Chalmers Tekniska Högskola

Constraint programming and applied optimization (EEN025)

Assignment 2

Group 10

Himanshu Sahni
sahni@chalmers.se
MPCAS

Prajwal Prashant Shetye
shetye@chalmers.se
MPSYS

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part in other solutions.

Task 1

Problem Overview

The JSSP involves scheduling a set of jobs on a set of machines while adhering to various constraints. The objective is to minimize the makespan, which is the completion time of the entire schedule. The solver utilizes Gurobi's CP and MILP solver to find an optimal or near-optimal solution within a specified time frame.

Objective Function

The solver's objective is to minimize the makespan (c_max), which represents the completion time of the entire schedule. The objective function aims to minimize the total time required to complete all jobs while adhering to the defined constraints.

$$\text{Minimize the Makespan: } c\_max$$

Solving using MILP:

Consider a set of jobs, denoted as J, and a set of machines, denoted as M. Each job j must adhere to a specific processing order represented as $(\sigma_1^j, \sigma_2^j, \ldots, \sigma_k^j)$, and each operation (m, j) has an associated processing time p. The decision variables of interest include the start time of job j on machine m, denoted as $x_{m,j}$, a binary variable $z_{m,i,j}$ that indicates the precedence of job i before job j on machine m, and the total makespan of the operations, denoted as C, which is the objective to be minimized.

We need to establish constraints to ensure the following:

1. The initiation time of job j on machine m should be greater than the sum of initiation time of the preceding operation of job j and its corresponding processing time.
2. Each machine can process only one job at a time. To enforce this, we specify that if job i precedes job j on machine m, the initiation time of job j on machine m must be greater than or equal to the initiation time of job i plus its processing duration.
3. For any pair of jobs i and j, it is necessary to establish precedence for one over the other on every machine m within the set M.
4. The overall makespan should be greater than or equal to the initiation time of every operation plus its corresponding processing time.

This leads us to the following mathematical formulation:

$$\begin{aligned}
&\min C \\
&\text{s.t.} \quad && x_{\sigma_{h-1}^j, j} + p_{\sigma_{h-1}^j, j} \le x_{\sigma_h^j, j} && \forall j \in J,\ h \in \{2, \ldots, |M|\} \\
& && x_{m,j} + p_{m,j} \le x_{m,k} + V(1 - z_{m,j,k}) && \forall j, k \in J,\ j \neq k,\ m \in M \\
& && z_{m,j,k} + z_{m,k,j} = 1 && \forall j, k \in J,\ j \neq k,\ m \in M \\
& && x_{\sigma_{|M|}^j, j} + p_{\sigma_{|M|}^j, j} \le C && \forall j \in J \\
& && x_{m,j} \ge 0 && \forall j \in J,\ m \in M \\
& && z_{m,j,k} \in \{0, 1\} && \forall j, k \in J,\ m \in M
\end{aligned}$$

where V is an arbitrarily large value (big M) of the "either-or" constraint

Solving using Constraint Programming:

Constraints

The solver incorporates the following constraints to model the JSSP:

1. **Task Sequencing Constraints:**
   Ensure that tasks are scheduled in the correct sequence on the same machine.
   For each job j, the start time of a task i on a machine $machines[j][i]$ must be greater than or equal to the completion time of the previous task $i-1$ on the same machine.

2. **Makespan Constraint:**
   Ensure the correct calculation of the makespan (completion time).
   Specifies that the completion time (c_max) must be greater than or equal to the start time of the last task on the last machine for each job.

$$
\begin{array}{lll}
\min & C_{\max} & \\
\text{s.t.} & x_{ij} \geq 0, & \forall j \in J, i \in M \\
& x_{\sigma_h^j, j} \geq x_{\sigma_{h-1}^j, j} + p_{\sigma_{h-1}^j, j}, & \forall j \in J, h = 2, \ldots, m \\
& C_{\max} \geq x_{\sigma_{m,j}^j} + p_{\sigma_{m,j}^j}, & \forall j \in J \\
& \text{disjunctive}\left(\{x_{i1}, \ldots, x_{in}\}, \{p_{i1}, \ldots, p_{in}\}\right), & \forall j \in J, i \in M
\end{array}
$$

Note: Solver Configuration

The solver has a time limit of 20 minutes ($model.Params.TimeLimit = 20 \times 60$) to find the optimal solution or an acceptable solution within the specified time frame.

