Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Car Sequencing with respect to Regular Expressions and Linear Bounds

Márton Drótos * **Tamás Kis** **

* *Institute for Computer Science and Control, Budapest, Hungary; e-mail: marton.drotos@sztaki.mta.hu).*
** *Institute for Computer Science and Control, Budapest, Hungary; e-mail: tamas.kis@sztaki.mta.hu).*

Abstract: In this paper we introduce a new model and a computational approach for sequencing assembly lines with two types of constraints: (i) patterns described by regular expressions and (ii) linear bounds on the number of certain products that may occur in pre-specified intervals. If we restrict the problem to the second type of constraints only we obtain a generalization of the familiar *car sequencing problem*, whereas constraints of type (i) may be useful to add extra structure. Constraints of both types may have priorities and can be violated, and a Pareto optimal solution is sought minimizing the violation of constraints in the given priority order. We describe a computational method based on mathematical programming and genetic algorithms for finding suboptimal solutions.

Keywords: Sequences, Multiobjective optimization, Manufacturing systems

1. INTRODUCTION

One way to express the capacity of assembly lines is to impose constraints on the sequence of products of the following form: in every subsequence of length ℓ_i there can be at most q_i products with option o_i (so-called *among constraint*). This is the familiar *Car Sequencing Problem*, see Parello et al. [1986] for the original definition, and Solnon et al. [2007] for an overview of the state-of-the-art methods. The current best exact method was proposed by Fliedner and Boysen [2008]. For a complexity analysis, see Kis [2004].

In this paper we want to present a different model in which instead of the above rules, the structure of sequences is described by regular expressions, such as "every 5 consecutive occurrences of products of type A must be followed by 3 consecutive products of type "B". This rule applies locally, and only if products of type A occur at least 5 times consecutively. In addition we also allow constraints of the following form: the subsequence from position p_1 to position p_2 may contain only (or at least) q products of type B or C. Here, p_1 and p_2 are fixed integers. So, this is a generalization of the car sequencing problem, where we use regular expressions to describe allowable patterns in the sequence. Thus, in contrast to the method of Van Hoeve et al. [2006], instead of using regular expressions to express the "among" constraint, we use them to describe patterns, and at the same time we also use the among constraint to express global properties as indicated above.

⋆ This work has been supported by the NFU grant No. ED_13-2-2013-0002.
⋆⋆The second named author is grateful for the support of the János Bolyai research grant BO/00412/12/3 of the Hungarian Acedemy of Sciences.

With this model, we can express a large variety of constraints, such as, for example:

- the sequence should be built of repeating blocks in the form $A^+ \to B^+ \to C^+$ (regular expression constraint $(A^+B^+C^+)^+$)
- product A should be produced in batches not smaller than 5 units and not larger than 10 units (regular expression constraint $((\neg A)^+A[5-10](\neg A)^+)^+$, where $\neg A$ means any product except A)
- due date: at least 10 units of product A should be produced until position p (linear bound constraint specifying a lower bound)
- capacity: at most 8 units of product A or product B should be produced from position p_1 to position p_2 (linear bound constraint specifying an upper bound)
- material availability: product A requires 4 units of some material m, product B and C requires 6 units of m, the initial amount of m is 15 units, and an additional 30 units will be delivered at position p (linear bound constraint with additional weights, specifying an upper bound)

Another major difference to car sequencing problems is that we do not only search feasible sequences, but try to minimize the violation of the various constraints. This is a very important property, because in realistic production environments it is quite common that no sequence exists that fulfills all of the constraints, but a good sequence should be given nevertheless.

2. THE MODEL

The input of our problem can be defined as follows. There is a set Σ of product types, and for product type $\sigma \in \Sigma$ we have to produce n_σ pieces. The length of the sequence is defined as $n = \sum_{\sigma\in\Sigma} n_\sigma$. There are two types

of constraints: regular expression constraints and linear bound constraints, and an input specifies a set of each type C_r and C_b respectively. Each constraint $c \in C_r \cup C_b$ has a priority P_c.

A *regular expression constraint* is given as the graph representation of a finite state automaton with input alphabet Σ. A sequence satisfies a regular expression constraint if the corresponding automaton accepts it. Note that it would be possible to merge multiple regular expression constraints as the intersection of the corresponding automata, eliminating the need to allow more than one of this constraint type. However this way we couldn't handle constraints with different priorities.

