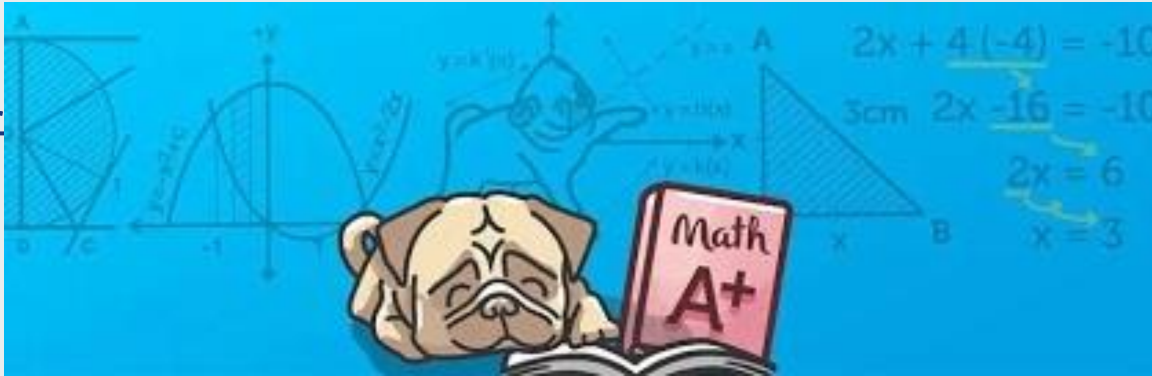
Algoritmi Avansați 2021
c-2
Algoritmi ρ-aproximativi

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.r

Grup Teams:



Din cursul anterior

Recapitulare:

reamintit ce estee acela un algoritm

complexitatea unui algoritm

timp determinist vs nedeterminist

crash-course in ce inseamna P, NP, NPC

Cursul prezent

- Motivație
- Terminologie de baza
- Un prim exemplu de algoritm aproximativ
- Un exemplu mai detaliat
- Un început pt Tema 1

Motivație

Q: Daca avem nevoie să aflăm raspunsul la o problemă NP-hard?
A: Nu prea sunt șanse să găsim un algoritm care să ruleze în timp polinomial

Așa că....

Motivație

Trebuie să renunțăm măcar la unul dintre următoarele 3 elemente:

1. Găsirea unui algoritm polinomial pentru problemă
2. Găsirea unui algoritm general (pentru o instanță oarecare) a problemei
3. Găsirea soluției exacte (optime) pentru problema

Basic Terminology & Notations

Problema de Optim:

Informal spus este problema in care trebuie sa gasesti o “cea mai buna” solutie/constructie fezabila.

“Cea mai buna” - poate avea doua sensuri:

Fie avem o problema de **minimizare** precum Problema Comis-voiajorului.

FIe o problema de **maximizare** precum cea de a găsi o acoperire de cardinal maxim pentru multimea varfurilor unui graf

Basic Terminology & Notations

Problema de Optim:

Fie *P* - o problema de optim, și *I* o intrare pe aceasta problema. Vom nota cu *OPT(I)* ”valoarea” soluției optime.

În mod analog, atunci când propunem un algoritm care să ofere o soluție fezabilă pentru problema noastră, vom nota ”valoarea” acelei soluții cu *ALG(I)*.

De cele mai multe ori, atunci când nu se crează confuzie, vom simplifica notațiile folosind termenii ”OPT”, respectiv ”ALG”

Pe parcursul prezentării vom presupune că atât *OPT*, cât și ALG sunt ≥ 0.

Basic Terminology & Notations

Problema de Optim:

Pentru a justifica un algoritm este *util*, acesta trebuie însoțit de o justificare că soluția oferită este fezabilă pentru problema, precum și o relație între *ALG* și *OPT*. Aceast tip de relație este descrisă astfel:

Definiție **1**

- Un algoritm ALG pentru o problema de **minimzare** se numește ρ-aproximativ, pentru o valoare $\rho > 1$, dacă $ALG(I) \leq \rho \cdot OPT(I)\ pt\ \forall I$ – intrare

- Un algoritm ALG pentru o problema de **maximizare** se numește ρ-aproximativ, pentru o valoare $\rho < 1$, dacă $ALG \geq \rho \cdot OPT(I)\ pt\ \forall I$ – intrare

Basic Terminology & Notations

OBSERVAȚIE

(pt probleme de minim) Orice algoritm ρ-aproximativ este la rândul lui ρ'-aproximativ pentru orice ρ'>ρ. De aceea, în cazul unui algoritm ALG pentru o problemă de minimizare, spre exemplu, trebuie ca justificarea ce însoțește pe ALG să ofere cea mai mică valoare ρ pentru care ALG este ρ-aproximativ.