We get the following values of time after running the code with the constraints mentioned above:

Table 1:

Instances	Time taken (MILP)	Time taken (CP)
ft06	56	30
ft10	972	393
la01	663	165
la05	537	164
la10	No solution found	191

Task 2

Given the layout of the Job shop floor:

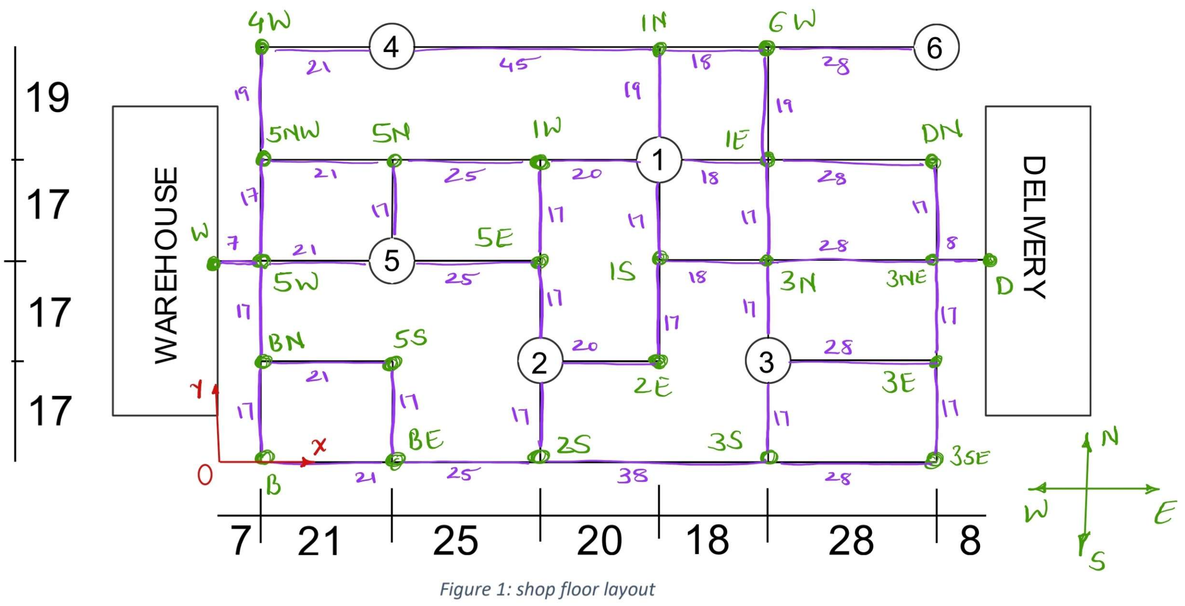

Figure 1: Shop Floor Layout

The following image shows the markings made by us where we have defined a set of nodes to represent distinct locations on the shop floor. Each node is assigned a unique identifier and is positioned relative to the warehouse ("W"). The following nodes have been configured:

1. Node "1", "2", "3", "4", "5", and "6" represent the positions of machines within the plant.
2. Node "W" represents the warehouse, located on the extreme left of the shop floor. Node "D" represents the delivery area, positioned on the extreme right of the shop floor.
3. Intermediate nodes such as "1N", "1E", "1S", "1W" etc., represent various directions or paths within the plant. All the points represented by green dots in the image represent the intermediate nodes.
4. Edges are used to connect nodes and represent potential paths or routes within the plant. Each edge is associated with a weight or distance that indicates the effort or cost required to traverse that particular path. It is important to note that all distances are measured relative to the warehouse ("W"). The lines marked with purple colour represent the edges between nodes.
5. The entire layout is represented as a graph, where nodes are the vertices, and edges are the connections between them. Nodes and edges are interconnected in a way that facilitates movement between machines, warehouse, and delivery areas efficiently.

