# Relative robust total completion time scheduling problem on a single machine


| Prabha Sharma<br>The NorthCap University<br>Gurugram, India | Diptesh Ghosh<br>Indian Institute of Management<br>Ahmedabad, India | Sandeep Singh<br>The NorthCap University<br>Gurugram, India |
|---|---|---|

# Minimizing completion times

We have a set of $n$ jobs $\{1, 2, \dots, n\}$.

The processing time of job $j$ is $p_j$.

Given a sequence $\pi = (\pi_{(1)}, \pi_{(2)}, \dots, \pi_{(n)})$ of jobs, the processing time for the sequence is

$$C(\pi) = \sum_{i=1}^{n} (n - i + 1) p_{\pi_{(i)}}.$$

Our objective is to find a sequence that minimizes $C(\pi)$.

Result:

The shortest processing time (SPT) rule generates an optimal sequence.

# Uncertainties

- Yang and Yu suggest that uncertainties can happen due to many reasons:
  - Machine breakdown
  - Non-availability of quality tools
  - Unstable workforce
  - Changes in the working environment
  - Many other complex external factors
- Probability distributions are used to model some of the uncertainties and expectations of the respective objective functions are optimized.
- Processing times estimated based on statistical data.
- Drawbacks of such estimations:
  - Variances can be large
  - The probability distributions assumed may be inaccurate

Yang, J., and G. Yu. "On the robust single machine scheduling problem." *Journal of Combinatorial Optimization* 6.1 (2002): 17-33.

# Alternate approaches

- All values in a finite interval $[a_i, b_i]$ for each job $i$ may be taken as the valid processing times for each job.
- Most of the work in the literature has used interval data.
- Kouvelis and Yu suggest a finite set of discrete values for each job. The processing time for the job will be chosen from among these values.

Kouvelis, P., and G. Yu. *Robust discrete optimization and its applications.* Vol. 14. Springer Science & Business Media, 2013.

# Problem formulation

- A scenario is obtained by assigning one of the possible processing times to each of the jobs.

E.g., s = {Job 1: 10, Job 2: 5, Job 3: 29, Job 4: 12, Job 5: 6}

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

# Problem formulation

- A scenario is obtained by assigning one of the possible processing times to each of the jobs.

E.g., s = {Job 1: 10, Job 2: 5, Job 3: 29, Job 4: 12, Job 5: 6}

- The set of all scenarios is denoted by $S = \{s\}$, where $s$ is a scenario.

E.g., $S = \{\{4, 5, 14, 12, 2\}, \{4, 5, 14, 12, 4\}, \ldots, \{17, 9, 29, 19, 6\}\}$

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

# Problem formulation

- A scenario is obtained by assigning one of the possible processing times to each of the jobs.

E.g., s = {Job 1: 10, Job 2: 5, Job 3: 29, Job 4: 12, Job 5: 6}

- The set of all scenarios is denoted by $S = \{s\}$, where $s$ is a scenario.

E.g., $S = \{\{4, 5, 14, 12, 2\}, \{4, 5, 14, 12, 4\}, \ldots, \{17, 9, 29, 19, 6\}\}$

- A job sequence is denoted by a permutation $\pi$.

E.g., $\pi = \{3,1,4,2,5\}$ meaning the job sequence is Job 3, then Job 1, then Job 4, then Job 2, and then Job 5.

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

# Problem formulation

- A scenario is obtained by assigning one of the possible processing times to each of the jobs.

E.g., s = {Job 1: 10, Job 2: 5, Job 3: 29, Job 4: 12, Job 5: 6}

- The set of all scenarios is denoted by $S = \{s\}$, where $s$ is a scenario.

E.g., $S = \{\{4, 5, 14, 12, 2\}, \{4, 5, 14, 12, 4\}, \dots, \{17, 9, 29, 19, 6\}\}$

- A job sequence is denoted by a permutation $\pi$.

E.g., $\pi = \{3,1,4,2,5\}$ meaning the job sequence is Job 3, then Job 1, then Job 4, then Job 2, and then Job 5.

