Scheduling for Transfers in Pickup and Delivery Problems with Very Large Neighborhood Search

Brian Coltin and Manuela Veloso
{bcoltin, veloso}@cs.cmu.edu
School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213

Abstract

In pickup and delivery problems (PDPs), vehicles pickup and deliver a set of items under various constraints. We address the PDP with Transfers (PDP-T), in which vehicles plan to transfer items between one another to form more efficient schedules. We introduce the Very Large Neighborhood Search with Transfers (VLNS-T) algorithm to form schedules for the PDP-T. Our approach allows multiple transfers for items at arbitrary locations, and is not restricted to a set of predefined transfer points. We show that VLNS-T improves upon the best known PDP solutions for benchmark problems, and demonstrate its effectiveness on problems sampled from real world taxi data in New York City.

Introduction

In a pickup and delivery problem (PDP), a set of vehicles pick up and deliver a set of items. The goal is to deliver as many items as possible at the lowest cost while obeying a set of constraints, such as time windows and capacities. PDPs are applicable to many scenarios, such as logistics problems, delivery robots, and transportation problems.

In the PDP with Transfers (PDP-T), we consider *transferring* items between vehicles. Transfers expand the search space of possible solutions exponentially, for a problem that is already NP-hard. However, with transfers, lower cost solutions can be found for many problems, as has been previously shown (Masson, Lehuédé, and Péton 2013a; Cortés, Matamala, and Contardo 2010).

We introduce the Very Large Neighborhood Search with Transfers (VLNS-T) algorithm to solve the PDP-T. Similar to the VLNS algorithm for the PDP (Ropke and Pisinger 2006), VLNS-T forms a schedule with simulated annealing, where "neighboring" states are found by removing a randomized set of items from the proposed scheduled, then reinserting the items into the schedule using a set of heuristics. We introduce two new insertion heuristics which insert transfers into the schedule, as well as an algorithm to determine the execution times of actions in a schedule with transfers while obeying all PDP-T constraints. We demonstrate VLNS-T's effectiveness on both benchmark instances and problems generated from New York City taxi data.

Related Work

The PDP has been solved optimally (Ropke, Cordeau, and Laporte 2007; Lu and Dessouky 2004) , and with heuristics (Lu and Dessouky 2006; Xu et al. 2003) and metaheuristics such as tabu search (Nanry and Wesley Barnes 2000; Li and Lim 2001), genetic algorithms (Jung and Haghani 2000), and VLNS (Ropke and Pisinger 2006).

Many large logistics systems, such as the airline industry, rely on hub and spoke networks which in effect transfer items between vehicles (Bryan and O'Kelly 2002). These problems differ from the PDP-T in that an algorithm does not plan for the transfers of individual items.

A related problem is the PDP with fixed "transshipment" points, where vehicles drop off items for another vehicle to pick up later. Researchers have found the optimal solution (Cortés, Matamala, and Contardo 2010), introduced heuristics (Mitrovic-Minic and Laporte 2006), and studied the effects of adding a single transshipment point (Nakao and Nagamochi 2008). An approximation algorithm has been introduced for the preemptive PDP, where objects may be dropped off at pickup points and retrieved by other vehicles later (Gørtz, Nagarajan, and Ravi 2009).

Transfers at any pickup or delivery location have been considered in the construction of heuristics for the PDP-T (Thangiah, Fergany, and Awan 2007; Shang and Cuff 1996). Transfers at arbitrary locations have been considered for an insertion heuristic for the dynamic PDP-T without time windows (Bouros et al. 2011).

A VLNS-based algorithm to solve PDPs with transfers has been previously proposed (Masson, Lehuédé, and Péton 2013b; 2013a; Masson, Lehuédé, and Péton 2014). Our work differs in that we allow a single item to be transferred multiple times, consider additional constraints such as maximum item transportation times and maximum route durations, allow transfers at any location, consider a cost for transfers, and require both vehicles to be present simultaneously for a transfer.

VLNS-T builds upon our own previous work of developing algorithms for simplified variants of the PDP-T, both for deploying autonomous indoor robots to pickup and deliver items with transfers and to solve ridesharing problems (Coltin and Veloso 2012; 2013; 2014).

The Pickup and Delivery Problem with Transfers

In a PDP, a set of vehicles $v \in V$ must pick up and deliver a set of items $p \in P$. Each vehicle v has:

- A starting location $\mathit{start}(v)$, and ending location $\mathit{end}(v)$. The ending location may be $\emptyset$, a wildcard location indicating that the vehicle may end anywhere.
- A capacity $\mathit{cap}(v)$, the maximum quantity of items that can be transported at one time.
- A velocity v_{vel}, traveling a distance d takes time d/v_{vel}.
- A time interval $W(v) = [e(v), l(v)]$ in which all actions must be completed. $l(v)$ may be ∞.
- A maximum route duration $\mathit{mrd}(v)$ v's operation time may not exceed, representing fuel or driver availability.
- A cost of transferring an item with the vehicle $c_t(v)$.

