Scheduling Parallel Jobs Under Power Constraints

Kunal Agrawal

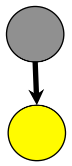
Washington University in St. Louis

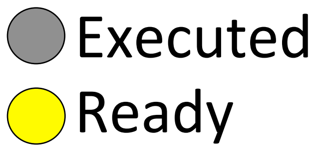
PARALLEL JOBS: DYNAMICALLY UNFOLDING DAG

- Node: Unit work task.
- Edge: Dependence between tasks.

PARALLEL JOBS: DYNAMICALLY UNFOLDING DAG

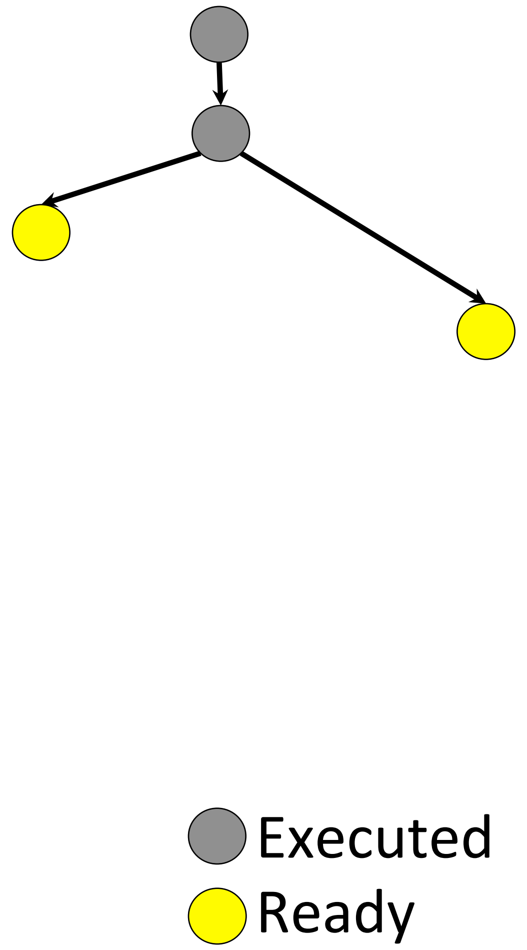
- Node: Unit work task.
- Edge: Dependence between tasks.





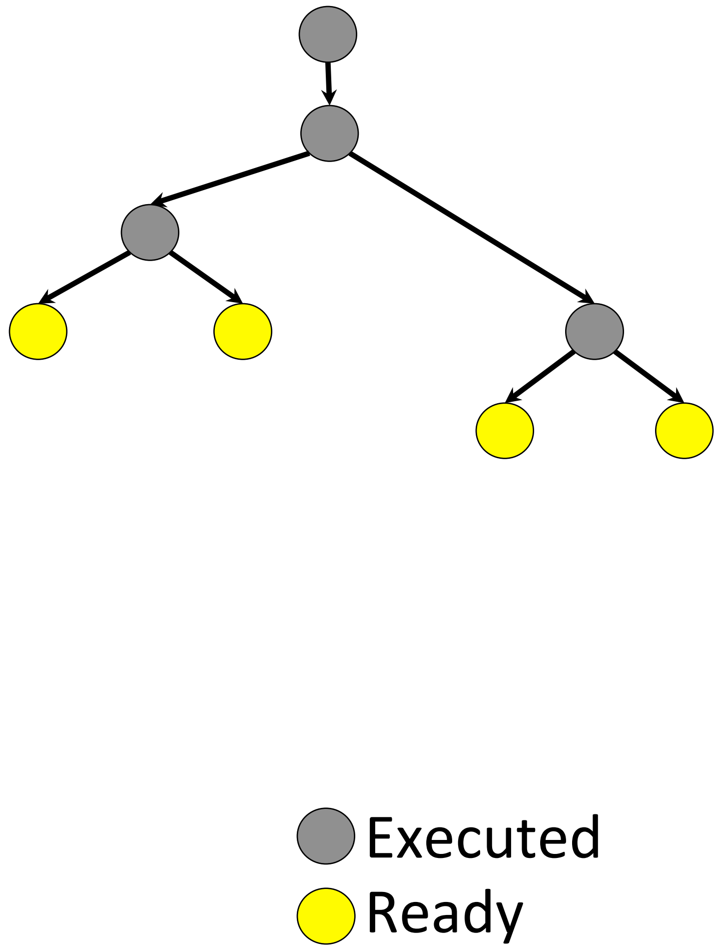
Parallel Jobs: Dynamically Unfolding DAG

- Node: Unit work task.
- Edge: Dependence between tasks.



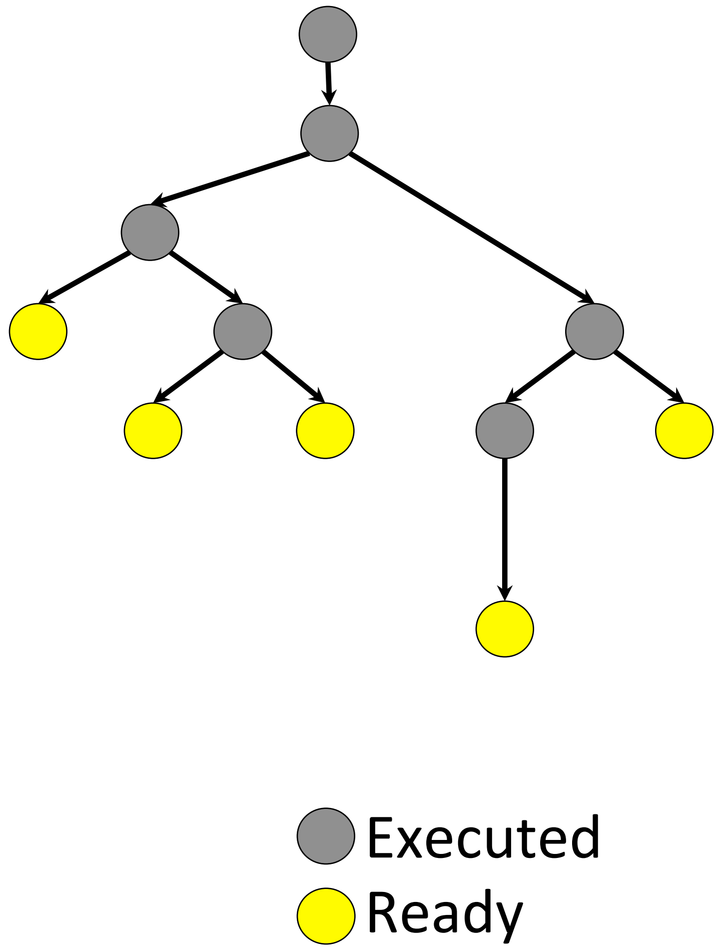
Parallel Jobs: Dynamically Unfolding DAG

- Node: Unit work task.
- Edge: Dependence between tasks.



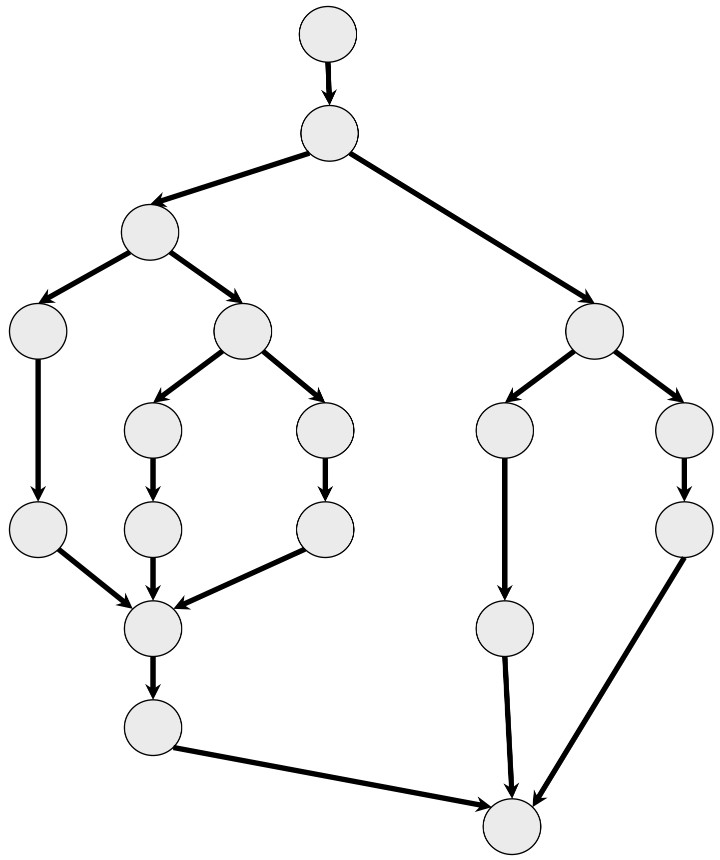
Parallel Jobs: Dynamically Unfolding DAG

- Node: Unit work task.
- Edge: Dependence between tasks.



