CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

The Pickup and Delivery Traveling Salesman Problem with First-In-First-Out Loading

Güneş Erdoğan
Jean-François Cordeau
Gilbert Laporte

December 2007

CIRRELT-2007-61

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

The Pickup and Delivery Traveling Salesman Problem with First-In-First-Out Loading

Güneş Erdoğan[1,2,3], Jean-François Cordeau[1,2,*], Gilbert Laporte[1,3]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

[3] Canada Research Chair in Distribution Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. This paper addresses a variation of the Traveling Salesman Problem with Pickup and Delivery in which loading and unloading operations have to be executed in a first-in-first-out fashion. It provides an integer programming formulation of the problem. It also describes five operators for improving a feasible solution, and two heuristics that utilize these operators: a probabilistic tabu search algorithm, and an iterated local search algorithm. The heuristics are evaluated on data adapted from TSPLIB instances.

Keywords. Traveling salesman problem, pickup and delivery, first-in-first-out, integer programming, tabu search, iterated local search.

Acknowledgements. This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 227837-04 and 39682-05. This support is gratefully acknowledged.

---

* Corresponding author: jean-francois.cordeau@hec.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2007

1 Introduction

The purpose of this paper is to introduce a new variant of the *Traveling Salesman Problem with Pickup and Delivery* (TSPPD) called the *TSPPD with First-In-First-Out Loading* (TSPPDF). The TSPPD consists of determining a minimum cost circuit traveled by a vehicle to service n requests. Each request is defined by an origin vertex, the pickup location, and a destination vertex, the delivery location. The vehicle starts from the depot and returns to it after all requests have been serviced. Every vertex except the depot must be visited exactly once, subject to the constraint that the pickup vertex of a request must be visited before its corresponding delivery vertex. The TSPPD has several practical applications in freight and passenger transportation. It arises, for example, in urban courier service operations, in less-than-truckload transportation, and in door-to-door transportation services for the elderly and the disabled (Cordeau et al., 2007b).

The TSPPDF is similar to the TSPPD, with the difference that the pickup and delivery operations must be executed in a first-in-first-out (FIFO) fashion. In other words, if the vehicle picks up request i before request j, then it cannot deliver request j before delivering request i. The TSPPDF has applications in the same areas as the TSPPD when the circumstances dictate the requests to be processed in a FIFO fashion. One such case occurs in dial-a-ride systems when fairness is a major concern, i.e. when the passengers resent another passenger being picked up after them but dropped off before them. Other potential industrial applications may arise in the management of automatic guided vehicles that load items on one end and unload them at the other end.

The TSPPD has been studied, among others, by Kalantari, Hill and Arora (1985), Fischetti and Toth (1989), Savelsbergh (1990), Healy and Moll (1995), Ruland and Rodin (1997), Renaud, Boctor and Ouenniche (2000), Renaud, Boctor and Laporte (2002), and Dumitrescu et al. (2007). We refer the reader to Berbeglia et al. (2007), Cordeau et al. (2007b), and Cordeau, Laporte and Ropke (2008) for recent surveys. The largest non-trivial TSPPD instance solved to optimality by Dumitrescu et al. (2007) involves 30 requests and required about two hours of computing time. To the best of our knowledge, the most effective heuristic is the solution perturbation approach developed by Renaud, Boctor and Laporte (2002).

An interesting variant of TSPPD, which is similar to the TSPPDF, is the *TSPPD with Last-In-First-Out Loading* (TSPPDL) in which pickup and delivery operations are executed in a last-in-first-out (LIFO) fashion. The TSPPDL and its variants have been studied by Ladany and Mehrez (1984), Pacheco (1994, 1995), Levitin

and Abezgaouzb (2003), Cassani (2004), Ficarelli (2005), Carrabs, Cerulli and Cordeau (2007), Carrabs, Cordeau and Laporte (2007), and Cordeau et al. (2007a). To the best of our knowledge, the best exact algorithm is that of Cordeau et al. (2007a) which solves instances of up to 25 requests within an hour of computing time. The most successful heuristic seems to be the variable neighborhood search algorithm of Carrabs, Cordeau and Laporte (2007) which uses new local search operators with complexities not exceeding $O(n^3)$.

Our aim is to develop and compare two heuristics for the TSPPDF: a probabilistic tabu search algorithm, and an iterated local search algorithm. The remainder of the paper is organized as follows. In Section 2 we present an NP-hardness proof for the problem, as well as an integer programming formulation. In Section 3, we develop five operators which are embedded within a probabilistic tabu search heuristic and an iterated local search heuristic, both described in Section 4. Computational results are presented in Section 5, followed by our conclusions in Section 6.

2 Problem description

The TSPPDF is defined on a weighted complete directed graph $G = (V, A)$, where V is the vertex set and A is the arc set. Every arc $(i, j) \in A$ has a cost c_{ij}. Let $K = \{1, ..., n\}$ be the set of transportation requests. A request $k \in K$ is composed of a pickup vertex k^+ and of a delivery vertex k^-. Let $P = \{1^+, ..., n^+\}$ be the set of pickup vertices and $D = \{1^-, ..., n^-\}$ the set of delivery vertices, and denote the depot by 0. Then $V = P \cup D \cup \{0\}$.

Proposition 1: TSPPDF is NP-Hard.

Proof: We proceed by a reduction from the *Asymmetric Traveling Salesman Problem* (ATSP). Consider an arbitrary instance of the ATSP on G. Construct a graph $G' = (V', A')$, where V' consists of the depot v_0 and two vertices v_i^+ and v_i^- for every vertex v_i of $V \setminus \{v_0\}$. For every arc (v_i, v_j) in A, define an arc (v_i^-, v_j^+) in A' with cost c_{ij}. Also define an arc (v_i^+, v_i^-) of cost 0 for every vertex pair v_i^+ and v_i^- in $V' \setminus \{v_0\}$. Observe that by construction, one must use the arc (v_i^+, v_i^-) to be able to visit v_i^-. Clearly, the ATSP instance has a solution of cost not exceeding z^* if and only if the TSPPDF instance has an objective function value not exceeding z^*. □

We now provide an integer programming formulation for the TSPPDF. It is similar to the flow based model of Wong (1980) for the ATSP, which uses $2n - 2$ commodities to ensure the connectivity of a selected vertex to the remaining vertices.