Each item m has the following properties:

- A starting location $\mathit{start}(m)$ and ending location $\mathit{end}(m)$.
- A pickup time window $W_p(m) = [ep(m), lp(m)]$ and delivery time window $W_d(m) = [ed(m), ld(m)]$ in which the item must be picked up and delivered in.
- An integer demand $\mathit{dem}(m)$. The total demand currently transported on a vehicle v cannot exceed $\mathit{cap}(v)$.
- A maximum transport time $\mathit{mtt}(m)$, the maximum time between object pickup and delivery.
- A set of allowed vehicles $\mathit{AV}(m)$ which can transport m.
- A time it takes to pick up the item $\delta_{\mathtt{p}}(m)$, to deliver the item $\delta_{\mathtt{d}}(m)$, and to transfer the item $\delta_{\mathtt{t}}(m)$.

The goal is to form a *schedule*, S, to deliver as many items as possible at the lowest cost, where $S = \cup_{v \in V} S^v$ is a set of schedules for all the vehicles. $S^v = (S_1^v, S_2^v, \ldots, S_n^v)$ is the sequence of actions $s_i^v \in A$ executed in order by vehicle v. Every action $a \in A$ occurs at a location $\mathit{loc}(a)$. The vehicle arrives at time $b(a)$ and executes a at time $t(a)$ which must fall in the interval $W(a) = [e(a), l(a)]$ and takes time $\delta(a)$ to execute. The allowed action types in A are:

- START: The first action of every plan. If START $= a$, then $\mathit{loc}(a) = \mathit{start}(v)$, $W(a) = W(v)$, $\delta(a) = 0$.
- END: The final action of every plan. If END $= a$, then $\mathit{loc}(a) = \mathit{end}(v)$. $W(a) = W(v)$, $\delta(a) = 0$.
- PICKUP(m): Pick up item m. If PICKUP$(m) = a$: $\mathit{loc}(a) = \mathit{start}(m)$, $W(a) = W_p(m)$, $\delta(a) = \delta_{\mathtt{p}}(a)$.
- DELIVER(m): Deliver item m. If DELIVER$(m) = a$: $\mathit{loc}(a) = \mathit{end}(m)$, $W(a) = W_d(m)$, $\delta(a) = \delta_{\mathtt{d}}(a)$.
- RECEIVE(v_2, m, l): A vehicle v_1 receives m from v_2 at location l. If RECEIVE$(v_2, m, l) = a$: $W(a) = [ep(m), ld(m)]$, $\delta(a) = \delta_{\mathtt{t}}(a)$.
- TRANSFER(v_2, m, l): A vehicle v_1 transfers m to v_2 at l. If TRANSFER$(v_2, m, l) = a$: $W(a) = [ep(m), ld(m)]$, $\delta(a) = \delta_{\mathtt{t}}(a)$. S^{v_2} has $\mathit{pair}(a) =$ RECEIVE(v_1, m, l) and $t(a) = t(\mathit{pair}(a))$.

A valid schedule has each PICKUP result in a delivery, possibly through a sequence of transfers, without violating any constraints. The objective is to minimize $cost(S) = \alpha D(S) + \beta \sum_{v \in V} t_{op}(v) + \gamma |\mathit{UND}(S)| + \sum_{v \in V} NT(S^v) c_t(v)$, where $\mathit{UND}(S)$ is the set of undelivered items in S, $t_{op}(v) = t\left(S^v_{|S^v|}\right) - t(S_1^v)$, $NT(S^v)$ is the number of transfers in S^v, and

$$D(S) = \sum_{v \in V} \sum_{i=2}^{|S^v|} d\left(\mathit{loc}(S_{i-1}^v), \mathit{loc}(S_i^v)\right).$$

The objective minimizes the distance travelled, the time the vehicles in operate, and the number of items not delivered.

Savings from Transfers

How much of an improvement in solution cost can we expect to achieve from transfers? For any problem, we do know that the cost of the optimal solution to the PDP-T is at most the cost of the optimal PDP solution, since any valid solution to the PDP is also a valid solution to the PDP-T. We can bound the potential improvement due to transfers, but only for certain classes of problems.

