Job Shop Scheduling Problem with GRASP

Joan Fernández Navarro

HEURISTIC ALGORITHMS IN TRANSPORT AND FINANCE

Contents

Introduction

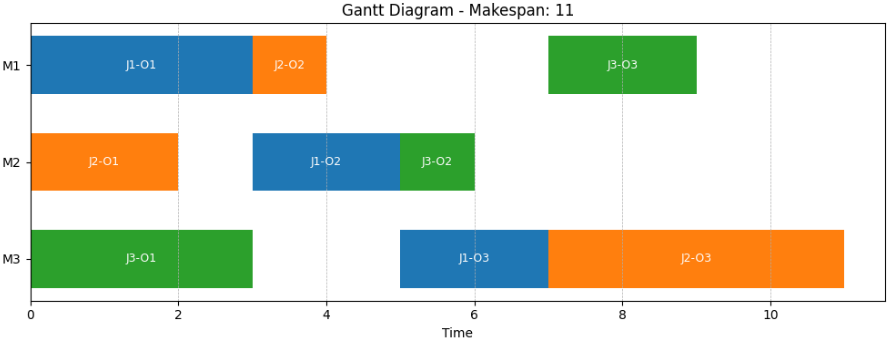
- **Problem**: Schedule a set of jobs on a set of machines
- **Each job** has a specific sequence of operations to follow
- **Each operation** must be processed on a specific machine for a given time
- **Constraints**:
  - One machine can process only one operation at a time
  - Operations of the same job must follow predefined sequence

Objective

Minimize the makespan
(Total completion time of all jobs)

Example

Job	Operations
J1	(M1,3), (M2,2), (M3,2)
J2	(M2,2), (M1,1), (M3,4)
J3	(M3,3), (M2,1), (M1,2)



Mathematical Definition

- **Jobs**: $J = \{J_1, J_2, ..., J_n\}$
- **Machines**: $M = \{M_1, M_2, ..., M_m\}$
- **Operations**: O_{ij} for job i on machine j
- **Processing time**: p_{ij} for operation O_{ij}
- **Machine sequence**: π_i for job i

Objective Function

$$\text{minimize } C_{max} = \max\{C_i : i = 1, ..., n\}$$

where C_i is completion time of job i

Instances

Source: JSSP Instances and Results Repository

- First line: number of jobs and machines
- Following lines: machine sequences and processing times per job
- Best Known Solution (BKS) available

```
20 15
  6 105   5  16   0  48   3 114   1  54   2  90   4 159  13 133  10  65  14 153   8 146   9  28  11  15  12  41   7 141
  6  29   1  82   3  40   0  45   2 151   4 103   5  24   7 117  11  55  12 185   9  25  13 119  10 168   8  58  14   3
  0  26   2 190   4  40   1 135   3 120   5   4   6 126   9  36   7 127  10  99  11  99  12  62   8 174  13 155  14 143
  0 140   1  21   4  74   2 194   5   9   3 173   6 169  13 128  12 183  10  93  11 122   7  91  14  56   8  18   9  52
  4  33   2 196   1 160   6   6   0  22   3 164   5 125  10 150   8  19  14  73  11  40   9 105  13  39  12  93   7 105
  3  89   6 174   4  75   2  35   0 118   5 117   1  33   8  40   9  97  11  82  13 113  10 130  12 130   7  99  14   5
  6 141   3 141   0  11   2  15   1  79   5  78   4 134  14  41   7 180  10 130  13 134   9 187  12 131   8  40  11  64
  1   9   2  98   5 172   0 101   6   2   4  74   3 135   8  32  13 176   7 174  12  95   9 105  10  36  14 165  11 190
  6 156   2 135   5  95   0 161   1 110   4 196   3  33   9 188   8  45  14 159   7 137  11  31  13 112  12  83  10  89
  2  30   0 166   5  99   6 165   3  59   1 151   4 122  10 128  13 193   8  38   7 118  11 136  14 127  12 154   9  82
  2  67   1 154   6 125   0  57   5  24   4  21   3  11  14  79   8 142  13  29  10 186  11 107  12  57   7 192   9 197
  5 118   1  54   6  74   4  94   3 123   0  88   2 182  14  35  13 120   7 190  11 159   9  92  10  34  12  80   8 101
  2  15   0 108   3 163   5 118   4 107   1  30   6 126   8 189  11   7   9 131  13 122   7 173  12 173  10 132  14   8
  1 127   2 151   4   1   3 116   5 158   0 148   6 131  12  36   9 194  13  92  11  23   8 177   7  65  10 165  14 182
  6 125   2   5   3  31   4  50   5 174   0  94   1  98   7  38   9  26   8 162  10   5  11  87  13 101  14 141  12  11
  1 198   6  65   0 173   3 169   4  46   2 164   5 151   9  14   8 150  14 108  13 123  10  67  12  14  11 166   7 116
  0 174   2 165   3 180   1 115   6 185   4 100   5 122  11  38  14 112   9  93  10 137   7  74  12  34  13  50   8  42
  3  95   6  15   1  13   4 103   2  42   0 187   5 188  12 143  13  87   9 182  14  21  10 165   7  69  11   4   8  55
  6  13   0 134   5 177   1  40   3 132   4   1   2  42  11  35  13  65  14  92   7 121   9 111  12  94  10  27   8 137
  5 111   6 167   3 117   0   3   4  76   2 110   1  79  13 142   9 200  14  33  11 148   8  90  10  82  12 163   7  25
```

Figure 1: Instance 'cscmax_20_15_1'

Implementation

Classes

JSSPInstance: Represents a problem instance

- Stores parameters: jobs, machines, sequences, times
- Methods: load instances from files

