Vera Hemmelmayr

Sequential and parallel large neighborhood search algorithms for the periodic location routing problem

Article (Accepted for Publication)
(Refereed)

Sequential and Parallel Large Neighborhood Search Algorithms for the Periodic Location Routing Problem

Vera C. Hemmelmayr

WU (Vienna University of Economics and Business),
Welthandelsplatz 1, 1020 Vienna, Austria
vera.hemmelmayr@wu.ac.at

We propose a large neighborhood search (LNS) algorithm to solve the periodic location routing problem (PLRP). The PLRP combines location and routing decisions over a planning horizon in which customers require visits according to a given frequency and the specific visit days can be chosen. We use parallelization strategies that can exploit the availability of multiple processors. The computational results show that the algorithms obtain better results than previous solution methods on a set of standard benchmark instances from the literature.
Keywords: location routing, large neighborhood search, parallel metaheuristics

1 Introduction

The periodic location routing problem (PLRP) is an important problem in supply chain management that combines location decisions with routing decisions and visit day assignment. A set of customers has to be served with a predefined frequency from a set of capacitated depots. Although the frequency is given, the exact service days have to be determined. Furthermore, there is a fixed cost that has to be paid for each vehicle used over the planning horizon. The objective is to minimize travel costs, opening costs of the depots and fixed costs of the used vehicles.

Solving location and routing decisions simultaneously is particularly appealing in problems where these two decisions are on the same decision level. This is frequently the case in problem settings where location decisions are easier to modify, for example, when they involve rented locations or do not require big investments. Furthermore, in many problem settings the depot costs can be broken down to the considered planning horizon so that they are at the same order of magnitude as the routing costs. Location routing problems are also interesting from a strategic point of view where building detailed routes can give a better approximation of future

routing costs. An extensive literature review including a description of different applications for location routing and a classification scheme is given in Nagy and Salhi (2007). More recent surveys can be found in Prodhon and Prins (2014), Drexl and Schneider (2013) and Lopes et al. (2013).

The PLRP was first introduced in Prodhon (2008) and it is related to the periodic vehicle routing problem (PVRP) and the location routing problem (LRP). The PVRP deals with serving a set of customers over a given planning horizon. Customers have predefined frequencies and a related set of predefined visit combinations.

The PVRP is a well-studied problem and a number of mainly heuristic solution methods have been proposed to solve it. A recent exact method was proposed in Baldacci et al. (2011a). Recent meta-heuristic approaches can be found in Gulczynski et al. (2011), where integer programming and the record-to-record travel algorithm were combined, and in Cordeau and Maischberger (2012), where a parallel iterated tabu search heuristic was proposed. Finally, in Vidal et al. (2012) evolutionary search, local search and elaborate population-diversity management schemes were combined.

When the visit frequency is not given, but must be decided, the problem results in a PVRP with service choice (Francis et al. 2006). However, in this paper we assume that the visit frequencies are given and only the particular visit days can be chosen from a set of visit day combinations.

Moreover, the PLRP can be seen as an extension of the LRP to a planning horizon. In the LRP, a set of possible capacitated depots is given to serve customer demand. The goal is to simultaneously select a subset of these depots to open and to solve the corresponding multi-depot vehicle routing problem (MDVRP) such that the routing cost and the opening cost of the depots are minimized. Recent heuristic methods for the LRP with capacitated depots and vehicles include Hemmelmayr et al. (2012), Contardo et al. (2013b), Escobar et al. (2013) and Ting and Chen (2013). Exact solution methods were developed in Baldacci et al. (2011b), Belenguer et al. (2011) and Contardo et al. (2013a).

There are only a few papers in the literature that combine location and routing decisions over a planning horizon in which customers have to be served multiple times. Albareda-Sambola et al. (2012) tackled a multiperiod location routing problem where location and routing decisions are made at different time scales. In this problem, decisions regarding facility location can only be made in selected time periods of the planning horizon and cannot be changed during time periods between them. Furthermore, unlike in the PLRP, the customer specifies the exact time periods of service.

Previous solution methods for the PLRP include an iterative heuristic (Prodhon 2008), a memetic algorithm with population management (Prodhon and Prins 2008) and an ELS with path relinking (Prodhon 2009). These methods are outperformed in Prodhon (2011) by a later method, an hybrid evolutionary algorithm (HELS). In this algorithm, an evolutionary local search (ELS) works on the selection of visit day combinations. The ELS is hybridized with a randomized extended Clarke and Wright algorithm (RECWA), which was originally designed for the LRP (Prins et al. 2006). It is used for each day separately. Then, the depots that are opened for all days of the planning period are chosen according to different ratios that measure

the suitability of opening a depot based on the percentage of daily use and the percentage of total demand satisfied per depot.

Pirkwieser and Raidl (2010) developed a variable neighborhood search (VNS) algorithm combined with ILP-based very large neighborhood searches for the LRP and the PLRP. The VNS uses neighborhoods that change visit combinations for customers, exchange segments of customers between routes of the same and of different depots and change the location decisions by opening or closing depots. They introduced three different ILP-based large neighborhood searches. The first neighborhood search, V_1, operates on a route level, in which depots can be opened or closed and routes can be relocated to different depots on the same day. The second approach, V_2 , also allows changing the visit day combinations of customers. The routes used in V_2 come from a set of feasible solutions of the VNS. The third approach is similar to ILP-based refinement techniques (De Franceschi et al. 2006). It removes sequences of customers from given routes and an ILP is used to find the optimal insertion points. This approach is solved for each day. These results were further improved in the thesis (Pirkwieser 2012) by adapted parameter settings, for instance a longer runtime.

Parallel computing enables the development of fast and robust solution methods for instance by exploiting the availability of multiple processors on computing clusters or multi-core processors. In the following, we will describe the most recent parallelization strategies for vehicle routing problems. For further information on parallel metaheuristics, we refer to the book of Alba (2005) and to the survey of Crainic and Toulouse (2010). Moreover, a survey with a focus on vehicle routing problems that covers heuristic and exact parallel solution methods can be found in Crainic (2008).

Crainic and Nourredine (2005) introduced a classification for parallel metaheuristics along three dimensions. The first dimension indicates whether the search is controlled by a master process (1C) or by several processes together (pC). The second dimension indicates the quantity and the quality of information exchanged. There is rigid (RS) and knowledge synchronization (KS) for synchronous communication and collegial (C) and knowledge collegial (KC) for asynchronous communication, where the respective difference is the amount and quality of information exchanged. The third dimension can be classified as SPSS, SPDS, MPSS or MPDS standing for same or multiple starting point and same or different search strategies used by the processes.

