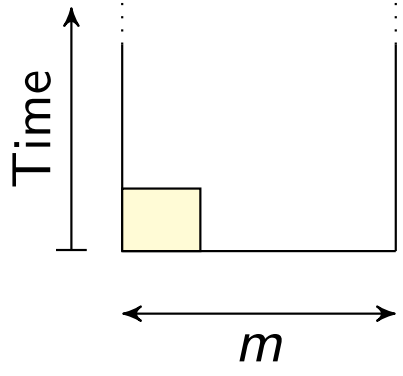


no preemptions
minimize makespan

Moldable Job Scheduling

- n jobs
- m machines
- job j has processing time $t_j(k)$ when processed on k machines
  work $w_j(k) = k \times t_j(k)$

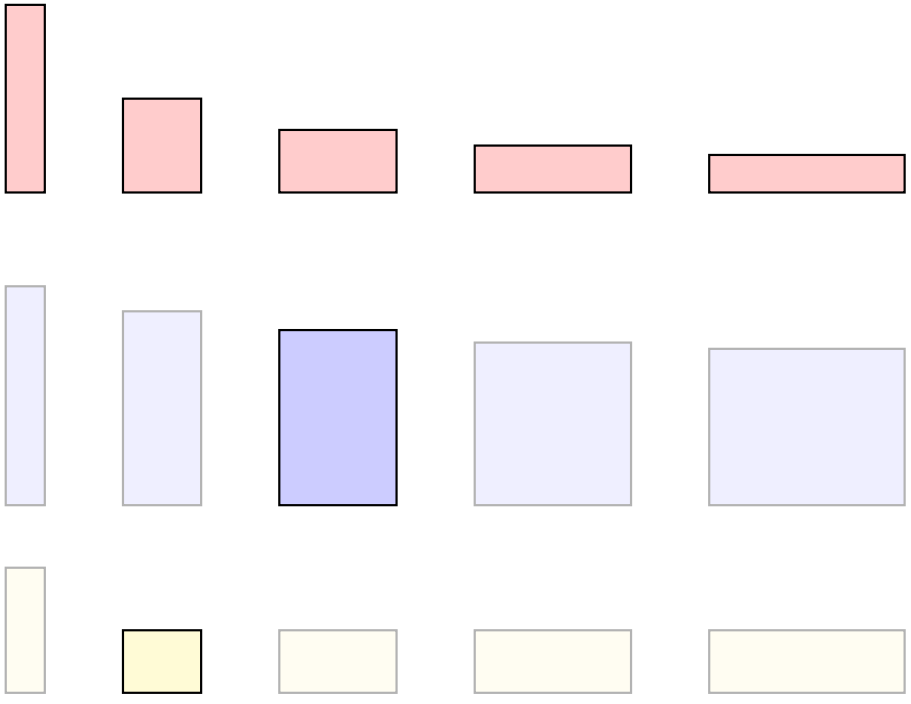


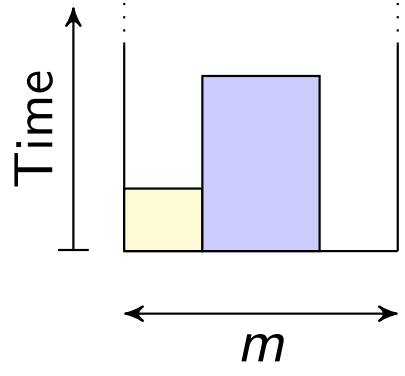


no preemptions
minimize makespan

Moldable Job Scheduling

- n jobs
- m machines
- job j has processing time $t_j(k)$ when processed on k machines
  work $w_j(k) = k \times t_j(k)$

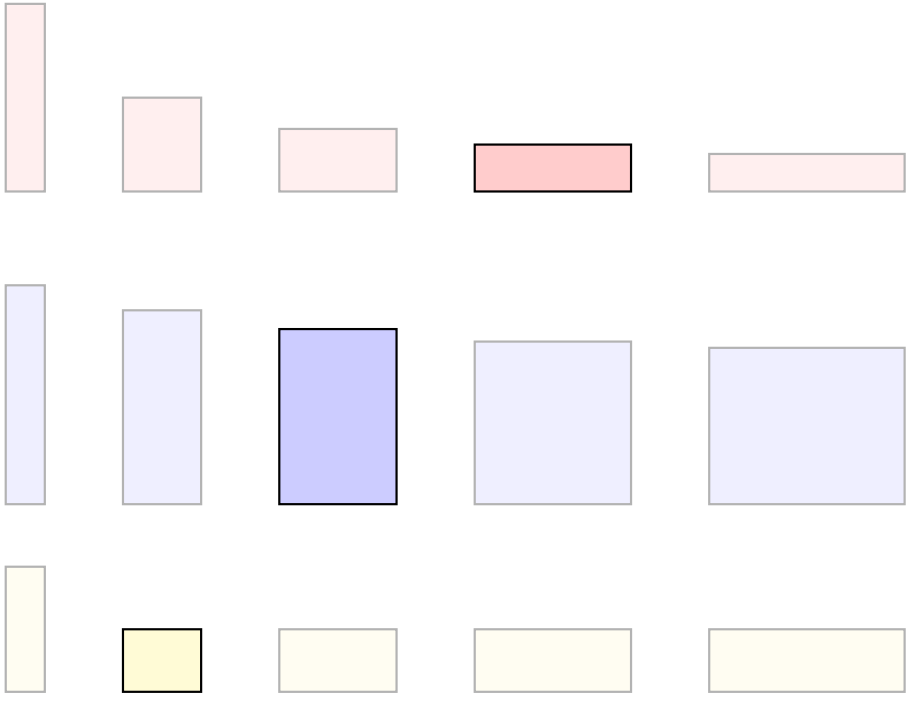


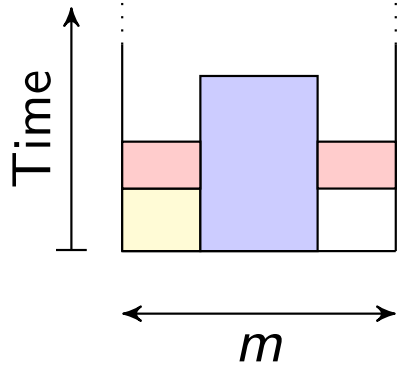


no preemptions
minimize makespan

Moldable Job Scheduling

- n jobs
- m machines
- job j has processing time $t_j(k)$ when processed on k machines
  work $w_j(k) = k \times t_j(k)$



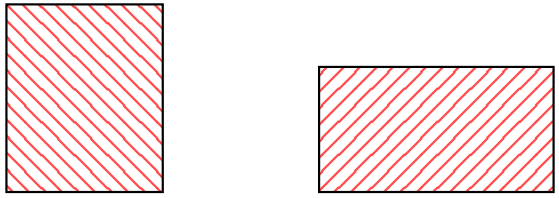


no preemptions
minimize makespan

Monotone Jobs

Definition

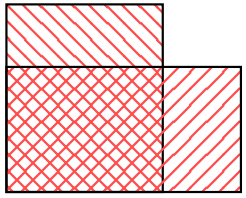
A job j is monotone iff its work function $w_j(k) = k \times t_j(k)$ is non-decreasing.



Monotone Jobs

Definition

A job j is monotone iff its work function $w_j(k) = k \times t_j(k)$ is non-decreasing.



Monotone Jobs

Definition

A job j is monotone iff its work function $w_j(k) = k \times t_j(k)$ is non-decreasing.

Encoding the Processing Times

Usual assumption: encode the nm processing times in the input

- $m \leq \|I\|$
- Algorithms can have running time $\mathrm{poly}(n, m)$

Encoding the Processing Times

Usual assumption: encode the nm processing times in the input

- $m \leq \|I\|$
- Algorithms can have running time $\mathrm{poly}(n, m)$

Often: compact encoding possible

- m can be exponential in $\|I\|$
- Algorithms should have running time $\mathrm{poly}(n, \log m)$

Known Results & This Work

General Jobs

	m unbounded	$m \leq \|I\|$	m constant
Hardness	$\frac{3}{2} - \varepsilon$ inapprox.	NP-hard for $m \geq 4$	
Approximation	$\frac{3}{2} + \varepsilon$	PTAS	EPTAS

Monotone Jobs

	m unbounded	$m \leq \|I\|$	m constant
Hardness	weakly NP-hard for $m \geq 2$		
Approximation	$\frac{3}{2}$	PTAS	EPTAS

Known Results & This Work

General Jobs

	m unbounded	$m \leq \|I\|$	m constant
Hardness	$\frac{3}{2} - \varepsilon$ inapprox.	NP-hard for $m \geq 4$	
Approximation	$\frac{3}{2} + \varepsilon$	PTAS	EPTAS

Monotone Jobs

	m unbounded	$m \leq \|I\|$	m constant
Hardness	strongly NP-hard		weakly NP-hard
Approximation	PTAS, fast $\frac{3}{2} + \varepsilon$	PTAS	EPTAS

Compression

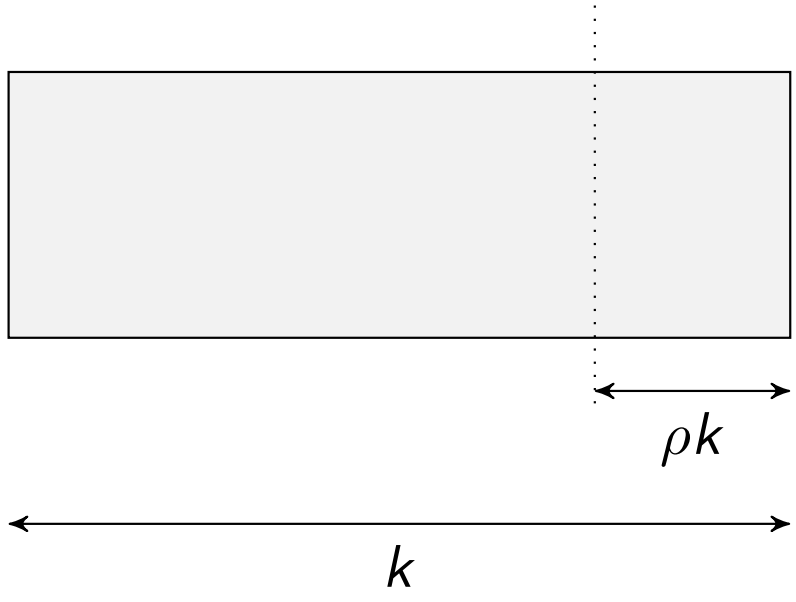
Lemma

Let $\rho \in (0, 1/4]$ *and* $k \in [m]$, $k \geq \frac{1}{\rho}$.
Then $\mathrm{t}_j(\lfloor k(1-\rho) \rfloor) \leq (1+4\rho)\,\mathrm{t}_j(k)$.

Compression

Lemma

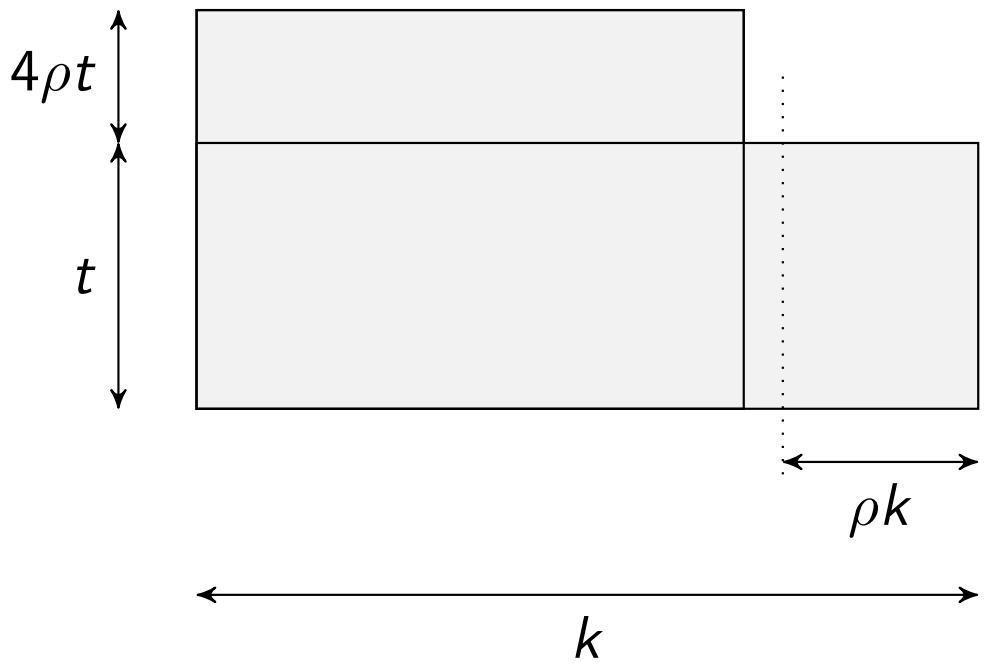
Let $\rho \in (0, 1/4]$ *and* $k \in [m]$, $k \geq \frac{1}{\rho}$.
Then $\mathrm{t}_j(\lfloor k(1-\rho) \rfloor) \leq (1+4\rho)\,\mathrm{t}_j(k)$.



Compression

Lemma

Let $\rho \in (0, 1/4]$ *and* $k \in [m]$, $k \geq \frac{1}{\rho}$.
Then $\mathrm{t}_j(\lfloor k(1-\rho) \rfloor) \leq (1+4\rho)\,\mathrm{t}_j(k)$.

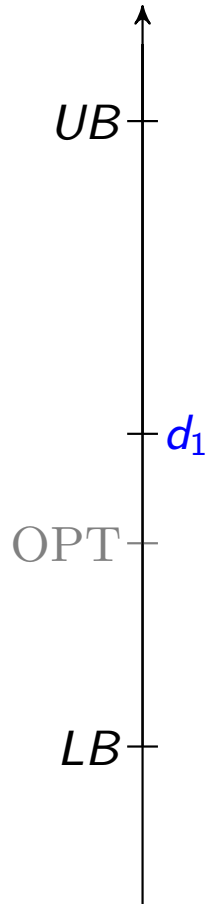


Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm

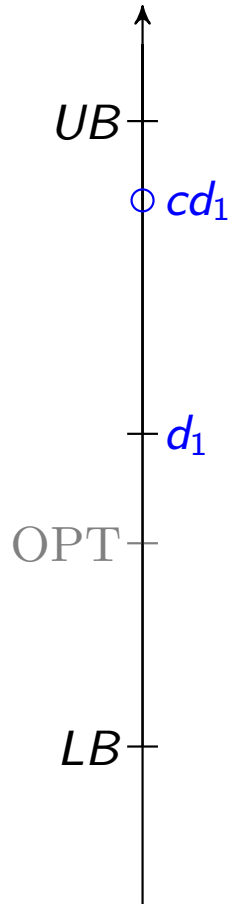
Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm



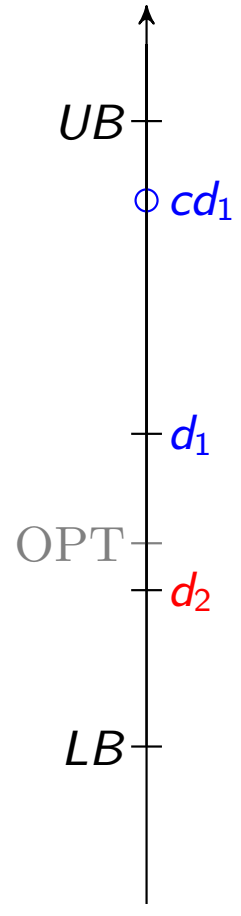
Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm



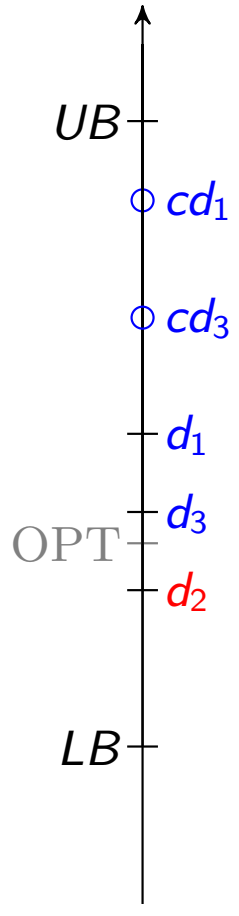
Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm



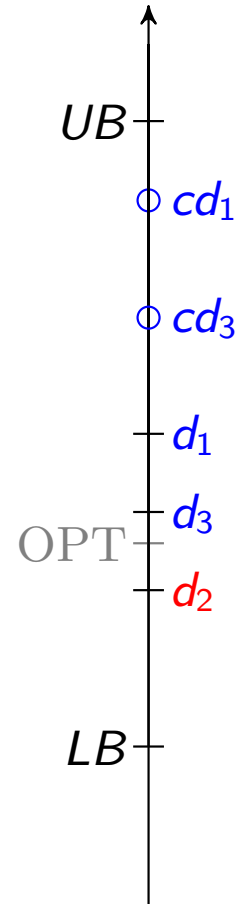
Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm

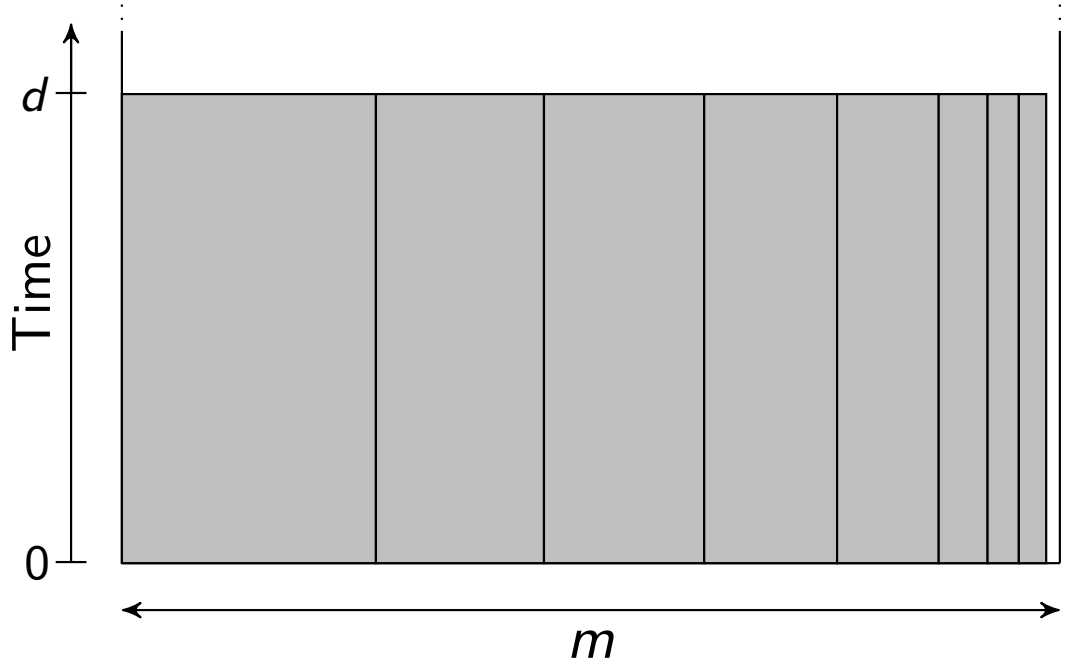


Dual Approximation [Hochbaum, Shmoys, 1987]