In our case, however, the connectivity of corresponding pick-up and delivery vertices must be satisfied, in addition to their individual connectivity to the depot. For this purpose we use a flow based model with $3n$ commodities.

Let $V' = V \setminus \{0\}$ and let A' be the subset of arcs A with both endpoints in V'. Let x_{ij} be a binary variable equal to one if and only if arc $(i,j) \in A$ is a part of the circuit. Furthermore, let $y^1_{ijk}, y^2_{ijk}, y^3_{ijk}$ be binary flow variables associated with arc (i,j) for the path used by request $k \in K$, from the depot to k^+, from k^+ to k^-, and from k^- to the depot, respectively. These variables are equal to 1 if and only if the respective path uses arc (i,j). The model is then:

(TSPPDF)

$$\text{Minimize} \sum_{(v_i,v_j)\in A} c_{ij}x_{ij} \tag{1}$$

subject to

$$\sum_{v_j\in V} x_{ij} = 1 \quad (v_i \in V) \tag{2}$$

$$\sum_{v_i\in V} x_{ij} = 1 \quad (v_j \in V) \tag{3}$$

$$\sum_{v_j:(v_i,v_j)\in A} y^1_{ijk} - \sum_{v_j:(v_j,v_i)\in A} y^1_{jik} = \begin{cases} 1 & \text{if } v_i = v_0 \\ -1 & \text{if } v_i = k^+ \\ 0 & \text{otherwise} \end{cases} \quad (v_i \in V, k \in K) \tag{4}$$

$$\sum_{v_j:(v_i,v_j)\in A} y^2_{ijk} - \sum_{v_j:(v_j,v_i)\in A} y^2_{jik} = \begin{cases} 1 & \text{if } v_i = k^+ \\ -1 & \text{if } v_i = k^- \\ 0 & \text{otherwise} \end{cases} \quad (v_i \in V', k \in K) \tag{5}$$

$$\sum_{v_j:(v_i,v_j)\in A} y^3_{ijk} - \sum_{v_j:(v_j,v_i)\in A} y^3_{jik} = \begin{cases} 1 & \text{if } v_i = k^- \\ -1 & \text{if } v_i = v_0 \\ 0 & \text{otherwise} \end{cases} \quad (v_i \in V, k \in K) \tag{6}$$

$$\sum_{v_i:(v_i,l^+)\in A'} y^2_{il^+k} + \sum_{v_i:(v_i,l^-)\in A'} y^2_{il^-k} \leq 1 \quad (k,l \in K : k \neq l) \tag{7}$$

$$y^1_{ijk} + y^3_{ijk} = x_{ij} \quad ((v_i, v_j) \in A \setminus A', k \in K) \tag{8}$$
$$y^1_{ijk} + y^2_{ijk} + y^3_{ijk} = x_{ij} \quad ((v_i, v_j) \in A', k \in K) \tag{9}$$
$$x_{ij} = 0 \text{ or } 1 \quad ((v_i, v_j) \in A) \tag{10}$$
$$y^1_{ijk} = 0 \text{ or } 1 \quad ((v_i, v_j) \in A, k \in K) \tag{11}$$
$$y^2_{ijk} = 0 \text{ or } 1 \quad ((v_i, v_j) \in A', k \in K) \tag{12}$$
$$y^3_{ijk} = 0 \text{ or } 1 \quad ((v_i, v_j) \in A, k \in K). \tag{13}$$

Constraints (4), (5), and (6) state that the solution must contain a path from the depot to the pickup vertex of request k, a path from the pickup vertex of request k to the delivery vertex of request k that does not pass through the depot, and a path from the delivery vertex of request k to the depot. Constraints (7) state that another request l may either be picked up or delivered on the path from k^+ to k^-, but not both. Constraints (8) and (9) bind the overall routing decisions with the individual routing decisions of every request.

Note that removing the constraint set (7) results in a formulation that is valid for the TSPPD. Furthermore, replacing it with:

$$\sum_{v_i:(v_i,l^+)\in A'} y^2_{il^+k} = \sum_{v_i:(v_i,l^-)\in A'} y^2_{il^-k} \quad (k, l \in K : k \neq l) \tag{14}$$

would imply that if request l is picked up while transporting request k, then it must be delivered before delivering request k. This change would result in a valid formulation for the TSPPDL. The above formulation can be strengthened with the following valid inequalities:

$$\sum_{v_i:(v_i,l^-)\in A} y^1_{il^-k} \leq \sum_{v_i:(v_i,l^+)\in A} y^1_{il^+k} \quad (k, l \in K : k \neq l) \tag{15}$$

$$\sum_{v_i:(v_i,l^+)\in A} y^3_{il^+k} \leq \sum_{v_i:(v_i,l^-)\in A} y^3_{il^-k} \quad (k, l \in K : k \neq l). \tag{16}$$

Inequalities (15) imply that if the delivery vertex of a request l is visited on the path from the depot to the source of request k, then its pickup vertex must also be visited on that path. Likewise, the valid inequalities (16) imply that if the pickup vertex of a request l is visited on the path from the destination of request k to the depot, then its delivery vertex must also be visited on that path.

3 Local search operators

We now describe four local search operators used to improve a given feasible TSP-PDF solution. We first provide the formal definitions associated with the structure of such solutions.

3.1 Definitions

A *couple* is the pair of pickup and delivery vertices (k^+, k^-) of a request $k \in K$. To simplify notation, we also write 0^+ and 0^- to denote 0. We define a *component* to be a path from a pickup vertex to a delivery vertex, at the beginning and end of which there are no requests being transported by the vehicle. See Figure 1(a) for an example of a couple and Figure 1(b) and 1(c) for examples of components. Clearly, the operations of exchanging two couples and exchanging two components preserve feasibility. The order in which a vertex v is visited in a circuit is called its *position* and is denoted by $\pi(v)$, with $\pi(0) = 0$. We define the order in which the requests are picked up as the *service sequence*, and we write $\sigma(i)$ to denote the i^{th} request in the service sequence, with $\sigma(0) = 0$. If $\pi(1^+) = p$, and $\sigma(q) = 1$ in the example of Figure 1, then $\pi(1^-) = p + 1$, $\pi(2^+) = p + 2$, $\pi(3^+) = p + 3$, $\pi(2^-) = p + 4$, $\pi(3^-) = p + 5$, $\sigma(q + 1) = 2$, and $\sigma(q + 2) = 3$.

