Discrete optimization in production and logistics systems

Mariusz Uchroński

Wrocław Centre for Networking and Supercomputing
mariusz.uchronski@pwr.edu.pl

Outline

- Discrete optimization problems.
- Tools for solving discrete optimization problems.
- Example of using HPC in discrete optimization.
- Conclusions.

Discrete optimization problems

The development of new technologies, on the one hand, require solving classical problems of huge size, and on the other hand, they generate new complex problems which are characterized by:

- large number of decision variables – from 100 to 10000,
- usually discrete optimization problems belongs to the strong NP-hard class – the computation time of exact algorithms grows exponentially,
- huge size of the solution space – for a job shop scheduling problem with 10 tasks and 10 machines – $(10!)^{10} \approx 4 \cdot 10^{65}$
- large number of local extremes from 10^{50} to 10^{500},
- strong opposing relationship between the quality of solutions and calculation time.

Discrete optimization problems

Discrete optimization problems considered during EuroHPC PL project:

- single machine problem of minimizing the sum of tardinesses costs,
- binary knapsack problem,
- flow shop scheduling problem with C_{max} minimization,
- two machine flow shop scheduling problem with on time jobs maximization,
- vehicle routing problem.

Tools

Examples of tools for solving discrete optimization problems

- Mathematical optimization solvers:
  - Gurobi – state-of-the-art solver for mathematical programming,
  - CPLEX – high–performance optimization solver for linear, mixed–integer and quadratic programming,
  - AMPL – *A Modeling Language for Mathematical Programming*.
- OR-Tools – Google Optimization Tools – open-source, fast and portable software suite for solving combinatorial optimization problems,
- HeuristicLab – framework for heuristic and evolutionary algorithms (Windows only),
- dedicated (parallel and distributed) metaheuristic algorithms,
- and so on.

The single-machine problem of minimizing the sum of tardinesses costs $1||\sum w_i T_i$

- The set of tasks is given $\mathcal{J} = \{1, 2, \ldots, n\}$.
- For the $i \in \mathcal{J}$ task, let us define:
  - p_i – processing time,
  - d_i – due date, and
  - w_i – weight of the cost function for the task's tardiness.
- Each task must be performed on the machine, the following restrictions must be met:
  (a) the machine can perform at most one task at any given time,
  (b) task execution cannot be interrupted,
  (c) the task execution may begin at time zero.

The single-machine problem of minimizing the sum of tardinesses costs $1||\sum w_i T_i$

Any solution to the considered problem can be represented by the sequence $S_1, S_2, \ldots, S_n$ of tasks starting times, with the constraints:

$$S_i + p_i \leqslant S_j \vee S_j + p_j \leqslant S_i, \ i \neq j, \ i, j = 1, 2, \ldots, n, \qquad (1)$$

$$S_i \geqslant 0, \ i = 1, 2, \ldots, n \qquad (2)$$

Due to regularity of the goal function of the form of sum of tardinesses, solution $S_1, S_2, \ldots, S_n$ can be represented by the order of execution of tasks expressed by a permutation $\pi \in \Pi$ of elements of the set $\mathcal{J}$, where Π is the set of all such permutations.

The single-machine problem of minimizing the sum of tardinesses costs $1||\sum w_i T_i$

Goal to minimize:

$$\sum_i w_i T_i \tag{3}$$

Subject to constrains:

$$S_j - T_j + p_j - d_j \leqslant 0 \quad j = 1, \ldots, n, \tag{4}$$

$$-T_j \leqslant 0 \quad j = 1, \ldots, n, \tag{5}$$

$$S_j + p_j - M \cdot (1 - x_{j,k}) \leqslant S_k \quad j < k,\ j, k = 1, \ldots, n, \tag{6}$$

$$S_k + p_k - M \cdot x_{j,k} \leqslant S_j \quad j < k,\ j, k = 1, \ldots, n, \tag{7}$$

$$-S_j \leqslant -S_0 \quad j = 1, 2, \ldots, n. \tag{8}$$

Example – Gurobi using Python

- On the one hand, we have a discrete optimization problem to solve, and on the other hand, opportunities - optimization tools and techniques and HPC infrastructures.
- How to proceed with Gurobi using python?
  - Define your problem using mathematical programming – mathematical model.
  - Login to the HPC machine.
  - Implemnet mathematical model using python.
  - Submit batch job.
  - Get the optimization results.