Parallel Jobs: Dynamically Unfolding DAG

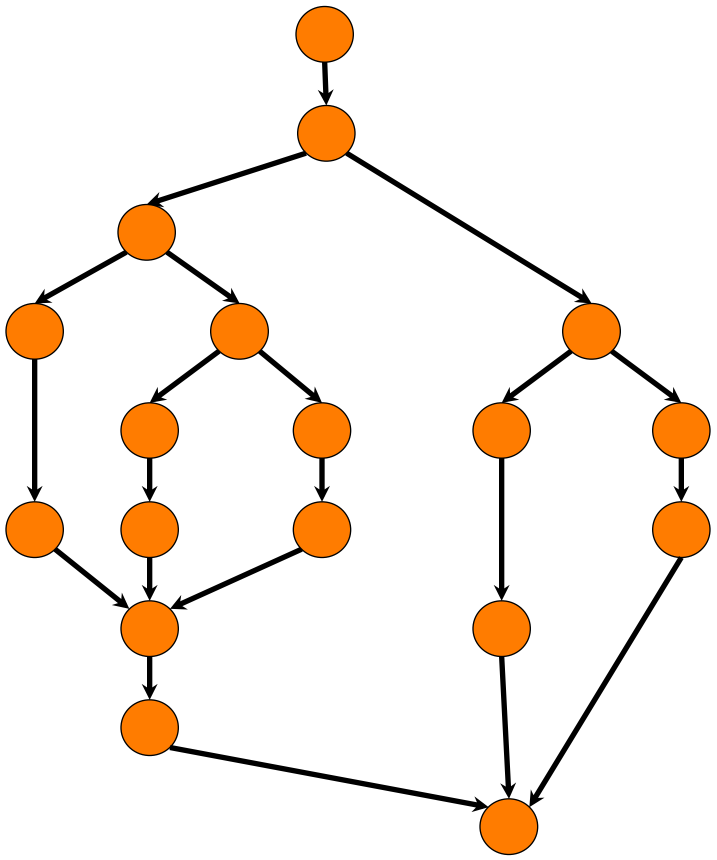
- Node: Unit work task.
- Edge: Dependence between tasks.



PARALLEL JOBS: DYNAMICALLY UNFOLDING DAG

- Node: Unit work task.
- Edge: Dependence between tasks.

- *W: Work* = Total number of nodes

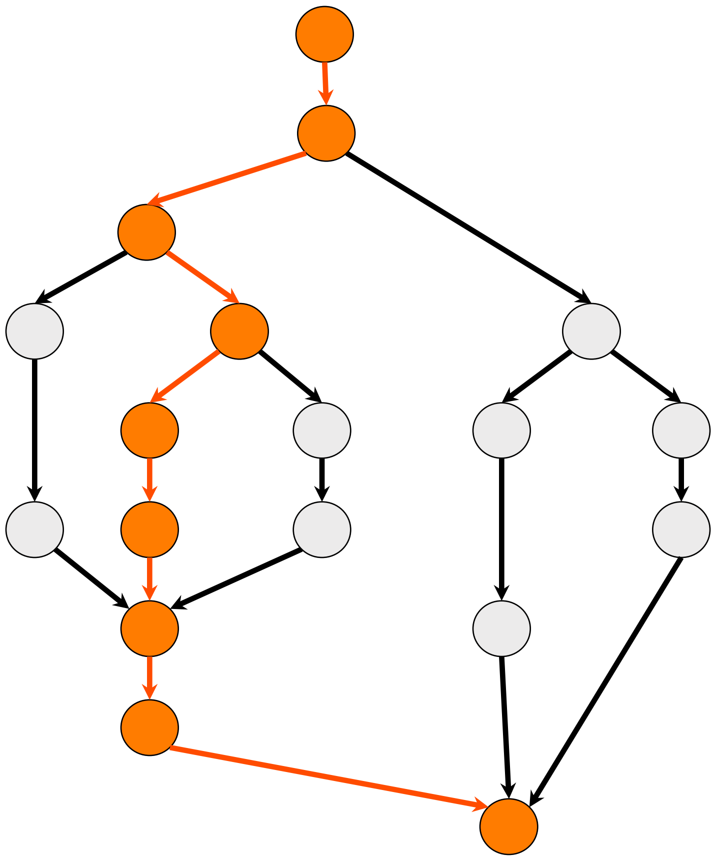


$W = 18$

PARALLEL JOBS: DYNAMICALLY UNFOLDING DAG

- Node: Unit work task.
- Edge: Dependence between tasks.

- *W: Work* = Total number of nodes
- *S: Span* = Length of the longest chain



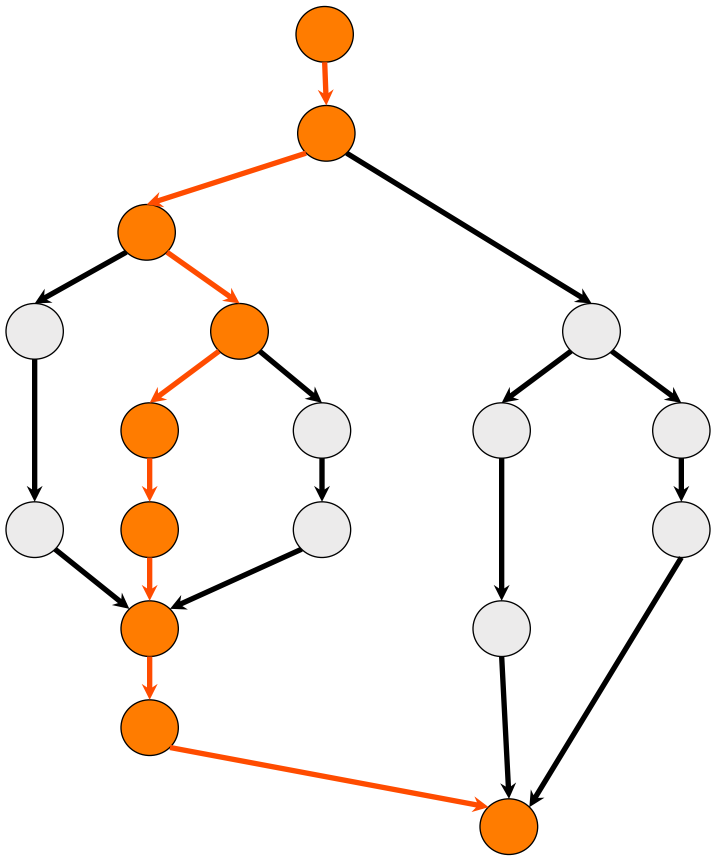
$W = 18$
$S = 9$

Parallel Jobs: Dynamically Unfolding DAG

- Node: Unit work task.
- Edge: Dependence between tasks.

- *W: Work* = Total number of nodes
- *S: Span* = Length of the longest chain

- **Assumption:** We do not know the work, span or the structure of the DAG in advance.



$W = 18$
$S = 9$

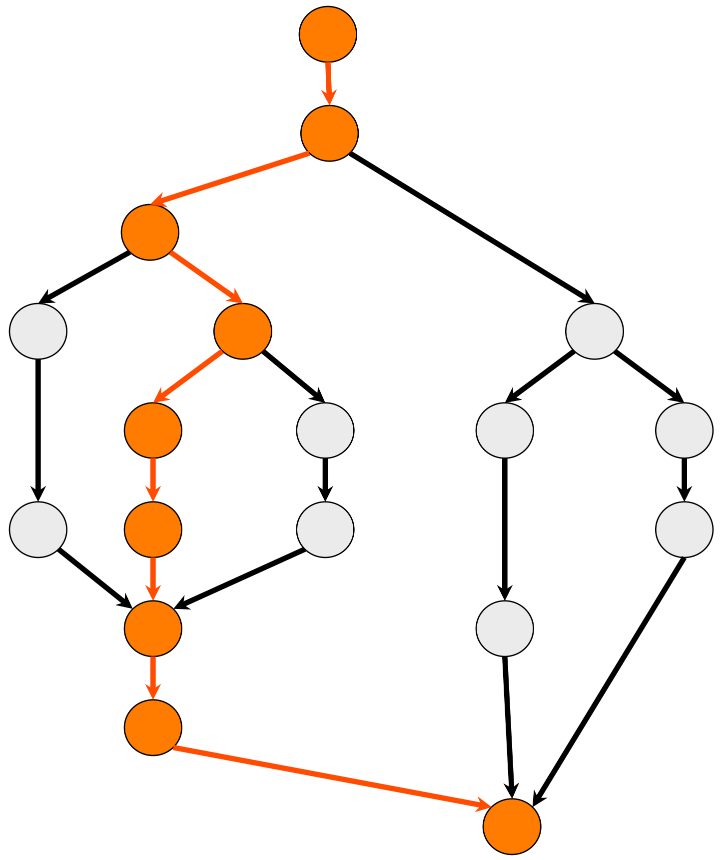
PARALLEL JOBS: DYNAMICALLY UNFOLDING DAG

- Node: Unit work task.
- Edge: Dependence between tasks.

- *W: Work* = Total number of nodes
- *S: Span* = Length of the longest chain

- **ASSUMPTION:** We do not know the work, span or the structure of the DAG in advance.

- With m processors of speed f, list scheduling guarantees a makespan of $W/fm + S/f$ (within 2 of optimal).