Cordeau and Maischberger (2012) proposed a parallel iterated tabu search heuristic for four different routing problems: the VRP, the PVRP, the MDVRP, and the site dependent VRP with and without time windows. In their parallel algorithm, each process starts from a different initial solution. Some of the parameters of the iterated tabu search are chosen independently in the processes. At given points, knowledge about the solutions is shared. Each process $p \in \{1, 2, ..., N\}$ decides whether to accept its working solution with a probability $1 - (\lambda/\eta)^2$, where λ is the current iteration and η the total number of iterations, or go to the j-th best solution, where $j = \lfloor \sqrt{p} \rfloor$. Therefore, most working solutions will be accepted in the beginning, while towards the end the best solutions will be accepted, so that the processes can focus on trying to improve the best solution. The authors can show that the algorithm yields very good results for several variants of vehicle routing

problems.

Groër et al. (2011) developed a parallel algorithm for the VRP that combines integer programming and heuristic search. A master process is used to control the search, while the remaining processes can be either heuristic solvers or set covering solvers. The heuristic solvers run a metaheuristic based on the record-to-record travel algorithm and the set covering solvers solve set covering problems with routes taken from the heuristic solvers.

Jin et al. (2012) designed an algorithm that achieves very competitive results for the VRP. They use four tabu search threads that each use different neighborhoods. The best solutions found are exchanged periodically through the use of a solution pool.

Lahrichi et al. (2012) developed integrative cooperative search (ICS) that can tackle multi-attribute combinatorial optimization problems. ICS performs a decomposition of the problem in partial problems that are solved by partial solvers. Integrators select partial solutions and combine them to complete solutions. These complete solutions are sent to the complete solver group. In this work, they apply the method to the MDPVRP and get results that improve upon previous methods.

In the context of parallel computing, it is also important to mention GPU based computing. It takes advantage of the rapid increase in GPU performance. Modern commodity PCs include a multi-core CPU and one or more GPUs. For this parallel, heterogeneous architecture, solution methods are developed. An introduction to modern computer architectures and GPU programming is given in Brodtkorb et al. (2013), which is part one of a survey in two parts. Part two (Schulz et al. 2013) gives a broad overview of the existing literature on parallel computing in discrete optimization aimed at modern PCs. The survey has a strong focus on routing problems. The authors conclude that GPU computing in discrete optimization is still in its infancy. The development of solution methods that exploit the heterogeneity of modern PCs efficiently is still an open research field that is interesting and highly relevant.

In this paper, we propose sequential and parallel variants of a large neighborhood search algorithm (LNS) to solve the PLRP. The computational results show that our algorithm outperforms previous solution methods in terms of solution quality. We can show that for standard benchmark instances from the literature, substantial improvements are possible both in the average solution quality as well as in the quality of best known solutions found. Moreover, we also develop two parallel versions of the algorithm that make use of the availability of clusters of computers. We propose a simple methodology for parallelization that can easily be applied to other problems and algorithms.

The remainder of this paper is organized as follows. Section 2 gives a detailed problem description, in Section 3 the solution method is outlined, while in Section 4, the computational experiments are described. Finally, Section 5 concludes the paper.

2 Problem Description

The PLRP can be defined on an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The set V consists of two subsets of nodes: the subset I of possible depot locations and the subset J of customers. Each depot has

a capacity W_i and an opening cost O_i, that is charged once in the planning horizon when the depot is opened. The traveling cost c_{ij} between node i and node j is given.

A planning period of several days H is considered in which customer demand has to be satisfied. Each customer j has a given visit frequency f_j. Associated with this visit frequency is a set of service day combinations C_j, i.e., the specific days on which the customer can be visited. For example, in a five day planning period, a customer with frequency one can be visited either on day one, two, three, four or five. For each customer, the total demand over the planning horizon, which is assumed to be cyclic, is given. The demand of customer j on day t of service combination $r \in C_j$ is given by d_{jtr}, which is the accumulated demand since the last service.

There is a maximum number of vehicles K. Vehicles are homogenous, capacitated and each vehicle is with a fixed cost when it is used. The vehicle cost is charged if a vehicle is used at least one time in the horizon. Each vehicle can only make a single route per day and must come back to its depot of departure. Let R_{it} be the number of routes assigned to depot i on day t, then the fleet size N_i of depot i is $N_i = \max\{R_{it} : t \in H\}$. The total number of vehicles required is $\sum_{i \in I} N_i$. Consider for example a solution with two open depots and two days, in which, on the first day, two routes originate from depot one and one from depot two, while on the second day, one originates from depot one and two from depot two. In that case four vehicles are necessary since for both depots, the number of vehicles required is two respectively.

In the PLRP the following decisions have to be made: selecting which depots to open, which service combination must be assigned to each customer and how many vehicles to assign to each depot such that customer demand is satisfied. The objective function considers the minimization of the sum of the routing cost, of the vehicle cost, and of opening cost of depots. Finally, the capacity constraints of depots and vehicles are respected, and customers are visited on feasible visit day assignments.

3 Solution Method

We developed a LNS algorithm to solve the problem. LNS was proposed by Shaw (1998), and is similar to the ruin and recreate principle proposed in Schrimpf et al. (2000). An overview of the LNS heuristic and its variants and extensions can be found in Pisinger and Ropke (2010).

The general idea of LNS is the usage of large neighborhoods that are usually composed of destroy and repair operators. A destroy operator ruins part of the solution while a repair operator is used to reconstruct it. In case of the PLRP, the destroy operators remove customers and put them to the temporary customer pool. Then the customers are reinserted in the partial solution by the repair operators. An extension of LNS, adaptive large neighborhood search (ALNS) was proposed by Ropke and Pisinger (2006) for the pickup and delivery problem. The difference is that the operators are chosen based on a score that reflects the past success of these operators.

We use several different types of destroy and repair operators that are explained below. Contrary to ALNS, in our LNS implementation, the operators are chosen randomly in each iteration no matter how successful or unsuccessful they were in

previous iterations.

The initial solution is constructed by assigning each customer a random combination, and by opening a random depot. The number of depots to be opened is a random number between the total demand divided by the average depot capacity, which should reflect an estimation of the minimum number of depots needed, and the total number of depots. Then, customers are assigned one by one to their closest depot and for each depot a set of routes is constructed by using Clarke and Wright (1964) Savings Algorithm. The depot capacity constraint can be violated in this step.