Example – Gurobi, variables

```
def define_variables(self):
    # Define integer variable for start time of job i
    self.s = self.model.addVars(
        self.n, lb=0, vtype=GRB.INTEGER,
        ub=sum(self.p), name="s"
    )
    # Define integer variable for tardiness of job i
    self.t = self.model.addVars(
        self.n, lb=0, vtype=GRB.INTEGER,
        ub=sum(self.p) + max(self.d), name="t"
    )
    # Add binary variable which equals to 1 if job i precedes jo
    self.x = self.model.addVars(
        ((i, j) for i in range(self.n)
        for j in range(self.n)),
        vtype=GRB.BINARY,
        name="x",
    )
```

Example – Gurobi, constraints

```
def add_big_m_constraint(self):
    for i in range(self.n):
        for j in range(self.n):
            if i > j:
                self.model.addConstr(
                    self.s[i] >= self.s[j] + self.p[j]
                                - BIG_M * self.x[(i, j)]
                )
                self.model.addConstr(
                    self.s[j] >= self.s[i] + self.p[i]
                                - BIG_M * (1 - self.x[(i, j)])
                )
```

Example

```
def add_tardiness_constraint(self):
    self.model.addConstrs(
        (self.t[i] - self.s[i] >= self.p[i] - self.d[i]
         for i in range(self.n)),
        name="t_ctr",
    )

def add_makespan_constraint(self):
    self.model.addConstrs(
        (sum(self.p) - self.s[i] >= self.p[i]
         for i in range(self.n)),
        name="makespan_ctr",
    )
```

Example – Gurobi, objective function

```
def add_constraints(self):
    self.add_tardiness_constraint()
    self.add_big_m_constraint()
    self.add_makespan_constraint()

def define_objective_function(self):
    obj_fn = sum(self.w[i] * self.t[i]
                 for i in range(self.n))
    self.model.setObjective(obj_fn, GRB.MINIMIZE)
```

Example – SLURM script

```
#!/bin/bash
#SBATCH --reservation=eurohpc1
#SBATCH --job-name=WiTi_c32
#SBATCH --output=output_%j.txt
#SBATCH -p short

#SBATCH --time=2500:00
#SBATCH -c32
#SBATCH --mem=32G

source /usr/local/sbin/modules.sh

module load Python

source /home/dominik1/gurobi_weighted_tardiness/venv/bin/ac

python3 gurobi_weighted_tardiness_32.py
```