$W = 18$
$S = 9$

Power Constraint

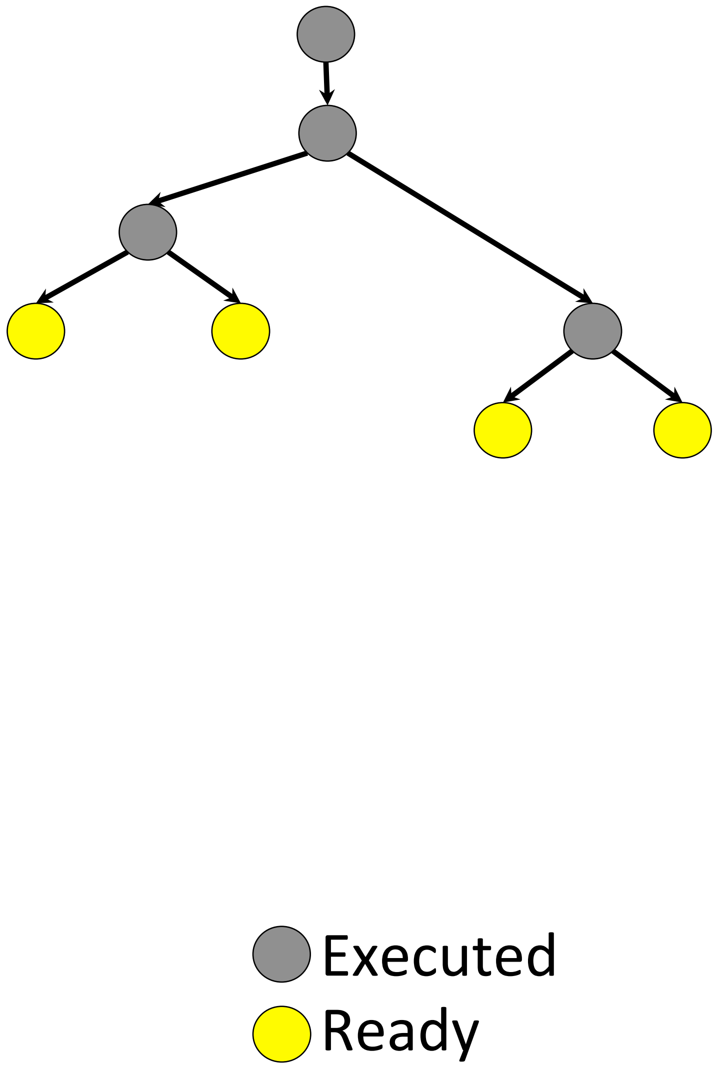
- At any time, we can only use power *P,* but can turn on or turn off processors.
- m processor running at speed f use power $P = mf^{\alpha}$ ($\alpha > 1$).
- With increasing m, the speed of individual processor (f) decreases, but you can do more work in each time step.
- m_{max}=maximum number of processors.
- We are allowed to change the configuration as the job executes, but fewer configuration changes is better.

m	*f*	*mf*
1	10	10
2	7.07	14.14
3	5.77	17.31
4	5	20
5	4.47	22.36

P=100, α=2, assuming all processors run at same speed.

PROBLEM DEFINITION

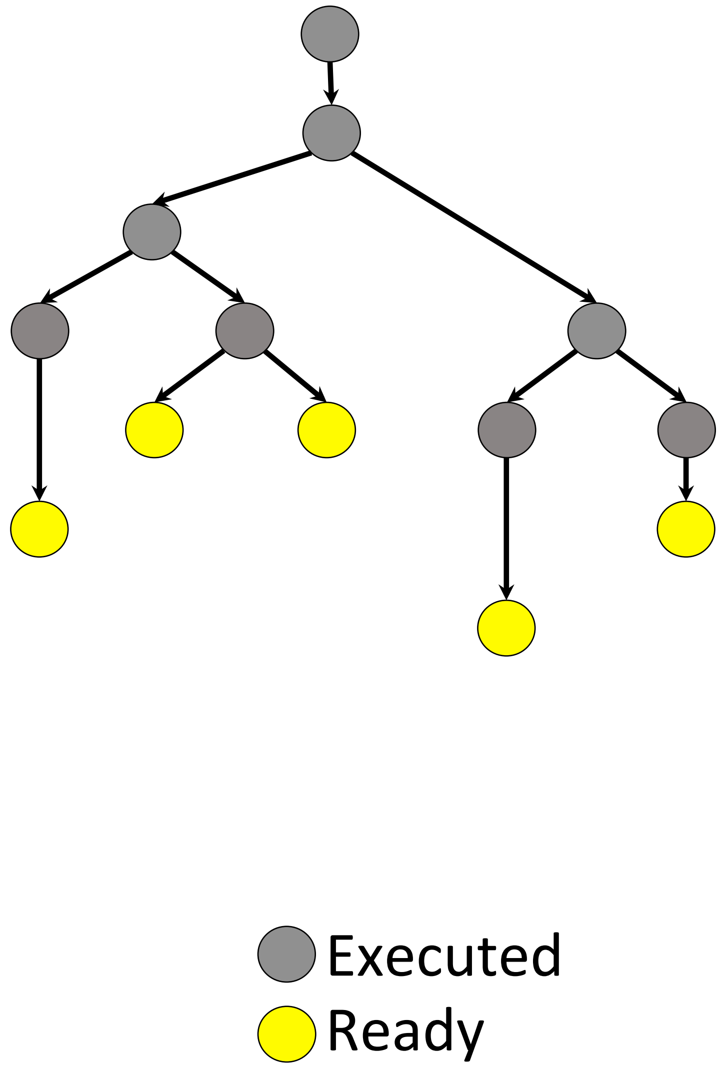
- **INTUITION**: We want to turn the maximum number of processors we can use.



PROBLEM DEFINITION

- **INTUITION**: We want to turn the maximum number of processors we can use.
- With *S* configuration changes, we get
  - optimal makespan if m_{max}> maximum width,
  - about 2-competitive otherwise.

- Question: What is the minimum number of configuration changes to get O(1)-competitive makespan?



The Home-Away-Pattern Set Feasibility Problem

Dirk Briskorn[1]

[1]Bergische Universität Wuppertal, Lehrstuhl für Produktion und Logistik

Single Round Robin Tournament

- $2n$ teams
- $2n-1$ rounds
- each team plays each other team exactly once
- each team plays exactly once per round

Single Round Robin Tournament

- $2n$ teams
- $2n-1$ rounds
- each team plays each other team exactly once
- each team plays exactly once per round

1	2	3	4	5
1-2	1-3	1-4	1-5	1-6
5-4	2-4	3-5	4-6	5-2
3-6	5-6	2-6	2-3	4-3

Scheduling SRRTs

- Scheduling SRRTs by First-break-then-schedule
  - First, the venue of each team in each round is fixed
  - Second, matches are arranged by pairing home-teams and away-teams
- Home-Away-Pattern Set Feasibility Problem: Given the venue for each team in each round, is there a corresponding SRRT?

Scheduling SRRTs

- Scheduling SRRTs by First-break-then-schedule
  - First, the venue of each team in each round is fixed
  - Second, matches are arranged by pairing home-teams and away-teams
- Home-Away-Pattern Set Feasibility Problem: Given the venue for each team in each round, is there a corresponding SRRT?

Scheduling SRRTs

- Scheduling SRRTs by First-break-then-schedule
  - First, the venue of each team in each round is fixed
  - Second, matches are arranged by pairing home-teams and away-teams
- Home-Away-Pattern Set Feasibility Problem: Given the venue for each team in each round, is there a corresponding SRRT?

	1	2	3	4	5
1	H	H	H	H	H
2	A	H	H	H	A
3	H	A	H	A	A
4	A	A	A	H	H
5	H	H	A	A	H
6	A	A	A	A	A

Home-Away-Pattern Set Feasibility

- Some obvious necessary conditions
  - Number of away-teams has to equal number of home-teams in each round.
  - There must not be two identical home-away patterns.
- Some less obvious ones
  - Miyashiro R., Iwasaki H., Matsui T. (2003) Characterizing Feasible Pattern Sets with a Minimum Number of Breaks. In: Burke E., De Causmaecker P. (eds) Practice and Theory of Automated Timetabling IV. PATAT 2002. Lecture Notes in Computer Science, vol 2740. Springer, Berlin, Heidelberg (for minimum number of breaks)
  - B. (2008): Feasibility of home-away-pattern sets for round robin tournaments, Operations Research Letters, Vol. 36, No. 3, pp 283-284.