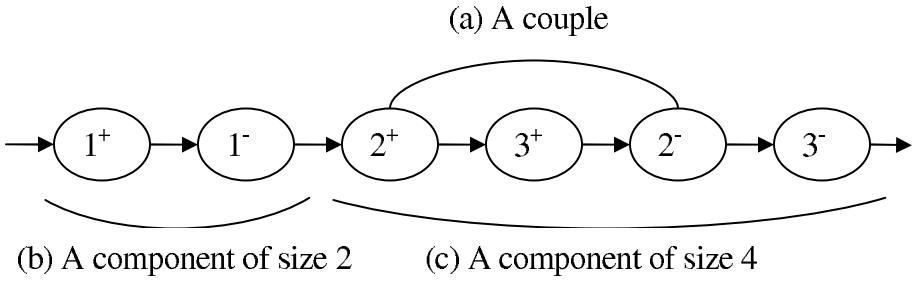

Figure 1: Example of a couple and components

3.2 Description of the operators

We now define the operators used within our heuristics.

3.2.1 The *couple-exchange* operator

The couple-exchange operator exchanges two couples. This operator can be implemented to run in $O(n^2)$ time using appropriate data structures.

3.2.2 The *component-exchange* operator

The component-exchange operator exchanges two components. There can be at most n components in a given circuit, each of which can be identified in $O(n)$ time. Evaluating the exchange of the components in a circuit therefore takes $O(n^2)$ time

3.2.3 The *relocate-couple* operator

The *relocate-couple* operator determines, for each couple (k^+, k^-), its best possible relocation in the circuit. The complexity of this operator is bounded by $O(n^3)$, considering that for each of the n couples, there are $O(n)$ insertion points for k^+, and $O(n)$ feasible insertion points for k^- consistent with the FIFO rule. We will perform a detailed analysis of the operator and show that it can be implemented to run in $O(n^2)$ time.

Proposition 2: The complexity of the *relocate-couple* operator is $O(n^2)$.

Proof: Assume that we are relocating the couple (k^+, k^-). We can construct a temporary circuit by removing both vertices from the circuit in two operations. On this temporary circuit composed of the pickup and delivery vertices of $n-1$ requests, we first look for a position to insert k^+, and then a corresponding feasible position to insert k^-. Note that for this temporary circuit there exists a corresponding temporary service sequence, with $\pi(\sigma(1)^+) = 1$ and $\pi(\sigma(n-1)^-) = 2n-2$. We will analyze this insertion for three cases:

1) Vertex k^+ is inserted between the depot and the pickup vertex of the first request in the service sequence. In this case, possible insertion points for k^- start from just after k^+ and end just before the delivery vertex of the first request, the number of such positions being $\pi(\sigma(1)^-) - \pi(\sigma(1)^+) + 1 = \pi(\sigma(1)^-)$.

2) Vertex k^+ is inserted between the pickup vertex of the last request in the service sequence and the depot. In this case, k^- can only be inserted between the delivery vertex of the last request and the depot. Hence the number of possible insertion points for k^+ is bounded above by $\pi(\sigma(n-1)^-) - \pi(\sigma(n-1)^+) + 1 = 2n - 1 - \pi(\sigma(n-1)^+)$.

3) Vertex k^+ is inserted in a position after the pickup vertex of the i^{th} request and before the pickup vertex of the $i+1^{\text{st}}$ request in the service sequence, for $i = 1$ to $n-1$. In this case, the possible insertion points start from just after the delivery vertex of the i^{th} request, and end just before the delivery vertex of the $i+1^{\text{st}}$ request in the service sequence.

We further break down case 3) into two subcases:

3i) The index of the pickup vertex for the $i+1^{\text{st}}$ request in the service sequence is less than the index of the delivery vertex of the i^{th} request. In this case, the possible insertion points of k^+ and k^- do not interfere with each other and the best combination can be found by identifying the best insertion points separately, which takes $\pi(\sigma(i+1)^+) - \pi(\sigma(i)^+) + \pi(\sigma(i+1)^-) - \pi(\sigma(i)^-) \leq \pi(\sigma(i+1)^-) - \pi(\sigma(i)^+)$ operations (see Figure 2a).

3ii) The index of the pickup vertex for the $i+1^{\text{st}}$ request in the service sequence is greater than the index of the delivery vertex of the i^{th} request. Then the vertices $\sigma(i)^-$ and $\sigma(i+1)^+$ must be visited consecutively (see Figure 2(b)). If the insertion position for k^+ is before $\sigma(i)^-$, the decisions do not interfere and it takes $\pi(\sigma(i)^-) - \pi(\sigma(i)^+) + \pi(\sigma(i+1)^-) - \pi(\sigma(i+1)^+) + 1$ operations to find the minimum cost insertion points separately. Within this operation, we also identify the insertion point for k^- between $\sigma(i+1)^+$ and $\sigma(i+1)^-$ with minimum possible cost. If the insertion position for k^+ is just after $\sigma(i)^-$, one can find the best possible insertion point for k^- by evaluating the possibilities of inserting both k^+ and k^- between $\sigma(i)^-$ and $\sigma(i+1)^+$, and inserting k^+ between $\sigma(i)^-$ and $\sigma(i+1)^+$ and k^- at the point with minimum possible cost between $\sigma(i+1)^+$ and $\sigma(i+1)^-$ (which has already been identified in the previous step), in two operations. This brings the sum to $\pi(\sigma(i)^-) - \pi(\sigma(i)^+) + \pi(\sigma(i+1)^-) - \pi(\sigma(i+1)^+) + 3 = \pi(\sigma(i+1)^-) - \pi(\sigma(i)^+) + 2$ operations.