Theorem 1. *For a PDP with only vehicle and item starting points and destinations (with unbounded time windows and capacities) the cost of the optimal PDP solution is at most twice the cost of the optimal PDP-T solution in terms of total distance ($\beta = 0$).*

Proof. Let the schedule R be the optimal solution to the PDP-T, without START or END actions. We give an algorithm to form a schedule S such that $cost(S) \leq 2cost(R)$: *1)* Initialize S such that $\forall v \in V$ $S^v = []$, a schedule with no actions. Set $v = v_1$, $a = R_1^v$, and $q = []$ (an empty stack). *2)* Mark a as visited. If a is not a transfer action, append a to S^v. Otherwise, if a' is the first unvisited action in $R^{v(\mathit{pair}(a))}$ (where $v(a)$ is the vehicle that executes action a) before $\mathit{pair}(a)$, mark $\mathit{pair}(a)$ as visited, push a to q, set $a = a'$, and go to *(2)*. If no such unvisited action a' exists, do nothing, skipping the transfer action. *3)* If a is not the last action in v's schedule, proceed to *(4)*. If q is not empty, pop an action $R_i^{v_j}$ from q, set $a = R_{i+1}^{v_j}$, and go to *(2)*. Otherwise, if q is empty, choose any vehicle n with unvisited actions. Set $a = R^n$, $v = n$. If no such vehicle n exists, then for all $v \in V$, prepend a START and append an END action to S^v, and terminate. *4)* Where $a = R_i^{v_j}$, set $a = R_{i+1}^{v_j}$. If a is marked as visited and q contains an action from v's schedule, pop an action $R_i^{v_j}$ from q, set $a = R_{i+1}^{v_j}$, and go to *(2)*. If a is marked as visited, go to *(4)*, otherwise go to *(2)*.

The algorithm creates a valid schedule that delivers all items. Every action before a transfer action on any vehicle is now executed on the same vehicle before all actions after the transfer, so every DELIVER action is preceded by the corresponding PICKUP. If the actions in R are seen as nodes in a graph, we traverse each edge in R at most twice in constructing S, so $cost(S) \leq 2cost(R)$. □

Furthermore, PDP problems exist where the optimal solutions with transfers is half the cost of the optimal solution

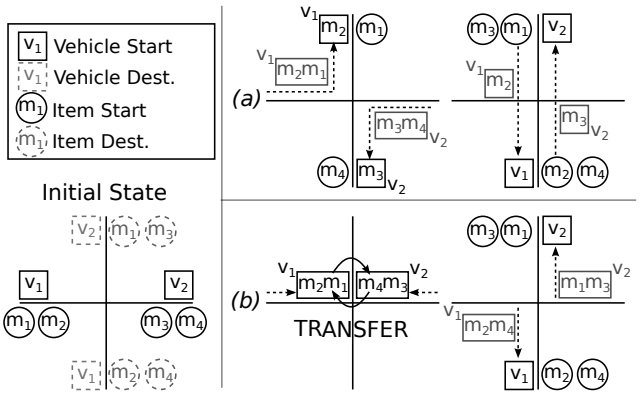

Figure 1: Two vehicles deliver two items to two locations. The cost to deliver the items is *(a)* 8 without transfers and *(b)* 4 with transfers.

without transfers. See Figure 1. For problems with capacities and time windows, this proof does not hold.

Transfers can have greater potential benefits with different metrics, such as minimizing the delivery time. Consider n vehicles at a hub which must deliver n items to distinct locations a distance 1 away from the hub like spokes on a wheel. All n items and a single vehicle begin a large distance D from the central hub. Without transfers, the single vehicle delivers all items at a distance and time cost of $D + 2n - 1$. With transfers, the single vehicle takes all items to the hub and transfers to the other vehicles for final delivery. The distance is reduced to $D + n$, and the delivery time becomes $D + 1$, which approaches a factor of n improvement.

Solving the PDP-T

Our algorithm to search for solutions to the PDP-T consists of three parts. From highest to lowest level, these are:

1. **Very Large Neighborhood Search**. The VLNS algorithm uses simulated annealing to randomly choose "neighboring" schedules and iteratively improve the schedule. Neighboring schedules are formed by removing random items and reinserting them with heuristics.
2. **Greedy Item Insertion Heuristics**. These heuristics insert items into the schedule (potentially with transfers) to find a new "neighboring" schedule.
3. **Valid Schedule Determination**. The insertion heuristics generate a list of actions for each vehicle, but not their execution times. Another routine computes the best times for the actions and determines if the schedule is valid.

We discuss these components of the algorithm in order.

Very Large Neighborhood Search with Transfers

VLNS-T is based on the Adaptive VLNS algorithm for the PDP without transfers (Ropke and Pisinger 2006), a variant of simulated annealing in which the neighborhood of states is "very large." In this case we remove random items from the schedule and then reinsert them with multiple heuristics to find "neighbors."

Algorithm 1 `vlns`(S): Form a schedule with VLNS from starting schedule S.

