UNIVERSITÉ DE GENÈVE

POLITECNICO DI MILANO

A complex multi-objective production problem: from exact methods to advanced metaheuristics

Jean Respen[1] - Nicolas Zufferey[1] - Edoardo Amaldi[2]

[1]HEC - University of Geneva

[2]Politecnico di Milano

1st workshop on large scale optimization 2012

Outline

- Problem formulation
- Proposed methods
  - Exact model
  - Greedy heuristics
  - Descents
  - GRASP
  - Tabu search
  - Adaptive memory algorithm
- Results
- Future work / conclusion

Considered problem (1)

- **Multi-objective** function to minimize with
  - Setup costs and times
  - Makespan
  - Smoothing
- **Lexicographic** approach
  - Makespan > smoothing > setup costs

Considered problem (2)

- **Non identical parallel machines**
  - Working at different speeds
- Eligibility constraints
- Instances with ($100 \leq n \leq 500$)
- **Families** ($f \leq 10$) of jobs
- Each job is associated with a family

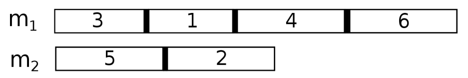
Setup costs and times

- Setups
  - **Costs c_j**, in the objective function
  - **Times s_j**, in makespan computation
  - Machine dependent
- 2 different types
  - Minor, if two jobs belong to the same family $s \in [5, 10]$
  - Major otherwise $s \in [30, 50]$

Smoothing issues

- Used to balance resource utilization and prevent overloading a production line
- **Based on family belonging**
- Ratio : 2/3 (every subsequence of 3 jobs can at most contain 2 jobs of the same family, otherwise: pay)