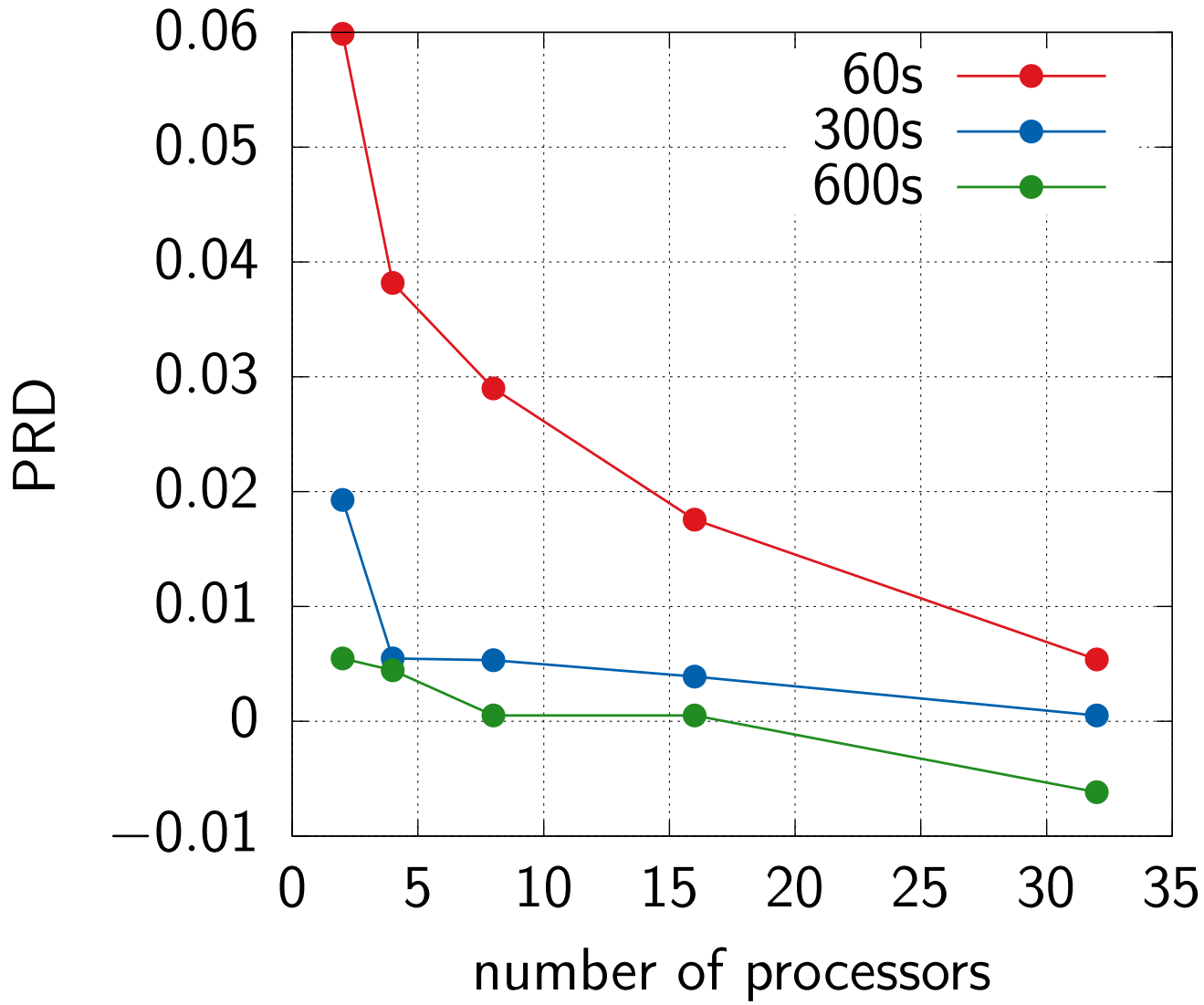
Example – Gurobi, output

```
Set parameter TimeLimit to value 600
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)

CPU model: Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz, instruction set [SSE2|AVX|AVX2|AVX512]
Thread count: 48 physical cores, 48 logical processors, using up to 32 threads

Optimize a model with 1680 rows, 1680 columns and 4840 nonzeros
Model fingerprint: 0xe5858f1e
Variable types: 0 continuous, 1680 integer (1600 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [1e+00, 1e+01]
  Bounds range     [1e+00, 4e+03]
  RHS range        [3e+00, 1e+04]
Presolve removed 80 rows and 820 columns
Presolve time: 0.01s
Presolved: 1600 rows, 860 columns, 4760 nonzeros
Variable types: 0 continuous, 860 integer (780 binary)
Found heuristic solution: objective 44199.000000

Root relaxation: objective 0.000000e+00, 741 iterations, 0.00 seconds (0.00 work units)
```