In both subcases, the number of operations is bounded above by $\pi(\sigma(i+1)^-) - \pi(\sigma(i)^+) + 2$. Summing up this term for $i = 1$ to $n-2$ yields $\pi(\sigma(n-1)^-) - \pi(\sigma(1)^+) + 2n - 4 = 4n - 7$.

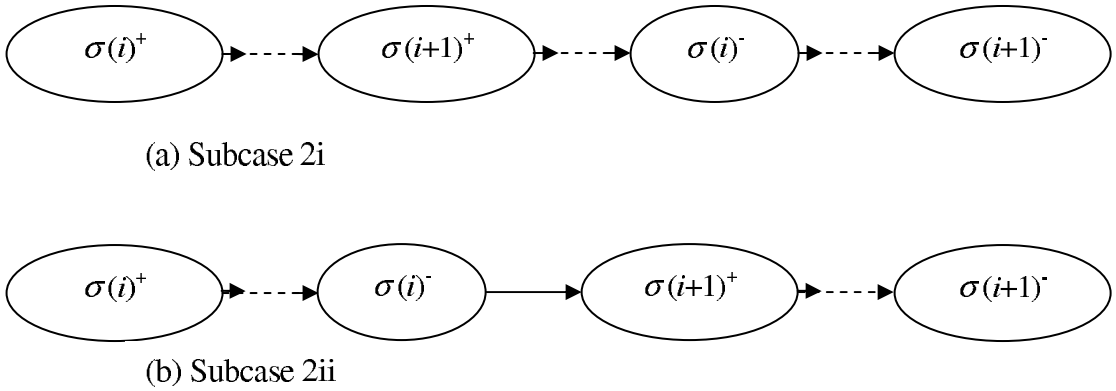

Figure 2: Two subcases of the second insertion case

Summing up the number of operations for the construction of the temporary circuit and cases 1, 2, and 3, we obtain $6n - 6 + \pi(\sigma(1)^-) - \pi(\sigma(n-1)^+)$. Since $\pi(\sigma(1)^-) - \pi(\sigma(n-1)^+) \leq 1$, the total number of operations is bounded above by $6n-5 = O(n)$. When applied to every possible (k^+, k^-) pair, the overall complexity of the operator becomes $O(n^2)$. □

3.2.4 The *relocate-component* operator

The relocate-component operator determines, for each component in a given circuit, its best possible relocation. Since the maximum number of components is n, the complexity of this operator is $O(n^2)$.

3.2.5 The *reroute-with-fixed-service-sequence* operator

Assume that the service sequence of the requests is fixed, and note that this also fixes the order in which the requests are delivered. We now prove that the number of feasible solutions is exponential for this tightly restricted case. For any fixed service sequence, let $t(i, j)$ be number of possible ways of serving the requests given that there remain i pickup requests and j delivery requests to be performed (see Figure 3). With this definition, $t(n, n)$ is the total number of feasible solutions when the service sequence is fixed, and $n!t(n, n)$ is the total number of feasible solutions for a TSPPDF instance involving n requests.

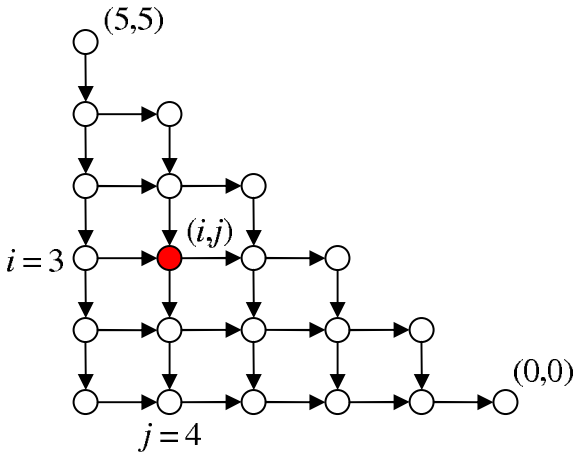

Figure 3: $t(i, j)$ is the number of ways of going from (i, j) to $(0, 0)$ by moving down (making a pickup) or right (making a delivery)

Proposition 3: $t(i, i) = t(i - 1, i)$, for $i > 0$.

Proof: Since the number of delivery requests is equal to the number of pickup requests, the only possible move from (i, i) is to $(i - 1, i)$. □

Proposition 4: Given $0 < i < j \leq n, t(i, j) = t(i - 1, j) + t(i, j - 1)$.

Proof: If there are still requests to be picked up and delivered at (i, j), the next move can be either to $(i - 1, j)$ (a pickup) or to $(i, j - 1)$ (a delivery). □

Proposition 5: Given $0 \leq j \leq n, t(0, j) = 1$.

Proof: If all requests of the service sequence have been picked up, the only possible move from $(0, j)$ is to $(0, j - 1)$, then to $(0, j - 1)$, until $(0, 0)$ is reached. □

Proposition 6: $t(n, n) > 2t(n - 1, n - 1)$ for $n \geq 3$.

Proof: $t(n, n) = t(n - 1, n) = t(n - 2, n) + t(n - 1, n - 1) = t(n - 3, n) + t(n - 2, n - 1) + t(n - 1, n - 1) = t(n - 3, n) + 2t(n - 1, n - 1) > 2t(n - 1, n - 1)$. The four equalities follow from Propositions 3, 4, 4, and 3, respectively. The inequality follows from Proposition 5, which implies that $t(n - 3, n) > 0$. □

Proposition 7: $t(n,n) \in \Omega(2^n)$ for $n \geq 3$.

