Journal of the Operational Research Society, ***to appear***

REVIEW AND COMPARISON OF THREE METHODS FOR THE SOLUTION OF THE CAR SEQUENCING PROBLEM

Marc GRAVEL[1], Caroline GAGNÉ[1] and Wilson L. PRICE[2]

[1]Département d'informatique et de mathématique
Université du Québec à Chicoutimi
555, Boul. de l'Université
Chicoutimi, Québec, Canada G7H 2B1
marc_gravel@uqac.ca, caroline_gagne@uqac.ca

[2]Faculté des Sciences de l'administration,
Université Laval
Ste-Foy, Québec, Canada G1K 7P4
wilson.price@fsa.ulaval.ca

ABSTRACT: The car sequencing problem is the ordering of the production of a list of vehicles which are of the same type, but which may have options or variations that require higher work content and longer operation times for at least one assembly workstation. A feasible production sequence is one that does not schedule vehicles with options in such a way that one or more workstations are overloaded. In variations of the problem, other constraints may apply. We describe and compare three approaches to the modeling and solution of this problem. The first uses integer programming to model and solve the problem. The second approaches the question as a constraint satisfaction problem (CSP). The third method proposes an adaptation of the Ant Colony Optimization for the car sequencing problem. Test-problems are drawn from CSP*Lib*, a publicly available set of problems available through the Internet. We quote results drawn both from our own work and from other research. The literature review is not intended to be exhaustive but we have sought to include representative examples and the more recent work. Our conclusions bear on likely research avenues for the solution of problems of practical size and complexity. A new set of larger benchmark problems was generated and solved. These problems are available to other researchers who may wish to solve them using their own methods.

Keywords: Car sequencing problem, Ant Colony Optimization, Integer Linear Programming, Constraint Satisfaction Problem

1. INTRODUCTION

Sequencing and scheduling problems are combinatorial optimization problems which may have more than one objective function. The feasible solution space is often quite large, thus increasing the difficulty of solution. For the *car sequencing problem*[1], the set of constraints that must be respected makes it difficult, and sometimes impossible to find a feasible solution. In such cases, one may be content with a solution that violates as few constraints as possible.

The *car sequencing problem* requires the determination of the order in which a set of cars should be manufactured in a plant composed of three consecutive shops: the body fabrication shop, the paint shop and the assembly shop. In the fabrication shop, metal stampings and pressings are welded together to form the car body. Setup times on the presses are such that one will tend to favor the longest possible sequences of similar vehicles. In the paint shop, car bodies are treated, prepared and painted. Purging paint nozzles when colors are changed leads to delays and to a loss of materials that constitute a significant cost. To minimize these costs, one will seek to form long sequences of cars of identical color. In the assembly shop, the thousands of mechanical, electrical and trim components required for a finished vehicle are added to the car body. Each car is characterized by a set of options, such as a sun-roof or air-conditioning, which may require high work-content operations. Cars may be grouped into classes of vehicles that

are identical. Vehicles requiring high work-content options must be dispersed throughout the sequence to smooth workload at the affected workstations. Since the components added in the assembly shop represent 70% of the value of the vehicle, the assembly sequence requires careful planning to ensure load balancing and component supply.

Each of the three workshops is therefore subject to specific constraints that tend to be in conflict. For example, a good sequence for the assembly shop may require very short color sequences in the paint shop. Because of the complexity of the problem, however, the car sequencing problem has sometimes been formulated in the literature taking into account only the constraints of the assembly shop. The sequences thus found are then applied to the entire production process.

Accepting this point of view which uses the assembly shop sequence as a solution, the car sequencing problem has been shown to be NP-hard in a strong sense[2]. Lopez & Roubellat[3] published a review of the relevant literature and report on both exact and heuristic solution methods. One obvious conclusion drawn from this review is that the size and the difficulty of problems for which solutions have been found is relatively small.

In other recent works, some authors have proposed heuristic methods for the solution of the car sequencing problem. For example, ant colony optimization was used by Solnon[4] and Gottlieb *et al.*[5] and neighborhood search methods were proposed by Davenport *et al.*[6], Davenport & Tsang[7], Putcha & Gottlieb[8] and Gottlieb *et al.*[5].

In this paper, we will first refer to work from the area of Constraint Satisfaction Programming (CSP) and to an existing benchmark set of car sequencing problems. We then show that an exact method, integer linear programming, can solve some of the problems in the CSP benchmark set using currently available solvers and computing resources. We follow the approach used in the CSP literature in solving these benchmark problems and sequence the assembly shop only. However, the solution times for exact methods, the large size of practical industrial problems and the level of difficulty of some problems lead us to present a metaheuristic solution method. We first present the results that we have obtained using an ant colony optimization metaheuristic to solve the *CSPLib* benchmark problems. We then propose and solve new and larger problems that we offer for inclusion in the benchmark set.

