An Optimization Model for Scheduling Problems with Two-Dimensional Spatial Resource Constraints

Christopher Garcia, Ghaith Rabadi

Engineering Management and Systems Engineering
Old Dominion University

cgarc001@odu.edu, grabadi@odu.edu

Abstract. Traditional scheduling problems involve determining temporal assignments for a set of jobs in order to optimize some objective. Some scheduling problems also require the use of limited resources, which adds another dimension of complexity. In this paper we introduce a spatial resource-constrained scheduling problem that can arise in assembly, warehousing, cross-docking, inventory management, and other areas of logistics and supply chain management. This scheduling problem involves a two-dimensional rectangular area as a limited resource. Each job, in addition to having temporal requirements, has a width and a height and utilizes a certain amount of space inside the area. We propose an optimization model for scheduling the jobs while respecting all temporal and spatial constraints.

INTRODUCTION

Scheduling problems arise in many areas of business and industry. Common to all types of scheduling problems is the need to assign a set of resources or jobs to a set of time slots. Beyond this, different types of scheduling problems present their own unique sets of objectives and constraints and require individualized formulation and solution methods. In this paper, we examine a problem that involves scheduling jobs that have two spatial dimensions, width (x) and height (y), in addition to having required processing times, deadlines, and earliest start times. These jobs must be processed inside a two-dimensional processing area which has its own width and height. Consequently, solving this problem involves simultaneously determining both the time each job should be processed as well as the spatial location and layout of each job within the processing area.

To demonstrate an instance of this problem, consider the following jobs listed below in Table 1. Assume that we are given a processing area having a width of 10 and a height of 8 in which these jobs must be processed, and that our objective is to minimize the total tardy time. In order to solve this problem we must determine both a start time and a coordinate for each job. To complicate matters further, we also assume that the *layout* of any job can be changed, which is accomplished by rotating the job by 90 degrees. This results in a swapping of the job's width and height. For instance, Job 1 has a width of 4 and a height of 5. If its layout is changed, Job 1 will have a width of 5 and a height of 4. An optimal solution to this problem is given in Table 2. The lower-left corner of the processing area can be understood as the (0, 0) coordinate, enabling the solution to be visualized as shown in Figure 1.

Although much literature exists for both box-packing problems and scheduling problems, there is relatively little literature that directly addresses spatial scheduling. Literature addressing this topic directly addresses a more specialized problem encountered in shipbuilding [1], [2]. To our knowledge the problem proposed in this paper has not been discussed in previous literature, and we are aware of no previous problem instances.

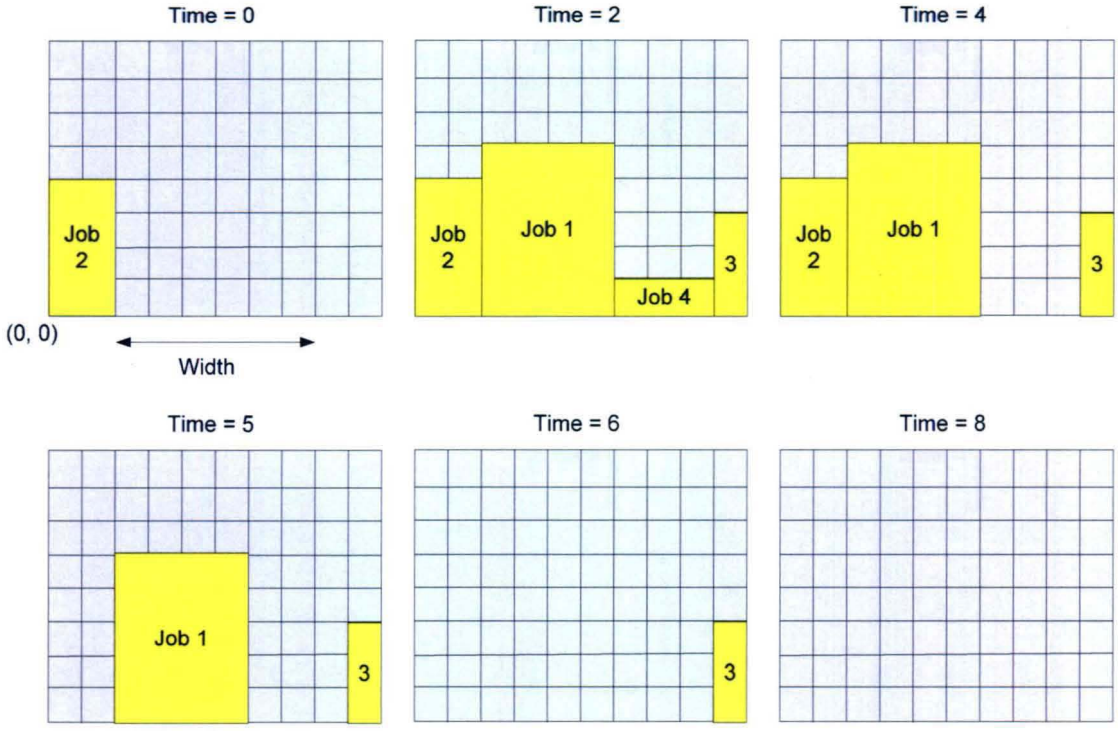
Table 1: A small problem instance

Job	Width	Height	Earliest Start	Processing Time	Deadline
1	4	5	2	4	8
2	2	4	0	5	6
3	1	3	2	6	10
4	3	1	0	2	9

