Constraint-based Vehicle Assembly Line Sequencing

Michael E. Bergen[1], Peter van Beek[1], and Tom Carchrae[2]

[1] Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{bergen,vanbeek}@cs.ualberta.ca
[2] TigrSoft Inc., Edmonton, Alberta, Canada T5J 3G2
carchrae@tigrsoft.com

Abstract In this paper, we consider the optimal sequencing of vehicles along multiple assembly lines. We present a constraint-based model of the problem with hard and soft constraints. An advantage of a constraint-based approach is that the model is declarative and there is a separation between the model and an algorithm for solving the model. As a result, once the model had been defined, we could experiment with different algorithms for solving the model, with few or no changes to the model itself. We present three approximation algorithms for solving the model—a local search algorithm, a backtracking algorithm with a constraint relaxation and restart scheme, and a branch and bound algorithm—and we compare the quality of the solutions and the computational performance of these methods on six real-world problem instances. For our best method, a branch and bound algorithm with a decomposition into smaller sub-problems, we obtained improvements ranging between 2% and 16% over an existing system based on greedy search.

1 Introduction

The vehicle assembly line sequencing problem is to determine the order in which a given list of vehicles should be produced on one or more assembly lines subject to a set of constraints. Determining a good sequence is important as the sequence chosen affects the cost of production, the quality of the vehicles produced, and even employee satisfaction.

The particular problem that we study comes from a North American manufacturing plant that produces approximately 36,000 vehicles in a month on two assembly lines and the sequencing is done once per month. A system developed by TigrSoft Inc., an Edmonton company that specializes in planning and scheduling software, currently schedules the production of vehicles. While our motivating application is quite specific, the constraints which define the quality of a sequence are shared with other sequencing and scheduling problems. For example, one important constraint on an acceptable sequence in our problem is that each day a worker on an assembly line should see as much diversity as the current orders permit, including making economy and luxury models, four and five door models, and so on. This "distribution" constraint allows the assembly line workers to maintain their skill set as well as ensuring that at least a certain amount of every order is produced prior to any unexpected line shutdowns. Similar distribution constraints arise in diverse scheduling problems from scheduling sports teams, where the issue of a fair distribution of rest and travel days arises, to other manufacturing problems where a robust schedule is desired in the face of possible machine breakdowns. A second example of important constraints on acceptable sequences are "change-over" constraints that prohibit undersirable transitions such as sequencing white vehicles immediately after red vehicles (the vehicles could come out an undesirable pink colour). Similar change-over constraints also arise in other manufacturing scheduling problems.

In this paper, we describe how we modeled and solved this real-world vehicle assembly line sequencing problem using a constraint-based approach. In a constraint-based approach to problem solving, a problem is modeled by specifying constraints on an acceptable solution, where a constraint is simply a relation among several unknowns or variables, each taking a value from a given domain (see [4] for an introduction). Our model contains both hard constraints (must be satisfied) and soft constraints (can be violated at a cost). Each soft constraint is associated with a penalty value that is incurred every time it is violated. Thus the problem is one of optimization on these penalty values.

An advantage of a constraint-based approach is that the model is declarative and there is a separation between the model and an algorithm for solving the model. As a result, once the model had been defined, we could experiment with different search algorithms for solving the model, with few or no changes to the model itself. We present three approximation algorithms for solving the model: a local search algorithm, a backtracking algorithm with a constraint relaxation and restart scheme, and a branch and bound algorithm. We also demonstrate the importance of decomposing the problem into one-day sub-problems. We compare the quality of the solutions and the computational performance of these methods on six real-world problem instances. For our best method, a branch and bound algorithm with a decomposition into smaller sub-problems, we obtained improvements ranging between 2% and 16% over the existing system developed by TigrSoft, which is based on greedy search.

The software we have developed is in a stand-alone prototype form. We are currently integrating our work into TigrSoft's deployed system and hope in the near future to perform user trials in a production environment.

Related work. Assembly lines are process-oriented and are arranged according to the sequence of operations needed to manufacture a product. This is in contrast to job shops which are job-oriented and machines which perform similar operations are spatially grouped together. While there has been an extensive amount of work on job shop scheduling (see [11] for an overview of constraint-based approaches), in spite of its importance, there has been little work reported specifically on the vehicle assembly line sequencing problem in the literature.

