Abstract

In this paper a model and several solution procedures for a novel type of vehicle routing problems where time windows for the pickup of perishable goods depend on the dispatching policy used in the solution process are presented. This problem is referred to as Vehicle Routing Problem with multiple interdependent time windows (VRPmiTW) and is motivated by a project carried out with the Austrian Red Cross blood program to assist their logistics department. Several variants of a heuristic constructive procedure as well as a branch-and-bound based algorithm for this problem were developed and implemented. Besides finding the expected reduction in costs when compared with the current procedures of the Austrian Red Cross, the results show that the heuristic algorithms find solutions reasonably close to the optimum in fractions of a second. Another important finding is that increasing the number of pickups at selected customers beyond the theoretical minimum number of pickups yields significantly greater potential for cost reductions.

Keywords: Vehicle routing problem; Multiple time windows; Interdependent time windows

1. Introduction

In this paper we will discuss a problem motivated by a project carried out in cooperation with the Austrian Red Cross blood program. This new type of problem is characterized by two fundamental extensions to the Vehicle Routing Problem with Time Windows (VRPTW) which result in a vehicle routing problem with multiple interdependent time windows (VRPmiTW) for each customer. The first extension is that goods are continuously produced at each customer location during a certain production time window and at a certain rate that is not necessarily constant.

The second extension introduces another dynamic component to the model by adding a time constraint for further processing of produced goods at the central depot. We assume that any goods produced at the customer locations are perishable and have to arrive at the depot within a certain time span after production. This leads to interdependencies between all pickups at a customer location and, as we will show later, between all pickups on a tour. This is because...
deterioration time and travel time to the depot determine a maximum time span between two subsequent pickups at a single customer location.

Assuming continuous production during production time windows as well as fixed deterioration and travel times enables us to calculate a minimum number of pickups for each customer. Given a fixed number of pickups the described problem structure allows us to further generate time windows for all pickups that are related to each other. More precisely, the time of a pickup at a given customer determines the latest pickup time for subsequent pickups at this customer.

The problem we deal with in this paper originates from the blood collection process of the Austrian Red Cross blood program. In Eastern Austria, all blood needs to be processed in one blood bank centrally located in Vienna within about 5 h after donation due to processing requirements. It is a stated objective of the Austrian Red Cross blood program that donated blood should not perish.

At the campaigns teams collect the donations and provide for a proper storage of these. Due to the perishability of the blood, the durations and the locations of the campaigns, it is generally not possible for the campaign teams to simply bring back all the donated blood to the blood bank upon the end of a campaign. Rather, additional vehicles need to be assigned to pick up collected blood from the different events in regular intervals. These additional vehicles are dispatched from the blood bank and have to return to the blood bank.

Currently, this blood transport is manually scheduled. Obviously, the Austrian Red Cross is interested in organizing donation transportation in a cost-efficient way. The goal is to find minimum cost tours and to allocate the appropriate transport devices to take back all the blood. The objective is to minimize the total driving time.

The critical issue in this problem is the fact that the dispatching decisions made in the solution procedure directly influence the time windows for visits at the campaign events. Consider the following example: there is one blood donation campaign from which it takes 60 min to return to the blood bank. Given the fact that donated blood must be no older than 300 min by the time it is further processed at the blood bank this implies that the time between two consecutive pickups at the campaign must not exceed 240 min, if no blood is to perish. Thus, if the dispatcher decides that the campaign is visited at 9 h in the morning the latest possible time for the consecutive pickup is at 13 h. While this calculation is trivial if one assumes that the vehicle directly returns to the blood bank after visiting the campaign location, it gets much more complicated if pickups at several campaigns are combined on one tour—especially when additional waiting times and different delivery dates must be combined.

The interdependencies of the multiple time windows at the customer locations constitute the main novelty in the problem formulation presented in this paper and the major challenge for the solution procedures developed. Because of this problem feature we can consider the problem at hand as a special case of the VRPTW. For a recent overview of the different algorithms for solving the VRPTW, see [1,2]. In the last years different metaheuristics mainly based on local search methods were developed which provide fairly good results [3–5]. There exist also a few exact approaches based on Lagrangian relaxation [6] and a branch-and-cut approach [7].

While problems with multiple time windows have been studied in the past, see e.g. [8], these previous works deal with cases where each customer has to be visited just once, within a choice of one of several possible time windows.