Definiție 1

- Un algoritm ALG pentru o problema de **minimzare** se numește ρ-aproximativ, pentru o valoare $\rho > 1$, dacă $ALG(I) \leq \rho \cdot OPT(I)\ pt\ \forall I$ – intrare

- Un algoritm ALG pentru o problema de **maximizare** se numește ρ-aproximativ, pentru o valoare $\rho < 1$, dacă $ALG \geq \rho \cdot OPT(I)\ pt\ \forall I$ – intrare

Basic Terminology & Notations

***Definiție* 2**

Fie *ALG* un algoritm ρ-aproximativ pentru o problema de minimizare. SPunem că factorul de aproximare este ”tight bounded” atunci când avem $\rho = supremum_I \frac{ALG(I)}{OPT(I)}$

Ca să arătăm că un algoritm este ρ-aproximativ ”tight bounded” , trebuie deci să justificăm următoarele 2 lucruri:

1. Trebuie să arătmăm că este ρ-aproximativ, adică *ALG(I)*≤ρx*OPT(I)* pentru orice intrare *I*
2. Pentru orice ρ’<ρ există un *I* pentru care *ALG(I)*>ρ’x*OPT(I)*. Adesea totuși ne este mai la îndemână să arătăm ca există un *I* pentru care *ALG(I)*=ρx*OPT(I)*

O primă provocare:
1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

O primă provocare:
1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

Presupunere: Fiecare obiect are o greutate mai mică sau egală cu capacitatea rucsacului!

O primă provocare:
1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

Rezolvare propusă:

Fie L *– lista obiectelor sortate după raportul* ${}^{valoare}/_{greutate}$

Fie O_p *– obiectul cu profitul cel mai mare din lista de obiecte.*

$S = 0,\ G = capacitatea\ rucsacului;$

Pentru fiecare $O : L$

    Dacă $greutate(O) \leq G$, *atunci* $S += val(O)$, $G -= greutate(O)$

$ALG(I) = max(S, O_p)$

O primă provocare:
1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului!

Rezolvare propusă:

$Fie\ L$ – *lista obiectelor sortate după raportul* $^{valoare}/_{greutate}$

$Fie\ O_p$ – *obiectul cu profitul cel mai mare din lista de obiecte.*

$S = 0,\ G = capacitatea\ rucsacului;$

$Pentru\ fiecare\ O{:}L$

$\quad Dacă\ greutate(O) \le G,\ atunci\ S += val(O),\ G -= greutate(O)$

$$ALG(I) = max(S, O_p)$$

O primă provocare: 1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului! Justificare S23 / Justificare S24

Rezolvare propusă:

$Fie\ L\ -\ lista\ obiectelor\ sortate\ după\ raportul\ {}^{valoare}/_{greutate}$

$Fie\ O_p\ -\ obiectul\ cu\ profitul\ cel\ mai\ mare\ din\ lista\ de\ obiecte.$

$S=0,\ G=capacitatea\ rucsacului;$

$Pentru\ fiecare\ O:L$

$\quad Dacă\ greutate(O) \le G,\ atunci\ S+=val(O),\ G-=greutate(O)$

$ALG(I)\ =\ max(S, O_p)$

Load Balancing Problem

Input:

- *m* calculatoare identice; *n* activitați ce trebuiesc procesate. Fiecare activitate *j* având nevoie de t_j unități de timp pentru execuție.
- Odată inițiată, fiecare dintre activități trebuie derulată în mod continuu pe același calculator
- Un calculator poate executa cel mult o activitate în același timp.

Scop:

Să asignăm fiecare activitate unui calculator astfel încât să minimizăm timpul până când toate activitățile sunt terminate.

Load Balancing Problem

Notații:

- J(i) -submulțimea tuturor activităților (job-urilor) care au fost programate să se desfășoare pe mașina i.
- L_i va reprezenta ”load-ul” (timpul de lucru) al mașinii i.
- $L_i = \Sigma_{j \in J(i)} t_j$

Scop: ???