Proof: $t(2,2) = t(1,2) = t(0,2) + t(1,1) = t(0,2) + t(0,1) = 2$ and $t(n,n) > 2t(n-1,n-1) > 4t(n-2,n-2) > 2^k t(n-k,n-k) > 2^{n-2}t(2,2) = 2^{n-1}$. For any constant $c \geq 2, c \times t(n,n) > 2^n$. $\square$

We now provide a dynamic programming formulation to identify the best of this exponential number of solutions in polynomial time. Define $f(i,j,k)$ as the minimum cost of serving all remaining requests after having picked up the i^{th} request in the service sequence and delivered the j^{th} request, where $j \leq i$, and the last operation is a pickup if $k = 0$ and a delivery if $k = 1$. The formulation is as follows:

$$
\begin{aligned}
f(i,j,0) &= \min \{c_{\sigma(i)^+,\sigma(i+1)^+} + f(i+1,j,0), \\
&\quad c_{\sigma(i)^+,\sigma(j+1)^-} + f(i,j+1,1))\} \quad (i,j : 0 \leq j < i < n) && (17)\\
f(i,j,1) &= \min \{c_{\sigma(j)^-,\sigma(i+1)^+} + f(i+1,j,0), \\
&\quad c_{\sigma(j)^-,\sigma(j+1)^-} + f(i,j+1,1))\} \quad (i,j : 0 \leq j < i < n) && (18)\\
f(i,i,1) &= c_{\sigma(i)^-,\sigma(i+1)^+} + f(i+1,i,0) \quad (i : 0 \leq i < n) && (19)\\
f(n,j,0) &= c_{\sigma(n)^+,\sigma(j+1)^-} + f(n,j+1,1) \quad (j : 0 \leq i < n) && (20)\\
f(n,j,1) &= c_{\sigma(j)^-,\sigma(j+1)^-} + f(n,j+1,1) \quad (j : 0 \leq i < n) && (21)\\
f(n,n,1) &= c_{\sigma(n)^-,0}. && (22)
\end{aligned}
$$

The dynamic programming formulation involves $n(n+1)$ states, each of which has a constant processing time. Consequently, the overall complexity of the *reroute-with-fixed-service-sequence* operator is $O(n^2)$.

4 Heuristics

We have developed two constructive heuristics and two improvement heuristics for the TSPPDF whose descriptions follow.

4.1 Constructive heuristics

The first constructive heuristic, called GENIUS start, is inspired by the constructive heuristic "H" of Gendreau, Laporte, and Vigo (1999) for a one-to-many-to-one variant of the TSPPD. It uses a *Traveling Salesman Problem* (TSP) solution to obtain a

TSPPD solution. The idea is to construct a minimum cost circuit on the vertices of the instance at hand, and to solve the instance on a reduced graph consisting only of the arcs in the circuit. This algorithm has a worst case performance ratio of 3 for the variant of the TSPPD mentioned above. To construct a minimal cost circuit, we use the GENIUS algorithm of Gendreau, Hertz, and Laporte (1992). This algorithm is based on a "generalized insertion" procedure called GENI, followed by a post-optimization "unstringing and stringing" procedure called US. Both procedures use a local search mechanism by which a vertex can only be inserted next to one of its p closest neighbors, where p is an input parameter. The second constructive heuristic, called FIFO nearest neighbor, is an adaptation of the well-known nearest neighbor heuristic for the TSP (Rosenkrantz, Stearns and Lewis, 1977). We will use these two heuristics as a starting point for the other two heuristics.

GENIUS start

Step 1 (Initialization). Compute a shortest Hamiltonian circuit on the vertices using GENIUS.

Step 2 (Service sequence extraction). Determine the service sequence in which the pickup vertices are visited.

Step 3 (Polishing). Apply the *reroute-with-fixed-service-sequence* operator to the service sequence.

FIFO nearest neighbor

Step 1 (Initialization). Set the depot to be the last visited vertex. Fill the pickup list with the pickup vertices of all the requests. Create an empty delivery queue.

Step 2 (Vertex selection). If there are no vertices in the pickup list or delivery queue, go to step 4. Else, select the closest vertex to the last visited vertex among the remaining requests in the pickup list and the first element in the delivery queue.

Step 3 (Vertex insertion). If the selected vertex is from the pickup list, remove it from the pickup queue, add it to the circuit and, add the delivery vertex of the associated request to the end of the delivery queue. Else, remove the vertex from the delivery queue and add it to the circuit. Go to Step 2.

Step 4 (Polishing). Apply the *reroute-with-fixed-service-sequence* operator to the final service sequence.

The complexity of the FIFO nearest neighbor heuristic is $O(n^2)$ and that of the GENIUS algorithm is $O(np^4 + n^2)$.

4.2 Improvement heuristics

Our two improvement heuristics are a probabilistic tabu search algorithm and an iterated local search algorithm. For the improvement heuristics, we have applied the first four operators to explore the solution space, and we have used the *reroute-with-fixed-service-sequence* operator as a post-optimizer to find the best possible routing for the new service sequence.

Probabilistic tabu search

This algorithm works with three parameters: κ is the number of iterations since the last update of the best solution value, μ is the maximum number of iterations without updating the best solution, and λ is a parameter used for random sampling.

Step 1 (Initialization). Construct a feasible circuit by using the FIFO nearest neighbor heuristic. Record the circuit as the best known solution. Set $\kappa = 1$.

Step 2 (Termination criterion). If $\kappa < \mu$, go to Step 3. Else, stop.

Step 3 (Local search). If $\kappa \leq \mu/2$, set $\lambda = \max\{0.05, \kappa/\mu\}$. Else, set $\lambda = \max\{0.05, (\mu - \kappa)/\mu\}$. Evaluate all possible moves for the *couple-exchange*, *component-exchange*, *relocate-couple*, and *relocate-component* operators. Record λ percent of the moves and also the best possible move. If the best possible move results in a better solution than the best known solution, apply it. Else, select the best recorded move and apply it. Apply the *reroute-with-fixed-service-sequence* operator to the resulting circuit to find the final circuit. If the final circuit has a lower cost than the best known solution, update the best known solution, and set $\kappa = 1$. Else, increment κ by 1. Go to Step 2.

Iterated local search

This algorithm uses the parameters κ and μ already defined, and a third parameter θ, which is a measure of perturbation of the solution at hand.

Step 1 (Initialization). Construct a feasible circuit by using the FIFO nearest neighbor heuristic. Record the circuit as the best known solution. Set $\kappa = 1$.

Step 2 (Perturbation). If $\kappa < \mu$, construct a perturbed temporary circuit by swapping θ randomly selected requests in the best known solution. Else, stop.

Step 3 (Local search). Apply the best improving move from the *couple-exchange*, *component-exchange*, *relocate-couple*, and *relocate-component* operators, improved by the *reroute-with-fixed-service-sequence* operator to the temporary circuit, until

no further improvement is possible. If the new circuit has a lower cost, update the best known solution, and set $\kappa = 1$. Else, increment κ by 1. Go to Step 2.