The problem considered in this paper is more closely related to the periodic VRPTW. In [9] the unified tabu search heuristic is applied to solve this problem, while a heuristic approach to a real world period vehicle routing problem for the collection of recycling paper containers is presented in [10].

For the problem class of the period travelling salesman problem, where over a period of time each customer must be visited at least once with some customers requiring several visits, Chao et al. [11] and Paletta [12,23] report on the development of heuristic approaches, while in [13] this problem class is solved by using a parallel genetic algorithm and local search heuristics. These problems do not have time window and maximum tour length constraints.

The design of an exact solution procedure based on integer programming for a VRPTW faced by the American Red Cross is studied in [14]. In that approach the objective is to bring a predetermined amount of blood from the campaign events back to the processing facility at a minimum cost, where the amount collected at each campaign depends on the time of the pickup. Only a single pickup per campaign event is considered thus eliminating interdependence between time windows. Still, a maximum tour length exists due to perishability of blood. Already in 1976 Or developed solution techniques for transportation problems in blood banks in his PhD-thesis [24].

Our problem is somewhat similar to the Inventory Routing Problem (see, e.g. [15]). In the Inventory Routing Problem the main time horizon is generally a few weeks or days and the main issue is to determine an optimal visit frequency at the customers to jointly minimize transportation and inventory costs. In this setting multiple visits per day at the same customer are very unlikely and thus the actual routing of the vehicles does not influence the feasibility of a given
frequency. On the contrary, in our problem changing the sequence of a route may render another route infeasible and lead to the necessity of additional pickups at several other customers. Besides this main difference in the problems the routing decisions do of course influence the transportation costs in both the Inventory Routing Problem and our VRPmiTW.

Summarizing, all these approaches are not applicable to our problem because some essential features are missing. Thus, to solve the VRPmiTW we develop and apply several variants of a constructive heuristic as well as an exact branch-and-bound algorithm. Both approaches are based on the savings idea [16]. Our results show that the heuristic techniques produce solutions reasonably close to the optimal solutions in negligible time for a basic case, where the number of pickups is fixed at the theoretical minimum number of pickups for each customer. Furthermore, increasing the number of pickups for selected customers beyond this minimum number was found to yield high additional cost reductions. While in these cases computational times for the exact method become prohibitively large our heuristics provide very good solutions within seconds. Thus, the contribution of our paper is twofold, namely (i) the modelling of this novel problem and (ii) the development of our solution procedures which are shown to efficiently find solutions of high quality.

The remainder of the paper is organized as follows. In Section 2 we describe our mixed-integer programming formulation for the problem at hand. Sections 3 and 4 present a general description of our heuristic and exact techniques followed by implementation issues associated with feasibility requirements and time window propagation. Results from our computational analysis are given in Section 5 before we conclude with an outlook on issues for further research.

2. Model formulation

In this section we give a complete mixed-integer programming model formulation for the problem described above. Let us first introduce necessary notation, data and decision variables.

Let \( C \) denote a set of customers, where \( i, j \in C \cup \{0\} \) are indices used for customers and the depot. The depot will always be given index 0.

Let \( P_i = \{1, \ldots, p_i^{\text{max}}\} \) denote a set of possible pickups at customer \( i \) and let \( h, k \in P_i \) be indices for pickups.

Let \( T \) denote a set of tours, where \( l \in T \) is an index for tours.

Let \( S = \{1, \ldots, s^{\text{max}}\} \) denote a set of stops at each tour, where \( m \in S \) is an index for stops. Note that the values for \( p_i^{\text{max}} \) and \( s^{\text{max}} \) are simply upper bounds for the number of pickups at a customer \( i \in C \) and the number of stops on a tour \( l \in T \), respectively. In our model these values are chosen large enough in order not to influence the solution.

Furthermore, we use parameter \( \tau \) to denote deterioration time for transported goods and data \( a_i \) and \( b_i \) to denote starting and ending time of production at customer \( i \). Finally, \( f_i \) is the service time at location \( i \) (customer or depot) \( d_{ij} \) indicates the distance between locations \( i \) and \( j \) where \( i, j \in C \cup \{0\} \).

