Multiple Variable Neighborhood Search Enriched with ILP Techniques for the Periodic Vehicle Routing Problem with Time Windows*

Sandro Pirkwieser and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{pirkwieser|raidl}@ads.tuwien.ac.at

Abstract. In this work we extend a VNS for the periodic vehicle routing problem with time windows (PVRPTW) to a multiple VNS (mVNS) where several VNS instances are applied cooperatively in an intertwined way. The mVNS adaptively allocates VNS instances to promising areas of the search space. Further, an intertwined collaborative cooperation with a generic ILP solver applied on a suitable set covering ILP formulation with this mVNS is proposed, where the mVNS provides the exact method with feasible routes of the actual best solutions, and the ILP solver takes a global view and seeks to determine better feasible route combinations. Experimental results were conducted on newly derived instances and show the advantage of the mVNS as well as of the hybrid approach. The latter yields for almost all instances a statistically significant improvement over solely applying the VNS in a standard way, often requiring less runtime, too.

1 Introduction

The *periodic vehicle routing problem with time windows* (PVRPTW) is a generalized variant of the classical *vehicle routing problem with time windows* (VRPTW) where customers must be served several times in a given planning period instead of only once on a single day. Applications exist in many real-world scenarios as in courier services, grocery distribution, or waste collection.

The PVRPTW is defined on a complete directed graph $G = (V, A)$ with $V = \{0, 1, \ldots, n\}$ being the set of vertices and $A = \{(i, j) \mid i, j \in V, i \neq j\}$ the set of arcs. A planning horizon of t days, referred to by $T = \{1, \ldots, t\}$, is considered. Vertex 0 represents the depot with time window $[e_0, l_0]$ at which are based m vehicles having capacities $Q_1, \ldots, Q_m$ and maximal daily working times $D_1, \ldots, D_m$. Each vertex $i \in V_C$, with $V_C = V \setminus \{0\}$, corresponds to a customer and has associated a demand $q_i \geq 0$, a service duration $d_i \geq 0$, a time window $[e_i, l_i]$, a service frequency f_i, and a set $C_i \subseteq T$ of allowed combinations of visit days. Each arc $(i, j) \in A$ has assigned a travel time (cost) $c_{ij} \geq 0$. The challenge

* This work is supported by the Austrian Science Fund (FWF) under contract number P20342-N13.

consists of selecting one visit combination per customer and finding (at most) m vehicle routes on each of the t days on G such that

- each route starts and ends at the depot,
- each customer i belongs to f_i routes over the planning horizon,
- for each vehicle $k = 1, \ldots, m$, the total demand of each route does not exceed capacity limit Q_k, and its daily duration does not exceed the maximal daily working time D_k,
- the service at each customer i begins in the interval $[e_i, l_i]$ and every vehicle leaves the depot and returns to it in the interval $[e_0, l_0]$, and
- the total travel cost of all vehicles is minimized.

Arriving before e_i at a customer i implies a waiting time until this start of the time window (without further cost). Arriving later than l_i is not allowed, i.e. we assume hard time window constraints. In this work, we further assume a homogeneous vehicle fleet with $Q_1, \ldots, Q_m = Q$ and $D_1, \ldots, D_m = D$.

This article introduces a multiple variable neighborhood search (VNS) variant, where several cooperative VNS instances are running in an intertwined way. To obtain even better results, we further consider an *integer linear programming* (ILP) approach based on a set covering formulation and a column generation algorithm and hybridize it in a collaborative way with the multiple VNS.

We refer to related work in Section 2. The variable neighborhood search and its multiple variant are described in Section 3, the ILP formulation in Section 4, and the proposed hybrid method in Section 5. Experimental results are given in Section 6, and Section 7 finishes the work with concluding remarks.

2 Related Work

The PVRPTW was first addressed in [1], where a tabu search algorithm is described for it. In our previous work [2] we suggested a variable neighborhood search (VNS), outperforming the former tabu search. Related VNS metaheuristics exist for the multi-depot VRPTW [3] and the *periodic vehicle routing problem* (PVRP) [4]. Earlier results of our current work, where only a standard VNS (i.e. a single search trajectory) has been combined with a column generation approach and a different cooperation mechanism was used, have been described in [5]. We are not aware of other exact or hybrid methods for the PVRPTW, yet similar PVRPs are dealt with in [6] and [7].

A more general survey of different PVRP variants and solution methods is given in [8]. A similar idea as the one followed in this work was recently applied to a ready-mixed concrete delivery problem [9]. Our work also extends this by highlighting further aspects of this kind of hybridization. In a somewhat related work Danna and Le Pape [10] apply an ILP solver for deriving improved integer solutions during a branch-and-price procedure. For a more general overview on ILP/metaheuristic hybrids we refer to [11].

Although we do not explicitly consider parallelization in this work, our intertwined approach shares features with the replicated parallel VNS variant introduced among other approaches in [12] and also applied in [13].

3 VNS for the PVRPTW

Variable neighborhood search (VNS) [14] is a metaheuristic that applies random steps in neighborhoods with growing size for diversification, referred to as shaking, and uses an embedded local search component for intensification. It has been successfully applied to a wide range of combinatorial optimization problems.In the following we give a rather short overview on our VNS for the PVRPTW as it has been already described in detail in [2].

