A Heuristic-Based CarShop Scheduling Application

Venkatesh Srinivasan
Department of Computer Science &
Center for Automation and Intelligent Systems Research,
Case Western Reserve University,
Cleveland, OH 44106.
venky@alpha.ces.cwru.edu

William Fabens
Information Technology Branch,
B.P. Research,
Warrensville, OH 44128.

Abstract
Scheduling is a complex process involving several jobs, resources and constraints. In this paper we describe the formulation of a heuristic based carshop scheduling application. The CarShop scheduling problem involves scheduling repair jobs on cars, given restrictions on operator availability and other resource/time constraints. The problem is solved by taking a intelligent generate and test approach and extending the simple notion of scheduling - the allocation of resources to tasks over time within constraints defining the system. Dispatch of tasks for scheduling and allocation of resources to them are guided by a set of heuristics. The system is built on a user-extensible knowledge base of rules and heuristics written in Prolog. The emphasis in the system is on providing a flexible AI problem representation and also collecting some empirical results on the performance of different heuristics in the system.

1 Introduction
CarShop scheduling like other scheduling problems is a NP Complete problem. Most literature available on scheduling is traditionally on Job Shop and Flow Shop scheduling [Bak74, Fre82]. The chief dimensions of any scheduling problem are Tasks, Resources and Time which map in the CarShop scheduling instance into car repair jobs, operators with tools/machines and time units. Each car repair job may be comprised of several tasks to be done on the same car, and is considered complete when all the tasks in it are finished. A related problem has been solved in [DSH88], where the emphasis is on sequencing.

Scheduling problems are routinely solved using Operations Research methods, which seek to represent the schedule generation machinery within mathematically defined constraints. Since the problem of CarShop scheduling is NP Complete, any method which seeks to pick an optimal schedule has a flavor of a generate and test strategy, whereby feasible solutions are generated and tested for optimality. Reported methods for scheduling range from "Constraint Directed" scheduling [Fox87] which focuses on knowledge representation and integration of constraints to the search process during scheduling; to an Expert Systems approach for scheduling as described in [GKM90, DS85, SC87]. In the present scheduling problem the primary objective of the schedule is to schedule all the jobs, subject to resource allocation and temporal constraints. Some of the secondary objectives considered are minimizing cost of schedule, where costs are incurred because operators have a cost associated for every unit period of allocation time; another secondary objective could be minimizing total completion time (makespan) of the schedule.

As the solution is of a generate and test nature, the scheduler picks up one of the available combination of task and operator selection heuristics, and generates complete schedules. The best solution in a finite number of iterations is reported. Other interesting scheduling applications written in Prolog have been reported, these include a scheduling system for an aircraft fleet [M+92], and another one for scheduling of Ship Hull production [G+92].

The ideas used in the CarShop scheduler are fairly generic and can be used to design schedulers in other domains.

2 CarShop Scheduling
The CarShop Scheduler requires to schedule a set of Car repair jobs $J = \{J_1, \ldots, J_n\}$, given a set of physical resources $R = \{R_1, \ldots, R_m\}$ and a set of constraints $C = \{C_1, C_2, \ldots, C_r\}$. Here $n_J$, $n_R$ and $n_C$ are the number of jobs, resources and constraints respectively. Each Job $J_i$ contains a set of tasks; $J_i = \{T_1^i, \ldots, T_{n^i}^i\}$, where $n^i_T$ is the number of tasks in job $J_i$. 

The current Research was supported in part by Cleveland Advanced Manufacturing Program (CAMP) grant 342-3914.
The scheduler needs to find values for variables which are defined below:

**Task Start Time:** The scheduler has to instantiate the starting time of each of the tasks \( S_i \) (1 \( \leq i \leq n_J \), 1 \( \leq k \leq n^J_k \)).

**Resource Allocation:** The scheduler has to allocate for each of the tasks \( T_k \) of a job \( J_i \), a resource set \( R_{ik} \) (1 \( \leq i \leq n_J \), 1 \( \leq k \leq n^J_k \), 1 \( \leq t \leq \text{END-TIME} \)), which indicates the set of resources assigned to task \( k \) of job \( i \) starting at time \( t \), given that the entire schedule lasts between the time 1 and END-TIME.

### 2.1 Scheduling Objectives

The problem has multiple objectives, the primary objective is to schedule all the jobs within temporal and operator availability constraints such that the schedule completes within the prescribed time. The secondary criterion may be any of the following:

- Minimize the maximum CompletionTime
- Minimize the cost of allocation.

The objective for obtaining any schedule \( S \) can also be restated as an Operations Research Formulation.