Using these data we can compute the minimum number of required pickups at customer \( i \), denoted by \( r_i \). As stated in the last section, depending on deterioration and transportation time each customer \( i \) needs to be visited multiple times to ensure that no goods are spoiled. Based on the deterioration time \( \tau \) we can define the maximum possible time span between two consecutive pickups at customer \( i \) under the assumption that the vehicle directly returns to the depot after picking up goods at customer \( i \). This time span, denoted by \( t_i \), can be computed as follows:

\[
t_i = \tau - f_i - d_{i0} - f_0, \quad i \in C.
\]

Using the value of \( t_i \) and the duration of goods production at customer \( i \) the minimum number of pickups at customer \( i \) is calculated as

\[
r_i = \left\lceil \frac{b_i - a_i}{t_i} \right\rceil, \quad i \in C.
\]

We define the following decision variables:

\[
y_{ik} = \begin{cases} 1 & \text{if the } k\text{th pickup of customer } i \text{ is realized,} \\ 0 & \text{otherwise,} \end{cases}
\]

\[
x_{iklm} = \begin{cases} 1 & \text{if the } k\text{th pickup of customer } i \text{ is performed as the } m\text{th-last stop on tour } l, \\ 0 & \text{otherwise,} \end{cases}
\]
\[ \begin{align*}
  z_{ijkh} &= \begin{cases} 
    1 & \text{if the } k \text{th pickup of customer } i \text{ is performed immediately before the } h \text{th pickup of customer } j, \\
    0 & \text{otherwise}, 
  \end{cases} \\
  u_{ik} &= \begin{cases} 
    1 & \text{if the } k \text{th pickup of customer } i \text{ is performed first on a tour}, \\
    0 & \text{otherwise}. 
  \end{cases}
\end{align*} \]

Additionally, we use \( v_{ik} \) to denote the arrival time at customer \( i \) for the \( k \)th pickup and \( w_l \) to denote the return time of tour \( l \) to the depot.

Note that in (4) the index \( m \) is counted backward from the last position on a tour. This is convenient for computing the return time \( w_l \) for each tour \( l \) in formula (23) below.

The objective function

\[
\min \sum_{i \in C} \sum_{j \in C} \sum_{k \in P_i} \sum_{h \in P_j} d_{ij} z_{ijkh} + \sum_{i \in C} \sum_{k \in P_i} \sum_{l \in T} d_{i0} x_{ikl} + \sum_{i \in C} \sum_{k \in P_i} d_{0i} u_{ik}
\]

minimizes the total distances between all successive stops on all tours (first term), from the last stop on each tour to the depot (second term) and from the depot to the first stop on each tour (third term). This objective has to be minimized subject to the following constraints, where \( M \) is used to denote a large integer number:

\[
\begin{align*}
  y_{ik} &\leq y_{ik-1} \quad \forall i \in C, \; k \in P_i \setminus \{1\}, \\
  y_{ir} &= 1 \quad \forall i \in C, \\
  y_{ik} &= \sum_{l \in T} \sum_{m \in S} x_{iklm} \quad \forall i \in C, \; k \in P_i, \\
  \sum_{k \in P_i} \sum_{m \in S} x_{iklm} &\leq 1 \quad \forall i \in C, \; l \in T, \\
  \sum_{i \in C} \sum_{k \in P_i} x_{ikl(m+1)} &\leq \sum_{j \in C} \sum_{h \in P_j} x_{jhlm} \quad \forall l \in T, \; m \in S \setminus \{s_{\text{max}}\}, \\
  z_{ijkh} &\geq x_{ikl(m+1)} + x_{jhlm} - 1 \quad \forall i, \; j \in C, \; l \in T, \; k \in P_i, \; h \in P_j, \; m \in S \setminus \{s_{\text{max}}\}, \\
  \sum_{i \in C} \sum_{k \in P_i} z_{ijkh} &\leq 1 \quad \forall j \in C, \; h \in P_j, \\
  \sum_{j \in C} \sum_{h \in P_j} z_{ijkh} &\leq 1 \quad \forall i \in C, \; k \in P_i, \\
  u_{ik} &\geq x_{iklm} - \sum_{j \in C} \sum_{h \in P_j} x_{jhl(m+1)} \quad \forall i \in C, \; k \in P_i, \; l \in T, \; m \in S \setminus \{s_{\text{max}}\}, \\
  u_{ik} &\geq x_{ikls_{\text{max}}} \quad \forall i \in C, \; k \in P_i, \; l \in T, \\
  v_{i0} &= a_i \quad \forall i \in C, \\
  v_{il} &\leq b_i - \tau + f_i + d_{i0} + f_0 \quad \forall i \in C, \\
  v_{ik} - v_{i(k-1)} &\geq 0 \quad \forall i \in C, \; k \in P_i, \\
  v_{ik} - M y_{ik} &\leq v_{i(k-1)} \quad \forall i \in C, \; k \in P_i \setminus \{1\}, \\
  v_{ik} + d_{ij} + f_i &\leq v_{jh} + M(1 - z_{ijkh}) \quad \forall i, \; j \in C, \; k \in P_i, \; h \in P_j,
\end{align*}
\]
\[ w_l \geq v_{ik} + f_i + d_{il} + f_0 - M(1 - x_{ikl1}) \quad \forall i \in C, \; k \in \mathcal{P}_i, \; l \in \mathcal{J}, \quad (23) \]
\[ w_l - v_{i(k-1)} - M \left( 1 - \sum_{m \in \mathcal{S}} x_{iklm} \right) \leq \tau \quad \forall i \in C, \; k \in \mathcal{P}_i \setminus \{1\}, \; l \in \mathcal{J}. \quad (24) \]