2. SOLVING THE CAR SEQUENCING PROBLEM VIA CSP

Constraint Satisfaction Programming (CSP) has evolved from research in the computer science and artificial intelligence community. It is of interest in the search for solutions to NP-complete combinatorial problems. A CSP is defined as feasibility rather than an optimization problem, and CSPs will tend to have a complex set of constraints. In some cases, problem size may prevent discovery of a feasible solution while in others such a solution may not exist. In such instances, the CSP may retain the solution that violates the least number of constraints or alternately, the least important constraints.

Formally, and following the notation used in the review by Lustig & Puget[9], each decision variable x_j of a set of n such variables has a set D_j of allowable values which is called the domain of the variable. A constraint $c(x_1, x_2, x_3, \ldots x_n)$ specifies a subset S of the overall variable domain $D_1 \times D_2 \ldots \times D_n$. A particular set of variable values $(\underline{x}_1, \underline{x}_2, \underline{x}_3, \ldots \underline{x}_n)$ satisfies the constraint if $c(\underline{x}_1, \underline{x}_2, \underline{x}_3, \ldots \underline{x}_n) \in S$. A solution to the CSP has been found if all constraints are satisfied and all variables are within their specified domains. Note that no hypotheses are made as to the characteristics of the domains or of the constraints.

A CSP is solved using algorithms based on the principles of *domain reduction* and *constraint propagation*. A variable is examined to see if its domain may be reduced following analysis of a constraint in which it appears. Domain reductions thus discovered are propagated to other constraints in which the variable in question appears and this may allow a further domain reduction in one or more variables. A widely used algorithm for domain reduction and constraint propagation was proposed by Van Hentenryck *et al.*[10]. Later, Van Hentenryck[11] described the Optimization Programming Language (OPL), which allows the expression of an optimization problem in a form suitable for solution either by mathematical programming or by CSP techniques.

The car sequencing problem is a PermutCSP (see, for example Solnon[4]), a particular type of problem such that all the solutions are permutations of a given 'tuple'. It has become well known in the CSP community because a version of this problem is used as a benchmark by researchers in the area. Because the various CSP algorithms are programmed differently, there is no universal and conventional form for stating a CSP. The problem formulation is not expressed in a set of algebraic relationships but in a computer format devised by the software supplier. The interested reader may consult a CSP formulation in OPL for a small car sequencing problem (CAR.zip) on the website of the ILOG Corporation (http://www2.ilog.com/oplmodels/).

Certain CSP researchers[12, 13] have created the *CSPLib* library of test problems published via the Internet to stimulate competition (http://4c.ucc.ie/~tw/csplib/schedule.html) and to provide benchmarks for various researchers working in the field. *CSPLib* offers a number of instances of the car sequencing problem as well as the best solutions currently known. Two problem sets have been proposed. The first[12] is composed of seventy problems for sequencing 200 cars having 5 options and from 17 to 30 classes and for which there is at least one feasible solution. However, problem difficulty increases as the utilization rate of the various options increases, so that a small problem may be harder to solve than one having more vehicles to sequence. Gent & Walsh[13] have therefore proposed a second group of nine more difficult problems for sequencing 100 cars having 5 options and from 19 to 26 classes.

For eight of the nine problems of the second set, Regin & Puget[14], Gent[15], Gottlieb *et al.*[5] and Boivin[16] either found a feasible solution or show that the problem is infeasible. Only one problem has not yet been so categorized.

It is evident that the efficiency of the approaches used decreases as the problem size and problem difficulty increases. When a feasible solution does exist, CSP approaches perform well, but when there is no feasible solution, they often terminate after long computing times without demonstrating infeasibility.

While in this paper we do not offer any new CSP algorithms, we do use the known *CSPLib* results in comparisons with our work.

3. INTEGER PROGRAMMING MODEL

Since the car sequencing problem has been shown to be NP-hard, it should not be expected that exact optimization techniques will be able to solve large problem instances. However, we have found that current solvers and computing resources allow the solution of some of the existing benchmark problems.

The formulation proposed groups identical cars, that is to say those having exactly the same options, into classes. The variable and parameter definitions are summarized in Table 1. For option k, the dispersion required for workload balancing is expressed by a ratio r_k/s_k meaning that in a consecutive sequence of s_k vehicles, at most r_k vehicles can require option k. A feasible solution to the problem is a sequence that respects the entire set of option workload constraints. If it is impossible to find such a set of constraints, the sequence violating the smallest number of constraints is often chosen.