$S_{best} \leftarrow S$
$T \leftarrow \frac{\omega}{-\ln 0.5} cost(S, \gamma = 0)$
for $i \in 1, .., N$ **do**
  $q \leftarrow$rand. integer in $[\texttt{min}(4, |P|), \texttt{min}(100, \xi|P|)]$
  $R \leftarrow \texttt{random\_items}(S, q)$
  $S' \leftarrow \texttt{remove\_items}(S, R)$
  $S' \leftarrow \texttt{insert\_items}(S', \textit{UND}(S))$
  if $cost(S') < cost(S_{best})$ **then**
    $S_{best} \leftarrow S'$
  if $random() < \exp -(cost(S') - cost(S))/T$ **then**
    $S \leftarrow S'$
  $T \leftarrow cT$
return S_{best}

Alg. 1 shows the VLNS-T algorithm. On each iteration, a random number of items q are removed from the schedule using one of several heuristics. Next, a second heuristic attempts to insert items that are not part of the schedule. The new schedule is accepted with a probability based on the temperature, which decreases over time. The initial temperature is based on the cost of the initial schedule, but with $\gamma = 0$ to ignore undelivered items. We use the same parameter values as VLNS.

For the `random_item` and `insert_item` routines, the VLNS algorithm is *adaptive* in the sense that it chooses from sets of heuristics and learns over time which removal and insertion heuristics are the most effective. For each insertion or removal heuristic i, we compute a weight $w_{j,i}$ over each adaptive "segment" j of length τ iterations of VLNS. Removal and insertion heuristics are selected randomly in proportion to their weight. Initially all heuristics have equal weight. The weight is modified by a different amount depending on whether the heuristic gives a new best overall solution, a new solution better than the previous solution, or a newly accepted solution worse than the current solution.

Removal Heuristics For `random_item`, we use the three heuristics introduced for VLNS: the *Shaw* removal heuristic, the *random* removal heuristic, and the *worst* removal heuristic (Ropke and Pisinger 2006).

The *Shaw* heuristic chooses items similar in distance, time, and demand to remove based on a relatedness metric which depends on location, time, and demand, choosing the first item at random and the remainder in proportion to their relatedness. The *random* removal heuristic simply selects q distinct items at random to remove, with equal probability. For the *worst* removal heuristic, items are selected for removal at random in proportion to the difference in cost of the current schedule and the current schedule with the item in question removed. With this heuristic, items which are more costly to deliver are more likely to be removed.

Insertion Heuristics Two greedy heuristics insert items into the schedule with transfers. The first, `split_routes`, takes existing pickup and delivery item route segments from

a single vehicle and splits them with another vehicle. The second, `insert_item_with_transfer`, searches to insert an item into the schedule with at most a single transfer.

Both heuristics are greedy but in different ways. Each relies on another heuristic to insert item pickups and deliveries into the schedule without transfers, the same heuristic used for VLNS. This heuristic is called `greedy_insert`$(items, regret, noise)$ and takes as parameters the items to insert, a value for the "regret," which is an integer, and an amount of noise to apply to the objective function. If we define $c_{i,j}$ as the j^{th} best cost to insert item i over all points to insert PICKUP and DELIVERY actions, we iteratively choose item i to insert such that $c_{i,regret} - c_{i,1}$ is maximized. VLNS-T chooses from a family of `split_routes` heuristics, with regrets of 1, 2, 3 and 4. For each value of regret, there are two choices of noise: one with 0 noise and one with noise $\eta \max d(l_1, l_2)$ where the noise is proportional to the maximum distance between any two locations specified in the problem. When we compute the cost in the greedy algorithm, we sample from $cost(S) \sim max(0, N(cost(S), \sigma^2))$ to increase the diversity of generated solutions. For further details regarding `greedy_insert`, see the original VLNS algorithm.

Greedy Insertion with Transfers

Given an initial schedule S and a set of items M_{new}, our goal is to return a schedule S' of low cost with the items M_{new} added into S'. It is prohibitively expensive to search over all possible schedules, so we use two greedy heuristics which iteratively insert individual items at a time. First we discuss a subroutines for inserting transfer points into vehicle schedules before delving into the higher-level greedy heuristics.

Transfer Insertion Both heuristics rely on the algorithm `insert_split`$(S, v_1, v_2, m, T_{rec}, T_{send}, c_{best})$ to insert transfers into the schedule. This routine finds the lowest cost schedule to transfer item m from vehicle v_1 to vehicle v_2. Vehicle v_1 receives item m with action T_{rec} (either a PICKUP or TRANSFER action) which must be inserted into S, or with an action that is already part of the schedule, in which case $T_{rec} = \emptyset$. Likewise the action T_{send} may be a DELIVER or RECEIVE action to insert into S^{v_2} after the new TRANSFER action, or $\emptyset$ if this action is already part of the plan. The `insert_split` algorithm searches over all possible places to insert the new RECEIVE and TRANSFER actions, as well as T_{rec} and $T_{deliver}$ (if applicable), into S,

For each pair of insertion points for the new transfer actions, we compute the location of the transfer point with `find_transfer_point`(a_1, a_2, b_1, b_2), which computes a transfer point given the endpoints to two line segments, a and b, between which the new transfer point is inserted on the two vehicles' schedules. On the Euclidean plane, we return the lines' intersection or the endpoints' mean if they do not intersect. This point is not necessarily the optimal transfer point, which may depend on the timing of the two vehicles' actions, but it is a reasonable and computationally inexpensive choice. A different method may be used to find transfer points, such as choosing from a suitable list.