Constraints (8) and (9) ensure that at least \( r_i \) pickups are performed at customer \( i \), and that the \( k \)th pickup can only be performed only if the \((k - 1)\)st pickup is performed. Constraints (10) express that any performed pickup has to be scheduled on a tour, while constraints (11) state that each customer can be visited at most once on a given route. Constraints (12) ensure that a pickup at a customer can be assigned to the \((m + 1)\)-last stop on a tour only if the \(m\)-last stop of this tour is performed. Constraints (13)–(15) constitute the routing and flow conservation constraints. Constraints (16) and (17) fix the customer to visit first for each tour, where only pickups that are performed qualify. Constraints (18) and (19) fix the relevant earliest and latest times for the first and last pickup at each customer, respectively. Constraints (20) and (21) require that the visiting times assigned to two consecutive pickups of a given customer are scheduled in correct order. Constraints (22) and (23) constitute the temporal reality on each route. In particular, constraints (23) determine the earliest return time of a given route to the depot. Finally, constraints (24) ensure that no goods picked up on a given route will perish.

The above model was implemented using Xpress-MP (Version 14.20.9) and run on a Pentium 4 with 2.4 GHz. It turned out that the bounds obtained in the root node of the branch-and-bound search are very poor such that even for very small problem sizes the integer part of the optimization takes several days. To speed up calculations we decided to no longer treat the number of pickups as decision variables in the model. Rather, we fix them in advance and supply them as data to the model. Clearly this procedure does not guarantee finding the optimal solution such that several values have to be systematically analyzed. More precisely for each customer we start with the minimum number of pickups \( p^{\text{max}}_i = r_i \) and increase this value stepwise. Despite this simplification, computational effort was in the range of days even for the smallest instance. Thus starting from this idea we developed custom-made heuristic and exact algorithms, which exploit specific problem structures in order to generate solutions more quickly. These approaches will be described in the next sections.

3. Constructive solution techniques

3.1. Tour construction approach

Let us start with a simplifying assumption that will be relaxed later. For each customer, we fix the number of pickups at the minimum number of required pickups \( r_i \) using Eqs. (1) and (2). This is based on the intuition that in general a smaller number of pickups at each customer should imply a smaller total distance. Based on the \( r_i \) values, we can then compute the preliminary time window for each of the pickups at customer \( i \). The term preliminary refers to the fact that fixing the time of a particular pickup changes the time windows of all other pickups for that customer.

The calculation of the preliminary time window of the \( k \)th pickup at customer \( i \) is given by its start time
\[ a_{ik} = b_i - (r_i - k + 1) \cdot t_i \quad \forall i \in C, \; k \in \mathcal{P}_i \quad (25) \]
and its end time
\[ b_{ik} = a_i + k \cdot t_i \quad \forall i \in C, \; k \in \mathcal{P}_i. \quad (26) \]

The latest arrival time \( b_{ik} \) for pickup \( k \) at customer \( i \) is the begin of production time at this customer \( a_i \) plus \( k \) times the maximum time span between pickups \( t_i \) at customer \( i \) as shown in Eq. (26). The initial earliest arrival time \( a_{ik} \) for pickup \( k \) at customer \( i \) is calculated as shown in Eq. (25). Its calculation follows the same idea but beginning at the end of production time at customer \( i \) and moving forward in time as values for \( k \) decrease. The forward and backward computation of these times is also shown in Fig. 1.