Load Balancing Problem

Notații:

- J(i) -submulțimea tuturor activităților (job-urilor) care au fost programate să se desfășoare pe mașina i.
- L_i va reprezenta ”load-ul” (timpul de lucru) al mașinii i.
- $L_i = \Sigma_{j \in J(i)} t_j$

Scop: O asignare a activităților astfel încât L_k este minimizat, unde k = $max_i(L_i)$, adică mașina cu cel mai mare load.

Load Balancing Problem

Pseudocodul:

$Load - Balance(m, t_1, t_2, \ldots, t_n)$

$for\ i = 1\ to\ m:$

$L_i = 0;\ J(i) = \varnothing$ # *initializare: Fiecare Load este 0 iar multimea joburilor este nula pt fiecare masina*

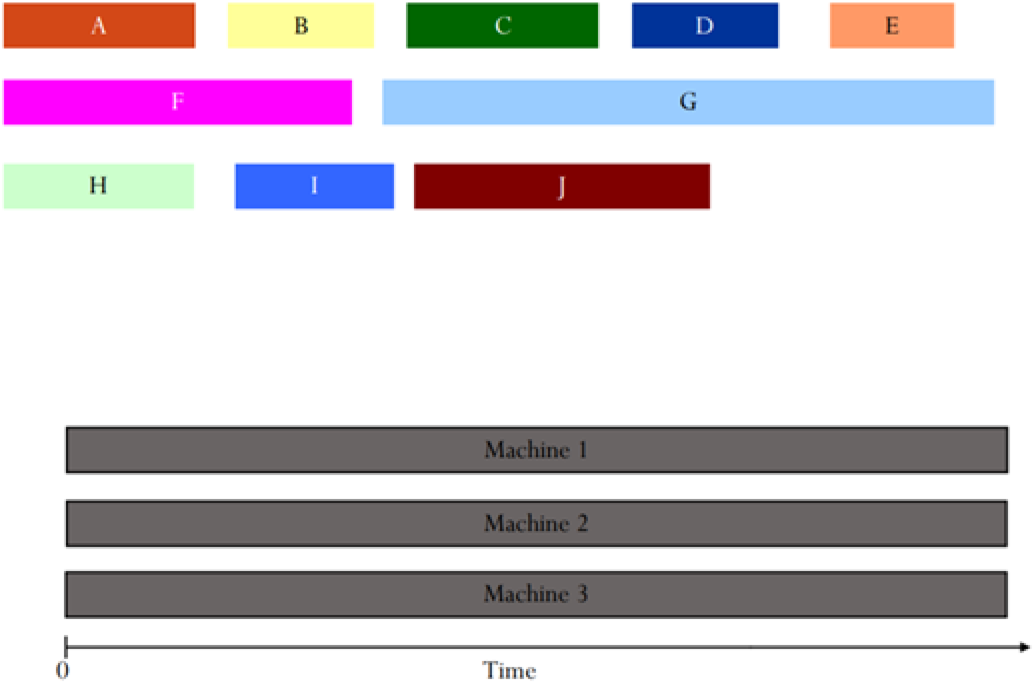
$for\ j = 1\ to\ n:$

$i = arg(min\{L_k \mid k \in \{1, \ldots, m\}\})$ # *i – masina cu incarcatura cea mai mica in acest moment*