### 2.2 Schedule Data

The input to the CarShop scheduling problems consists of static and dynamic information. The static information provides data about the resource requirements of each task, the performance characteristics of resources in terms of time taken by different operators and cost of operator allocation. The dynamic data consists of task input information in terms of a list of cars to be scheduled along with tasks to be performed in each of them, and information about resource availability in terms of availability periods of different operators. The constraints in the system can be dynamically stated with the problem, or stored as predicates in the form of static data. The priority of tasks indicate a preference ordering while dispatching them during the scheduling process. Some sample CarShop information is provided in the table 1.0 below.

<table>
<thead>
<tr>
<th>Jobs/Tasks:</th>
<th>Car</th>
<th>Tasks</th>
<th>Priority</th>
<th>Arr/Time</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>car1</td>
<td>fix transmission</td>
<td>4</td>
<td>0</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>car2</td>
<td>tuneup</td>
<td>2</td>
<td>0</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>car3</td>
<td>fix brakes</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>car3</td>
<td>fix transmission</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>car3</td>
<td>fix gasket</td>
<td>3</td>
<td>5</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

### 2.3 Constraints:

**Temporal Constraints:** All jobs in the schedule workshop must be started and completed between Start/End time slots. Furthermore a task must be completed before its due time if provided as input. Precedence Relations are other temporal constraints in the system which force certain tasks to be scheduled before others.

**Resource Constraints:** Resource constraints curtail the allocation of resource combinations to tasks. The first type of resource constraints forbid an operator from working on certain tasks while another resource constraint forbids certain set of operators from working together.

### 3 Schedule Components

The CarShop scheduler system can be functionally decomposed as shown in figure 1.0.
The CarShop scheduling problem is decomposed into tasks, resources, and constraints; these are fed into the Schedule Pre-analyzer, which checks the partial feasibility of the system. The core scheduling engine consists of a task dispatcher, resource allocator and constraint propagator which generate the schedule. The control of the scheduling mechanism is provided by the heuristic selector, which guides the schedule by selecting the task selection and operator selection heuristics.

The Task Dispatcher dispatches different tasks to be scheduled in the system, the order of dispatch is determined by the task ordering heuristics. The Resource Allocator allocates resources to each schedulable task and is guided by resource selection heuristics to determine operator selection for each task. The resource allocator also determines the start time of each task, as it depends both on the temporal constraints on the task, and the periods of availability of the operator.

The Constraint Propagator propagates the temporal and resource constraints in the system.

The Heuristics Selector provides flow of control information to the Task Allocator and Resource Dispatcher units. Each task ordering heuristics in the current system assigns dispatch weights to different tasks, while each resource ordering heuristics provides a way of determining the most appropriate resource to be allocated to a schedulable task.

The CarShop scheduling algorithm is now presented.

### 4 Scheduling Algorithm:

**Input:** Set TaskList \( T \), AvailableResources \( R \), constraints \( C \), earliest start time \( EST_j \) of each task \( T_j \), Heuristic Weights, Heuristics, MaxIterationsToTry \( N \).

**Output:** "Good" Feasible Schedule \( S \).

**S1: Task Selection:**
1. If (\( T \) is empty) goto S5. /*end of current schedule*/
2. Select task \( T_j \) in \( T \) within current Dispatch contention window (DCW) i.e. \( DW_1 \leq EST_j \leq DW_2 \).

   where \( T_j \) has maximum heuristic dispatch weight in the DCW determined by Task Ordering heuristic.

**S2: Resource Selection:**
1. Select a resource \( R_j \) \( \in R \) based on Resource Selection heuristic subject to resource constraints.
2. If no resource found goto S6.

**S3: Allocation and Updating:**
1. Assign \( R_j \) to \( T_j \).
2. Calculate Finish Time \( FT_j \) remove \( T_j \) from \( T \).
3. Calculate new Availability hours of resource \( R_j \).
4. If \( (R_j, \text{Available Hours} == \text{Nil}) \) then
   Remove Resource \( R_j \) from \( R \).

**S4: Constraint Propagation:**
1. For all partially ordered tasks \( T_k \) such that \( \text{precedes}(T_j, T_k) \) set \( EST_k = FT_j \).
2. Set \( EST_j = FT_j \) for all tasks with known EST's. /*Tasks with unscheduled preceding tasks have uninstantiated EST's */
3. Update the availability of resource \( R_j \) from step S3.
4. Calculate new dispatch window boundaries \( DW_1 \) and \( DW_2 \) and goto S1.

**S5: Calculations:**
1. Calculate cost of schedule.
2. If current schedule \( S \) is cheaper than the previous best schedule then store \( S \).
3. If (#ScheduleIterations > N) goto S9.