Finally, we can define an initial slack time for pickup \( k \) of customer \( i \) as the difference between the earliest and latest possible times of each pickup \( k \):
\[ s_{ik} = b_{ik} - a_{ik}. \quad (27) \]
3.2. Feasible concatenation of tours

Our constructive solution procedures start with a situation where every pickup at a customer forms a single tour. As a basic principle in each step two pickups of two distinct tours are joined to form a new tour. A number of constraints have to be fulfilled to ensure that a candidate combination of two pickups can be joined to a feasible tour. The first four of these constraints are related to the feasibility of two partial tours to be joined, while the remaining two constraints deal with previous and successive pickups at customers visited and thus affected by a possible join.

Let $P$ and $Q$ denote two partial tours, 

$$
P = (p_1, p_2, \ldots, p_n) \quad \text{and} \quad Q = (q_1, q_2, \ldots, q_m), 
$$

which are sequences of pickups on these partial tours, where each element in $P$ and $Q$ denotes a pickup, i.e. pickup $p_k$ corresponds to the $k$th pickup at customer $i$. Further, let $a_{p_k}$ and $b_{p_k}$ denote the earliest and latest possible pickup times at this customer. Finally, let $c_P$ and $c_Q$ correspond to the total time consumption needed to go from the first customer of tours $P$ and $Q$ to the depot and $s_P$ and $s_Q$ denote the slack times on the two tours $P$ and $Q$, respectively. On a partial tour the slack times at each pickup on this tour are identical, such that they can be represented as a single slack time for the tour. A concatenation of the two partial tours $P$ and $Q$ is possible if the following constraints are met.

**Premature pickup constraint:** Rather than a constraint this is an observation. It says that on each tour, at most one pickup of any given customer has to be scheduled. This is straightforward. In fact, consider the case where a customer is visited more than once on a feasible tour. Then, all the goods picked up at earlier pickups of the considered customer along the tour were collected prematurely. By simply eliminating these earlier pickups and assigning all the customers’ goods to the last pickup, the tour gets shorter and still remains feasible.

**Reachability constraint:** Eq. (29) is basically a time window constraint and ensures that the vehicle performing partial tour $P$ can visit the first customer (and thus all the others also) on partial tour $Q$ in time:

$$
a_{p_n} + f_{p_n} + d_{p_n}q_1 \leq b_{q_1}. 
$$

**Detour constraint:** This constraint ensures that the slack time of partial tour $P$ is sufficient to cover the detour caused by appending partial tour $Q$ at the end of partial tour $P$. Note that the left-hand side is the latest time at which goods collected on partial tour $P$ can arrive at the depot:

$$
b_{p_n} + f_{p_n} + d_{p_n}q_1 \geq a_{p_n} + f_{p_n} + d_{p_n}q_1 + c_Q + d_{q_1}q_0. 
$$

**Punctuality constraint:** Constraint (31) deals with the delay caused by appending partial tour $Q$ at the end of partial tour $P$. Obviously, constraints (30) and (31) both deal with the perishability of goods on partial tour $P$. The latter additionally ensures, that both partial tours can be synchronized:

$$
b_{p_n} + f_{p_n} + d_{p_n}q_1 \geq a_{q_1} + c_Q + d_{q_1}q_0. 
$$

**Extended detour constraint:** Concatenation of two partial tours $P$ and $Q$ requires enough slack time at partial tour $P$ to cover the arising detours like expressed in constraint (30). Furthermore, detours cause the earliest pickup time at the first customer on a tour to shift to a later date. As a consequence also pickup times of earlier pickups at the same customer are shifted to a later date in order to prevent from exceeding the maximum time span between consecutive pickups. Whenever earlier pickups at the same customer location exist for any pickup in partial tour $P$, their current slack times have to be examined and it has to be checked, whether their respective detours could be covered or not.
3.3. Time window propagation

After a join of partial tours \( P \) and \( Q \) to a new tour \( E \) was accepted and performed as denoted in formula (32), the time windows of the joined tours have to be updated. Also, time windows of the previous and following pickups at the related customers or already established other tours involving these customers must be adjusted. Note that this propagation may affect quite many different pickups on different tours. For the VRPTW an approach for time window propagation and computing slack times was introduced in [17].