All five operators we have presented have a complexity of $O(n^2)$, and therefore the complexity per iteration of each of our two improvement heuristics is $O(n^2)$.

5 Computational Results

We have coded the heuristics just described in C++, and we have conducted computational experiments on a workstation with a 64-bit CPU running at 2.4GHz, and 16 GB of RAM. We have used the same data as in Carrabs, Cordeau, and Laporte (2007) for the TSPPDL, which consist of 72 instances adapted from 2D Euclidian instances of TSPLIB (Reinelt, 1991), with vertices matched to yield couples and distances rounded to the closest integer. We have run the improvement algorithms for both choices of starting constructive heuristics. Since the algorithms involve random components, we have performed five replications for each instance, resulting in 1440 runs in total. For the GENIUS algorithm we have used $p = 3$, for the probabilistic tabu search algorithm we have used $\mu = 10000$, and for the iterated search algorithm we have used $\mu = 250$ and $\theta = \min\{5, \lceil n/10 \rceil + 1\}$. The results for the probabilistic tabu search algorithm are presented in Table 1, while those for the iterated local search algorithm are presented in Table 2. Finally, a comparison of all four combinations of algorithms is presented in Table 3.

The column headings are defined as follows:

Instance	**:**	The TSPLIB data file from which the TSPPDF instance is adapted.
Initial Obj.	**:**	Cost of the solution found by the constructive heuristic.
Best Obj.	**:**	Cost of the best solution found by an algorithm for the instance.
Avg. Seconds	**:**	Average CPU time required by the algorithm for the instance.
Avg. Dev.	**:**	Average deviation of the solutions found at each replication of an algorithm from the best known solution for the instance.
Opt. Count	**:**	Number of times an algorithm successfully finds the best known solution for the instance.

We have also implemented the integer programming formulation presented in Section 2 using CPLEX 10.1.1, and run it with a time limit of four hours. To test

the computational reach of the formulation, we have constructed instances with $|V| = 15, 21, 31, 37$, and 45. The computational reach ends at $|V| = 25$, until which we could prove the optimality of certain best known solutions found by our algorithms. For $|V| = 15$, we were able to prove the optimality of the best known solution for all instances. For $|V| > 15$, we experienced increasing optimality gaps, with an average of 1.5% for $|V| = 21$, 5% for $|V| = 25$, and 20% or more for larger instances. The linear programming relaxations of the instances withe $|V| \geq 75$ could not be solved within the time limit.

The objective function values of the instances for which an algorithm found the best known solution have been emphasized in boldface in Tables 1 and 2. The maximum of all the runtimes is 12,048.90 seconds, while the maximum average runtime is 7379.93 seconds. The average runtimes of the iterated local search algorithm are the lowest for $|V| \leq 101$. However, for $|V| \geq 251$ the runtimes of the probabilistic tabu search become smaller. All four algorithms usually find the same best solution for instances for $|V| \leq 45$. For the 24 larger instances with $|V| = 101, 251, 501$, and 751, the best solution is found 15 times by one of the probabilistic tabu search algorithms, eight times by one of the iterated local search algorithms eight times, and once by both types of heuristics. This indicates that the probabilistic tabu search algorithm is superior. The average deviations for the probabilistic tabu search algorithms are also lower than their iterated local search counterparts, pointing to the same conclusion. There is no clear winner among the two starting constructive algorithms, both in terms of the initial and final objective values.

6 Conclusions

We have defined and analyzed a variant of the TSPPD in which the pickup and delivery operations must be carried out in a FIFO fashion. We have presented an integer programming formulation, a proof of NP-hardness, five operators, and four heuristics. One of our operators is based on a dynamic programming formulation for a restricted case of the TSPPDF, which still has an exponential number of solutions. The complexity per iteration of our improvement algorithms does not exceed $O(n^2)$. We have applied the algorithms on data adapted from the TSPLIB, and presented extensive computational results. The overall best algorithm consists of applying GENIUS start, followed by the probabilistic tabu seach algorithm.

Acknowledgments: This work was partially funded by the Canadian Natural Sciences and Engineering Research Council under grants 227837-04 and 39682-05. This support is gratefully acknowledged.

Bibliography

[1] Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., Laporte, G., 2007. Static Pickup and Delivery Problems: A Classification Scheme and Survey, TOP 15, 1–31.

[2] Carrabs, F., Cerulli, R., Cordeau, J.-F., 2007. An Additive Branch-and-Bound Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading, Technical report CIRRELT–2007–12.

[3] Carrabs, F., Cordeau, J.-F., Laporte, G., 2007. Variable Neighborhood Search for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading, INFORMS Journal on Computing 19, 618–632.

[4] Cassani, L., 2004. Algoritmi euristici per il TSP with rear-loading. Degree thesis, Universit di Milano, Italy, http://www.crema.unimi.it/righini/Papers/Cassani.pdf.

[5] Cordeau, J.-F., Iori, M., Laporte, G., and Salazar González, J.-J., 2007a. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks, forthcoming.

[6] Cordeau, J.-F., Laporte, G., Potvin, J.-Y., and Savelsbergh, M.W.P., 2007b. Transportation on demand. In Transportation, Handbooks in Operations Research and Management Science 14, Barnhart, C., and Laporte, G. (eds). Elsevier, Amsterdam, 429–466.

[7] Cordeau, J.-F., Laporte, G., and Ropke, S., 2008. Recent Models and Algorithms for One-to-One Pickup and Delivery Problems. In Vehicle Routing: Latest Advances and Challenges, B. L., Golden, S. Raghavan and E. A. Wasil (eds), Kluwer, Boston, forthcoming.

[8] Dumitrescu, I., Ropke, S., Cordeau, J.-F., and Laporte, G., 2007. The traveling salesman problem with pickup and deliveries: Polyhedral results and branch-and-cut algorithm. Submitted for publication.

[9] Ficarelli, F., 2005. Mathematical programming algorithms for the TSP with rear-loading. Degree Thesis, Universit di Milano, Italy. http://optlab.dti.unimi.it/Papers/Ficarelli.pdf.