We allow the constraints on the number of vehicles, the depot capacity and the vehicle capacity to be violated. Therefore, a weighted penalty function is applied. The objective function is given by $f(s) = c(s)+\alpha d(s)+\beta e(s)+\gamma g(s)$. The travel cost, opening cost of depots and fixed cost for each vehicle used over the planning horizon are captured in $c(s)$. The violations of the vehicle capacity, number of vehicles, and satellite capacity constraints are $d(s)$, $e(s)$ and $g(s)$ respectively, while α, β and γ represent the corresponding weights. These weights are adjusted during the search process. Whenever a violation occurs, the respective weight is multiplied by a factor $\delta > 1$, when the solution is feasible with respect to the respective constraint, its weight is divided by δ as long as the weight remains in the interval $[\iota; \kappa]$. In a feasible solution $d(s)$, $e(s)$ and $g(s)$ equal zero.

3.1 Destroy operators

Each operator l removes a number of q_l customers, where q_l is a random uniformly distributed integer between ρ_l and τ_l. We use in total eight destroy operators that can be differentiated by the level on which they operate. There are three destroy operators that change the depot configuration (*close-depot*, *open-depot*, *swap-depot*). There are two destroy operators that assign new visit day combinations to customers (*change-combination*, *multiday-route-removal*) and finally there are three destroy operators that only operate on a given day (*related-removal*, *worst-removal*, *route-removal*).

The operator *close-depot* closes a random depot and moves all customers currently assigned to it to the customer pool. The operator *open-depot* opens a randomly selected depot among the ones currently closed and the q_l closest customers to this depot are moved to the customer pool. In the operator *swap-depot* one depot is closed, while another one is opened. All customers assigned to the closed depot are removed. The new depot to be opened is chosen in a roulette wheel selection based on the inverse distance to the closed depot.

Change-combination and *multiday-route-removal* remove q_l customers and reassign new visit day combinations to them. While *change-combination* removes random customers, *multiday-route-removal* tries to decrease the number of vehicles used by removing routes and assigning customers previously associated with these routes to new visit days. The vehicle to be removed is the vehicle that is needed the least number of days, where ties are broken arbitrarily. All customers that are assigned to these routes are removed and are assigned new visit day combinations.

The remaining three destroy operators only operate per day, i.e., customers are not assigned a new visit combination, but are simply removed from their current position on a given day. For these operators, a day is chosen randomly in a roulette

wheel selection. The probability that a day is chosen is proportional to the total demand on that day.

In *related-removal*, a random customer is picked as a seed customer and the $q-1$ closest customers are removed from their positions on the given day. The q worst customers are removed from their positions in *worst-removal*. Worst refers to customers for which the difference in cost of the solution with the customer to the solution without the customer is high. This difference, the so-called removal cost, is divided by the average travel cost of the ingoing arcs of the corresponding node. Furthermore, the removal cost is perturbed by a factor $d \in [0.8, 1.2]$ to randomize the search. The operators *related-removal* and *worst-removal* are modified versions of the operators suggested in Ropke and Pisinger (2006). In *route-removal*, a route is chosen randomly and all assigned customers are removed. The opening of a new route at that depot, on that day is forbidden, unless it is the only open depot. Preliminary experiments showed that all these operators are useful and needed to obtain good solutions. The operators *change-combination* and *multiday-route-removal* are new, the operators that change the depot configuration were previously introduced in Hemmelmayr et al. (2012) and the operators that only operate per day were also previously introduced in Hemmelmayr et al. (2012) and are slight modifications of the ones used in Ropke and Pisinger (2006).

3.2 Repair operators

The repair operators insert customers that have been removed by the destroy operators in the partial solution. We use one main operator, *greedy-insertion* and variations of it. It is based on the one in Ropke and Pisinger (2006).

The operator *greedy-insertion* inserts a customer in a new or existing route that minimizes the insertion cost, i.e., the sum of travel cost, depot opening cost and vehicle usage cost. This operator is a faster version of the one suggested in Ropke and Pisinger (2006). While in their paper, customers are ordered according to the minimum insertion cost, we use a random order because we wanted to have a fast and simple but still efficient operator. Hence, after the insertion of one customer, there is no need to update the insertion cost of the customers remaining in the list. There are two variations of the basic *greedy-insertion* operator. The operator *greedy-insertion-perturbation* uses a perturbation in the insertion cost as a diversification mechanism. Therefore, the insertion cost is perturbed by a factor $d \in [0.8, 1.2]$. After the *change-combination* and *multiday-route-removal* operators, the *greedy-insertion-multiday* version can be selected that computes the best insertion over all visit day combinations.

3.3 Local search

A local search procedure is performed for promising solutions, i.e, solutions that are within a threshold θ of the best found solution. The following operators are used for local search: *move*, *swap* and *2-opt*. The operators are performed sequentially for the daily VRPs of each open depot. Therefore, the assignment of customers to depots and also the assignment of visit day combinations are not changed during the local search phase. They are used in a first-improvement fashion as long as improvements can be realized. The *move* and *swap* operators are used inter-route and intra-route,

while *2-opt* is used intra-route. Move tries to relocated single customers. Swap exchanges two segments of customers. Segment lengths of one to three customers are possible and the two segments exchanged can have different lengths.

3.4 Parallelization strategies and different versions

We developed four different versions of the algorithm: two sequential and two parallel. The first version is the regular sequential version (LNS-S), which is shown in Algorithm 1. It starts from an initial solution. In each iteration, a destroy and a repair operator are chosen randomly until the stopping condition is reached. For the acceptance decision simulated annealing (SA) is used as a mechanism to also accept non-improving solutions. The parameters such as the temperature and the cooling scheme are described in section 4.1.

Algorithm 1 Basic steps of LNS-S

```
1: s, s* ← InitialSolution
2: repeat
3:     s' ← DestroyAndRepair(s, D, R)
4:     if f(s') < (1 + θ)f(s*) then
5:         s' ← LocalSearch(s')
6:     end if
7:     if AcceptanceDecision(s, s') then
8:         s ← s'
9:         s* ← min(s*,s)
10:    end if
11: until stopping condition is met
12: return s*
```

The second version is a hierarchical version (LNS-HIER-S). Algorithm 2 shows the basic steps. It is similar to LNS-S except that the problem is decomposed in two levels. The first decision level decides which depots to open, while in the second level the resulting MDPVRP is solved, with depots fixed to open or close according to the first stage decision. In the first level only destroy neighborhoods of the set D_f, that change the depot configuration, are applied. These are *swap-depot*, *open-depot* and *close-depot*. In the second level only the remaining destroy operators are used. The repair operators can be used in either level. The algorithm starts from the initial solution, performs one LNS iteration in the first level and fixes the depots to open or close accordingly. Then the second stage LNS is performed until the maximum number of second-stage iterations is reached and the solution is returned to the first level. Finally, in the first level, it is decided whether to accept the new incumbent or not and the next iteration starts. The acceptance decision in the first level is based on SA, while in the second level only improving solutions are accepted. The main idea of this version is that it follows a decomposition of the problem in a location part and a (MDPVRP) routing part.