\[ E = P \otimes Q = (p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_m) = (e_1, e_2, \ldots, e_{n+m}). \]

The change in the earliest possible pickup times of partial tours \( P \) and \( Q \) is given as follows:

\[ a_{q_1} = a_{q_1} - (a_{p_n} + f_{p_n} + d_{p_n q_1}), \quad \forall p_k \in P, \]

\[ a_{q_k} := a_{q_k} + \max(0, \Delta a), \quad \forall q_k \in Q. \]

These equations show that all (possible) waiting times are eliminated from the tours. In fact, this is intuitively clear if we consider the following case: If tour \( P \) starting at its earliest possible time were to arrive early at the first customer to be visited on partial tour \( Q \) and the combined tour was feasible in spite of the resulting waiting time, then partial tour \( P \) could be shifted to eliminate this waiting time without violating the feasibility of the resulting tour.

Having updated the earliest possible pickup times at all customers on tour \( E \), we can now turn to the update of the latest possible pickup times at all customers. This update is determined by a higher number of influencing factors than the previous calculation of earliest pickup times.

Let us first turn to the latest possible pickup times on partial tour \( P \) and its adjustment period \( \Delta b_P \):

\[ \Delta b_P = \max(d_{p_n q_1} + c_Q + d_{q_m 0} - d_{p_n 0}, b_{p_n} + f_{p_n} + d_{p_n q_1} - b_{q_1}), \quad \forall p_k \in P. \]

Clearly, the latest possible pickup times cannot increase. This is ensured by the fact that the first term in Eq. (36) is strictly nonnegative (due to the triangle inequality), which implies that \( \Delta b_P \geq 0 \) and thus according to Eq. (37), \( b_{p_k} \) will decrease for all \( k \). In fact, Eq. (36) tells us whether the detour constraint (first term) or the fact that goods from partial tour \( Q \) must not get spoiled (second term) is more restrictive for partial tour \( P \).

The latest possible pickup times for customers on partial tour \( Q \) have to be computed differently. The formal update is given as follows:

\[ \Delta b_Q = \max(0, (b_{q_1} + c_Q + d_{q_m 0}) - (b_{p_n} + f_{p_n} + d_{p_n 0})), \quad \forall q_k \in Q. \]

As was true for partial tour \( P \), the latest possible pickup times for partial tour \( Q \) cannot increase either. In fact, depending on which of the two partial tours has to reach the depot earlier to avoid goods from perishing, they can either remain unchanged or decrease by the difference in the latest possible delivery times at the depot. More precisely, if the goods on partial tour \( P \) have to reach the depot earlier than the goods on partial tour \( Q \), i.e., \( (b_{p_n} + f_{p_n} + d_{p_n 0}) < (b_{q_1} + c_Q + d_{q_m 0}) \), the latest possible pickup times for \( Q \) change, otherwise they remain unchanged.

Having updated both the earliest and latest possible pickup times for all customers on tour \( E \), the last step is to propagate these time window changes to the remaining previous and successive pickups at these customers. More precisely, if the \( k \)th pickup of customer \( i \) is on tour \( E \) we need to update the latest possible pickup time for all successive pickups at \( i \), i.e., for all pickups \( h = k + 1, \ldots, r_i \). Analogously, we need to update the earliest possible pickup times for all previous pickups at customer \( i \), i.e., for all pickups \( h = 1, \ldots, k - 1 \).
3.4. Heuristic solution procedures

Let us now turn to the heuristic approach based on the savings idea and thus constitute a parallel tour construction algorithm. For each pair of customers \( i \) and \( j \) the following savings measure is calculated, see [16]:

\[
\Delta d_{ij} = d_{i0} + d_{0j} - d_{ij},
\]

where \( d_{ij} \) denotes the distance between customers \( i \) and \( j \) and the index 0 denotes the depot. Thus, the values \( \Delta d_{ij} \) contain the savings of combining pickups at customers \( i \) and customers \( j \) on one tour as opposed to serving them on two different tours.

Having computed the savings values for each pair of customers, the iterative part of the algorithm starts. This phase consists of a feasibility check, the selection of a feasible combination and the time window update and propagation.