- Known bounds $LB \leq \mathrm{OPT} \leq UB$, $\frac{UB}{LB} = \mathrm{O}(1)$
- A c-dual approximation algorithm
  - gets a guess d of the optimum makespan
  - computes a schedule with makespan $\leq cd$
  - if $d < \mathrm{OPT}$: may reject
- Obtain a $(c + \varepsilon)$-approximate algorithm with $\mathrm{O}(\log \frac{1}{\varepsilon})$ rounds of dual algorithm
- Use 2-approximate algorithm [Ludwig, Tiwari, 1994] to obtain bounds
  Running time $\mathrm{O}(n \log^2 m)$

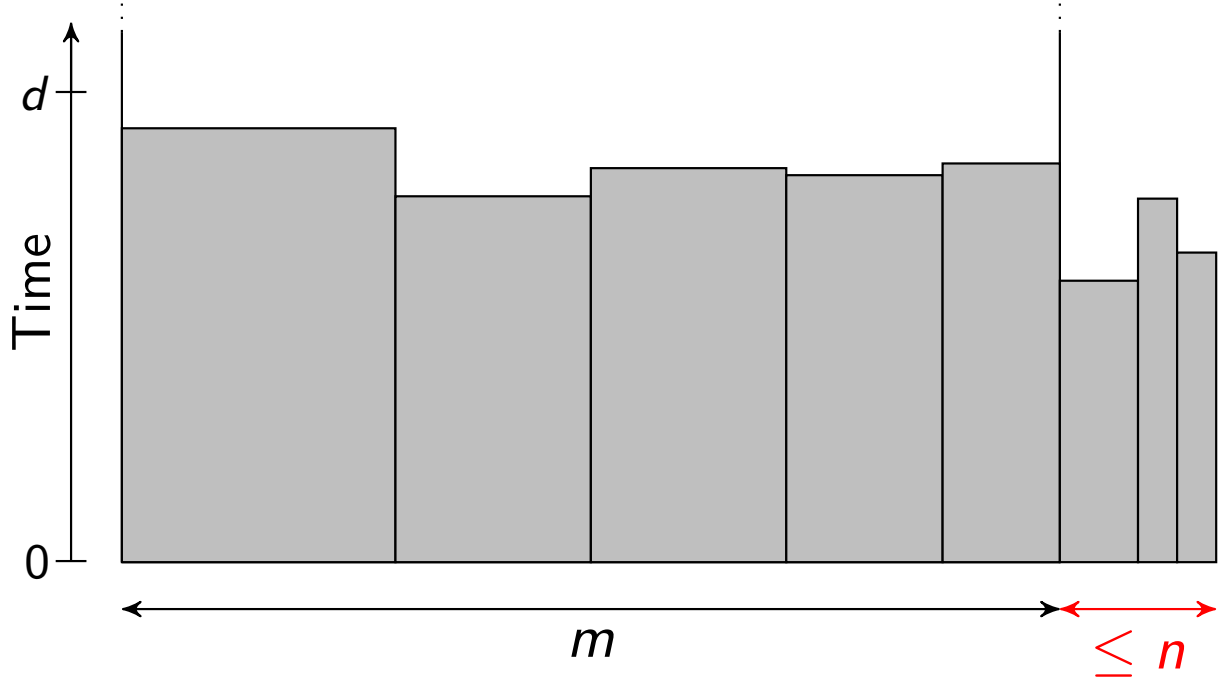


FPTAS: $(1+\varepsilon)$-dual algorithm for large m



- Ideal: Fractional schedule with makespan d

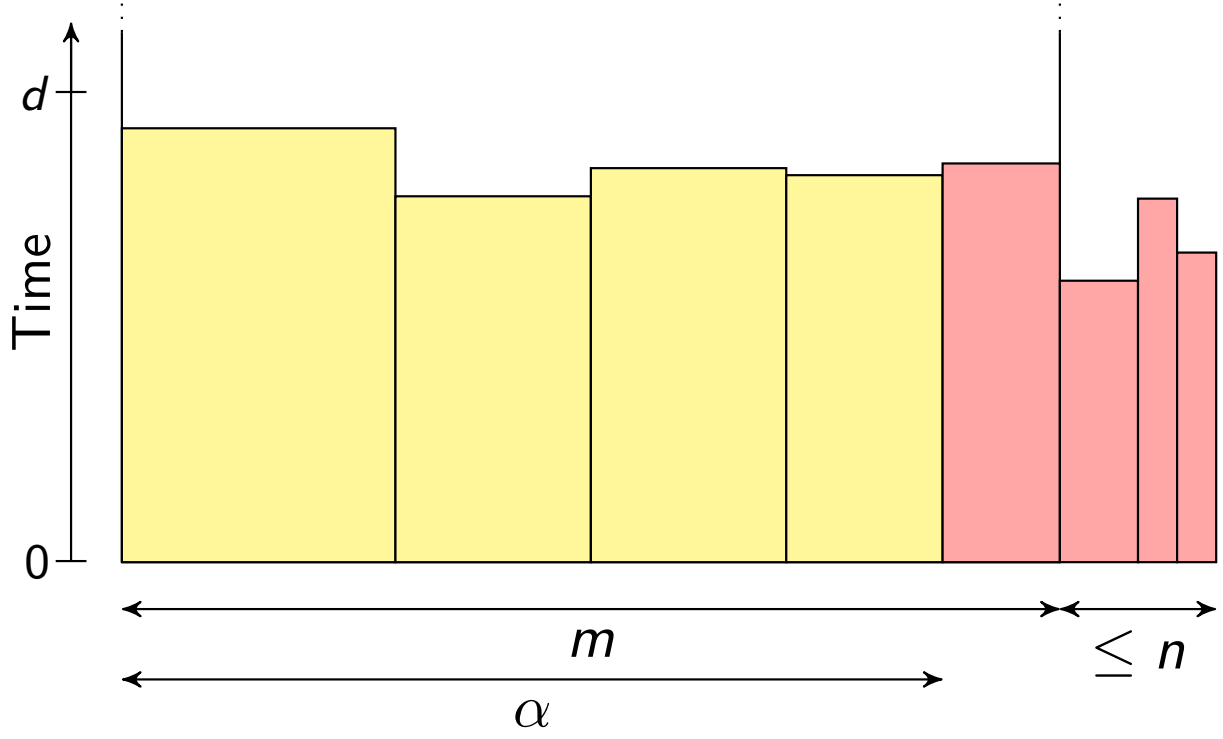
FPTAS: $(1+\varepsilon)$-dual algorithm for large m



- Schedule all jobs in parallel such that they process just in d time units

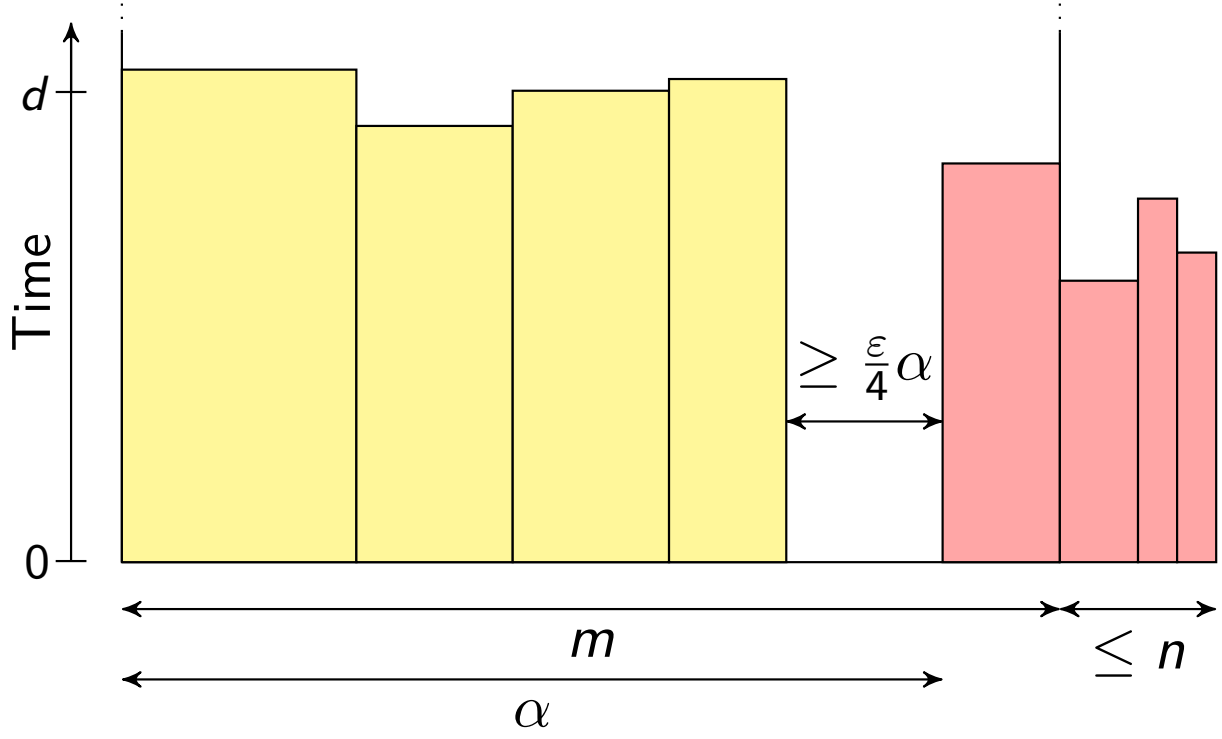
$$\gamma(j,d) = \min\{k \in [m] \mid \mathrm{t}_j(k) \leq d\}$$

FPTAS: $(1+\varepsilon)$-dual algorithm for large m



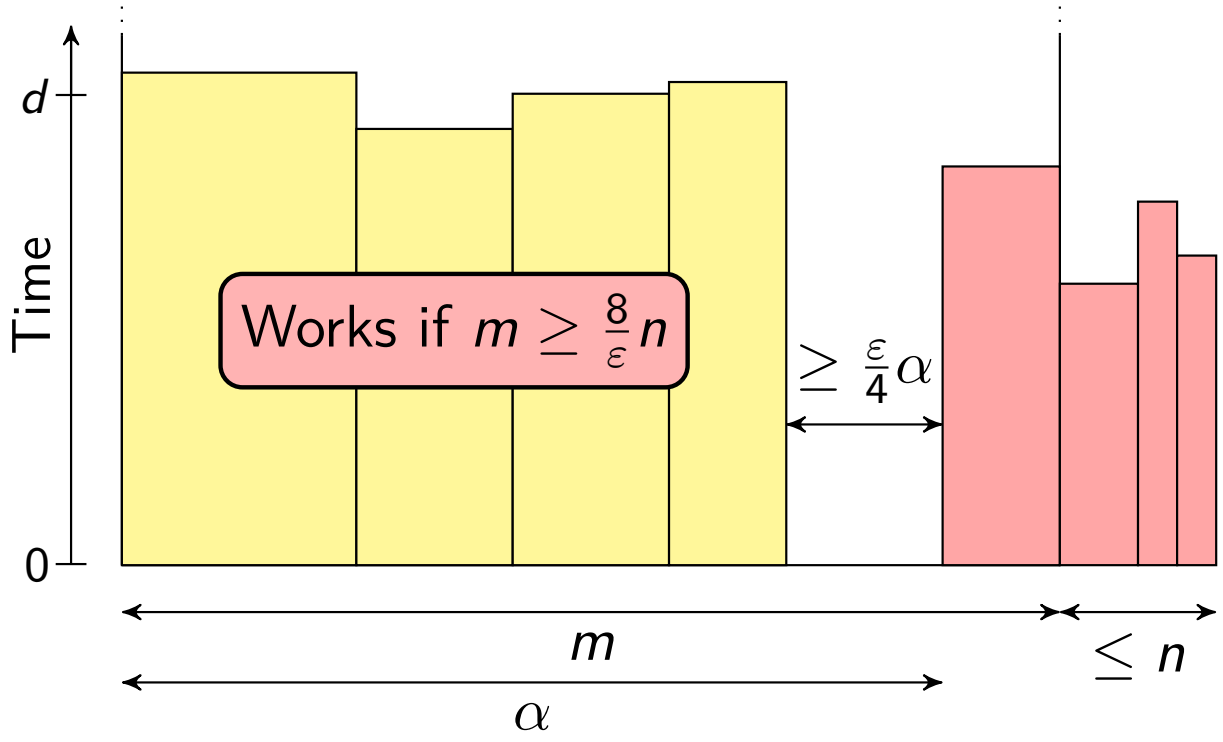
- Schedule all jobs in parallel such that they process just in d time units
- Compress all compressible jobs with factor $\rho = \frac{\varepsilon}{4}$

FPTAS: $(1+\varepsilon)$-dual algorithm for large m



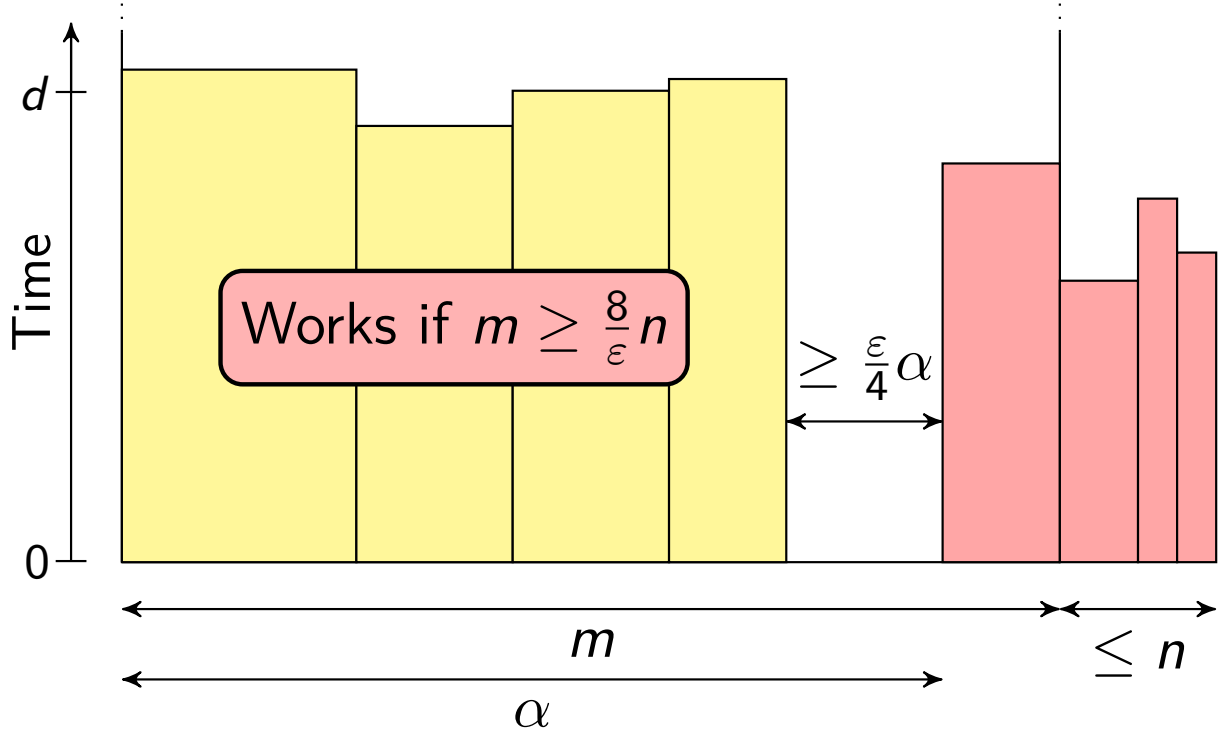
- Schedule all jobs in parallel such that they process just in d time units
- Compress all compressible jobs with factor $\rho = \frac{\varepsilon}{4}$

FPTAS: $(1+\varepsilon)$-dual algorithm for large m



- Schedule all jobs in parallel such that they process just in d time units
- Number of machines used by non-compressible jobs is at most $n/\rho = 4n/\varepsilon$

FPTAS: $(1+\varepsilon)$-dual algorithm for large m



- Schedule all jobs in parallel such that they process just in ~~d~~ $(1+\varepsilon)d$ time units
- Running time $\mathrm{O}(n \log^2(\frac{1}{\varepsilon} + \frac{\log(\varepsilon m)}{\varepsilon}))$

A PTAS: general case

- If $m \geq \frac{8}{\varepsilon} n$, use our $(1+\varepsilon)$-dual algorithm
- Otherwise, $m = \mathrm{O}(\frac{n}{\varepsilon})$; use PTAS for that case [Jansen, Thöle, 2010]

A Linear $(\frac{3}{2} + \varepsilon)$-approximate Algorithm

Improves upon $(\frac{3}{2} + \varepsilon)$-dual approximate algorithm with running time $\mathrm{O}(nm \log(1/\varepsilon))$ [Mounié, Rapine, Trystram, 2007]

A Linear $(\frac{3}{2}+\varepsilon)$-approximate Algorithm

Improves upon $(\frac{3}{2}+\varepsilon)$-dual approximate algorithm with running time $\mathrm{O}(nm\log(1/\varepsilon))$ [Mounié, Rapine, Trystram, 2007]

Steps of the Algorithms

1. Remove Small Jobs
2. Find two shelf schedule s.t. total work (including removed jobs) at most md
3. Repair two shelf schedule
4. Insert small jobs with next fit