To smooth the search space, the VNS relaxes the vehicle load, route duration, and time window restrictions and adds penalties corresponding to the excess of these constraints to the cost function. (All three kinds of penalty terms are weighted by a constant factor of 100.) The creation of the initial solution was kept quite simple by selecting a single visit day combination per customer at random and afterwards partitioning the customers at each day into routes. This partitioning is performed by sorting the customers according to the angles they make with the depot—ties are broken using the center of the time windows $(e_i + l_i)/2$—and inserting the customers in this order and a greedy fashion into at most m routes. This insertion is performed in such a way that all but the last routes of each day will comply to the load and duration constraints, while time window constraints might be violated by all routes. The procedure is similar to those introduced in [1].

In the shaking phase we utilize three different neighborhood structures, each with six moves of increasing perturbation size, yielding a total of 18 shaking neighborhoods (i.e. $k_{\max} = 18$) : (i) randomly changing up to six visit combinations with greedy insertion for the new visit days, whereas we also allow reassigning the same visit combination, (ii) moving a random segment of up to six customers of a route to another one on the same day, and (iii) exchanging two random segments of up to six customers between two routes on the same day. In the latter two cases the segments are occasionally reversed. In this work we only consider a fixed shaking neighborhood order.

For intensification we apply the well-known 2-opt intra-route exchange procedure in a best improvement fashion, only considering routes changed during shaking. Additionally each new incumbent solution is subject to a 2-opt* inter-route exchange heuristic [15]. Hereby for each pair of routes of the same day all possible exchanges of the routes' end segments are tried.

To enhance the overall VNS performance not only better solutions are accepted, but sometimes also solutions having a worse objective value. This is done in a systematic way using the Metropolis criterion like in simulated annealing [16]. A linear cooling scheme is used in a way such that the acceptance rate of worse solutions is nearly zero in the last iterations.

3.1 Multiple VNS

We extend the traditional VNS, which only has a single search trajectory, by considering multiple cooperating VNS instances performed in an intertwined way. Thus, our concern here is to investigate the possible benefits of a *sequential*

Algorithm 1: Multiple VNS: $\#VNS$ refers to the number of VNS instances, $\#sec$ to the number of sections per VNS instance, and $iter_{max}$ is the total number of allowed iterations.

```
for i = 1 to #VNS do
    initialize VNS[i];
iter_sec ← ⌈iter_max/(#VNS · #sec)⌉;
for sec = 1 to #sec do
    for i = 1 to #VNS do
        execute VNS[i] for iter_sec iterations;
    x ← best solution of all VNS instances;
    Replace solution of worst VNS instance by x;
Return best solution of all VNS instances;
```

cooperative multistart search. This new VNS variant is denoted as *multiple VNS* (mVNS).

Although it would be straight-forward to parallelize this approach, parallelization is not the issue we want to focus on here. A somewhat related approach is *replicated parallel VNS* [12, 13], in which multiple VNS instances are performed independently in parallel; the overall best solution is finally returned. In this case, the gain in performance is (almost) entirely due to the parallelization. In contrast, we aim at achieving better results within the same total CPU-time as required by a simple VNS run.

The multiple VNS algorithm is shown in Algorithm 1. We initialize each VNS instance independently by performing the method for creating a random solution 100 times and taking the best solution. This way each VNS instance most likely starts with a different initial solution. In the following the VNS instances are executed section-wise by setting an appropriate iteration limit given the total iteration limit and the number of sections. After each block of section-wise executions the actual best solution is determined and replaces the solution of the worst VNS instance. The latter is the cooperative part, where, considered locally, a worse performing VNS is supported by the best one, and seen from a global perspective, the search is intensified in the neighborhood of the so far best solution.

In some sense VNS instances can be said to be adaptively allocated to promising areas of the search space: If a solution is best after one iteration of the outer loop, one additional search trajectory is started from it. If the solution remains the incumbent over further iterations, more VNS instances are restarted from this point and a corresponding stronger intensification takes place. If, however, a new incumbent is found, no further VNS instances will be restarted from the previous one. Of course, in the unlucky event of a very captious local optimum this behavior could lead to a situation where all VNS instances are restarted from the same solution and no further progress is achieved, though this would be no worse than in the single VNS instance case.

In Section 6 we will experimentally compare this multiple VNS to a standard VNS execution and see the advantages w.r.t. final solution quality. The next section introduces a column generation based ILP approach for the PVRPTW problem with which we hybridize the multiple VNS in Section 5 to achieve even better results.

4 Set Covering ILP Model for the PVRPTW

We express the PVRPTW by the following set covering model:

$$\min \sum_{\tau \in T} \sum_{\omega \in \Omega} \gamma_\omega \, \chi_{\omega\tau} \tag{1}$$

$$\text{s.t.} \quad \sum_{r \in C_i} y_{ir} \geq 1 \qquad \forall i \in V_C \tag{2}$$

$$\sum_{\omega \in \Omega} \chi_{\omega\tau} \leq m \qquad \forall \tau \in T \tag{3}$$

$$\sum_{\omega \in \Omega} \alpha_{i\omega} \, \chi_{\omega\tau} - \sum_{r \in C_i} \beta_{ir\tau} \, y_{ir} \geq 0 \qquad \forall i \in V_C; \forall \tau \in T \tag{4}$$

$$y_{ir} \in \{0,1\} \qquad \forall i \in V_C; \forall r \in C_i \tag{5}$$

$$\chi_{\omega\tau} \in \{0,1\} \qquad \forall \omega \in \Omega; \forall \tau \in T \tag{6}$$

The set of all feasible routes (satisfying the first, third, and fourth condition from our problem definition in Section 1) visiting a subset of customers is denoted by Ω. Obviously, this set is exponentially large w.r.t. the instance size. For each route $\omega \in \Omega$, let γ_ω be the corresponding costs. We introduce binary variables $\chi_{\omega\tau}$ indicating whether or not route ω is used on day τ, $\forall \omega \in \Omega$, $\tau \in T$. Furthermore, for each customer $i \in V_C$, binary variables y_{ir} indicate whether or not visit combination $r \in C_i$ is chosen. The objective function (1) corresponds to the total costs of all selected routes. Cover constraints (2) guarantee that at least one visit day combination is selected per customer, fleet constraints (3) restrict the number of daily routes to not exceed the available vehicles m, and visit constraints (4) link the routes and the visit combinations, whereas $\alpha_{i\omega}$ and $\beta_{ir\tau}$ are binary constants indicating whether or not route ω visits customer i and if day τ belongs to visit combination $r \in C_i$ of customer i, respectively.