Home-Away-Pattern Set Feasibility

	1	2	3	4	5
1	H	H	H	A	H
2	A	H	H	H	A
3	A	H	H	H	A
4	A	A	A	H	H
5	H	A	A	A	H
6	H	A	A	H	A

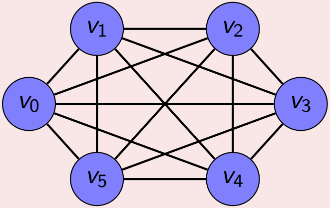
The Routing Open Shop Problem: Some Open Problems

Ilya Chernykh   Alexandr Kononov

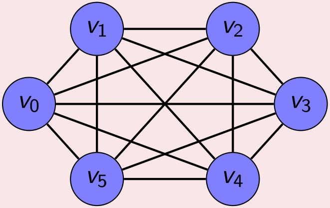
Sobolev Institute of Mathematics
Novosibirsk, Russia
{idchern,alvenko}@math.nsc.ru

Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP



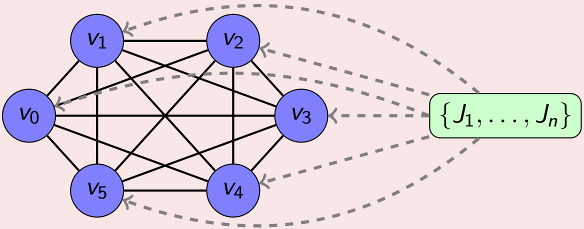
The combination of OPEN SHOP and Metric TSP



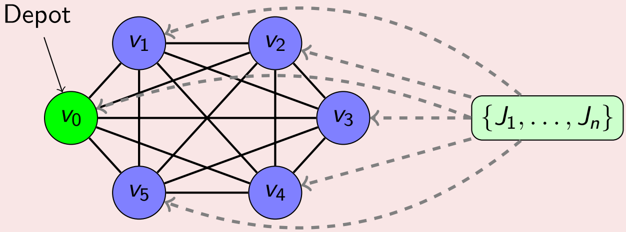
$\{J_1, \ldots, J_n\}$

Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP

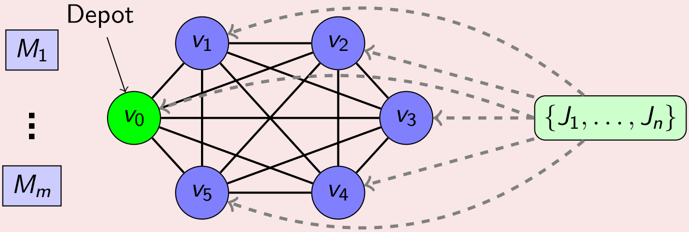


The combination of OPEN SHOP and Metric TSP



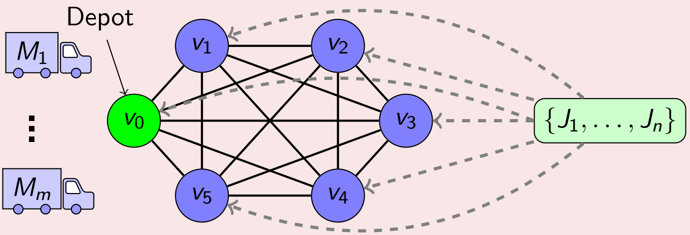
Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP



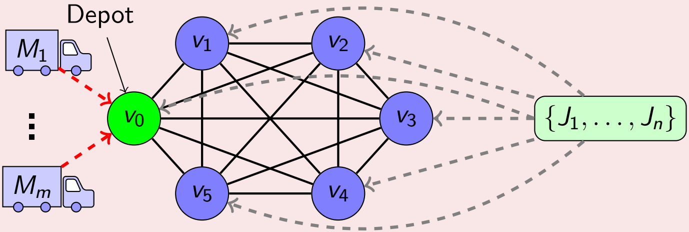
Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP



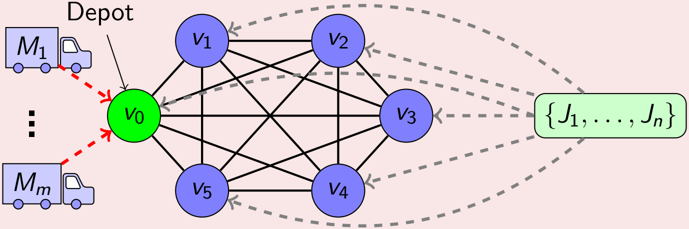
Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP



Informal introduction to the Routing Open Shop Problem

The combination of OPEN SHOP and Metric TSP



- p_{ji} — processing time of the operation of job J_j and machine M_i;
- $G = \langle V, E \rangle$ — transportation network;
- τ_{kl} — travel time between v_k and v_l;
- $R_i(S) = \max\limits_{k} \left(\max\limits_{J_j \in \mathcal{J}_k} C_{ji}(S) + \tau_{0k} \right)$;
- $R_{\max}(S) = \max R_i(S) \to \min_S$ — the makespan.

Lower bound

- $\ell_i = \sum_{j=1}^{n} p_{ji}$ — load of machine M_i,
- $d_j = \sum_{i=1}^{m} p_{ji}$ — length of job J_j,
- $\ell_{\max} = \max \ell_i$ — maximal machine load,
- $d^k_{\max} = \max_{J_j \in \mathcal{J}_k} d_j$ — maximal length of job from v_k,
- $\Delta^k = \sum_{J_j \in \mathcal{J}_k} d_j$ — total load of vertice v_k,
- T^* — length of the shortest route over G (TSP optimum)

Standard lower bound

$$\bar{R} = \max\left\{\ell_{\max} + T^*, \max_k \left(d^k_{\max} + 2\tau_{0k}\right)\right\}$$

Open Problems (not a complete list)

Routing open shop

Some Known Facts

1. NP-hard even for $\langle m = 2, G = K_2 \rangle$ (and a bunch of polynomially solvable classes for that case).
2. For general case best known approximation algorithm is $O(\log m)$-approximate.

Open Problems

1. Is there an *const*-approximation for general case?
2. Consider function $F(m) = \sup_{I \in \mathcal{I}_m} \frac{R^*_{\max}(I)}{\bar{R}(I)}$. Is $F(m)$ bounded by any constant?

Routing open shop with preemptions

Some Known Facts

1. Problem with $\langle m = 2, G = K_2 \rangle$ is polynomially solvable (and $R^*_{\max} = \bar{R}$).
2. Problem with $G = K_2$ is strongly NP-hard if m is a part of input.
3. Problem with $\langle m = 2, G = K_3 \rangle$ is polynomially solvable IF for some node $\Delta^k > \bar{R} - 2\tau_{0k}$.

Open Problems

1. Complexity of
$\langle m = 2, G = K_3 \rangle$,
$\langle m = 3, G = K_2 \rangle$,
$\langle m = 2, G = K_{const} \rangle$,
$\langle m = const, G = K_2 \rangle$ cases.

The 13th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2017)

Delayed-Clairvoyant Scheduling

Sorrachai Yingchareonthawornchai, Eric Torng
Michigan State University, MI, USA

15 June 2017

MICHIGAN STATE
UNIVERSITY

Delayed-Clairvoyant Scheduling

$$1|online-time-clv, pmtn, r_j|\sum F_j$$

Shortest Remaining Processing Time (SRPT) is optimal

Delayed-Clairvoyant Scheduling

$1|online-time-clv, pmtn, r_j|\sum F_j$

Shortest Remaining Processing Time (SRPT) is optimal

$1|online-time-nclv, pmtn, r_j|\sum F_j$

Any deterministic algorithm is $\Omega(n^{\frac{1}{3}})$ competitive

Delayed-Clairvoyant Scheduling

$$1|online - time - clv, pmtn, r_j|\sum F_j$$

Shortest Remaining Processing Time (SRPT) is optimal

$$1|online - time - nclv, pmtn, r_j|\sum F_j$$

Any deterministic algorithm is $\Omega(n^{\frac{1}{3}})$ competitive

Shortest Elapsed Time First (SETF) is $(1+\epsilon)$ speed $1+\dfrac{1}{\epsilon}$ competitive

Delayed-Clairvoyant Scheduling

$1|online - time - clv, pmtn, r_j|\sum F_j$

Shortest Remaining Processing Time (SRPT) is optimal

$1|online - time - nclv, pmtn, r_j|\sum F_j$

Any deterministic algorithm is $\Omega(n^{\frac{1}{3}})$ competitive

Shortest Elapsed Time First (SETF) is $(1+\epsilon)$ speed $1+\frac{1}{\epsilon}$ competitive

$1|online - time - delayed - clv, pmtn, r_j|\sum F_j$ Delay factor $\alpha \in [0, 1]$

Delayed-Clairvoyant Scheduling

$1|online - time - clv, pmtn, r_j|\sum F_j$

Shortest Remaining Processing Time (SRPT) is optimal

$1|online - time - nclv, pmtn, r_j|\sum F_j$

Any deterministic algorithm is $\Omega(n^{\frac{1}{3}})$ competitive

Shortest Elapsed Time First (SETF) is $(1+\epsilon)$ speed $1 + \frac{1}{\epsilon}$ competitive

$1|online - time - delayed - clv, pmtn, r_j|\sum F_j$ Delay factor $\alpha \in [0, 1]$

For $\alpha < 1$ SETF+SRPT is $\frac{1+\alpha}{1-\alpha}$ competitive

Open problems

- Formalize non-uniform delay factor reduction
- Get an analogous result for clairvoyance when $\alpha < 1$
  - Weighted Flow time:

HDF is $(1+\epsilon)$ speed $1 + \frac{1}{\epsilon}$ competitive

Is WSETF+HDF $(1+\epsilon)$ speed $1 + \frac{1}{\epsilon}$ competitive for $\alpha = \frac{1}{1+\epsilon}$?