The notations used by us in our implementation(code) are explained as follows:

1. There are a total of 6 machine nodes in the plant layout, which have been conveniently grouped and represented as an array named `machine_node`.

2. There are a total of 30 nodes in the plant layout, which have been grouped and represented by an array named `nodes`. every node present in the array `nodes` is stored as a key value pair.

   for example: node4W = Node("4W", "x":7, "y":70)

   'node4w' as shown in the image represents the node that is located in the west of machine node 4. The values x and y represent the distance of the node from the warehouse (W).

3. All the edges between the nodes are stored in an array named `edges`.

   for example: Edge(node4W, node4, 21), creates an edge between two nodes, "4W" and "4," with a cost of 21 units. This edge likely represents a specific pathway or route between these two locations on the shop floor layout.

A* Algorithm Implementation

We have described here the implementation of the A* algorithm for finding the shortest path in the graph. The algorithm is described in pseudocode, emphasizing key steps and logic, and is an integral part of our project for solving complex routing problems.

Algorithm 1 A* Algorithm

```
procedure AStar(start : Node, end : Node, heuristic : callable = default_heuristic)
    open_list ← []                                   ▷ Priority queue for nodes to explore
    closed_set ← set()                               ▷ Set of visited nodes
    g_scores ← {}                                    ▷ Cost from start to each node
    f_scores ← {}                                    ▷ Estimated total cost from start to end through each node
    parent_map ← {}                                  ▷ Parent pointers to reconstruct the path
    g_scores[start] ← 0                              ▷ Initialize scores for start node
    f_scores[start] ← g_scores[start] + heuristic(self, end, start)
    open_list.append((f_scores[start], start))
    while open_list is not empty do
        open_list.sort()                             ▷ Sort the open list by $f$-score
        _, current_node ← open_list.pop(0)
        if current_node == end then
            path ← []
            while current_node do
                path.insert(0, current_node)         ▷ Reconstruct the path
                current_node ← parent_map.get(current_node)
            return g_scores[end], path               ▷ Return cost and path
        closed_set.add(current_node)
        for edge in self.edges do
            if edge.start == current_node and edge.end not in closed_set then
                tentative_g_score ← g_scores[current_node] + edge.weight
                tentative_f_score ← tentative_g_score + heuristic(self, end, edge.end)
                if tentative_f_score < f_scores.get(edge.end, float('inf')) then
                    parent_map[edge.end] ← current_node
                    g_scores[edge.end] ← tentative_g_score
                    f_scores[edge.end] ← tentative_f_score
                    open_list.append((f_scores[edge.end], edge.end))
```

We have used this to calculate the shortest paths between any pair of points in the plant. For example, here are the results that we obtained for the job schedule "ft06" when we ran the algorithm.

Problem Overview

The JSSP involves scheduling a set of jobs on a set of machines while adhering to various constraints. The objective is to minimize the makespan, which is the completion time of the entire schedule. This corresponding formulation considers both the scheduling of jobs on machines and the travel times between machines while minimizing the makespan.

Objective Function

The objective is to minimize the makespan (c_max), which represents the completion time of the entire schedule i.e., to complete all jobs while adhering to the defined constraints.

$$\text{Minimize the Makespan: } c\_max$$

Problem Formulation:

In this case, the addition we have is the travelling time between machines which needs to be taken into account while solving.