**S6: Preferential Dispatch Weight Adjustment:**
1. Preferentially increase heuristic weights of all current unscheduled tasks \( T_j \) in \( T \).
2. Preferentially increase heuristic weights of all \( T_k \) where \( \text{precedes}(T_j, T_k) \) and \( T_j \in T \)
3. goto S8.

**S7: Dispatch Weight Adjustment:**
1. Update Heuristic Dispatch Weight of all \( T_j \in \text{original tasklist} \) by multiplying original heuristic dispatch weight with a random factor.

**S8: Re-Initialisation:**
1. Reset tasklist \( T \) with original tasks having updated weights.
2. Calculate initial dispatch window boundaries.
3. Increment #ScheduleIterations by 1.
4. goto S1.

**S9: Schedule Output:**
1. Print best solution \( S \) generated.
2. Stop.
5 Schedule Iteration

A brief explanation of the various steps in the scheduling algorithm is now presented. The main features of a typical iteration are illustrated in figure 2.0 by considering an example. Figure 2.0a indicates the current operator availability. Task selection heuristic decides the next task to be scheduled e.g. $T_3$, and Operator selection heuristics determines the most suited operator for the selected task e.g. $O_{p1}$. Allocation and updating of resources are shown in the figure 2.0c. Once the allocation has been done and the partial schedule updated, a new set of tasks contend to be dispatched.

5.1 Task Selection

In each iteration tasks are chosen from the set of *Earliest Schedulable Tasks*, the starting times of these tasks fall within the dispatch contention window shown by the shaded region in figures 2.0b,d. The task ordering heuristic associated a dispatch weight $w_f$ for each task $T_f$ and the task with the highest dispatch weight is chosen for dispatch.

**Dispatch Contention Window**: The *Dispatch contention window* determines the task which contend to be dispatched. Its boundaries $DW_1$ and $DW_2$ mark a region on the time line and tasks whose earliest start time (EST's) lie within this region are dispatchable in the current iteration.

On applying the task selection heuristics, if for example task $T_1$ is selected to be dispatched, then during the next iteration new boundaries of the dispatch contention are determined by Task $T_2$ which has the next least EST. The window now starts from EST of $T_2$= 3 to 8. New task $T_5$ comes into the dispatch contention window and now tasks $T_2, T_3, T_4, T_5$ contend for dispatch.

If the a dispatch contention window of size 0 is used in the system, as shown in figure 3.0c; then the task with the least EST is dispatched. Dispatch contention may still need to be resolved between tasks having the same lowest EST's, i.e. between tasks $T_1$ and $T_2$ as shown in the figure 3.0c.

5.2 Constraint Propagation

Partial temporal ordering between different task are maintained in the system and Earliest Start Times of tasks are updated by the constraint propagator after each iteration. Due to precedence constraints certain tasks which were previously unschedulable, could also become schedulable and are placed on the time line by the constraint propagator. Since tasks in the same job...
are mutually exclusive, the constraint propagator also adjusts the earliest start times of other unscheduled tasks of the job, one of whose tasks is scheduled during the current iteration. Figure 4.0 illustrates how earliest schedulable times of tasks of the same job are maintained and updated.

![Figure 4.0: Temporal Constraint Propagation](image)

Tasks $T_1^1, T_2^1, T_3^1$ can all start at the same time, task $T_2^1$ with a time duration of 4 is chosen from amongst the schedulable tasks. By temporal constraint propagation the earliest starting times of $T_1^4, T_3^4$ now become 6, the EST of task $T_1^4$ which was uninstantiated before (assuming a precedence relationship between $T_1^4$ and $T_2^1$), is now determined as 6; as all its preceding tasks have been scheduled.

### 5.3 Dispatch Weight Adjustments

Every task selection heuristic associates a heuristic weight with each task. If the task has a priority greater than 1, then the final dispatch weight of task $T_i$ is calculated as the product of the heuristic weight and the priority.

Once a schedule has been generated, the next schedule is generated by changing the dispatch weight of each task and repeating the scheduling procedure again. This is done in step $S7$ of the algorithm. For each task, a random number ranging from 1 to maximum heuristic weight of the chosen heuristic is generated and multiplied with the original dispatch weight to give the new dispatch weight.

If the schedule iteration cannot be extended in step $S8$ then the dispatch weights of the unscheduled tasks are preferentially increased in step $S6$ so that they may be scheduled earlier in the next iteration. The dispatch weights of tasks which must precede the unscheduled tasks due to precedence constraints, are also preferentially increased. This helps in getting them scheduled earlier and hopefully pushing their succeeding tasks forward in the time line.

