Approximation schemes

Scheduling problems

Polynomial Time Approximation Scheme (PTAS)

Let Π be a minimization problem.

- An **approximation scheme** for problem Π is a family of $(1+\varepsilon)$-approximation algorithms A_ε for problem Π over all $0 < \varepsilon < 1$.
- A polynomial time approximation scheme (PTAS) for problem Π is an approximation scheme whose time complexity is polynomial in the input size.

Fully Polynomial Time Approximation Scheme (FPTAS)

- A fully polynomial time approximation scheme (FPTAS) for problem X is an approximation scheme whose time complexity is polynomial in the input size and also polynomial in $1/\varepsilon$.

How design a PTAS or FPTAS

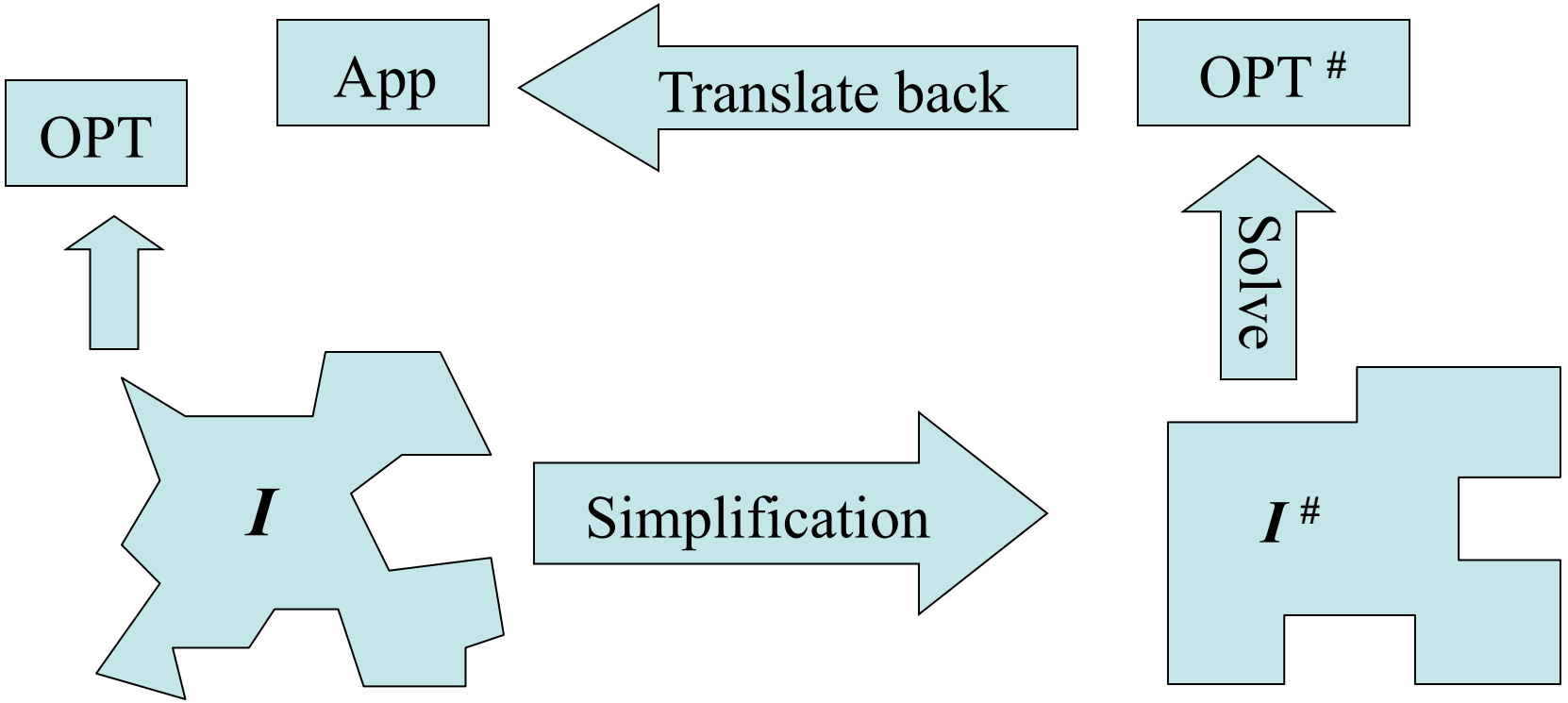
- Let us start by considering an exact algorithm A that solves problem X to optimality. Algorithm A takes an instance I of X, processes it for some time, and finally outputs the solution $A(I)$ for instance I.
- Since the optimization problem X is difficult to solve, the exact algorithm A will have a bad (exponentional) time complexity and will be far away from yielding a PTAS or yielding an FPTAS.
- How can we improve the behavior of such an algorithm and bring it closer to PTAS?