Due to the huge amount of variables it is not possible to directly solve this ILP formulation for instances of practical size. *Column generation*, however, provides a reasonable way to approach such situations [17]: One starts with a small set of initial variables (routes, corresponding to columns in the matrix notation of the ILP) and iteratively extends this set by adding potentially improving variables. In each iteration, the linear programming (LP) relaxation of this reduced problem is (re-)solved in order to finally obtain the solution to the whole LP and thus a lower bound for the original ILP.

This procedure can in principle be combined with branch-and-bound to derive integer solutions, too (and eventually prove their optimality). However, this

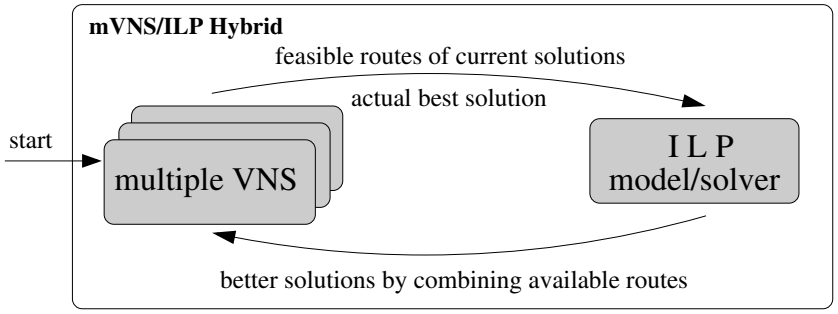

Fig. 1. Information exchange between multiple VNS and the ILP model/solver.

is another line of research followed by us which turns out to be applicable in a reasonable time to relatively small instances only (roughly about 30 customers at the time of writing). Here, we use the multiple VNS as the sole provider of columns for the set covering model, and restricted ILPs are solved via the general purpose ILP solver CPLEX. This hybrid method is explained in the next section.

5 Hybridizing the Multiple VNS with the ILP Approach

The motivation for the hybrid method is to exploit feasible routes of VNS solutions to derive a new and better solution by applying an ILP solver on the set covering ILP model of Section 4 enriched by these routes. Therefore a pool of solutions' routes is gathered at certain times and provided to the ILP model. This ILP is solved by a branch-and-bound based generic ILP solver, gradually fixing the visit combination (5) and route variables (6). A similar approach was introduced in [9], where the authors highlight the "global view" property of such an exact model. For a set of solutions' routes the ILP solver might be able to derive a more favorable combination and provide a better (less costly) solution in this way. Obviously the potential of the ILP solver depends on the routes contained in the model since it is neither able to alter routes nor to create some on its own. Hence it is crucial to provide (i) a suitable amount of routes, (ii) cost-effective routes, and (iii) diverse enough routes. Adding not enough or only weak routes usually will prevent the ILP solver from finding a better solution at all; on the other hand, a too large set naturally increases the runtime, which might also prevent finding better solutions quickly enough in case a time limit is given. Therefore routes to be added to the model must be carefully selected.

We apply the following hybrid scheme, which can be regarded an intertwined collaborative cooperation [11]; see Figure 1 and Algorithm 2. Concerning the hybridization with the multiple VNS a natural and suitable way is to apply the ILP solver after a block of section-wise executions. This way the number of solutions is given by the number of VNS instances, from which we use the actual best solutions. Due to different search trajectories these solutions' routes are

assumed to be diverse enough. Hence, conditions (i)–(iii) are fulfilled. Contrary to [9] we restrict ourselves to the actual best solutions, but we have more VNS instances available. The ILP solver is allotted the same amount of CPU-time than the multiple VNS.

The application of the ILP solver can in some way be regarded as a recombination operator taking into account all available solutions provided by the "population" of the VNS instances. In case a solution is not feasible as a whole, its feasible routes are added anyway, whereas the ILP solver is only applied if at least one feasible solution exists. An additional point of concern is the route injection scheme: It is possible to either add the route for the corresponding day only or for all days. The latter scheme would produce significantly larger ILP models (of factor t) which might yield better solutions at the expense of longer runtimes to solve the ILP model. However, preliminary results as well as results in our previous work [5] suggest to inject routes for the corresponding day only. Further, the ILP solver is always initialized with the current best solution to speed up the process.

If the ILP solver is able to improve on the current best solution this new solution is transferred to the multiple VNS, where as usual the solution of the worst VNS instance is replaced. Before such a transfer eventually over-covered solutions are repaired by choosing exactly one visit combination (the first active) and omitting customers from following routes if they are already covered or do not need to be visited on this day. This over-covering might happen since we use a set covering model. In contrast, a set partitioning model (derived by turning inequalities (2) and (4) into equalities) would yield only feasible solutions but at the same time exclude many potentially improving combinations. Finally, before injecting this solution the previously mentioned 2-opt* improvement procedure is also applied to it. In case routes were altered during this transfer process, corresponding new columns are also added to the ILP model.

