An Efficient Heuristic for the Multi-Vehicle One-to-One Pickup and Delivery Problem with Split Loads

Mustafa Şahin [a], Dilek Tüzün Aksu [b], Gizem Çavuşlar [a], Temel Öncan [c] and Güvenç Şahin [a,1]

[a] *Manufacturing Systems and Industrial Engineering, Sabanci University*
Tuzla, İstanbul, 34956, Türkiye
{mustafasahin}{gizemc}{guvencs}@sabanciuniv.edu

[b] *Department of Systems Engineering, Yeditepe University*
Kadıköy, İstanbul, 34755, Türkiye
dtuzun@yeditepe.edu.tr

[c] *Endüstri Mühendisliği Bölümü, Galatasaray Üniversitesi*
Ortaköy, İstanbul, 34357, Türkiye
ytoncan@gsu.edu.tr

Abstract

In this study, we consider the Multi-vehicle One-to-one Pickup and Delivery Problem with Split Loads (MPDPSL). This problem is a generalization of the one-to-one Pickup and Delivery Problem (PDP) where each load can be served by multiple vehicles as well as multiple stops by the same vehicle. In practice, split deliveries is a viable option in many settings where the load can be physically split, such as courier services of third party logistics operators. We propose an efficient heuristic that combines the strengths of Tabu Search and Simulated Annealing for the solution of MPDPSL. Results from experiments on two problems sets in the literature indicate that the heuristic is capable of producing good quality solutions in reasonable time. The experiments also demonstrate that up to 33% savings can be obtained by allowing split loads; however, the magnitude of savings is dependent largely on the spatial distribution of the pickup and delivery points.

Keywords: Pickup Delivery, Vehicle Routing, Split Loads, Tabu Search, Simulated Annealing.

[1] Corresponding author.

1 Introduction

The Pickup and Delivery Problem has attracted the interest of various researchers in the last three decades (see e.g. recent surveys by Cordeau et al.[4], Gribkovskaia and Laporte [7]). PDP consists of finding the least cost route of a single vehicle which picks up a load from an origin and delivers it to its destination. If each origin is associated with a single destination, making up a pickup-delivery (p-d) pair the problem is called the one-to-one PDP. As stated in [4], this version of PDP differs from the other two variants in the literature; the many-to-many PDP where a commodity may be picked up at one of many origins and then delivered to one of many destinations, and the one-to-many-to-one PDP where all loads to be picked-up and delivered originate at a common depot. In a conventional PDP setting, each p-d pair is visited by a single vehicle. In this article, we study a variant where the load of a p-d pair can be split between multiple vehicles and/or multiple stops of the same vehicle. The single vehicle version of this problem, namely the Pickup and Delivery Problem with Split Loads (PDPSL) was first introduced by Nowak et al. [9].

The Multi-vehicle One-to-one Pickup and Delivery Problem with Split Loads (MPDPSL), which is a generalization of the PDPSL is defined on a directed graph $G = (V, A)$ where V is the vertex set and A is the arc set. The vertex set is partitioned as $V = \{P, D, \{0, 2n+1\}\}$. For a given set of n pickup delivery pairs, $P = \{1, 2, \ldots, n\}$ is the set of pickup vertices and $D = \{n+1, \ldots, 2n\}$ is the set of delivery vertices where i and $n+i$ represent a pickup and delivery vertices for load i. Furthermore, $\{0, 2n+1\}$ includes two copies of the depot location and the set of arcs are defined as follows $A = \{(i,j) : i = 0, j \in P\} \cup \{i, j \in P \cup D, i \neq j, i \neq n+j\} \cup \{(i,j) : i \in D, j = 2n+1\}$. Let $K = \{1, 2, \ldots, m\}$ denote the set of available vehicles. For each vertex $i \in V$, q_i denotes the pickup or delivery quantity where $q_i > 0$ for $i \in P, q_i = -q_{i-n}$ for i$\in D$ and $q_0 = q_{2n+1} = 0$. Each vehicle $k \in K$ has a capacity of Q. We assume that the load of any p-d pair can be provided by a single vehicle, i.e. $q_i \leq Q$. d_{ij} is the travel distance associated with arc $(i,j) \in A$ and D is the maximum travel distance of any vehicle route.

Although MPDPSL has not been widely studied in the literature, possibility of load splitting exists in many PDP settings. Novak et al. [9] describe the less-than-truckload service of a third-party logistics company where load splitting results in significant benefits. Similar savings might be achieved in other areas where loads can be split among multiple vehicles, such as bulk product transportation by ship, where each load is already packaged into multiple containers, or courier services that deliver multiple packages between the same origin-destination pair. As long as loads can be physically split between

vehicles, MPDPSL can lead to savings over the classical PDP by reducing the unused vehicle capacity.

Nowak et al. [9] have shown that the optimal load size for splitting is just above one half of a truckload and demonstrated the relation between the benefit of split loads and the problem characteristics on the third-party logistics case study. The benefit of split loads and the problem characteristics that are affected most by load splitting have been experimentally analyzed in their subsequent work [10]. The authors have empirically observed that when the demands are distributed between 0.51 % and 0.60 % of the truckload (vehicle capacity) up to 30 % savings in route cost can be achieved. Other factors increasing the benefit of split loads are the number of loads available at a common location for pickup or delivery and the average distance from an origin to a destination relative to the distance from origin to origin and destination to destination. The authors have observed that an increase in these two factors both result in an increase in the benefit of split loads.

The split load case is also considered within the VRP context. The well-known Split Delivery Vehicle Routing Problem (SDVRP) tries to find a set of minimum cost routes of a fleet of capacitated homogeneous vehicles available to serve a set of customers. Each customer can be visited more than once and the demand of each customer may be greater than the vehicle capacity. The earliest papers on the SDVRP trace back to the work by Dror and Trudeau ([6,6]). For a recent survey on the SDVRP, we refer to Archetti and Speranza [2]).

To the best of our knowledge, there is no other work in the literature that deals with the multi-vehicle extension of the PDPSL. The motivation of this work is first to present the MPDPSL, and then to introduce our Tabu Embedded Simulated Annealing (TESA) algorithm specially tailored for this problem. Computational experiments are performed on both PDPSL test problems and split load versions of PDP instances from the literature. Our main contributions in this work are as follows:

- we introduce a generalized version of the PDPSL with multiple vehicles;
- a metaheuristic algorithm is designed to solve the MPDPSL;
- we perform computational experiments to
  - show the efficiency and effectiveness of our algorithm,
  - analyze the benefit of split loads for the multi-vehicle problem with different network structures; and
- we report the first results on a set of MPDPSL test problems.

The remainder of this work is organized as follows. In Section 2, we discuss the details of TESA heuristic algorithm. We present the results of computational

experiments in Section 3. Finally, we close with concluding remarks in Section 4.

2 MPDPSL Heuristic

We propose a Tabu Embedded Simulated Annealing (TESA) heuristic for MPDPSL. The heuristic starts by creating an initial solution using a variant of the savings heuristic by Clarke and Wright [3]. This solution is then improved by searching neighboring solutions through a variety of moves, including insertion of a p-d pair into a different route, split of a p-d pair between routes as well as insertion and swap of route segment(s) that consist of multiple p-d pairs. The heuristic utilizes features of both tabu search and simulated annealing to search the solution space efficiently.

In the literature, tabu search and simulated annealing methods have been used together in metaheuristic algorithms. Li et. al. [8] and Thangiah et. al. [13] use simulated annealing to obtain non-tabu neighbor solutions for the Vehicle Routing Problem with Time Windows (VRPTW). Osman [11] employs this approach to solve the Generalized Assignment Problem. Our approach follows the footsteps of Li et. al. [8] and Thangiah et. al. [13]; we use a simulated annealing-like procedure to select among a set of possible eliticized moves while searching the neighborhood of a current given solution. In particular, in order to jump to a neighboring solution, we do not necessarily choose to apply the best possible move; we consider a list of good moves, and use a randomized procedure to select a move from this list. The simulated annealing procedure is employed to direct the randomization scheme.

Before we explain the details of TESA heuristic, we describe the local neighborhood structures used in this heuristic. The search neighborhoods to be used in a metaheuristic is very critical to the efficiency of the search performed. This is especially true for an MPDPSL heuristic where the neighborhoods are potentially very large due to many possible ways of splitting a load between available routes. To overcome the excessive computational cost of exploring these large neighborhoods, we use the following three neighborhoods in our heuristic.