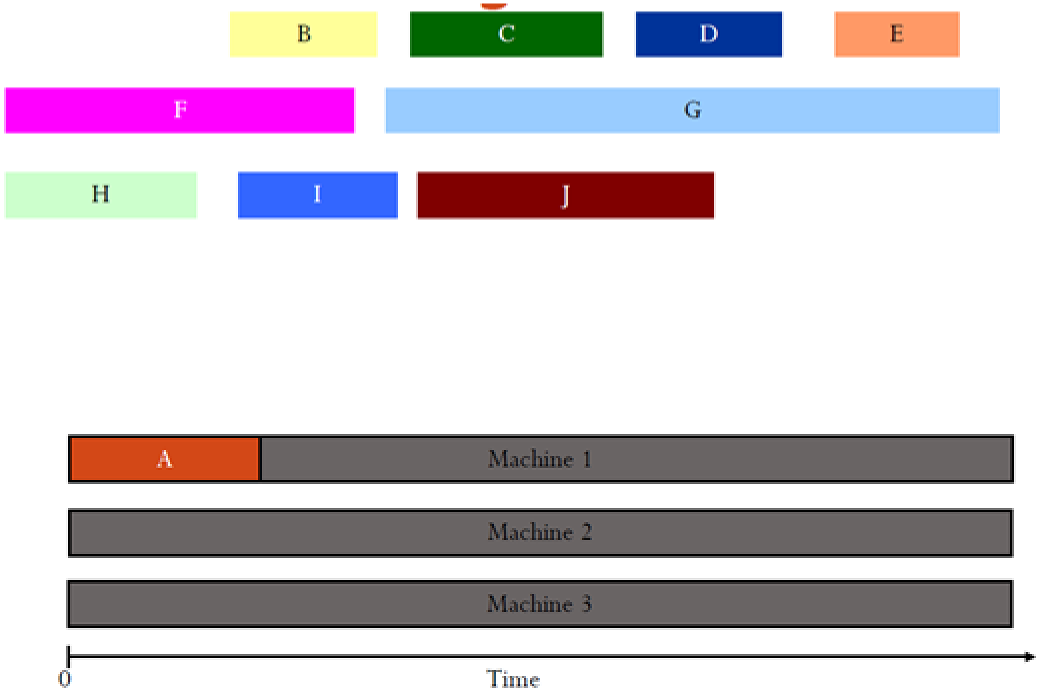
$J(i) = J(i) \cup \{j\}$

$L_i += t_j$

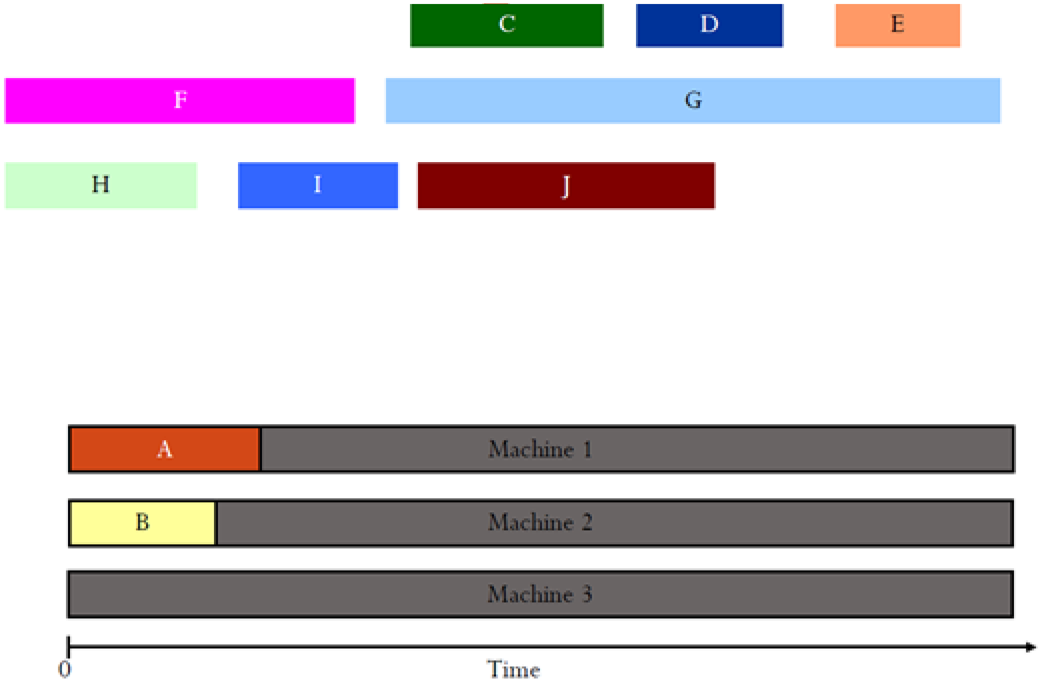
Step-by-step example



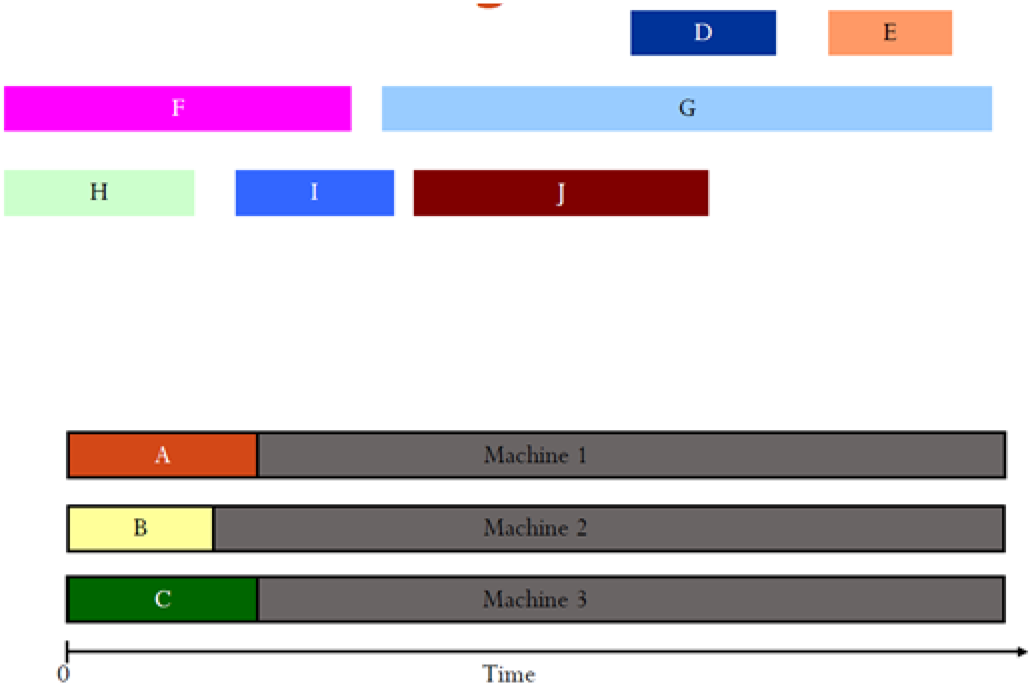