There are also two options regarding the lifetime of the routes added to the ILP model: Either we only consider the actual solutions' routes, i.e. they are discarded afterwards, or we keep all inserted routes and the ILP model gradually grows, i.e. realizing a long term memory. However, it is clear that a model of continuously increasing size in general demands more and more computation time to be solved, which could in turn lead to worse solutions when setting a time limit as in our case although the larger search space might also contain better solutions. If routes are kept then the ILP solver is only applied if non-existing routes could be added after a multiple VNS section. Both variants are examined in Section 6.

6 Experimental Results

The algorithms have been implemented in C++, compiled with GCC 4.1 and executed on a 2.83 GHz Intel Core2 Quad Q9550 with 8 GB RAM. We derived new PVRPTW instances from the Solomon VRPTW benchmark instances[1] by

[1] available at `http://web.cba.neu.edu/~msolomon/problems.htm`

Algorithm 2: Multiple VNS / ILP Hybrid: $\#VNS$ refers to the number of VNS instances, $\#sec$ to the number of sections per VNS instance and to the maximal number of ILP solver applications, and $iter_{max}$ is the total number of allowed iterations.

```
Ω' ← ∅; // start with empty model
for i = 1 to #VNS do
    initialize VNS[i];
iter_sec ← ⌈iter_max/(#VNS · #sec)⌉;
for sec = 1 to #sec do
    Ω'_sec ← ∅;
    for i = 1 to #VNS do
        execute VNS[i] for iter_sec iterations;
        add VNS[i] solutions' routes to Ω'_sec; // gather columns
    x* ← actual best solution;
    Ω' ← Ω' ∪ Ω'_sec; // enrich ILP model
    x ← apply ILP solver on Ω', initialized with x*;
    Replace solution of worst VNS instance by x;
Return best solution of all VNS instances;
```

evenly assigning the possible visit combinations to the customers at random. We did so for the first five instances of type random (R), clustered (C), and mixed random and clustered (RC) for a planning horizon of four ($t = 4$) and six days ($t = 6$), denoted by p4 and p6, respectively. For a planning horizon of four days the customers need to be visited either once, two, or four times, for a planning horizon of six days once, two, three, or six times. The number of vehicles m was altered (reduced) in such a way that few or none empty routes occur in feasible solutions, yet it is not too hard to find feasible solutions quite early in the solution process. All instances contain 100 customers and the capacity constraint was left untouched.

For the standard VNS we set an iteration limit of either 10^6 or $2 \cdot 10^6$, an initial temperature of 10 and apply linear cooling every 100 iterations. The multiple VNS as well as the mVNS/ILP hybrid are also allowed 10^6 VNS iterations in total and $\#sec$ is consistently set to 10 (as determined by preliminary tests); i.e. they apply ten sequences of $\#VNS$ VNS instances, each one running for $10^6/(\#VNS \cdot 10)$ iterations per section. For solving the ILP model in the mVNS/ILP hybrid we apply the general purpose MIP solver ILOG CPLEX 11.2.

Both mVNS variants were initially run with 5, 10, and 15 VNS instances, these are denoted by $\text{mVNS}_{\#VNS,\#sec}$ and $\text{mVNS/ILP}_{\#VNS,\#sec}$. Each algorithm setting is run 30 times per instance and we report average results, stating the average travel costs (avg.), corresponding standard deviations (sdv.) and average CPU-times in seconds (t[s]).

The results of the standard VNS are given in Table 1, where we also state the number of vehicles m; this information is omitted in the remaining tables. As expected, the double amount of iterations also approximately doubles the CPU-

time, and for all but instances p4r103 and p6r105 improvements are achieved. In the following the newly introduced algorithm variants will be compared to this "baseline".

Table 1. Results of standard VNS on derived periodic Solomon instances with a planning horizon of four and six days.

Instance		VNS (10^6)			VNS ($2 \cdot 10^6$)		
Id	m	avg.	sdv.	t[s]	avg.	sdv.	t[s]
p4r101	14	4141.04	22.52	22.6	4134.47	18.81	44.9
p4r102	13	3759.61	19.43	23.5	3756.77	15.45	46.6
p4r103	10	3191.59	13.56	24.2	3194.75	14.84	47.9
p4r104	7	2613.83	15.76	27.2	2604.74	12.94	54.1
p4r105	11	3697.78	13.84	24.0	3692.96	14.74	47.7
p4c101	10	2910.72	0.53	22.1	2910.53	0.37	44.0
p4c102	8	2963.50	34.01	24.7	2960.70	37.37	48.8
p4c103	7	2806.39	40.13	27.5	2793.80	30.71	54.6
p4c104	7	2481.58	18.14	26.4	2476.27	19.95	52.5
p4c105	8	3025.67	82.95	24.3	2973.57	42.71	48.3
p4rc101	10	4003.77	12.01	25.6	4001.34	14.66	50.7
p4rc102	10	3814.02	19.59	25.3	3798.00	17.47	50.5
p4rc103	8	3500.84	29.40	27.5	3494.06	23.22	55.0
p4rc104	7	3069.41	16.62	27.9	3058.48	18.83	55.4
p4rc105	11	4008.80	25.16	24.5	4001.89	20.06	48.9
p6r101	14	5418.76	10.31	25.9	5417.67	20.27	51.4
p6r102	12	5276.07	23.34	27.0	5261.35	17.65	53.7
p6r103	9	4035.13	28.85	28.8	4014.16	21.37	57.7
p6r104	8	3389.61	16.03	29.5	3380.17	13.35	58.8
p6r105	9	4355.02	27.43	28.3	4355.28	31.80	56.6
p6c101	7	4084.67	36.86	30.1	4076.20	34.75	59.8
p6c102	7	3888.96	20.98	29.7	3876.88	17.27	59.2
p6c103	6	3616.61	44.79	33.8	3583.02	33.04	66.9
p6c104	6	3295.32	18.09	32.8	3291.93	16.76	64.6
p6c105	7	4164.39	64.90	29.9	4139.66	53.95	59.3
p6rc101	10	5846.32	24.69	27.4	5833.84	26.41	54.5
p6rc102	9	5483.15	26.63	28.2	5473.67	24.34	56.1
p6rc103	7	4370.85	25.63	32.2	4360.17	21.22	64.1
p6rc104	7	4147.69	26.00	31.4	4127.46	23.06	63.0
p6rc105	9	5340.30	22.08	28.9	5330.60	19.67	57.7