### 6 Scheduling Heuristics

This section describes some of the heuristics used in the scheduling process. The use of heuristics in search during scheduling, has been proposed at several other instances [FSB89, Gra86]. The heuristic weights associated with each task selection heuristic are based on the average time taken for the task to be completed. There are no heuristic weights associated with operators who are ordered by operator selection heuristics depending on the task to be performed or their present availability.

#### 6.1 Heuristics for task Selection

The following are some heuristics used to obtain task dispatch ordering amongst tasks:

**6.1.1 Highest Priority First (hpf)**

This heuristic dispatches tasks purely based on their priorities, tasks with higher priorities are assigned higher weights.

**6.1.2 Weighted Shortest task First (wstf)**

This heuristic uses the average time duration of a task as the heuristic parameter in order to obtain a rating. Tasks which can be completed in a shorter time are given higher weights.

**6.1.3 Weighted Longest task First (wltf)**

The heuristic parameter used here is again the task duration. In this the task which takes the longest time is given preference over all other tasks and attempted to be scheduled earlier. Longer tasks are considered more difficult because they require resources for a longer time. This heuristic is used with the intention of trying to schedule the more difficult tasks first, and doing the easier tasks later.

#### 6.2 Operator Selection Heuristics

Heuristics are required to allocate operators to perform the selected task. Some heuristics used to select operator orderings are provided below.

**6.2.1 Minimizing Operator WorkLoad (min-ld)**

This heuristics guides operator allocation by trying to maintain minimum operator workload amongst all operators. The number of time slots of work put in by each operator is maintained. All the operators are sorted increasing in the number of already assigned hours. Ties between operators are resolved by picking up the operator with a lower index.

**6.2.2 Schedule Earliest Finishing Operator/Operators First (ef-time)**

This heuristics guides the search by choosing the available operator/operators who can finish the task in the shortest time. If there is a tie between two operators or operator combination then the operators with a lower workload are selected.
6.2.3 User Preference Allocation

The user preference allocation strategy tries to allocate resources to tasks guided by user preference. For each task a list of ordered operators is given, indicating the preference of the user in allocating operators to the given task. The schedule produced may not be good with respect to either time or makespan; as allocation is not a function of the current work load of the operators.

7 Empirical Results

The CarShop scheduler was implemented in Prolog, no assert, retracts were used although cuts were used purely for efficiency reasons. The scheduling problem is NP Complete but simulation results indicate that for a real CarShop with about 8-10 workers and 30-35 jobs the scheduler is able to give good schedules as solutions. The emphasis in the system is on providing a flexible AI problem representation and also collecting some Empirical results on the performance of different heuristics in the system.

Presenting a formal analysis of the scheduler performance is difficult because the solution method is empirical, and scheduling results were quite dependent on task requirements and operator availability. The presence of constraints simplifies the problem as the number of solutions become fewer, but the solutions produced by the different heuristics become less different. Hence an example with many constraints was not chosen to illustrate performance measures of different heuristics. Backtracking in the system however increases with an increase in the number of constraints.

A small problem with an output is presented below. The best schedule with respect to both - cost of the schedule and makespan was obtained by choosing the weighted shortest task heuristic and assigning the earliest finishing operators to each task. The results are followed by some statistics about results obtained by using different heuristics on the problem.

Cost and Makespan: Schedules optimized with respect to cost, also turned out to be optimized with respect to the makespan, in cases where hours of availability of different operators did not vary by more than 30%. For schedule shown above the cost was 59, while the worst reported schedule had a cost of 73 units, and a makespan of 16 time units. The schedules which allocated operators based on earliest finish time operators minimized the makespan.

Dispatch Contention Window Size: An optimal dispatch contention window is needed to minimize the overall cost of the schedule or the makespan. If the window size is 0, then dispatch contention occurs only between few tasks - all having the same earliest start times. With larger dispatch contention window size the backtracking in the system comes down and a better quality schedule resulted in a number of test cases, Dispatch window size of about 30-35% of the entire scheduling
span produced better schedules with lower overall costs, makespan and reduced amount of backtracking, however in certain cases as the example problem a size less than 30% produced the best solution. The best solution for the problem previously described was found for DCWS=3, and had a cost of 59, with a makespan of 12. The average flowtime was 7.2, with maximum and average operator loads being 6 and 4.6 respectively.

Task Selection Heuristics: When a large number of schedules are possible, weighted shortest time task first (wstf) performed better than only highest priority first (hpf) or choosing the weighted longest task first (wltf). When the number of solutions are not very large then scheduling weighted longest time task first showed significantly less number of backtracking. The weighted shortest task first heuristic in combination with choosing the earliest finishing operator heuristic produced a lower flow time, a lower maximum task load and lower average task load on operators, for a large number of problems.