Insert/Split Neighborhood. A neighboring solution in this neighborhood is constructed by removing a p-d pair from its current route and either inserting the entire load into a different route or splitting it between two routes. Only feasible moves that do not violate the precedence, vehicle capacity and route length constraints are included in the insert/split neighborhood. We limit the

number of splits for a given p-d pair to a maximum of k. At any iteration of TESA algorithm, there are at most $2nk$ stops in the solution. Thus, for a given p-d pair, there are $O(n^2)$ ways of inserting the pair into two positions in another route, and $O(n^4)$ ways of splitting the pair between two other routes. As a result, the computational complexity of evaluating the entire insert/split neighborhood for a particular p-d pair and identifying the best insert/split move among all p-d pairs are $O(n^4)$ and $O(n^5)$, respectively.

Block Insert Neighborhood. A *block* of consecutive pickup and delivery stops is removed from its current route and inserted into a new route as a whole. A block is a route segment that starts and ends with an empty vehicle. While there might be several blocks within a vehicle route, a route may also consist of a single block. The capacity and precedence constraints remain intact when a block is removed from a route and inserted into another. Therefore, only route length constraints need to be checked to ensure the feasibility of solutions in this neighborhood. In the worst case, a solution may contain nk blocks, each containing a single p-d pair. In this case, there would be nk different positions where the block can be inserted, resulting in a $O(n)$ moves in the neighborhood of a particular block. Thus, for a given solution, selecting the block insert move that results in the most route length reduction is $O(n^2)$. Compared to the insert/swap neighborhood, evaluating the block insert neighborhood is very fast.

Block Swap Neighborhood. This neighborhood consists of solutions that are obtained by swapping the routes of two blocks and inserting those blocks at the best possible positions on their new routes. Similar to the block insert move, block swap moves also do not violate the precedence and capacity constraints. It suffices to check only the route length constraints to create feasible moves. In a feasible solution, there are $O(n^2)$ possible block swaps, and the computational complexity of determining the best position for a block is $O(n)$. Therefore, computational complexity of selecting the pair of blocks that yield the most route length reduction is $O(n^3)$.

TESA heuristic that employs the above neighborhood structures is outlined in Algorithm 1. The algorithm starts by finding an initial feasible solution. This initial solution is, then, improved by searching new solutions in the neighborhood of an incumbent solution. The algorithm subsequently goes through series of improving phases. Among these phases, three of them are based on searching the above neighborhoods of the incumbent solution.

Algorithm 1 MPDPSLAlgorithm

Input: $N_{iter1}, N_{iter2}, N_{move}, N_{pair}, Pr$
Output: S_b
$S_b \leftarrow InitialSolutionHeuristic()$
while $Pr > 0.1$ **do**
  $S_b \leftarrow Insert/SplitPhase(S_b, N_{iter1}, Pr)$
  $S_b \leftarrow IntraRouteHeuristic(S_b)$
  $S_b \leftarrow BlockInsertPhase(S_b, N_{iter2})$
  $S_b \leftarrow BlockSwapPhase(S_b, N_{iter2})$
  $Pr \leftarrow Pr/3$
end while
return S_b

2.1 Initial Solution Heuristic

The heuristic algorithm starts by creating an initial solution. This initial solution, which is based on the savings algorithm, does not contain any split loads. In parallel with the original savings algorithm, we first create a solution where each p-d pair is initially served by a separate vehicle route of the form $(0, i, i+n, 0)$. We then compute a savings value for every pair of pickup point i and delivery point j such that $j \neq i+n$. The savings value is the difference in total route length when the routes that serve the two points i and j are combined into a single route. Next, we sort the pairs in non-increasing order of their savings. Starting from the first pair on the list, we merge the routes that visit the pairs with positive savings while ensuring the feasibility of the resulting routes. Finally, we carry out an improvement step where the pickup point i and delivery point j are moved forward and backward respectively in the merged route provided that such a move results in further savings.

2.2 Insert/Split Phase

In this phase, we improve the current solution through subsequent insert/split moves. In each iteration of this phase, a p-d pair is removed from its current route and either the entire load is inserted into a different route or it is split between two routes. Since computational complexity of an insert/split move for a particular p-d pair is very high, it is quite time consuming to evaluate all p-d pairs at every iteration. Therefore, we use a binary heap implementation to improve the average computational performance.

In order to select the p-d pair to be inserted/split in each iteration, we first evaluate the insert/splitneighborhood for all p-d pairs to determine the best

N_{move} non-tabu moves that results in the lowest ΔC value for each pair where ΔC denotes the change in total route length as a result of the move. A positive ΔC value corresponding to an increase in the route length. In evaluating the neighborhood, we limit our search to only feasible insert and split moves that satisfy the precedence, route length and vehicle capacity restrictions.

The tabu structure in this algorithm consists of four dimensions: the route length, number of stops and number of vehicles in the visited solution, as well as the p-d pair inserted/split to create that solution. The reason for using such a complex tabu structure is due to the special property of the test problems by Nowak et al.[9] In these test problems, all p-d pairs are generated from a limited set of pickup and delivery points. Since there are many nodes that share the same pickup and delivery coordinates, conventional tabu structures that keep track of involved node(s) are rendered ineffective for these test problems. Using the multi-dimensional tabu structure that keeps track of the features of the visited solution (in addition to the move applied to reach that solution) allows the algorithm to identify the previously visited solutions correctly and avoids visiting them repeatedly later on.

As discussed earlier, evaluating the insert/split neighborhood for all p-d pairs results in a complexity of $O(n^5)$ for every iteration. In order to avoid this excessive computational burden, we evaluate the insert/split neighborhood using a binary heap implementation. While the worst-case behavior of the binary heap implementation is the same ($O(n^5)$), our computational experience indicates that the average running time is only $O(n^3 \log n)$, which is significantly better than the original implementation. (A detailed description of the binary heap implementation and its computational complexity can be found in Appendix A.)

Once we create a list of the top N_{move} non-tabu moves with the lowest ΔC values among all feasible insert and split moves for every p-d pair, we employ a simulated annealing based selection criterion to select one candidate move for each p-d pair. In a conventional simulated annealing algorithm, a move is selected with a probability $e^{-\Delta C/temp}$, where $temp$ denotes the current temperature of the simulated-annealing-like randomized selection procedure. The average value of ΔC may fluctuate significantly between p-d pairs. Therefore, the acceptance probability of a move with an average ΔC value also changes widely. To prevent this, we use the following approach: Instead of varying the temperature $temp$ from one p-d pair to the next, we vary the probability of accepting an average move, Pr. For each p-d pair, we first calculate the average ΔC value for the best N_{move} non-tabu moves in the neighborhood. Then, a temperature $temp$ is calculated such that the probability of accepting an average move is Pr. Finally, a move is selected randomly among the N_{move} moves; this move is accepted with a probability of $e^{-\Delta C/temp}$. If the current

move is rejected, then another move is selected randomly from the list until one of the feasible moves is accepted. After the candidate moves for each p-d pair are determined in this fashion, the move to be implemented is selected randomly out of the best N_{pair} p-d pairs with moves resulting in the highest ΔC values.

Following each insert/split move, we check if the solution can be improved by merging any pickup or delivery stops for the p-d pair involved in this move. This corrective move exploits an optimality condition for MPDPSL.

Optimality Condition. In an optimal solution of MPDPSL where the distance matrix satisfies the triangular inequality, multiple pickup (deliveries) stops of a p-d pair cannot exist on the same vehicle route without a delivery (pickup) stop of the same p-d pair in between.

We can show that a reduction in route length can be achieved for solutions that do not satisfy this condition by eliminating all but the last (first) pickup (delivery) without violating the capacity constraints. After each insert/split move, we check whether the above condition is violated for the p-d pair under consideration and merge multiple stops into one wherever possible. In the case of multiple pickup stops of the same p-d pair, the entire load is picked up at the last stop, eliminating all other pickup stops and resulting in a reduction of route length. Conversely, all deliveries but the first one are eliminated for the case of multiple delivery stops without a pickup of the same load in between.

The insert/split phase is terminated after no improvement can be achieved over the best solution for a consecutive N_{iter1} iterations. The pseudo code of the insert/split phase is provided in Algorithm 2.