Tables 2 and 3 show the results of the mVNS as well as the mVNS/ILP hybrid for the p4 and p6 instances, with the setting of storing all injected routes. At the bottom of each table we additionally state how often the hybrid variant was significantly better or worse than the multiple VNS, as well as how many

times both variants were significantly better or worse than the standard VNS variants, whereas we used a Wilcoxon rank sum test with an error level of 5% for testing statistical significance. Best average results are marked bold. Looking at the mVNS it is already quite often better than the standard VNS with 10^6 iterations (up to 80% for p4 instances and 73% for p6 instances) as well as the VNS with $2 \cdot 10^6$ iterations (p4: 60%, p6: 33%), achieving this without additional CPU-time consumption. However, combining the mVNS with ILP techniques in the mVNS/ILP hybrid consistently yields even more satisfying results. In case of mVNS/ILP$_{5,10}$ this is also possible without considerable increase in CPU-time. Although for the variants with $\#\,VNS$ set to 10 and 15 the runtime approaches that of the VNS with $2 \cdot 10^6$ iterations (which is per setting the upper limit) for some instances, and thus the comparison to this latter VNS variant is fairer in some sense. For the p4 instances this still results in a 100% success rate, whereas for the p6 instances mVNS/ILP$_{10,10}$ performs best and is 9 times (60%) better and once worse (6.6%) than VNS with $2 \cdot 10^6$ iterations. Looking at the results of the p6 instances, we decided to fine-tune the number of VNS instances, assuming that the performance peak is somewhere between 5 and 10 instances. Therefore we conducted experiments with $\#\,VNS$ ranging from 6 to 9; see Table 4. As can be observed the best results are obtained with mVNS/ILP$_{8,10}$, which is, among other settings, always better than the pure mVNS—of course also consuming more CPU-time—and is 12 times (80%) better than the VNS with $2 \cdot 10^6$ iterations, and again only once worse (6.6%). In general, the mVNS/ILP hybrid achieves to a large extent significantly better results than the latter standard VNS yet it still consumes very often less CPU-time.

So far we only considered the strategy to keep all routes in the model once they were added. The results of the hybrid algorithm when resetting the columns after each application of the ILP solver are given in Table 5. Due to space limitations we only state the results of the statistical significance tests. For the p4 instances there is clearly no gain, whereas not storing the routes has the greatest impact when using 15 VNS instances, i.e. when most columns are added. Here, the reduced size of the model leads to more improvements in the limited time. Nevertheless, apart from less runtime in total for obvious reasons, it seems generally better to work with an ILP model of increasing size and hence exploit information from the search trajectory.

7 Conclusions

We extended our previously introduced (standard) VNS for the periodic vehicle routing problem with time windows (PVRPTW) to a multiple VNS (mVNS) where several VNS instances are applied cooperatively in an intertwined way. This mVNS puts emphasis on the so far best solution found within a major iteration by restarting the worst performing VNS instances with it. In this way, mVNS investigates multiple search trajectories from incumbent solutions, and from a global perspective it can be seen to adaptively allocate VNS instances to promising areas of the search space. Further an intertwined cooperative combina-

Table 2. Results of mVNS and mVNS/ILP on periodic Solomon instances with a planning horizon of four days.