Table 2: An optimal solution to the small problem instance, in ascending start-time order

Job	Start	End	Due	Tardy	X	Y	Width	Height
2	0	5	6	0	0	0	2	4
1	2	6	8	0	2	0	4	5
4	2	4	9	0	6	0	3	1
3	2	8	10	0	9	0	1	3

Figure 1: A visualized solution to the small problem instance



A GREEDY APPROXIMATION HEURISTIC FOR OPTIMIZATION

The incorporation of spatial resources into scheduling problems introduces a considerable amount of complexity [6]. Additionally, because we were not aware of any prior work on this problem our aim was to develop a computationally fast approximation heuristic to provide baseline solutions for evaluating the performance of future methods. The algorithm we developed combines a two-dimensional box-packing algorithm with an earliest-deadline-first scheduling algorithm. Each job is understood to be a structure having an (X,Y) coordinate, width, height, processing time (duration), earliest allowable start time, deadline, start time, and end time.

Spatial Operations

The spatial operations are used to determine the spatial location and layout for each job within the processing area. We assume we are given a width and height of the processing area as well as constants *m* and *n*. The area is divided into *(m x n)* discrete units – *m* units in the width dimension and *n* in the height dimension. Thus, each unit's width is *(width / m)* and its height is *(weight / n)*. An *(m x n)* matrix of binary values called *Area* is used to keep track of which units are covered by a job. Here, 1 denotes that the corresponding unit is occupied and 0 denotes that it is not. The packing procedure returns either a coordinate for the job (in the case of success) or Ø if it is unable to fit the job into the area.

Three specific procedures are used: PACK, GREEDY_PACK, and REMOVE. PACK is the top-level packing procedure and attempts to find a feasible coordinate and layout for a given job. GREEDY_PACK attempts to fit a job into the available space without changing the layout. REMOVE frees the space occupied by a job. These procedures are described in pseudocode in Table 3. Comments follow the # symbol.

Table 3: The PACK, GREEDY_PACK, and REMOVE procedures

```
Procedure PACK (Job) DO:
  Coordinate := GREEDY_PACK(Job)
  IF Coordinate = Ø DO:
    # Change layout – rotate job 90 degrees
    SWAP(Job.width, Job.height)
    RETURN GREEDY_PACK(Job)
  ELSE
    RETURN Coordinate
  END
END
```

```
Procedure GREEDY_PACK (Job) DO:
  FOR 1 ≤ i ≤ m – ⌈(m)(Job.width) / width⌉ DO:
    FOR 1 ≤ j ≤ n - ⌈(n)(Job.height) / height⌉ DO:
      P := { p | i ≤ p ≤ i + ⌈(m)(Job.width) / width⌉ }
      Q := { q | j ≤ q ≤ i + ⌈(n)(Job.height) / height⌉ }
      IF Area[p][q] = 0 for all p ∈ P and q ∈ Q
       DO:
        Area[p][q] := 1 for all p ∈ P and q ∈ Q
        Job.X := (width)(i) / m
        Job.Y := (height)(j) / n
        RETURN (Job.X, Job.Y)
      END
    END
  END
  RETURN Ø
END
```

```
Procedure REMOVE (Job) DO:
  i := Job.X / m
  j := Job.Y / n
  P := { p | i ≤ p ≤ i + ⌈(m)(Job.width) / width⌉ }
  Q := { q | j ≤ q ≤ i + ⌈(n)(Job.height) / height⌉ }
  FOR ALL p ∈ P and q ∈ Q DO:
    Area[p][q] : = 0
  END
END
```

The Scheduling Algorithm

The scheduling algorithm is concerned with two types of events: 1) the next time one or more jobs are eligible to be processed/added to the area, and 2) the next time one or more jobs are complete and can be removed from the processing area. Two job lists are utilized to keep track of these concerns. *Open* is a list of all unscheduled jobs, sorted in ascending order by earliest start time. *In_Processing* is a list of all jobs currently inside the processing area and is sorted in ascending order by end time. It should be noted that ADD and DELETE operations on these lists preserve their order. The algorithm assigns job times as it moves jobs from *Open* into *In_Processing*. The pseudocode for this procedure is found in Table 4 below, and comments follow the # symbol.