2.3 Intra-Route Phase

On a feasible vehicle route, if a pickup node is shifted towards its corresponding delivery node, or a delivery node is shifted towards its pickup node, the precedence and capacity constraints will not be violated. Therefore, a pickup (delivery) node can be shifted towards its delivery (pickup) node, as long as the shift does not violate the route length constraint. The intra-route phase exploits this property within the blocks. For each block, the heuristic considers shifting all pickup nodes forward and delivery nodes backward in the route, starting from the first node in the block. If a decrease in route length can be achieved, the move is implemented and the heuristic proceeds with the next block.

Algorithm 2 Insert/SplitPhase

Input: S_c, N_{iter1}
Output: S_b $//S_b$ denotes the best solution
$S_b \leftarrow S_c$ $//S_c$ denotes the current solution
$T \leftarrow \emptyset$ $//T$ denotes the list of best N_pair insert/split moves
$i \leftarrow 0$
while $i < N_{iter1}$ **do**
  for all $(p, d) \in S_c$ **do**
    if $c(Insert/Split(S_c, (p, d), Pr)) < c(maxCost(T))$ **then**
      if $|T| = N_{pair}$ **then**
        $T \leftarrow T - maxCost(T)$ **and** $T \leftarrow T \cup Insert/Split(S_c, (p, d), Pr)$ $//maxCost(T)$ returns the insert/split position $< \ell^{p1}, \ell^{d1} > - < \ell^{p2}, \ell^{d2} >$ with the largest cost from the list T
      else
        $T \leftarrow T \cup Insert/Split(S_c, (p, d), Pr)$
      end if
    end if
  end for
  Select a pair (p^*, d^*) randomly from T
  $UpdateSolution(S_c, (p^*, d^*), <\ell^{p1^*}, \ell^{d1^*}> - <\ell^{p2^*}, \ell^{d2^*}>)$
  if $c(S_c) < c(S_b)$ **then**
    $S_b \leftarrow S_c$
    $i \leftarrow 0$
  else
    $i \leftarrow i + 1$
  end if
end while
return S_b

2.4 Block Insert Phase

In one iteration of the block insert phase, the heuristic selects the block insert move that results in the lowest ΔC value and performs the corresponding insert move. This phase is terminated after no improvement can be achieved over the current best solution for a consecutive N_{iter2} iterations.

2.5 Block Swap Phase

In each iteration of the block swap phase, the block swap move that yields the lowest ΔC value is identified and implemented. The block swap move is also

terminated after no improvement can be achieved over the best solution for a consecutive N_{iter2} iterations.

The termination of the subsequent improvement phases depends on the value of Pr, selection probability employed in the insert/split phase. For a given value of Pr, the algorithm executes the four phases described above subsequently before it decreases Pr by a factor of $\beta < 1$. This step is parallel to the reduction of the temperature by a cooling factor in a conventional simulated annealing algorithm. At the beginning of the algorithm, the probability of selecting inferior solutions is quite high, which allows the search to diversify and avoid the local minima. As the algorithm progresses, the probability of selecting inferior solutions decreases gradually, so that the algorithm converges to a good solution. Preliminary computational experiments indicated that the algorithm did not have the capability of avoiding the local minima without this simulated annealing based strategy. The solution quality improved considerably after the introduction of the probabilistic selection criterion. It is important to emphasize that the selection criterion is implemented on the best N_{move} non-tabu results, which ensures that the algorithm does not spend time in parts of the solution space that are not promising. Moreover, utilizing a tabu structure ensures that the algorithm does not revisit the same solution. This overcomes one of the major weaknesses of simulated annealing, that it lacks any memory features to track previously visited solutions. The resulting algorithm combines powerful features of both tabu search and simulated annealing. The computational results reported in the next section demonstrate that this algorithm is capable of producing good solutions for MPDPSL in reasonable time.

3 Computational Results

3.1 Parameter Tuning for TESA Algorithm

Our preliminary experiments indicate that four of the parameters used in TESA algorithm have a significant impact on both solution quality and CPU time: N_{iter1}, N_{iter2}, N_{pair} and N_{move}. For the purpose of parameter tuning, we only consider these parameters. To simplify the experimental design, we use, in a given setting, the same values for the number of nonimproving iterations in any phase, i.e. N_{iter1} for the Insert/Split phase and N_{iter2} for Block Insert and Block Swap phases; we denote this parameter as N_{iter}. N_{move} determines the size of the list that keeps the least costly moves for a selected p-d pair; the likelihood of selecting a poor quality move increases as N_{move} gets larger. N_{pair} determines the size of the list that keeps the best pairs for the current solution

in implementing a move. Similarly, when N_{pair} gets larger, the likelihood of selecting an inferior pair is increased.

The computational study for these parameters is conducted with the multi-vehicle version of the randomly generated problems in Nowak et al. [9]. Parameter tuning is done for the set of problems of 100 p-d requests with a load range of 0.51-0.6 truckload. For $N_{iter} \in \{25, 50, 75, 100, 125\}$, $N_{move} \in \{5, 7, 10, 15\}$, and $N_{pair} \in \{1, 3, 5, 7, 10\}$, we conduct the tuning study for 100 possible parameter settings. For a given parameter setting, the algorithm is restarted 20 times. To evaluate the quality of a parameter setting, we use the average improvement attained over the route length of the initial solution with no splits (see Section 2.1) as the effectiveness measure and the average CPU time as the efficiency measure. Computational experiments are conducted on a single core of a computer with Intel Core2Quad Q8200 @2.33 gHz CPU and 3.46gB of RAM.

According to the results of our study:

- The most influential parameter is the number of nonimprovements, N_{iter}, as the improvement in the route length decreases by 2% when N_{iter} is increased from 25 to 125. The solution quality changes by only 0.2% between the best and the worst settings of N_{pair} and N_{move}. The CPU time also increases when the value of each of these three parameters is increased.
- When the effect of each parameters is inspected individually (see Figures 1, 2, 3), $N_{iter} = 125$, $N_{move} = 10$ and $N_{pair} = 5$ appears to be the best setting of parameters.
- Inspecting the combined effect of N_{iter} and N_{pair} (see Figure 5), and the combined effect of N_{iter} and N_{move} (see Figure 4), it is clear that the effect of N_{iter} overweighs and best results are attained when N_{iter} is as large as possible (i.e. $N_{iter} = 125$). According to the combined effect of N_{move} and N_{pair} in Figure 6, $N_{move} = 15$ and $N_{pair} = 3$ yields the best solution quality. Thus, when the combined effect is considered, the best setting for all three parameters appears to be $N_{iter} = 125$, $N_{move} = 15$ and $N_{pair} = 3$.

Taking into account the trade-off between the improvement in the route length and CPU time, it would be more reasonable to set the value of N_{iter} based on the availability of computing power. In essence, one might prefer a smaller value of N_{iter} if the computing resources are restricted and CPU time is an important concern. The values of N_{pair} and N_{move} are dependent on the problem size, i.e. the number of p-d requests. Therefore, we adjust the values of these parameters in proportion to the number of p-d requests in a given instance.

Another important parameter that affects the CPU time is the number of restarts. We do not include this parameter directly in the parameter tuning

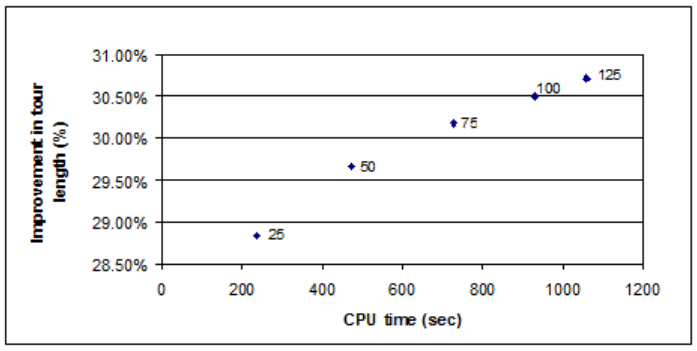


Fig. 1. Improvement in route length versus CPU time for different values of N_{iter}.