Add structure …

- The addition of structure to the input.
- The addition of structure to the output.
- The addition of structure to the execution of the algorithm *A*.

Structuring the input

The main idea is to turn a difficult instance into a more primitive instance that is easier to tackle. Then we use the optimal solution for the primitive instance to get a grip on the original instance.



Three-step Procedure

A. **Simplify.** Simplify instance I into a more primitive instance $I^{\#}$. This simplification depends on the desired precision ε of approximation; the closer ε is to zero, the closer instance $I^{\#}$ should resemble instance I. The time needed for the simplification must be polynomial in the input size.

B. **Solve.** Determine an optimal solution $\mathrm{OPT}^{\#}$ for the simplified instance $I^{\#}$ in polynomial time.

C. **Translate back.** Translate the solution $\mathrm{OPT}^{\#}$ for $I^{\#}$ back into an approximate solution App will stay close to $\mathrm{OPT}^{\#}$ which in turn is close to OPT. In this case we find an excellent approximation.

Happy medium

- Of course, finding the right simplification in step (A) is not simple.
- If instance $I^{\#}$ is chosen too close to the original instance I, then $I^{\#}$ might still be NP-hard to solve to optimality.
- On the other hand, if instance $I^{\#}$ is chosen to far away from the original instance I, then solving $I^{\#}$ will not tell us anything about how to solve I.

Standard approaches (1, 2)

- **Rounding.** The simplest way of adding structure to the input is to **round** some of the numbers in the input. For instance, we may round all job lengths to perfect powers of two, or we may round non-integral due dates up to the closest integers.
- **Cutting.** Another way of adding structure is to cut away irregular shaped pieces from the instance. For instance, we may remove a small set of jobs with a broad spectrum of processing times from the instance.

Standard approaches (3, 4)

- **Merging.** Another way of adding structure is to **merge** small pieces into larger pieces of primitive shape. For instance, we may merge a huge number of tiny jobs into a single job with processing time equal to the processing time of the tiny jobs.
- **Aligning.** Another way of adding structure to the input is to **align** the shapes of several similar items. For instance, we may replace ninety-nine different jobs of roughly equal length by ninety-nine identical copies of the job with median length.

$P2||C_{\max}$

- $J=\{1,..., n\}$ is set of jobs.
- M_1 and M_2 are two identical machines.
- $j : p_j > 0$ (i=1,…, n).
- The goal is to schedule the jobs on two identical parallel machines so as to minimize the maximum job completion time, the so-called **makespan** $C_{\max}$.
- All jobs are available at time zero.
- Preemption is not allowed.

Lower bounds

$$p_{sum} = \sum_{j=1}^{n} p_j$$

$$p_{\max} = \max_{j=1}^{n} p_j$$

$$C^*_{\max} \geq L = \max\left\{\frac{p_{sum}}{2}, p_{\max}\right\}$$

(A) How to simplife an instance ($I \rightarrow I^{\#}$)

- **Big** = $\{ j \in J \mid p_j \geq \varepsilon L \}$
  - The instance $I^{\#}$ contains all the big jobs from instance I.
- **Small** = $\{ j \in J \mid p_j < \varepsilon L \}$
  - Let $X = \Sigma_{j \in \mathbf{Small}} p_j$.
  - The instance $I^{\#}$ contains $\lfloor X/\varepsilon L \rfloor$ jobs of length εL.
- The small jobs in I are first glued together to give a long job of length X, and then this long job is cut into lots of chunks of length εL.