Step-by-step example



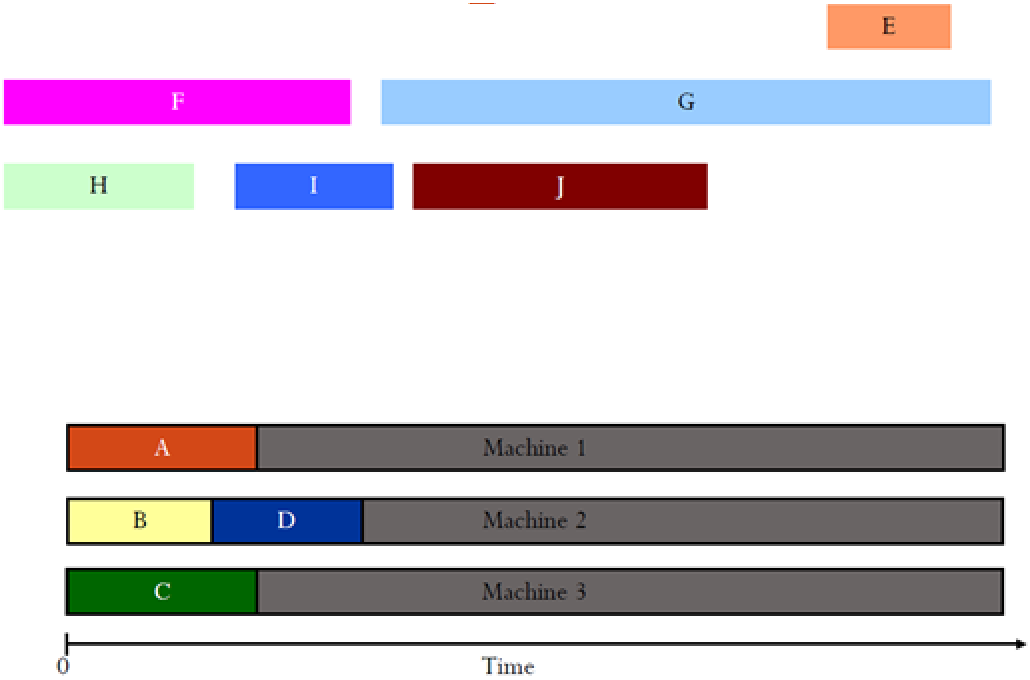
Step-by-step example



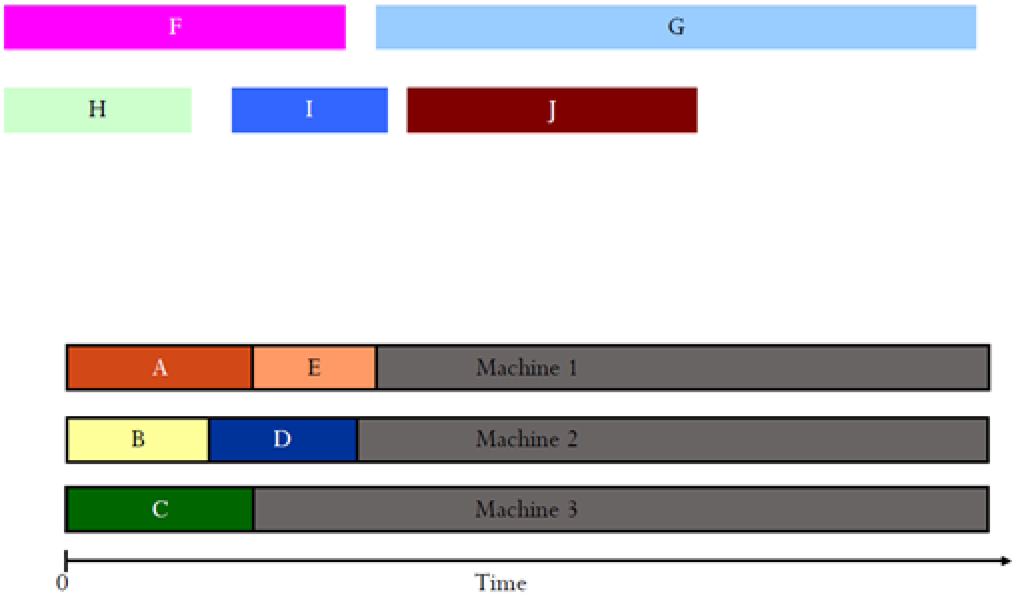
Step-by-step example