Table 1: The variable and parameter definitions

opt :	the total number of options
cl :	the total number of classes
n_i :	the total number of cars in class i
nc :	the total number of cars
o_{ik} :	a boolean parameter showing if cars of class i require option k
s_k	the length of a consecutive sequence of cars, some of which require option k
r_k :	the maximum number of cars that can have option k in a consecutive sequence of s_k cars.
M :	a large scalar value that may be fixed to (s_k - r_k)
C_{ij} :	a boolean variable showing if a car of class i is assigned to position j in the sequence (C_{ij} = 1)
Y_{kj} :	a boolean variable showing if the number of cars requiring option k in a sequence of s_k cars starting at position j in the sequence respects r_k, the maximum allowed (Y_{kj} = 0)

The formulation as an integer linear program is as follows:

$$\text{Min } F = \sum_{k=1}^{opt} \sum_{j=1}^{nc-s_k+1} Y_{kj} \quad (1)$$

subject to

$$\sum_{i=1}^{cl} C_{ij} = 1 \qquad \text{for } j = 1..\ nc \quad (2)$$

$$\sum_{j=1}^{nc} C_{ij} = n_i \qquad \text{for } i = 1..\ cl \quad (3)$$

$$\sum_{i=1}^{cl} \sum_{l=j}^{j+s_k-1} o_{ik} * C_{il} \leq r_k + M * Y_{kj} \qquad \text{for } k = 1..\ opt \text{ and } j = 1..\ (nc - s_k + 1) \quad (4)$$

C_{ij} boolean *for i = 1.. cl and j = 1.. nc*

Y_{kj} boolean *for k = 1.. opt and j = 1.. nc- s_k + 1*

In the objective function of expression (1), we seek to minimize the number of times that the required work content will exceed the capacity of the workstations for which we have expressed a constraint of the type r_k/s_k for an option k. For each option, we must therefore consider all sequences of length s_k. The first group of constraints represented in expression (2) indicates that a car from only one class may be assigned to each position of the sequence. Expression (3) ensures that all cars of each class are assigned to a position in the sequence. Expression (4) formalizes the constraints of type r_k/s_k linked to the various options but allows them to be violated at a cost if no feasible solution can be found.

Using the power of a linear integer programming solver (XPRESS® from Dash Optimization), this formulation allows us to find feasible solutions to all of the first group of problems proposed by Lee *et al.*[12]. For the second group of more difficult problems provided by Gent & Walsh[13], four problems (16-81, 26-82, 41-66, 4-72) were quickly solved to feasibility using XPRESS as is shown in Table 2. Solution times quoted were obtained using a computer having a Xeon 3.06Ghz chip and 1024 MB of RAM. The four problems that were shown to be infeasible (10-93, 19-71, 6-76, 36-92), and the problem for which no conclusion has been yet reached (21-90), required long computation times and we were only able to reproduce the previously known solutions. Table 2 shows the time taken by the solver to attain the best previously known solution for each problem. Note that we used the value of these previously known solutions as an upper bound so as to accelerate the exploration of the solution tree. It is therefore of interest to examine heuristics in order to compare their efficiency in finding feasible solutions and in comparing solution quality.

Table 2: Solution times using integer linear programming

Problem	**16-81**	**26-82**	**41-66**	**4-72**	**10-93**	**19-71**	**6-76**	**36-92**	**21-90**
Unsatisfied constraints	0	0	0	0	3	2	6	2	2
Solution status	Optimal	Optimal	Optimal	Optimal	*Infeasible*	*Infeasible*	*Infeasible*	*Infeasible*	Un-known
Computing time (seconds)	30*	21*	10*	15*	61 135**	4**	75**	195**	34**

** Time to find the optimal solution by a complete search.*
*** Time to reach the best previously known solution.*

3. AN ANT COLONY OPTIMIZATION HEURISTIC FOR THE CAR SEQUENCING PROBLEM

In this section we describe ant colony optimization (ACO) heuristic for the car sequencing problem. As in the previous section, the objective of the optimization is to minimize the total number of violations of constraints of the type r_k/s_k for all options k. Further to the definitions of Table 1, we require those shown in Table 3. Figure 1 shows the steps of a summary presentation of our ACO heuristic for the car sequencing problem.