Table 4: The scheduling algorithm

```
Procedure SCHEDULE (Jobs)  DO:
   Open := SORT Jobs BY earliest start time
   In_Processing := Ø
   Time := 0

   WHILE Open IS NOT EMPTY DO:

     Next_Finished_Job := NEXT_ELEMENT(In_Processing)

     # Remove jobs that are finished
      WHILE In_Processing IS NOT EMPTY AND Next_Finished_Job.end_time ≤ Time  DO:
          REMOVE(Next_Finished_Job)   # Procedure defined in Table 3 above
          DELETE Next_Finished_Job FROM In_Processing
          Next_Finished_Job := NEXT_ELEMENT(In_Processing)
      END

      # Add jobs until we run out of space or all open jobs have been added
      Next_Job := NEXT_ELEMENT(Open)
      Time := Next_Job.earliest_allowable_start_time
      Coordinate := PACK(Next_Job)   # Procedure defined in Table 3 above

      WHILE Open IS NOT EMPTY DO AND Coordinate ≠ Ø DO:
           Next_Job.start_time := Time
           Next_Job.end_time := Time + Next_Job.duration
           DELETE Next_Job  FROM Open
           ADD Next_Job TO In_Processing
           Next_Job := NEXT_ELEMENT(Open)
           Time := Next_Job.earliest_allowable_start_time
           Coordinate := PACK(Next_Job)
      END
   END
END
```

RESULTS

We developed a Java implementation of this greedy spatial scheduling algorithm. We also developed an algorithm to generate random problem instances. For each job, the width and height were generated using a uniform distribution over [1, width or height of area]. Durations were generated using a uniform distribution over [5, 25]. Earliest allowable start times and deadlines were generated using an incremented *current time* and a *tightness* factor

ranging from 1 to 10, with 10 generating the most tightly-packed problems. Earliest allowable start times were generated by

E = current time – r

where *r* is a random number uniformly distributed over [0, *(current time) / tightness*]. Deadlines were generated by

*D = current time + current job duration + (r * tightness)*

where *r* is a random number uniformly distributed over [0, *current job duration*]. Finally, *current time* is initialized to 0 prior to the generation of any job and subsequently incremented after each job generation by

*Increment = 10 * r / tightness*

where *r* is a random number uniformly distributed over [0, *current job duration*].

Ten generated problem instances of varying sizes were selected to be solved by the spatial scheduling algorithm, and the results are reported in Table 4 below. In each problem instance a width of 10 and a height of 7 were specified for the processing area. As can be seen, a higher tightness parameter results in a lower maximal deadline for a given number of jobs. Thus, a higher tightness parameter results in a more tightly-packed problem instance.

Table 4: The results for several generated problem instances

Problem Instance	Number of Jobs	Tightness	Maximal Deadline	Total Tardiness (Objective Function)	Computational Time (milliseconds)
E-100	100	3.3	2608	0	16
H-100	100	9.9	877	497	16
E-500	500	3.3	11,231	0	94
H-500	500	9.9	3640	51219	94
E-1000	1000	3.3	21,971	0	234
M-1000	1000	8.5	8393	132	312
H-1000	1000	9.9	7239	180,473	328
E-10000	10,000	3.3	222,491	0	10,172
M-10000	10,000	8.9	80,259	22,903	12,812
H-10000	10,000	9.9	73,381	3,426,391	13,062

FUTURE WORK

Much literature exists for many types of scheduling and box-packing problems. There is relatively little literature, however, that directly addresses the topic of spatial scheduling. The work in this paper is quite preliminary in nature, and there are many aspects of this scheduling sub-discipline to be explored. Future theoretical work includes the development of mathematical models for different types of spatial scheduling problems as well as an analysis of the complexity of these problems. Future applied work includes the development of new algorithms and heuristics that can provide reliably near-optimal solutions within a reasonable amount of time.

REFERENCES

1. *Modeling and Solving the Spatial Block Scheduling Problem in a Shipbuilding Company.* **Kyungchil Park, Kyungsik Lee, Sungsoo Park, Sunghwan Kim.** 1996, Computers & Industrial Engineering 30, pp. 357-364.

2. *A Spatial Scheduling System and its Application to Shipbuilding: DAS-Curve.* **Kyoung Jun Lee, jae Kyu Lee.** 1996, Expert Systems with Applications 10, pp. 311-324.

3. *Spatial Block Arrangement in Shipbuilding Industry Using Genetic Algorithm for Obtaining Solution for Anticipated Bottleneck.* **Ranjan Varghese, Duck Young Yoon.** Seoul, Korea : The International Society of Offshore and Polar Engineers, 2005. Proceedings of the Fifteenth International Offshore and Polar engineering Conference.

4. *A new exact method for the two-dimensional bin-packing problem.* **Francois Clautiaux, Jacques Carlier, Aziz Moukrim.** 2007, European Journal of Operations Research 183, pp. 1196-1211.

5. *Two- and three-dimensional parametric packing.* **F.K. Miyazawa, Y. Yakabayashi.** 2005, Computers and Operations Research 34, pp. 2949-2603.

6. *Adjacent Resource Scheduling: Why Spatial Resources are so Hard to Incorporate.* **Jacob Jan Paulus, Johann Hurink.** 2006, Electrinic Notes on Discrete Mathematics 25, pp. 113-116.