We need to establish constraints to ensure the following:

1. The initiation time of job j on machine m should be greater than the sum of the initiation time of the preceding operation of job j and its corresponding processing time, considering travel times:

$$x_{\sigma^j_{h-1},j} + p_{\sigma^j_{h-1},j} + t_{\sigma^j_{h-1},m} \leq x_{\sigma^j_h,j} \quad \forall j \in J,\, h \in \{2,\ldots,|M|\}$$

2. Each machine can process only one job at a time. To enforce this, we specify that if job i precedes job j on machine m, the initiation time of job j on machine m must be greater than or equal to the initiation time of job i plus its processing duration.

$$x_{m,j} + p_{m,j} \leq x_{m,k} + V(1 - z_{m,j,k}) \quad \forall j,k \in J,\, j \neq k,\, m \in M$$

3. For any pair of jobs i and j, it is necessary to establish precedence for one over the other on every machine m within the set M:

$$z_{m,j,k} + z_{m,k,j} = 1 \quad \forall j,k \in J,\, j \neq k,\, m \in M$$

4. The overall makespan should be greater than or equal to the initiation time of every operation plus its corresponding processing time along with the traveling time:

$$x_{\sigma^j_{|M|},j} + p_{\sigma^j_{|M|},j} \leq C \quad \forall j \in J$$

5. Non-negativity constraints:

$$x_{m,j} \geq 0 \quad \forall j \in J,\, m \in M$$

$$z_{m,j,k} \in \{0,1\} \quad \forall j,k \in J,\, m \in M$$

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 2)	142	28.4	W-5W-5-5E-2-2S-3S-3
(2, 0)	52	10.4	3-3N-1S-1
(0, 1)	54	10.8	1-1W-5E-2
(1, 3)	118	23.6	2-5E-1W-1-1N-4
(3, 5)	91	18.2	4-1N-6W-6
(5, 4)	127	25.4	6-6W-1N-1-1W-5N-5
(4, D)	133	26.6	5-5E-1W-1-1E-DN-3NE-D

Table 2: JOB 1

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 1)	70	14.0	W-5W-5-5E-2
(1, 2)	72	14.4	2-2S-3S-3
(2, 4)	114	22.8	3-3N-1S-1-1W-5N-5
(4, 5)	127	25.4	5-5E-1W-1-1E-6W-6
(5, 0)	65	13.0	6-6W-1N-1
(0, 3)	64	12.8	1-1N-4
(3, D)	135	27.0	4-1N-6W-1E-DN-3NE-D

Table 3: JOB 2

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 2)	142	28.4	W-5W-5-5E-2-2S-3S-3
(2, 3)	116	23.2	3-3N-1S-1-1N-4
(3, 5)	91	18.2	4-1N-6W-6
(5, 0)	65	13.0	6-6W-1N-1
(0, 1)	54	10.8	1-1W-5E-2
(1, 4)	42	8.4	2-5E-5
(4, D)	133	26.6	5-5E-1W-1-1E-DN-3NE-D

Table 4: JOB 3

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 1)	70	14.0	W-5W-5-5E-2
(1, 0)	54	10.8	2-2E-1S-1
(0, 2)	52	10.4	3-1E-3N-3
(2, 3)	116	23.2	3-3N-1S-1-1N-4
(3, 4)	78	15.6	4-4W-5NW-5N-5
(4, 5)	127	25.4	5-5E-1W-1-1E-6W-6
(5, D)	100	20.0	6-6W-1E-DN-3NE-D

Table 5: JOB 4

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 2)	142	28.4	W-5W-5-5E-2-2S-3S-3
(2, 1)	72	14.4	3-3S-2S-2
(1, 4)	42	8.4	2-5E-5
(4, 5)	127	25.4	5-5E-1W-1-1E-6W-6
(5, 0)	65	13.0	6-6W-1N-1
(0, 3)	64	12.8	1-1N-4
(3, D)	135	27.0	4-1N-6W-1E-DN-3NE-D

Table 6: JOB 5

Result:

The time taken to complete the JOB1 (ft06) with the new set of constraints comes to 134 units.