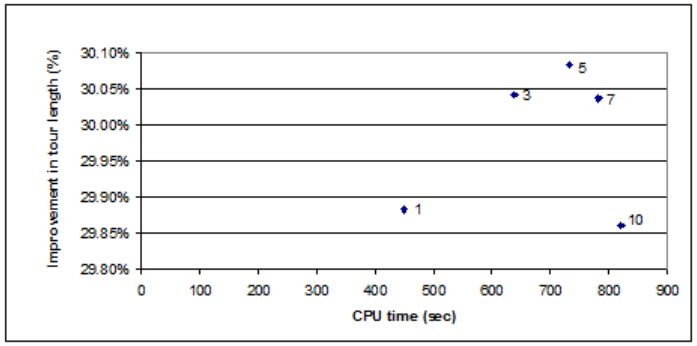


Fig. 2. Improvement in route length versus CPU time for different values of N_{pair}.

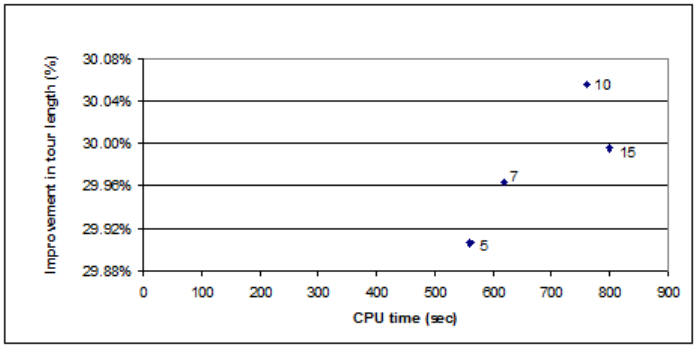


Fig. 3. Improvement in route length versus CPU time for different values of N_{move}.

study. Instead, we use each restart as a sampling procedure since each restart of the algorithm randomizes the selection of moves. A high number of restarts increases the likelihood of diversification among the obtained solutions. In order to validate this anticipation, we closely investigate the improvement pattern through the restarts of the algorithm over a maximum of 100 restarts. In Figure 7, we show the route length obtained at every restart of the algorithm along with the best route length obtained until after that restart. In these three examples, we observe that the best route length is obtained at restart 23 (in part (a)), restart 47 (in part (b)) and restart 96 (in part (c)). This observation

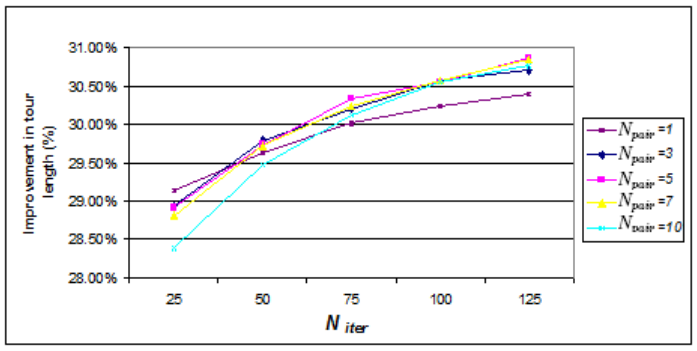


Fig. 4. Combined effect of N_{iter} and N_{pair} on the improvement in route length.

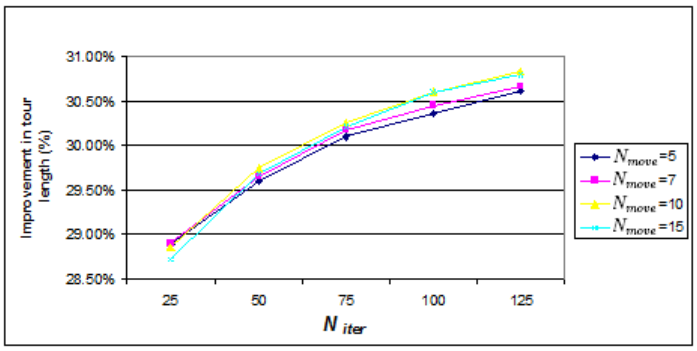


Fig. 5. Combined effect of N_{iter} and N_{move} on the improvement in route length.

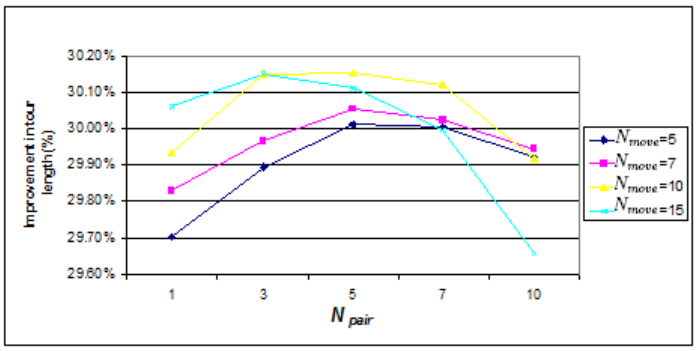
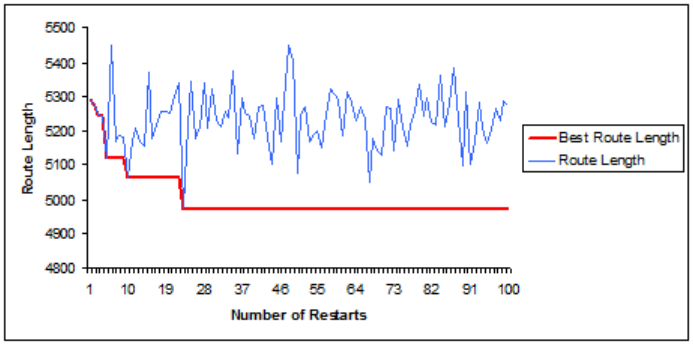
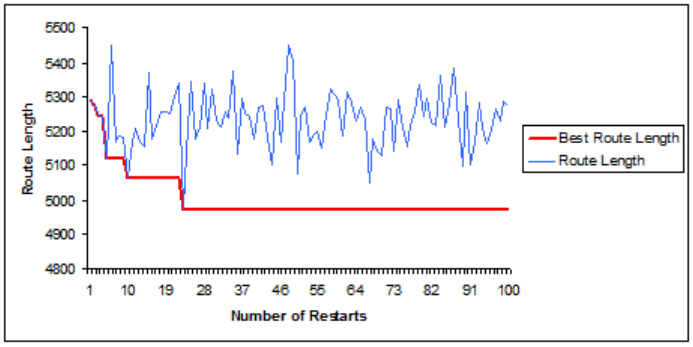


Fig. 6. Combined effect of N_{move} and N_{pair} on the improvement in route length.

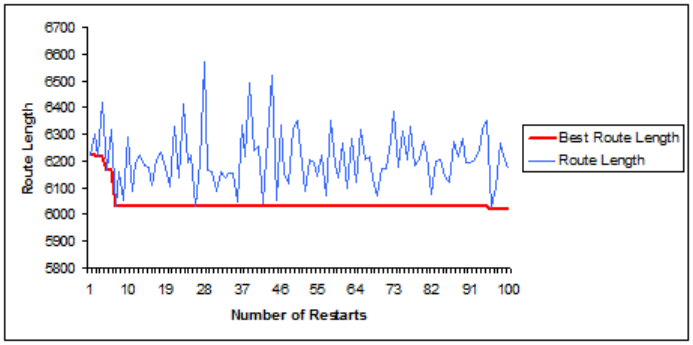
validates the significance of restarts and the effect of randomization on the solution quality through the diversification mechanism induced by restarts. It is clear that the solution quality might improve when more effort is spent by increasing the number of restarts. As in the case of N_{iter1}, one should consider the trade-off between the CPU time and number of restarts with respect to the availability of computing resources.



(a)



(b)



(c)

Fig. 7. The pattern of route length improvement through the restarts of the algorithm.

3.2 Computational Results on Test Problems

Nowak et al. [9] has investigated the improvement in route length when splits are allowed over the route length of the solution with no splits. The aim of their analysis is to show that one may benefit from splitting some p-d requests (i.e. stopping at the origin and destination of the requests more than once) instead of visiting the p-d pair only once as in the traditional setting of PDP. An extensive analysis on the benefit of splitting deliveries is studied

in Archetti et al. [1]. In this study, we extend Nowak et al.'s analysis to the multiple vehicle version of the problem. Moreover, we analyze the benefit of splitting loads on other PDP instances from the literature. As detailed below, Nowak et al.'s instances have special characteristics that do not exist in other PDP settings. In order to explore the impact of problem characteristics on the benefit of splitting loads, we use another set of problem instances adopted from Ropke and Pisinger [12] whose characteristics are significantly different. We also verify the quality of our algorithm through a comparative analysis against the PDP results given in Nowak et al. [9]. Since these results are for the single vehicle case, we convert our solutions to a single route using a very simple algorithm.