- **Observation 6.1**

$$\mathrm{OPT}(I^{\#}) \leq (1+\varepsilon)\mathrm{OPT}(I).$$

Proof

- Denote by X_i the total size of all small jobs on machine M_i in an optimal schedule for I.
- On M_i, leave every big job where it is, and replace the small jobs by $\lceil X_i/\varepsilon L \rceil$ chunks of length εL.
- $\lceil X_1/\varepsilon L \rceil + \lceil X_2/\varepsilon L \rceil \geq \lfloor X_1/\varepsilon L + X_2/\varepsilon L \rfloor = \lfloor X/\varepsilon L \rfloor$
- $\lceil X_i/\varepsilon L \rceil \varepsilon L - X_i \leq (X_i/\varepsilon L + 1)\,\varepsilon L - X_i \leq \varepsilon L$
- $\mathrm{OPT}(I^{\#}) \leq \mathrm{OPT} + \varepsilon L \leq (1+\varepsilon)\mathrm{OPT}(I)$

(B) How to solve the simplified instance

- How many jobs are there in instance $I^{\#}$?
- Each job in $I^{\#}$ has length at least εL.
- The total processing time of all jobs in $I^{\#}$ is at most $p_{sum} \leq 2L$.
- There are at most $I^{\#} \leq 2L/\varepsilon L = 2/\varepsilon$ jobs in instance $I^{\#}$.
- The number of jobs $I^{\#}$ is bounded by a finite constant that only depends on ε and thus is completely independent of the number n of jobs in I.
- We try all possible schedules.
- There are at most $2^{2/\varepsilon}$ possible schedules.
- The makespan of each of these schedules can be determined in $O(2/\varepsilon)$ times .

(C) How to translate the solution back?

- Let $\sigma^{\#}$ be an optimal schedule for the simplified instance $I^{\#}$.
- Let $L_i^{\#}$ be the load of machine M_i in $\sigma^{\#}$,
- $B_i^{\#}$ be the total size of the big jobs on M_i, and
- $X_i^{\#}$ be the size of the small jobs M_i in $\sigma^{\#}$.
- $L_i^{\#} = B_i^{\#} + X_i^{\#}$.

$$X_1^{\#} + X_2^{\#} = \varepsilon L \left\lfloor \frac{X}{\varepsilon L} \right\rfloor > X - \varepsilon L$$

Transformation ($\sigma^{\#}(I^{\#}) \rightarrow \sigma(I)$)

- Every big job is put onto the same machine as in schedule $\sigma^{\#}$.
- We reserve an interval of length $X_1^{\#} + 2\varepsilon L$ on machine M_1, and an interval of length $X_2^{\#}$ on machine M_2.
- We then greedily put the small jobs into these reserved intervals: First, we start packing small jobs into the reserved interval on M_1, until we meet some small job that does not fit any more.
- All remaining unpacked small jobs together will fit into the reserved interval on machine M_2.

Let us compare the loads of the machines in σ to the machine completion times in $\sigma^{\#}$.