Of the work that has been reported, most has focused on the specification of the vehicle assembly line sequencing problem introduced by Parrello *et al.*

[6]. Van Hentenryck *et al.* [10] and Régin and Puget [7] solve this version of the problem using backtracking search with specialized propagators to maintain arc consistency during the search. Local search techniques have also been developed for this version of the problem including a hill-climbing approach [3] and a simulated annealing approach [9]. However, while this specification has demand and capacity constraints, it omits time-window, change-over, and balancing constraints important in our version of the problem.

More directly related is the work done by ILOG on the vehicle sequencing problem for Chrysler. Unfortunately, there is no published information about this research beyond a press release [8] and a set of presentation slides [2]. The problem they address also has distribution and change-over constraints similar the problem addressed in this paper. Their solution decomposes the problem into smaller sub-problems on which it performs backtracking search, attempting to satisfy constraints with the highest priorities first.

2 The Problem Domain

The manufacturing plant that we study produces approximately 36,000 vehicles in a month on two assembly lines and the sequencing is done once per month. The input to the problem is a list of orders (an order is a quantity of identical vehicles) that need to be produced during that month, capacity values that specify how many vehicles can be produced on each day on each assembly line, and the user-specified constraints. As a first step, each order is split into several smaller quantities of vehicles called *lots* such that the size of each lot is less than or equal to 60 vehicles, called the batch size. The lots are then grouped together into *batches* by putting together similar lots with sizes that add up to the batch size (see Table 1(a) for an example). Each batch is assumed to take one hour of time to produce on an assembly line. A typical problem instance has lots with between one and 60 vehicles, and batches with between one and ten lots, with the majority of batches having only one lot. It is important to note that after batching, the lots are not sequenced in a batch and thus sequencing actually occurs at the lot level.

The lots and batches have attributes. Some attributes are common to all problem instances and others are user-definable and thus specific to a problem instance. Common attributes include the assembly lines that a lot can be produced on, the date a lot must be produced after (line-on date), and the date a lot must be produced by (line-off date). User definable attributes are either selected from a set of basic attributes such as vehicle model, exterior colour, type of engine, and type of transmission; or are constructed from these basic attributes by Cartesian-product. A batch's attribute values are taken from the attribute values of its lots. Each attribute has a different method for deriving the batch attribute value from the lot values when the lot values differ.

The capacity values specify the number of batches that can be produced on each assembly line on each day. If no vehicle production is desired on a particular day, then the capacities for that day are zero. The capacities are assigned such that the sum of all the capacities for each day and assembly line equals the total number of batches that need to be produced for the month. Hence, there is no excess capacity. A day's production on an assembly line is sub-divided into consecutive intervals of time called slots which have a fixed start time and a duration of one hour (since each batch is assumed to take one hour of time to produce). In a final sequence, every slot is assigned one and only one unique batch. A typical problem instance consists of two assembly lines each with 20 days of non-zero capacities. Each of these daily capacities is approximately fifteen batches, which gives a total capacity of 600 batches or 36,000 vehicles.

Each problem contains constraints that restricts which sequences are acceptable. Each constraint is over one or more slots, each slot taking a value from the set of all batches. The constraints can be classified as either a batch constraint or a lot constraint. Lot constraints rely on lot attributes, and influence the sequencing of lots and hence the sequencing of batches. Batch constraints rely on batch attributes and influence the sequencing of batches with no concern for the sequencing of lots within a batch. Constraints can also be classified as either soft or hard. A hard constraint cannot be violated, while a soft constraint can be violated but imposes a penalty value for each violation. Each soft constraint has a penalty value that is given as part of the input of the problem; the higher the penalty value, the more undesirable the violation.

There are eight constraint types. Six of the constraint types—the assembly line, line-on and line-off, even distribution, distribution exception, batting order, and all-different—define hard, batch constraints. The remaining two constraint types—the run-length and change-over—define soft, lot constraints. We now describe these constraints types in detail.

Assembly Line. The manufacturing plant contains two assembly lines. Because of unique equipment, some vehicles can only be assembled on one of the lines, while others can be assembled on either line. If a batch contains a lot that can only be assembled on one of the assembly lines, then the batch must be assembled on that assembly line. There is an assembly line constraint over each slot. Since each slot belongs to an assembly line, only batches that can be made on that assembly line can be assigned to the slot.