Below, we describe the two problem sets used in our analysis and how they are adapted to create instances for MPDPSL:

(1) The first set consists of the randomly generated problems in Nowak et al. [9]. In generating this set of problems, they have used the following idea: they randomly generate a set of pickup points and another set of delivery points; then, they create a set of p-d requests from every pickup point to every delivery point. As a result, a p-d request is still one-to-one but there are multiple deliveries from a pickup location as well to a delivery location. In addition, Nowak et al. considers a single-vehicle problem which constructs a single route as a solution. Since we study the multi-vehicle case, our solution consists of multiple routes, one for each vehicle. This setting requires introducing a maximum route length restriction for each vehicle route. For each problem instance from Nowak et al., we introduce a maximum route length as a function of the distances between points.

(2) In a traditional PDP setting, each physical pickup point is usually associated with a single delivery point. In this respect, the problems in Nowak et al. may not be considered as representative of a traditional PDP setting in terms of the spatial distribution of the pickup and delivery points. For problems that are more representative of a traditional PDP setting, we resort to Ropke and Pisinger [12]. In this set of problems, all p-d requests are generated one-to-one between a randomly generated pickup point and a randomly generated delivery point. Ropke and Pisinger suppose that each vehicle starts the route from a designated point considered as the vehicle's depot. However, in MPDPSL we suppose that there exists only one depot from where all vehicles start their routes. Therefore, based on the network data in Ropke and Pisinger's instances, we have created a single vehicle depot at the center of the two-dimensional space on which the points are generated randomly. The maximum route length is used as specified in this study.

Number of Requests	Location Set	Average Imp. (%)	Average
75	1	33,11	32,04
	2	30,81	
	3	32,21	
100	1	33,09	32,45
	2	32,01	
	3	32,26	
125	1	32,28	32,07
	2	31,54	
	3	32,38	

Table 1
Improvement in the route length obtained by split loads for problems in Nowak et al. [9] for a load size range of [0.51-0.6] truckload.

For both sets of problems, the results presented in this section are the first results for MPDPSL in the literature.

3.2.1 Analysis of the benefit obtained by split loads

In Nowak et al. [9], the benefit obtained by splitting loads is studied for different load size intervals. Their results clearly show that the improvement in the route length is closely related with the size of the loads. When the loads are within 0.51-0.60 of vehicle capacity, the improvement in the route length is more significant when compared to other ranges of the load size. We want to understand if this finding is still true when the problem is solved for multiple vehicles instead of a single vehicle. For this analysis, we solve each problem instance using two versions of our algorithm: the original version that allows split loads and a modified version which does not do any split moves.

In Table 1, we present the percentage improvement in the route length for the algorithm with splits over that of the no-split version for the three problem sets in Nowak et al. [9]. The percentage improvement in the route length is more than 30% on average when the load size range is 0.51-0.60. The observed level of improvement is in line with Nowak et al.'s findings where the improvements is around 25% for the same load size range.

In order to study the benefit of splitting loads in a more traditional PDP setting where the pickup and delivery points are distributed over the service area, we solve the problems in Ropke and Pisinger [12] to obtain both the split and no-split route lengths. To observe the difference in route length improvement between different load size ranges, we have studied the load range 0.25-1.00 (using the loads in the original problem data) and the load range 0.51-0.60 (for which we have randomly generated new load data). Complete results are given

in Tables B.1-B.4 of Appendix B. For the problems with 50, 100 and 250 requests, we attain the best results over 20 restarts while the problems with 500 requests is solved only with five restarts due to excessive CPU time for a single restart of the algorithm. the resulting CPU times are provided in the Online Supplement (available at http://people.sabanciuniv.edu/guvencs/MPDPSL/). a summary of results grouped by the number of requests and load size of problem instances is presented in Figure 8. In this figure, we observe that

- the improvement in route length for the load range 0.51-0.60 is more significant when compared to the load range 0.25-1.00, and
- the improvement in route length increases as the problem size increases.

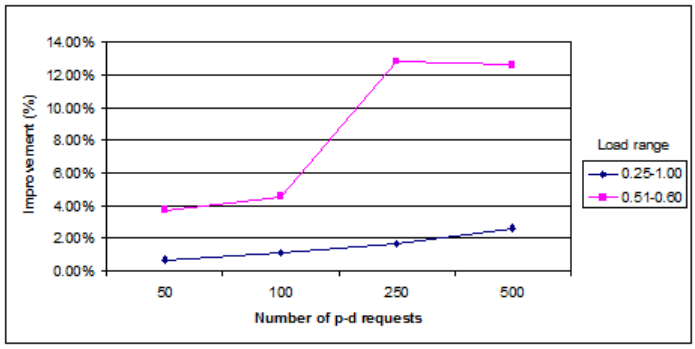


Fig. 8. Effect of the load range on the improvement in route length.

A comparison among the two load ranges help us conclude that the improvement in route length due to split loads is more significant when the load range is just above one half of a truck load (0.51-0.60) compared to a much wider load range. This observation is in parallel with the empirical analysis in Nowak et al.[9]. Yet, it is also clear that this difference becomes more significant as the problem size increases and the magnitude of improvement is not as much as it is observed in Nowak et al. Therefore, in addition to the load range size, we observe that

- spatial distribution of the pickup and delivery requests and
- number of p-d requests in the problem

also play an important role on the level of improvement in route length. In Nowak et al., although the problem is a one-to-one PDP, each pickup (delivery) point is associated with several requests. In the problems by Ropke and Pisinger [12], each pickup (delivery) point is associated with only one delivery (pickup) point. We also note that the observed improvement is not due to the algorithm used to solve the problems as the level of improvement in Nowak et al.'s experimental results is around 25-30% while it is in the range of 30-33% in ours. In addition, we suspect that the multi-vehicle version of the problem may be even more prone to improvements obtained by split loads.

The importance of spatial distribution becomes even more apparent when we look into the results from the Ropke and Pisinger [12] instances in more detail. These instances are designed in three categories according to the spatial distribution of the pickup and delivery locations: uniform, semi-clustered and clustered. In Figure 9, we display the improvement in route length both by load size and problem category. While the importance of load size is clear in this figure, one can also observe that the benefit of split loads is also affected significantly by the problem category. The largest improvement in route length is realized for clustered data, especially for larger problem sizes. Although the difference between semi-clustered and uniform data is not very significant for small problems, as the number of p-d pairs in the problem increase, semi-clustered problems also benefit greatly from splitting loads. Overall, the results indicate that the benefit to be gained from load splitting is more pronounced for problems with clustered data, even when the clustering is partial. In clustered problems, vehicle routes also tend to appear in clusters, which increases the likelihood of having multiple vehicle routes in close vicinity among which the load can be split. Similarly, the number of splitting options are higher for larger problem sizes. This observation may help explain the results obtained for problem instances from different categories and problem sizes.

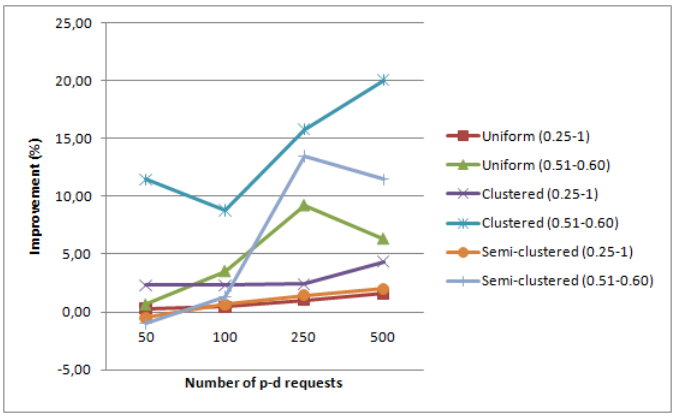


Fig. 9. Effect of the load range and spatial distribution of points on the improvement in route length.

We also note that this is the first study to report results on the problems in Ropke and Pisinger [12] for the case of split loads .