Table 3: Variable and parameter definitions used in the ACO

$p_{uv}^{m}(t)$	the probability that ant m will choose v as the class of car to follow class u at time t
d_v	an indicator of the difficulty of scheduling cars of class v
$\tau_{uv}(t)$	the cumulative pheromone trail between classes u and v at time t
F^*	the best solution so far found
F_+	the best solution found at the end of a cycle
ρ_ℓ	the local persistence of the pheromone trail
ρ_g	the global persistence of the pheromone trail
a_k	the total number of cars requiring option k
a'_k	the remaining number of cars requiring option k at a given time
nc'	the remaining number of open slots in a sequence of cars at a given time
$Util_k$	the utilization rate of the option k
α, β, δ	the parameters expressing the relative influence of the pheromone trail and of the different types of visibility
t_{Max}	the total number of cycles
nb_ants	the total number of ants
$Tabu_m$	the list of classes for which there remain no further cars to be sequenced for the ant m

Figure 1. Summary presentation of an ACO heuristic for the car sequencing problem

$t = 0$;
Set the pheromone trail matrix $\tau(t)$ at τ_0 (a small positive value) for each pair of classes uv;
FOR $t = 1$ to t_{Max} DO
  Initialize the first position of the sequence for each ant by a random choice from among the classes of cars.
  FOR $pos = 2$ to nc DO
    FOR $m = 1$ to nb_ants DO
      Choose the next class v, $v \notin Tabu_m$, to be added to the sequence after the class u among the cl candidate classes using the transition rule (Relation 6);
      Local update of the pheromone trail for (u,v) ;
  FOR $m = 1$ to nb_ants DO
    Evaluate solution m;
  Global update of the pheromone trail using F_+, the best solution of the current cycle;
  Update of F^*, the best solution so far found;

The first cycle of the heuristic commences with an empty sequence and proceeds by construction, adding cars one by one. A number of *ants*, which may be thought of as independent *agents*, are set in motion, each ant building a complete sequence. A car is added to the sequence using the transition rule. This transition rules take into account the desirability (*visibility*) of a car type and the accumulated *pheromone trail*, which, in cycles following the first, 'remembers' the quality of previous solutions that have been found. As each car is added, a local update of the pheromone trail is made. When all ants have constructed a complete sequence, a global update of the pheromone trail is done using the best sequence found in the cycle. The best sequence so far found is retained, and a new cycle is started.

Note that the ACO heuristic incorporates multiple visibility matrices to guide the search[17] as well as the improvements to the transition rule and the candidate list proposed by Dorigo & Gambardella[18]. These improvements affect three areas : the first subdivides the original transition rule[19] into a probabilistic part and a deterministic part; the second updates the pheromone trail both locally and globally; the third uses a candidate list to limit the choices that must be considered at each step. In recent books, Bonabeau *et al.*[20] and Dorigo & Stützle[21] offer detailed information on the ACO metaheuristic and the choice of the various parameters.

We will now describe the main elements of the transition rule of the ACO of Figure 1. We first point out that problem difficulty cannot be simply expressed by the total number of cars (*nc*) to sequence. One must also consider the utilization rate (*Util*) of the various options. This can be expressed for an option k as a ratio between a_k, the total number of vehicles requiring option k, and the maximum number of vehicles that can receive this option ($nc * r_k/s_k$) so that the constraint r_k/s_k is satisfied. It is evident that there is no

feasible solution if the utilization rate for any option is greater than 1. On the other hand, even if all utilization rates are equal to or less than 1, a feasible solution does not necessarily exist.

An indicator of the global level of difficulty of class v (d_v) is obtained by summing the utilization rates of the options that this class requires as shown in relation 5. Note that the utilization rate for each option is computed dynamically taking into account the remaining number of cars requiring option k (a'_k) and the number of remaining open slots in sequence (nc').

$$d_v = \sum_{k=1}^{cl} o_{vk} \cdot Util_k \qquad (5)$$

The transition rule may be described as follows: τ_{uv} is the cumulative pheromone trail between classes u and v, $new\_conflicts_v$ is the number of new instances of constraint violation caused by placing a car of class v at the next open slot in the sequence, d_v is an indicator of the global level of difficulty of class v (relation 5) and α, β, and δ are parameters expressing the relative influence of the pheromone trail and of the different types of visibility of the problem. Therefore the transition rule for an ant m to choose v as the class of car to follow class u ($v \notin Tabu_m$ i.e. v being a class for which there remains at least one car to be sequenced) is given by:

$$v = \begin{cases} arg\ max_{\ell \notin Tabu_m} \left\{ \left[\tau_{u\ell}(t)\right]^{\alpha} \cdot \left[\frac{1}{1+new\_conflicts_{\ell}} \right]^{\beta} \cdot \left[d_{\ell}\right]^{\delta} \right\} & if\ q \le q_0 \\ V & if\ q > q_0 \end{cases}$$

where V is chosen according to the probability : (6)

$$p_{uv}^{m}(t) = \frac{\left[\tau_{uv}(t)\right]^{\alpha} \cdot \left[\frac{1}{1+new\_conflicts_{v}} \right]^{\beta} \cdot \left[d_{v}\right]^{\delta}}{\sum_{\ell \notin Tabou_m} \left[\tau_{u\ell}(t)\right]^{\alpha} \cdot \left[\frac{1}{1+new\_conflicts_{\ell}} \right]^{\beta} \cdot \left[d_{\ell}\right]^{\delta}}$$