Instance	$mVNS_{5,10}$			$mVNS/ILP_{5,10}$			$mVNS_{10,10}$			$mVNS/ILP_{10,10}$			$mVNS_{15,10}$			$mVNS/ILP_{15,10}$		
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]
p4r101	4119.88	16.75	22.9	4114.01	13.89	23.3	4121.55	12.42	23.1	4096.09	10.49	26.5	4118.60	12.07	23.2	**4090.09**	5.29	29.0
p4r102	3744.40	6.12	23.8	3744.65	8.15	24.3	3741.79	5.01	24.1	3735.85	4.00	26.6	3742.85	5.03	24.2	**3732.34**	1.94	30.9
p4r103	3186.14	11.19	24.3	3174.98	8.97	25.3	3184.83	9.37	24.8	3171.48	8.44	28.4	3186.70	7.27	24.8	**3165.72**	6.95	33.9
p4r104	2602.15	11.10	27.7	2600.89	10.49	28.9	2602.87	8.63	27.9	**2595.44**	8.20	42.7	2605.36	8.81	28.3	2598.18	10.62	56.0
p4r105	3688.64	12.86	24.1	3681.23	10.05	27.3	3687.94	8.92	24.5	**3679.66**	11.93	48.5	3690.29	8.24	24.6	3686.14	9.16	49.3
p4c101	2910.17	0.26	22.2	2910.24	0.27	22.9	2910.06	0.44	22.6	2909.72	0.67	23.7	2909.91	0.72	22.7	**2909.39**	0.76	24.7
p4c102	2929.76	17.88	24.7	2934.88	23.11	25.2	2930.78	19.63	25.3	2917.46	18.25	28.7	2921.32	20.02	25.4	**2905.16**	14.38	36.4
p4c103	2786.69	31.78	27.8	2774.61	23.01	28.5	2783.91	28.61	28.4	2762.38	18.47	38.6	2777.30	20.18	28.7	**2759.78**	13.82	51.8
p4c104	2465.70	11.65	26.9	2459.19	8.17	27.8	2468.14	10.72	27.0	2459.52	7.60	41.9	2471.83	11.24	27.1	**2454.69**	12.32	49.2
p4c105	2962.17	34.34	24.6	2957.97	36.68	24.8	2952.66	27.78	24.8	2924.55	19.26	31.6	2942.26	29.05	25.0	**2906.69**	15.55	37.6
p4rc101	3988.69	11.61	25.8	3989.75	9.45	26.3	3992.15	10.29	26.1	3978.48	8.95	33.7	3996.60	11.92	26.2	**3974.09**	6.40	46.8
p4rc102	3806.24	17.11	25.6	3796.09	17.08	26.4	3802.65	16.02	26.1	3778.23	14.71	33.9	3801.01	14.74	26.1	**3764.99**	6.76	43.4
p4rc103	3494.62	18.55	27.9	3484.76	18.97	28.5	3500.27	17.08	28.2	3475.95	16.58	31.0	3502.18	23.94	28.3	**3466.99**	12.01	34.7
p4rc104	3042.86	11.52	29.1	3036.39	10.56	30.2	3048.23	11.97	29.2	3036.61	12.43	54.1	3051.19	11.53	29.3	**3031.49**	17.15	57.0
p4rc105	3995.56	13.49	25.1	3991.06	17.18	25.5	3997.55	12.10	25.2	3976.16	8.09	30.1	4000.98	10.76	25.3	**3970.49**	5.67	45.0
significantly better/worse than corresponding mVNS																		
				7×/0×						**15**×/0×						**15**×/0×		
significantly better/worse than VNS (10^6)																		
	12×/0×			**15**×/0×			12×/0×			**15**×/0×			12×/0×			**15**×/0×		
significantly better/worse than VNS ($2 \cdot 10^6$)																		
	7×/1×			12×/0×			9×/0×			**15**×/0×			7×/0×			**15**×/0×		

Table 3. Results of mVNS and mVNS/ILP on periodic Solomon instances with a planning horizon of six days.

Instance	$mVNS_{5,10}$			$mVNS/ILP_{5,10}$			$mVNS_{10,10}$			$mVNS/ILP_{10,10}$			$mVNS_{15,10}$			$mVNS/ILP_{15,10}$		
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]
p6r101	5402.01	8.78	26.4	5399.76	9.80	27.8	5404.53	7.85	26.6	5390.21	6.10	37.8	5402.38	6.81	26.8	**5385.03**	3.33	49.0
p6r102	5255.09	14.27	27.5	**5244.59**	12.11	32.5	5252.94	11.79	27.9	5249.80	13.81	55.3	5255.77	13.88	28.0	5249.56	14.74	55.6
p6r103	4011.33	20.59	29.4	**3991.46**	12.83	32.5	4006.43	13.54	29.6	3996.78	16.43	58.0	4015.02	14.71	30.0	4008.83	18.72	59.8
p6r104	3376.35	12.41	30.0	3373.36	12.14	34.3	3377.63	10.21	30.4	**3372.81**	9.75	59.1	3382.94	9.40	30.5	3381.98	12.65	60.1
p6r105	4340.76	15.94	28.8	**4337.54**	18.15	31.3	4346.55	15.84	29.2	4337.64	17.22	58.2	4350.98	15.18	29.0	4353.84	18.24	59.4
p6c101	4078.48	27.72	30.6	4060.83	37.72	31.4	**4049.95**	24.47	30.9	4055.21	25.91	59.2	4054.29	20.17	31.1	4050.81	26.17	61.8
p6c102	3867.79	16.38	30.6	3868.96	13.08	32.2	3866.77	13.95	30.6	3861.91	10.82	58.7	**3861.62**	10.81	30.8	3861.86	8.69	62.2
p6c103	3584.92	23.11	34.6	3583.20	29.79	37.5	3589.74	24.60	34.7	**3576.50**	21.80	67.6	3587.16	16.01	35.1	3580.92	18.52	70.3
p6c104	3286.67	15.15	33.5	**3284.07**	15.72	37.5	3287.39	15.14	33.6	3289.73	13.06	65.6	3293.76	17.19	34.1	3291.96	11.87	68.0
p6c105	4124.30	33.03	30.5	4112.89	38.21	31.4	4115.20	27.39	30.8	4119.17	39.47	60.0	4118.48	29.05	30.6	**4104.31**	23.19	61.5
p6rc101	5830.37	13.51	28.0	5831.77	13.67	31.7	5833.97	15.10	28.3	**5821.63**	14.27	56.2	5839.42	14.70	28.5	5833.46	16.74	57.0
p6rc102	5449.68	23.63	29.0	5466.58	25.31	33.0	5468.39	31.07	29.2	**5446.00**	25.84	58.5	5474.09	27.07	29.6	5460.70	30.92	59.0
p6rc103	4366.40	14.85	32.6	**4351.50**	18.34	35.6	4373.90	19.24	33.0	4359.78	18.02	65.1	4373.31	15.87	33.2	4366.85	13.43	66.1
p6rc104	**4130.70**	17.69	31.9	4132.90	21.11	38.3	4146.00	17.74	32.0	4139.72	16.05	63.8	4154.20	18.96	32.3	4149.87	15.64	64.2
p6rc105	5331.66	22.97	29.5	**5321.82**	17.66	32.4	5339.50	14.64	29.5	5329.84	18.28	59.2	5346.52	16.70	30.0	5340.75	15.67	59.7
significantly better/worse than corresponding mVNS																		
				4×/1×						**9**×/0×						4×/0×		
significantly better/worse than VNS (10^6)																		
	11×/0×			**15**×/0×			10×/0×			13×/0×			7×/0×			10×/0×		
significantly better/worse than VNS ($2 \cdot 10^6$)																		
	5×/0×			8×/0×			4×/3×			**9**×/1×			4×/3×			6×/2×		