JSSPSolution: Represents a candidate solution

- Stores: schedule, makespan, feasibility
- Methods: makespan calculation, feasibility verification

```
class JSSPInstance:
    def __init__(self):...

    def load_instance(self, instance_name: str):...


class JSSPSolution:
    def __init__(self, instance: JSSPInstance):...

    def calculate_makespan(self):...

    def is_feasible(self):...
```

Greedy Solution

Algorithm 1: Greedy Solution

Create list of available operations;
Sort operations by processing time (shortest first);
while *available operations not empty* **do**
    Select operation with shortest processing time;
    Schedule at max(machine available time, job available time);
    Update machine and job available times;
    Add next operation of the job to available list;
end
return solution;

Algorithm 2: GRASP

Initialize solution and data structures;
while *iterations* $\leq$ *max. iterations* **do**
    solution $\leftarrow$ GRASP Construction;
    improved $\leftarrow$ Local Search(solution);
    if *improved* $\geq$ *best solution* **then**
        best solution $\leftarrow$ improved;
        Update data structures;
    end
end
return best solution;

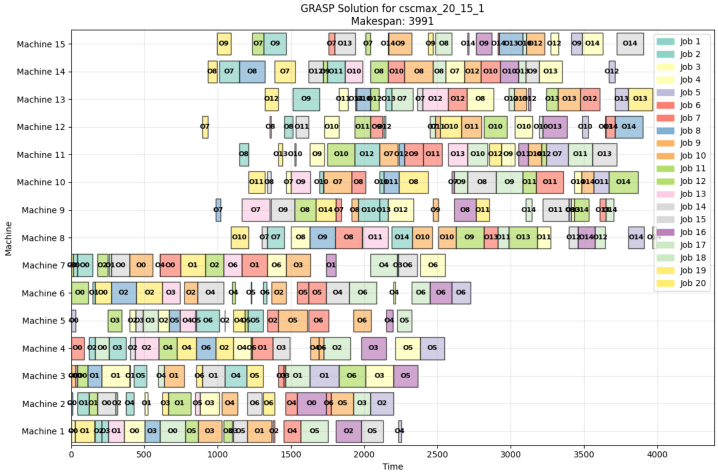
GRASP Solution

Algorithm 3: GRASP Construction

```
Initialize solution and data structures;
Create candidate list of available operations;
while available operations not empty do
    Calculate estimated finish times for all candidates;
    Sort candidates by estimated finish time;
    Calculate threshold using α;
    Build RCL with candidates ≤ threshold;
    Select one candidate randomly from RCL;
    Schedule selected operation;
    Update data structures;
end
return solution;
```

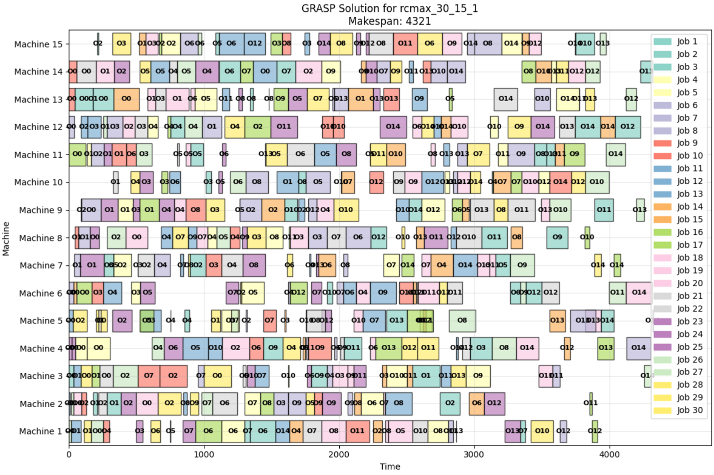
Results

Solution for Instance 'cscmax_20_15_1'



Greedy: 14,389 - **GRASP**: 3,991 - **Best Known**: 3,272

Solution for Instance 'rcmax_30_15_1'



Greedy: 22,426 - **GRASP**: 4,321 - **Best Known**: 3,343

Convergence Plot

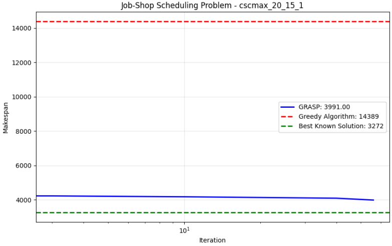


Figure 2: Convergence plot for instance 'cscmax_20_15_1'

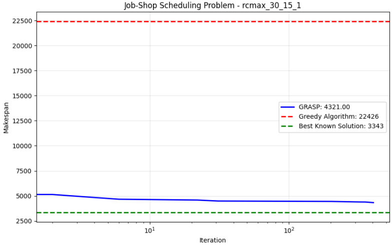


Figure 3: Convergence plot for instance 'rcmax_30_15_1'

Computational Results

Instance	Greedy	GRASP	Best Known	Time (s)
cscmax_20_15_1	14,389	3,991	3,272	30.12
rcmax_30_15_1	22,426	4,321	3,343	40.76
rcmax_30_20_10	34,104	4,888	3,814	58.62
rcmax_40_20_3	48,442	6,369	4,848	80.47
cscmax_50_15_4	32,050	8,267	6,196	73.86

Table 1: Comparison of results for different JSSP instances

Performance Analysis

- **GRASP vs Greedy**: Average improvement of **81%**
- **GRASP vs Optimal**: Average gap of **24%**
- **Computation Time**: Scales with instance size (30-75 seconds)
- **Solution Quality**: Consistent performance across instances

Conclusion

- **Quality Gap**: GRASP solutions remain worse than optimal solutions
- **Effective Balance**: Good trade-off between solution quality and computation time
- **Robust Performance**: Consistent results across different problem instances
- **Reduce Iterations**: The convergence plots show that best solutions are reached at early stages

Thank You!