3.2.2 Comparative Analysis of Algorithm Performance

We evaluate the performance of our algorithm on the single vehicle version of the problem using a comparative analysis against the results presented in Nowak et al. [9]. The aim of our analysis is two-fold; we want to

- validate the quality/strength of our algorithm in order to justify the use of

our algorithm in obtaining the results in the previous sections, and
- show that our algorithm has the potential to be used for the single vehicle case by demonstrating an improvement upon the results obtained in Nowak et al. [9].

We particularly note that our algorithm is designed for the multiple vehicle version, and we do not make a special effort to tailor it to solve the single vehicle case. In order to obtain a single vehicle route from the multi-vehicle solution of the algorithm, we use a very simplistic greedy approach. Among all vehicle routes in the solution, we identify the two that yield the minimum route length increase when merged into a single route. Then, at every iteration we add a new route to the combined route such that the incremental route length increase is minimized. The procedure stops when all routes are merged into a single vehicle route.

For the three sets of problem instances (i.e. 75,100,125-request sets), we solve the single vehicle problems with the load factor 0.51-0.60; the results are presented in Tables 2-4. To make a fair comparison, we account for the difference in the computing power of our computer with the one used in Nowak et al. [9]. For this purpose, we present our results with two different levels of computational effort. In the scaled time approach, for each problem set, we scale the average CPU time reported in Nowak et al. by the difference in the computing power of the computers using the method presented in Dongarra [5]. As our computer is two times more efficient than the one in Nowak et al., we use half of the average CPU time reported in Nowak et al. as the scaled time. In the equal time approach, we use as much CPU time as reported in Nowak et al. without any scaling. In Tables 2-4, the first three columns describe the problem characteristics (Location Set and Load Size Set) based on Nowak et al. and the length of the route obtained with their algorithm. The fourth and fifth columns respectively show the length of the route and percentage improvement over Nowak et al.'s solution using the scaled time approach. The sixth and seventh columns show the same results when the CPU time is not scaled.

According to computational experiments reported in Tables 2-4, we obtain the following results:

- For the problem set with 75 requests,
  - in scaled time (12.75 minutes), we have improved the route length of 9 problems (out of 15) and the average improvement is 2.10% while we are only 0.90% away from the best known solution in the worst case;
  - in equal time (25.50 minutes), we have improved the route length of 13 problems and the average improvement is 2.48 %.
- For the problem set with 100 requests,

Problem (Nowak et al.)			Scaled Time		Equal Time	
Location	Load Size	Route Length	Route Length	Imp. (%)	Route Length	Imp. (%)
1	1	3830.123	3840.602	-0.27	3840.602	-0.27
1	2	3857.117	3859.936	-0.07	3828.613	0.74
1	3	3810.502	3831.256	-0.54	3809.375	0.03
1	4	3799.324	3833.559	-0.90	3833.559	-0.90
1	5	3868.957	3887.298	-0.47	3855.453	0.35
2	1	3313.483	3171.766	4.28	3171.766	4.28
2	2	3296.364	3162.015	4.08	3144.403	4.61
2	3	3203.245	3210.825	-0.24	3199.044	0.13
2	4	3266.417	3180.605	2.63	3180.605	2.63
2	5	3332.589	3188.812	4.31	3146.602	5.58
3	1	4058.369	3990.376	1.68	3954.308	2.56
3	2	4172.418	3926.42	5.90	3926.42	5.90
3	3	4090.647	3934.566	3.82	3934.566	3.82
3	4	4110.389	3936.68	4.23	3936.68	4.23
3	5	4052.233	3925.661	3.12	3908.682	3.54
			9	**2.10**	**13**	**2.48**

Table 2
Results for the single vehicle problems with 75 requests from Nowak et al. [9].

Problem (Nowak et al.)			Scaled Time		Equal Time	
Location	Load Size	Route Length	Route Length	Imp. (%)	Route Length	Imp. (%)
1	1	5073.395	5016.763	1.12	5014.602	1.16
1	2	5036.547	5077.608	-0.82	5077.608	-0.82
1	3	5029.381	5033.059	-0.07	5024.470	0.10
1	4	5012.974	4965.867	0.94	4965.867	0.94
1	5	5130.154	5022.422	2.10	5022.422	2.10
2	1	4450.058	4249.250	4.51	4216.749	5.24
2	2	4484.466	4302.366	4.06	4302.366	4.06
2	3	4473.387	4292.695	4.04	4268.062	4.59
2	4	4424.569	4283.759	3.18	4259.868	3.72
2	5	4559.259	4272.694	6.29	4247.892	6.83
3	1	5294.367	5029.347	5.01	5029.347	5.01
3	2	5371.740	5158.324	3.97	5127.881	4.54
3	3	5216.797	5149.056	1.30	5115.153	1.95
3	4	5467.788	5132.782	6.13	5132.782	6.13
3	5	5572.472	5141.317	7.74	5097.744	8.52
			13	**3.30**	**14**	**3.60**

Table 3
Results for the single vehicle problems with 100 requests from Nowak et al. [9].

- in scaled time (25.60 minutes), we have improved the route length of 13 problems (out of 15) and the average improvement is 3.30% while we are only 0.82% away from the best known solution in the worst case;

Problem (Nowak et al.)			Scaled Time		Equal Time	
Location	Load Size	Route Length	Route Length	Imp. (%)	Route Length	Imp. (%)
1	1	6020.046	5924.608	1.59	5924.608	1.59
1	2	5938.943	6020.369	-1.37	6008.256	-1.17
1	3	5977.69	5929.926	0.80	5929.926	0.80
1	4	6138.936	6022.086	1.90	6017.348	1.98
1	5	6024.26	6028.898	-0.08	5996.25	0.46
2	1	5717.536	5452.732	4.63	5410.327	5.37
2	2	5745.378	5494.878	4.36	5470.25	4.79
2	3	5667.263	5456.303	3.72	5456.303	3.72
2	4	5778.58	5428.613	6.06	5409.127	6.39
2	5	5780.014	5471.003	5.35	5430.16	6.05
3	1	6934.046	6322.772	8.82	6272.689	9.54
3	2	6918.162	6318.539	8.67	6318.539	8.67
3	3	6607.296	6330.95	4.18	6330.95	4.18
3	4	7239.787	6412.622	11.43	6404.65	11.54
3	5	6776.373	6320.865	6.72	6320.865	6.72
			13	**4.45**	**14**	**4.71**

Table 4
Results for the single vehicle problems with 125 requests from Nowak et al. [9].

  - in equal time (56.20 minutes), we have improved the route length of 14 problems and the average improvement is 3.60 %.
- For the problem set with 125 requests,
  - in scaled time (47.95 minutes), we have improved the route length of 13 problems (out of 15) and the average improvement is 4.45% while we are 1.37% away from the best known solution in the worst case;
  - in equal time (95.90 minutes), we have improved the route length of 14 problems and the average improvement is 4.71%.

These results indicate that we have improved most of the route lengths reported in Nowak et al. [9] even with an algorithm which is not tailored for the single vehicle version. Therefore, our algorithm can be considered as an efficient and effective algorithm for solving not only the multiple vehicle but also the single vehicle problems as well.

4 Concluding Remarks

In this study, we present an efficient heuristic for MPDPSL. To the best of our knowledge, this is the first algorithm in the literature for the multi-vehicle version of the problem. Due to the many possible ways of splitting a load among vehicle routes, search neighborhoods for MPDSL are of high computa-

tional complexity. In our heuristic, we employ a binary heap implementation to search the neighborhood efficiently and avoid excessive computation times. Moreover, we combine powerful features of Tabu Search and Simulated Annealing to obtain an algorithm that exhibits both intensification and diversification capabilities.

This work also presents two sets of test problems that are obtained by modifications to test instances for similar problems in the literature. Results on these problems can serve as benchmark for future research on MPDPSL. Since no computational results are available on MPDPSL, we compare the solution quality of our heuristic with Nowak et al's [9] results on the single vehicle version. Even though the heuristic is not designed for this version, we demonstrate that it is capable of producing comparable and mostly higher quality solutions in comparable time. In the absence of any benchmark results on the second problem set derived from the Ropke and Pisinger [12] instances, we present the first results and explore the benefit of split loads on this new set which have different characteristics than the first one. We observe that load splitting provides significant benefits on these instances as well, however the benefits are relatively small compared to the first problem set. We attribute the difference to the spatial distribution of the pickup and delivery points in the two problem sets. Based on our observations on the two problem sets, we conclude that both load size range and spatial distribution of the pickup and delivery points are important factors in the magnitude of benefit that can be obtained from split loads.