A *linear bound constraint* is defined as $(p_1, p_2, \mathbf{a}, b)$, where $1 \leq p_1 \leq p_2 \leq n$ are two positions, $\mathbf{a}$ is a weight vector of dimension $|\Sigma|$, and b is a bound. A constraint of this type is satisfied if $\sum_{\sigma \in \Sigma} a_\sigma s_\sigma \leq b$, where s_σ is the number of occurrences of product σ in the closed interval $[p_1, p_2]$. a_σ and b may be positive or negative (or 0), meaning that it is possible to specify both lower and upper bounds with these constraints.

The goal is to provide a sequence π of the input products that minimizes the violation of the constraints in a lexicographic order according to the constraint priorities. Defining this measure is straightforward for the linear bounds as the amount of violation, but it is harder for the regular expressions. We have chosen the *edit distance* as a measure, i.e. the minimal number of products that have to be inserted, deleted or changed in a sequence π to obtain a sequence π' that is accepted by the given automaton.

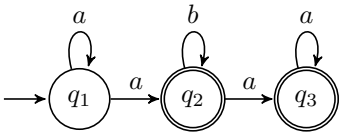
Calculating the edit distance for a given sequence $\pi = (\sigma_1, \ldots, \sigma_n)$ and a regular expression constraint $c \in C_r$ can be formulated as a shortest path problem in a directed graph (Van Hoeve et al. [2006]). Assuming that G_c is the graph representation of c with m nodes $q_1, \ldots, q_m$, we construct a layered graph G'_c with $(n+1)m + 2$ nodes as follows: a source and a terminal node s and t will be introduced, respectively, along with $n+1$ copies of $q_1, \ldots, q_n$. $q_{i,j}$ will represent the i^{th} copy of node q_j. An arc of the graph will have a label $l_e \in \Sigma \cup \{*\}$ and a weight $w_l \in \{0, 1\}$. The arcs of G'_c are defined as follows:

- $e = (s, q_{1,1}) \in E(G'_c)$, $l_e = *$, $w_e = 0$, assuming that q_1 is the initial state of G_c
- $\forall i = 1, \ldots, n+1 : e = (q_{i,j_1}, q_{i,j_2}) \in E(G'_c) \Leftarrow (q_{j_1}, q_{j_2}) \in E(G_c)$, $j_1 \neq j_2$; $l_e = *$, $w_e = 1$ (deletion arcs)
- $\forall i = 1, \ldots, n, j = 1, \ldots, n, \sigma \in \Sigma : e = (q_{i,j}, q_{i+1,j}) \in E(G'_c)$; $l_e = \sigma$, $w_e = 1$ (insertion arcs)
- $\forall i = 1, \ldots, n, \sigma \in \Sigma : e = (q_{i,j_1}, q_{i+1,j_2}) \in E(G'_c) \Leftarrow e_0 = q_{j_1,j_2} \in E(G_c)$; $l_e = \sigma$, $w_e = 0$ if $\sigma = l_{e_0}$, $w_e = 1$ otherwise (normal and substitution arcs)
- $e = (q_{n+1,j}, t) \in E(G'_c)$, $l_e = *$, $w_e = 0$, if q_j represents an accepting state in G_c

The construction of G'_c is illustrated on Figure 1.

Van Hoeve et al. [2006] showed that the weight of a shortest path from s to t in G'_c is equal to the edit distance of π if only arcs with the following properties may be used: (i) arc e has label $l_e = *$ (ii) arc $e = (q_{i_1,j_1}, q_{i_2,j_2})$ has label

Graph G_c of the automaton of constraint c:



Graph G'_c:

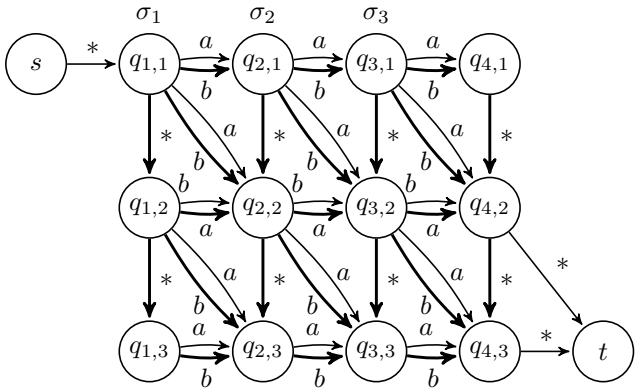


Fig. 1. Example of constructing the graph G'_c from the graph G_c of the automaton of constraint c, in order to calculate the edit distance. In this example, the length of the sequence (n) is 3. Thin arcs have weight 0, while thick arcs have weight 1.