- The completion time for a job sequence $\pi$ in scenario $s$ is denoted as $C(\pi, s)$.

E.g., if $\pi = \{3,1,4,2,5\}$ and $s = \{4, 7, 14, 15, 6\}$ then $C(\pi, s) = 5 \cdot 14 + 4 \cdot 4 + 3 \cdot 15 + 2 \cdot 7 + 1 \cdot 6 = 151$.

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

# Problem formulation

- A scenario is obtained by assigning one of the possible processing times to each of the jobs.
- The set of all scenarios is denoted by $S = \{s\}$, where $s$ is a scenario.
- A job sequence is denoted by a permutation $\pi$.
- The completion time for a job sequence $\pi$ in scenario $s$ is denoted as $C(\pi, s)$.

# Problem formulation

- The best (i.e., least) possible completion time for any job sequence in scenario $s$ is denoted as $C^*(s)$. The job sequence that has this completion time is obtained by the SPT rule.
- The deviation for a job sequence $\pi$ in scenario $s$ is $d(\pi, s) = C(\pi, s) - C^*(s)$.
- The maximum deviation for a job sequence $\pi$ is $\max_{s \in S}\{d(\pi, s)\}$.
- The scenario for which the $d(\pi, s)$ is a maximum is called a worst-case scenario for $\pi$.

# Problem formulation

The relative robust total completion time problem with discrete data is that of finding a job sequence $\pi$ for which the maximum deviation is minimum.

$$\min_{\pi} \max_{s \in S} d(\pi, s) = \min_{\pi} \max_{s \in S} (C(\pi, s) - C^*(s))$$

# About the problem

- Yang and Yu say that this robust objective is to hedge against the worst-case scenario.
- They say that discrete sets of processing times best capture the correlation among the processing times of different jobs.
- Kouvelis and Yu show that the problem is NP-complete even when $|S| = 2$.
- Yang and Yu have designed an exact dynamic programming algorithm and given two polynomial time heuristics.

Yang, J., and G. Yu. "On the robust single machine scheduling problem." *Journal of Combinatorial Optimization* 6.1 (2002): 17-33.

Kouvelis, P., and G. Yu. *Robust discrete optimization and its applications.* Vol. 14. Springer Science & Business Media, 2013.

# A lemma

Lemma

For any job sequence $\pi$, its worst-case scenario will either have the processing times of each of the jobs at their maximum value or at their minimum value.

Implication:

If each job $j$ has $k_j$ possible processing time values, this lemma reduces the effective size of $S$ from $\prod_{j=1}^{n} k_j$ to $2^n$.

# A local search algorithm

Algorithm localSearch() {

$\pi \leftarrow$ an initial job sequence generated based on problem data;

$\pi^{out} \leftarrow$ solution generated by a neighborhood search procedure;

output $\pi^{out}$;

}

# Generating the initial job sequence ($\pi$)

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

Min = {5, 1, 2, 4, 3} Max = {3, 4, 1, 2, 5}
Job sequence {#, #, #, #, #}

Min = {1, 2, 4, 3} Max = {3, 4, 1, 2}
Job sequence {5, #, #, #, #}

Min = {1, 2, 4} Max = {4, 1, 2}
Job sequence {5, #, #, #, 3}

Min = {2, 4} Max = {4, 2}
Job sequence {5, 1, #, #, 3}

Min = {2} MAX = {2}
Job sequence {5, 1, #, 4, 3}

Min = {} MAX = {}
Job sequence {5, 1, 2, 4, 3} ← Initial job sequence $\pi$

# Generating the initial job sequence ($\pi$)

```
function createInitialSequence{(* creates the initial job sequence*)

        Max ← list of jobs ordered in non-increasing order of their maximum
processing times;
        Min ← list of jobs ordered in non-decreasing order of their minimum
processing times;
        s ← (#, #, . . . , #, #); (* empty sequence of place-holders *)
        for (i from 1 to ⌊n/2⌋){
                j ← first element of Min;
                s[i] ← j;
                remove j from both Max and Min lists;
                j ← first element of Max;
                s[n + 1 − i] ← j;
                remove j from both Max and Min lists;
        }
        if (n is odd) assign the unassigned job to s[⌊n/2⌋ + 1];
        return s;
}
```