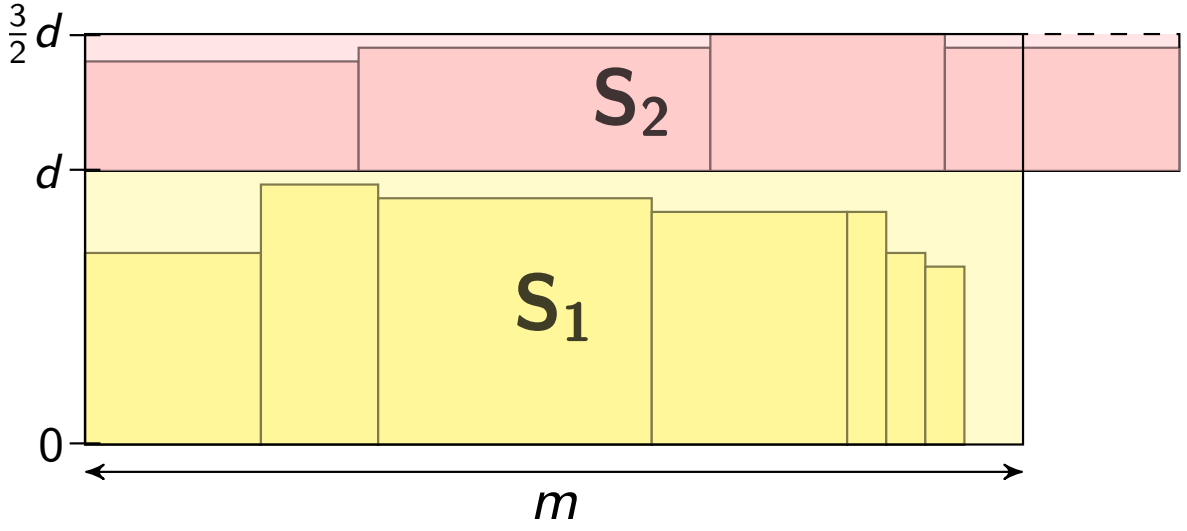
A Linear $(\frac{3}{2}+\varepsilon)$-approximate Algorithm

Improves upon $(\frac{3}{2}+\varepsilon)$-dual approximate algorithm with running time $\mathrm{O}(nm\log(1/\varepsilon))$ [Mounié, Rapine, Trystram, 2007]

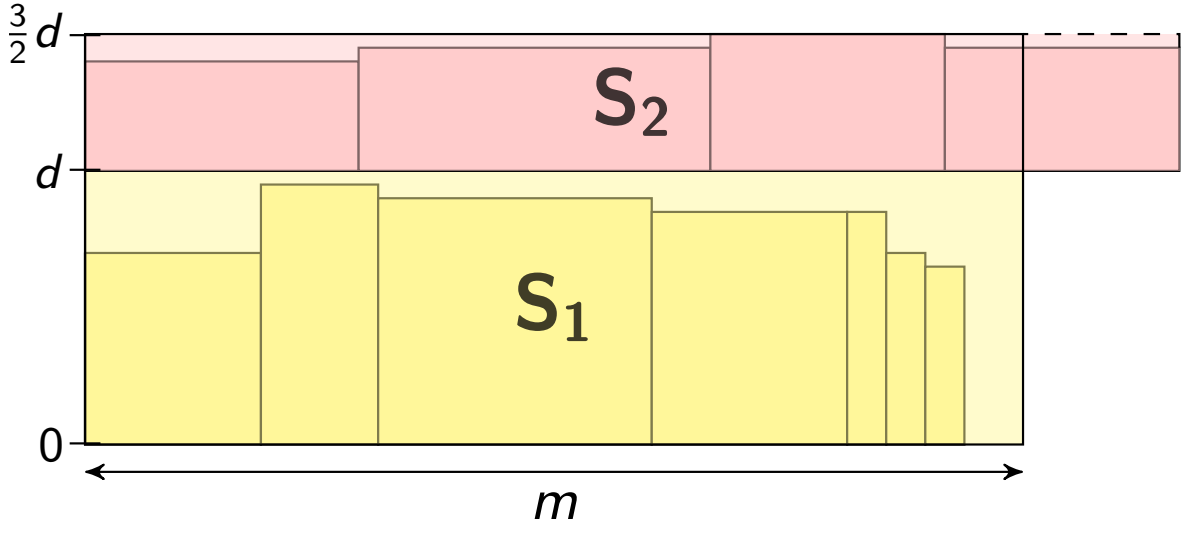
Steps of the Algorithms

1. Remove Small Jobs
2. Find two shelf schedule s.t. total work (including removed jobs) at most ~~md~~ $m(1+\frac{2}{3}\varepsilon)d$
3. Repair two shelf schedule
4. Insert small jobs with next fit

Two Shelf Schedules

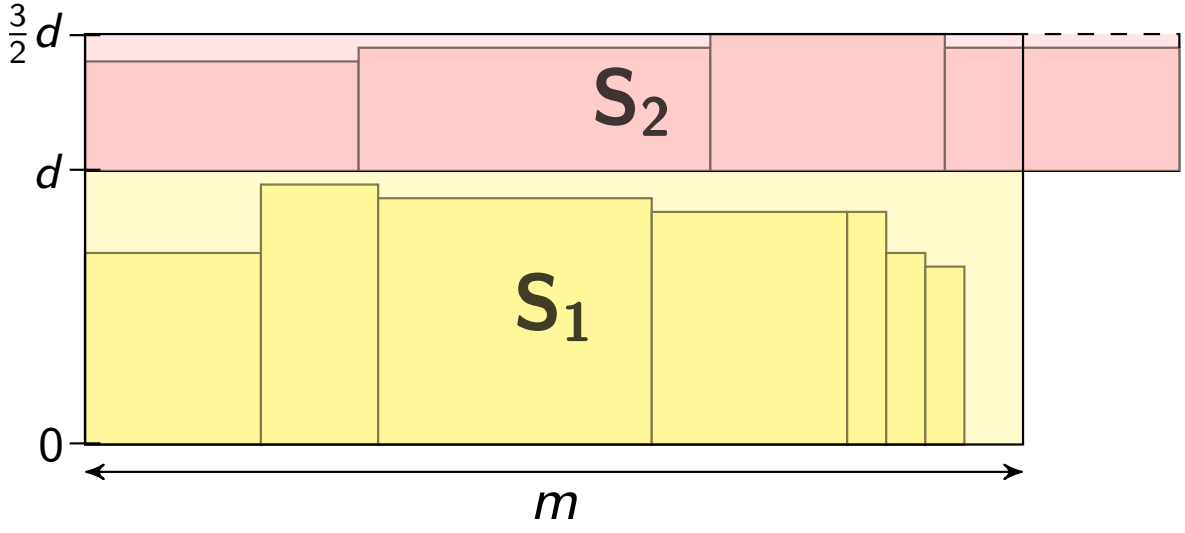


Two Shelf Schedules



- Original algorithm: find two shelf schedule with minimum total work

Two Shelf Schedules

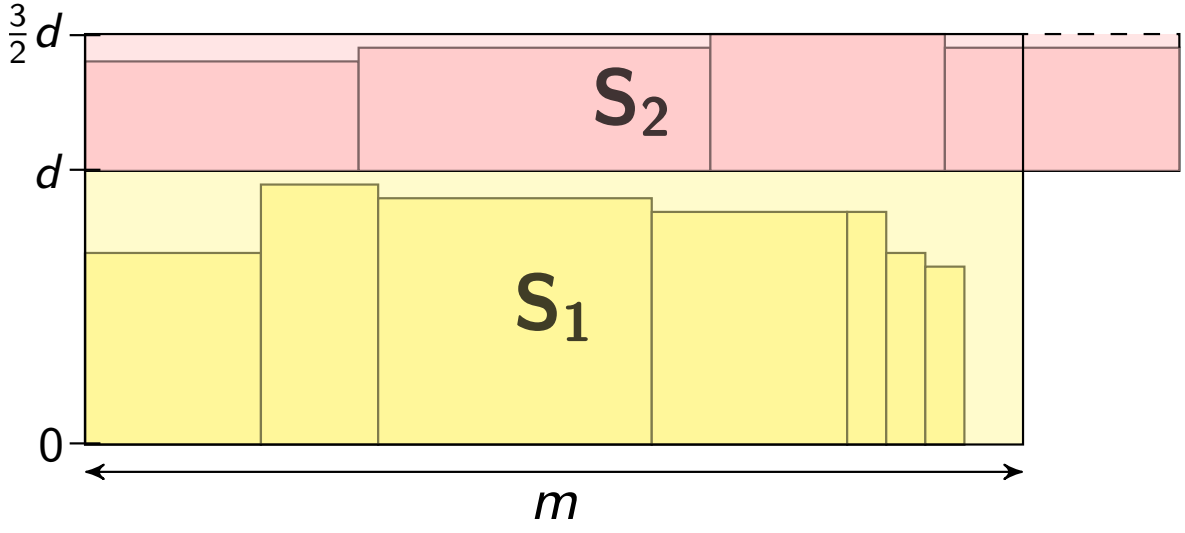


- Original algorithm: find two shelf schedule with minimum total work

$$\max_{J' \subseteq J_B} \sum_{j \in J'} w_j \left(\gamma\left(j, \frac{d}{2}\right) \right) - w_j(\gamma(j,d))$$

$$\sum_{j \in J'} \gamma(j,d) \leq m$$

Two Shelf Schedules

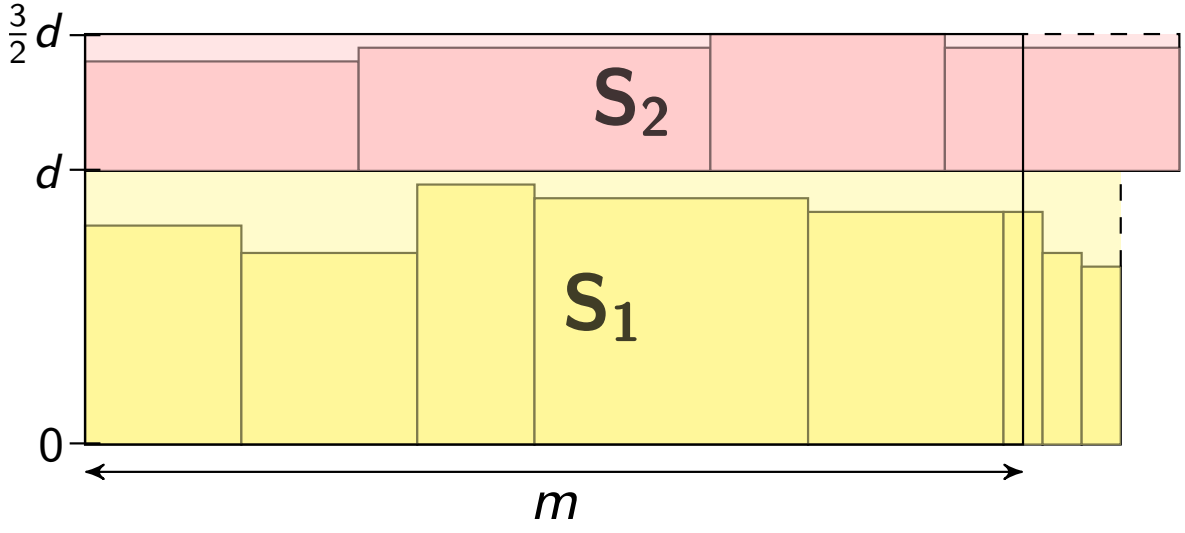


- Original algorithm: find two shelf schedule with minimum total work
- Our algorithm: knapsack with compressible items

$$\max_{J' \subseteq J_B} \sum_{j \in J'} w_j \left(\gamma\left(j, \frac{d}{2}\right) \right) - w_j(\gamma(j,d))$$

$$\sum_{j \in J'} \gamma(j,d) \leq m$$

Two Shelf Schedules

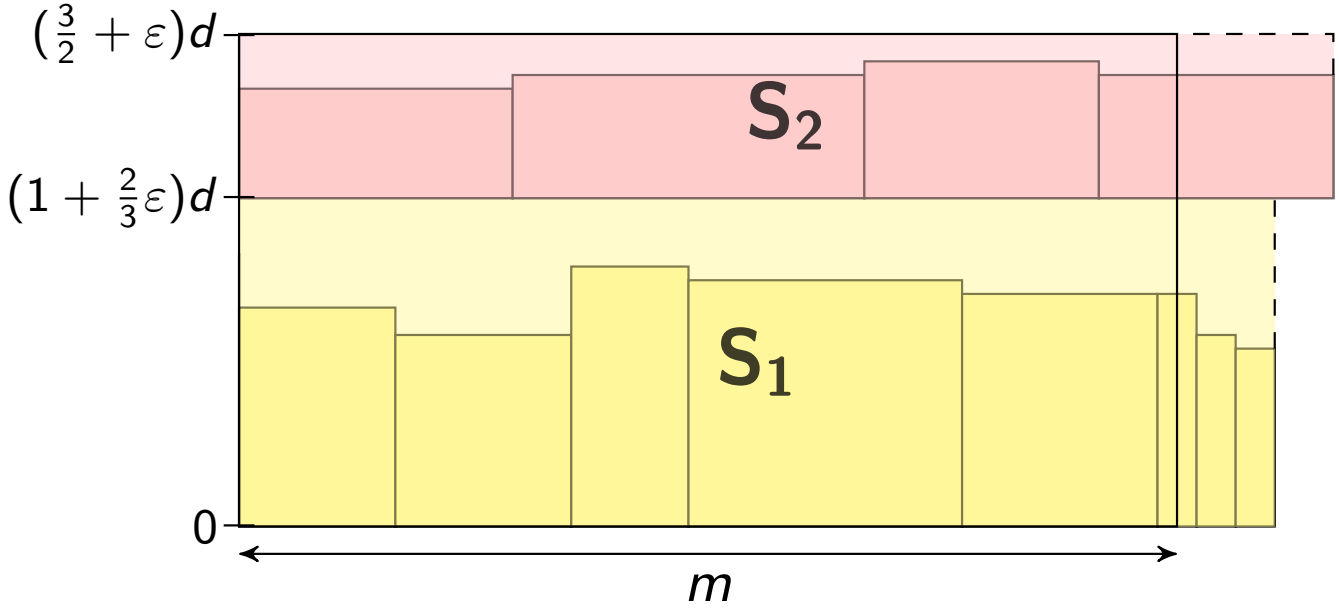


- Original algorithm: find two shelf schedule with minimum total work
- Our algorithm: knapsack with compressible items

$$\max_{J' \subseteq J_B} \sum_{j \in J'} w_j\left(\gamma\left(j, \frac{d}{2}\right)\right) - w_j(\gamma(j,d))$$

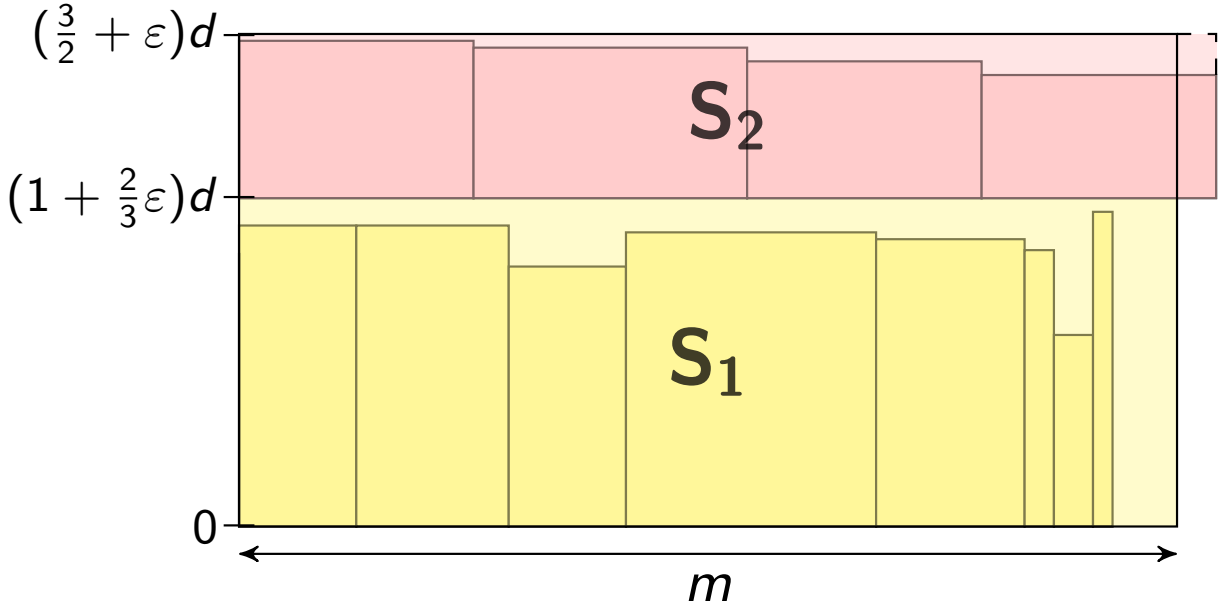
$$\sum_{j \in J' \cap J^c} (1-\rho)\,\gamma(j,d) + \sum_{j \in J' \setminus J^c} \gamma(j,d) \leq m$$