$l_e = \sigma_{i_1}$ (i.e. the arc has the same label as the product in the position corresponding to its source node).

Now we define the integer linear programming formulation (IP) of the constraints separately for each type. For a regular expression constraint $c \in C^{\text{re}}$, we formulate the following problem:

$$\min \sum_{e \in E(G'_c)} w_e x_e \tag{1}$$

s.t.

$$\sum_{e=(\_,q_{i,j})} x_e = \sum_{e=(q_{i,j},\_)} x_e \quad \begin{matrix} \forall i = 1, \ldots, n+1, \\ \forall j = 1, \ldots, m \end{matrix} \tag{2}$$

$$\sum_{e=(s,\_)} x_e = 1 \tag{3}$$

$$\sum_{e=(\_,t)} x_e = 1 \tag{4}$$

$$\sum_{e:l_e=\sigma} x_e = n_\sigma \quad \forall \sigma \in \Sigma \tag{5}$$

$$x_e \in \{0, 1\} \quad \forall e \in E(G'_c) \tag{6}$$

A decision variable x_e is 1 iff the shortest path from s to t in the optimal sequence uses arc e. Constraints (2)-(4) ensure that the selected arcs form a path from s to t, while constraint (5) prescribes that exactly the required amount of products are present in the sequence for each product type. Note that in a feasible solution arcs having their source and terminal nodes in different layers must have a label from Σ, and exactly one arc is on the path between each layer. This ensures that a sequence can be easily determined from a feasible solution of this IP. For

the sake of the easier description we didn't include the variables and constraints that could be used to directly retrieve the actual solution.

A linear bound constraint $c \in C_b$ can be formulated as an assignment problem: we define a complete bipartite graph with n vertices in each class; one of the classes represents the positions $i = 1, \ldots, n$, the other represents the products to be sequenced $j = 1, \ldots, n$. The notation $\sigma(j)$ is used to represent the product type of the j^{th} product. We can now formulate the IP describing this problem:

$$\min z \tag{7}$$

s.t.

$$\sum_{j=1}^{n} y_{i,j} = 1 \quad \forall i = 1, \ldots, n \tag{8}$$

$$\sum_{i=1}^{n} y_{i,j} = 1 \quad \forall j = 1, \ldots, n \tag{9}$$

$$\sum_{i=p_1}^{p_2} a_{\sigma(j)} y_{i,j} - b \leq z \tag{10}$$

$$0 \leq z \tag{11}$$

$$y_{i,j} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n \tag{12}$$

A decision variable $y_{i,j}$ is 1 iff product j is assigned to position i, while the slack variable z represents the violation of c. Constraints (8)-(9) ensure that we find a perfect matching, while constraint (10) calculates the actual violation. Note that this problem is trivial to solve, but we will need this formulation to use it as a building block later.

In the following, for a constraint $c \in C_r \cup C_b$, we will refer to the corresponding objective function expression and the IP constraints as OBJ_c and CNSTR_c, respectively.

2.1 Measuring Solution Quality

Given a sequence π, let the violation of constraint c be $f_c(\pi)$. The violation of constraints that share the same priority P is defined as $f_P(\pi) = \sum_{c \in C_r \cup C_b, P_c = P} f_c(\pi)$. Assuming that the priorities are from a totally ordered set $\mathcal{P}$, we say that sequence π_1 is better than sequence π_2 if $\exists P \in \mathcal{P} : \forall P' < P : f_{P'}(\pi_1) = f_{P'}(\pi_2)$ and $f_P(\pi_1) < f_P(\pi_2)$.

In cases where we need to describe the quality of a sequence with one number, we calculate weights W_P for each priority to ensure that $f(\pi) = \sum_{P \in \mathcal{P}} W_P f_P(\pi)$ provide the same ordering as the lexicographic comparison described previously. We try to avoid this representation when possible because (depending on the input) it requires performing calculations with arbitrary precision.

2.2 Complexity

Solving a regular expression problem without the linear bound constraints is already NP-hard: a reduction from the directed s-t-Hamilton path problem can be given. Solving a set of the linear bound constraints is also NP-hard, as a reduction from 3-Partition can be shown.

3. MULTIOBJECTIVE OPTIMIZATION BY INTEGER PROGRAMMING AND GENETIC ALGORITHM

We have devised a three-phase method to solve the problem, as follows.