$$OPT \geq L = \max\left\{\frac{p_{sum}}{2}, p_{\max}\right\}$$

$$L_i^{\#} \leq OPT^{\#} \overset{\text{L 5.1}}{\leq} (1+\varepsilon)OPT$$

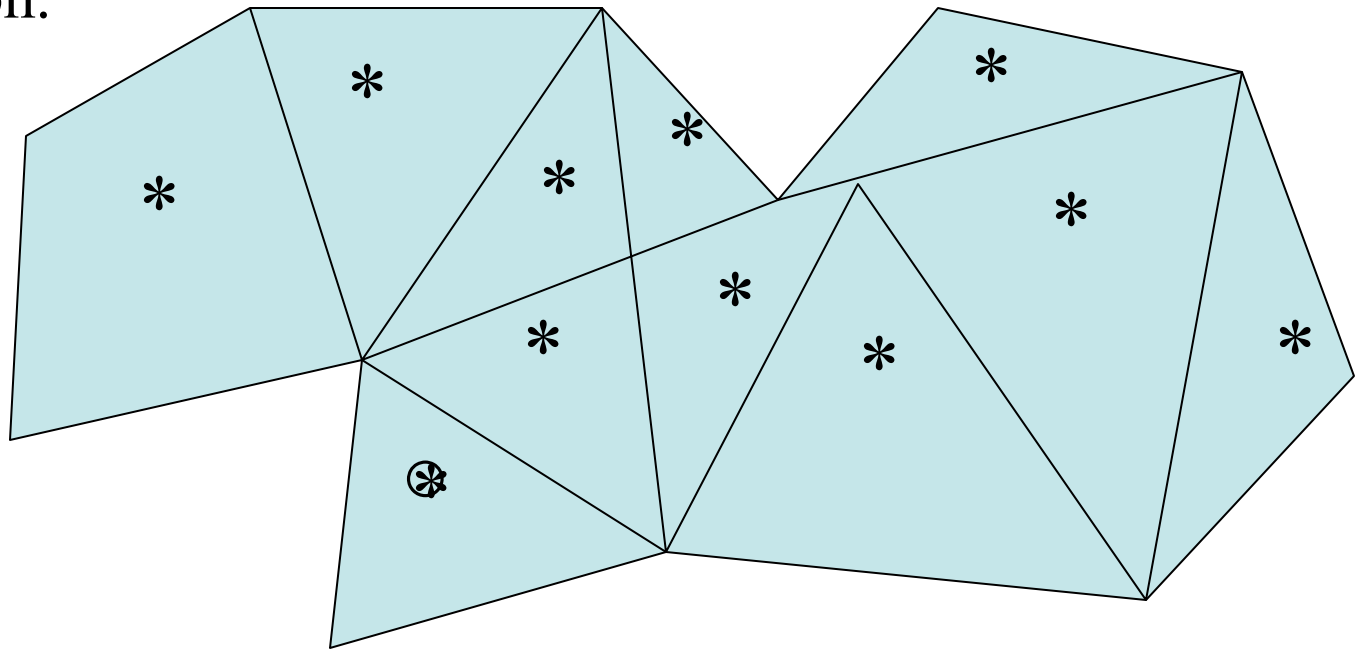
$$L_i \leq B_i^{\#} + \left(X_i^{\#} + 2\varepsilon L\right) = L_i^{\#} + 2\varepsilon L \leq$$

$$\leq (1+\varepsilon)OPT + 2\varepsilon OPT = (1+3\varepsilon)OPT$$

We obtain the first PTAS, say PTAS-1.

Structuring the output

- The main idea is to cut the output space (i.e., the set of feasible solutions) into lots of smaller regions over which the optimization problem is easy to approximate. Tackling the problem separately for each smaller region and taking the best approximate solution over all regions will then yield a globally good approximate solution.



Three-step Procedure

A. Partition. Partition the feasible solution space $\boldsymbol{\Phi}$ of instance I into a number of ***districts*** $\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)},\ldots,\boldsymbol{\Phi}^{(d)}$ such that $\bigcup_{l=1}^{d} \Phi_l = \Phi$. This partition depends on the desired precision ε of approximation. The closer ε is to zero, the finer should this partition be. The number d of districts must be polynomially bounded in the size of the input.

B. Find representatives. For each district $\boldsymbol{\Phi}^{(l)}$ determine a good representative whose objective value $App^{(\ell)}$ is a good approximation of the optimal objective value $\mathrm{Opt}^{(\ell)}$ in $\boldsymbol{\Phi}^{(l)}$. The time needed for finding the representative must be polynomial in the input size.