Two Shelf Schedules



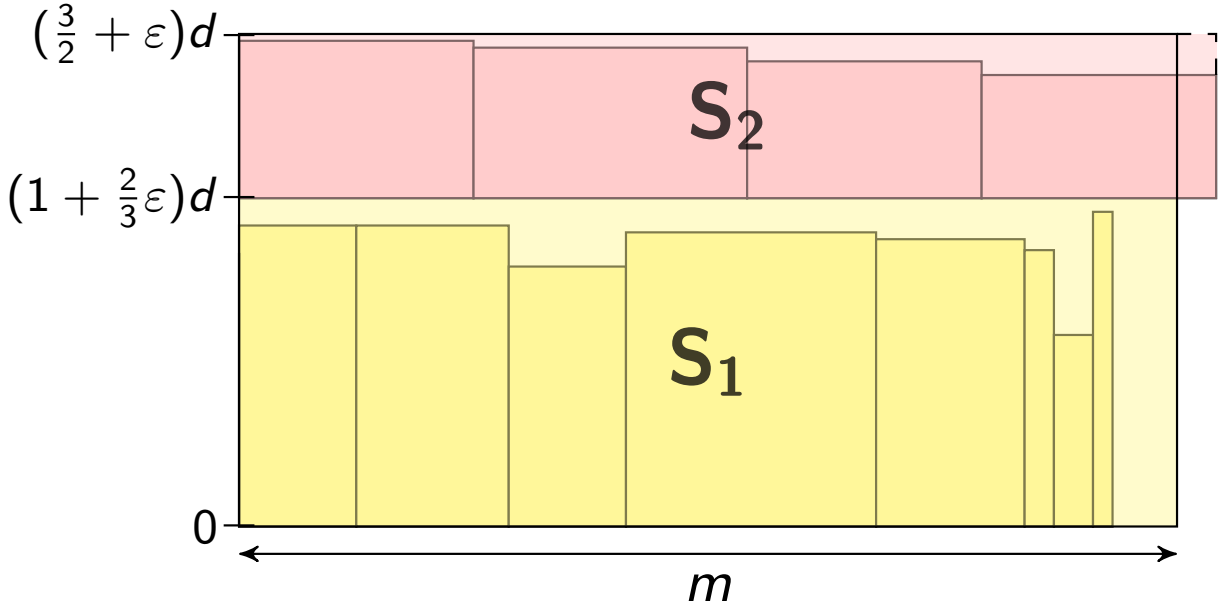
- Original algorithm: find two shelf schedule with minimum total work
- Our algorithm: knapsack with compressible items
- Increase shelf height – compressed jobs fit, allows for work slightly above md

Two Shelf Schedules



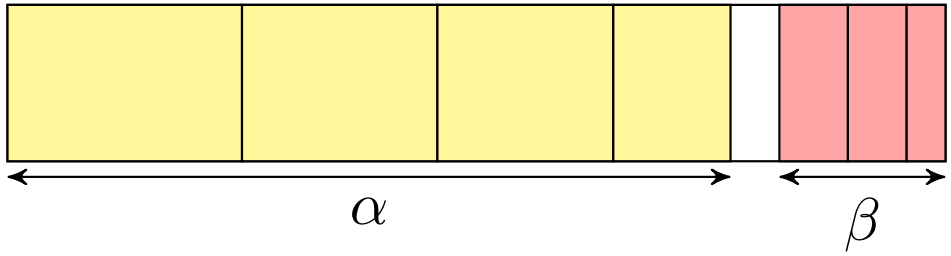
- Original algorithm: find two shelf schedule with minimum total work
- Our algorithm: knapsack with compressible items
- Increase shelf height – compressed jobs fit, allows for work slightly above md

Two Shelf Schedules



- Original algorithm: find two shelf schedule with minimum total work
- Our algorithm: knapsack with compressible items
- Increase shelf height – compressed jobs fit, allows for work slightly above md
- Work does not increase – repair procedure works

Solve Knapsack with Compressible Items

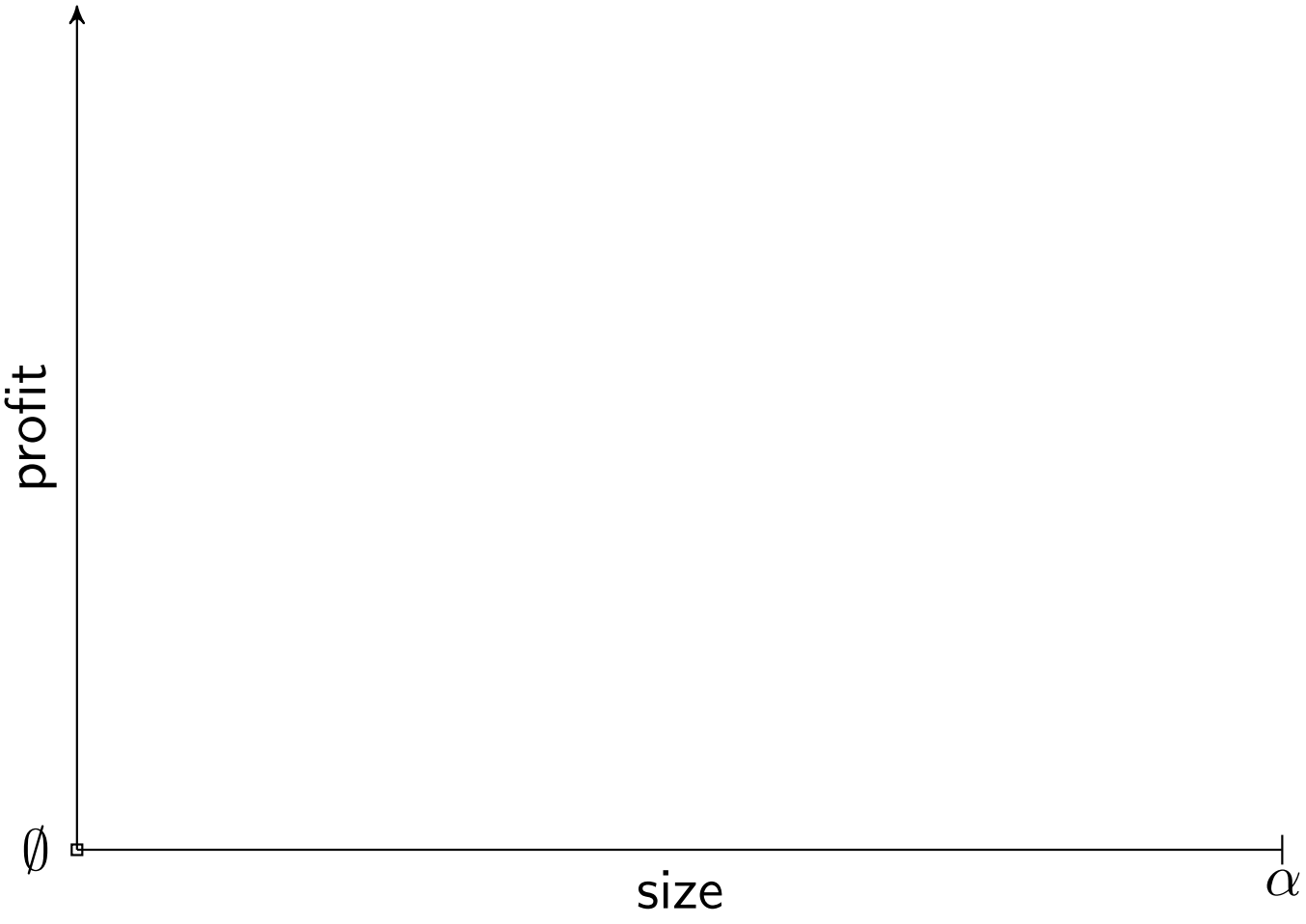


- Handle compressible and incompressible items separately
- Guess α, β approximately
- Knapsack problem for incompressible items has bounded capacity
- Use compression to solve knapsack problem for compressible item

Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

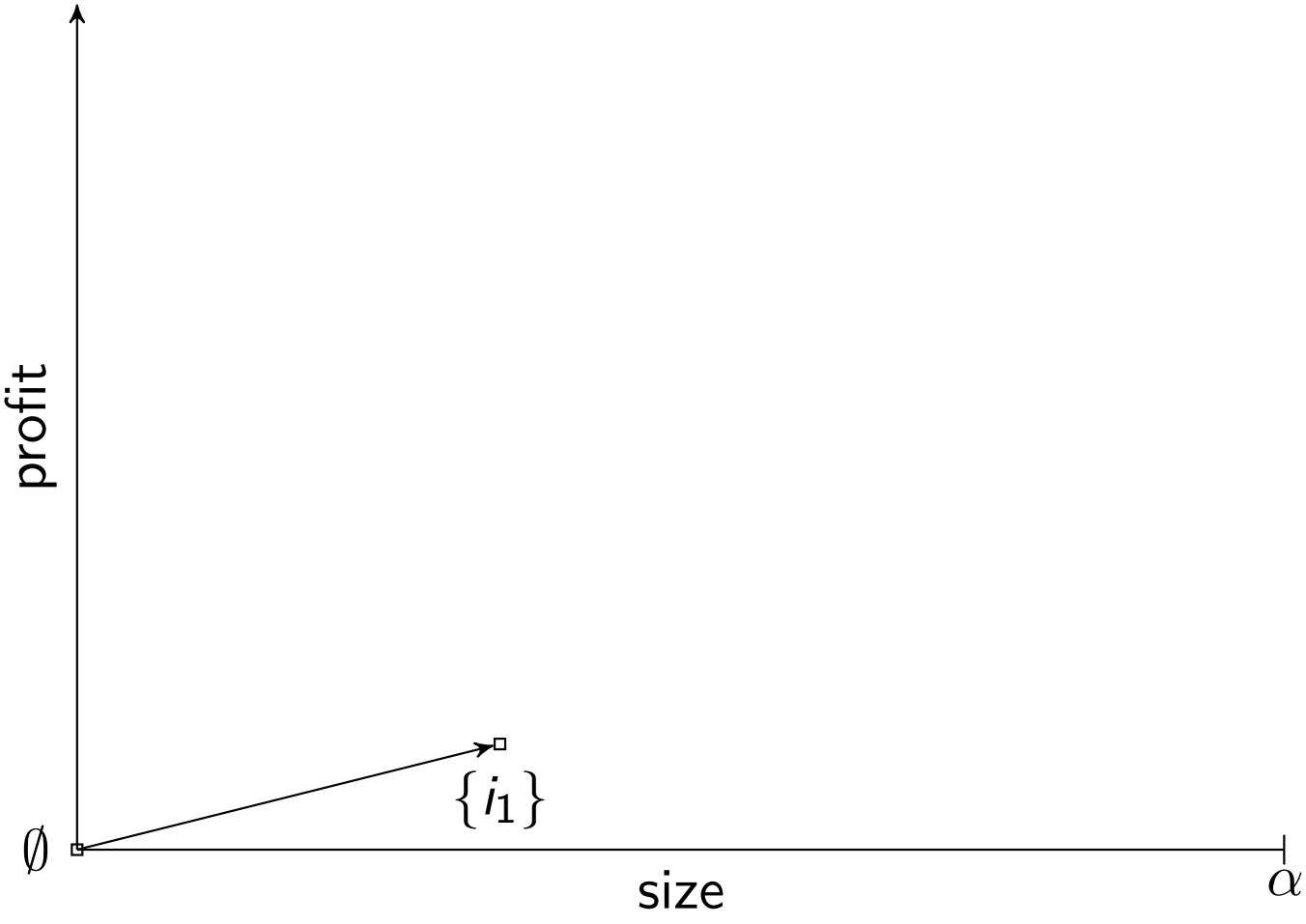
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

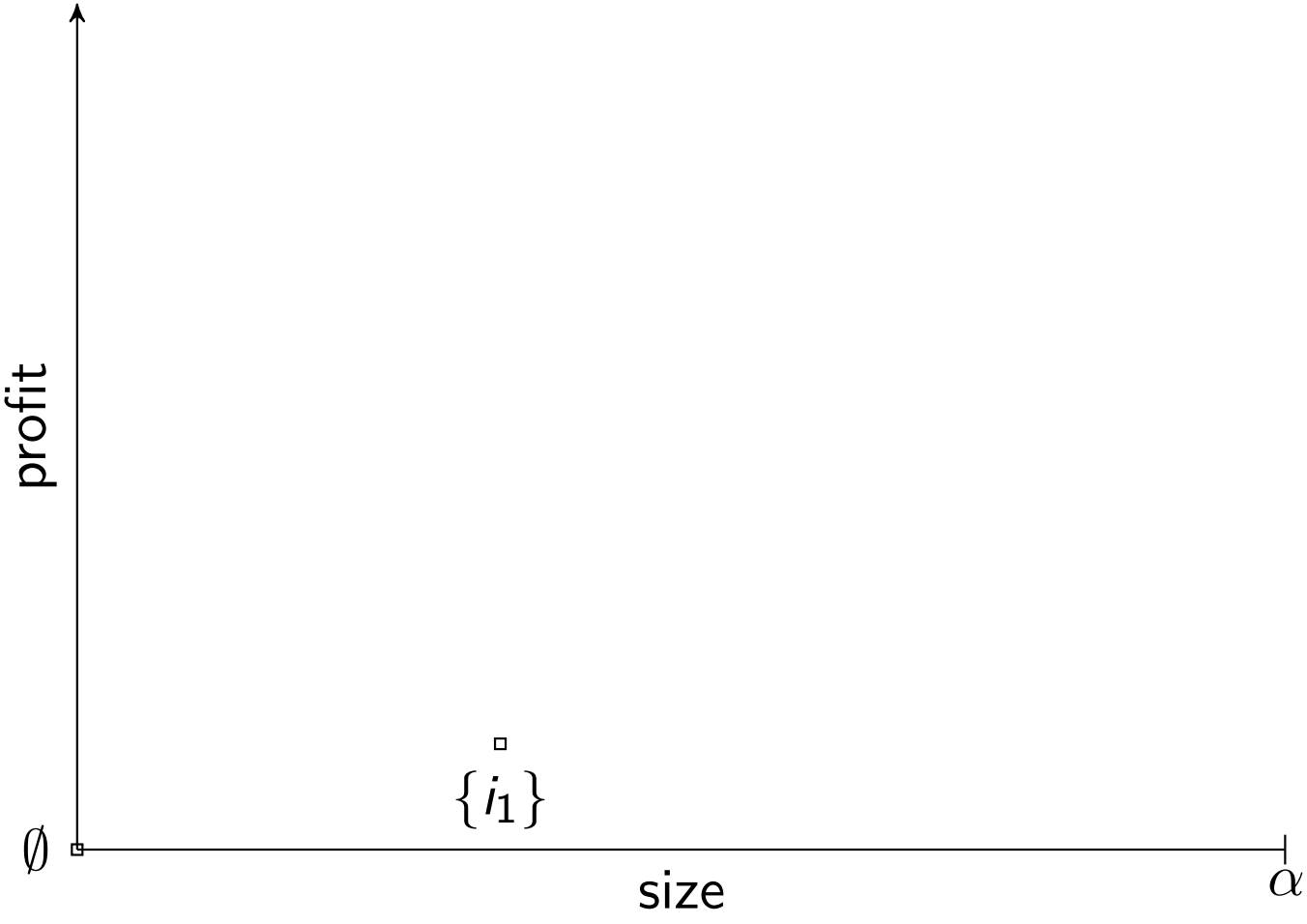
1 Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

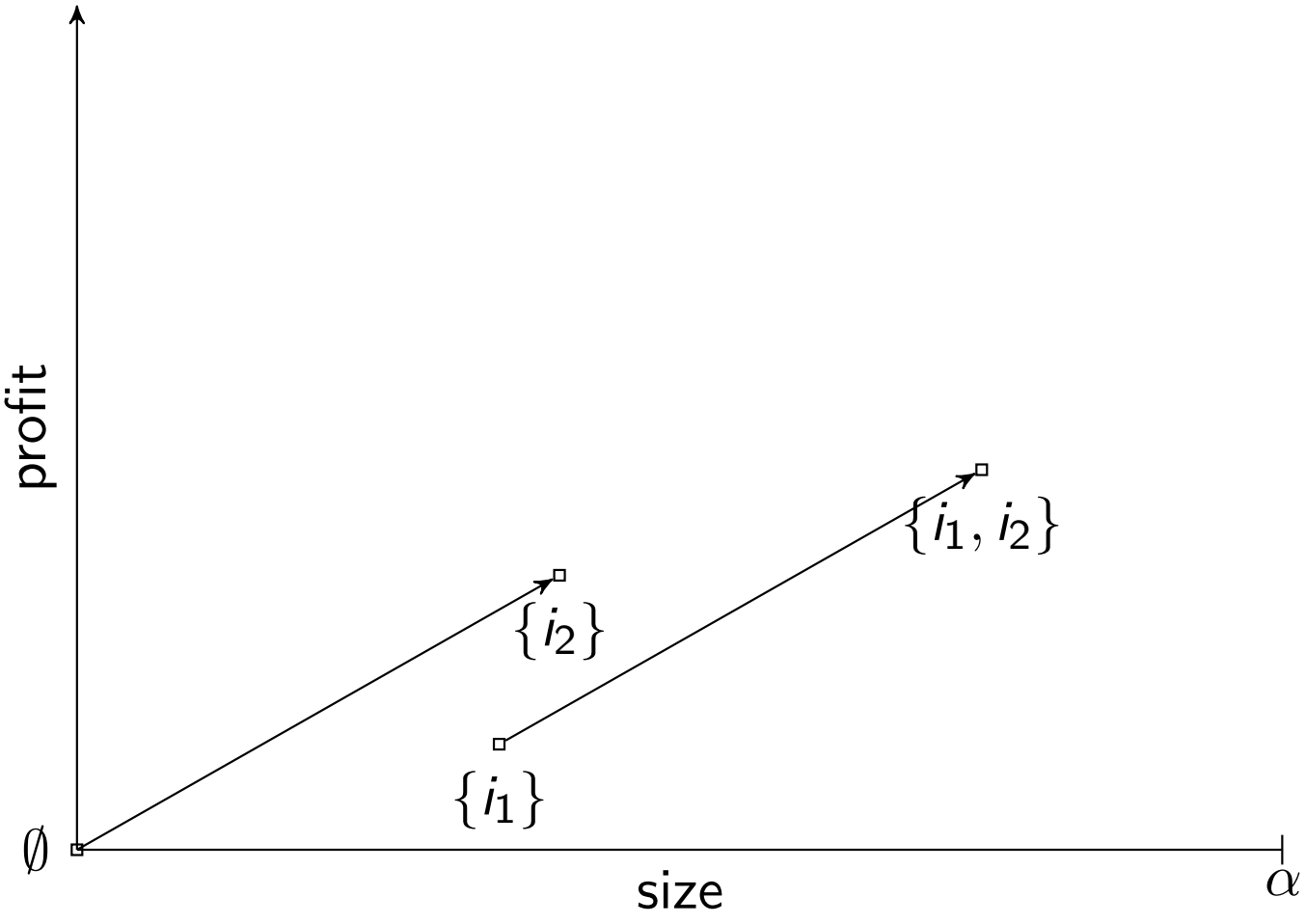
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