Operator Selection Heuristics: Choosing the Earliest Finishing operator first (ef.time), takes more execution time but yields the solutions with lower costs and makespan. It has to evaluate the finish time of the tasks given all feasible operator combinations. Choosing operators with minimum load (min.ld) leads to more backtracking in the system compared to the previous heuristic.

Some sample data for the example problem:

<table>
<thead>
<tr>
<th>DCWS=0</th>
<th>Task</th>
<th>ReqHdr</th>
<th>Cat</th>
<th>MksSpan</th>
<th>Avg PT</th>
<th>Max Ld</th>
<th>Avg Ld</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpf</td>
<td>min ld</td>
<td>73</td>
<td>16</td>
<td>8.2</td>
<td>10</td>
<td>8</td>
<td>4.8</td>
</tr>
<tr>
<td>hpf</td>
<td>eT</td>
<td>61</td>
<td>13</td>
<td>8.2</td>
<td>10</td>
<td>8</td>
<td>4.8</td>
</tr>
<tr>
<td>wstf</td>
<td>eT</td>
<td>63</td>
<td>13</td>
<td>7.4</td>
<td>10</td>
<td>6</td>
<td>4.8</td>
</tr>
<tr>
<td>wstf</td>
<td>min ld</td>
<td>67</td>
<td>15</td>
<td>9.0</td>
<td>10</td>
<td>6</td>
<td>4.8</td>
</tr>
</tbody>
</table>

| DCWS=3 |
|--------|--------|-----|---------|--------|--------|--------|
| hpf    | min ld | 66   | 14   | 8      | 8      | 4.8    |
| hpf    | eT    | 61   | 12   | 8.2    | 10     | 6      | 4.8    |
| wstf   | min ld | 68   | 16   | 7.4    | 9      | 6      | 4.6    |
| wstf   | eT    | 64   | 13   | 8.2    | 8      | 4.6    |

| DCWS=5 |
|--------|--------|-----|---------|--------|--------|--------|
| hpf    | min ld | 66   | 14   | 8      | 8      | 4.8    |
| hpf    | eT    | 61   | 12   | 8.2    | 10     | 6      | 4.8    |
| wstf   | min ld | 68   | 16   | 7.4    | 9      | 6      | 4.6    |
| wstf   | eT    | 64   | 13   | 8.2    | 8      | 4.6    |

Best Solutions if All operator are available from (1,16):

| DCWS=7 |
|--------|--------|-----|---------|--------|--------|--------|
| hpf    | eT    | 61   | 12   | 8.2    | 8      | 4.8    |
| wstf   | min ld | 68   | 16   | 7.4    | 9      | 6      | 4.6    |
| wstf   | eT    | 64   | 13   | 8.2    | 8      | 4.6    |

Since there are more solutions possible now, longest task heuristic did not yield a better solution. The dispatch contention window size of about 30% of the entire schedule span produced the best solution. The backtracking significantly reduced from the previous case for all heuristic combinations due to increased availability of operators.

Good schedules for problems depend on a host of factors, it is felt that a dispatch contention window size of about 25-30% of the schedule span, choosing weighted shortest task first and allocating earliest finishing operators to tasks yielded better solutions on a large range of problems.

8 Summary

In the previous sections the design and implementation of an AI application - "A Heuristic Based CarShop Scheduling Application" has been discussed. Each schedule consists of a series of resource allocations to tasks, and the objective of the schedule is to try and make "good" assignments of resources to all tasks. Different heuristics for task dispatch and operator allocation were discussed. The scheduler is a generate and test scheduler which tries to find a good schedule with respect to cost of the schedule or makespan of the schedule depending on the secondary objective criterion chosen by the user. Issues in backtracking over schedules and partial schedules were discussed and intelligent backtracking was achieved by preferentially increasing the weights of unscheduled tasks and tasks which precede them due to precedence constraint. Several schedule iterations are tried and the best solution is finally reported to the user, the strate-
gies help in generating only potentially good and feasible solutions. Prolog is a good language for developing schedulers and scheduler prototypes due to the enormous flexibility in the language. The system is easily extensible, as AI techniques and representation schemes used in developing the application are more general and flexible when compared to traditional Operations Research methods.

9 Acknowledgements

We are grateful to Leon Sterling at Case Western Reserve University, for many useful discussions of this work. Thanks also to Umit Yalçınalp at BP Research Warrensville OH for helping to clarify the problem and providing suggestions during the work.

References