C. Take the best. Select the best of all representatives as approximate solution with objective value App for instance I.

“Good” running time and a “good” approximation

The overall time complexity of this approach is polynomial:

- There is a polynomial number of districts.
- Each district is handled in polynomial time in step (B).
- Step (C) optimizes over a polynomial number of representatives.
- The globally optimal solution with objective value Opt must be contained in at least one of the districts, say in district $F^{(\ell)}$. Then $Opt = Opt^{(\ell)}$.

Since the representative for $F^{(\ell)}$ gives a good approximation of $Opt^{(\ell)}$, it also yields a good approximation of the global optimum. Hence, also the final output of the algorithm will be a good approximation of the global optimum.

$P2||C_{\max}$

- $J=\{1,..., n\}$ is set of jobs.
- M_1 and M_2 are two identical machines.
- $j : p_j > 0$ (i=1,…, n).
- The goal is to schedule the jobs on two identical parallel machines so as to minimize the maximum job completion time, the so-called **makespan** $C_{\max}$.
- All jobs are available at time zero.
- Preemption is not allowed.

How to define the districts

- **Big** = $\{ j \in J | \, p_j \geq \varepsilon L \}$
- **Small** = $\{ j \in J | \, p_j < \varepsilon L \}$
- Let $\boldsymbol{\Phi}$ be a set of feasible solutions.
- Every feasible solution $\sigma \in \boldsymbol{\Phi}$ specifies an assignment of the n jobs to the two machines.
- We define the districts $\boldsymbol{\Phi}^{(1)}$, $\boldsymbol{\Phi}^{(2)}$,… . according to the assignment of the big jobs to the two machines: Two feasible solutions σ_1 and σ_2 lie in the same district if and only if σ_1 assigns every big jobs to the same machine as σ_2 does.
- Note that the assignment of the small jobs remains absolutely free.

How many districts we obtain?

- There are at most $2L/\varepsilon L = 2/\varepsilon$ big jobs.
- There are at most $2^{2/\varepsilon}$ different ways for assigning these jobs to two machines.
- The number of districts in our partition is bounded by $2^{2/\varepsilon}$!
- This value is independent of the input size.

How to find good representatives

- Consider a fixed district $\boldsymbol{\Phi}^{(l)}$, and denote by $\mathrm{OPT}^{(l)}$ the makespan of the best schedule in this district. In $\boldsymbol{\Phi}^{(l)}$ the assignments of the big jobs to their machines are fixed, and we denote by $B_i^{(l)}$ (i = 1, 2) the total processing time of big jobs assigned to machine M_i.
- $T := \max\{B_i^{(1)}, B_i^{(2)}\} \leq \mathrm{OPT}^{(l)}$.
- We assign the small jobs one by one to the machines; every time a job is assigned, it is put on the machine with the currently smaller workload.
- The resulting schedule $\sigma^{(l)}$ with makespan $A^{(l)}$ is our representative for the district $\boldsymbol{\Phi}^{(l)}$. Clearly, $\sigma^{(l)}$ is computable in polynomial time.

How close is $A^{(I)}$ to $\mathrm{OPT}^{(I)}$?

1. In case $A^{(I)} = T$ holds, we have $A^{(I)} = \mathrm{OPT}^{(I)}$.
2. In case $A^{(I)} > T$ holds, we consider the machine M_i with higher workload in the schedule $\sigma^{(I)}$.
   - Then the last job that was assigned to M_i is a small job and thus has processing time at most εL.
   - At the moment when this small job was assigned to M_i the workload of M_i was at most $p_{sum} / 2$.
   - $A^{(I)} \leq (p_{sum} / 2) + \varepsilon L \leq (1 + \varepsilon)\mathrm{OPT} \leq (1 + \varepsilon)\mathrm{OPT}^{(I)}$

We obtain the second PTAS, say PTAS-2.