# Neighbourhood search

Two job sequences $\pi^1$ and $\pi^2$ are said to be neighbours (i.e., adjacent) if $\pi^2$ can be constructed from $\pi^1$ by interchanging the positions of exactly two jobs in $\pi^1$.

(1, 2, 3, 4, 5) and (1, 5, 3, 4, 2) are neighbours
but
(1 2 3 4 5) and (1, 3, 5, 2, 4) are not neighbours

# Neighbourhood search



function createNeighbors($\pi$){ *(* creates the neighbourhood of a given solution $\pi$*)*

N ← ∅;
$\pi_n \leftarrow \pi$;
for (i from 1 to n − 1){
for (j from i + 1 to n){
swap jobs in the i-th and j-th position in $\pi_n$;
add a copy of $\pi_n$ thus formed to N;
swap back jobs in the i-th and j-th position in $\pi_n$;
}
}
return N;
}

# Neighbourhood search



```
algorithm neighbourhoodSearch{

        π ← createInitialSequence;
        localOptFlag ← FALSE;
        while (localOptFlag = FALSE){
                localOptFlag ← TRUE;
                N ← createNeighbors(π);
                π_b ← solution in N with a minimum maximum deviation value;
                if (π_b has a lower worst case deviation than π){
                        π ← π_b;
                        localOptFlag ← FALSE;
                } (* if loop *)
        } (* while loop *)
        output π;
}
```

# Computing deviations (Example)

Suppose $\pi = (3, 1, 4, 2, 5\}$ and $s = \{17, 5, 29, 12, 2\}$.

SPT sequence $= \{5, 2, 4, 1, 3\}$

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

SPT completion time $= 5 \cdot 2 + 4 \cdot 5 + 3 \cdot 12 + 2 \cdot 17 + 1 \cdot 29 = 129$

Completion time for $\pi = 5 \cdot 29 + 4 \cdot 17 + 3 \cdot 12 + 2 \cdot 5 + 1 \cdot 2 = 261$

Deviation $C(\pi, s) = 261 - 129 = 132.$

# Computing maximum deviation



- Computing exact values of maximum deviation is expensive. So we compute an approximate value of maximum deviation of a solution $\pi$ through local search on a neighbourhood of scenarios.

- Based on our lemma, we need to look at only those scenarios in which the processing time of each of the jobs is either the minimum or the maximum in the set of its possible processing times.

# Computing maximum deviation

- Two scenarios $s_1$ and $s_2$ are said to be neighbours (i.e., adjacent) if they differ in the processing time of exactly one of the jobs. One of the scenarios will have the processing time of that job at the minimum level, while the other will have the processing time at the maximum level.

Scenarios {4, 5, 29, 19, 2} and {4, 9, 29, 19, 2} are neighbours but
Scenarios {4, 5, 29, 19, 2} and {17, 5, 14, 19, 6} are not.

Job 1 {4, 10, 17}
Job 2 {5, 7, 9}
Job 3 {14, 17, 29}
Job 4 {12, 15, 19}
Job 5 {2, 4, 6}

Assume a function deviationLocalSearch($\pi, s$) that computes an approximate value of the maximum deviation of sequence $\pi$ through local search, starting from scenario $s$.