Table 4. Results of mVNS and mVNS/ILP on periodic Solomon instances with a planning horizon of six days.

Instance	$mVNS_{6,10}$			$mVNS/ILP_{6,10}$			$mVNS_{7,10}$			$mVNS/ILP_{7,10}$			$mVNS_{8,10}$			$mVNS/ILP_{8,10}$			$mVNS_{9,10}$			$mVNS/ILP_{9,10}$		
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]	avg.	sdv.	t[s]
p6r101	5403.42	8.92	26.2	5397.93	7.14	27.9	5402.42	7.19	26.2	5395.88	5.64	28.7	5400.68	7.10	26.3	5392.52	5.13	30.6	5406.09	7.49	26.3	**5391.93**	5.16	33.2
p6r102	5250.62	13.72	27.4	5247.88	14.15	40.2	5251.83	14.73	27.3	5247.96	15.05	50.3	5256.14	14.34	27.4	**5244.00**	13.18	53.7	5251.95	13.12	27.5	5252.02	15.60	54.8
p6r103	4004.61	15.35	29.0	4002.86	16.69	35.3	4009.22	15.93	29.1	3996.57	11.56	39.3	4010.68	10.51	29.1	3998.23	16.13	48.2	4014.87	17.70	29.5	**3996.23**	14.52	54.2
p6r104	3376.94	7.32	29.8	3375.07	10.21	36.4	3381.93	11.76	29.8	3372.69	10.20	44.3	3378.82	10.88	29.8	**3370.82**	11.53	50.0	3383.11	13.68	30.0	3379.34	10.80	56.4
p6r105	4342.18	16.27	28.8	4334.24	15.18	33.5	4344.48	17.94	28.7	**4328.45**	16.34	41.2	4343.22	17.48	28.9	4338.28	14.92	52.3	4344.83	16.69	28.8	4335.15	20.68	55.1
p6c101	4066.66	25.84	30.4	4059.06	24.12	32.6	**4051.95**	31.83	30.4	4056.32	23.01	38.9	4055.49	25.70	30.4	4058.62	33.58	43.4	4059.66	25.49	30.5	4052.32	26.96	51.9
p6c102	3871.16	15.55	30.2	3870.10	14.69	37.5	3868.40	11.06	30.4	**3864.41**	11.47	39.0	3866.25	12.30	30.4	3864.94	12.46	46.4	3868.40	15.05	30.4	3867.13	11.67	54.2
p6c103	3581.14	21.95	34.3	3575.80	24.17	41.0	3593.04	28.02	34.2	3579.95	23.88	48.7	3589.50	22.25	34.5	**3574.98**	24.81	54.6	3579.45	19.34	34.6	3578.30	22.87	61.9
p6c104	3291.55	21.77	33.1	3282.65	16.62	39.6	3287.61	13.54	32.9	3280.17	14.42	46.3	**3278.92**	18.69	33.4	3282.54	15.49	52.8	3288.56	16.01	33.2	3288.17	15.70	60.1
p6c105	4124.76	28.06	30.2	4113.98	27.74	34.8	4110.27	26.88	30.2	4111.19	26.09	40.9	4112.76	25.38	30.2	**4108.25**	27.22	47.8	4111.91	26.67	30.2	4122.30	30.33	57.7
p6rc101	5832.36	14.05	27.7	5823.02	12.67	40.2	5830.79	13.48	28.0	5821.85	13.31	50.0	5834.87	10.53	28.0	**5815.80**	16.78	53.7	5834.19	14.66	28.0	5820.97	17.15	55.4
p6rc102	5464.50	23.69	28.8	5447.06	25.85	37.8	5464.60	25.56	28.8	**5440.55**	27.78	44.1	5461.56	25.45	28.9	5448.11	30.07	53.0	5460.10	22.01	29.0	5442.78	23.82	56.9
p6rc103	4369.68	15.29	32.7	4356.50	15.68	37.5	4365.63	19.79	32.6	**4352.63**	19.56	45.8	4367.29	14.22	32.6	4353.06	19.43	54.2	4368.73	16.70	32.7	4357.15	17.52	60.8
p6rc104	4136.32	17.98	31.6	**4134.89**	20.02	46.7	4146.65	15.45	31.4	4139.70	18.34	55.1	4146.49	14.88	31.6	4136.09	19.17	62.3	4142.93	14.39	31.8	4143.03	17.67	62.9
p6rc105	5326.71	17.63	29.2	5324.04	19.89	37.4	5331.00	20.33	29.4	5323.60	22.41	46.4	5337.35	16.03	29.3	**5318.69**	16.99	54.8	5338.34	18.61	29.5	5325.33	22.58	58.0
significantly better/worse than corresponding mVNS																								
				7×/0×						**15**×/0×						**15**×/0×						**15**×/0×		
significantly better/worse than VNS (10^6)																								
	12×/0×			**15**×/0×			9×/0×			14×/0×			11×/0×			**15**×/0×			9×/0×			13×/0×		
significantly better/worse than VNS ($2 \cdot 10^6$)																								
	2×/2×			10×/1×			5×/2×			11×/1×			6×/1×			**12**×/1×			6×/1×			8×/1×		