A Chair’s Scheduling Problem

Samir Khuller

University of Maryland

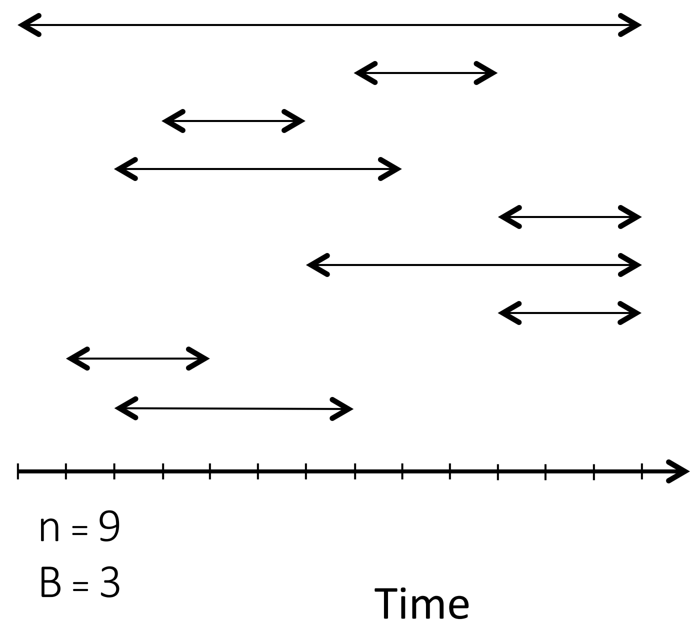
Research supported by NSF CCF 0937865, CCF 1217890 and Google.

Whats the problem?

- Alumni, Companies, Deans office, Provost's office, unhappy students, PhD students, faculty candidates, hiring meetings, faculty, staff....
- Lots of hour long meetings (some multiple)!
- Meetings from 9am to 6pm only
- GOAL: Maximize number of days at home!

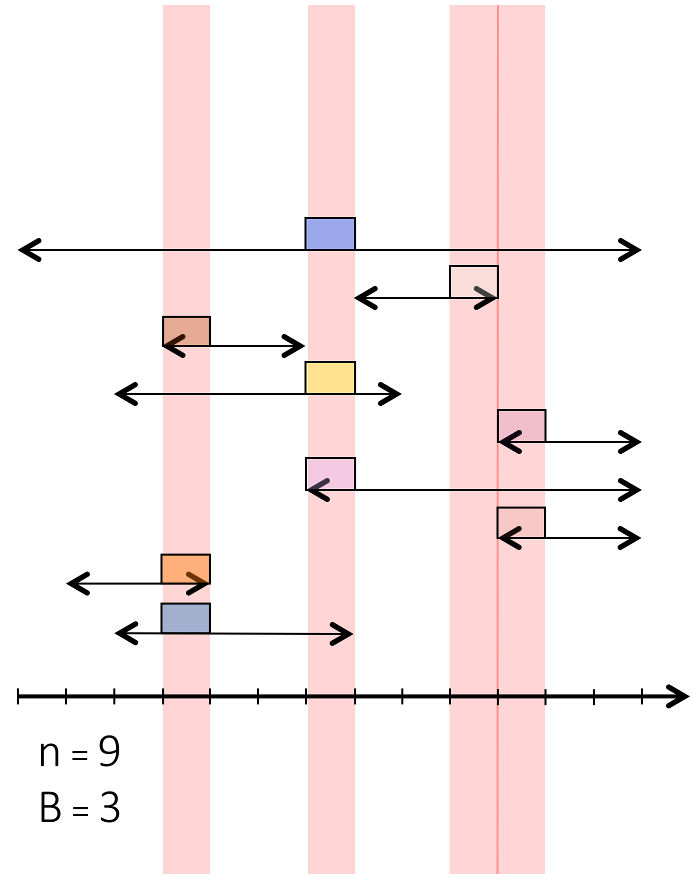
Scheduling to Minimize Active-Time

- *n* jobs
  - release times and deadlines
  - length
- batch machine
  - time is slotted
  - in each slot, “active” or “inactive”
  - “active” slot → can schedule ≤ B jobs
- minimize number of “active” slots



Scheduling to Minimize Active-Time

- *n* jobs
  - release times $r_i \in Z$ and deadlines $d_i \in Z$
  - length ℓ_i
- batch machine
  - time is slotted
  - in each slot, “active” or “inactive”
  - “active” at t → can schedule ≤ B jobs
- minimize number of “active” slots



FOUR ACTIVE SLOTS

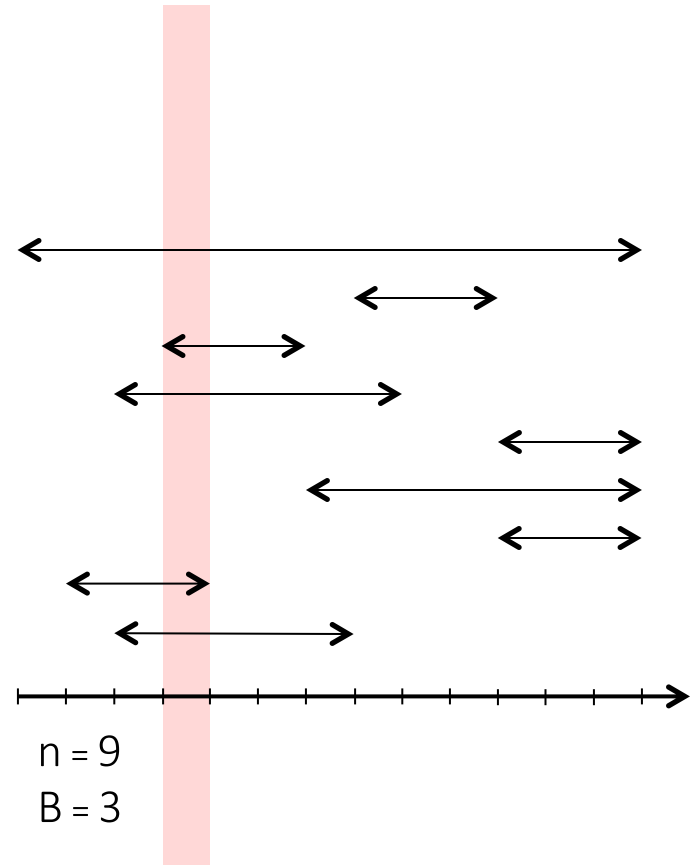
Lets focus on UNIT length jobs for now

Batching Algorithms

- Wolsey's greedy algorithm [Wolsey, 1982]
  - O(log n)-approximation
- Exact alg via Dynamic Programming
  [Even, Levi, Rawitz, Schieber, Shahar, Sviridenko, 2008]
  - Time complexity: $O(n^2 T^2 (n+T))$
- Faster exact algorithm?
- Also models taxi drop offs to the train station from Dagstuhl.

Lazy Activation

- *n* jobs
  - release times and deadlines
  - length
- batch machine
  - time is slotted
  - in each slot, “active” or “inactive”
  - “active” slot → can schedule ≤ B jobs
- minimize number of “active” slots

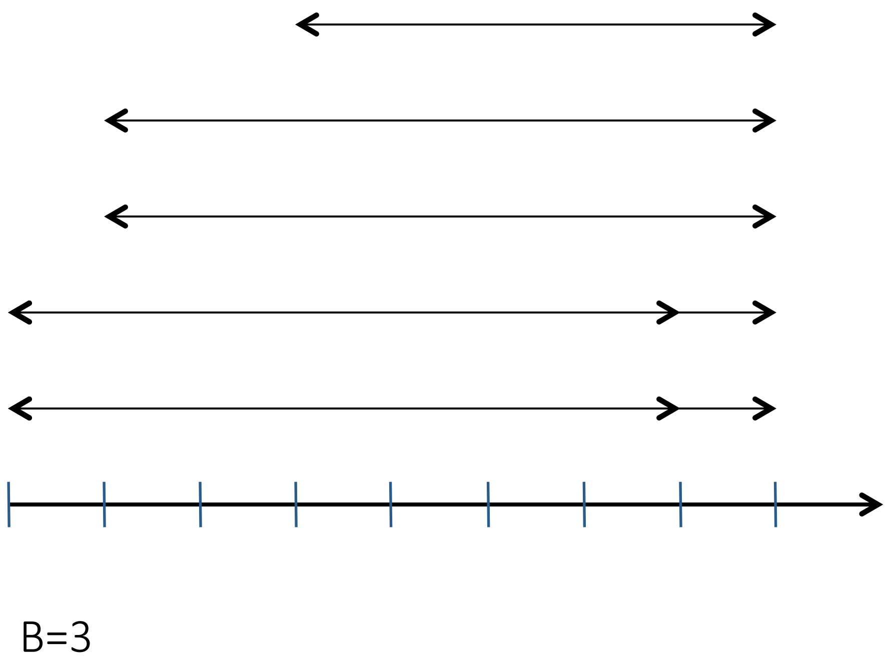


Each column is a set of capacity B

Lets focus on UNIT length jobs for now

Lazy Activation [Chang, Gabow, K., 2012]

- **Step I.** Scan slots right to left, and decrement deadlines in overloaded slots
  - favor decrementing deadlines of jobs with earlier release times