# Computing maximum deviation

```
function computeMaximumDeviation(π){ (* returns an approximate value of the max. deviation of π *)

        Σ ← set of 9 randomly generated scenarios;
        J1 ← set of the first ⌊n/2⌋ jobs in π;
        J2 ← set of all jobs not in J1;
        σ ← scenario with the processing times of all jobs in J1 set to their
              maximum processing times and  all jobs in J2 set to their minimum
             processing times;
        Σ ← Σ ∪ σ; (* Σ now has 10 scenarios *)
        maxDeviation ← −1; (* this is guaranteed to be updated *)
        for (each σ ∈ Σ){
                deviation ← deviationLocalSearch(s, σ);
                if (deviation > maxDeviation)
                        maxDeviation ← deviation;
        }
        return maxDeviation;
}
```

# Computational experiments

- We used four sets of instances with 10 instances in each set for our experiments.
- The four sets have problems with $n = 5$, $10$, $15$, and $20$ respectively.
- Each job in each instance has a set of three possible processing times (generated randomly).
- Optimal job sequences could be computed using exhaustive enumeration for sets with $n = 5$ and $n = 10$ only.
- The maximum deviation values presented are obtained from computeMaximumDeviation.

# Computational experiments


We report

- Start: the maximum deviation value for the solution obtained by createInitialSequence.
- End: the maximum deviation value of the solution output by neighborhoodSearch.
- Impr. %: the percentage improvement of End over Start.
- Locally optimal sequence: The sequence obtained by neighborhoodSearch.

# Results from the set with $n = 5$

| Inst. | Start | End | Impr. % | Locally optimal sequence |
|---|---|---|---|---|
| 1 | 374 | 188 | 49.73% | {2,0,3,4,1} |
| 2 | 357 | 281 | 21.29% | {3,0,4,2,1} |
| 3 | 293 | 263 | 10.24% | {4,0,1,2,3} |
| 4 | 299 | 227 | 24.08% | {0,1,3,4,2} |
| 5 | 232 | 136 | 41.38% | {1,4,3,2,0} |
| 6 | 342 | 336 | 1.75% | {3,4,0,2,1} |
| 7 | 118 | 118 | 0.00% | {0,3,2,4,1} |
| 8 | 122 | 122 | 0.00% | {4,2,3,1,0} |
| 9 | 128 | 107 | 16.41% | {0,2,3,1,4} |
| 10 | 168 | 96 | 42.86% | {2,3,1,0,4} |

Optimal

*Optimal sequence is {0,4,1,3,2} with maximum deviation 224.*

Optimal

# Results from the set with $n = 10$

| Inst. | Start | End | Impr. % | Locally optimal sequence |
|---|---|---|---|---|
| 1 | 742 | 463 | 37.60% | {5,0,9,6,3,2,4,1,7,8} |
| 2 | 821 | 668 | 18.64% | {4,8,6,1,0,7,2,9,5,3} |
| 3 | 537 | 249 | 53.63% | {6,2,9,7,0,5,8,3,1,4} |
| 4 | 952 | 444 | 53.36% | {5,6,2,8,0,3,7,9,1,4} |
| 5 | 661 | 411 | 37.82% | {3,1,2,7,6,4,9,0,5,8} |
| 6 | 1098 | 920 | 16.21% | {7,9,0,2,8,5,6,3,4,1} |
| 7 | 773 | 620 | 19.79% | {4,6,7,0,9,5,8,1,2,3} |
| 8 | 988 | 726 | 26.52% | {2,9,0,6,8,3,5,4,1,7} |
| 9 | 1362 | 972 | 28.63% | {4,6,7,0,8,5,9,3,2,1} |
| 10 | 657 | 471 | 28.31% | {4,0,3,6,7,9,8,5,2,1} |