The third version is a parallel hierarchical version (LNS-HIER-P) that parallelizes the second level of LNS-HIER-S. The first stage is computed by only one process, the

Algorithm 2 Basic steps of LNS-HIER

```
s, s* ← InitialSolution
repeat
    s̃ ← DestroyAndRepair(s,D_f, R)
    repeat
        s' ← DestroyAndRepair(s̃, D \ D_f, R)
        if f(s') < (1 + θ)f(s̃) then
            s' ← LocalSearch(s')
        end if
        s̃ ← min(s̃,s')
    until second level stopping condition is met
    if AcceptanceDecision(s, s̃) then
        s ← s̃
        s* ← min(s*, s)
    end if
until first level stopping condition is met
return s*
```

master process, while the second stage is computed by all processes in parallel. In the first stage, the solution is destroyed with an operator from the set D_f and repaired again. Using this solution as a starting solution, each process solves the resulting MDPVRP in the second stage until the maximum number of second stage iterations is reached. Once all processes are finished, the best solution among all processes is taken and returned to the first stage. There it is decided whether to accept it or not and the next iteration starts. Algorithm 3 shows a pseudocode for LNS-HIER-P in which P represents the number of processes. According to the classification in Crainic and Nourredine (2005) this is a 1C/KS/SPSS algorithm. There is 1-control by the master process and knowledge Synchronization (KS) because the master delegates a larger part of its work. Moreover, the algorithm uses same initial point, same search strategy, which means that the processes start from the same initial solution and use the same algorithm. This is a synchronous parallelization strategy in which all processes stop at predefined intervals to exchange information.

Finally, the fourth version is a parallelization of LNS-S called LNS-P, which is displayed in Algorithm 4. The processes start from different, randomly generated initial solutions and they all use different starting temperatures. Whenever a new best solution is found, it is communicated to central memory. After κ iterations without improvement, a new search segment starts. Then each process either continues with its current working solution or requests the current best solution from central memory. The working solution is kept with probability $1-(\lambda/\eta)^2$, where λ is the current iteration and η the total number of iterations. This acceptance decision is the same as in Cordeau and Maischberger (2012), but unlike them we do not consider the j-th best, but the global best solution. This makes sure that in the early stages of the algorithm, the processes are more likely to continue with their working solution, while in the end all processes try to improve the best solution. Moreover, towards the end of the search, after κ' iterations, the temperature is increased to

Algorithm 3 Basic steps of LNS-HIER-P

$s, s^* \leftarrow$ *InitialSolution*
repeat
  $\tilde{s} \leftarrow$ *DestroyAndRepair*(s,D_f, R)
  for each $p = 1, ..., P$ **do in parallel**
    $\tilde{s_p} \leftarrow \tilde{s}$
    repeat
      $s'_p \leftarrow$ *DestroyAndRepair*($\tilde{s_p}$, $D \setminus D_f$, R)
      if $f(s'_p) < (1+\theta) f(\tilde{s_p})$ **then**
        $s'_p \leftarrow$ *LocalSearch*(s'_p)
      end if
      $\tilde{s_p} \leftarrow min(s'_p, \tilde{s_p})$
    until second level stopping condition is met
  end for
  $\tilde{s} \leftarrow min_{p=1,...,P}(\tilde{s_p})$
  if *AcceptanceDecision*(s, $\tilde{s}$) **then**
    $s \leftarrow \tilde{s}$
    $s^* \leftarrow min(s^*, s)$
  end if
until first level stopping condition is met
return s^*

the initial temperature whenever a new search segment is started so that the search can escape from local optima in the final stages of the algorithm. To speed up the search and prevent that too much unnecessary information is being sent, a new best solution is only sent to central memory after an initial of 200 iterations and there is a minimum of 100 iterations after which a new best solution is send to central memory. This algorithm classifies (Crainic and Nourredine 2005) as pC/C/MPDS with p-control and an asynchronous collegial cooperation model through the use of a shared memory. Multiple points different strategies are used since each process uses its own starting solution and the strategies use a different parameter for the SA temperature. In this asynchronous communication each process is responsible for its own search and for the communication with other processes.

4 Computational Experiments

We have tested our algorithm on the set of 30 instances proposed in Prodhon (2008). The instances differ in the number of depots $m \in \{5, 10\}$, the number of customers $n \in \{20, 50, 100, 200\}$, vehicle capacity $Q \in \{70, 150\}$ and the number of clusters $\beta \in \{0, 2, 3\}$, where 0 corresponds to a uniform distribution. The instances are named accordingly: n-m-β-$[a, b]$, where a refers to the low capacity case (80) and b to the high capacity case (130). The planning horizon consists of five working days and two idle days. Each customer has a predefined visit frequency. There are three different possible frequencies: one, two or three. The corresponding visit day combinations for frequency one are {{1},{2},{3},{4},{5}}, for frequency two {{1,3}

Algorithm 4 Basic steps of LNS-P

$s^* \leftarrow DummyInitialSolution$
for each $p = 1, ..., P$ **do in parallel**
  $s_p, s_p^* \leftarrow InitialSolution$, $T_p \leftarrow initializeTemperature$
  repeat
    if κ iterations without improvement **then**
      if iter $\geq \kappa'$ **then**
        $T_p \leftarrow reheatTemperature()$
      end if
      if $requestGlobalBestCondition(\lambda, \eta)$ **then**
        $s_p \leftarrow requestGlobalBestSolution()$ /* get s^* */
      end if
    end if
    $s'_p \leftarrow DestroyAndRepair(s_p, D, R)$
    if $f(s'_p) < (1+\theta) f(s_p^*)$ **then**
      $s' \leftarrow LocalSearch(s'_p)$
    end if
    if $AcceptanceDecision(s_p, s'_p)$ **then**
      $s_p \leftarrow s'_p$
    end if
    if $f(s_p) < f(s_p^*)$ **then**
      $s_p^* \leftarrow s_p$
      $sendToCentralMemory(s_p^*)$ /* update s^* if improving */
    end if
    $iter \leftarrow iter + 1$
  until stopping condition is met
end for
return s^*

{1,5}{2,5}} and for frequency three {1,3,5}. The demand d_{jrt} of customer j on day t depends on the combination r chosen. We refer to Prodhon (2011) for a detailed description of how to compute d_{jrt} and for further details on the instances.