Structuring the execution of an algorithm

- The main idea is to take an exact but slow algorithm A, and to interact with it while it is working.
- The algorithm accumulates a lot of auxiliary data during its execution, then we may remove part of this data and clean up the algorithm's memory. As a result the algorithm becomes faster (since there is less data to process) and generates an incorrect output (since the removal of data introduces errors).
- In the ideal case, the time complexity of the algorithm becomes polynomial and the incorrect output constitutes a good approximation of the true optimum.

Some ideas

- This approach can only work out if the algorithm itself is highly structured.
- Let us consider rather primitive algorithms that do not even try to optimize something. They simply generate all feasible solutions and only suppress obvious duplicates. They are of a severely restricted form and work in a severely restricted environment.

$P2||C_{\max}$

- $J=\{1,..., n\}$ is set of jobs.
- M_1 and M_2 are two identical machines.
- $j : p_j > 0$ (i=1,…, n).
- The goal is to schedule the jobs on two identical parallel machines so as to minimize the maximum job completion time, the so-called **makespan** $C_{\max}$.
- All jobs are available at time zero.
- Preemption is not allowed.

Encoding of solutions

- Let σ_k be a feasible schedule for k jobs $\{1,..., k\}$.
- We encode a feasible schedule σ_k with machine loads L_1 and L_2 by the two-dimensional vector $[L_1, L_2]$.
- Let V_k be the set of all vectors, that corresponded to feasible schedules of k jobs $\{1,..., k\}$.

Dynamic programming

Input ($J=\{1,\ldots, n\}$, $p: J \rightarrow \mathbf{Z}^+$)

1) Set $V_0=\{[0,0]\}$, $i=0$.

2) **While** $i \neq n$ **do:**

   for every vector $[x,y]$ in V_i put $[x + p_i ,y]$
   and $[x,y + p_i]$ in V_{i+1};
   $i := i +1$;

3) Let $[x^*,y^*] \in V_n$ be the vector that minimizes the value $\max_{[x,y]\in V_n}\{x,y\}$.

Output ($[x^*,y^*]$)

Running time

- The coordinates of all vectors in all sets V_i are integers in the range from 0 to p_{sum}.
- The cardinality of every vector set V_i is bounded from above by $(p_{sum})^2$.
- The time complexity of DP is proportional to $O(n(p_{sum})^2)$.
- Size of the input $|I|$ is bounded by $O(\log(p_{sum}))=O(\ln(p_{sum}))$ or $O(n \log p_{\max})$.
- The overall time complexity of DP will be exponential in the size of the input, and hence algorithm A will not have polynomial time complexity.
- The algorithm DP has a pseudo-polynomial time complexity.

How to simplify the vector sets?

- All considered vectors correspond to geometric points in the rectangle $[0, p_{sum}]\times[0, p_{sum}]$.
- We subdivide this rectangle with horizontal and vertical cuts into lots of boxes.
- In both directions these cuts are made at the coordinates Δ^i, where $\Delta = 1+ (\varepsilon/2n)$, $i = 1, 2, \ldots, K$.
- $K = \lceil \log_\Delta(p_{sum})\rceil = \lceil \ln(p_{sum})/\ln \Delta\rceil = \lceil ((1+2n)/\varepsilon) \ln(p_{sum})\rceil$.

Selection of vectors

- Let two vectors $[x_1,y_1]$ and $[x_2,y_2]$ are in the same box.
- $x_1/\Delta \leq x_2 \leq x_1\Delta$ and $y_1/\Delta \leq y_2 \leq y_1\Delta$.
- Out of every box that has non-empty intersection with V_i we select a single vector and put it into the so-called *trimmed vector set* $V_i^{\#}$.
- All remaining vectors from the vector set V_i that have not been selected are lost for the further computations.
- And in phase $i + 1$, the so-called *trimmed algorithm generates its new vector* set from the smaller set $V_i^{\#}$ and not from the set V_i.