Algorithm 2 `split_routes`$(S, M_{new}, r, \sigma^2)$: Insert items M_{new} into the schedule S, with the specified regret r and noise σ^2.

```
S ← greedy_insert(S, M_new, r, σ²)
M_new ← M_new \ UND(S)
while |M_new| > 0 do
    Pop random item m from M_new
    V_m ← set of vehicles that transport m in S
    S_b ← S, c ← cost(S), c_b ← N(c, σ²)
    for v_1 ∈ V_m, v_2 ∈ V \ V_m do
        S' ← split_single_route(S, v_1, v_2, m, c − cost(S_b))
        c' ← N(cost(S'), σ²)
        if c' < c_b then
            S_b ← S', c_b ← c'
    if S_b ≠ S then
        S ← S_b, M ← M ∪ {m}
return S
```

For every considered set of action insertions, the satisfiability of the proposed schedule's time constraints is checked with the `update_times` algorithm, which is explained in a later section. We also check that the new actions we insert do not create a cycle of transfers such that the vehicles deadlock. To check this, we start from the actions following each of the two provided paired actions and move forward through the schedule, checking that we do not later reach one of the two initial actions again.

Greedy Route Splitting The first heuristic, shown in Alg. 2, first greedily inserts items into the plan without transfers. Then, it selects random items and greedily inserts the best transfers into their schedules using `split_single_route`. If a transfer can be inserted, we attempt to insert more transfers recursively. Due to the increased possibilities for where to insert transfers and the consequential increase in computational cost, unlike `greedy_insert`, we split the route of one item at a time instead of choosing the best split over all routes.

The subroutine `split_single_route`(S, v_1, v_2, m), takes segments of each item's route on individual vehicles and "splits" those segments with other vehicles. For example, if S^{v_1} has the actions $a_1 =$ PICKUP(m) and $a_2 =$ DELIVER(m) (these could be transfer actions instead), the heuristic attempts to make two insertions using `insert_split`. First, it attempts to remove a_1 from S^{v_1}, insert RECEIVE$(v_2, m,)$ in its place, and add PICKUP(m) and TRANSFER$(v_1, m,)$ into S^{v_2}. Second, it attempts to remove a_2 from S^{v_1}, insert TRANSFER$(v_2, m,)$ instead, and add RECEIVE$(v_1, m,)$ and DELIVER(m) to S^{v_2}.

Split Item Insertion For the second heuristic, instead of adding transfers to existing item delivery routes, we plan for a single transfer point from the beginning. We go through the items in a random order, and attempt to insert each in the schedule without transfers. We then apply the `insert_split` routine to see if a delivery can be made at lower cost by using a single transfer point, exploring the pos-

Algorithm 3 `fslack`$(action, M_n, M_{ex})$: Determine the time a's execution can be delayed without violating additional constraints. Returns a tuple containing the slack, the total waiting time, and whether the slack depends on pickup actions to be delayed which are not in M_{ex}, the set of items which have already had their pickup actions delayed.

$slack \leftarrow \infty, \Sigma_{wait} \leftarrow 0, redo \leftarrow$ False
for a from $action$ to end of plan **do**
  if $a \neq action$ **then**
    $\Sigma_{wait} \leftarrow \Sigma_{wait} + (t(a) - b(a))$
    if a is TRANSFER or RECEIVE **then**
      $(s_1, \Sigma_1, r_1) \leftarrow$ `fslack`(a, M_n, M_{ex})
      $(s_2, \Sigma_2, r_2) \leftarrow$ `fslack`$(pair(a), M_n, M_{ex})$
      $s_1 \leftarrow s_1 + \Sigma_{wait}, s_2 \leftarrow s_2 + \Sigma_{wait}$
      $\Sigma_{wait} \leftarrow \Sigma_{wait} + \min(\Sigma_1, \Sigma_2)$
      if $s_1 < slack$ **then**
        $slack \leftarrow s_1, redo \leftarrow r_1$
      if $s_2 < slack$ **then**
        $slack \leftarrow s_2, redo \leftarrow r_2$
      return $(slack, \Sigma_{wait}, redo)$
  $redo' \leftarrow False, slack' \leftarrow l(a) - t(a)$
  if $a =$ DELIVER(m), $\exists mtt(m), m \notin M_n$ **then**
    $x \leftarrow t(m_{delivery}) - (t(m_{pickup}) + \delta(m_{pickup}))$
    $slack' \leftarrow \max(0, \min(slack, mtt(m) - x))$
    $redo' \leftarrow (m \notin M_{ex})$
  $slack' \leftarrow slack' + \Sigma_{wait}$
  if $slack' < slack$ **then**
    $slack \leftarrow slack', redo \leftarrow redo'$
return $(slack, \Sigma_{wait}, redo)$

sible insertion points for the PICKUP, DELIVER, RECEIVE and TRANSFER actions. This heuristic is more expensive than the previous since it searches for insertion points for four actions over all pairs of vehicles, but finds some solutions the first heuristic cannot.