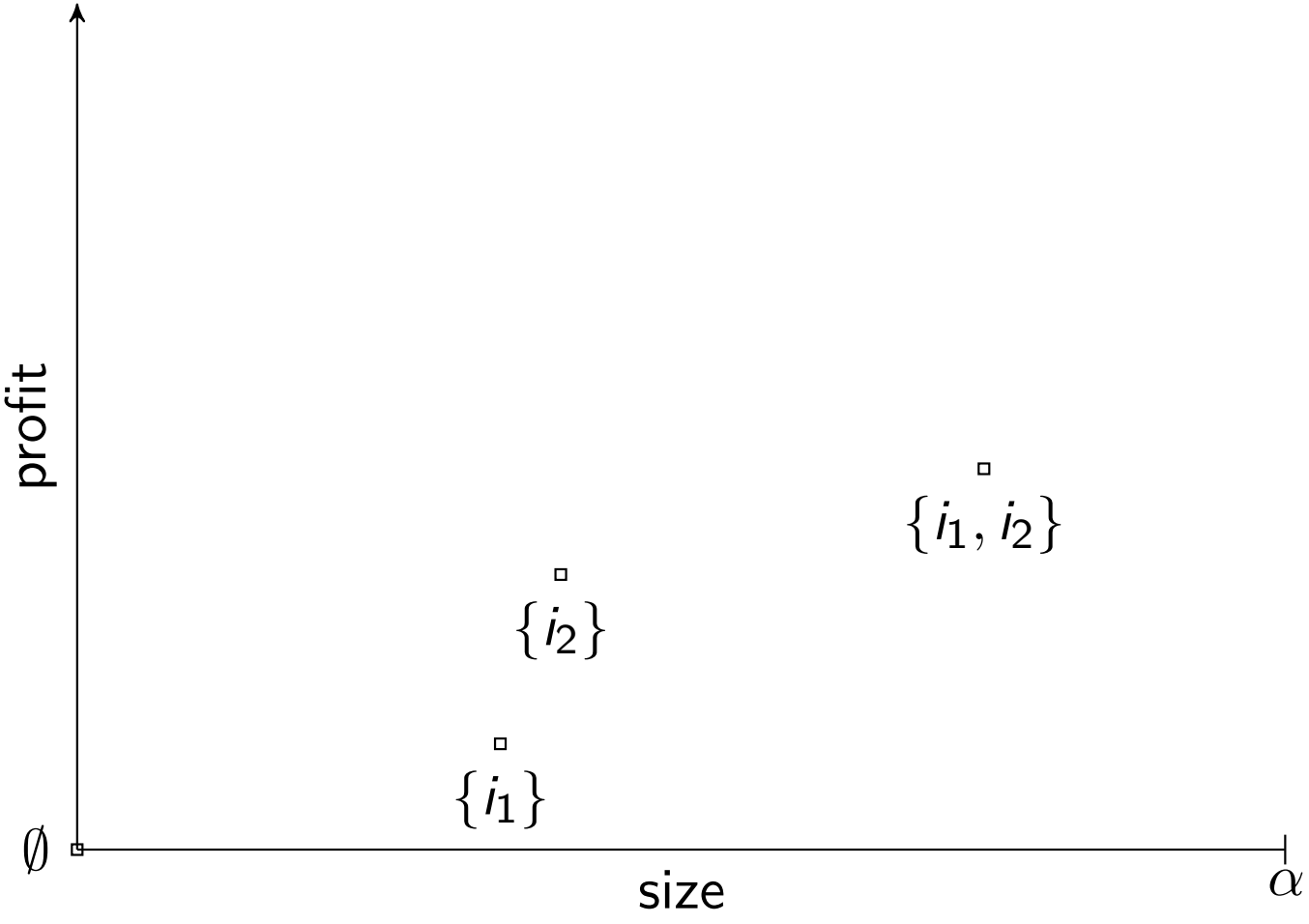
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

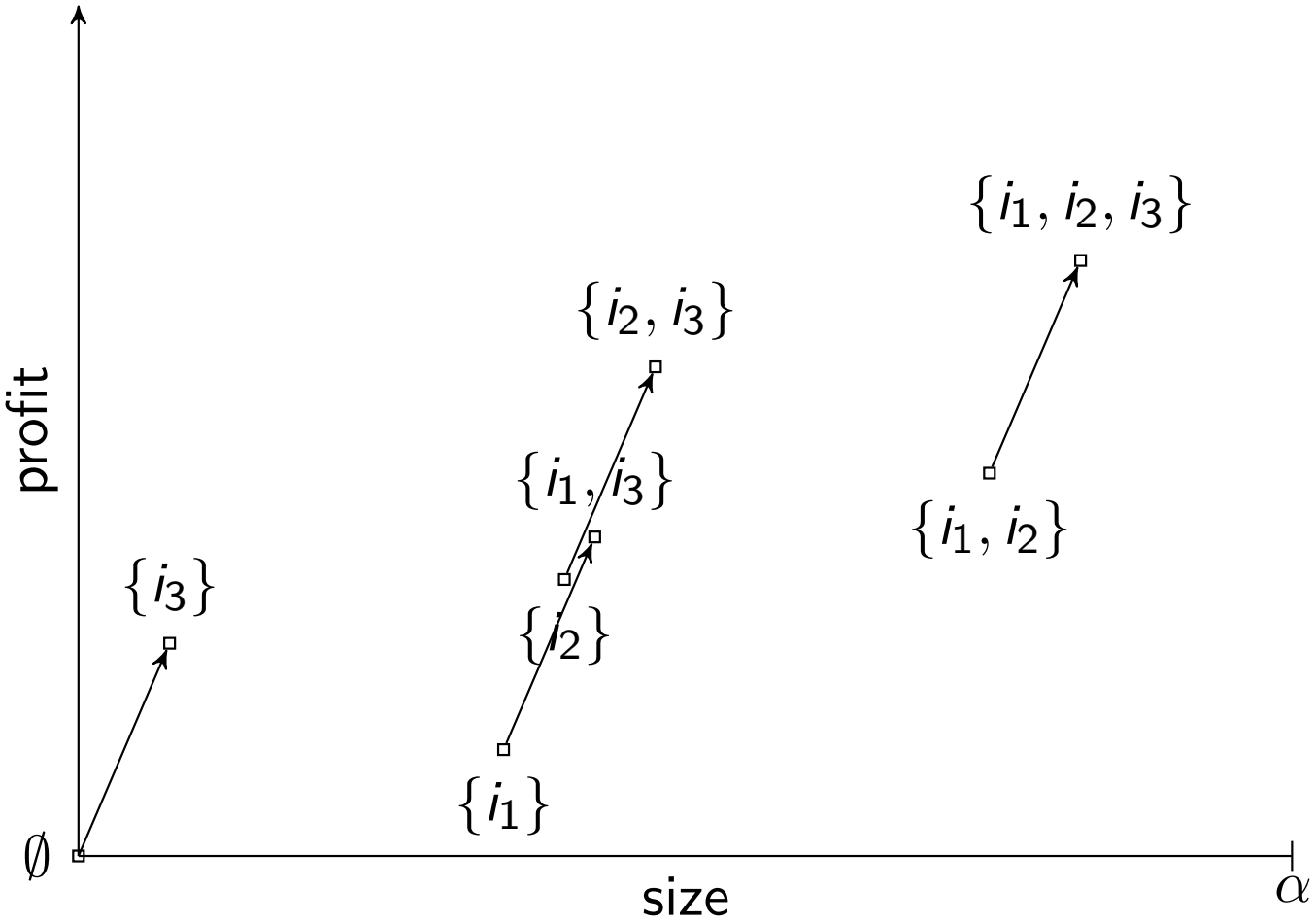
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

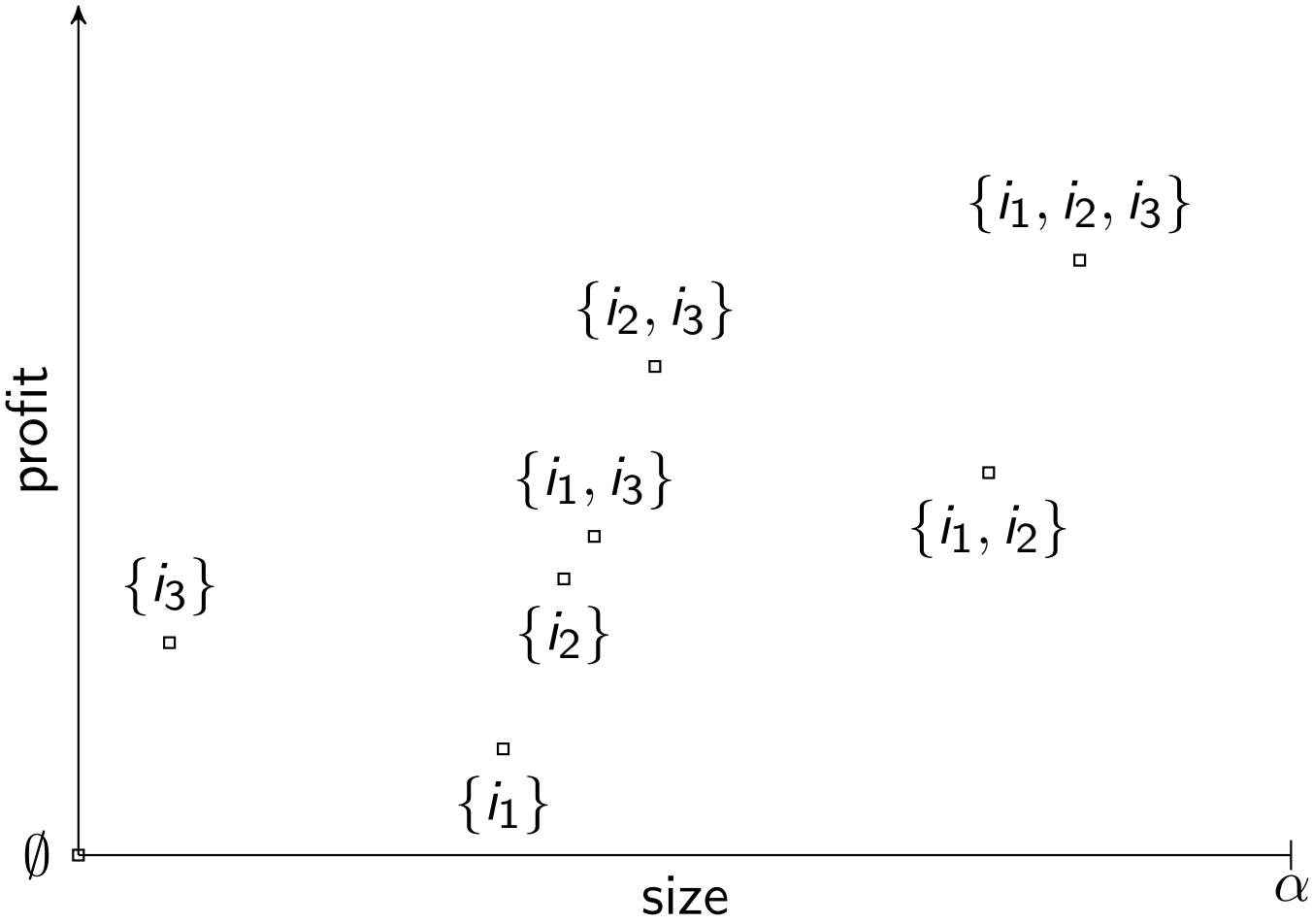
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

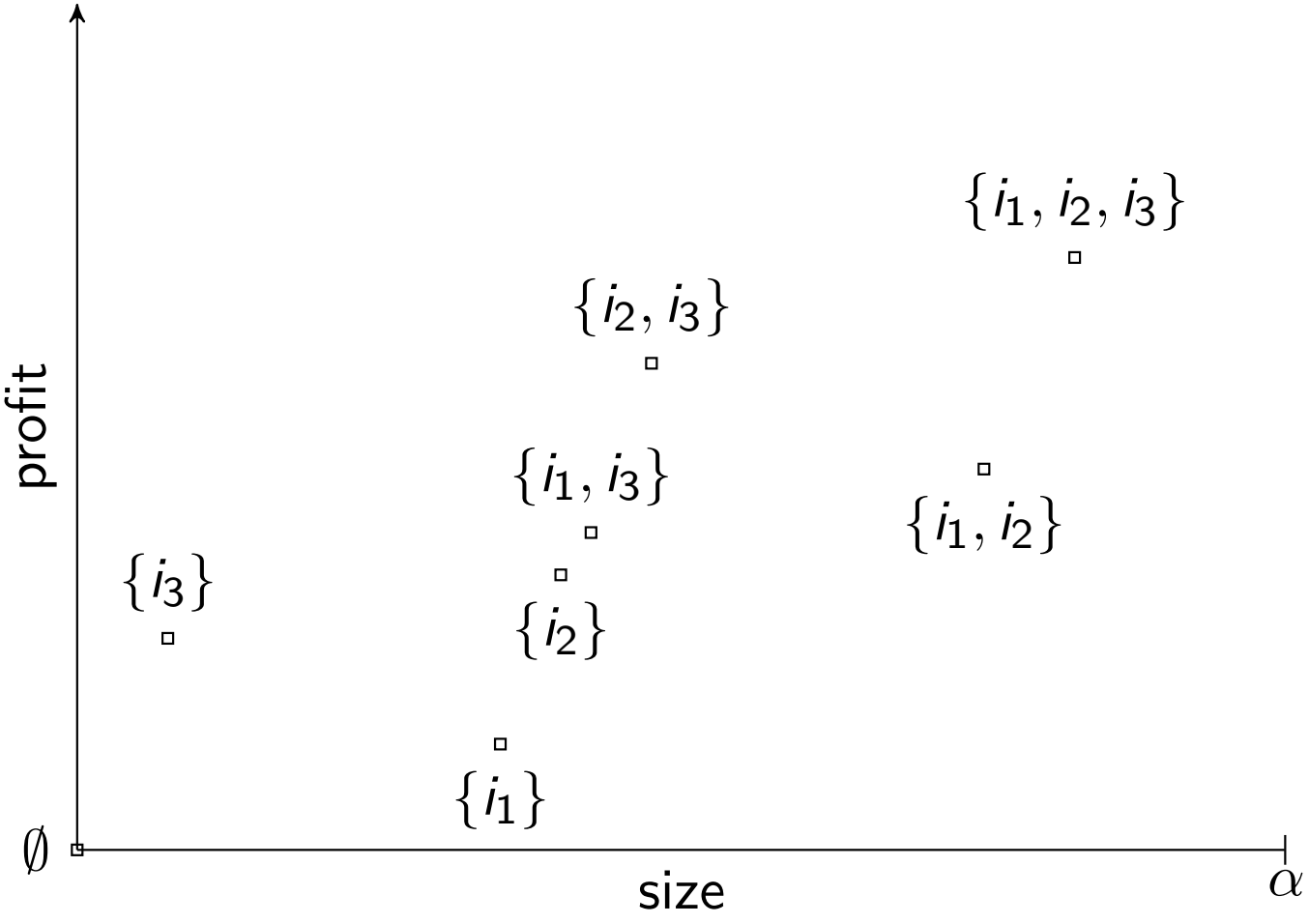
1. Iteratively build list of solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

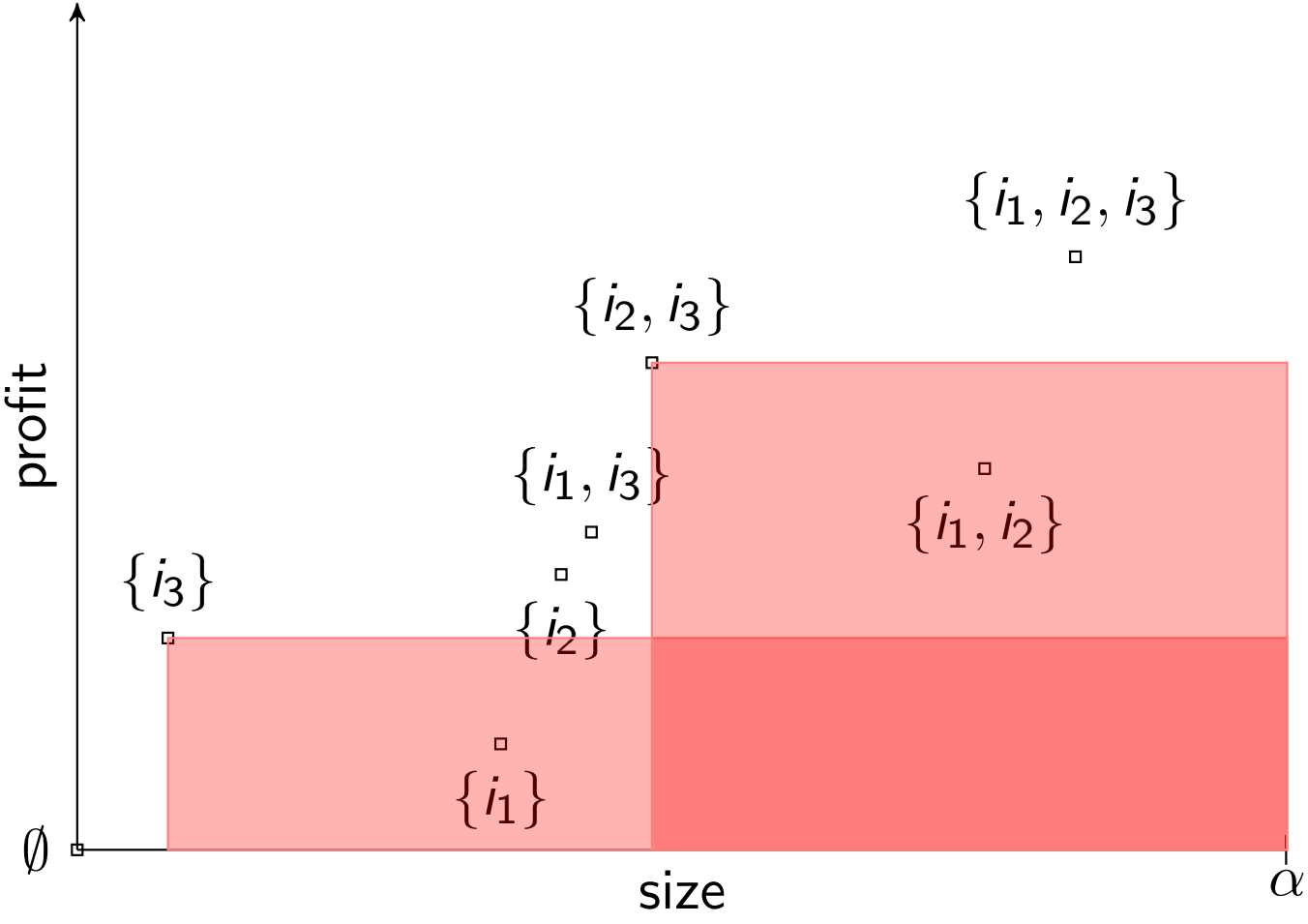
2 Remove dominated solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