We note that in this expression, q is a random number and q_0 is a parameter; both are between 0 and 1. Parameter q_0 determines the relative importance of the *exploitation* of existing information and the *exploration* for new solutions. Note that the transition rule allows the choice of the next class of car v from among a list of candidate classes. A car class is automatically a candidate if its choice does not engender new conflicts. If no such class exists, all classes of cars are placed in the candidate list.

A local update, diminishing the pheromone trail after each choice (*uv*), as shown in relation (7) will tend to prevent too many ants from repeating the same choice of classes and will favour diversity in the search process.

$$\tau_{uv}(t) = \rho_{\ell} \cdot \tau_{uv}(t) + (1-\rho_{\ell}) \cdot \Delta\tau_{uv} \qquad \text{where } \Delta\tau_{uv} = \tau_0 \qquad (7)$$

The best sequence found at the end of each cycle (F_+) determines how the update of the pheromone trail should be done, as shown in relation 8. Evaporation of the pheromone trail is also carried out for all pairs of classes that are not part of the best sequence.

$$\tau_{uv}(t+1) = \rho_g \cdot \tau_{uv}(t) + (1-\rho_g) \cdot \Delta\tau_{uv}(t) \qquad (8)$$

where $\Delta\tau_{uv}(t) = F^*/F_+$ and ρ_g represents the persistence of the trail.

The algorithm terminates after completion of a specified number of cycles (t_{Max}). In most cases, the values of the parameters in the ACO algorithm follow the suggestions given by Dorigo & Gambardella[18] and in the remaining cases, the values have been determined through direct numerical experimentation and sensitivity analysis. The parameters have been assigned the following values: $\tau_0 = 0.005$, $\rho_g = \rho_{\ell} = 0.99$, $nb\_ants = 15$, $q_0 = 0.9$, $t_{Max}=1000$ and $\{\alpha, \beta, \delta\} = \{1,6,3\}$. We will now show how this algorithm performs for the car sequencing problem.

For the first group of seventy problems of the *CPSLib*[12] for which there exist a solution violating no constraints, the ACO finds the optimal solution more rapidly than ILP. Table 4 presents the average run times obtained by both methods for each group of problems. Note that the ACO process stops as soon as a conflict-free solution is found. The ACO results represent the average of 100 trials for each problem of the group.

Table 4. Comparison of ILP and ACO solution times

PROBLEM GROUP	NUMBER OF PROBLEMS	UNSATISFIED CONSTRAINTS ILP	RUN TIME ILP* (seconds)	UNSATISFIED CONSTRAINTS ACO	RUN TIME ACO* (seconds)
prob60	10	0	31.5	0	0.02
prob65	10	0	65.5	0	0.02
prob70	10	0	84.8	0	0.02
prob75	10	0	106.6	0	0.02
Prob80	10	0	173.8	0	0.03
Prob85	10	0	174.7	0	0.04
Prob90	10	0	208.8	0	0.03

* Xeon 3.06Ghz chip and 1024 MB of RAM

For the second group of problems proposed by Gent & Walsh[13], the most difficult of which were not solved via ILP, columns 2 and 3 of Table 5 present the means and

standard deviations of results obtained in 100 trials of the ACO for these 9 problems. Comparing with the results of Table 2, we note that for problems 26-82, 41-66 and 4-72, the ACO always finds the optimal solution. For problem 6-76, we note that the ACO always found the best known solution and for problems 16-81, 19-71, 36-92 and 21-90 the best known solution is frequently found although, as the standard deviation shows, there is some variation. The most difficult problem to solve appears to be 10-93, where the best solution is found only occasionally. In Table 2, we previously showed that this problem was also the most difficult for the ILP algorithm as compared to the others. Columns 4 and 5 of Table 5 show the minimum and maximum numbers of unsatisfied constraints found by the ACO in the 100 trials. It should be noted that the minimum values found by the ACO correspond to those of the best known solutions for these problems as presented in Table 2. As well, we note in the last two columns that for all problems, the best solution is found in average well before attaining the maximum number of cycles allowed by the algorithm and that the average run time is quite low. Note that the term "Mean Cycle" refers to the average cycle, over the 100 trials, at which the best solution was found and is a measure often used in describing the performance of an ACO. Finally, we can explain the very short run times for some problems by the fact that computations are terminated as soon as a conflict-free solution has been found.