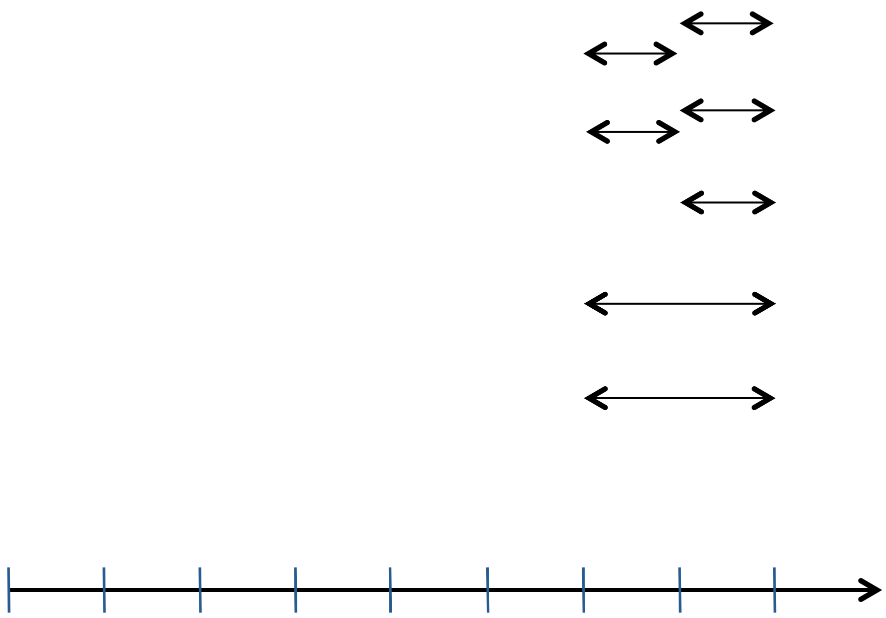


Lazy Activation [Chang, Gabow, K., 2012]

- **Step I.** Scan slots right to left, and decrement deadlines in overloaded slots
  - favor decrementing deadlines of jobs with earlier release times
- **Step II.**
  - Order jobs s.t.
  - Consider deadlines LTR:
    - Schedule at any outstanding jobs with deadline
    - Fill the remaining capacity with feasible jobs of later deadline, favoring those with earlier deadline

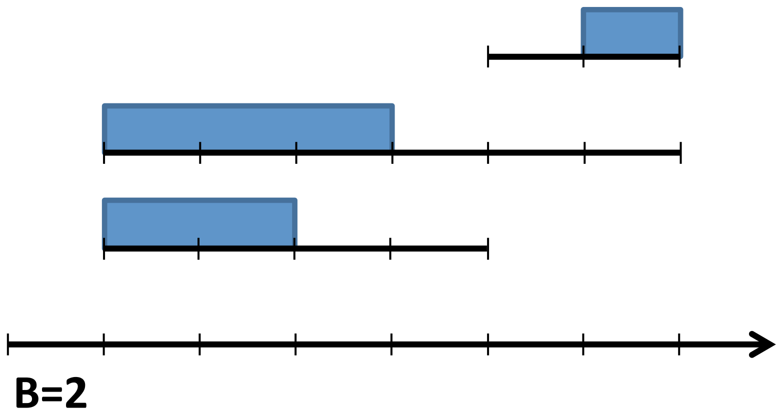
Lazy Activation Maximizes Throughput

- On infeasible instances, Step I preserves the maximum number of jobs

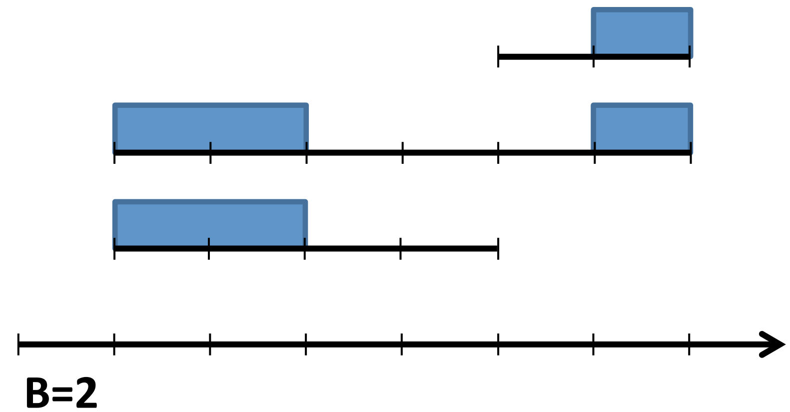


B = 3

Arbitrary-length Jobs [Chang,K,Mukherjee SPAA 2014]



active time: 4



active time: 3

NON-PREEMPTIVE:
NP-hard via 3-PARTITION

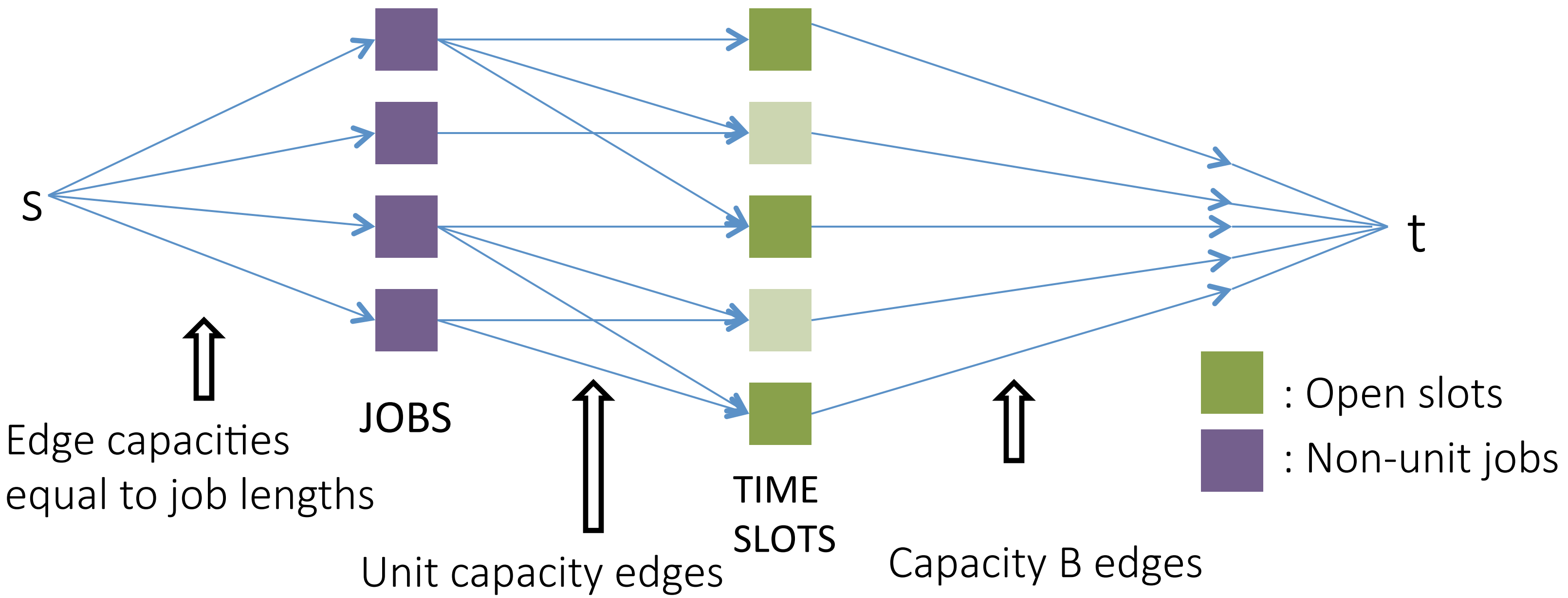
PREEMPTIVE complexity?

We have a 3 approximation (BusyTime)

We have a 2 approximation

Relation with max-flow

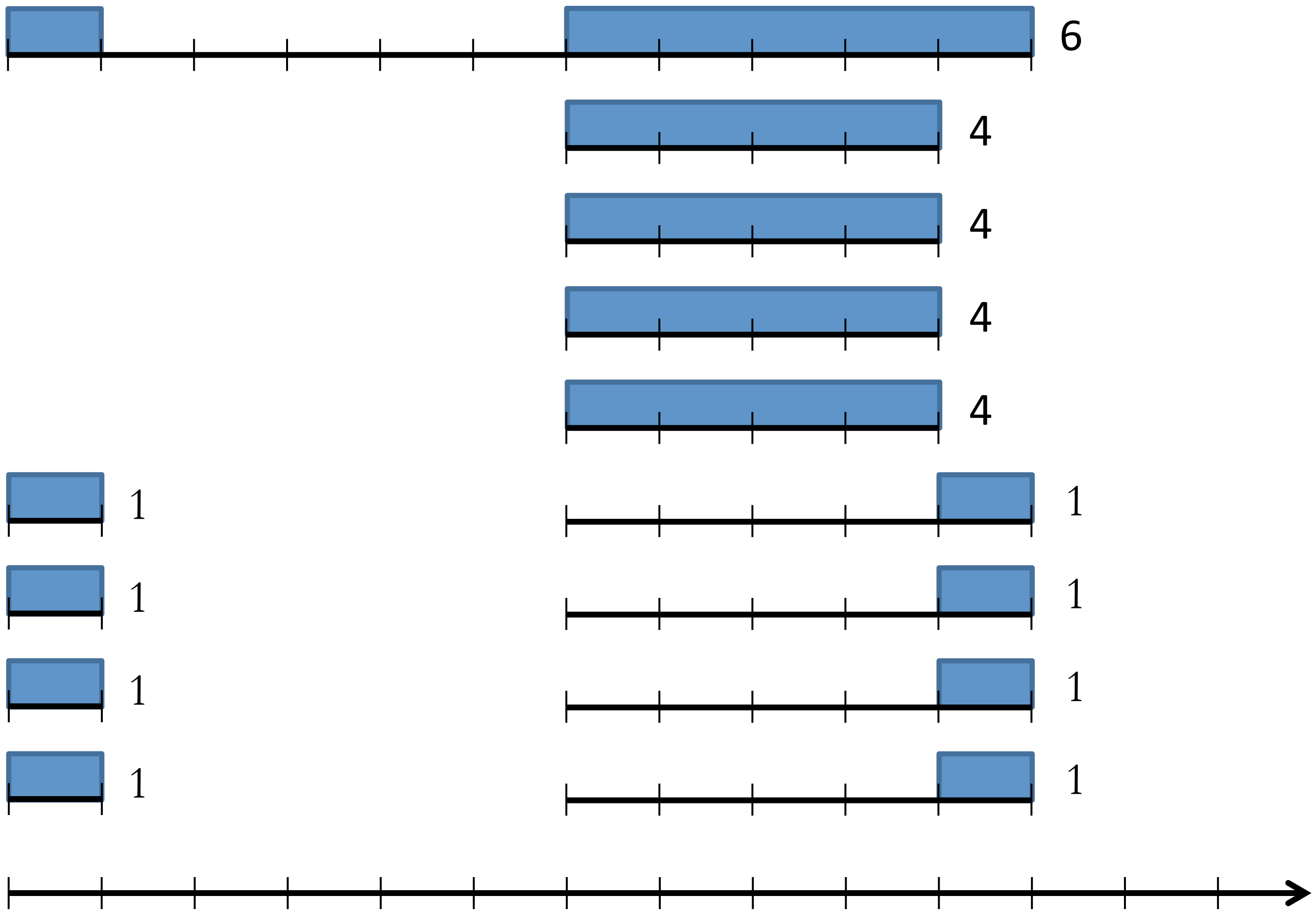
- Cost of a solution: number of open or active slots.
- Observation: Given a set of integrally open slots, max-flow will find a feasible integral assignment of jobs, if there exists one.
- This follows from max-flow integrality theorem.