[10] Fischetti, M., and Toth, P., 1989. An additive bounding procedure for combinatorial optimization problems. Operations Research 37, 319–328.

[11] Gendreau, M., Hertz, A., and Laporte, G., 1992. New insertion and postoptimization procedures for the traveling salesman problem. Operations Research 40, 1086–1094.

[12] Gendreau, M., Laporte, G., and Vigo, D., 1999. Heuristics for the traveling salesman problem with pickup and delivery. Computers & Operations Research 26, 699–714

[13] Healy, P., and Moll, R., 1995. A new extension of local search applied to the dial-a-ride problem. European Journal of Operational Research 83, 83–104.

[14] Kalantari, B., Hill, A.V., and Arora, S.R., 1985. An algorithm for the traveling salesman problem with pickup and delivery requests. European Journal of Operational Research 22, 377–386.

[15] Ladany, S.P., and Mehrez, A., 1984. Optimal routing of a single vehicle with loading and unloading constraints. Transportation Planning and Technology 8, 301–306.

[16] Levitin, G., and Abezgaouzb, R., 2003. Optimal routing of multiple-load AGV subject to LIFO loading constraints. Computers & Operations Research 30, 397–410.

[17] Reinelt, G., 1991. TSPLIB - A traveling salesman problem library. ORSA Journal on Computing 3, 376–384.

[18] Renaud, J., Boctor, F.F., and Ouenniche, J., 2000. A heuristic for the pickup and delivery traveling salesman problem. Computers & Operations Research 27, 905–916.

[19] Renaud, J., Boctor, F.F., Laporte, G., 2002. Perturbation heuristics for the pickup and delivery traveling salesman problem. Computers & Operations Research 29, 1129–1141.

[20] Rosenkrantz, D.J., Stearns, R.E., and Lewis II, P.M., 1977. An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing, 6, 563–581.

[21] Ruland, K.S., Rodin, E.Y., 1997. The pickup and delivery problem: faces and branch-and-cut algorithm. Computers and Mathematics with Applications 33, 1–13.

[22] Savelsbergh, M.W.P., 1990. An efficient implementation of local search algorithms for constrained routing problems. European Journal of Operational Research 47, 75–85.

[23] Wong, R.T., 1980. Integer programming formulations of the traveling salesman problem. Proceedings of the IEEE International Conference of Circuits and Computers 149–152.

Instance	$\|V\|$	GENIUS start			FIFO nearest neighbor		
		Initial Obj.	Best Obj.	Avg. Seconds	Initial Obj.	Best Obj.	Avg. Seconds
brd14051	25	10038	**4341**	0.44	11949	**4341**	0.45
	51	18569	**7447**	2.06	17192	**7447**	2.61
	75	25304	**6773**	8.64	27150	6856	5.42
	101	38929	**9667**	17.78	39008	10634	16.25
	251	97631	21009	146.32	102745	**20916**	137.35
	501	215852	47863	1034.08	216417	**45357**	1248.39
	751	333209	77294	3072.93	315269	78752	3414.74
d15112	25	139199	**97806**	0.47	130299	**97806**	0.46
	51	226610	**137611**	2.57	218610	**137611**	2.94
	75	407758	**193482**	5.27	406692	197860	8.93
	101	547303	262271	12.95	508587	262145	12.58
	251	1220017	508528	225.98	1290163	**501714**	111.69
	501	2331433	866394	1073.01	2316424	858871	1083.69
	751	3370327	**1186067**	3621.63	3369702	1198451	2521.60
d18512	25	10038	**4341**	0.44	11949	**4341**	0.45
	51	23550	**7197**	2.83	23906	**7197**	2.06
	75	22263	8469	5.71	23833	**8274**	6.65
	101	37659	**9546**	11.89	38461	9605	13.24
	251	90119	22464	123.08	86475	22295	113.05
	501	210369	45797	813.05	199710	46862	1070.24
	751	330371	75994	2791.36	316009	77358	3454.36
fnl4461	25	4783	**2285**	0.44	3482	**2285**	0.44
	51	9543	**4125**	3.31	8793	4158	3.23
	75	14946	5679	6.54	14137	5796	5.93
	101	19577	7964	15.33	18961	**7816**	23.41
	251	72180	**26192**	145.38	69718	26313	155.96
	501	271893	68981	738.73	264988	**62619**	985.05
	751	493697	109872	3434.58	499111	108204	3050.54
nrw1379	25	5153	**3137**	0.45	6109	**3137**	0.53
	51	13076	**5194**	3.05	10155	**5194**	2.79
	75	16629	**6716**	6.50	17602	**6716**	10.60
	101	24484	**8637**	16.58	22754	8784	13.04
	251	72154	23401	135.37	74719	23450	230.54
	501	149702	**53319**	811.26	155263	53582	804.89
	751	269001	93118	3651.71	268028	**92397**	4154.28
pr1002	25	24275	**18439**	0.47	22804	**18439**	0.45
	51	61143	**29001**	3.07	60542	**29001**	3.48
	75	73487	44490	6.80	74300	44078	8.78
	101	124934	**58767**	19.38	121368	59025	15.83
	251	545264	180318	108.94	513706	**176940**	93.42
	501	1187901	428428	798.92	1176970	**422268**	615.46
	751	2185112	**702368**	4315.18	2171548	732741	2161.63

Table 1: Computational results for the probabilistic tabu search algorithm.