Table 5. ACO solutions to the CSPLib problems

PROBLEM	UNSATISFIED CONSTRAINTS				MEAN CYCLE	RUN TIME ** (seconds)
	Mean	Std deviation	MIN*	MAX		
16-81	0.1	0.33	0	1	329.10	4.23
26-82	0.0	0.00	0	0	83.72	0.74
41-66	0.0	0.00	0	0	5.84	0.04
4-72	0.0	0.00	0	0	58.72	0.50
10-93	4.2	0.47	3	5	298.93	10.10
19-71	2.1	0.27	2	3	309.48	9.26
6-76	6.0	0.00	6	6	1.01	7.94
36-92	2.3	0.46	2	3	346.02	8.50
21-90	2.6	0.49	2	3	324.39	8.73

* Corresponding to the best known solutions for these problems
** Xeon 3.06Ghz chip and 1024 MB of RAM

In a further set of numerical experiments, a local search method was used in conjunction with the ACO of Figure 1. The results of Table 6 show that the application of a local search method to the best solution found at each cycle as well as to the best overall solution yields the minimum values found in Table 5 in each of the 100 trials for all problems with the exception of problem 10-93. In this latter case, the performance of the algorithm improved but the best know solutions were not attained at each trial. We note as well that local search reduces the average number of cycles required to attain the best known solution, with the exception in the case of problem 10-93. Run time is reduced for those problems having a conflict-free solution, because computations are halted when

such a solution is found. Computing times increase for the other problems but remain relatively low.

The local search procedure applied to the best solution of a cycle is to reverse the order of a sub-sequence. First, one randomly selects the position of a car forming part of sequence of length s_k containing a conflict for some option. A second position is randomly chosen among all positions and the sub-sequence including and linking these two positions is inversed. A similar approach called Lin2Opt is described by Putcha & Gottlieb[8] except that the two positions are randomly chosen among all positions. If the solution is improved, it is retained as the best current solution. The procedure is repeated (2 * *nc*) times or is halted if a conflict-free solution is found.

The local search procedure applied to the best overall solution (*F**) is the Lin2Opt procedure of Putcha & Gottlieb[8] and it is repeated (2000 * *nc*) times or is halted if a conflict-free solution is found.

Table 6. ACO with local search solutions to the CSPLib problems

PROBLEM	UNSATISFIED CONSTRAINTS				MEAN CYCLE	RUN TIME ** (seconds)
	Mean	Std deviation	MIN*	MAX		
16-81	0.0	0.00	0	0	149.03	1.75
26-82	0.0	0.00	0	0	28.33	0.26
41-66	0.0	0.00	0	0	2.98	0.03
4-72	0.0	0.00	0	0	30.54	0.27
10-93	3.8	0.45	3	5	507.22	13.79
19-71	2.0	0.00	2	2	125.61	13.04
6-76	6.0	0.00	6	6	1.00	11.76
36-92	2.0	0.00	2	2	145.27	12.66
21-90	2.0	0.00	2	2	146.81	12.60

* Corresponding to the best known solutions for these problems
** Xeon 3.06Ghz chip and 1024 MB of RAM

A set of new and larger problems was randomly generated in order to demonstrate the performance of the ACO and to allow other researchers to better test the performance of their methods. These problems treat more cars (200, 300 and 400 cars) but their number remains beneath what may typically be found for one day's production in an industrial plant. These new problems maintain the same characteristics as those of Gent & Walsh[13] as to the number and type of options (5) and the number of classes (17 to 30). The average utilization rate of the options exceeds 90% and the combinations of the different options within the classes render these problems as difficult as those of Gent & Walsh. Interested researchers may obtain the new problem set from the Internet at http://wwwdim.uqac.ca/~c3gagne/.

Table 7 shows the results obtained in 100 trials with the ACO and local search. These results illustrate the difficulty of the new problems for which, for the most part, we have not found conflict-free solutions. The columns 2 and 3 show the average number of unsatisfied constraints and standard deviations over the 100 trials. The following columns present the minimum and maximum number of conflicts found, the average number of cycles at which the best solution was found and the average run time. We note that the best solution is generally found after a larger number of cycles than was the case for the problems of Table 6. When the local search procedure is applied to the best overall solution found by the ACO, it very often finds the best known solution for the problem. The computing times remain acceptable and, we believe, would allow an increase in the number of cycles or an intensification of the local search.

It is our hope that these new and more difficult problems and the solutions that we offer will present a new challenge to the community of researchers who are interested in the car sequencing problem.

