An algorithm for the car sequencing problem of the ROADEF 2005 Challenge

Max Risler
max.risler@karlshof.de

Marco Chiarandini
Luis Paquete
Tommaso Schiavinotto
Thomas Stützle
{machud,lpaquete,schiavin,tom}@intellektik.informatik.tu-darmstadt.de

TU Darmstadt, Computer Science, Intellectics Group
Alexanderstr. 10, 64283 Darmstadt, Germany

November 30, 2004

1 Introduction

The algorithm we present for solving the *Car Sequencing Problem* as defined in the *Roadef Challenge 2005* consists of the following components. First, a constructive heuristic is used to generate an initial sequence. Next, a local search is applied to improve the solution quality. Because a simple local search would stop in a local optimum as soon as no more improvements can be achieved with any single local exchange, we apply metaheuristics to help the underlying local search to escape from local optima and to find better solutions.

We implemented and tested about 10 different constructive heuristics. These heuristics have the goal to create a sequence that serves as a good starting point for the local search in a reasonable amount of time. Among the different algorithms tested we finally selected two, each of which performs best on a different type of problem instances. They are described in Section 2.

A local search is performed using a specific neighborhood structure, that is a set of neighbors for each sequence. Among the different neighborhood structures we tested, those which proved the most effective and which we use in the final algorithm are:

- exchanging two vehicles in the sequence (*exchange*),
- moving a vehicle to a different position (*insert*),
- exchanging two batches of vehicles having the same color (*batch exchange*).

We experimented with many different metaheuristics [5] like Iterated Local Search [4] and Iterated Greedy Search [6][1]. Our final algorithm described in

Section 5 is a hybrid method using Variable Neighborhood Search [2] which is described in Section 3, and Simulated Annealing [7][3], which is described in Section 4.

2 Creating initial solutions

In the following, we describe how the initial solution is created. The method to create the initial solution should be fast so that there is still plenty of time to improve the solution found by local search but is also should provide good quality solutions in the first place.

The two methods we found to perform best are fairly simple approaches. The first one is used for problem instances with highest priority to color changes or for instances where the ratio constraints are easy in average according to their utilization rate which is obtained from instance characteristics such as the number of associated vehicles, the number of total vehicles, and the ratio of the constraint. We use the second method for the problem instances where the highest priority is given to ratio constraints.

2.1 Insertion method

Algorithm 1 Insertion method to create starting sequence π

$\pi \leftarrow$ empty sequence
while $|\pi| < n$ **do**
  $v \leftarrow$ random vehicle $\notin \pi$
  choose position x where inserting v in π leads to minimal increase of the evaluation function value
  $\pi \leftarrow (\pi_1, \ldots, \pi_{x-1}, v, \pi_x, \ldots, \pi_{|\pi|})$
end while

In the insertion method we start from an empty sequence and then simply insert vehicles one by one at the best possible position. This is done by first computing for each position the resulting difference of the evaluation function then choosing the position with the minimal value. Ties are broken favorising the leftmost position. By using the evaluation function for comparing different insertion points, we do not need to treat the three objectives separately. As the evaluation function weights them differently the given priorities are respected correctly.

2.2 Sequential method

This method is very similar to the insertion method. The sequence is created by adding one vehicle after another but in this case the position for the next vehicle is fixed at the end of the sequence and instead the vehicle which is added is chosen according to the minimal evaluation function value.

Another difference to the insertion method is that for breaking ties in the choice of the next vehicle we always take the vehicle which has the most ratio constraints weighted by their difficulty associated. The idea is to sequence difficult vehicles as soon as possible without producing constraint violations.