A basic example

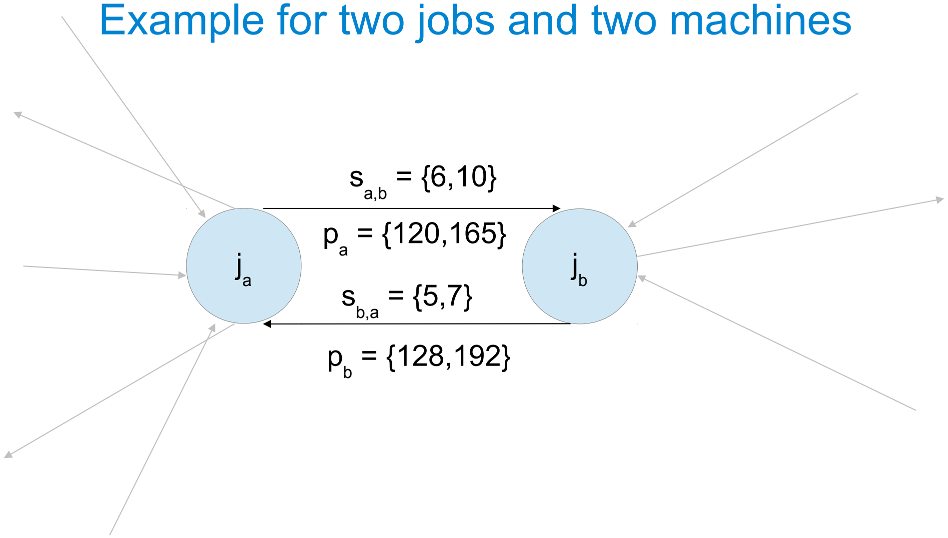


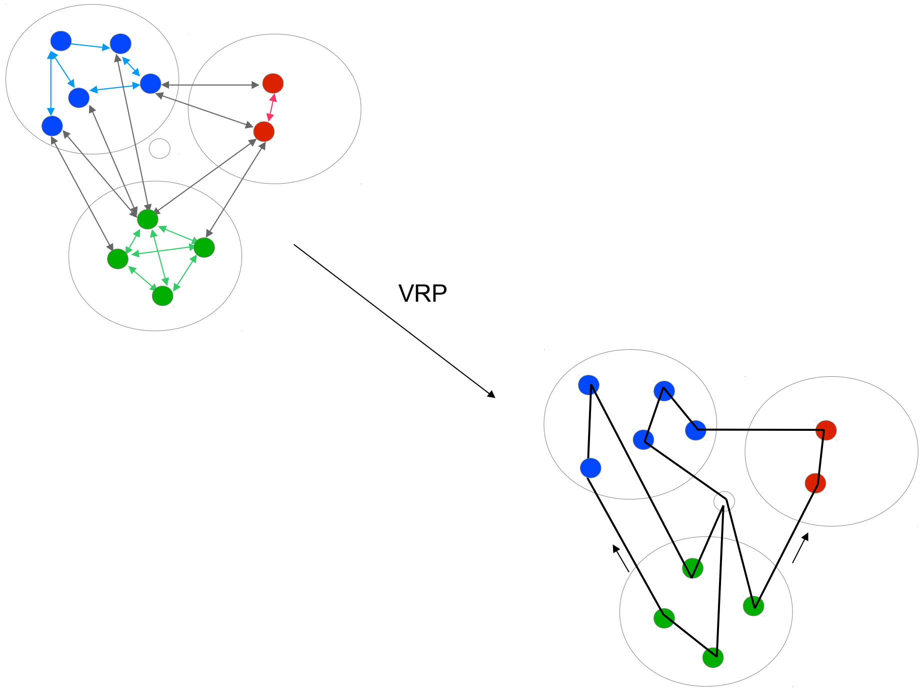
Applications

- Industry (assembly line)
- Car sequencing problems
- Vehicle routing problems

Graph representation

Example for two jobs and two machines





Proposed methods

- Exact method, tackled with AMPL and CPLEX
- Heuristics
  - **Three greedy algorithms**
  - **Descent**
  - **Descent with learning process**
  - **Tabu search**
  - **Adaptive memory algorithm**
  - **GRASP**

Exact method: objective function

- Parameters
  - α=1, β=10^3, δ=10^6
  - ω=1
- Decision variables
  - z_j^i = 1 if *j* on machine *i*
  - $x_{jj'}^i$ = 1 if *j,j'* consecutive of machine *i*
  - $y_{jj'j''}^i$= 1 if *j,j',j''* are consecutive on machine *i* and of the same family

$$\min \left(\alpha \cdot \sum_i \sum_{j,j'} c_{jj'}^i \cdot x_{jj'}^i + \beta \cdot \sum_i \sum_{j,j',j''} k_j \cdot y_{jj'j''}^i + \gamma \cdot \omega \cdot C_{max} \right)$$

Exact method

$$\min \left(\alpha \cdot \sum_i \sum_{j,j'} c^i_{jj'} \cdot x^i_{jj'} + \beta \cdot \sum_i \sum_{j,j',j''} k_j \cdot y^i_{jj'j''} + \gamma \cdot \omega \cdot C_{max} \right)$$

s.t.

$$\begin{array}{rc}
\forall i & C^i_{max} = \sum_{j \in N} p^i_j \cdot z^i_j + \epsilon \sum_{j,j' \in N} s^i_{jj'} \cdot x^i_{jj'} \\
\forall i & C^i_{max} \leq C_{max} \\
\forall i, j, j' & (2 - u^i_j - u^i_{j'}) \cdot x^i_{jj'} = 0 \\
\forall i, j & z^i_j \leq u^i_j \\
\forall i, j, j' & z^i_j + z^i_{j'} \geq 2 \cdot x^i_{jj'} \\
\forall i, j, j', j'' & 2 \cdot y^i_{jj'j''} \leq (x^i_{jj'} + x^i_{j'j''}) \cdot f_{jj'j''} \\
\forall i, j, j', j'' & (x^i_{jj'} + x^i_{j'j''}) \cdot f_{jj'j''} - 1 \leq y^i_{jj'j''} \\
\forall i & \sum_{j \in N} z^i_j - 1 = \sum_{j,j' \in N} x^i_{jj'} \\
\forall i, j & \sum_{j' \in N} x^i_{jj'} \leq 1 \\
\forall i, j' & \sum_{j \in N} x^i_{jj'} \leq 1 \\
\forall i, j & z^i_j \leq r^i_j \leq |N| \cdot z^i_j \\
\forall i, j, j' & r^i_{j'} \geq (r^i_j + 1) - |N| \cdot (1 - xjj'^i) \\
\forall i, j, j' & x^i_{jj'} + x^i_{j'j} \leq 1 \\
\forall j & \sum_{i \in M} z^i_j = 1 \\
\forall i, j, j', j'' & 0 \leq y^i_{jj'j''} \leq 1 \\
\forall i, j, j' & x^i_{jj'}, z^i_j \in \{0, 1\}
\end{array}$$

Three greedy algorithms

- Selection of the next job to schedule
  - **Random:** a random selection of jobs to insert
  - **Exhaustive:** test all the insertions, keep the best one
  - **Flexibility:** insert jobs in a least flexible order
  - Job flexibility: number of machines it can be performed on
- Jobs are always inserted at minimum cost

Descent

- Starts from an initial solution
- Performs moves to improve current solution
- Stops when improvement is not possible anymore
- Restarts

Learning descent

- Iteratively try each greedy heuristic
- Perform a descent on each initial solution
- After T/2, compute statistics and give weights
- Restarts
- Return best visited solution