Example – Gurobi, output

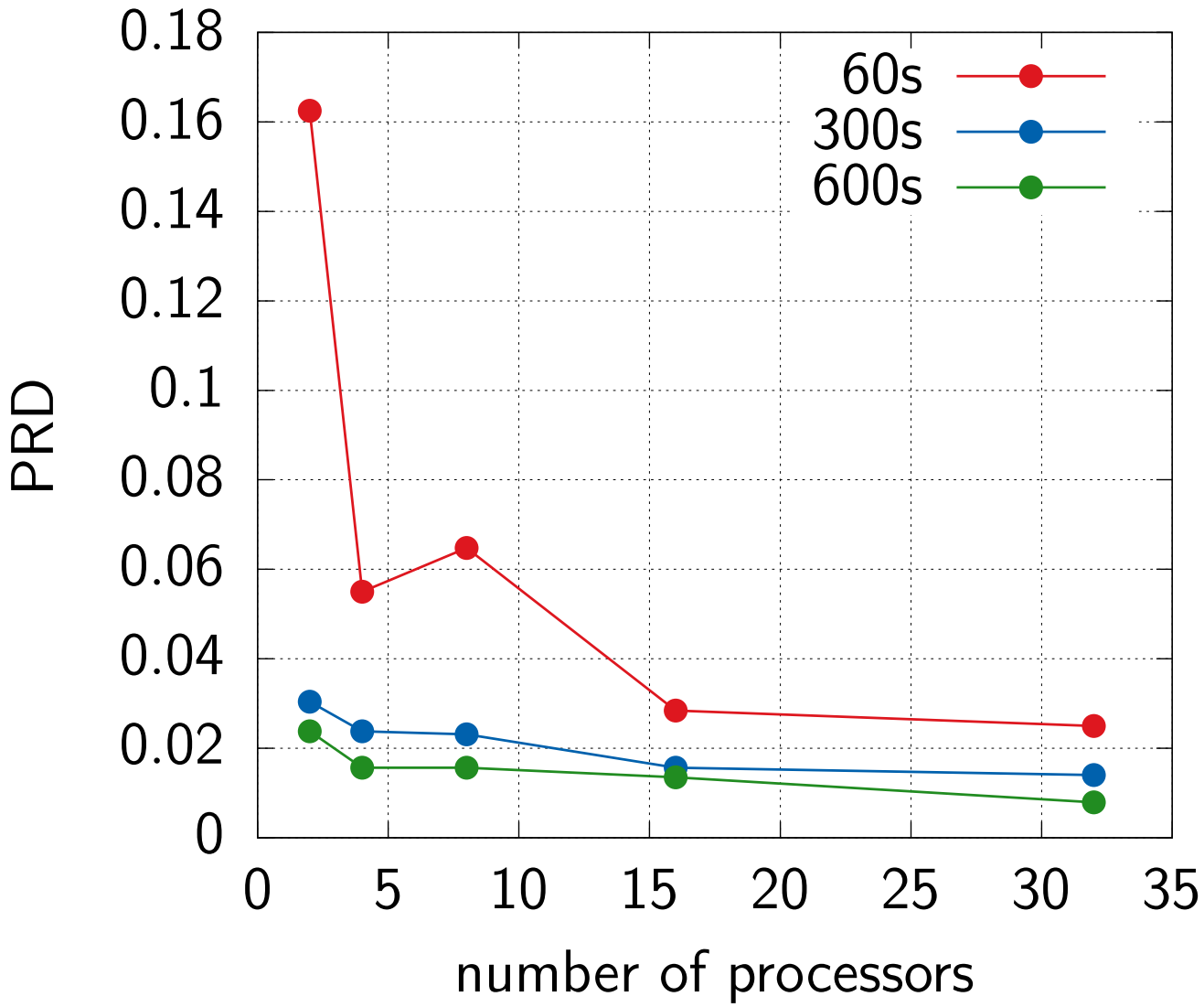
```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0   15 44199.0000    0.00000   100%     -    0s
     0     0    0.00000    0  179 44199.0000    0.00000   100%     -    0s
H    0     0                      42473.000000    0.00000   100%     -    0s
H    0     0                      27512.000000    0.00000   100%     -    0s
H    0     0                      16896.000000    0.00000   100%     -    0s
     0     0    0.00000    0   22 16896.0000    0.00000   100%     -    0s
     0     0    0.00000    0   23 16896.0000    0.00000   100%     -    0s
H    0     0                       3132.0000000    0.00000   100%     -    0s
H    0     0                       1524.0000000    0.00000   100%     -    0s
     0     0    0.00000    0    7 1524.00000    0.00000   100%     -    0s
H    0     0                       1286.0000000    0.00000   100%     -    0s
     0     0    0.00000    0   11 1286.00000    0.00000   100%     -    0s
     0     0    0.00000    0   15 1286.00000    0.00000   100%     -    0s
     0     0    0.00000    0   19 1286.00000    0.00000   100%     -    0s
     0     0    0.00000    0   28 1286.00000    0.00000   100%     -    0s
     0     0    0.00000    0    9 1286.00000    0.00000   100%     -    0s
     0     2    0.00000    0    9 1286.00000    0.00000   100%     -    0s
H  117   106                       1040.0000000    0.00000   100%   183    1s
H  158   152                        206.0000000    0.00000   100%   149    1s
H  162   152                        203.0000000    0.00000   100%   146    1s
H  188   152                        172.0000000    0.00000   100%   131    1s
  9996  4396    0.00000   27   82  172.00000    0.00000   100%  64.0    5s
 30475  3216 infeasible   33       172.00000    0.00000   100%  50.4   10s
 55917  3280    0.00000   35   62  172.00000    0.00000   100%  60.7   15s
 79006  3463    0.00000   32   88  172.00000    0.00000   100%  66.4   22s
 91118  3390 infeasible   35       172.00000    0.00000   100%  69.0   25s
 118109  3576    0.00000   46   24  172.00000    0.00000   100%  73.7   31s
 134203  2833   87.00000   40   30  172.00000    0.00000   100%  76.8   35s

Cutting planes:
```