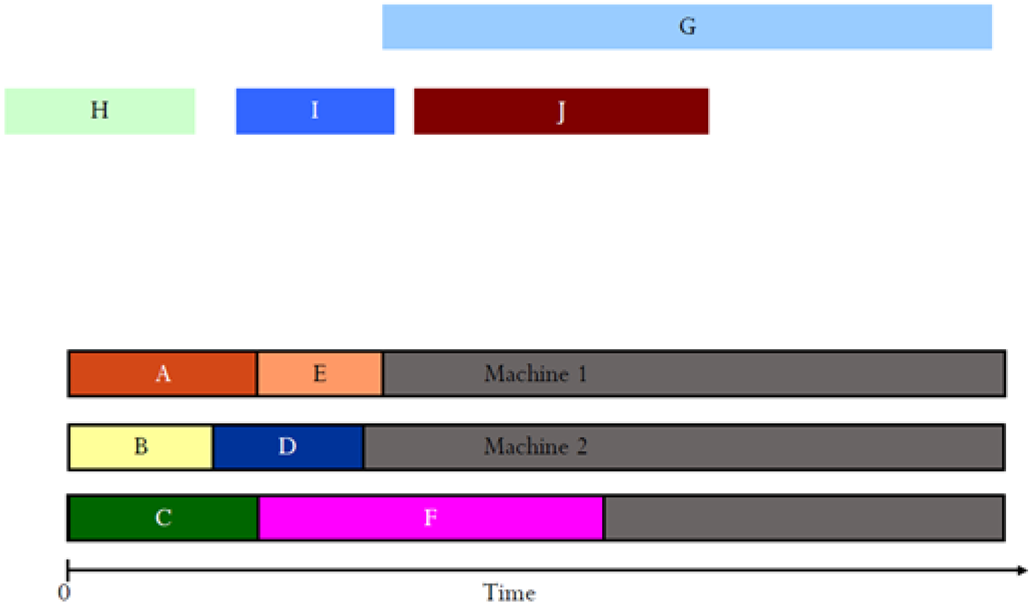
Step-by-step example



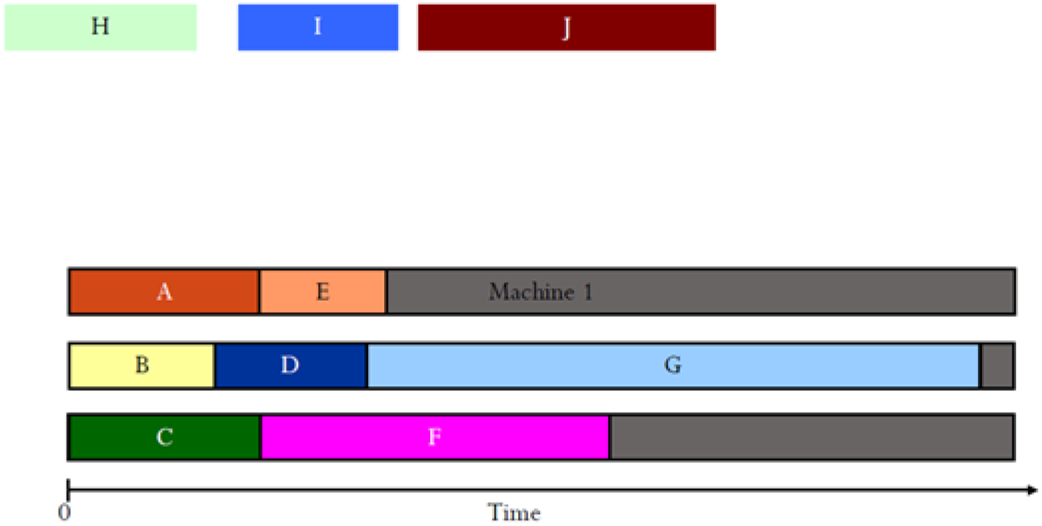
Step-by-step example



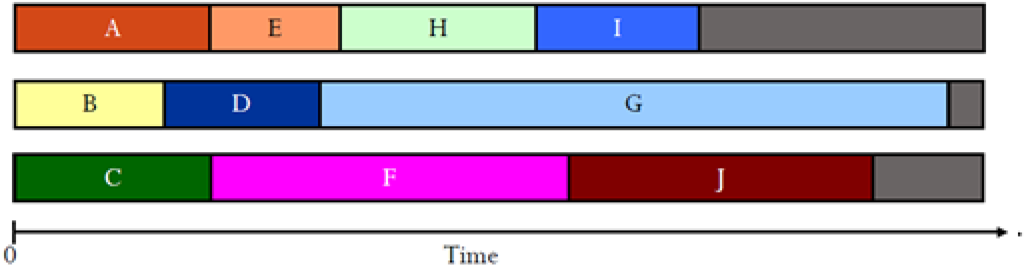