Minimal Feasible Solutions

- Getting job assignments from a set of active slots: network flow computation
- Minimal feasible solutions (MFS): shutting down any active slot → infeasible
- Start from all slots being active, as long as a feasible schedule is possible, close a slot

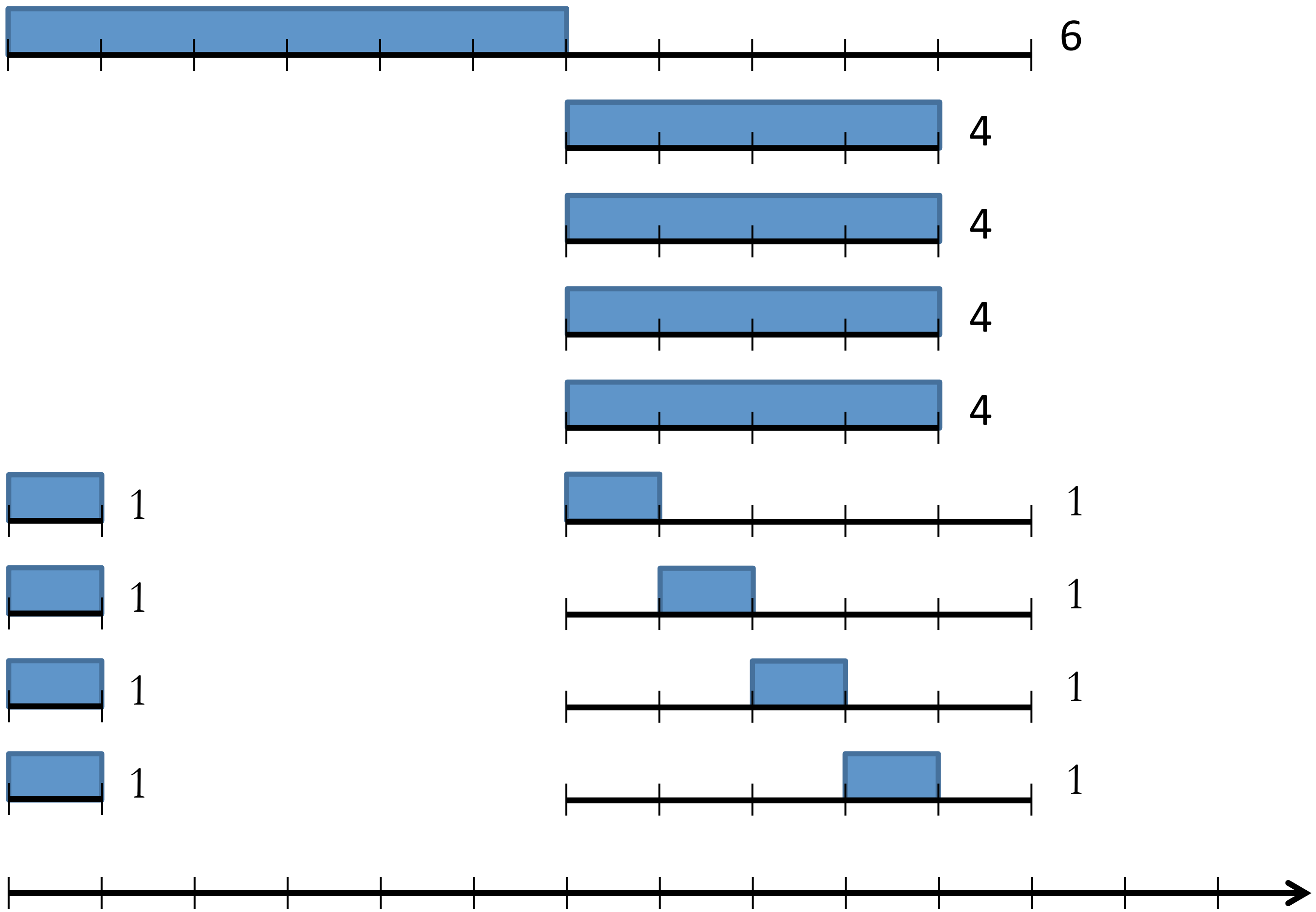
Minimal Feasible Solutions



B=5

OPTIMAL SOLUTION. COST=B+1

Minimal Feasible Solutions

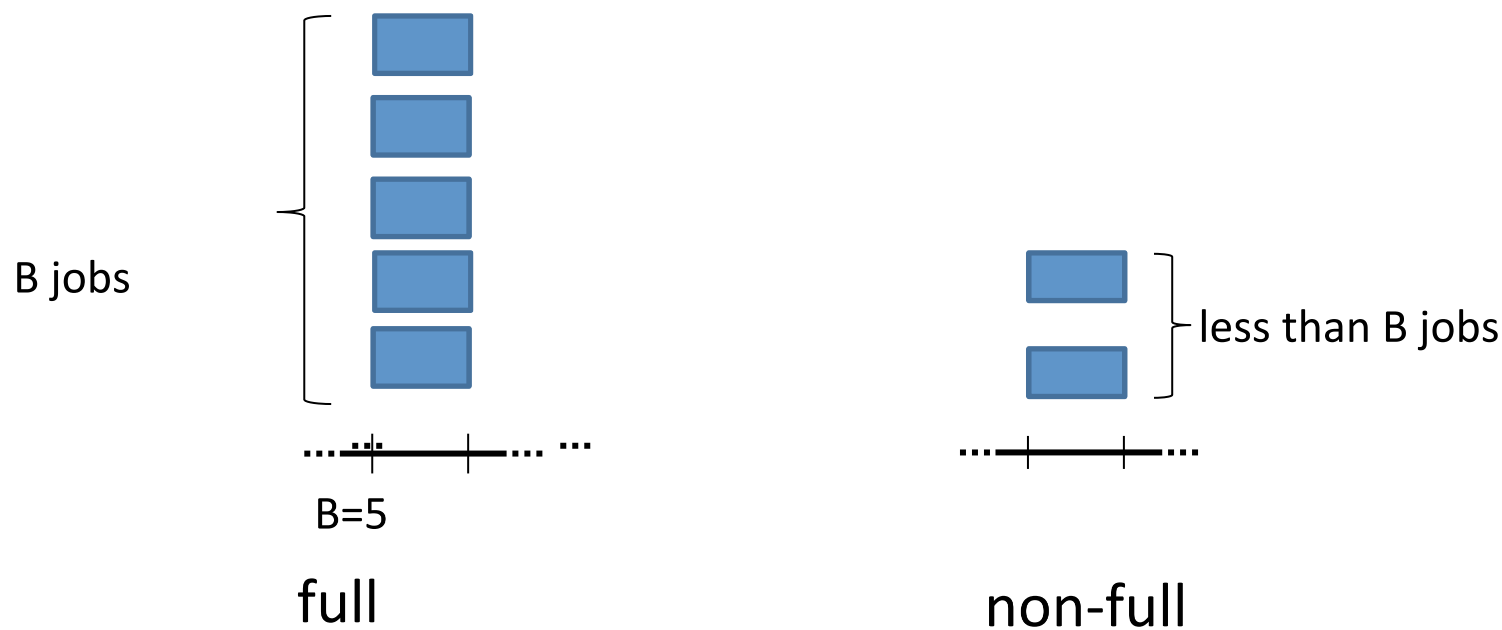


B=5

A MFS SOLUTION. COST=2B

Every MFS is 3-approximate

- Every MFS can be “left-shifted”
- Dichotomy of active slots



LP rounding based algorithm

$$\min \sum_{t \in T} y_t$$

$$\text{s.t. } x_{t,j} \le y_t \quad \forall\, j \in J,\ t \in T$$

$$\sum_{j \in J} x_{t,j} \le B\, y_t \quad \forall\, t \in T$$

$$\sum_{t \in [r_j, d_j]} x_{t,j} \ge p_j \quad \forall\, j \in J$$

$$0 \le y_t \le 1, \forall\, t \in T$$

$$x_{t,j} \ge 0, \forall\, t \in [r_j, \ldots, d_j]$$

What does the LP give?