The algorithm was implemented in C++, compiled with Intel compiler v12.0 and run on a cluster of Intel Xeon X5670 at 2.93Ghz. The parallel versions were run with OpenMPI version 1.4.3. Four processors were used by LNS-HIER-P and seven by LNS-P.

We compared the algorithm to the best solution methods from the literature. These are the HELS proposed by Prodhon (2011) and the VNS with ILP-based neighborhoods of Pirkwieser and Raidl (2010) and Pirkwieser (2012).

4.1 Parameter Settings

The parameter settings were decided according to experiments on a subset of instances. The number of customers to remove in the destroy phase depends on the operator chosen. In every iteration, it is a random integer between ρ_l and τ_l, except

for the operators *close-depot*, *swap-depot*, *multiday-route-removal* and *route-removal*. For these operators the customers that are removed are the customers assigned to the depots or routes that will be closed. For the operator *open-depot*, the number is chosen between $0.2\frac{\#Visits}{T}$ and $0.6\frac{\#Visits}{T}$, where $\#Visits$ represents the total number of visits, which is the number of customers multiplied by their respective frequency, and T is the number of days in the planning period. For the operator *change-combination*, q_l is selected between 1 and $0.5N_c$, where N_c represents the customers that have more than one visit combination and hence can be considered in the operator *change-combination*. Finally, for the operators *related-removal* and *worst-removal*, q_l is between one and the number of customers visited on that day. These operators can remove a fairly large number of customers. Preliminary tests showed that removing a smaller number of customers yields worse results.

For the weighted penalty function the following parameters were chosen: δ was set to 1.1, ι to 5 and κ to 10,000. Our tests showed that changes in these parameters have only a minor influence on the performance of the algorithm.

The stopping condition of the algorithm is 5×10^6 iterations. The parallel version stops after every process has finished this number of iterations. For the hierarchial version, 500 iterations on the upper level and 10,000 iterations on the lower level are performed. The threshold θ that identifies a promising solution that will undergo local search is set to 5%.

The parameters used for the SA acceptance decision are set such that solutions that are worse than w% of the initial solution are accepted with a probability of 0.5, where w is set to 0.1. In the parallel implementation, LNS-P, the different processes select their w randomly between 0.01 and 0.3. The cooling is performed in a way that in every iteration the temperature T is decreased by $\frac{T_{start}}{\eta}$, where η is the total number of iterations. Preliminary experiments showed that the algorithm is insensitive to the cooling scheme used, but that it is necessary to use a mechanism that allows acceptance of non-improving solutions.

Moreover, a tabu list is used that forbids that an operator is used to open or close a depot that has just recently been opened or closed. The size of the tabu list is $\lceil 0.1m \rceil$, where m is the number of depots. So for the instances with 5 depots, there is no tabu list, for instances with 10 depots, the size of the tabu list is 1.

In the LNS-P algorithm, the number of iterations without improvement, κ, after which a new solution can be requested is set to 5000 and the number of iterations after which the temperature is increased to the initial temperature, κ', is 0.8η.

4.2 Results on the benchmark instances for the PLRP

We compare our solution method to the HELS algorithm of Prodhon (2011) and to the best performing method of Pirkwieser and Raidl (2010), which is the VNS+$V_{1,2}$. We will also use the results of Pirkwieser (2012), which are newer results of an improved version of the algorithm in Pirkwieser and Raidl (2010). These methods will be denoted as P (Prodhon 2011), PR (Pirkwieser and Raidl 2010) and PT (Pirkwieser 2012) in the sequel. In Prodhon (2011), the best value of the objective function over 10 runs is reported, while in Pirkwieser and Raidl (2010), the average solution over 30 runs is shown and in Pirkwieser (2012) the average cost over 10 runs as well as the minimum out of these 10 runs is given. The results for our algorithms are the average and minimum over 5 runs.

Concerning runtimes, the following computing environments were used: Pirkwieser and Raidl (2010) coded their algorithm in C++, compiled it with Intel compiler v12.0 and executed it on a single core of a Intel Core2 Quad Q9550 at 2.83 GHz with 8GB RAM. In Pirkwieser (2012), the algorithms were compiled with GCC 4.5 and executed on a single core of a 2.53 GHz Intel Xeon E5540 with 3GB RAM dedicated per core. Finally, Prodhon (2011) used an Intel Centrino 2 at 1.2 GHz and 1 GB of RAM for the experiments.

We will show the average and minimum results, if available, of the algorithms P, PR and PT and of our algorithm (version LNS-S). Table 1 shows the percentage gap to the new best known solutions for the average and the minimum of the respective algorithms. Furthermore, T^*, the time needed to find the best solution in each run, and T, the duration of one run of the algorithm until the maximum number of iterations, are given. Both T and T^* are averaged over five runs. From the previous algorithms, P is outperformed by PT and PR. As mentioned above, PT are newer results of a slightly improved version of PR with longer runtime. It yields better results than PR in terms of solution quality, but needs a longer runtime. LNS-S can achieve better results than PT. LNS-S needs a longer runtime, but it can reduce the average gap to the best known solutions substantially from 3.76% to 0.74%.

Table 1: Comparison of the algorithms P (Prodhon 2011), PR (Pirkwieser and Raidl 2010), PT (Pirkwieser 2012) and LNS-S in terms of %-gap to the best known solutions and runtime in seconds.

	P	PR	PT	LNS-S
$\%gap_{avg}$	–	6.80	3.76	0.74
$\%gap_{min}$	8.49	–	2.47	1.77
T^*	104.3	–	–	271.4
T	165.3	10.7	91.9	354.4

Table 2 shows an analysis of the insertion operators and local search. The table shows the % gap to a version of the algorithm that uses all the insertion operators and θ equals 5% (row LS: θ 5) and the runtime in seconds. The tests were performed for a subset of instances and results are averaged over these instances.

We tested different versions of the algorithm by varying the insertion operators used. By keeping everything else the same, there is one version without *greedy-insertion*, one without *greedy-insertion-perturbation* and one without *greedy-insertion-multiday*. The results show that the solution quality deteriorates slightly and the runtime increases slightly when the operators *greedy-insertion* and *greedy-insertion-perturbation* are left out. Omitting *greedy-insertion-multiday* results in a deterioration of 1% and is therefore not recommendable.