Optimal

*Optimal sequence is {6,2,5,9,7,0,8,3,1,4} with maximum deviation 237.*

Optimal

# Results from the set with $n = 15$

| Inst. | Start | End | Impr. % | Locally optimal sequence |
|---|---|---|---|---|
| 1 | 1916 | 1457 | 23.96% | {12,1,14,0,10,5,2,7,13,11,6,4,9,8,3} |
| 2 | 2456 | 1841 | 25.04% | {0,1,6,7,14,5,2,3,13,8,12,10,11,9,4} |
| 3 | 2021 | 1434 | 29.05% | {3,8,12,2,4,13,11,9,6,0,5,7,1,10,14} |
| 4 | 1855 | 1388 | 25.18% | {3,12,10,5,6,11,9,7,8,1,0,2,14,13,4} |
| 5 | 1571 | 1045 | 33.48% | {10,13,3,2,9,5,6,0,12,4,14,7,11,1,8} |
| 6 | 3042 | 1850 | 39.18% | {1,11,4,12,13,7,2,9,8,5,10,6,0,14,3} |
| 7 | 2577 | 1862 | 27.75% | {1,5,0,4,8,12,6,7,14,10,11,2,13,9,3} |
| 8 | 1602 | 956 | 40.32% | {8,1,9,12,0,13,3,11,10,5,2,6,4,14,7} |
| 9 | 1518 | 901 | 40.65% | {6,13,1,0,11,8,5,7,2,3,12,9,10,14,4} |
| 10 | 2081 | 1457 | 29.99% | {12,10,4,0,14,8,7,6,5,11,3,9,13,1,2} |

*Optimal sequences could not be computed for these instances.*

# Results from the set with $n = 20$

| Inst. | Start | End | Impr. % | Locally optimal sequence |
|---|---|---|---|---|
| 1 | 3237 | 2087 | 35.53% | {18,1,13,7,14,2,11,17,3,19,5,9,6,0,16,4,10,8,12,15} |
| 2 | 5236 | 3526 | 32.66% | {3,18,17,4,1,12,14,15,8,7,9,10,6,16,19,0,2,11,13,5} |
| 3 | 3501 | 2567 | 26.68% | {16,13,7,15,4,1,9,8,17,6,0,19,2,18,14,10,5,11,12,3} |
| 4 | 4596 | 3344 | 27.24% | {14,2,5,7,3,0,18,16,9,11,12,1,10,8,19,15,17,6,13,4} |
| 5 | 3902 | 2887 | 26.01% | {5,7,15,13,14,16,1,19,11,17,10,12,8,2,9,18,6,3,0,4} |
| 6 | 4425 | 3434 | 22.40% | {5,13,15,18,6,3,12,1,17,10,0,19,14,11,8,4,9,7,16,2} |
| 7 | 4411 | 2734 | 38.02% | {7,17,4,0,15,18,10,19,16,5,2,12,1,14,6,13,3,8,9,11} |
| 8 | 3183 | 2232 | 29.88% | {3,19,17,10,14,18,0,9,6,13,8,2,1,7,4,16,12,15,11,5} |
| 9 | 3498 | 2443 | 30.16% | {6,7,11,9,16,1,18,2,5,3,13,4,0,17,8,10,12,19,15,14} |
| 10 | 3885 | 2453 | 36.86% | {6,13,9,1,17,14,11,16,19,4,12,7,10,0,8,5,18,3,15,2} |

*Optimal sequences could not be computed for these instances.*

# Future work

- To completely characterize the worst-case scenario for a given job sequence. This will eliminate the need for searching the maximum deviation using deviationLocalSearch($\pi, s$).
- To obtain a good upper bound for the problem and to compare the performance of our algorithm with this upper bound.
- To extend the work when the scenario set is an arbitrary collection of the processing times of each job.
- To consider weighted version of the problem described here and a few of its variants.

# References


1. Yang, Jian, and Gang Yu. "On the robust single machine scheduling problem." *Journal of Combinatorial Optimization* 6.1 (2002): 17-33.
2. Kouvelis, Panos, and Gang Yu. *Robust discrete optimization and its applications*. Vol. 14. Springer Science & Business Media, 2013.
3. Daniels, Richard L., and Panagiotis Kouvelis. "Robust scheduling to hedge against processing time uncertainty in single-stage production." *Management Science* 41.2 (1995): 363-376.
4. Du, Charles, and Michael Pinedo. "A note on minimizing the expected makespan in flowshops subject to breakdowns." *Naval Research Logistics* (NRL) 42.8 (1995): 1251-1262.