GRASP

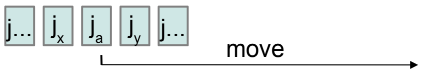
- Two steps
  - First build a solution in a constructive way
  - Each insertion, select the best x candidates and pick one at random
  - Second perform a descent

Tabu search

- Glover, 1986, formalized in 1989
- Start with an initial solution s
- While a stopping condition is not met, do
  - Move to a neighbor solution s' Є N(s)
  - Each time a move is performed (job swap, job exchange, job move, etc.) to reach a neighbor solution s', forbid the inverse move for Θ iterations.
  - s = s'
- Return s*

Tabu search: job moves

- Two tabu status



Forbidden to move j_a for Θ iterations

Forbidden to put j_a back between j_x and j_y for 2*Θ iterations

- Tabu tenure is a uniformly distributed random value between n/25 and n/13
- Neighborhood set to 50%

Diversification

- Every p_1 iteration, perform a diversification procedure
- Perform p_2 random moves and set them as tabu

Intensification

- Each time a solution beats the record, perform an intensification procedure
- Perform a descent on each machine

Adaptive memory algorithm

- Based on Rochat & Taillard 1995
- Population of 10 solutions
- Generate an offspring following different rules
- Improve offspring with 500 iterations of tabu search
- Replace the worst solution (or the oldest) in the population with the offspring

Instances

- Number of jobs **n Є** {100, 200, 300, 400, 500}
- Number of machines **m Є** {3,...,8}
- Number of families **f Є {**2,...,10}
  - For each family, a list of jobs
- Smoothing costs, **f** values

Tests configuration

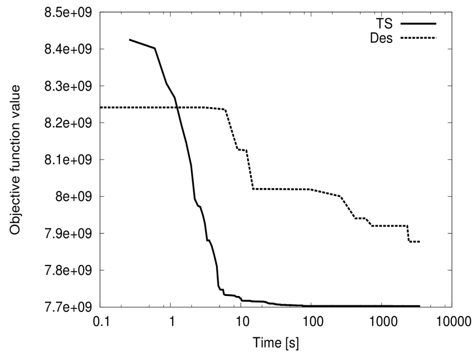
- Each instance
  - Greedy, GRASP and descent algorithms: 30 minutes with restart
  - Tabu search & AMA: 3 runs of 30 minutes with an exhaustive greedy initial solution
- Test lab
  - Quad-core Intel i7 2.93 ghz
  - 8 GB DDR3

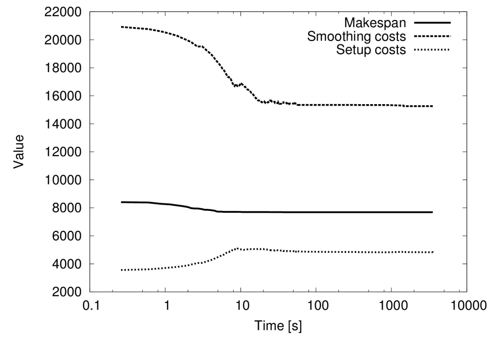
Results

Minimum values

n	m	f*	RG	EG	FG	GRASP	D	LD	TS	TS Di	TS In	Ama
100	3	4324195884	5.70	1.29	6.04	1.72	**0.11**	0.12	0.87	0.87	0.87	0.28
100	4	3371875845	6.59	0.60	21.01	0.95	0.60	**0.58**	4.19	0.72	1.42	0.80
200	4	6590233745	10.10	2.43	8.61	7.41	**0.05**	0.02	0.65	0.65	0.65	0.16
200	5	5353706666	10.23	11.49	8.60	4.73	1.71	2.00	0.60	0.60	0.60	**0.19**
300	5	7558778608	12.47	9.74	13.44	11.13	10.67	10.64	0.88	0.89	0.90	**0.04**
300	6	6016848499	14.29	0.10	12.77	9.65	0.27	0.15	0.24	0.25	0.25	**0.13**
400	6	8126390757	15.61	0.42	15.93	12.01	0.28	**0.15**	0.25	0.25	0.26	**0.15**
400	7	7383088654	13.83	6.66	13.12	13.56	6.44	6.51	1.72	1.72	1.78	**0.24**
500	7	9006737509	15.96	2.20	17.47	12.68	1.41	2.03	0.46	0.46	0.46	**0.07**
500	8	7649900636	16.31	9.64	17.31	12.67	3.38	3.06	1.64	1.64	1.64	**0.29**
AVG			**12.11**	**4.46**	**13.43**	**8.65**	**2.49**	**2.53**	**1.15**	**0.80**	**0.88**	**0.24**

Lexicographic order





Work in progress

- Improve diversification (dynamic version, etc.)
- Tune adaptive memory algorithm

Questions?