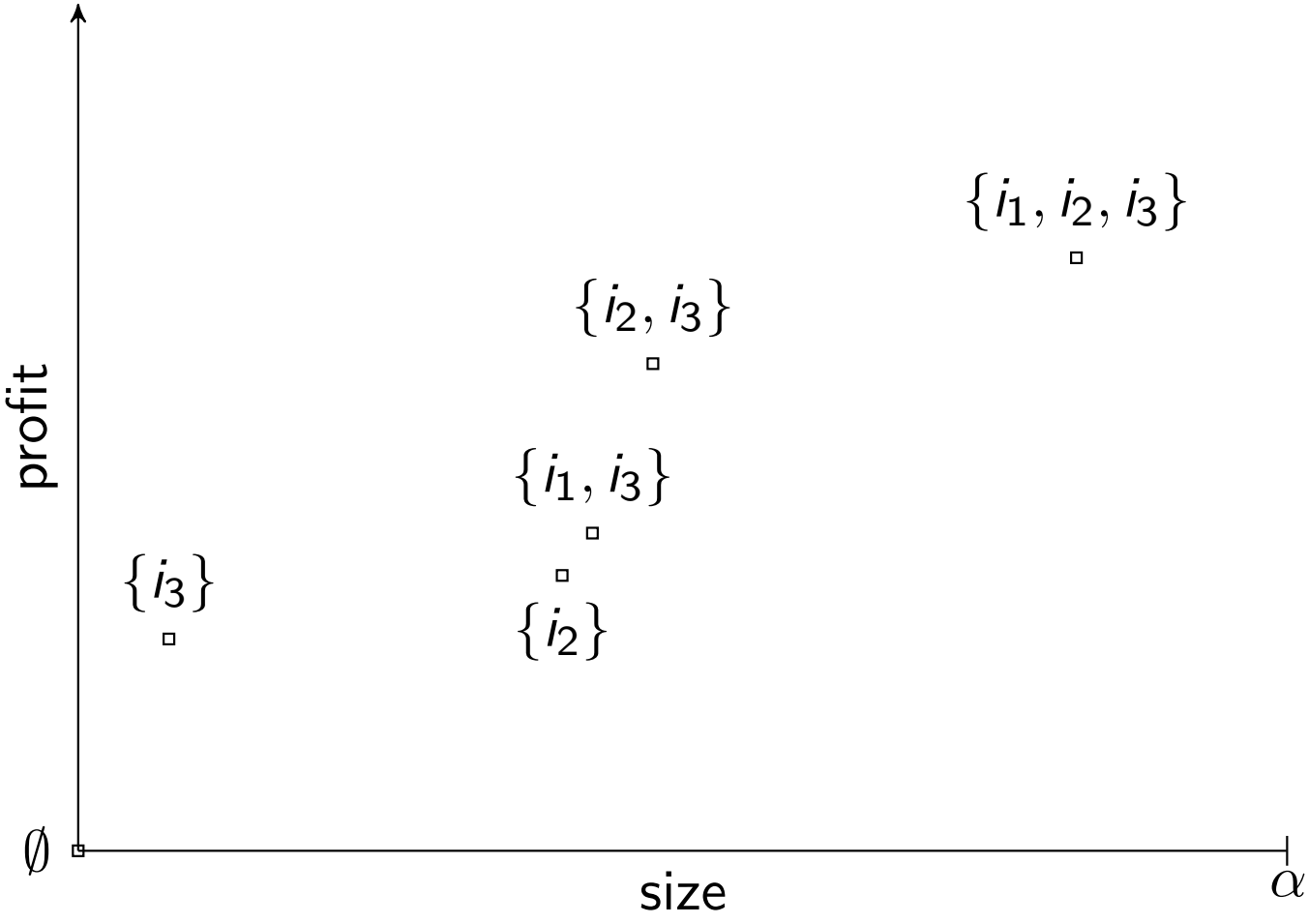
2 Remove dominated solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

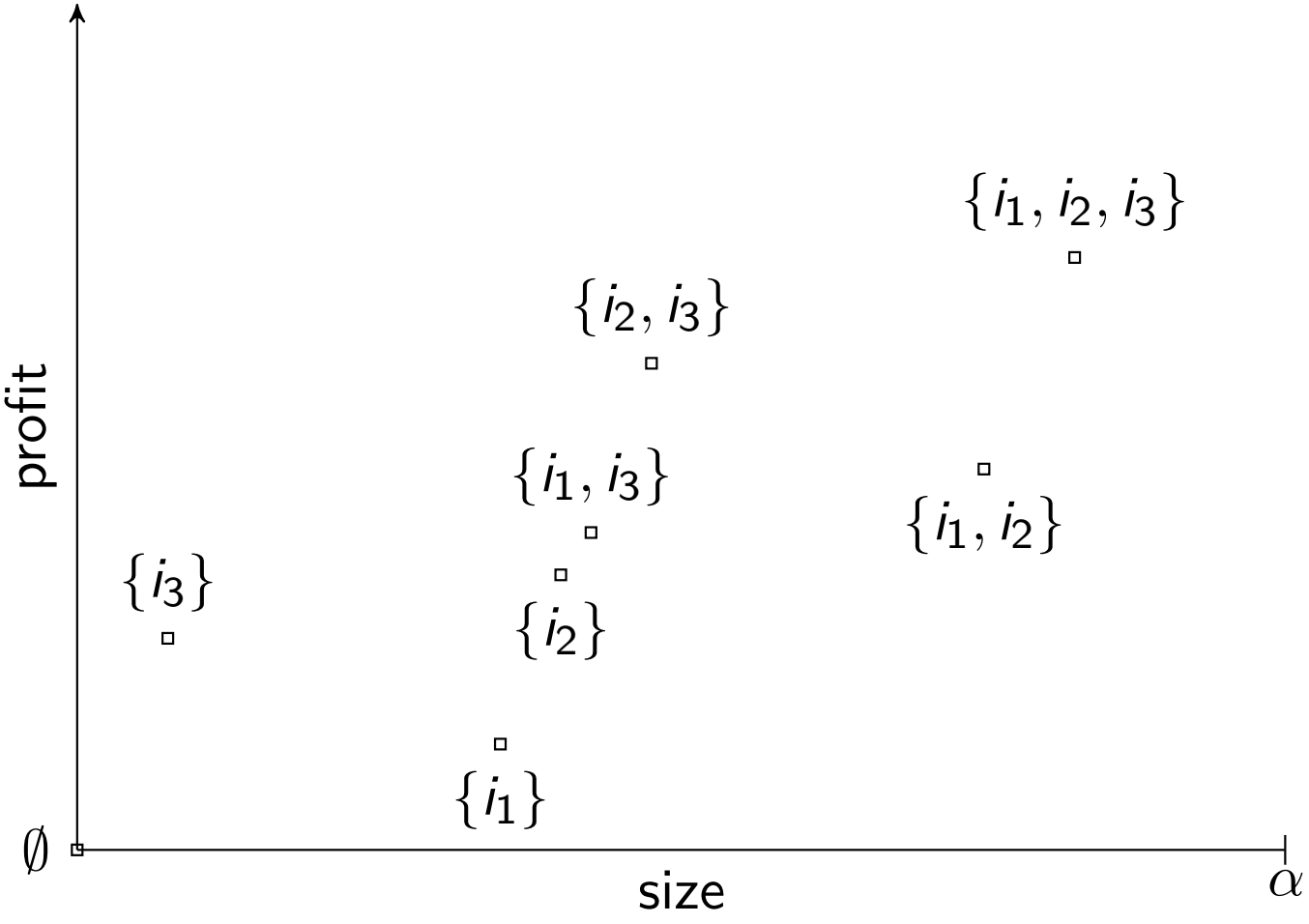
2 Remove dominated solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