**Step I: Determine feasible combinations.** First, we have to check the constraints to determine if there exist feasible combinations of any two pickups of two different customers. Note that at each customer location a number of pickups may be necessary, so that combinations between all of these pickups have to be checked. A combination is only feasible if all the constraints discussed in Section 3.2 are satisfied.

**Step II: Selection of a combination.** Given all feasible combinations, we have to choose a combination according to some criteria. We have implemented different variants of the realization of this combination procedure, denoted by Greedy, Savings and Savings\(^+\).

1. **Procedure Greedy:** Starting from the earliest time window of any pickup at any customer, we check whether this pickup can be combined with any other pickup. If applicable, we accept the combination. Then, proceeding in time we sequentially perform all feasible combinations. If, in any case, multiple options are available, the one with the largest savings value is chosen. A preliminary version of this variant was presented in [18].
2. **Procedure Savings:** In procedure Savings we do not combine two pickups according to the earliest possible starting time, but by using the largest savings value. Apart from that, Savings is identical to Greedy.
3. **Procedure Savings\(^+\):** This is an improved variant of the procedure Savings. In case of several identical savings values, we always choose the one where the first pickup is farthest away from the depot. This leads to later pickups at all customers. Thus, the collected goods are less likely to perish.

Note that using the savings values as described before, individual customers or partial tours are combined by concatenating the tours to each other (see Sections 3.2 and 3.3).

**Step III: Time window update.** After joining customers \( i \) and \( j \), the time windows for the next pickups (in Greedy) and also for the previous pickups (in Savings and Savings\(^+\)) of these two customers and also the list of the remaining possible combinations are updated. Clearly, the procedures Savings and Savings\(^+\) are more complex but also more promising than procedure Greedy. The time window update procedure as well as the propagation of the time windows was discussed in detail in Section 3.3.

Steps I–III are repeated as long as feasible combinations of pickups exist.

In order to better evaluate the performance and solution quality of the heuristic procedures described above, we have implemented a branch-and-bound algorithm to optimally solve the problem instances as a benchmark.

4. Branch-and-bound solution procedure

The main idea of the branch-and-bound algorithm is to use the savings values and to search through all possible combinations of pickups in a clever way. Note that, as for the heuristics, it is also necessary for the branch-and-bound algorithm to define the number of pickups at each customer location a priori.

At the initialization step of the branch-and-bound algorithm the upper bound \( \bar{d} \) of a solution is determined by heuristic Savings\(^+\) and the lower bound \( d \) is set to zero. Then two vectors \( Y_F \) and \( Y_J \) are initialized. Vector \( Y_F \) contains all pairs of pickups that are feasible at each step of the algorithm. A savings value \( \Delta d_\pi \) is assigned to each pair \( \pi \) and used to sort the pairs in \( Y_F \) in order of decreasing savings \( \Delta d_\pi \). Vector \( Y_J \) contains already joined pairs of pickups throughout the algorithm. In the initial step, \( Y_F \) holds all feasible combinations of pickups and vector \( Y_J \) is empty.

In each branching step the first pair of pickups in \( Y_F \) which obviously provides the largest savings value, is either joined or not. In both cases the combination is removed from the set \( Y_F \). Further, if it is joined it is added to the
vector $Y_1$. In this case, we also have to eliminate all pairs from $Y_F$ which are no longer feasible. Note that we have a feasible solution in each branching step. Thus, after performing a join we can check the quality of the new solution against the upper bound and update the latter if applicable.

We can also compute lower bounds in each step in the following way. Let $g_R$ denote the cumulated savings obtained through all joined pairs in $Y_J$, and $g_P$ denote the sum of the savings of all potentially joinable pairs in $Y_F$ in a branching step.

\begin{align}
  g_R &= \sum_{l \in Y_J} \Delta d_l, \\
  g_P &= \sum_{l \in Y_F} \Delta d_l.
\end{align} \hfill (41)

Then, the lower bound $\bar{d}$ can be computed by subtracting the cumulated savings $g_R$ and $g_P$ from the overall tour length $d$ without any joined pickups.

\begin{equation}
  \bar{d} = d - g_R - g_P.
\end{equation} \hfill (42)

Nodes of the branch-and-bound tree can be fathomed if not even the joining of all pairs in $Y_F$ does improve upon $\bar{d}$.