Step-by-step example



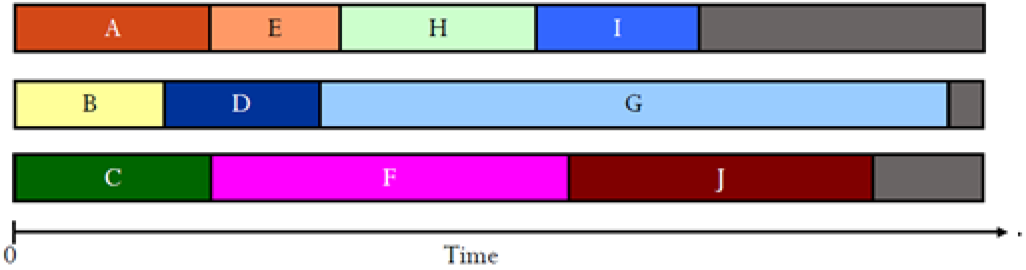
Step-by-step example



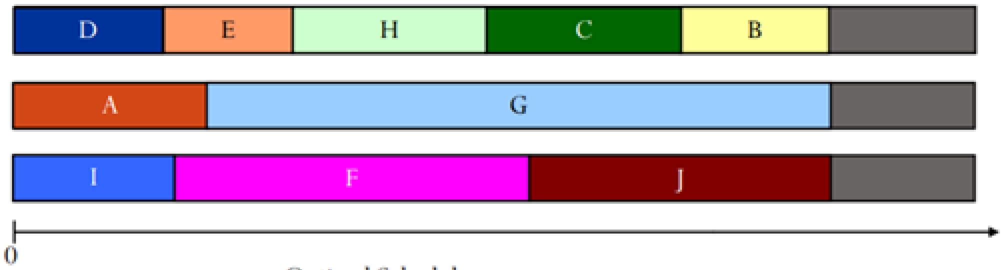
Step-by-step example (3 steps)