Line-On and Line-Off. Each vehicle that is ordered must be produced sometime during the month. However, because of part availability or shipping deadlines, some orders have more stringent scheduling requirements. For this reason, each lot has a line-on and line-off day. A lot must be produced on or after its line-on day, and on or before its line-off day. A batch's line-on day is the maximum line-on day of its lots and its line-off day is the minimum line-off day of its lots. There is a line-on and line-off constraint over each slot.

Even Distribution. An assembly line should produce a variety of different types of vehicles each day and the production of similar types of vehicles should be spread evenly over the month. Reasons for this include maintaining workers skills for making all types of vehicles, part availability, and producing certain amounts of each type of vehicle prior to any unexpected assembly line shutdown.

Table 1. (a) Example lots and their attributes. Lots are grouped together into batches of size 60. The attributes of a single-lot batch are the same as those of its lot. The derived attributes of the multi-lot batches are shown underlined. (b) Example even distribution values. (c) One possible sequencing of the batches and lots over two days.

(a)

Lot	Batch	Lot Size	Line On	Line Off	Model	Exterior Colour	Sun Roof
L01	B01	60	1	2	M1	Blue	Y
L02	B02	20	1	2	M1	Red	Y
L03	B02	40	$\underline{1}$	$\underline{1}$	$\underline{\text{M1}}$	$\underline{\text{Red}}$	$\underline{\text{N}}$
L04	B03	10	1	2	M2	Green	Y
L05	B03	20	$\underline{2}$	$\underline{2}$	$\underline{\text{M2}}$	Red	N
L06	B03	30	1	2	M2	$\underline{\text{Blue}}$	$\underline{\text{Y}}$
L07	B04	10	1	2	M3	Red	N
L08	B04	10	$\underline{1}$	$\underline{2}$	$\underline{\text{M3}}$	Green	Y
L09	B04	10	1	2	M3	Red	Y
L10	B04	30	1	2	M3	$\underline{\text{Green}}$	$\underline{\text{N}}$
L11	B05	60	1	2	M1	Green	N
L12	B06	60	1	2	M1	Blue	Y
L13	B07	60	2	2	M1	Blue	Y
L14	B08	60	1	1	M1	Blue	N
L15	B09	60	2	2	M1	Green	N
L16	B10	60	1	2	M2	Red	Y
L17	B11	60	1	1	M2	Red	Y
L18	B12	60	1	2	M2	Green	N
L19	B13	60	1	2	M3	Red	N
L20	B14	60	1	2	M3	Green	Y

(b)

Attribute	Day 1	Day 2
M1-Y	2	1
M1-N	2	2
M2-Y	1	2
M2-N	1	0
M3-Y	0	1
M3-N	1	1

(c)

Day	Slot	Batch	Lots
1	1	B06	L12
	2	B08	L14
	3	B01	L01
	4	B02	L02, L03
	5	B11	L17
	6	B12	L18
	7	B04	L08, L07, L09, L10
2	1	B05	L11
	2	B09	L15
	3	B07	L13
	4	B10	L16
	5	B03	L05, L04, L06
	6	B14	L20
	7	B13	L19

The even distribution constraint spreads the batches by specifying the number of batches with a particular attribute value that must be produced on each day. There is an even distribution constraint for each production day and the constraint is over all of the slots that belong to that day.

Distribution Exception. Sometimes an even distribution is inappropriate. For example, when a new model year is introduced, production teams need time to learn new procedures and the distribution of new models should be restricted so that fewer are produced early in the month. To do this, a distribution exception constraint specifies a minimum and maximum number of batches with a particular attribute value that can be produced on each day during a specified period of days in the month. There is a distribution exception constraint for each production day and the constraint is over all of the slots that belong to that day.

Batting Order. Each day, a similar sequencing pattern should be followed on each assembly line. One reason for this is to sequence simple vehicles at the beginning of the day and gradually progress to more difficult vehicles. This allows the production teams to warm up before building more complicated vehicles. To do this, batting order constraints are defined on user-specified attributes and on user-specified orderings of those attributes' values. Specifically, on each day, a batch must be produced before another batch if its attribute value is ordered before the attribute value of the other batch. There is a batting order constraint between each pair of consecutive slots that are on the same day.

All-Different. A constraint is needed to ensure that every batch appears exactly once in any sequence. The all-different constraint is defined over all the slots.