Determining Action Execution Times

Finally, we discuss the lowest level of the PDP-T algorithm, determining the execution times of the actions in a schedule and whether the schedule is valid. The `update_times`(S, V) routine is an extension of a time determination algorithm presented for a PDP Tabu search algorithm (Cordeau and Laporte 2003) which computes times when transfers are involved. The function makes use of two key subroutines:

- `earliest_times`$(S, [(a_1, t_1), \ldots])$: Given a list of actions a_i and corresponding execution times t_i, update the actions following a_i in the schedule to be executed as early as possible, obeying only time window constraints. This function is Step 2 in the algorithm detailed for Tabu search (Cordeau and Laporte 2003).We extend this algorithm for transfers such that when a transfer action a's times are updated, we update the times of the actions following $pair(a)$.
- `fslack`(a, M_n, M_{ex}): Determine the amount of time the execution of task a can be delayed without violating additional constraints. These constraints include the transport time constraints for only items in M_{ex}. See Alg. 3 for details. The slacks are computed recursively for transfer actions. Note that if the lowest slack occurred due to an item's maximum transport time constraint that has not yet been satisfied, we may need to redo the computation later.

Given these two routines, the `update_times`(S, V) algorithm executes the following steps to update the action arrival and execution times and determine schedule validity:

1. *Determine affected vehicles.* Add all vehicles to V which transfer to or from vehicles in V. The times for these vehicles' actions must also be updated.
2. *Compute the earliest possible execution times.* Call the `earliest_times`(S, A, T) function, where A contains the initial action for each vehicle $v \in V$ and T contains the earliest possible start time $e(v)$. If any time window constraints are not satisfied, the schedule is invalid.
3. *Enforce maximum vehicle route durations.* Call `earliest_times`(S, A, T) again with the initial actions A, but with $t \in T$ corresponding to $a \in A$ for vehicle v set to $e(v) + \min(slack, sum)$ where $(slack, sum, redo) =$ `fslack`$(a, \emptyset, \emptyset)$. Doing so delays the start of each vehicle's execution as much as possible while still obeying time windows. If after this step any vehicle v's route duration exceeds $mrd(v)$, the schedule is invalid. If the problem has no maximum route durations, this step can be skipped.
4. *Enforce maximum transportation times.* For each item m, in increasing order of $t(m_{pickup})$, delay the action execution by time $\min(slack, sum)$ where $(slack, sum, redo) =$ `fslack`$(m_{pickup}, \emptyset, M_{ex})$. After the times are updated with the `earliest_times` routine, add m to M_{ex}, and, if $redo$ is True, add m to a list of actions that must be redone. After attempting to delay every item m, remove the items that were marked for redos from M_{ex} and attempt to delay their execution again until no items are left. Retrying may be necessary because the computation of the forward slack assumes that all items are either in M_{ex} with their time constraints already enforced, or the items are picked up after the starting action. Without transfers this is always the case, but with transfers the item may be picked up on a different vehicle such that the maximum transportation time constraint has not yet been enforced. If any maximum transportation time constraint is violated afterwards the schedule is invalid.

Experiments

We introduce several simple problems that demonstrate the advantages of transfers, before showing results from complex benchmark problems and problems based on taxi data.

Example Problems

As a basic test to illustrate the potential of transfers, we ran VLNS-T on the problem in Figure 1. VLNS-T found the expected solution with a factor of two improvement.

In the second problem, ten hubs are evenly distributed around a circle of radius 50 on the Euclidean plane. One