Table 7. ACO solutions with local search to a new set of difficult problems

PROBLEM	UNSATISFIED CONSTRAINTS				MEAN CYCLE	RUN TIME * (seconds)
	Mean	Std deviation	MIN	MAX		
200_01	1.00	0.67	0	3	985.79	33.42
200_02	2.41	0.49	2	3	962.63	33.44
200_03	6.04	0.82	4	8	998.58	34.45
200_04	7.57	0.56	7	9	968.62	34.66
200_05	6.40	0.51	6	8	986.70	32.01
200_06	6.00	0.00	6	6	314.36	32.52
200_07	0.00	0.00	0	0	986.47	28.65
200_08	8.00	0.00	8	8	45.98	31.10
200_09	10.00	0.00	10	10	988.06	34.00
200_10	19.09	0.29	19	20	998.96	32.68
300_01	2.15	0.77	0	4	990.38	65.29
300_02	12.02	0.14	12	13	749.79	66.38
300_03	13.06	0.24	13	14	858.31	67.32
300_04	8.16	0.81	7	10	966.18	65.99
300_05	32.28	1.10	29	35	1000	66.09
300_06	4.38	0.99	2	7	983.85	65.92
300_07	0.59	0.64	0	2	973.01	62.64
300_08	8.00	0.00	8	8	869.05	63.21
300_09	7.46	0.67	7	9	885.19	63.47
300_10	22.60	0.85	21	25	1000	63.91
400_01	2.52	0.63	1	4	695.43	96.84

400_02	17.37	0.90	16	20	1000	96.70
400_03	9.91	0.43	9	11	954.46	94.87
400_04	19.01	0.10	19	20	994.68	98.98
400_05	0.01	0.10	0	1	979.21	76.85
400_06	0.33	0.53	0	2	997.54	87.56
400_07	5.44	0.80	4	7	989.49	93.29
400_08	5.30	0.81	4	7	1000	93.39
400_09	7.63	0.95	5	10	1000	97.75
400_10	0.95	0.72	0	3	991.09	93.89

* Xeon 3.06Ghz chip and 1024 MB of RAM

4. CONCLUSIONS

We have presented two methods for the solution of the car sequencing problem: an exact method via ILP and an heuristic method via an ACO. Our objective was to compare the results found through these methods to those obtained for two sets of benchmark problems by other researchers who used either a CSP or various heuristic algorithms. With relative ease, the integer linear model was able to solve those problems for which there exists a solution presenting no conflicts in the option constraints. However, despite improvements in the power of commonly available computers and in the efficiency of commercially available linear solvers, for those problems for which there is no known conflict-free solution, the ILP model did not find the optimal solutions. Using the objective-function values of the best known solutions as a bound in the ILP, we were able to confirm these solutions themselves. While we expect that there will be further improvements in both computers and linear solvers, given that the problem itself is NP-hard, we may also expect that ILP will not be able to solve what may be considered to be the more difficult problems.

Concerning the results produced by the ACO, we note that, with quite short computing times, we were able to produce solutions of a quality equal to that of the best known solutions for all of the benchmark problems. Even where the ILP model was able to solve the benchmark problem, the ACO found the same optimal solution more rapidly. In comparison with other search heuristics, it is difficult to make meaningful comparisons because of the small size and low level of difficulty of the problems. We have therefore constructed, solved and rendered available a new and more difficult set of benchmark problems that we hope will also be solved by other researchers using their own methods. In this way more significant comparisons may emerge.

We have used the same “sliding window” method of evaluation of solutions that is described in the existing literature, but we suggest that this method should be revised. This method tends to amplify the number of option constraint violations by double counting. Consider, for example, an option that should be present in only one car in three in the sequence. By the current method of evaluation, the sequence (1 0 1 1 0 0) is said to have three violations in the windows (1 0 1), (0 1 1) and (1 1 0). In fact, the same two

cars cause the latter two conflicts and removing either one or the other of them from this part of the sequence will solve both conflicts. In actual fact, the assembly line will be overloaded at only two points in time and not three as the evaluation method seems to indicate. It would seem advisable to revise the evaluation method to remove this bias because solutions with the same apparent number of violations might in fact have different impacts on the shop floor.

In future work, it would seem to us important that the dynamic aspects of such a production plan should be considered. The current approach does not recognize that the solution retained for the current period will have an impact on the following planning period, which must start up in the state where the current period has left it. One remedy would be to continue the evaluation of the sequence beyond the last "sliding window" of length s_k with the hypothesis that cars in the following period require no options.