Algorithm:

(1) The regular expression constraint problems are solved independently, resulting in a set of solutions.
(2) Using these solutions as the initial population, an optimization by a genetic algorithm is performed.
(3) Finally, the best individual is improved further by a tabu search method. The best solution found in this phase will be the resulting sequence.

In the following, we provide the details of each phase.

3.1 Phase 1: Finding Initial Solutions

In this phase we solve $|C_r| + 1$ problems using the models described in Section 2 as building blocks. Namely, for each $c_r \in C_r$ we create the following IP:

$$\min W_{c_r} \text{OBJ}_{c_r} + \sum_{c' \in C_b} W_{c'} \text{OBJ}_{c'} \tag{13}$$

s.t.

$$\text{CNSTR}_{c_r} \tag{14}$$

$$\text{CNSTR}_c \quad \forall c \in C_b \tag{15}$$

$$\mu_{i,j}^{c_r} = \mu_{i,j}^{c} \quad \forall c \in C_b, \forall i, j = 1, \ldots, n \tag{16}$$

Constants W_c are weights calculated in a way to ensure the lexicographic ordering of the objective function values according to the priorities P_c, and $\mu_{i,j}^c$ are the appropriate decision variables in the IP model of constraint $c \in C_r \cup C_b$ that describe the assignment of product j to position i. In other words, the objective function values are summarized for the given regular expression constraint and all of the linear bound constraints, their IP constraints are merged, and the corresponding variables are connected to enforce the same solution in all of the problems.

Furthermore, an additional IP is created with only the linear bound constraints, similarly as above. This can be useful in cases where the solution of regular expression problems may be terminated prematurely because of a time limit, but the solution of the linear bound constraints alone may be fast enough.

This approach could be generalized as to select k suitable constraint sets from $C_r \cup C_b$ that seem to be tractable depending on the actual input, and create k initial solutions using the same idea as described above.

3.2 Phase 2: Optimization with a Genetic Algorithm

The sequences are represented by string genomes, where the alleles of the genes are the elements of Σ. We restrict the population to feasible solutions of our problems, i.e. only individuals of length n are considered with the appropriate number of occurrences of each $\sigma \in \Sigma$. To this end, we start from an initial population that satisfies

these assumptions and we use such genetic operators that preserve the required properties.

The initial population consists of the solutions of the previous phase, and some randomly generated individuals if needed. The latter ensures that we can handle problems where the solution of the IP-s are intractable as well as problems with fewer constraints than the desired size of the initial population.

We use the Generalized Order Crossover operator (Goldberg [1989]) for the crossover and a simple Swap Mutator for mutation. The evolution process is performed by a Steady State Genetic Algorithm that uses overlapping populations (De Jong [1975]). The selection of the individuals for crossover is performed by the Roulettewheel Method (Goldberg [1989]).

3.3 Phase 3: Tabu Search

In the last phase we try to improve the best individual found by the genetic algorithm by a local search method. The neighborhood of a sequence $\pi = (\sigma_1, \ldots, \sigma_n)$ is defined as all the sequences obtained by swapping two non-identical elements in π, i.e. for all pairs $i \neq j$ a neighbor π' is generated by swapping σ_i and σ_j if $\sigma_i \neq \sigma_j$. As a result, the size of the neighborhood is $O(n^2)$.

Because of the large neighborhood, evaluating all neighbors would be too time consuming. To overcome this problem, we evaluate a neighbor only with a probability $0 < p \leq 1$. On the one hand, this can speed up the search considerably, and on the other hand, this can help to diversify the solutions if the search space contains a large amount of similar solutions.

To avoid trapping in locally optimal solutions, we guide the search by tabu list control. We maintain a tabu list that consists of pairs (i, σ) of positions and product types, forbidding the insertion of product type σ to position i. When selecting a neighbor in an iteration by swapping σ_i and σ_j in positions i and j, the pairs (i, σ_i) and (j, σ_j) are introduced to the tabu list. The neighborhood in each iteration is restricted to neighbors that are accessible by non-tabu operations. However the tabu status does not affect the choice of a neighbor that is better than the best sequence found so far.

Using this method ensures that the final solution will be locally optimal (if given enough time), hence a user of a system implementing this method can't improve the solution by applying small changes. This property is very important to help the acceptance of automated optimization systems.

4. IMPLEMENTATION AND COMPUTATIONAL RESULTS

We have implemented our method as a part of a larger industrial information system. For solving the Integer Programming problems, we use the COIN-OR Branch and Cut library, while the genetic algorithm is implemented using the open source GAlib library of Matthew Wall.

