Project 1
Open shop problem

Santin Filippo
Knowledge and Data Mining
28 September 2021

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Presentation of the problem

In the open shop problem:

- m processors P_1,...,P_m
- k tasks T_1,...,T_k
  Each task T_i is executable by a processor Proc(T_i)
- n jobs J_1,...,J_n
  Each job J_i is a set of Task(J_i) $\subseteq \{T_1, ..., T_k\} \neq \emptyset$
- Construct a schedule so that all jobs are completed

Constraints

Presentation of the problem

- Each task has to be processed for one time unit on a specific processor
- Different tasks of the same job cannot be processed simultaneously
- No processor can work on two tasks at the same time

Change of the preference

Presentation of the problem

Initial propose:

- Minimization of the number of processors

I decided to change it, opting for another choice:

- Number of the processors is given indirectly as input
- New preference: Minimization of time used to execute all the jobs, respecting the other requests

Python

Programming language

Python is an interpreted programming language created by Guido van Rossum in 1990.

- It is fully typed dynamically
- It supports multiple programming paradigms, including structured, object-oriented and functional programming
- intuitive, immediate and elegant

Solver Code "OpenShopSat"

Fundamental steps of my solver function OpenShopSat():

1. Definition of data
2. Definition of variables
3. Definition of constraints and preference
4. Statement of the model and solver
5. Output

Other particular example

Problem Data

Solver Code

```
jobs_values = [[(0, 1), (1, 1), (2, 1), (3,1)],
               [(0, 1), (2, 1), (1, 1)],
               [(3, 1), (1, 1)],
               [(2,1), (1,1), (2,1)]]
```

- **jobs_values = [[job0], [job1], [job2], [job3], ...]**
- **job = [(task0), (task1), (task2), ...]**
- **task = (processor_id_assigned, unit_time)**

```
number_processors = max(task[0] for job in jobs_values
                        for task in job) + 1
```

Variables

Solver Code

Basic variables

```
start_var
end_var
interval_var
```

More complicated variables

```
all_the_tasks = {}
```

I created a start, an end and an interval (of one unit) for each task

```
processors_intervals = collections.defaultdict(list)
```

I associated each processor to intervals (without specific values) of the tasks entrusted to it

```
assigned_jobs = collections.defaultdict(list)
```

I created a list of assigned tasks for each processor

Constraints and Preference

Solver Code

Constraints:

No Overlapping

```
for processor_id in processors:
   model.AddNoOverlap(processors_intervals[processor_id])
```

Precedence inside a job

```
for job_id, job in enumerate(jobs_values):
   for task_id in range(len(job) - 1):
      model.Add(all_the_tasks[job_id, task_id + 1].start
      >= all_the_tasks[job_id, task_id].end)
```

MAX-SAT, minimizing time:

```
time = model.NewIntVar(0, k, 'time')
model.Minimize(time)
```

CP-SAT Solver

Solver Code

At the beginning:

```
model = cp_model.CpModel()
```

In the middle:

```
solver = cp_model.CpSolver()
solution = solver.Solve(model)
```

Output

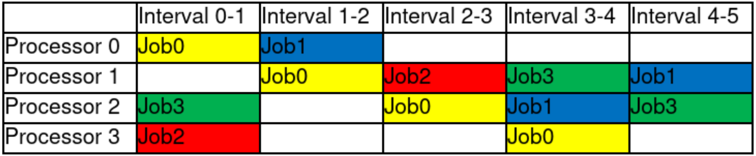
Solver Code

```
Processor 0:    job_0_0  job_1_0
                 [0,1]    [1,2]
Processor 1:    job_0_1  job_2_1  job_3_1  job_1_2
                 [1,2]    [2,3]     [3,4]    [4,5]
Processor 2:    job_3_0  job_0_2  job_1_1  job_3_2
                 [0,1]    [2,3]     [3,4]    [4,5]
Processor 3:    job_2_0  job_0_3
                 [0,1]    [3,4]


Number of processors:  4
Optimal Schedule Time: 5
```

Graphic representation of my output

Solver Code



	Interval 0-1	Interval 1-2	Interval 2-3	Interval 3-4	Interval 4-5
Processor 0	Job0	Job1			
Processor 1		Job0	Job2	Job3	Job1
Processor 2	Job3		Job0	Job1	Job3
Processor 3	Job2			Job0	

THANK YOU FOR THE ATTENTION