Instance	$\|V\|$	GENIUS start			FIFO nearest neighbor		
		Initial Obj.	Best Obj.	Avg. Seconds	Initial Obj.	Best Obj.	Avg. Seconds
brd14051	25	10038	**4341**	0.10	11949	**4341**	0.10
	51	18569	**7447**	0.86	17192	**7447**	0.72
	75	25304	**6773**	3.48	27150	**6773**	3.27
	101	38929	9821	11.08	39008	**9667**	10.93
	251	97631	22053	178.36	102745	22191	210.87
	501	215852	47635	2105.27	216417	46072	1945.53
	751	333209	78245	7379.93	315269	**77147**	6891.45
d15112	25	139199	**97806**	0.08	130299	**97806**	0.08
	51	226610	**137611**	0.51	218610	**137611**	0.62
	75	407758	197860	1.94	406692	197266	2.07
	101	547303	261553	10.10	508587	**259013**	7.87
	251	1220017	514572	88.51	1290163	511968	87.42
	501	2331433	**855939**	1140.91	2316424	881198	860.64
	751	3370327	1199695	3545.55	3369702	1202478	4265.52
d18512	25	10038	**4341**	0.09	11949	**4341**	0.10
	51	23550	**7197**	0.62	23906	**7197**	0.61
	75	22263	8277	3.29	23833	8277	3.37
	101	37659	9563	14.26	38461	9563	10.94
	251	90119	22214	174.88	86475	**21822**	135.09
	501	210369	**45290**	2790.34	199710	48431	1860.55
	751	330371	**75677**	6082.13	316009	77659	4756.27
fnl4461	25	4783	**2285**	0.09	3482	**2285**	0.09
	51	9543	**4125**	0.66	8793	**4125**	0.70
	75	14946	**5640**	3.59	14137	**5640**	2.64
	101	19577	7817	9.19	18961	7817	9.43
	251	72180	26730	110.26	69718	26695	118.73
	501	271893	68833	1475.13	264988	67365	1764.81
	751	493697	**106951**	6125.05	499111	109754	6289.39
nrw1379	25	5153	**3137**	0.10	6109	**3137**	0.10
	51	13076	5209	0.50	10155	5209	0.59
	75	16629	**6716**	2.91	17602	**6716**	3.60
	101	24484	8806	10.62	22754	8806	11.86
	251	72154	23775	145.34	74719	**23271**	121.12
	501	149702	54183	1142.84	155263	54775	1319.04
	751	269001	94148	3785.97	268028	94639	3744.99
pr1002	25	24275	**18439**	0.10	22804	**18439**	0.09
	51	61143	**29001**	0.64	60542	**29001**	0.74
	75	73487	43978	2.06	74300	**43920**	2.67
	101	124934	59536	8.02	121368	59892	6.45
	251	545264	185801	135.30	513706	184608	133.54
	501	1187901	433791	866.22	1176970	440720	968.92

Table 2: Computational results for the iterated local search algorithm.

		GENIUS-PTS		FIFO-PTS		GENIUS-ILS		FIFO-ILS	
Instance	$\|V\|$	Avg. Dev.	Opt. Count	Avg. Dev.	Opt. Count	Avg. Dev.	Opt. Count	Avg. Dev.	Opt. Count
brd14051	25	0.00%	5	0.00%	5	0.00%	5	0.00%	5
	51	3.33%	1	0.98%	1	3.12%	4	0.01%	3
	75	5.70%	1	9.08%	0	3.28%	2	4.16%	3
	101	4.19%	1	9.35%	0	5.37%	0	5.34%	1
	251	4.60%	0	6.85%	1	7.29%	0	9.29%	0
	501	6.70%	0	4.45%	1	9.47%	0	7.55%	0
	751	3.89%	0	4.75%	0	3.87%	0	2.29%	1
d15112	25	0.00%	5	0.00%	5	0.00%	5	0.00%	5
	51	1.12%	1	2.00%	1	0.86%	2	1.37%	1
	75	2.21%	1	3.50%	0	3.70%	0	2.08%	0
	101	3.19%	0	3.39%	0	2.84%	0	2.26%	1
	251	3.37%	0	4.05%	1	4.91%	0	5.14%	0
	501	2.91%	0	1.42%	0	3.34%	1	5.03%	0
	751	2.11%	1	2.51%	0	5.19%	0	3.02%	0
d18512	25	0.00%	5	0.00%	5	0.00%	5	0.00%	5
	51	0.78%	2	1.38%	1	0.88%	4	1.82%	2
	75	11.00%	0	4.36%	1	7.04%	0	6.48%	0
	101	3.63%	1	5.71%	0	5.80%	0	5.20%	0
	251	5.57%	0	5.65%	0	7.30%	0	4.04%	1
	501	6.25%	0	5.09%	0	3.71%	1	9.31%	0
	751	3.14%	0	3.66%	0	2.05%	1	5.95%	0
fnl4461	25	0.00%	5	0.00%	5	0.00%	5	0.00%	5
	51	3.92%	1	3.54%	0	4.27%	2	0.12%	2
	75	3.14%	0	6.25%	0	3.20%	3	0.44%	3
	101	3.06%	0	2.26%	1	3.85%	0	1.38%	0
	251	3.13%	1	1.94%	0	4.43%	0	3.67%	0
	501	12.20%	0	10.70%	1	12.16%	0	11.18%	0
	751	5.14%	0	3.59%	0	5.63%	1	7.52%	0
nrw1379	25	0.00%	5	0.00%	5	0.34%	4	0.00%	5
	51	1.08%	1	2.79%	1	0.29%	0	0.59%	0
	75	3.01%	2	1.19%	2	1.08%	3	3.01%	2
	101	3.30%	1	4.20%	0	4.23%	0	3.50%	0
	251	3.85%	0	2.94%	0	5.11%	0	4.29%	1
	501	2.36%	1	3.16%	0	4.21%	0	4.21%	0
	751	2.72%	0	2.38%	1	4.10%	0	4.41%	0
pr1002	25	0.00%	5	0.00%	5	0.00%	5	0.00%	5
	51	0.00%	5	0.47%	4	0.00%	5	0.00%	5
	75	1.82%	0	1.66%	0	1.31%	0	2.12%	1
	101	2.28%	1	1.51%	0	3.32%	0	3.48%	0
	251	6.28%	0	3.44%	1	8.30%	0	6.15%	0
	501	3.94%	0	3.35%	1	3.81%	0	5.54%	0
	751	2.84%	1	6.36%	0	7.11%	0	7.85%	0
Avg:		1.96%	1.26	2.05%	1.17	2.24%	1.38	2.15%	1.36
Max:		12.20%	5	10.70%	5	12.16%	5	11.18%	5

Table 3: Comparison of four algorithms (GENIUS = GENIUS start; FIFO = FIFO nearest neighbor; PTS = Probabilistic tabu search; ILS = Iterated local search)