Example – optimization results



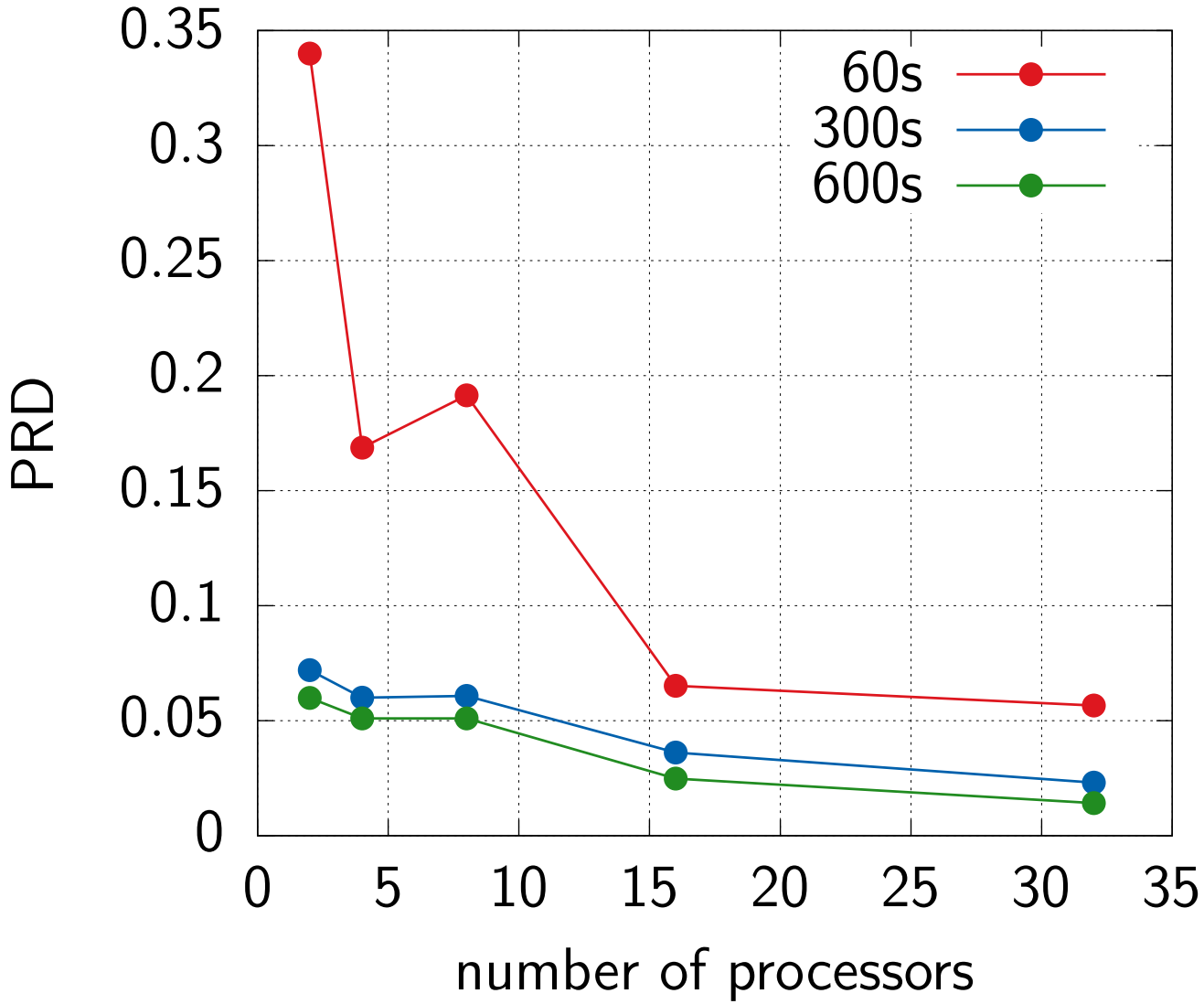
Rysunek: PRD values for $n = 30$.

Example – optimization results



Rysunek: PRD values for $n = 40$.

Example – optimization results



Rysunek: PRD values for $n = 50$.

Conclusions

- A wide range of tools are available to solve discrete optimization problems in production and logistics using parallel computing on HPC infrastructures.
- Using HPC infrastructures allows to solve discrete optimization problems in a reasonable time with satisfied solutions quality.
- Lack of tools for solving directly (with minimal programming effort) discrete optimization problems using distributed computing (multi–nodes).