Algorithm 2 Sequential method to create starting sequence π

```
π ← empty sequence
for all vehicles v do
    M_v ← Σ_{r∈R_v} M_r        % R_v ⊆ R ratio constraints associated with vehicle v
                               % M_r ratio constraint utilization rate r
end for
while |π| < n do
    choose vehicle v for which appending v to π leads to minimal increase of
    the evaluation function value, break ties in favor of highest M_v
    π ← (π, v)
    for all remaining vehicles v do
        update M_v according to changed ratio constraint difficulties
    end for
end while
```

To achieve this we use the utilization rates of the ratio constraints on each vehicle. After adding each vehicle these utilization rates have to be updated to reflect the correct ratio constraint difficulty for each of the remaining vehicles.

3 Variable Neighborhood Descent

Variable Neighborhood Descent(VND) is a metaheuristic which improves local search result by systematically changing the neighborhood structure. The neighborhood is changed repeatedly until the final solution is a local optimum for all neighborhoods. To achieve this first an order is defined on the neighborhood structures under consideration. Then, the first neighborhood structure for which a solution is no longer a local optimum is selected and local search is performed in this neighborhood until it stops. This is repeated until the solution is a local optimum with respect to all neighborhood structures.

Our algorithm applies VND to improve the initial solution before Simulated Annealing is performed as well as to improve specific solutions produced in the run of Simulated Annealing. We use *exchange*, *insert*, and *batch exchange* neighborhoods in that order. For the instances that have the least priority to paint color changes we omit batch exchanges, since good solutions for these instances tend to have relatively small batches of same-colored vehicles and therefore the batch exchange local search would take much time with only little effect.

4 Simulated Annealing

Simulated Annealing iteratively visits a sequence of neighboring solutions. In constrast to an iterative improvement local search not only improving moves are accepted, thus, making it possible to leave a local optimum. The neighborhood structure used here is the exchange-neighborhood. Repeatedly a random pair of vehicles is selected. If exchanging the two vehicles leads to an improvement the move is accepted. A worsening move may be accepted with a probability

$$p(\Delta) = e^{-\frac{\Delta}{T}},$$

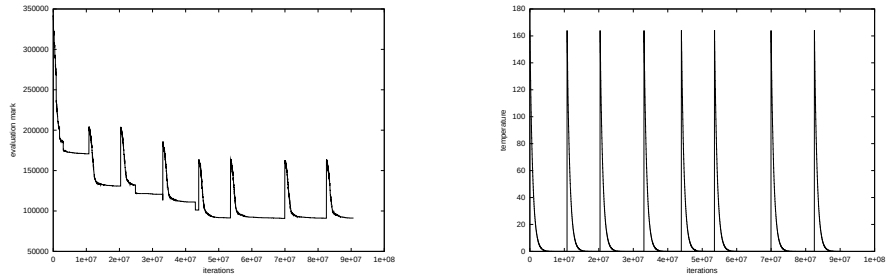

Figure 1: Profile of evaluation mark (left) and temperature (right) for one representative run of the Simulated Annealing algorithm on instance 024_38_3_EP_RAF_ENP.

where Δ is the evaluation function difference resulting from the exchange and T is a control parameter called temperature. The temperature is decreased over time. As a lower temperature results in a lower probability for accepting a worsening move, after a while almost only improving moves will be accepted. When the solution did not improve for a given number of iterations the temperature is increased. This re-heating allows to escape from attractive local minimum areas.

5 Final Algorithm

Algorithm 3 Final algorithm using Simulated Annealing and Variable Neighborhood Search

```
create initial sequence π
perform variable neighborhood search on π
T ← initial temperature
while time limit not reached do
  choose random 0 ≤ i < j < |π|
  Δ ← evaluation function delta for exchanging π_i and π_j
  if Δ ≤ 0 or e^(−Δ/T) > random[0,1) then
    π ← (π_1, ..., π_{i−1}, π_j, π_{i+1}, ..., π_{j−1}, π_i, π_{j+1}, ..., π_{|π|})
  end if
  update temperature T
  if re-heating occured then
    perform variable neighborhood search on π
  end if
  π_best ← best of π and π_best
end while
```