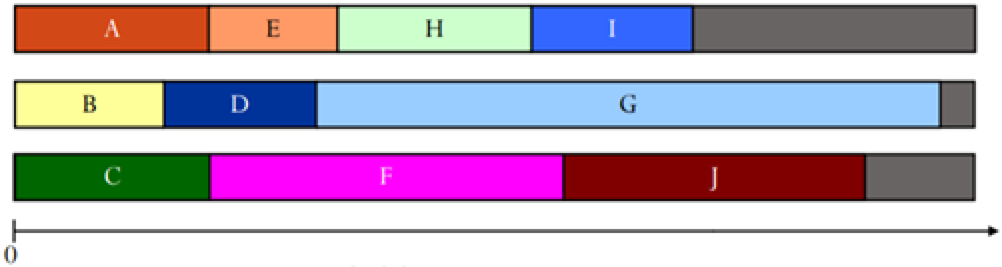
Step-by-step example (3 steps)



Este Optim?



Optimal Schedule



List schedule

NU

Care este Factorul de Aproximare?

Care este Factorul de Aproximare?

Lema 1.

$$OPT \geq max\left(\frac{1}{m}\sum_{1 \leq j \leq n} t_j,\ max\{t_j| \}1 \leq j \leq n\right)$$

Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.
Altfel spus, fie T - max(L_i| i ∈{1,...,m}) masina “cea mai incarcata”. Avem de arătat că T≤2xOPT

Care este Factorul de Aproximare?

Lema 1.

$$OPT \geq max\left(\frac{1}{m}\sum_{1 \leq j \leq n} t_j,\ max\{t_j \mid\}1 \leq j \leq n\right)$$

Lema 2.
Algoritmul descris anterior este un algoritm 2-Aproximativ.
Altfel spus, fie T - max(L_i| $i \in$ {1,...,m}) masina "cea mai incarcata". Avem de arătat că T≤2xOPT

Justificari pt Seria 23 & Seria 24

Care este Factorul de Aproximare?

Lema 2.
Algoritmul descris anterior este un algoritm 2-Aproximativ.
Altfel spus, fie T - max(L_i| $i \in \{1,...,m\}$) masina “cea mai incarcata”. Avem de arătat că $T \leq 2xOPT$

Justificari pt Seria 23 & Seria 24

Este ”tight bound”? Ce ar mai putea fi de facut?

Se poate imbunatății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
c) Un cu totul alt Algoritm care poate da un total alt LB.

Se poate imbunatății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit

b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități

c) Un cu totul alt Algoritm care poate da un total alt LB.

Se poate imbunatății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit

Teorema: Algoritmul Greedy descris anterior este un algoritm 2-1/m aproximativ

Justificari (continuare) Seria 23 & Seria 24

Se poate imbunatății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
c) Un cu totul alt Algoritm care poate da un total alt LB.

Se poate imbunatății LB-ul?

3 abordari:

Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități

Nu se poate!
m mașini, m(m-1) activitati de cost 1 si o activitate de cost *m*

Se poate imbunatății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
c) Un cu totul alt Algoritm care poate da un total alt LB.

Se poate imbunatății LB-ul?

Un cu totul alt Algoritm care poate da un total alt LB.

Ordered-Scheduling Algorithm

Fie algoritmul precedent la care adaugam următoarea preprocesare:

Înainte de a fi programate, activitățile sunt sortate descrescător după timpul de lucru.

Tema (preludiu)

Lema 3.

Fie o multime de n activitati cu timpul de procesare $t_1, t_2, \ldots, t_n$ astfel incat $t_1 \geq t_2 \geq \ldots t_n$

Daca $n > m$, *atunci* $OPT \geq t_m + t_{m+1}$

TEOREMA 2

Algoritmul descris anterior (Ordered-Scheduling Algorithm) este un algoritm 3/2-aproximativ

Next time:

Saptamana 3:

Veti primi prima parte din tema 1.

Curs 3: TSP & Christofides