We also tested a version where the *regret-insertion* operator was used additionally to the three standard insertion operators. *Regret-insertion* was proposed in Ropke and Pisinger (2006). It computes a regret value that expresses the regret of not placing a customer in the best positions, but in the second best or third best and so on. Customers are then sorted according to their regret values in decreasing order, so that customers with a high regret are inserted earlier to avoid that their best insertion position is no longer available. After each insertion of a customer from the

list of untreated customers, the regret values of the remaining customers have to be updated by taking the respective changes of the insertion positions into account. This operator is more sophisticated than *greedy-insertion*, but it also needs a longer runtime to incorporate the updates performed. We saw that using *regret-insertion* does not improve the solution quality, but increases the runtime. Therefore, it was decided to omit this operator since its role can be replaced by LS.

Moreover, we also experimented with the usage of LS. LS is performed for any solution with an objective value less than or equal to $(100 + \theta)$ % of the best found solution so far. The table shows results for θ equals 0, 3, 5, and 10%. Please note that 5 is the default version. As can be seen, a solution where there is no LS performed, yields quite bad results. Moreover, there is a trade-off between runtime and solution quality that has to be considered when deciding for a more time-consuming insertion operator or to perform more LS. We found that θ equals 5 is a good parameter setting.

Table 2: Insertion operator and local search analysis

	%gap	T
no *greedy-perturbation*	0.10	494.4
no *greedy*	0.09	518.0
no *greedy-multiday*	1.00	417.8
with regret	0.04	1195.1
LS: θ 0	1.47	317.8
LS: θ 3	0.16	368.2
LS: θ 5 (default)	0.00	480.4
LS: θ 10	0.08	886.0

Table 3 shows results for the sequential and parallel variants compared to the best performing method by Pirkwieser (2012). LNS-HIER-P is run on four parallel processors and LNS-P is run on 7 parallel processors where one of the processors is managing the central memory. The total number of iterations is 5,000,000. We also report results for a shorter version of LNS-P with 714,500 iterations, where the number of iterations multiplied by the number of processors used roughly matches the number of iterations of the sequential version. The time T indicates wall-clock time, i.e., the time needed from start to the end of the parallel algorithm.

The hierarchical versions follow a natural decomposition of the problem. However, these versions are outperformed by their non-hierarchical counterparts. They need a longer runtime and have an inferior solution quality. In the upper level, the depot configuration is fixed while in the lower level the problem is solved for this given configuration. One disadvantage of this method is that it also explores "bad" depot configurations and hence wastes computation time exploring these regions. Moreover, the parallel version uses synchronous communication. Synchronous strategies have the disadvantage that there is a larger computational overhead, since all processes must wait for each other before they can proceed with their tasks.

Comparing the parallel and sequential non-hierarchical versions of LNS, we can see that LNS-P can obtain a better solution quality than LNS-S. Even for the shorter version of LNS-P the solution quality is still better than LNS-S so that the use

of parallelism is an important advantage. Results show that the parallel versions perform better than the sequential ones. The main advantage is that it provides an easy diversification mechanism. With a comparable number of iterations, the parallel version still outperforms the sequential one. Moreover, we can also see that the minimum of LNS-S is worse than LNS-P with 5000k iterations, so it can show the value of communication used in the parallel version.

Detailed results can be found in Table 5 in the Appendix. All four versions can obtain a solution quality that is between 1.5 and 3% better than the results of Pirkwieser (2012). However, they have a smaller runtime, with 91.9 seconds on average.

Table 3: Comparison to the method of Pirkwieser (2012)

	PT	LNS-HIER-S	LNS-HIER-P	LNS-S	LNS-P	
#iterations		5000k	5000k	5000k	714.5 k	5000k
%-gap to PT	0	-1.50	-2.47	-1.89	-2.44	-3.02
min		-2.09	-2.99	-2.88	-2.96	-3.42
T (sec)	91.9	2685.5	1755.3	354.4	88.2	530.7

Table 4 gives the new found best known solutions compared to the previously best found solutions. These are the best solutions identified during all the experiments. The previous solutions were found by Pirkwieser (2012) except for instance 20-5-1b for which the solution was reported in Prodhon (2011). Our algorithm finds new best know solutions or ties in all but one instance. On average, the new best found solutions are more than 2% better than the previous ones.

5 Conclusion

We have presented new sequential and parallel algorithms for the PLRP. The PLRP is an important problem, but has not received much attention in the literature so far. Our algorithms can outperform previous solution methods in terms of solution quality. Compared to the best performing algorithm so far, a VNS with ILP-based neighborhood searches (Pirkwieser 2012), our algorithms are between 1.5 and 3% better. We can also find new best known solutions or ties for all 30 instances but one with an average improvement of 2.17%.

Moreover, we proposed a simple methodology for parallelization that can easily be adopted to other algorithms and problems and showed that it can improve sequential versions. So whenever multiple processes are available it pays off to use parallelization to get better solutions in a shorter time.

Future work will focus on improving the parallel version of the algorithm. One direction is to not only exchange the best solution, but several and include diversity metrics when accepting a new solution. Another interesting approach is to run different searches in parallel. At the moment only the initial temperature is different for the processes, but by running more different algorithms in parallel the performance can probably be improved.

Table 4: Previous best known solutions and our best know solutions

Instance	previous BKS	best	%-gap
P20.5.0a	78477	78477	0
P20.5.0b	75554	76102	0.73
P20.5.2a	77784	77784	0
P20.5.2b	62133	62133	0
P50.5.0a	145639	143327	-1.59
P50.5.0b	134997	133076	-1.42
P50.5.2a	138486	137187	-0.94
P50.5.2b	110526	109030	-1.35
P50.5.2a'	168721	165849	-1.7
P50.5.2b'	100836	97845	-2.97
P50.5.3a	152530	152067	-0.3
P50.5.3b	108790	107770	-0.94
P100.5.0a	335702	327987	-2.3
P100.5.0b	223757	216266	-3.35
P100.5.2a	257710	253831	-1.51
P100.5.2b	162799	156240	-4.03
P100.5.3a	214118	210196	-1.83
P100.5.3b	166726	160694	-3.62
P100.10.0a	255630	250988	-1.82
P100.10.0b	207326	199488	-3.78
P100.10.2a	255345	248925	-2.51
P100.10.2b	166703	160234	-3.88
P100.10.3a	253280	246550	-2.66
P100.10.3b	188678	185300	-1.79
P200.10.0a	431131	416089	-3.49
P200.10.0b	359182	346872	-3.43
P200.10.2a	374016	366098	-2.12
P200.10.2b	308316	291456	-5.47
P200.10.3a	521229	508605	-2.42
P200.10.3b	339142	323136	-4.72
avg.			-2.17

Acknowledgments

The author would like to thank two anonymous referees for their valuable comments that helped to improve the quality of this paper.