Our study expands the findings of Nowak et al. on the benefit of split loads to the multiple vehicle version of the problem. We hope that these results draw attention to the potential savings that can be achieved by allowing load splits in many application areas of PDP. While the proposed heuristic offers promising results, it is still worthwhile to explore exact solution approaches for the solution of MPDPSL. Experience from earlier work on similar problems suggest that exact approaches will be limited to produce optimal solutions for only very small size problems. Still, these solutions can serve as benchmark results to evaluate the performance of heuristics. Moreover, ideas from exact solution approaches can be utilized to build effective heuristics and lower bounds for the same problem. This is an avenue of research that we hope to pursue in the future.

Acknowledgements

This research has been supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 109M139.

References

[1] C. Archetti, M. W. P Savelsbergh, and M. F. Speranza. To split or not to split: That is the question. *Transportation Research Part E*, 44:114–123, 2008.

[2] C. Archetti and M. G. Speranza. The split delivery vehicle routing problem: A survey. In Ramesh Sharda, Stefan Voß, Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 103–122. Springer US, 2008.

[3] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[4] J-F. Cordeau, G. Laporte, and S. Ropke. Recent models and algorithms for one-to-one pickup and delivery problems. In Ramesh Sharda, Stefan Voß, Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 327–357. Springer US, 2008.

[5] J. J. Dongarra. Performance of various computers using standard linear equations software, (linpack benchmark report). *University of Tennessee Computer Science Technical Report*, 2010.

[6] M. Dror and P. Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–145, 1989.

[7] I. Gribkovskaia and G. Laporte. One-to-many-to-one single vehicle pickup and delivery problems. In Ramesh Sharda, Stefan VoSS, Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 359–377. Springer US, 2008.

[8] H. Li and A. Lim. Local search with annealing-like restarts to solve the VRPTW. *European Journal of Operational Research*, 150(1):115–127, 2003.

[9] M. Nowak, O. Ergun, and C. C. White III. Pickup and delivery with split loads. *Transportation Science*, 42(1):32–43, 2008.

[10] M. Nowak, O. Ergun, and C. C. White III. An empirical study on the benefit of split loads with the pickup and delivery problem. *European Journal of Operational Research*, 198(3):734–740, 2009.

[11] I. H. Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *OR Spectrum*, 17:211–225, 1995. 10.1007/BF01720977.

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

[13] S. R. Thangiah, I. H. Osman, and T. Sun. Hybrid genetic algorithm simulated annealing and tabu search methods for vehicle routing problem with time windows. *Technical Report 27, Computer Science Department, Slippery Rock University*, 1994.

Appendices

A Algorithm Description and Computational Complexity Analysis

A.1 Notation

A *pair* (p, d) consists of a pickup stop and the corresponding delivery stop with indices p and d. The (possibly partial) load quantity of pair (p, d) is q_p. $s(p, d)$ denotes the saving obtained by removing the pair (p, d) from its current route. A *solution* $S = \{R_1, R_2, \ldots, R_{|S|}\}$ is a collection of routes R that consist of pickup and delivery stops. Also, $R_{(p,d)}$ denotes the route of (p, d). $c(R)$ denotes the cost of route R and $c(S)$ denote the cost of the solution S. The tuple $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$ denotes the positions for splitting the pair (p, d) such that the pickup stops of the first and second splits are inserted after ℓ^{p1} and ℓ^{p2} and similarly, the delivery stops of the first and second splits are inserted after ℓ^{d1} and ℓ^{d2}. If the pair (p, d) is inserted rather than split, then the pickup stop is inserted after ℓ^{p1}, the delivery stop is inserted after ℓ^{d1} and $<\ell^{p2},\ell^{d2}> = \emptyset$. $c(<\ell^{p},\ell^{d}>)$ is the cost of inserting p and q after the position ℓ^p and ℓ^d, respectively. Note that by triangular inequality, $s(p, d)$ and $c(<\ell^{p},\ell^{d}>)$ are always nonnegative. Finally, $r(<\ell^{p},\ell^{d}>)$ is the residual vehicle capacity for inserting p and d after position ℓ^p and ℓ^d, respectively.

A.2 Insert/Split Phase

The detailed pseudo code for selecting a candidate move for a given pair $(p, d)$is provided in Algorithm 3. Note that $minInsert(T)$ is a function that returns the best insert position $<\ell^{p},\ell^{d}>$ where the entire load q_p can be inserted and function $minSplit(T)$ returns the best two split positions $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$ where load q_p can be served in two partial shipments on the list T. The time complexity proofs for worst and best cases of this phase are as follows.

Proposition 1 *The worst case running time of the* Insert & Split Phase *is in* $O(n^5)$ *and the best case running time is in* $\Omega(n^3)$.

Proof.
Worst Case Analysis. Since there are n pairs, the upper bound on the number of *stops* is $2nk$, where k denotes the maximum number of splits allowed for one *pair*. Between lines 9-15 of Algorithm 3, we check for all possible positions $<\ell^{p},\ell^{d}>$ such that ℓ^p comes before ℓ^d, which takes exactly,

Algorithm 3 Insert/Split

Input: $S, (p, d), Pr$ //S: Solution, (p,d): Selected pair
Output: $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$ //Best position for insert or split
$H \leftarrow makeHeap()$ //H: Binary heap of $<\ell^{p},\ell^{d}>$ with key $c(<\ell^{p},\ell^{d}>)$
$T \leftarrow \emptyset$ //T: An array of $<\ell^{p},\ell^{d}>$
$SimList \leftarrow \emptyset$ //SimList: An array of $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$
$ChosenInsert \leftarrow \emptyset$ //Initialize best insert position
$ChosenSplit \leftarrow \emptyset$ //Initialize best split positions
$isMoveChosen \leftarrow false$
for all $R_i \in S$ **do**
  for all $r(<\ell^{p},\ell^{d}>) > 0 \in R_i$ **do**
    if $c(<\ell^{p},\ell^{d}>) + c(R_i) \leq D$ **then**
      $insert(H, <\ell^{p},\ell^{d}>)$
    end if
  end for
end for
while $H \neq \emptyset$ **or** $|SimList| \leq N_{move}$ **do**
  if $r(minimum(H)) \geq q_p$ **and** $<\ell^{p},\ell^{d}>$ is not TABU **then**
    $SimList \leftarrow SimList \cup extractMin(H)$
  else
    for all $<\ell^{p},\ell^{d}> \in T$ **do**
      if $r(minimum(H)) + r(<\ell^{p},\ell^{d}>) \geq q_p$ **and** $minimum(H) - <\ell^{p},\ell^{d}>$ is not TABU **and** $c(minSplit(SimList)) > c(minimum(H)) + c(<\ell^{p},\ell^{d}>)$ **then**
        $SimList \leftarrow SimList \cup (minimum(H) - <\ell^{p},\ell^{d}>)$
      end if
    end for
    $insert(T, extractMin(H))$
  end if
end while
$temp \leftarrow CalculateTemperature(Pr)$
$ChosenInsert \leftarrow minInsert(SimList)$
$ChosenSplit \leftarrow minSplit(SimList)$
while $isMoveChosen = false$ **do**
  Extract a $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$ randomly from $SimList$
  $prob \leftarrow generateProbability()$ //Generate a number $\in (0, 1)$
  if $prob \leq e^{-\Delta C/temp}$ **then**
    if $<\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$is split **then**
      $ChosenSplit \leftarrow <\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$
    else
      $ChosenInsert \leftarrow <\ell^{p1},\ell^{d1}>-<\ell^{p2},\ell^{d2}>$
    end if
    $isMoveChosen = true$
  end if
end while
return $\arg\min(c(ChosenInsert), c(ChosenSplit))$

$$|R_i| + (|R_i| - 1) + \ldots + 1 = \frac{|R_i|(|R_i| + 1)}{2} \qquad \forall R_i \in S$$

operations for one route. Since one operation is in $O(1)$, for one route it is in $\Theta(|R_i|^2)$. For a particular solution S, i.e. $\forall R_i \in S$, it takes

$$\sum_{R_i \in S} \Theta(|R_i|^2)$$

which is in $O(n^2)$, since

$$\sum_{R_i \in S} |R_i| \leq 2nk \qquad \text{and} \qquad \sum_{R_i \in S} |R_i|^2 \leq \left(\sum_{R_i \in S} |R_i| \right)^2 \leq (2nk)^2$$

In the worst case, for each loop between 10-14, we insert an element to the Binary Heap created at the beginning of the every iteration. Since inserting an element to a heap is in $O(\log |H|)$, the entire loop takes up to $O(n^2 \log n^2) = O(n^2 2 \log n) = O(n^2 \log n)$.