Run-Length. Each day, it is desirable that certain attribute values are not repeated too often. Avoiding monotony of an attribute value can improve the effectiveness of production and quality inspection teams, and avoid part supply problems. A run-length constraint is a soft constraint that incurs a penalty whenever the number of consecutive vehicles with a particular attribute value exceeds a specified limit called the run-length. The run-length constraint is applied to consecutive slots. One penalty value is counted for each lot that exceeds the run-length value. Typical instances have around five different run-length constraints defined and the penalty values for these constraints range between ten and 300.

Change-Over. In a sequence, transitions from one lot attribute value to another lot attribute value may be undesirable. For instance, painting a white vehicle immediately after a red one is undesirable because the vehicle may turn out pink. A change-over constraint is a soft constraint that incurs a penalty value whenever an undesirable transition occurs. The change-over constraint is applied to consecutive slots. It relies on two user-specified attributes, called the former and the latter attributes, to evaluate a transition between two sequenced lots. Typical instances have around forty different change-over constraints defined and the penalty values for these constraints range between one and 100.

A solution to the vehicle assembly line sequencing problem consists of an assignment of batches to slots and a sequencing of the lots within batches such that all the hard constraints are satisfied. The quality of a solution is measured by the total penalty values that are incurred by violations of the soft constraints. The lower the total penalty values, the higher the quality of the solution.

Example 1. Table 1(a) shows an example set of lots and their grouping into batches. The batches are to be sequenced on one assembly line over two days, where each day has a capacity of seven batches. Suppose we define the following constraints. An even distribution constraint is defined on the Cartesian-product of the model and sun-roof attributes and the distribution values are as listed in Table 1(b). To illustrate, there are three batches with attribute values Model "M1" and Sun-roof "Yes" and the distribution values specify that two of these batches must be sequenced on the first day and one batch must be sequenced on the second day. A distribution exception constraint is defined on the Exterior Colour attribute value "Green" for the first of the two days with a minimum value of one batch and a maximum value of two batches. A batting order constraint is defined on the attribute Model specifying that on each day, M1 batches should be produced first, followed by M2 batches, and then M3 batches. A run-length constraint is defined on the Exterior Colour attribute value "Red" with a run-length value of 120 vehicles and a penalty value of 200. Thus, sequencing lots L16, L17, and L19 consecutively would incur a penalty value of 200. A change-over constraint is defined on the Exterior Colour attribute with a penalty value of 100. The former attribute value is "Red" and the latter attribute value is "NOT Red", where "NOT Red" means any colour except "Red". Thus, sequencing lot L17 followed by L18 would incur a penalty value of 100. Table 1(c) gives one possible sequencing of the batches and lots. The change-over constraint is violated three times (L17 $\rightarrow$ L18, L09 $\rightarrow$ L10, and L05 $\rightarrow$ L04) and the run-length constraint is not violated at all for a total penalty value of 300.

3 Solution Techniques

Since the problem is large, we solved the constraint-based model approximately rather than optimally. We describe three algorithms for solving the vehicle sequencing problem: a local search method, a backtracking method, and a branch and bound method. All of the algorithms used the following two techniques for simplifying the problem. First, the overall problem was split into equal sized sub-problems by placing, for a particular assembly line, a specified number of consecutive production days in each sub-problem. To determine which batches should go with which sub-problem, we used the solution found by the greedy search algorithm and assigned a batch to a sub-problem if its placement within the solution fell on one of those days. The sub-problems were then solved in order of the days they contain. Since soft constraint violations can occur between sub-problems, after a sub-problem is solved, the batch that was sequenced last is added to the beginning of the next sub-problem. Second, the sequencing of batches and the sequencing of the lots within batches were decoupled and done in stages rather than simultaneously. In stage one, the lots within each batch were sequenced without consideration of the other batches and then in stage two, the batches were sequenced with the lots considered fixed. The sequencing of the lots was done either by using the solution provided by the greedy search algorithm, or by optimizing the lot sequence according to the soft constraints using a simple generate and test procedure.

Local search. Local search is a general approach to solving combinatorial optimization problems (see [1] for an overview). To apply local search to the vehicle assembly line sequencing problem, we need to define a cost function and a neighborhood function. The cost function takes as its input a solution to the hard constraints and returns the total number of penalty values incurred by the soft constraints. Thus, all the soft constraints are moved into or are represented by the cost function. There are many possible ways to define a neighborhood function. In general, the way that the neighborhood function is defined influences the quality of the solutions that a local search algorithm finds and the cost of searching the solution space. In our experiments, we define the neighborhood of a solution to consist of any solution where two variables' values have been swapped and no hard constraint is violated.