Problem			Tabu	VLNS		VLNS-T		
#	$\|V\|$	$\|M\|$	Cost	Cost	Time	Cost	Tr.	Time
01	3	24	190.02	190.02	0.06	**186.46**	8	0.23
02	5	48	302.08	303.39	0.18	**290.89**	8	1.62
03	7	72	532.08	544.00	0.35	**531.02**	13	4.78
04	9	96	572.78	581.39	0.65	**558.15**	15	10.89
05	11	120	636.97	640.91	0.92	**629.02**	23	29.73
06	13	144	801.40	805.33	1.38	**772.02**	32	60.55
07	4	36	291.71	291.71	0.11	**288.82**	6	0.50
08	6	72	494.89	505.74	0.34	**480.17**	12	4.68
09	8	108	672.44	675.64	0.82	**639.64**	21	18.90
10	10	144	878.76	873.58	1.44	**861.73**	18	52.85
11	3	24	**164.46**	**164.46**	0.07	**164.46**	0	0.17
12	5	48	296.06	297.67	0.24	**292.22**	9	1.43
13	7	72	493.30	491.92	0.43	**479.75**	14	4.20
14	9	96	535.90	550.37	0.85	**527.31**	16	13.38
15	11	120	589.74	589.67	1.31	**584.60**	24	33.86
16	13	144	743.60	754.95	1.80	**736.82**	24	47.91
17	4	36	248.21	248.21	0.13	**245.69**	4	0.40
18	6	72	462.69	466.82	0.45	**457.82**	10	5.17
19	8	108	601.96	600.24	1.14	**585.36**	11	17.31
20	10	144	798.63	811.36	2.00	**787.36**	18	51.41

Table 1: Cordeau benchmark results. All times are in hours. The "Tabu" column gives the best results for Tabu search (Cordeau and Laporte 2003), the "VLNS" section for one run of VLNS (Ropke and Pisinger 2006), and the "VLNS-T" section for one run of our algorithm. The "Tr." column is the number of transfers.

vehicle begins at each hub, and its destination is its starting hub. Items begin at random hubs, and are assigned a random destination from the three hubs on the opposite side of the circle. All time windows begin at time 0 and end at time 200. The vehicles have capacity 5 and maximum route duration 150. Ten problem instances with $|M| = 15$ were all solved by the PDP-T algorithm, but no solution could be found for the PDP, since the maximum route duration of 150 prevents vehicles from delivering items to the other side of the circle and returning to their starting positions.

These two examples serve to demonstrate that the addition of transfers to the PDP can drastically reduce the solution cost and make previously infeasible problems feasible.

Benchmark Problems

We run VLNS-T on a set of benchmark problems (Cordeau and Laporte 2003), comparing to reported results without transfers from a Tabu search heuristic and to our own implementation of VLNS (Ropke and Pisinger 2006). All vehicles begin and end their routes at the same central depot. The vehicles have maximum route durations and capacities, and the items have maximum transportation times. We set $\forall v \in V\ c_t(v) = 0.25$, run 25000 iterations of VLNS-T, and optimize for distance ($\alpha = 1$, $\beta = 0$).

Table 1 shows the results. VLNS-T finds lower cost solutions for 19 of 20 problems, and still finds the best known solution for the final problem. The cost improvements may not seem large in proportion to the total scheduling cost, but note that in the PDP literature, the percentage improvement is often under 2% (Polacek et al. 2004) or only a binary result of whether the best known solution was improved upon is given (Ropke and Pisinger 2006; Hasle and Kloster 2007), and improvements of this order of magnitude are significant. VLNS-T finds better solutions, but this comes at a cost of computation time. We have done little optimization of our implementations of VLNS and VLNS-T as our focus is on determining the effectiveness of transfers, but heuristics to exclude transfer points from the search are a promising area for future work. The number of transfers per item is low, i.e., Problem 6 has 32 transfers for 144 items, which is 0.22 transfers per item. An improvement in 19 out of 20 problems suggests that many PDP problems may benefit from allowing transfers. The benefit of transfers may be greater if we optimize for criteria other than distance, such as delivery time, which these benchmarks do not address.

New York Taxi Problems

We constructed a set of problems sampled from taxi routes in New York, using over 400,000 data points for Tuesday, 9/25/2012 (Ferreira et al. 2013). The data includes taxi ride start and end times, and GPS coordinates of the start and end. First, we cluster the points to find the most popular pickup and dropoff locations, and select the 20 most popular points in Manhattan and the 80 most popular points elsewhere. We sample from taxi routes which start and end near these points which occur within the same hour window. Distances and travel times are estimated using OpenStreetMap with OSRM (Luxen and Vetter 2011). Vehicles have capacity 4.

In the first set of problems, all items are given the same hour long time window. We varied $|V|$ from 20 to 40 and $|M|$ from 25 to 100, solving 20 instances for each parameterization. With $|V| = 40$, $|M| = 100$, the average distance improvement from VLNS-T was 7.37%, and the maximum was 12%. The average improvement was between 4.72% and 7.59% for all values. For the second set of problems, we began the item time windows at the pickup times in the original data, and set the delivery deadline as the actual delivery time plus the estimated travel time. For $|V| = 30$ and $|M| = 60$, the average improvement was 6.78%, and the maximum was 10.4%. For all selected $|V|$ and $|M|$, the average improvement of VLNS-T over VLNS varied between 4.27% and 6.82%. These results indicate that transfers can improve efficiency in real-world transportation domains, for example alternative shared taxi services or ridesharing.