For future work, it would be interesting to include additional elements drawn from the actual industrial context. For example, one might seek to include the impact of the constraints in the paint shop which precedes the assembly line. Moreover, in scheduling overall operations, the constraints of the assembly line are often divided into priority and non-priority constraints. The multi-objective aspects implicit in scheduling both painting and assembly render the actual industrial problem more complex than the idealized form considered in the literature and they must be adequately dealt with to have an impact in industry. Finally, the sequencing problem that we have presented may be seen in a broader frame such as that described by Lockledge *et al.*[22].

REFERENCES

1. Dincbas M., Simonis H. and Van Hentenryck P. (1988). Solving the car-sequencing problem in constraint logic programming. In: Y. Kodratoff (ed.), Proceedings of the European Conference on Artificial Intelligence (ECAI-88), Munich, Germany, Pitmann Publishing, London, pp 290-295.
2. Kis T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters* **32**(4): 331-336.
3. Lopez P. and Roubellat F. (2001). *Ordonnancement de la production*. Paris, Hermès science publications, 431 pages.
4. Solnon C. (2000). Solving car sequencing problems with artificial ants. In: H. Werner (ed.), European Conference on Artificial Intelligence (ECAI-2000), Berlin, Germany, IOS Press, pp 118-122.
5. Gottlieb J., Puchta M. and Solnon C. (2003). A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In: G. R. Raidl *et al.* (eds.), Applications of Evolutionary Computing, Lecture Notes in Computer Science, Springer-Verlag Heidelberg, pp 246-257.
6. Davenport A., Tsang E., Zhu K. and Wang C. (1994). GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement, In: Proceedings of AAAI'94, Seattle, Washington, AAAI Press, Menlo Park, California, pp 325-330.

7. Davenport A. and Tsang E. (1999). Solving constraint satisfaction sequencing problems by iterative repair, In: Proceedings of the First International Conference on the practical Applications of Constraint Technologies and Logic Programming, London, England, Practical Applications Company, pp 345-357.
8. Puchta M. and Gottlieb J. (2002). Solving Car Sequencing Problems by Local Optimization. In: S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. R. Raidl (eds), Proceedings EvoWorkshops, Lecture Notes in Computer Science 2279, Kinsale, Ireland, Springer, pp 132-142.
9. Lustig I. and Puget J. F. (2001). *Constraint Programming*. Encyclopedia of Operations Research and Management Science (Saul Gass and Carl M. Harris, eds), Kluwer Academic Publishers, Boston, Massachusetts, 917 pages.
10. Van Hentenryck P., Deville Y. and Yeng C. M. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* **57**(2-3): 291-321.
11. Van Hentenryck P. (1999). *The OPL Optimization Programming Language*. MIT Press, Cambridge, Massachusetts, 255 pages.
12. Lee J., Leung H. and Won H. (1998). Performance of a comprehensive and efficient constraint library using local search. In: G. Antoniou and J. K. Slaney (eds), 11th Australian Joint Conference on Artificial Intelligence, Brisbane, Australia, Springer-Verlag Heidelberg, pp 191-202.
13. Gent I. P. and Walsh T. (1999). *CSPLib: a benchmark library for constraints*. Research Reports of the APES Group, APES-09-1999, available from http://4c.ucc.ie/~tw/csplib/schedule.html.
14. Regin J. C. and Puget J. F. (1997). A Filtering Algorithm for Global Sequencing Constraints. In: G. Smolka (eds), Principles and Practice of Constraint Programming, Linz, Austria, Springer, pp 32-46.
15. Gent I. P. (1998). *Two Results on Car-sequencing Problems*. Research Reports of the APES Group, APES-02-1998, Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.
16. Boivin S. (2004). *Résolution d'un problème de satisfaction de contraintes pour l'ordonnancement d'une chaîne d'assemblage automobile*. Université du Québec à Chicoutimi.
17. Gagné C., Gravel M. and Price W. L. (2002). Algorithme d'optimisation par colonie de fourmis avec matrices de visibilité multiples pour la résolution d'un problème d'ordonnancement industriel. *Information Systems and Operational Research (INFOR)* **40**(2): 259-276.
18. Dorigo M. and Gambardella L. M. (1997). Ant colonies for the traveling salesman problem. *BioSystems* **43**: 73-81.
19. Dorigo M. (1992). *Optimization, learning and natural algorithms*. Italy, Ph.D. Thesis, Politecnico di Milano.
20. Bonabeau E., Dorigo M. and Theraulaz G. (1999). *Swarm intelligence: from nature to artificial systems*. New York, NY, Oxford University Press, 308 pages.
21. Dorigo M. and Stützle T. (2004). *Ant Colony Optimization*. Cambridge, MA, MIT Press, 328 pages.
22. Lockledge J., Mihailidis D., Sidelko J. and Chelst K. (2002). Prototype fleet optimization model. *Journal of the Operational Research Society* **53**: 833-841.