References

Alba, E. (2005). *Parallel metaheuristics: a new class of algorithms*. Wiley Series on Parallel and Distributed Computing. Wiley.

Albareda-Sambola, M., Fernández, E., and Nickel, S. (2012). Multiperiod location-routing with decoupled time scales. *European Journal of Operational Research*, 217(2):248–258.

Baldacci, R., Bartolini, E., Mingozzi, A., and Valletta, A. (2011a). An exact algorithm for the period routing problem. *Operations research*, 59(1):228–241.

Baldacci, R., Mingozzi, A., and Wolfler-Calvo, R. (2011b). An exact method for the capacitated location-routing problem. *Operations Research*, 59:1284–1296.

Belenguer, J.-M., Benavent, E., Prins, C., Prodhon, C., and Wolfler Calvo, R. (2011). A branch-and-cut method for the capacitated location-routing problem. *Computers & Operations Research*, 38(6):931–941.

Brodtkorb, A., Hagen, T., Schulz, C., and Hasle, G. (2013). GPU computing in discrete optimization. part I: Introduction to the GPU. *EURO Journal on Transportation and Logistics*, 2(1-2):129–157.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581.

Contardo, C., Cordeau, J.-F., and Gendron, B. (2013a). An exact algorithm based on cut and column generation for the capacitated location-routing problem. *INFORMS Journal on Computing*. to appear.

Contardo, C., Cordeau, J.-F., and Gendron, B. (2013b). A GRASP + ILP-based metaheuristic for the capacitated location-routing problem. *Journal of Heuristics*. to appear.

Cordeau, J.-F. and Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050.

Crainic, T. and Nourredine, H. (2005). Parallel metaheuristics applications. In Alba, E., editor, *Parallel Metaheuristics*, pages 447–494. Wiley.

Crainic, T. G. (2008). Parallel solution methods for vehicle routing problems. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 171–198. Springer US.

Crainic, T. G. and Toulouse, M. (2010). Parallel meta-heuristics. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 497–541. Springer US.

De Franceschi, R., Fischetti, M., and Toth, P. (2006). A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming*, 105(2-3):471–499.

Drexl, M. and Schneider, M. (2013). A survey of location-routing problems. Technical Report LM-2013-03, Gutenberg School of Management and Economics, Johannes Gutenberg University, Mainz.

Escobar, J. W., Linfati, R., and Toth, P. (2013). A two-phase hybrid heuristic algorithm for the capacitated location-routing problem. *Computers & Operations Research*, 40(1):70–79.

Francis, P., Smilowitz, K., and Tzur, M. (2006). The period vehicle routing problem with service choice. *Transportation Science*, 40(4):439–454.

Groër, C., Golden, B., and Wasil, E. (2011). A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2):315–330.

Gulczynski, D., Golden, B., and Wasil, E. (2011). The period vehicle routing problem: New heuristics and real-world variants. *Transportation Research Part E: Logistics and Transportation Review*, 47(5):648–668.

Hemmelmayr, V. C., Cordeau, J.-F., and Gabriel Crainic, T. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39:3215–3228.

Jin, J., Crainic, T. G., and Løkketangen, A. (2012). A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. *European Journal of Operational Research*, 222(3):441–451.

Lahrichi, N., Crainic, T., Gendreau, M., Rei, W., Crisan, G., and Vidal, T. (2012). An integrative cooperative search framework for multi-decision-attribute combinatorial optimization. Technical Report CIRRELT-2012-42, Université de Montréal.

Lopes, R. B., Ferreira, C., Santos, B. S., and Barreto, S. (2013). A taxonomical analysis, current methods and objectives on location-routing problems. *International Transactions in Operational Research*, 20(6):795–822.

Nagy, G. and Salhi, S. (2007). Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177:649–672.

Pirkwieser, S. (2012). *Hybrid Metaheuristics and Matheuristics for Problems in Bioinformatics and Transportation.* PhD thesis, Vienna University of Technology.

Pirkwieser, S. and Raidl, G. R. (2010). Variable neighborhood search coupled with ILP-based very large-neighborhood searches for the (periodic) location-routing problem. In *Hybrid Metaheuristics - Seventh International Workshop, HM 2010*, volume 6373 of *Lecture Notes in Computer Science*, pages 174–189, Vienna.

Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US.

Prins, C., Prodhon, C., and Wolfler Calvo, R. (2006). Solving the capacitated location-routing problem by a GRASP complemented by a learning process and a path relinking. *4OR: A Quarterly Journal of Operations Research*, 4:221–238.

Prodhon, C. (2008). A metaheuristic for the periodic location-routing problem. In Kalcsics, J. and Nickel, S., editors, *Operations Research Proceedings 2007*, volume 2007 of *Operations Research Proceedings*, pages 159–164. Springer Berlin Heidelberg.

Prodhon, C. (2009). An ELSxpath relinking hybrid for the periodic location-routing problem. In Blesa, M., Blum, C., Gaspero, L., Roli, A., Sampels, M., and Schaerf, A., editors, *Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin Heidelberg.

Prodhon, C. (2011). A hybrid evolutionary algorithm for the periodic location-routing problem. *European Journal of Operational Research*, 210(2):204–212.

Prodhon, C. and Prins, C. (2008). A memetic algorithm with population management (MA| PM) for the periodic location-routing problem. In Blesa, M., Blum, C., Cotta, C., Fernández, A. J., Gallardo, J. E., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 5296 of *Lecture Notes in Computer Science*, pages 43–57. Springer.

Prodhon, C. and Prins, C. (2014). A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1 – 17.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171.

Schulz, C., Hasle, G., Brodtkorb, A., and Hagen, T. (2013). GPU computing in discrete optimization. part II: Survey focused on routing problems. *EURO Journal on Transportation and Logistics*, 2(1-2):159–186.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, pages 417–431. Springer.

Ting, C.-J. and Chen, C.-H. (2013). A multiple ant colony optimization algorithm for the capacitated location routing problem. *International Journal of Production Economics*, 141(1):34–44.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624.