The actual parameters of the algorithm are summarized in Table 1. In order to balance between slow exhaustive evaluation and probabilistic evaluation, we use different parameters during different states of the tabu search method. Initially the probability of evaluating a neighbor is 1, i.e. we evaluate all neighbors. During the search, if the best known sequence was found in one of the last 10 iterations, this probability is set to 1 (exploring the search space around our best solution), otherwise to 0.4 (diversifying the search while traversing the search space rapidly).

Genetic algorithm	
Population size	90
Replacement of population between generations	80%
Mutation probability	5%
Tabu search	
Tabu list length	$n/4$
Probability p of evaluating a neighbor	1 or 0.4, depending on the status of the search

Table 1. Parameters of the optimization method.

4.1 A Numerical Example

In this section we provide an example that shows a small, realistic problem setting, for which we provide a description using our framework, and we demonstrate the best solution found by our method. In our example, there are 10 product types, $\Sigma = \{a, \ldots, j\}$. For the easier presentation, these products are partitioned into product families, namely $A = \{a, b, c\}$, $D = \{d, e, f, g, h\}$, and $I = \{i, j\}$. This isn't a generalization of the problem, as these families are used only for conveniently referring to a set of products. When referring to a product family, we mean any of its products. In the description of regular expression constraints, we refer to products not in a family F as $\neg F = \Sigma \setminus F$.

A sequence should satisfy the following requirements:

c_1: No more than 2 pieces of product family A should be processed consecutively.
c_2: If the previous constraint is violated, no more than 4 pieces of product family A should be processed consecutively.
c_3: After processing products from family A consecutively, at least 2 products from family D should be processed consecutively.
c_4: After processing products from family A consecutively, the next 3 products can't be from family A.
c_5: Products from family I should be produced in batches of size 2 or 3.
c_6: After processing products from family I consecutively, the next 3 products should be from family D.
c_7: At least 2 pieces of product a should occur in the first 12 positions.
c_8: At least 1 piece of product b should occur in the first 20 positions.
c_9: At least 3 pieces of product e should occur in the first 10 positions.
c_{10}: At least 2 pieces of product i should occur in the first 15 positions.
c_{11}: At least 1 piece of product j should occur in the first 12 positions.

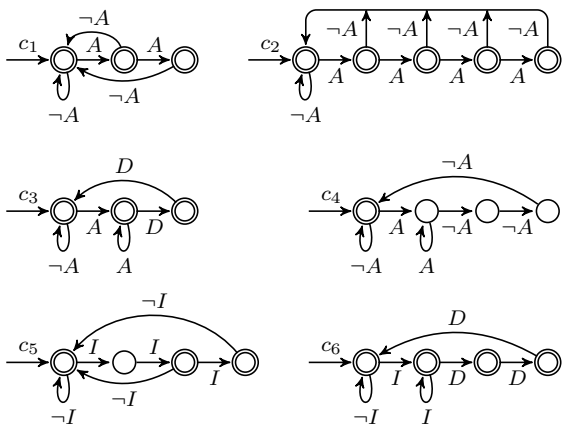


Fig. 2. The automata corresponding to the numerical example. The initial states are labeled with the numbers of the corresponding regular expression constraints.

	p_1	p_2	a_a	a_b	a_c	a_d	a_e	a_f	a_g	a_h	a_i	a_j	b
c_7	1	12	-1										-2
c_8	1	20		-1									-1
c_9	1	10					-1						-3
c_{10}	1	15									-1		-2
c_{11}	1	12										-1	-1
c_{12}	1	24	6				4	4				5	24
c_{13}	1	15		2						1	1		6
c_{14}	1	10	1	1	1								3
c_{15}	11	20	1	1	1								3
c_{16}	21	30	1	1	1								3

Table 2. Parameters of the linear bound constraints of the numerical example. Zero weights are not displayed.

c_{12}: Products a, e, f and j require the amount 6, 4, 4 and 5 units of material m_1, respectively, which is on stock in a limited amount of 24 units until position 10.

c_{13}: Products b, h and i require the amount 2, 1 and 1 units of material m_2, respectively, which is on stock in a limited amount of 6 units until position 15.

c_{14}: Products from product family A should occur at most 3 times from position 1 to position 10.

c_{15}: Products from product family A should occur at most 3 times from position 11 to position 20.

c_{16}: Products from product family A should occur at most 3 times from position 21 to position 30.