Machine Pairs Nodes	Distances between Machines Nodes	Time taken	Path from start to end node
(W, 1)	70	14.0	W-52-5-5E-2
(1, 3)	118	23.6	2-5E-1W-1-1N-4
(3, 5)	91	18.2	4-1N-6W-6
(5, 0)	65	13.0	6-6W-1N-1
(0, 4)	62	12.4	1-1W-5N-5
(4, 2)	114	22.8	5-5E-2-2S-3S-3
(2, D)	53	10.6	3-3E-3N3-D

Table 7: JOB 6

Task 3

The motive of this task was to find 10 shortest path from machine 5 to machine 3.

Constraints

The constraints for this problem are:

1. For both source and destination nodes, only one of the neighboring edges will be part of the path. i.e. sum of all edges directly connected to source node must be equal to 1 (similar for destination node).

2. We must never land into nodes that are dead ends i.e. blocking nodes. i.e. sum of all edges directly connected to blocking node must be equal to 1.

3. All other nodes (except the source and destination nodes) will either be

   (a) part of the path or

   (b) not part of the path.
   A *pathCheck()* function is implemented for the same as shown in Listing (1).

```
def pathCheck(node:Int, edge_list:list, n:int) -> Bool:
    # here, n is number of edges active at a time.
    # n = 2 if there exist seperate entry and exit edges to and from
    node
    # n = 1 if there is same edge for entry and exit to and from
   node
    return n*IntNot(node) + sum(edge_list) == n

```

Listing 1: Task 3: pathCheck function to establish valid connections between nodes

Objective function

$$\text{minimize}\left(\sum_{\forall i}\text{weight}_i \times \text{var}_i\right) \tag{1}$$

where, weight_i represents cost of edge i and var_i represents boolean variable of edge i for selection.

Integer variables treated as Boolean

The modelling was meant to be done using boolean satisfiability (SAT). Although z3 supports variables of boolean data type, integer variables were used as booleans i.e. integers with domain $[0, 1]$ since they have same meaning ($0 =$ False, $1 =$ True).

This was done so that arithmetic operations can be performed directly on the variables rather than performing other manipulations. To perform logical operation *Not()* an implementation of *IntNot()* was done as shown in Listing (2)

```python
def IntNot(x:Int) -> Int:
    return (1 - x)
```

Listing 2: Task 3: Integer version of Not() function

Results

The result of this task is illustrated in Table (8). It may be noted that e_5_5E represents the edge connection node 5 and node 5E with respect to Figure (1).

#	Path from machine 5 to machine 3	Obj
0	[e_5_5E, e_5E_2, e_2_2E, e_1S_2E, e_1S_3N, e_3N_3]	114
1	[e_5_5E, e_5E_2, e_2_2S, e_2S_3S, e_3_3S]	114
2	[e_5_5E, e_1W_5E, e_1W_1, e_1_1S, e_1S_3N, e_3N_3]	114
3	[e_5N_5, e_5N_1W, e_1W_1, e_1_1S, e_1S_3N, e_3N_3]	114
4	[e_5_5E, e_1W_5E, e_1W_1, e_1_1E, e_1E_3N, e_3N_3]	114
5	[e_5N_5, e_5N_1W, e_1W_1, e_1_1E, e_1E_3N, e_3N_3]	114
6	[e_5N_5, e_5N_1W, e_1W_5E, e_5E_2, e_2_2E, e_1S_2E, e_1S_3N, e_3N_3]	148
7	[e_5N_5, e_5N_1W, e_1W_5E, e_5E_2, e_2_2S, e_2S_3S, e_3_3S]	148
8	[e_5_5E, e_5E_2, e_2_2E, e_1S_2E, e_1_1S, e_1_1E, e_1E_3N, e_3N_3]	148
9	[e_5_5E, e_1W_5E, e_1W_1, e_1N_1, e_1N_6W, e_6W_1E, e_1E_3N, e_3N_3]	152

Table 8: Ten shortest path from machine 5 to machine 3