Table 5. Results of mVNS/ILP on periodic Solomon instances with a planning horizon of four and six days when resetting the columns.

Instances	$\text{mVNS/ILP}_{5,10}$	$\text{mVNS/ILP}_{10,10}$	$\text{mVNS/ILP}_{15,10}$
significantly better/worse than corresponding mVNS			
p4	4×/0×	10×/0×	14×/0×
p6	2×/1×	4×/0×	9×/0×
significantly better/worse than VNS (10^6)			
p4	14×/0×	15×/0×	14×/0×
p6	14×/0×	12×/0×	11×/0×
significantly better/worse than VNS ($2 \cdot 10^6$)			
p4	10×/0×	13×/0×	13×/0×
p6	6×/1×	6×/1×	8×/1×

tion of this mVNS and a generic ILP solver applied to a suitable set covering ILP formulation was proposed. The mVNS provides the exact method with feasible routes of the actual best solutions, and the ILP solver takes a global view and seeks to determine better feasible route combinations. For testing we derived new PVRPTW instances with a planning horizon of four and six days from the 100 customer Solomon VRPTW benchmark instances. Experimental results showed the advantages of the mVNS as well as of the hybrid approach, the latter yielding for 80%–100% of all conducted tests a statistically significant improvement over solely applying the VNS in a standard way. It has become evident that keeping the routes (columns) in the model once they were added is beneficial, though one has to keep in mind the longer runtimes of this setting than when considering the actual best solutions' routes only. Nevertheless, even the mVNS/ILP hybrid with this continuously increasing ILP model—clearly performing best of all variants—requires for most of the instances less CPU-time than the standard VNS with more iterations.

As future work we might consider some sort of column management for the ILP to have an additional option in-between resetting the columns or persistent storage. Also of interest for the mVNS would be a special perturbation operator (probably a more destructive shaking) in case the same local optimal solution is injected more than once in a VNS instance. From a practical perspective, dealing with customer demands depending on the visit day would be interesting, too. Last but not least, we want to remark that the general approach described here also might be promising for many other combinatorial optimization problems.

References

1. Cordeau, J.F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society **52** (2001) 928–936

2. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In Prodhon, C., et al., eds.: Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France (2008)
3. Polacek, M., Hartl, R.F., Doerner, K., Reimann, M.: A variable neighborhood search for the multi depot vehicle routing problem with time windows. Journal of Heuristics **10** (2004) 613–627
4. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research **195**(3) (2009) 791–802
5. Pirkwieser, S., Raidl, G.R.: Boosting a variable neighborhood search for the periodic vehicle routing problem with time windows by ILP techniques. In Caserta, M., Voß, S., eds.: Proceedings of the 8th Metaheuristic International Conference (MIC 2009), Hamburg, Germany (2009)
6. Francis, P., Smilowitz, K., Tzur, M.: The period vehicle routing problem with service choice. Transportation Science **40**(4) (2006) 439–454
7. Mourgaya, M., Vanderbeck, F.: Column generation based heuristic for tactical planning in multi-period vehicle routing. European Journal of Operational Research **183**(3) (2007) 1028–1041
8. Francis, P.M., Smilowitz, K.R., Tzur, M.: The period vehicle routing problem and its extensions. In Golden, B., et al., eds.: The Vehicle Routing Problem: Latest Advances and New Challenges. Springer (2008) 73–102
9. Schmid, V., Doerner, K.F., Hartl, R.F., Savelsbergh, M.W.P., Stoecher, W.: A hybrid solution approach for ready-mixed concrete delivery. Transportation Science **43**(1) (2009) 70–85
10. Danna, E., Le Pape, C.: Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In Desaulniers, G., et al., eds.: Column Generation. Springer (2005) 99–129
11. Raidl, G.R., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In Blum, C., et al., eds.: Hybrid Metaheuristics: An Emerging Approach to Optimization. Volume 114 of Studies in Computational Intelligence. Springer (2008) 31–62
12. García-López, F., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M.: The parallel variable neighborhood search for the p-median problem. Journal of Heuristics **8**(3) (2002) 375–388
13. Moreno-Pérez, J.A., Hansen, P., Mladenović, N.: Parallel variable neighborhood search. In Alba, E., ed.: Parallel Metaheuristics: A New Class of Algorithms. John Wiley & Sons, NJ, USA (2005) 247–266
14. Hansen, P., Mladenović, N.: Variable neighborhood search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Boston MA (2003) 145–184
15. Potvin, J.Y., Rousseau, J.M.: An exchange heuristic for routeing problems with time windows. Journal of the Operational Research Society **46** (1995) 1433–1446
16. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598) (1983) 671–680
17. Desrosiers, J., Lübbecke, M.E.: A primer in column generation. In Desaulniers, G., et al., eds.: Column Generation. Springer (2005) 1–32