The local search algorithm we devised is a simple hill-climbing algorithm. Our algorithm begins with an initial solution that satisfies all of the hard constraints. The default initial solution is the solution provided by the greedy search algorithm. Of the solutions in the neighborhood, the solution that reduces the total penalty value the most is selected. This process is repeated until no solution can be found in the current neighborhood that improves on the quality of the current solution.

Backtracking with relaxation and restart. Standard backtracking requires the satisfaction of all constraints. However, in a problem that contains soft constraints, it is common that some of the soft constraints are not satisfied. Two modifications to make backtracking applicable are possible (see [12] for an overview). The optimistic approach first searches for a solution satisfying all of the constraint and then iteratively removes or relaxes constraints—the weakest first—until a solution is found. The pessimistic approach first searches for a solution satisfying the strongest constraints and then iteratively adds more constraints—the strongest first—until no solution is found. We chose to pursue an optimistic or relaxation approach.

For our relaxation approach, each soft constraint's instances that belong to the same day and assembly line are grouped together into a parameterized hard constraint. Since soft constraint violations can occur between lots that are sequenced on different days, the last slot of the previous day is included in each of these parameterized constraints. Let p represent the parameter for an instance of a parameterized constraint. For a run-length constraint p represents the maximum run-length that can occur on the day. For a change-over constraint p represents the maximum number of change-over violations that can occur. If more than p violations occur, then the parameterized change-over constraint is not satisfied. This method has advantages over simply removing selected soft constraints from the problem as it decreases the number of possible selections that need to be made and leaves more decision power to the search algorithm.

The backtracking algorithm begins with each parameterized change-over constraint initialized with a value of zero and each parameterized run-length constraint initialized with the run-length value of the constraint. As the backtracking algorithm attempts to solve the problem a count is kept of how many times each parameterized constraint fails. Associated with each parameterized constraint is a failure limit that is proportional to its penalty value. If any parameterized constraint fails more often than its failure limit the search stops. If the search stopped without finding a solution, a parameterized constraint is chosen to be relaxed by selecting a constraint with the smallest penalty value that failed at least once. The chosen constraint is relaxed by adding a value to its parameter. For a change-over constraint, its parameter is incremented by one. For a run-length constraint, its parameter is incremented by the batch size, increasing the run-length by sixty vehicles. The backtracking algorithm is then restarted, and the relaxation and restart processes is continued until a solution is found.

The efficiency of the backtracking algorithm was improved by using variable and value ordering heuristics and by reducing the search space by constraint propagation (see [4] for an overview). The variable ordering selects the variable belonging to the earliest day with ties broken by smallest domain size. The value ordering is based on the greedy search solution. For each variable, the value assigned in the greedy search solution is placed first in the variable's domain. To achieve a high level of propagation with limited computation, specialized propagators, which take advantage of the constraint's structure, were devised for the all-different constraint and the distribution constraints.

Branch and bound. To apply branch and bound search to the vehicle assembly line sequencing problem, we need to define a cost function and a function to provide a lower bound on the cost of any partial solution (see [5] for an overview). As with the local search approach, the cost function takes as its input a solution to the hard constraints and returns the total number of penalty values incurred by the soft constraints. The lower bound function takes as its input a partial solution to the hard constraints and returns the total number of penalty values incurred by the batches that have been sequenced so far.

The algorithm begins with an upper bound on the cost of an optimal solution and tightens the bound until no solution is found. The upper bound is initialized to be the cost of the solution returned by the greedy search algorithm. After backtracking finds a solution, we take the total penalty value for the solution, reduce it by the smallest constraint penalty value in the problem instance (a value of one for the problem instances we examine), and set this as the new bound value. The branch and bound algorithm then continues, and backtracks whenever the lower bound on the cost of a partial solution exceeds the current bound. If it finds a solution with the current bound value, we reduce the bound value again. This process is continued until no solution can be found. In this case, the last solution found is an optimal solution.