Conclusion

We have introduced the VLNS-T algorithm to solve the PDP-T. VLNS-T searches over schedules by removing and reinserting a randomized set of items. We introduced two heuristics to insert items into a schedule with transfers, and an algorithm to compute the execution times for actions in such a schedule. We demonstrated that VLNS-T improves upon state-of-the-art solutions for a set of PDP benchmark problems by using transfers, and showed the effectiveness of VLNS-T for transportation problems generated from real-world taxi data in New York City.

Acknowledgements

We thank Huy Vo and Claudio Silva, from the Center for Urban Science and Progress (CUSP) of New York University, for providing the New York City taxi data used in this work to us. Manuela Veloso spent the academic year of 2013/2014 on sabbatical at CUSP and truly appreciates their help.

This research was partially sponsored by the Office of Naval Research under grant number N00014-09-1-1031, and by the National Science Foundation under award number NSF IIS-1012733. The views and conclusions expressed are those of the authors only.

References

Bouros, P.; Sacharidis, D.; Dalamagas, T.; and Sellis, T. 2011. Dynamic pickup and delivery with transfers. *Advances in Spatial and Temporal Databases* 6849:112–129.

Bryan, D., and O'Kelly, M. 2002. Hub-and-spoke networks in air transportation: An analytical review. *Journal of Regional Science* 39(2):275–295.

Coltin, B., and Veloso, M. 2012. Optimizing for transfers in a multi-vehicle collection and delivery problem. In *Proc. of Distributed Autonomous Robotic Systems (DARS)*.

Coltin, B., and Veloso, M. 2013. Towards ridesharing with passenger transfers. In *Proc. of the Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1299–1300.

Coltin, B., and Veloso, M. 2014. Online pickup and delivery planning with transfers for mobile robots. In *To Appear, Proc. of the Int. Conf. on Robotics and Automation (ICRA)*.

Cordeau, J.-F., and Laporte, G. 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological* 37(6):579–594.

Cortés, C.; Matamala, M.; and Contardo, C. 2010. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research* 200(3):711–724.

Ferreira, N.; Poco, J.; Vo, H. T.; Freire, J.; and Silva, C. T. 2013. Visual exploration of big spatio-temporal urban data: A study of New York City taxi trips. *IEEE Transactions on Visualization and Computer Graphics* 19(12):2149–2158.

Gørtz, I.; Nagarajan, V.; and Ravi, R. 2009. Minimum makespan multi-vehicle dial-a-ride. *Algorithms-ESA 2009* 540–552.

Hasle, G., and Kloster, O. 2007. Industrial vehicle routing. *Geometric Modelling, Numerical Simulation, and Optimization* 397–435.

Jung, S., and Haghani, A. 2000. Genetic algorithm for a pickup and delivery problem with time windows. *Transportation Research Record: Journal of the Transportation Research Board* 1733(-1):1–7.

Li, H., and Lim, A. 2001. A metaheuristic for the pickup and delivery problem with time windows. In *Proc. of the Int. Conf. on Tools with Artificial Intelligence*, 160–167. IEEE.

Lu, Q., and Dessouky, M. 2004. An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science* 38(4):503–514.

Lu, Q., and Dessouky, M. 2006. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research* 175(2):672–687.

Luxen, D., and Vetter, C. 2011. Real-time routing with OpenStreetMap data. In *Proc. of the Int. Conf. on Advances in Geographic Information Systems*, 513–516.

Masson, R.; Lehuédé, F.; and Péton, O. 2013a. An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science* 47(3):344–355.

Masson, R.; Lehuédé, F.; and Péton, O. 2013b. Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters* 41(3):211–215.

Masson, R.; Lehuédé, F.; and Péton, O. 2014. The dial-a-ride problem with transfers. *Computers & Operations Research* 41:12–23.

Mitrovic-Minic, S., and Laporte, G. 2006. The pickup and delivery problem with time windows and transshipment. *Information Systems and Operational Research* 44(3):217–228.

Nakao, Y., and Nagamochi, H. 2008. Worst case analysis for a pickup and delivery problem with single transfer. *Numerical Optimization methods, theory and applications* 1584:142–148.

Nanry, W., and Wesley Barnes, J. 2000. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological* 34(2):107–121.

Polacek, M.; Hartl, R. F.; Doerner, K.; and Reimann, M. 2004. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics* 10(6):613–627.

Ropke, S., and Pisinger, D. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40(4):455–472.

Ropke, S.; Cordeau, J.; and Laporte, G. 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks* 49(4):258–272.

Shang, J., and Cuff, C. 1996. Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering* 30(4):631–645.

Thangiah, S.; Fergany, A.; and Awan, S. 2007. Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research* 15(4):329–349.

Xu, H.; Chen, Z.; Rajagopal, S.; and Arunapuram, S. 2003. Solving a practical pickup and delivery problem. *Transportation Science* 37(3):347–364.