Constraints c_1-c_6 describe the structural requirements of the sequence, c_7-c_{11} are due date constraints, c_{12}-c_{13} are material constraints, and c_{15}-c_{16} are capacity constraints. We have formulated requirements c_1-c_6 as regular expression constraints, and the others as linear bound constraints. The finite state automata of $C_r = \{c_1, \ldots, c_6\}$, and the parameters of $C_b = \{c_7, \ldots, c_{16}\}$ are shown on Figure 2 and Table 2, respectively.

The priorities of the constraints and the amounts of the products are shown on Table 3. The idea behind assigning the priorities is the following: material constraints are the most important, as material shortage makes the production physically impossible. In order of importance, next are the due dates, as serving the customers is typically more important than internal preferences. Assigning priorities to the structural requirements was following the idea of

σ	a	b	c	d	e	f	g	h	i	j
n_σ	3	3	2	3	3	4	2	1	5	4

c	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
P_c	6	3	5	4	6	5	2	2

c	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}
P_c	2	2	2	1	1	7	7	7

Table 3. Amount of the products to be processed, and priorities of the constraints. Lower numbers represent higher priorities.

products of family A being "hard" to produce, products of family D being the "easy" ones, while products of family I are moderately hard to produce. In this example, capacity constraints represent the desire not to overload any part of the sequence with products of family A.

We have run our algorithm on the input described above for 2 minutes, divided equally among the three phases. The genetic algorithm generated ∼4000 generations, and the tabu search performed ∼4500 iterations. The best result was the following sequence:

*AADD I I I DDDAA***A***DD I I I DDDAADDD I I I*
c c g g i i j e e e a a **a** *b f f i i i h f f b b d d d j j j*

For the sake of the easier overview, both the products and the corresponding product families are shown. The intervals $[p_1, p_2]$ of the linear bound constraints are drawn with light gray lines for reference, in the order of their definition.

The only violations of this sequence are the following, each having a low priority:

- c_1: the violation is marked with bold in the sequence: there are more than 2 products from family A scheduled consecutively.
- c_{15}: between positions 11 and 20, there are 4 products from family A, however there should be at most 3.

All other constraints are perfectly satisfied.

4.2 Experiments on Industrial Data

We have performed preliminary testing on actual industrial data. The typical size of the problems is summarized in Table 4. We have set the time limit of the test runs to 10 minutes, which was divided equally among the phases. On average, the genetic algorithm has generated ∼4000 generations, and the tabu search algorithm performed ∼2500 iterations on a personal computer with Intel Core2 Quad CPU running on 2.33GHz. Results have shown that only some low priority constraints were violated, and the quality of the solutions overperformed the ones made by the planning personnel with hours of work. Only 2-3 constraints with the lowest priorities were violated, and the violations were due to typically one, rarely two products being in a wrong position. This amounted to sequences of ∼80 products of which only 2-5 weren't in a perfect position.

Parameter	Value
n	$\sim$80
$\|\Sigma\|$	$\sim$10
$\|C_r\|$	$\sim$10
$\|C_b\|$	$\sim$5
Number of states in regular expression constraints	2-10
Average number of states in regular expression constraints	5.5

Table 4. Typical parameters of the industrial data.

5. CONCLUSIONS AND FUTURE RESEARCH

In this paper we have defined a model that generalizes the well known car sequencing problem by introducing regular expression constraints and general linear bound constraints, and we have also proposed a solution method that can be used in practical applications. To assess the performance of our algorithm, a systematic evaluation is still needed. As our method is capable of describing car sequencing problems, it would be interesting to test it on standard benchmark problems. Furthermore, the quality of the solutions should be investigated for different constraint settings, regular expressions, and problem sizes.

REFERENCES

K. A. De Jong. *Analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, 1975. Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

M. Fliedner, N. Boysen. Solving the car sequencing problem via Branch & Bound. *European Journal of Operational Research*, 191(3): 1023-1042, 2008.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989. Reading, MA.

T. Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4), 331–335, 2004.

D. Parello, W.C. Kabat, and L. Wos. Job-shop scheduling using automated reasoning: a case study of the car sequencing problem. *Journal of Automated Reasoning*, 2: 1–42, 1986.

C. Solnon, V.D. Cung, A. Nguyen, C. Artigues, The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the ROADEF2005 challenge problem, *European Journal of Operational Research*, 191: 912–927, 2008.

W.-J. Van Hoeve, G. Pesant, L.-M. Rousseau, Sabharwal, Revisiting the cart sequencing problem, *CP 2006, LNCS*, 4204: pp. 620–634, 2006.