For the branch and bound algorithm, the variable ordering was fixed to be the ordering of the slots in time. This was chosen to simplify the way the lower bound function was implemented. The value ordering and constraint propagation techniques were the same as described for the relaxation approach.

4 Evaluation

In this section, we present the results of applying the three solution methods to six real-world problem instances. Each problem instance represents a month's worth of orders for a vehicle manufacturing plant with two assembly lines. We use the quality of the solutions produced by the existing system developed by Tigr-Soft as our base of comparison. The existing system, which is based on greedy search, took about 15 seconds to solve each of the instances (all experiments were run on 450 MHz Pentium III's with 256 Megabytes of memory).

Table 2 summarizes the results for the three methods. In all of the reported results, each problem instance was divided into one day sub-problems. We also examined the effect of dividing into two and three day sub-problems and found

Table 2. (a) Total penalties and (b) percentage improvement over greedy search of hill climbing methods, backtracking methods, and branch and bound methods with a decomposition into sub-problems of a single day. For branch and bound a time limit per sub-problem of either 2 hours or 1 minute was used.

(a)

	Greedy	Hill climbing			Backtracking			Branch and bound			
								2 hours		1 minute	
#	GS	HC	HC-R	HC-O	RR	RR-N	RR-O	BB	BB-O	BB	BB-O
1	8,018	7,036	7,623	7,055	7,337	8,718	7,721	7,002	6,925	7,002	6,925
2	5,042	4,579	5,059	4,470	4,640	5,353	4,549	4,528	4,403	4,528	4,399
3	3,412	3,347	3,543	3,409	3,357	3,441	3,443	3,306	3,348	3,306	3,348
4	2,498	2,233	2,269	2,174	2,308	2,532	2,265	2,206	2,137	2,218	2,145
5	2,956	2,885	2,996	2,605	2,883	3,069	2,618	2,762	2,479	2,762	2,485
6	2,818	2,560	2,605	2,577	2,602	3,028	2,625	2,489	2,500	2,489	2,500

(b)

	Hill climbing			Backtracking			Branch and bound			
							2 hours		1 minute	
#	HC	HC-R	HC-O	RR	RR-N	RR-O	BB	BB-O	BB	BB-O
1	12	5	12	8	−9	4	13	14	13	14
2	9	0	11	8	−6	10	10	13	10	13
3	2	−4	0	2	−1	−1	3	2	3	2
4	11	9	13	8	−1	9	12	14	11	14
5	2	−1	12	2	−4	11	7	16	7	16
6	9	8	9	8	−7	7	12	11	12	11

that for each of the three methods, the CPU time increased (sometimes dramatically) but the quality of the solutions did not change significantly. It appears that the even distribution constraint significantly reduces the possibility of improving the solution by solving multiple days at a time. For all of the problem instances examined, the even distribution constraint was defined on an attribute that contained more than 200 attribute values and many attribute values only had one or two batches associated with them. Since the even distribution constraint defines for each day and attribute value the number of batches with the attribute value that can be assigned to the day, many days did not share batches. Thus when solving multi-day problems of two or three days, it was unlikely that the days within a sub-problem would share batches.

For the local search algorithms, the sequencing of lots within a batch was fixed to be the sequence of the lots within the solution determined by the greedy search (HC and HC-R) or was fixed to be the optimized sequence of the lots (HC-O). The initial solution given to the hill-climbing algorithm to improve upon was either the sequencing of the batches provided by the greedy search algorithm (HC and HC-O) or a random sequencing of the batches (HC-R). The HC and HC-O hill climbing algorithms took between two and three minutes to solve each of the instances; the HC-R algorithm took on average double the CPU time. We note that when a random initial solution was used, the results were poorer. These results indicate the importance of a good initial solution when using a hill-climbing method on the problem.

For the backtracking algorithms that used a relaxation and restart approach, the failure limits for each soft constraint were set by multiplying each constraint's penalty value by 200 (the value chosen is somewhat arbitrary; we have verified that choosing a different multiplicative value does not materially change the conclusions that we draw from our study). The sequencing of lots within a batch was fixed to be the sequence of the lots within the solution determined by the greedy search (RR and RR-N) or was fixed to be the optimized sequence of the lots (RR-O). The approaches RR and RR-O used a value ordering that was based on the greedy search solution. Each slot's domain values were ordered by placing the batch that was assigned to the slot in the greedy search solution first in the slot's domain. As well, since the choice of which constraint to relax next may not be perfect, after a sub-problem was solved with the backtracking algorithm, the sub-problem solution was compared with the greedy search solution and the sub-problem solution with the lowest total penalty value was selected. The algorithms took between five and fifteen minutes to solve each of the instances. We note that when the value ordering and the best solution selection process was removed (RR-N) the quality of the solutions decreased significantly. The value ordering appears to give the backtracking algorithm a good solution to build on.