- Factor 2 approximation (tight).
- [Kumar-Khuller] MFS that shuts slots left to right also gives factor 2 approximation.
- Local-Search is not optimal but might be <2.
- Still do not know if its NP-complete..

Parallel Machine Scheduling with Weighted Completion Time Objective and Online Machine Assignment

Sven Jäger

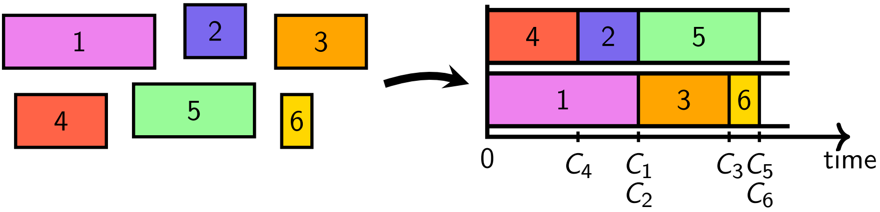
Combinatorial Optimization
and Graph Algorithms
Technische Universität Berlin

MAPSP Open Problem Session
13 June 2017

$P||\sum w_j C_j$

Given: Jobs with processing times $p_j \geq 0$ and weights $w_j \geq 0$, $j = 1, \ldots, n$ and number m of machines

Task: Process each job non-preemptively for p_j time units on one of the m machines such that the total weighted completion time $\sum_{j=1}^{n} w_j C_j$ is minimized.



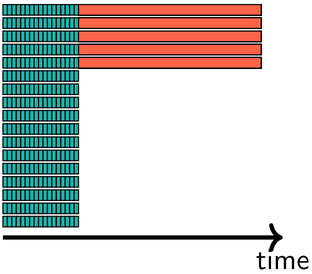
WSPT Rule

WSPT Rule

1. Sort jobs by non-increasing ratios w_j/p_j.
2. Do list scheduling in the obtained order.

Theorem [KK86] The WSPT rule has performance guarantee $\frac{1+\sqrt{2}}{2} \approx 1.207$.

Worst case instance: $w_j = p_j$ for all j.



Online Machine Assignment

- Jobs arrive sequentially and must immediately be assigned to the machines.
- After all jobs are assigned, the jobs on every machine can be sequenced optimally.

Online Machine Assignment

- Jobs arrive sequentially and must immediately be assigned to the machines.
- After all jobs are assigned, the jobs on every machine can be sequenced optimally.

MinIncrease

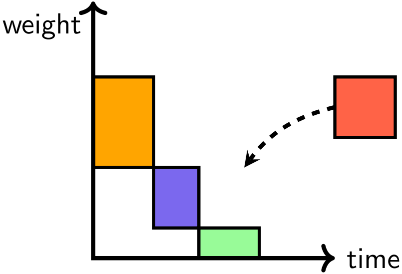
Assign each job to the machine that minimizes the increase of the current objective value.

Online Machine Assignment

- Jobs arrive sequentially and must immediately be assigned to the machines.
- After all jobs are assigned, the jobs on every machine can be sequenced optimally.

MinIncrease

Assign each job to the machine that minimizes the increase of the current objective value.

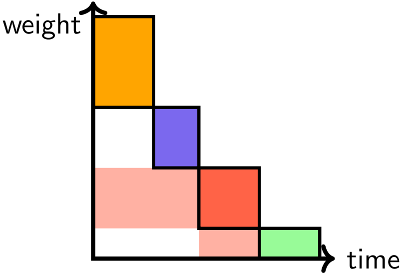


Online Machine Assignment

- Jobs arrive sequentially and must immediately be assigned to the machines.
- After all jobs are assigned, the jobs on every machine can be sequenced optimally.

MinIncrease

Assign each job to the machine that minimizes the increase of the current objective value.



Competitive ratio of MinIncrease

Observations

- If jobs arrive ordered by decreasing w_j/p_j, $\textsc{MinIncrease}$ does the same as list scheduling.

Competitive ratio of MinIncrease

Observations

- If jobs arrive ordered by decreasing w_j/p_j, $\textsc{MinIncrease}$ does the same as list scheduling.
- If jobs arrive ordered decreasingly or increasingly by w_j/p_j, $\textsc{MinIncrease}$ is $\frac{1+\sqrt{2}}{2}$-competitive.

Competitive ratio of MinIncrease

Observations

- If jobs arrive ordered by decreasing w_j/p_j, $\textsc{MinIncrease}$ does the same as list scheduling.
- If jobs arrive ordered decreasingly or increasingly by w_j/p_j, $\textsc{MinIncrease}$ is $\frac{1+\sqrt{2}}{2}$-competitive.
- In general, $\textsc{MinIncrease}$ is $\frac{3}{2} - \frac{1}{2m}$-competitve.

Competitive ratio of MinIncrease

Observations

- If jobs arrive ordered by decreasing w_j/p_j, $\textsc{MinIncrease}$ does the same as list scheduling.
- If jobs arrive ordered decreasingly or increasingly by w_j/p_j, $\textsc{MinIncrease}$ is $\frac{1+\sqrt{2}}{2}$-competitive.
- In general, $\textsc{MinIncrease}$ is $\frac{3}{2} - \frac{1}{2m}$-competitve.

Open Question

Is $\textsc{MinIncrease}$ always $\frac{1+\sqrt{2}}{2}$-competitive?

Appendix

Competitive Ratio for Stochastic Counterpart

Theorem (MUV06)

The algorithm that assigns each job to the machine with minimal increase of expected weighted completion time is $1 + \frac{(m-1)(\Delta+1)}{2m}$-competitive, where Δ is an upper bound on the coefficient of variation of the processing times.

References

- T. Kawaguchi and S. Kyan: *Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-time Problem*, SIAM J. Comput. 15(4):1119-1129, 1986
- U. Schwiegelshohn: *An Alternative Proof of the Kawaguchi-Kyan Bound for the Largest-Ratio-First Rule*, Oper. Res. Lett. 39:255-259, 2011
- N. Megow, M. Uetz, and T. Vredeveld: *Models and Algorithms for Stochastic Online Scheduling*, Math. Oper. Res. 31(3):513-525, 2006

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]
- $2^{O(p_{\max}^2 \log p_{\max})} \cdot (\log n + m)^{O(1)}$ [Knop, Koutecky; J. Sched. 2017]

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]
- $2^{O(p_{\max}^2 \log p_{\max})} \cdot (\log n + m)^{O(1)}$ [Knop, Koutecky; J. Sched. 2017]

number of distinct processing times $\overline{p}$: $\overline{p} \ll p_{\max}$

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]
- $2^{O(p_{\max}^2 \log p_{\max})} \cdot (\log n + m)^{O(1)}$ [Knop, Koutecky; J. Sched. 2017]

number of distinct processing times $\overline{p}$: $\overline{p} \ll p_{\max}$

- $m^{\overline{p}}(\overline{p})^{O(\overline{p})}(\log \Delta)^{O(\overline{p})}$ [Goemans, Rothvoss; SODA 2014]

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]
- $2^{O(p_{\max}^2 \log p_{\max})} \cdot (\log n + m)^{O(1)}$ [Knop, Koutecky; J. Sched. 2017]

number of distinct processing times $\overline{p}$: $\overline{p} \ll p_{\max}$

- $m^{\overline{p}} (\overline{p})^{O(\overline{p})} (\log \Delta)^{O(\overline{p})}$ [Goemans, Rothvoss; SODA 2014]
- $|V|^{2^{O(\overline{p})}} \cdot (n + \log m)^{O(1)}$ [Jansen, Klein; SODA 2017]

Makespan minimization on parallel machines

$P||C_{\max}$: Given a set J of n jobs with processing times $p_j \in \mathbb{N}, j \in J$, schedule all jobs in J on m parallel machines so as to minimize the makespan T.

- $2^{2^{O(T)}} \cdot (n + \log m)^{O(1)}$ [Alon, Azar, Woeginger; SODA 1997]

maximum processing time $p_{\max}$: $p_{\max} \ll T$

- $2^{2^{O(p_{\max}^2 \log p_{\max})}} \cdot (n + \log m)^{O(1)}$ [M., Wiese; IPCO 2014]
- $2^{O(p_{\max}^2 \log p_{\max})} \cdot (\log n + m)^{O(1)}$ [Knop, Koutecky; J. Sched. 2017]

number of distinct processing times $\overline{p}$: $\overline{p} \ll p_{\max}$

- $m^{\overline{p}}(\overline{p})^{O(\overline{p})}(\log \Delta)^{O(\overline{p})}$ [Goemans, Rothvoss; SODA 2014]
- $|V|^{2^{O(\overline{p})}} \cdot (n + \log m)^{O(1)}$ [Jansen, Klein; SODA 2017]

$P||C_{\max}$ in time $f(\overline{p}) \cdot (n + \log m)^{O(1)}$ for some function f

???