Appendix

Table 5: Detailed results for the comparison to the method of Pirkwieser (2012)

	PT		LNS-HIER-S			LNS-S			LNS-HIER-P			LNS-P			LNS-P-short		
	avg	T	avg	T	%-gap	avg	T	%-gap	avg	T	%-gap	avg	T	%-gap	avg	T	%-gap
P20.5.0a	78828	18.9	78602.8	96.4	-0.29	78738.4	53.2	-0.11	78574.4	70.8	-0.32	78477	132.032343	-0.45	78605.8	17.8	-0.28
P20.5.0b	77196.1	19.7	76102	90.2	-1.42	76887.8	55.8	-0.4	76182.2	68.6	-1.31	76102	146.492867	-1.42	76102	22	-1.42
P20.5.2a	78347	18.6	78533.2	91.6	0.24	79041	54.8	0.89	78379.6	70.2	0.04	77784	158.826704	-0.72	77962.8	23.8	-0.49
P20.5.2b	62536.4	18.9	62311.2	86.2	-0.36	63959.6	57.6	2.28	62314.8	66.6	-0.35	62313.8	152.859071	-0.36	62560.6	26.9	0.04
P50.5.0a	148465	43.9	144122.2	263.6	-2.93	145782.6	104	-1.81	144545.2	160.2	-2.64	143708.2	343.55565	-3.2	143715.8	39.8	-3.2
P50.5.0b	138618.9	45.7	135616.6	298.4	-2.17	136976.6	113.8	-1.18	134442.8	179.2	-3.01	133950.6	278.914902	-3.37	135047.2	45.1	-2.58
P50.5.2a	140028.5	43	137414	303.2	-1.87	138801.2	113.4	-0.88	137589.8	207.4	-1.74	137385.4	306.481	-1.89	137617.6	48.6	-1.72
P50.5.2b	112028.6	42.9	110087.8	386.6	-1.73	111027.6	114	-0.89	109997.6	295.8	-1.81	109938.4	194.961779	-1.87	110206.6	30.8	-1.63
P50.5.2a'	170160.1	34.3	166819.2	427	-1.96	167240.6	157.2	-1.72	167292	347.4	-1.69	167280.2	358.11216	-1.69	167244.6	42.3	-1.71
P50.5.2b'	102100.4	36.4	97853.4	359.4	-4.16	97845	125	-4.17	97845	284.6	-4.17	97847.2	190.743465	-4.17	97871.2	31.8	-4.14
P50.5.3a	154099.6	40.7	153346.6	335.4	-0.49	155907.8	130.2	1.17	152623.6	208	-0.96	152349.4	395.360296	-1.14	152422.6	52	-1.09
P50.5.3b	110331.9	42	108464	362.2	-1.69	109599.2	108.2	-0.66	109081	234.6	-1.13	108242.4	259.64329	-1.89	108956.8	42	-1.25
P100.5.0a	342945.9	65.7	338838.8	1060.4	-1.2	330350.4	284.6	-3.67	335541.2	823.4	-2.16	329808.4	582.762924	-3.83	332563.4	76.5	-3.03
P100.5.0b	227701.3	77.2	225077.6	1366.2	-1.15	221016.8	278.2	-2.94	221098	871	-2.9	217680.4	310.531968	-4.4	219552.8	67.1	-3.58
P100.5.2a	261394.9	66	256802.8	1299.4	-1.76	255419.8	405.8	-2.29	254734.4	1058.2	-2.55	254205	516.466316	-2.75	255124.6	104.5	-2.4
P100.5.2b	164494.6	62.5	159225.6	1436.2	-3.2	158046.6	269.4	-3.92	158067	957.4	-3.91	158036.2	287.456271	-3.93	158610.4	63.3	-3.58
P100.5.3a	219215	71	211591.4	972.4	-3.48	211856	260.2	-3.36	211051.8	672.2	-3.72	210876.6	376.157227	-3.8	212303.4	65.2	-3.15
P100.5.3b	171381.9	85.8	165465.2	1000	-3.45	162876.8	222.2	-4.96	161225.6	612.6	-5.93	161772.8	396.064017	-5.61	163344.6	67.9	-4.69
P100.10.0a	259201.3	159.4	260365.8	935.6	0.45	255120.8	253.2	-1.57	254526.4	626	-1.8	252605.2	589.646426	-2.54	254977.8	84.7	-1.63
P100.10.0b	208568.5	146.7	209791.4	1116	0.59	203768	246.8	-2.3	204284	757	-2.05	202260.6	358.865524	-3.02	203959.2	69	-2.21
P100.10.2a	259468.8	105.7	253898.4	1464.2	-2.15	250591.2	364	-3.42	250648.2	1278	-3.4	250123	616.461397	-3.6	251805.4	85.3	-2.95
P100.10.2b	169623.4	93.6	163432.4	1342	-3.65	163133.4	275.2	-3.83	160517.4	1017.2	-5.37	161230.2	337.188662	-4.95	162861.4	52.2	-3.99
P100.10.3a	256907.6	131.5	253235.2	886	-1.43	249652.4	303.8	-2.82	250111.4	711.8	-2.65	247949	471.059452	-3.49	249885	81	-2.73
P100.10.3b	192022.6	120.5	191226.6	1324.4	-0.41	190913	304.6	-0.58	186861.2	1045.8	-2.69	186734.4	420.981954	-2.75	187682.4	63.5	-2.26
P200.10.0a	433030.7	216.2	428029.8	3037.8	-1.15	420560	788.8	-2.88	426462.4	2156.8	-1.52	418962.6	1335.79554	-3.25	423797.2	210.2	-2.13
P200.10.0b	367650	240.1	365635.8	3924.2	-0.55	351249	778.8	-4.46	356731.4	2997.8	-2.97	348706	867.246158	-5.15	353415.6	180.2	-3.87
P200.10.2a	378350.9	168	375013	3480.4	-0.88	379815.4	1152.2	0.39	370246	2590.6	-2.14	371270.8	1397.57476	-1.87	373178	259.4	-1.37
P200.10.2b	311796.9	182.5	304559.2	37949.8	-2.32	299556.2	1024.4	-3.93	298004.8	22998.4	-4.42	295127.2	1371.14731	-5.35	297399.4	222.2	-4.62
P200.10.3a	528412.1	179.4	529347.8	6166.5	0.18	523171.8	1263.2	-0.99	522604.2	4491.4	-1.1	511767.4	1658.85401	-3.15	516402.4	293	-2.27
P200.10.3b	343923.6	180.2	341810.4	8703.3	-0.61	337879.4	969.6	-1.76	331010.2	4799.8	-3.75	327199	1107.39886	-4.86	332404.2	178.3	-3.35
avg	215627.68	91.90	212754.01	2685.50	-1.50	210892.81	354.41	-1.89	210086.45	1755.29	-2.47	208389.78	530.65	-3.02	209919.36	88.22	-2.44