For the branch and bound algorithms, the sequencing of lots within a batch was fixed to be the sequence of the lots within the solution determined by the greedy search (BB) or was fixed to be the optimized sequence of the lots (BB-O). Time limits were set on how much CPU time could be spent on each sub-problem. If the algorithm had not completed within the time limit, the best solution found so far was used. We report the results for time limits of two hours and of one minute. When the time limit per sub-problem was two hours, four of the six instances had all of their sub-problem solutions proven optimal. The other two instances had in total only five sub-problems with potentially sub-optimal solutions. These five sub-problem solutions may in fact be optimal, but they were not proven so within the time limit. The total CPU time required to solve an instance when the time limit per sub-problem was two hours varied significantly, ranging between five minutes and fourteen hours. When the time limit per sub-problem was reduced from two hours down to one minute, only one problem instance's total penalty values slightly increased. However, although almost all of the solutions found were of the same quality, few of these solutions were proven optimal within the reduced time limit. On these instances finding an optimal solution to a sub-problem was relatively easy, but proving its optimality was often hard. The total CPU time required to solve an instance when the time limit per sub-problem was one minute varied between five and 25 minutes.

5 Conclusion

We introduced a real-world optimization problem that we modeled and solved using constraint-based approaches. We also demonstrated the importance of decomposing the problem into one-day sub-problems. We argued that because of the tightness of the even distribution constraint such a decomposition had little effect on the quality of the overall solution. For nearly all of these one-day sub-problems, we proved optimal solutions within a reasonable amount of time using the branch and bound technique. In even less time, the branch and bound method was able to find nearly identical results without proving optimality for many sub-problems. The local search method was also able to find relatively good solutions. Given the simplicity of this algorithm, it is likely that even better results could be found with a local search approach. The relaxation approach was the least successful of the three algorithms. Improving this approach is likely possible, but the usefulness of such an improvement is questionable due to the quality of the solutions obtained by the other two simpler algorithms.

In the preliminary stages of this research, we established with TigrSoft the criteria by which our results would be judged a "real-world" success. It was determined that solutions with a 5% reduction in penalty values that could be found in less than 30 minutes would be considered significant. All three algorithms were capable of finding solutions to the six problem instances within 30 minutes. For four of the six problem instances we were able to obtain more than a 5% improvement with any of the three solution methods. For the best method, a branch and bound algorithm with a decomposition into one-day sub-problems and a one minute time limit on each sub-problem, we obtained improvements ranging between 2% and 16% and averaging 11.6% over the existing system.

References

1. E. Aarts and Lenstra J. K., editors, *Local Search in Combinatorial Optimization*, John Wiley & Sons Ltd., 1997.
2. T. Chase *et al.* Centralized vehicle scheduler: An application of constraint technology. http://www.ilog.com/products/optimization/tech/research/cvs.pdf, 1998.
3. A. Davenport and E. Tsang. Solving constraint satisfaction sequencing problems by iterative repair. In *Proc. of PACLP-99*, pages 345–357, 1999.
4. K. Marriott and P. J. Stuckey. *Programming with Constraints.* MIT Press, 1998.
5. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.
6. B. D. Parrello *et al.* Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *J. of Automated Reasoning*, 2:1–42, 1986.
7. J-C. Régin and J-F. Puget. A filtering algorithm for global sequencing constraints. In *Proc. of CP-97*, pages 32–46. Springer-Verlag, 1997.
8. ILOG Press Release. ILOG drives productivity improvements at Chrysler. http://www.ilog.com/success/chrysler/index.cfm, 1997.
9. K. Smith *et al.* Optimal sequencing of car models along an assembly line. In *Proc. 12th Nat'l Australian Society for Operations Research*, pages 580–603, 1993.
10. P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.
11. M. Wallace. Applying constraints for scheduling. In B. Mayoh and J. Penjaam, editors, *Constraint Programming.* Springer-Verlag, 1994.
12. M. Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996.