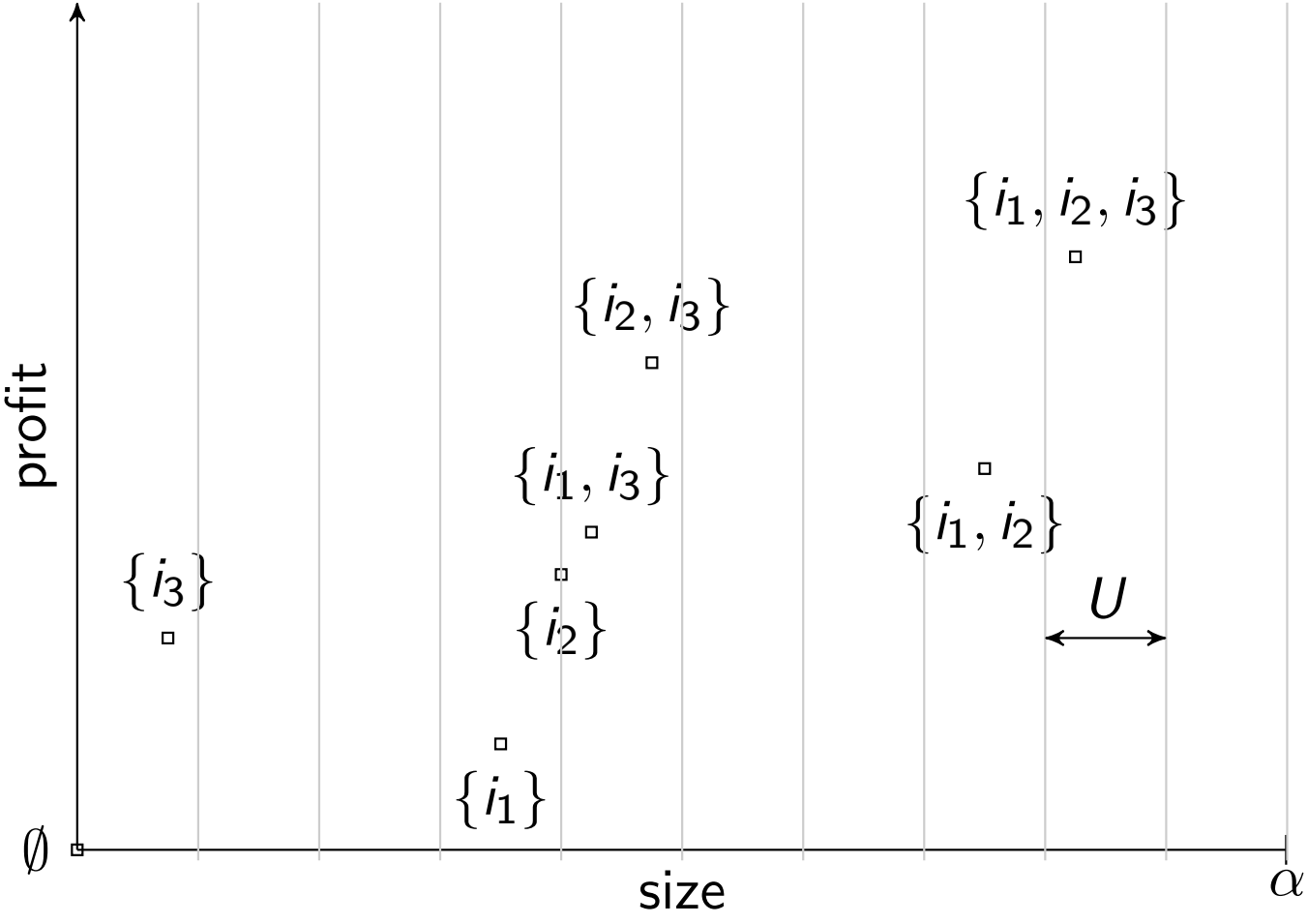
1.5 New: Normalize sizes



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

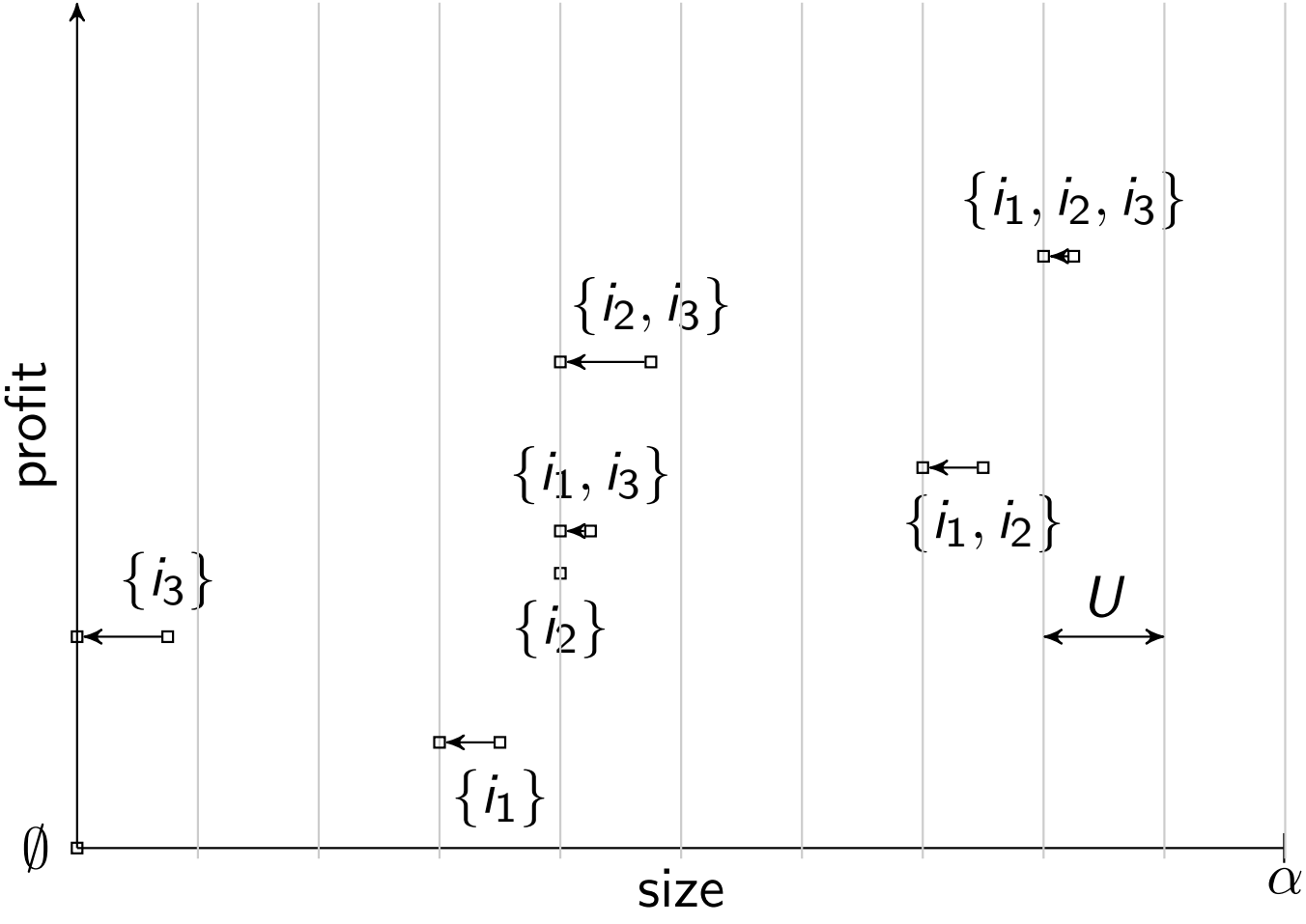
1.5 New: Normalize sizes



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

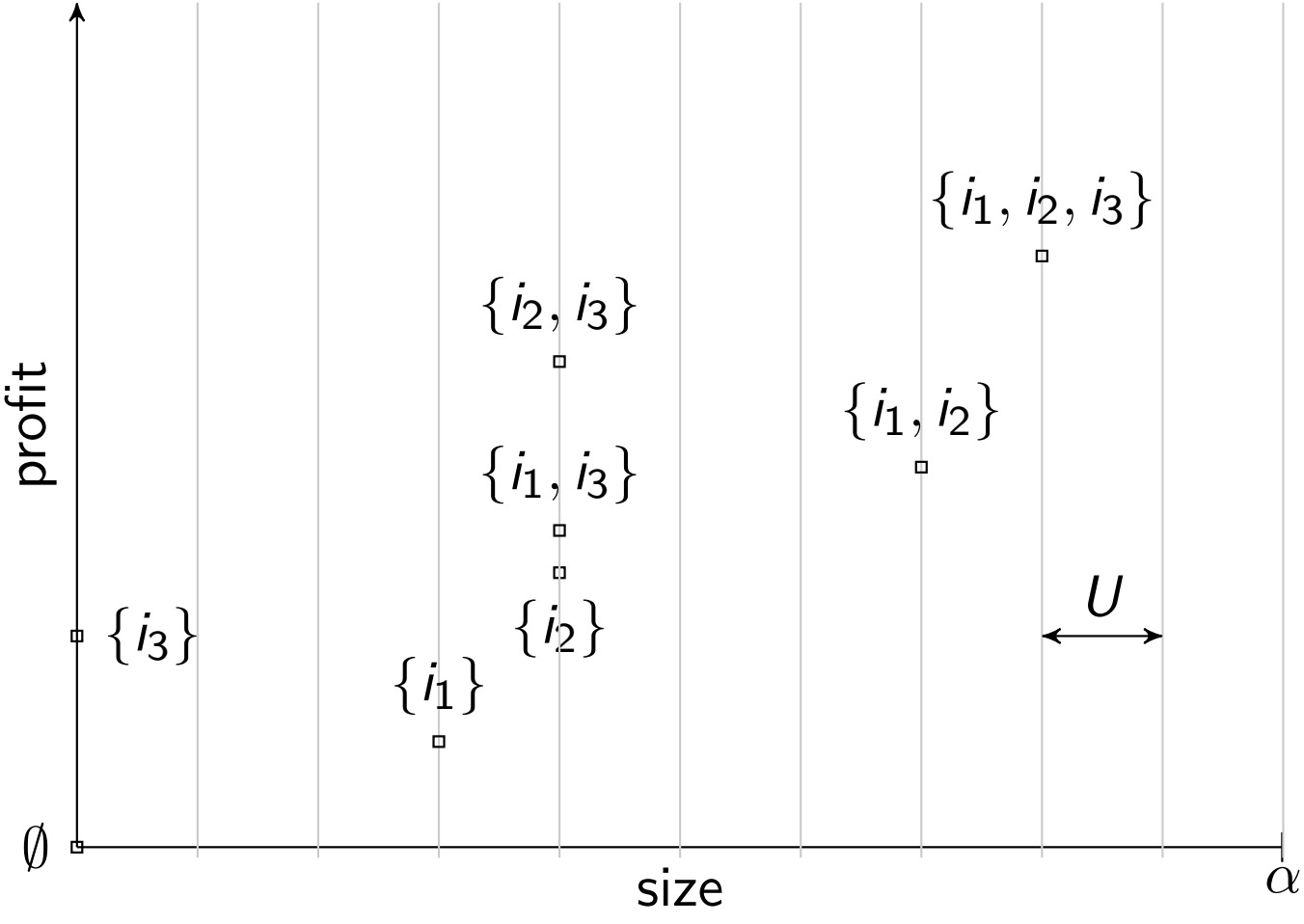
1.5 New: Normalize sizes



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

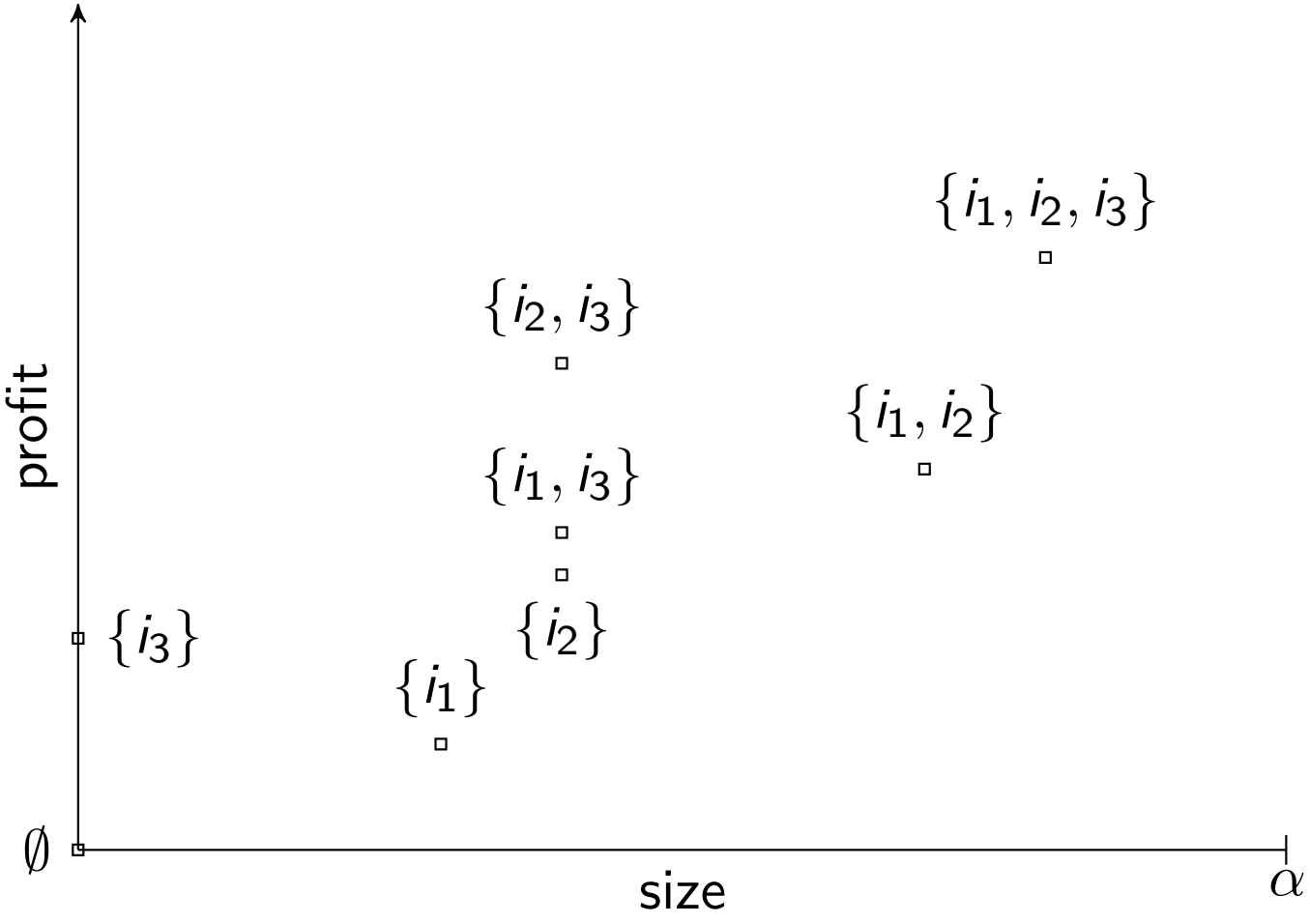
1.5 New: Normalize sizes



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

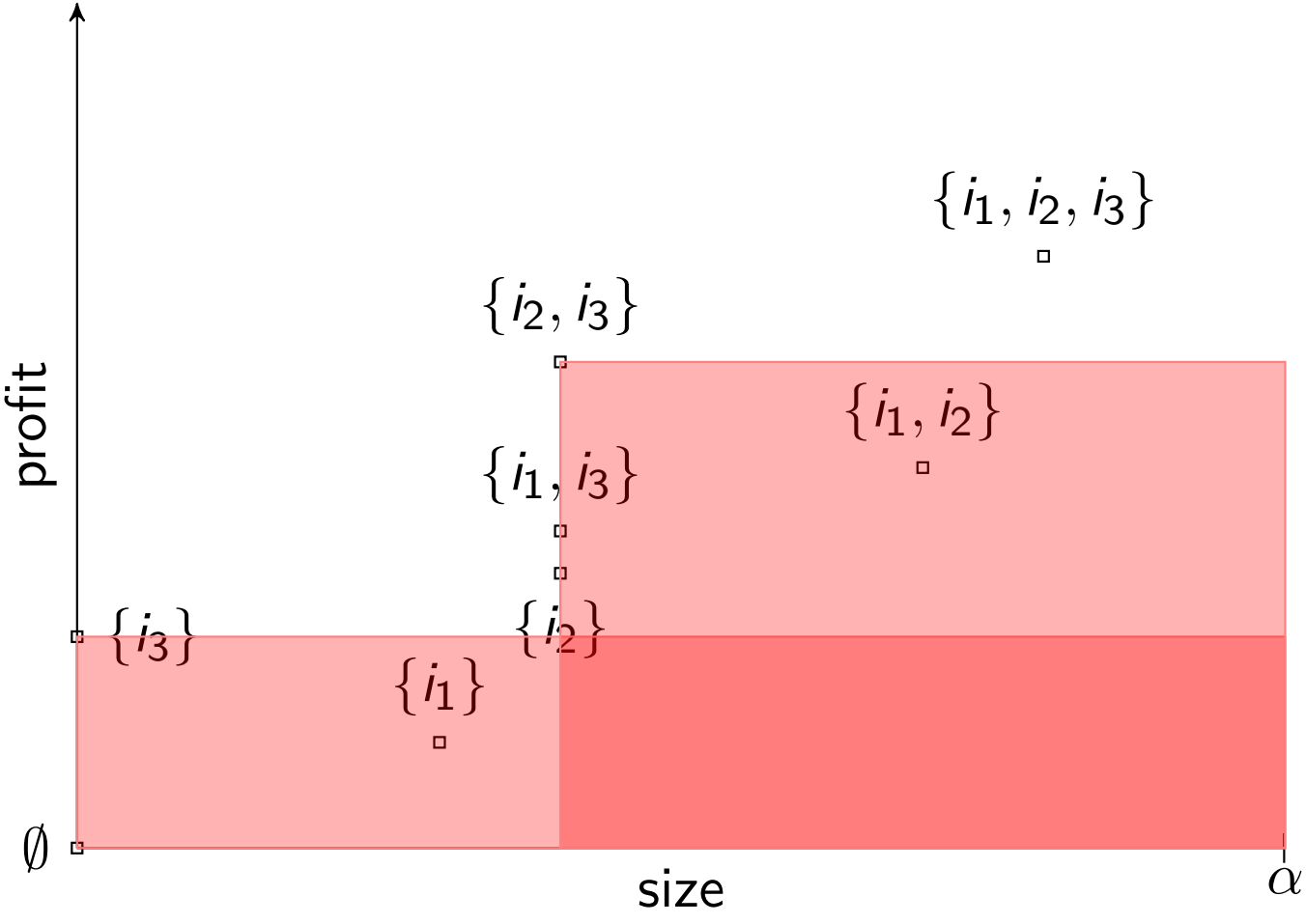
2 Remove dominated solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

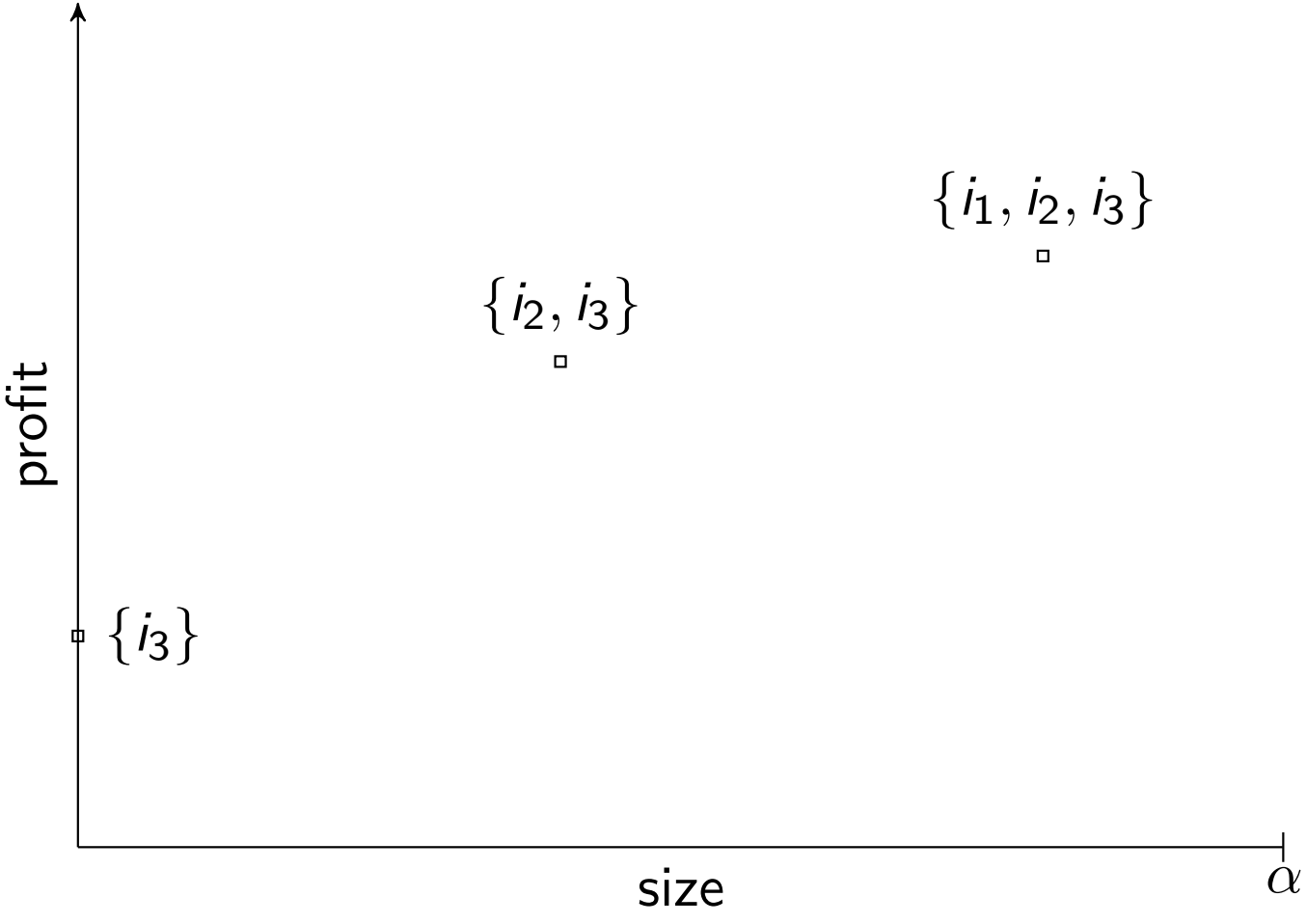
2 Remove dominated solutions



Solve Knapsack with Compressible Items

Dynamic programming framework [Lawler, 1979]

2 Remove dominated solutions



Solve Knapsack with Compressible Items

- Choose $U = \frac{\rho}{(1-\rho)n}\alpha$
- $\lceil \frac{\alpha}{U} \rceil = \mathrm{O}(\frac{n}{\rho})$ many intervals (at most one solution per interval not dominated)
- Solution may use up to $\alpha + nU$ machines
- After compression: uses at most

$$(1-\rho) \times (\alpha + nU) = (1-\rho) \times \left(1 + \frac{\rho}{1-\rho}\right)\alpha = \alpha$$

machines

Solve Knapsack with Compressible Items

- Choose $U = \frac{\rho}{(1-\rho)n}\alpha$
- $\lceil \frac{\alpha}{U} \rceil = \mathrm{O}(\frac{n}{\rho})$ many intervals (at most one solution per interval not dominated)
- Solution may use up to $\alpha + nU$ machines
- After compression: uses at most

$$(1-\rho) \times (\alpha + nU) = (1-\rho) \times \left(1 + \frac{\rho}{1-\rho}\right)\alpha = \alpha$$

  machines

- Running time $\mathrm{O}(n^3)$
- Even more techniques in IPDPS 2018 paper,
- Running time $\mathrm{O}(\frac{n}{\varepsilon^2} \log m \, (\frac{\log(m)}{\varepsilon} + \log^3(\varepsilon m)))$

Improvement (Euro-par 2023)

- Define $S_\rho = [b] \cup \{\lfloor (1+\rho)^i b \rfloor : i \in [\lceil \log_{1+\rho}(m/b) \rceil]$ for $\rho = \varepsilon/4$ and $b = 1/\rho$.
- Then $|S_\rho| = O(\frac{1}{\varepsilon} + \frac{\log(\varepsilon m)}{\varepsilon})$.

Improvement (Euro-par 2023)

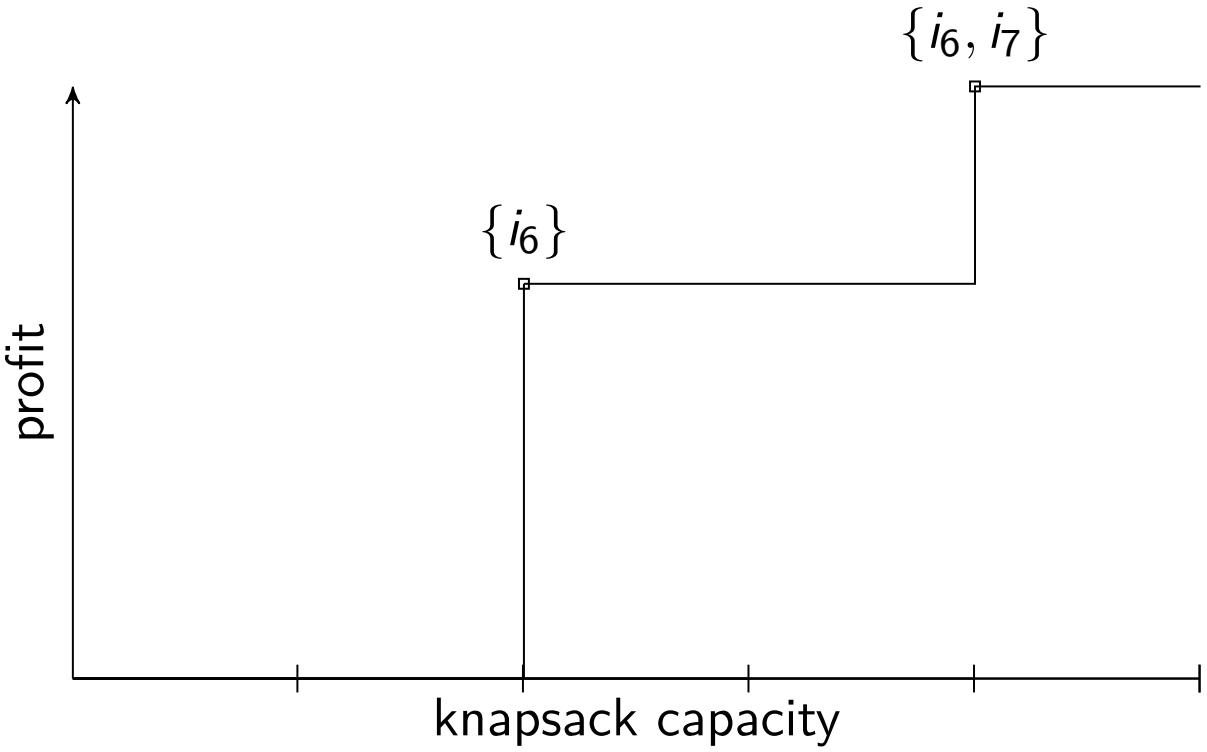
- Define $S_\rho = [b] \cup \{\lfloor (1+\rho)^i b \rfloor : i \in [\lceil \log_{1+\rho}(m/b) \rceil]$ for $\rho = \varepsilon/4$ and $b = 1/\rho$.
- Then $|S_\rho| = O(\frac{1}{\varepsilon} + \frac{\log(\varepsilon m)}{\varepsilon})$.
- Reduce machine counts to

$$\gamma'(j,s) = \max\{k \in S_\rho : k \leq \gamma(j,s)\}$$

for $s \in \{d/2, d\}$.