The loop between 16-27 continues until the heap is empty. Since the size of the heap is in $O(n^2)$ and there is an inner loop between 20-24 that compares each element in the array T with the current $minimum(H)$, during the entire loop, $O(n^2 + (n^2 - 1) + \ldots + 1) = O(n^4)$ comparisons are made. Each comparison takes $O(1)$ and $minimum(H)$ also takes $O(1)$. Also, there is an $extractMin(H)$ operation at each iteration, which takes $O(\log |H|)$. Therefore, the entire loop takes up to $O(n^2 \log n^2 + n^4) = O(n^4)$. However, this is only for one selected pair (p,d), since there are $O(n)$ pairs to be considered, the worst case running time of the *Insert/Split Phase* is in $O(n)(O(n^2 \log n) + O(n^4)) = O(n^5)$.

Best Case Analysis. For the loop between 9-15, we still have to examine $O(n^2)$ positions $<\ell^p, \ell^d>$, however, due to the distance and capacity constraints, there might not be $O(n^2)$ elements in the heap. In fact, it is possible that there is only one element in the heap, which makes the loop between 16-27 run in $\Omega(1)$. Again, there are $O(n)$ pairs to consider, so, in the best case, the running time of the *Insert/Split Phase* is in $O(n)\Omega(n^2) = \Omega(n^3)$. ■

Even though we have a worst case running time of $O(n^5)$, computational experiments showed that on average, there are $O(n)$ elements in the heap, which makes the average running time of the phase $O(n)(O(n^2 \log n) + O(n^2)) = O(n^3 \log n)$.

B Results of Ropke Problem Instances

Load Range	0.25-1				0.51-0.60			
Instance	Initial	Split	No-split	Imp. (%)	Initial	Split	No-split	Imp. (%)
A	17621.58	15677.49	15695.30	0.11	17379.03	16729.01	16676.01	-0.32
B	18404.11	16069.38	16082.02	0.08	17402.16	16331.27	16864.73	3.16
C	15707.79	14219.00	14264.23	0.32	15510.45	15187.87	14905.13	-1.90
D	16779.31	15014.44	15121.69	0.71	16662.84	15879.59	16139.55	1.61
E	12047.49	10673.44	11025.65	3.19	12004.37	10018.12	12101.74	17.22
F	12463.43	9300.00	9886.72	5.93	10238.10	8574.31	10586.80	19.01
G	10814.60	9733.02	9767.69	0.35	13060.61	9635.78	10683.87	9.81
H	9215.83	8269.58	8250.69	-0.23	9314.01	9197.99	9186.60	-0.12
I	15875.92	13068.48	13154.92	0.66	14968.59	14612.21	14551.36	-0.42
J	14671.55	12830.79	12626.18	-1.62	14349.02	13416.36	13165.43	-1.91
K	14411.39	12501.61	12429.17	-0.58	14301.42	13164.66	12942.81	-1.71
L	15560.98	13595.82	13542.14	-0.40	16399.50	14679.65	14680.40	0.01
Average				0.71				3.70

Table B.1
Results of the problems in Ropke and Pisinger [12] with 50 nodes comparing the tour length of split solutions versus no-split solutions.

Load Range	0.25-1				0.51-0.60			
Instance	Initial	Split	No-split	Imp. (%)	Initial	Split	No-split	Imp. (%)
A	31003.41	25831.58	25383.32	-1.77	29543.02	27242.04	27857.50	2.21
B	31383.95	24758.19	25371.87	2.42	28386.51	25947.79	28233.49	8.10
C	31959.82	25448.51	25603.29	0.60	29235.65	26984.26	27521.78	1.95
D	31939.37	27748.51	27893.86	0.52	31403.10	29055.41	29605.19	1.86
E	18055.41	15767.00	16083.98	1.97	18563.96	15814.11	17502.66	9.65
F	19714.36	15276.41	15359.55	0.54	18129.46	15804.52	18005.28	12.22
G	19964.11	14453.83	14795.43	2.31	22121.55	18443.08	17692.47	-4.24
H	21055.13	16226.10	16970.18	4.38	18559.40	15644.92	18985.21	17.59
I	28674.25	24547.88	24747.82	0.81	29114.47	24957.27	27098.02	7.90
J	25719.10	22324.67	22376.54	0.23	25452.34	23496.20	23148.95	-1.50
K	31055.48	22755.02	22797.16	0.18	32118.90	26515.08	26074.91	-1.69
L	26584.23	21157.30	21440.29	1.32	24778.18	23917.45	24060.70	0.60
Average				1.13				4.55

Table B.2
Results of the problems in Ropke and Pisinger [12] with 100 nodes comparing the tour length of split solutions versus no-split solutions.

Load Range	0.25-1				0.51-0.60			
Instance	Initial	Split	No-split	Imp. (%)	Initial	Split	No-split	Imp. (%)
A	68806.59	57198.89	57772.20	0.99	64657.44	58093.91	63970.44	9.19
B	65680.36	54902.64	55382.50	0.87	63230.91	56972.61	62538.64	8.90
C	66912.80	56401.98	56881.15	0.84	63353.50	56478.37	62569.26	9.73
D	66896.49	57699.49	58485.03	1.34	64698.62	58549.06	64391.78	9.07
E	41316.27	29873.35	30198.33	1.08	37890.81	30984.94	37486.16	17.34
F	36325.18	28862.15	29818.50	3.21	33552.50	27048.29	33143.07	18.39
G	38698.26	29535.93	30311.05	2.56	34876.53	28954.03	34393.67	15.82
H	38990.89	30016.04	30897.73	2.85	36151.47	31500.77	35678.94	11.71
I	62167.25	50238.13	50891.55	1.28	57721.25	49509.76	56599.69	12.53
J	67448.83	53845.75	54581.37	1.35	61133.19	52787.53	60428.53	12.64
K	62832.43	50405.45	51304.41	1.75	57895.87	49065.95	56931.24	13.82
L	67103.88	53191.29	53935.14	1.38	63662.70	52949.00	62274.19	14.97
Average				1.63				12.84

Table B.3
Results of the problems in Ropke and Pisinger [12] with 250 nodes comparing the tour length of split solutions versus no-split solutions.

Load Range	0.25-1				0.51-0.60			
Instance	Initial	Split	No-split	Imp. (%)	Initial	Split	No-split	Imp. (%)
A	122227.20	107061.00	108668.60	1.48	117079.50	111145.10	116659.90	4.73
B	130599.30	113899.70	115462.70	1.35	124167.50	114929.60	123334.90	6.82
C	130002.80	109031.70	110876.10	1.66	120185.00	111553.50	119672.30	6.78
D	131867.20	111276.30	113370.10	1.85	121301.50	112057.30	120479.70	6.99
E	90946.61	66438.42	69280.32	4.10	82358.28	65019.84	81870.98	20.58
F	97917.73	72015.02	75270.35	4.32	90732.82	70727.76	89827.22	21.26
G	99146.69	72373.64	75801.21	4.52	91885.86	71640.62	91383.87	21.60
H	82228.20	62231.76	65090.16	4.39	76734.43	63446.30	76285.07	16.83
I	115321.50	93708.45	96401.48	2.79	108328.90	94601.00	107356.60	11.88
J	123209.50	101224.90	102132.50	0.89	113252.10	99481.13	112167.70	11.31
K	121653.40	102869.20	104487.50	1.55	114033.40	100438.30	113421.30	11.45
L	120380.00	100849.10	103687.70	2.74	115473.00	101723.70	114501.20	11.16
Average				2.64				12.62

Table B.4
Results of the problems in Ropke and Pisinger [12] with 500 nodes comparing the tour length of split solutions versus no-split solutions.