FPTAS

Input ($J=\{1,\ldots, n\}$, $p: J \rightarrow \mathbf{Z}^+$)

1) Set $V_0^{\#}=\{[0,0]\}$, $i=0$.

2) **While** $i \neq n$ **do:**

for every vector $[x,y] \in V_i^{\#}$ put $[x + p_i, y]$

and $[x, y + p_i]$ в V_{i+1};

$i := i + 1$;

Transform vector set V_i into trimmed vector set $V_i^{\#}$.

3) Let $[x^*,y^*] \in V_n^{\#}$, be the vector that minimizes the value $\max_{[x,y] \in V_n^{\#}} \{x,y\}$

Output ($[x^*,y^*]$)

Running time of FPTAS

- The trimmed vector set $V_i^{\#}$ contains at most one vector from each box in the subdivision.
- Altogether there are K^2 boxes.
- The running time of FPTAS $O(nK^2)$.
- $nK^2 = n\lceil((1+2n)/\varepsilon)\ln(p_{sum})\rceil^2$
- FPTAS has a time complexity that is polynomial in the input size and in $1/\varepsilon$.

How to analyze the worst case behavior?

- **Lemma 6.2**
- For every vector $[x,y] \in V_i$ there exists a vector $[x^{\#},y^{\#}] \in V_i^{\#}$, such that $x^{\#} \leq \Delta^i x$ and $y^{\#} \leq \Delta^i y$.

Proof (by induction)

- $i = 1$: $(x_1/\Delta \le x_2 \le x_1\Delta$ и $y_1/\Delta \le y_2 \le y_1\Delta)$
- $i-1 \rightarrow i$: consider an arbitrary vector $[x,y] \in V_i$.
- The untrimmed algorithm puts this vector into V_i when it adds job J_i to some feasible schedule for the first $i-1$ jobs.
- It follows that $\exists\, [a,b] \in V_{i-1}$, either $[x,y] = [a+p_k,b]$, or $[x,y] = [a,b+p_k]$.
- Hence, $\exists\, [a^{\#},b^{\#}] \in V_{i-1}^{\#}$: $a^{\#} \le \Delta^{i-1}a$, $b^{\#} \le \Delta^{i-1}b$.
- Algorithm FPTAS generates vector $[a^{\#}+p_k, b^{\#}]$ and select $[\alpha,\beta]$ such that $\alpha \le \Delta(a^{\#}+p_k)$ и $\beta \le \Delta b^{\#}$.
- We get $\alpha \le \Delta(a^{\#}+p_k) \le \Delta^i a + \Delta p_k \le \Delta^i(a+p_k) = \Delta^i x$ and $\beta \le \Delta^i y$.

We obtain FPTAS.

$$\max\left\{x^{\#}, y^{\#}\right\} \overset{L\,5.4}{\leq} \max\left\{\Delta^n x, \Delta^n y\right\} = \Delta^n \max\left\{x, y\right\} = \Delta^n OPT$$

$$\left(1+\frac{z}{n}\right)^n \leq 1+2z \Leftarrow (0 \leq z \leq 1)$$

$$\Delta^n OPT = \left(1+\frac{\varepsilon}{2n}\right)^n OPT \leq (1+\varepsilon) OPT$$

Exercise

- In the PTAS-1 for P2||$C_{\max}$, we replaced the small jobs in instance I by lots of chunks of length εL in instance $I^{\#}$. Consider the following alternative way of handling the small jobs in I: Put all the small jobs into a canvas bag. While there are at least two jobs with lengths smaller than εL in the bag, merge two such jobs. That is, repeatedly replace two jobs with processing times $p', p'' \leq \varepsilon L$ by a single new job of length $p' + p''$. The simplified instance $I^{\#}$ consists of the final contents of the bag. Will this lead to another PTAS for P2||$C_{\max}$?
- Does the Observation 6.1 still hold true?
- How can you bound the number of jobs in the simplified instance $I^{\#}$?
- How would you translate an optimal schedule for $I^{\#}$ into an approximate schedule for I?