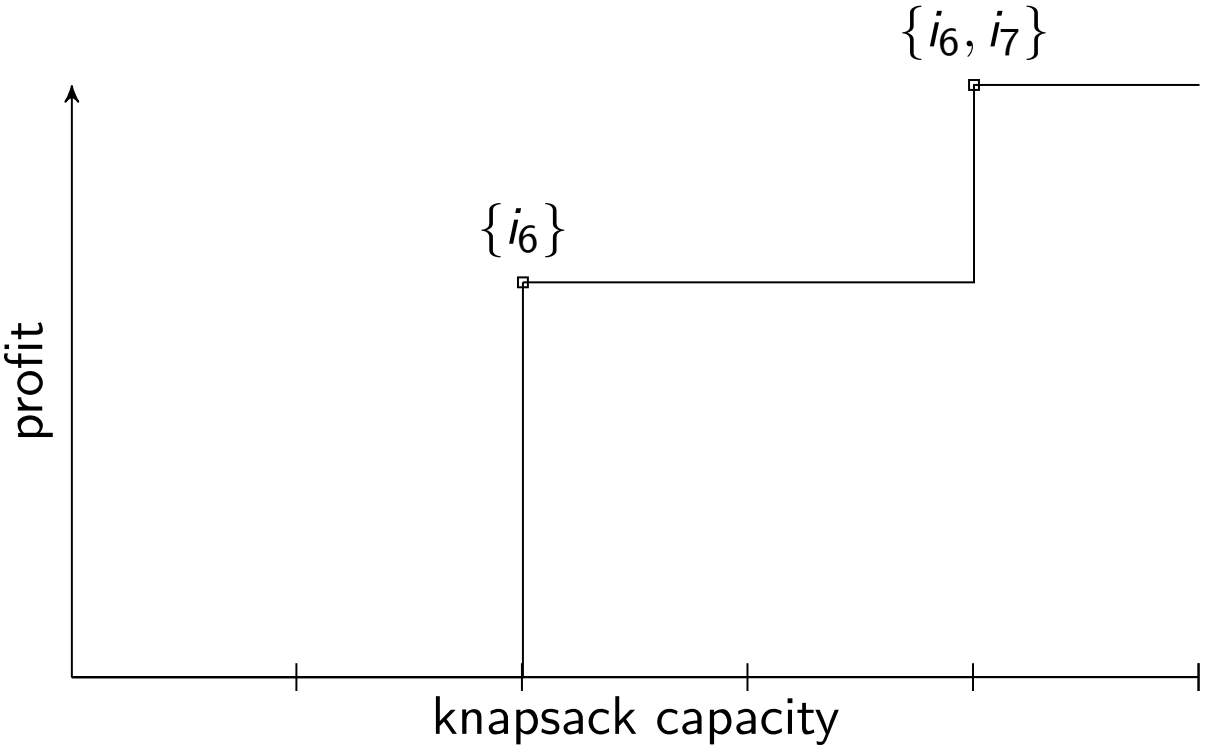
Solving Knapsack (Axiotis and Tzamos)

1 Construct a solution sequence R_i for each item weight $w_i \in S_\rho$ separately



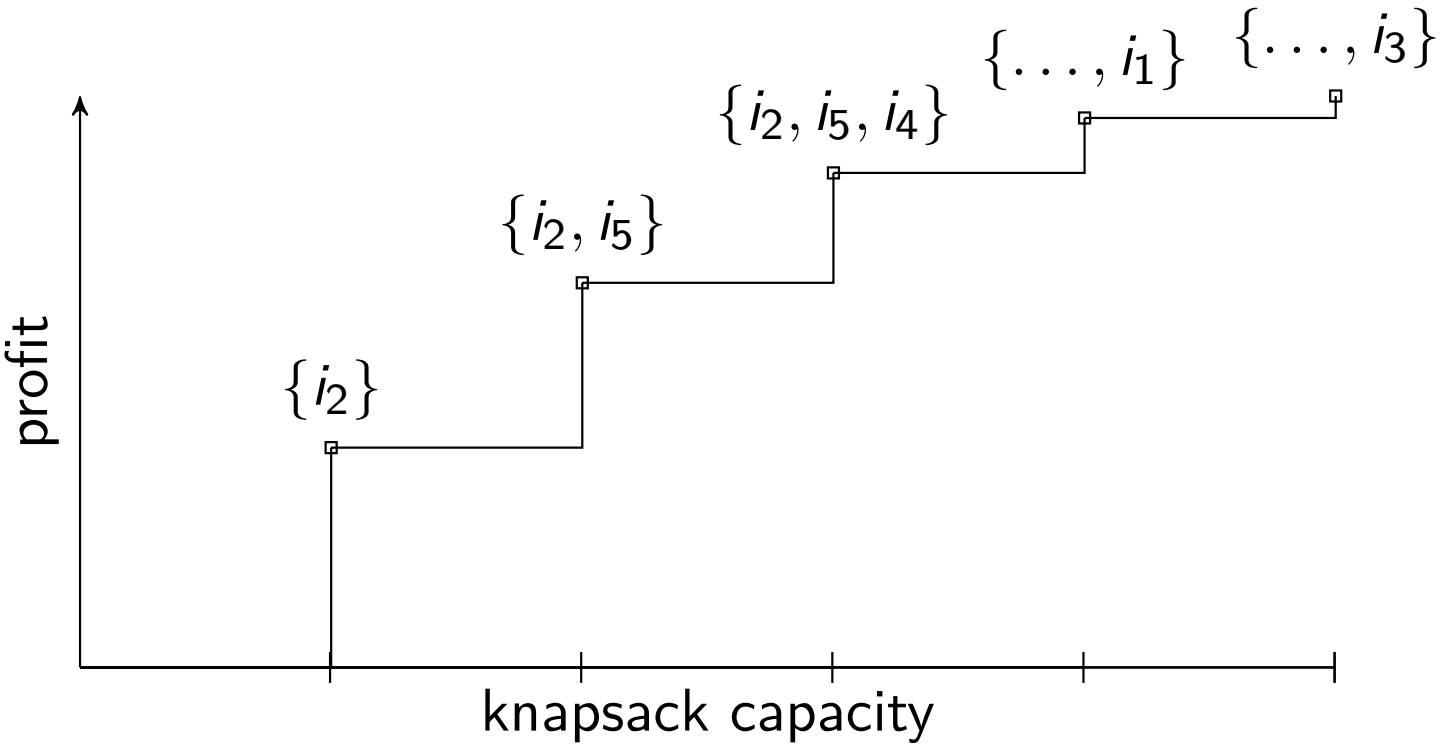
Solving Knapsack (Axiotis and Tzamos)

1 Construct a solution sequence R_i for each item weight $w_i \in S_\rho$ separately
   1.1 $w(i_6) = w(i_7) \in S_\rho$



Solving Knapsack (Axiotis and Tzamos)

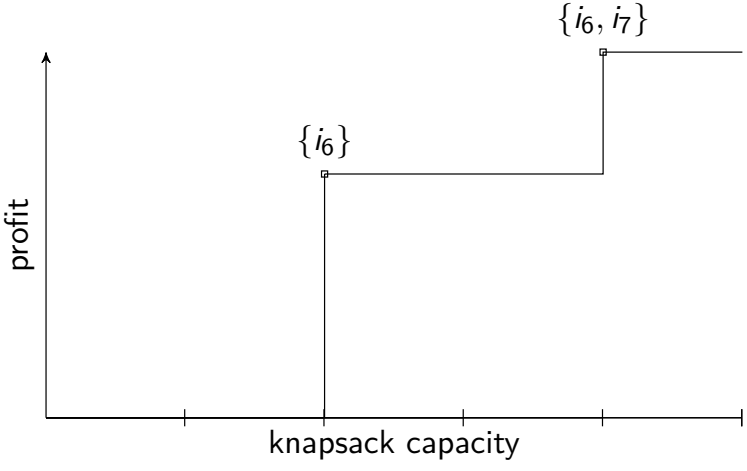
1 Construct a solution sequence R_i for each item weight $w_i \in S_\rho$ separately
   1.2 $w(i_1) = w(i_2) = w(i_3) = w(i_4) = w(i_5) \in S_\rho$

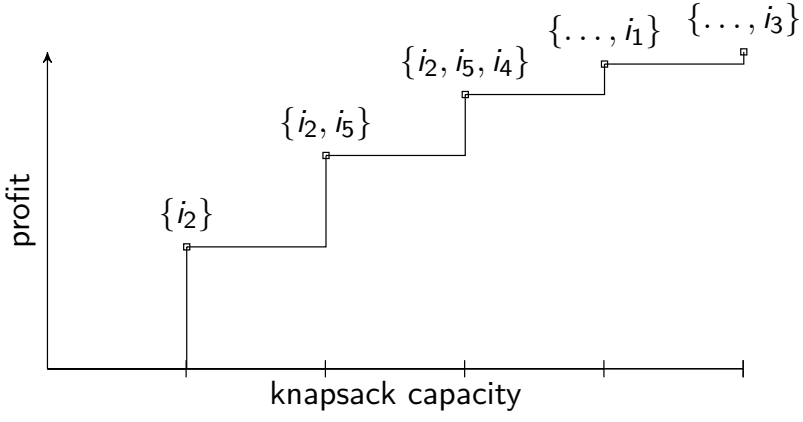


Solving Knapsack (Axiotis and Tzamos)

2 Combine $(((R_1 \oplus R_2) \oplus ...) \oplus R_{|S_\rho|})$ via *(max,+)-convolution*

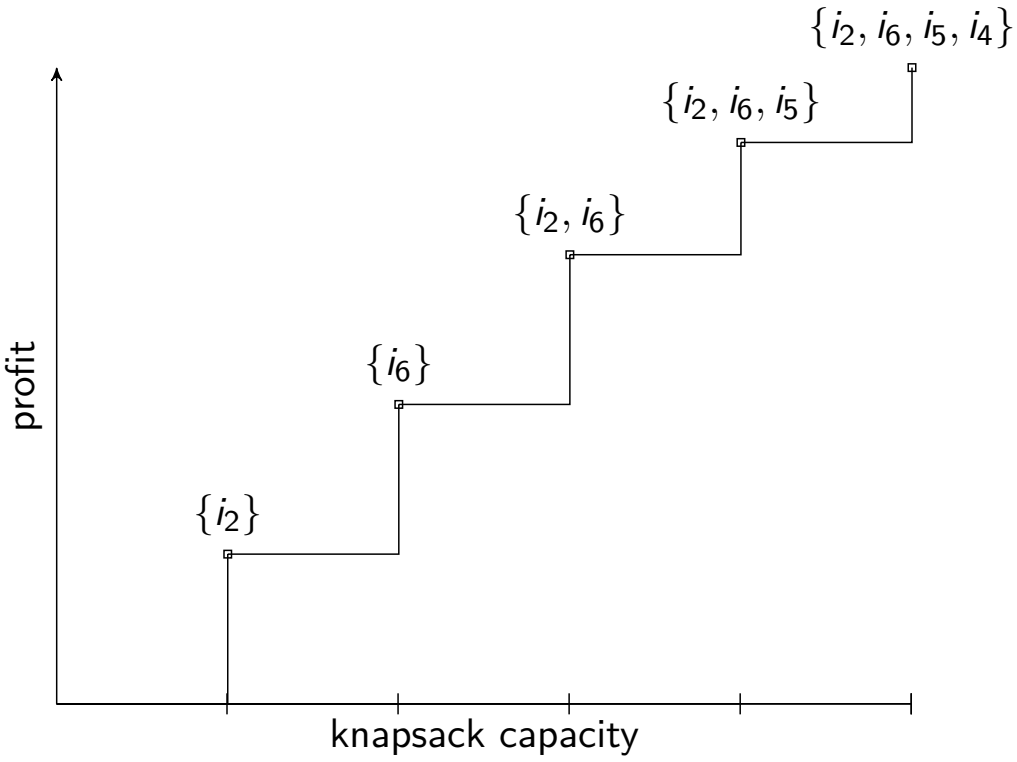
- efficient for *k-step concave* sequences





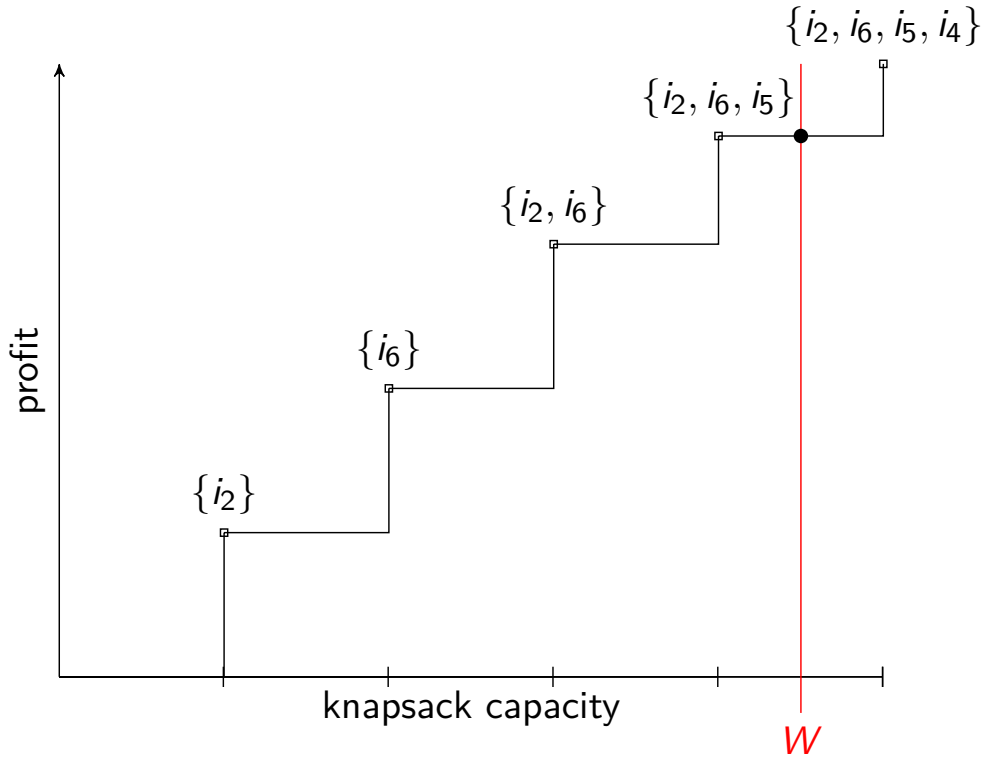
Solving Knapsack (Axiotis and Tzamos)

2 Combine $(((R_1 \oplus R_2) \oplus ...) \oplus R_{|S_\rho|})$ via *(max,+)-convolution*
- efficient for *k-step concave* sequences



Solving Knapsack (Axiotis and Tzamos)

3 Final solution is given in by entry at capacity W



Improvement (Euro-par 2023)

- Distinguish between wide/narrow jobs in S_1 and S_2:
  - Jobs narrow in both shelfs: profit is reduced to next multiple of εd.
  - Jobs wide in both shelfs: modify processing time to $\frac{s}{1+4\rho}$, work to $\frac{s}{1+4\rho}\gamma'(j,s)$ for $s \in \{d/2, d\}$.

Improvement (Euro-par 2023)

- Distinguish between wide/narrow jobs in S_1 and S_2:
  - Jobs narrow in both shelfs: profit is reduced to next multiple of εd.
  - Jobs wide in both shelfs: modify processing time to $\frac{s}{1+4\rho}$, work to $\frac{s}{1+4\rho}\gamma'(j,s)$ for $s \in \{d/2, d\}$.
  - Jobs narrow in one shelf and wide in the other: $\gamma'(j,d/2) \geq b > \gamma'(j,d)$
    - wide part: use processing time $\frac{1}{1+4\rho}\frac{d}{2}$ and work $w'(j,d/2) = \frac{1}{1+4\rho}\frac{d}{2}\dot{\gamma}'(j,d/2)$
    - narrow part: round down the work to next multiple of εd

Improvement (Euro-par 2023)

- Distinguish between wide/narrow jobs in S_1 and S_2:
  - Jobs narrow in both shelfs: profit is reduced to next multiple of εd.
  - Jobs wide in both shelfs: modify processing time to $\frac{s}{1+4\rho}$, work to $\frac{s}{1+4\rho}\gamma'(j,s)$ for $s \in \{d/2, d\}$.
  - Jobs narrow in one shelf and wide in the other: $\gamma'(j,d/2) \geq b > \gamma'(j,d)$
    - wide part: use processing time $\frac{1}{1+4\rho}\frac{d}{2}$ and work $w'(j,d/2) = \frac{1}{1+4\rho}\frac{d}{2}\dot{\gamma}'(j,d/2)$
    - narrow part: round down the work to next multiple of εd
- Use knapsack algorithm by Axiotis and Tzamos (partition items into sets of the same weight, solve separately for each weight and combine solutions with $(\max,+)$-convolution).
- Running time $\mathrm{O}(n\log^2(\frac{1}{\varepsilon}+\frac{\log(\varepsilon m)}{\varepsilon})+\frac{n}{\varepsilon}\log(\frac{1}{\varepsilon})\log(\varepsilon m))$

Conclusion

- Resolved complexity for monotone jobs
  - Previous: Weakly NP-hardness
  - New: Strongly NP-hardness
- Resolved approximability for monotone jobs
  - Previous: $(\frac{3}{2} + \varepsilon)$ and $\frac{3}{2}$-approximation
  - New: Polynomial time approximation scheme (PTAS)
- Efficient algorithms for monotone jobs
  - New: FPTAS when $m \geq \frac{8}{\varepsilon} n$
  - New: $(\frac{3}{2} + \varepsilon)$-approximation: fully polynomial, linear in n/ε

Open Questions

- Can we further reduce running time of our algorithms?
  - via new approximation schemes for knapsack
  - via new parameterized algorithms for knapsack

Open Questions

- Can we further reduce running time of our algorithms?
  - via new approximation schemes for knapsack
  - via new parameterized algorithms for knapsack
- Can we get a faster PTAS for $m \leq O(n/\varepsilon)$ using monotony?

Open Questions

- Can we further reduce running time of our algorithms?
  - via new approximation schemes for knapsack
  - via new parameterized algorithms for knapsack
- Can we get a faster PTAS for $m \leq O(n/\varepsilon)$ using monotony?
- Can we achieve more efficient approximation algorithms with ratio $3/2$ or even $< 3/2$?