After running various experiments on different algorithms we decided on first starting from the solution we obtain from the described constructive heuristic. Next, one Variable Neighborhood Descent is run. Then Simulated Annealing is performed until the time limit is reached. Before re-heating another Variable

Instances with easy-to-satisfy highest priority ratio constraints

instance	results					median
022_3_4_EP_RAF_ENP	3601	3400	3202	3300	3301	3301
025_38_1_EP_ENP_RAF	14178	16806	15682	13913	13803	14178
025_38_1_EP_RAF_ENP	25832	25922	25726	26020	25871	25871
064_38_2_EP_RAF_ENP_ch1	13861	13761	14059	13860	13859	13860
064_38_2_EP_RAF_ENP_ch2	2976	2976	2893	2893	2899	2899
064_ch2_s24_mar	17261	17261	17261	17261	17261	17261
					average	**12895**

Instances with difficult-to-satisfy highest priority ratio constraints

instance	results					median
024_38_3_EP_ENP_RAF	42946	42221	41874	42031	42314	42221
024_38_3_EP_RAF_ENP	68259	71137	107853	88633	68435	71137
024_38_5_EP_ENP_RAF	46084	45898	46103	46042	45978	46042
024_38_5_EP_RAF_ENP	71007	70801	70797	90800	71130	71007
039_38_4_EP_RAF_ch1	145500	196000	144700	145100	165000	145500
048_39_1_EP_ENP_RAF	7080	7092	7020	7176	7373	7092
048_39_1_EP_RAF_ENP	19245	19256	19333	19347	18948	19256
048_ch2_s25_mar	8740	8640	8650	8542	8640	8640
					average	**51362**

Instances with highest priority to paint changes

instance	results					median
022_3_4_RAF_EP_ENP	113904	113905	113904	114005	114005	113905
039_38_4_RAF_EP_ch1	699600	699900	699700	699800	701800	699800
039_ch1_s26_mar	853644	853643	853648	853647	853643	853644
064_38_2_RAF_EP_ENP_ch1	673092	673090	673093	673090	673089	673090
064_38_2_RAF_EP_ENP_ch2	306752	306752	306752	306752	306752	306752
					average	**529438**

Table 1: Results obtained from 5 runs on each instance.

Neighborhood Descent is performed to ensure the solution is in a local optimum for all neighborhood structures. The algorithm continues until the time limit is reached and the best found solution is returned.

We implemented our algorithm in *C++*.

6 Analysis of experimental results

In order to tune the algorithm we had to find the values for several parameters, like initial temperature, cooling rate, and number of non-improving iterations before re-heating, etc.. This was done by running the algorithm with as many different parameter combinations as possible, and comparing quality and variance of the solutions found in dependence of the parameter values. As a result, we obtained three sets of parameter values, among which one gets selected depending on instance characteristics like objective order and problem size.

The results in Table 1 were computed on an *Athlon MP/1.2 Ghz/1 GByte RAM* machine running *Suse Linux 7.3*. Unfortunately we were not able to do the runs on the same machine as specified for the challenge but we were able to compare the runtime and therefore did the run with a time limit of 544 seconds to match the difference in processing speed.

References

[1] J.C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical Report 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992.

[2] P. Hansen and N. Mladenovic. Variable neighborhood search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics.*, pages 145–184. Kluwer Academic Publishers, Norwell, MA, 2002.

[3] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[4] H.R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics.*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[5] O. Rossi-Doria, M. Samples, M. Birattari, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquete, B. Paechter, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In E. Burke and P.D. Causmaecker, editors, *Practice and Theory of Automated Timetabling IV: 4th International Conference, PATAT 2002, Gent, Belgium, August 2002, Selected Revised Papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer-Verlag, Berlin Heidelberg, 2003.

[6] R. Ruiz and T. Stützle. Iterated greedy for the permutation flow shop problem. Manuscript, 2004.

[7] V. Černý. Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal Of Optimization Theory And Applications*, pages 41–51, 1985.