**Fig. 2** shows an example of the procedure for a problem instance with five potential pairs of pickups to join. In the top left branch we join the pair 1 with the largest savings value and the remaining vector $Y_F$ is updated. Note that in this particular case some pairs (namely 3 and 5) are no longer feasible and thus eliminated from $Y_F$. Consequently, the value of $g_P$ also changes to $g_P = s_1 + s_4$. In the top right branch we decide not to join pair 1, therefore all the remaining combinations stay feasible.

## 5. Computational analysis

The heuristic and exact approaches described in the last section were implemented and integrated into a prototype system that provides route planning and basic vehicle scheduling functions via a graphical user interface. The system is able to grab input data for real-world problem instances from the database system of the Austrian Red Cross. Relevant distance information is computed by using the commercial GIS software ArcView version 3.2 and the included routing tool Network Analyst version 1.0b developed by ESRI (see [19]). Basic road data are taken from the ArcAustria package provided by WiGeoGis (see [20]). The system uses the described solution procedures to find a set of routes and reports the recommended combinations of customers, the number of required pickups and the tour lengths. The tours are visualized on a map using the GIS system. The route calculation is followed by a greedy procedure which assigns the tours to vehicles. This procedure is necessary for practical purposes but is left out of the discussion in this paper. Basically, it creates a timetable and a list of customers to be visited for each vehicle.

All solution procedures were implemented as a Dynamic Link Library (DLL) module using ANSI C ++ and the GNU C ++ compiler version 3.2. We applied our implementation to a test set of problem instances originating from real-world data from the Austrian Red Cross. To be more specific, our computational analysis is based on 22 data sets corresponding to typical campaign days from the period May 2001 to August 2003. The number of campaign events varies between 6 and 15, the number of required pickups at each event ranges from 0 (short campaigns where the team takes all the donated blood to the blood bank at the end of the campaign) to 4. Given the number of campaign events and the required pickups at these events the number of pickups per day goes up to about 60. All our runs were performed on a Pentium 4 with 2.4 GHz under the SuSE Linux operating system.
Table 1
Results of the heuristics and the branch-and-bound algorithm for the base case

<table>
<thead>
<tr>
<th>Date</th>
<th>No. of locations</th>
<th>Greedy</th>
<th>Savings</th>
<th>Savings+</th>
<th>Branch-and-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Opt.</td>
</tr>
<tr>
<td>20.5.01</td>
<td>14</td>
<td>2889</td>
<td>2498</td>
<td>2498</td>
<td>2380</td>
</tr>
<tr>
<td>3.6.01</td>
<td>9</td>
<td>1928</td>
<td>1757</td>
<td>1757</td>
<td>1733</td>
</tr>
<tr>
<td>9.6.01</td>
<td>12</td>
<td>1232</td>
<td>1130</td>
<td>1130</td>
<td>1130</td>
</tr>
<tr>
<td>16.6.01</td>
<td>6</td>
<td>501</td>
<td>431</td>
<td>435</td>
<td>414</td>
</tr>
<tr>
<td>17.6.01</td>
<td>15</td>
<td>2196</td>
<td>1889</td>
<td>1889</td>
<td>1876</td>
</tr>
<tr>
<td>24.6.01</td>
<td>13</td>
<td>2903</td>
<td>2677</td>
<td>2677</td>
<td>2664</td>
</tr>
<tr>
<td>1.7.01</td>
<td>15</td>
<td>3889</td>
<td>3594</td>
<td>3581</td>
<td>3360</td>
</tr>
<tr>
<td>3.3.02</td>
<td>13</td>
<td>3076</td>
<td>2815</td>
<td>2815</td>
<td>2815</td>
</tr>
<tr>
<td>23.3.02</td>
<td>8</td>
<td>1245</td>
<td>1127</td>
<td>1127</td>
<td>1067</td>
</tr>
<tr>
<td>24.3.02</td>
<td>9</td>
<td>1713</td>
<td>1645</td>
<td>1645</td>
<td>1645</td>
</tr>
<tr>
<td>29.6.03</td>
<td>11</td>
<td>3040</td>
<td>2934</td>
<td>2934</td>
<td>2823</td>
</tr>
<tr>
<td>6.7.03</td>
<td>13</td>
<td>3791</td>
<td>3612</td>
<td>3595</td>
<td>3593</td>
</tr>
<tr>
<td>13.7.03</td>
<td>13</td>
<td>2435</td>
<td>2274</td>
<td>2274</td>
<td>2274</td>
</tr>
<tr>
<td>20.7.03</td>
<td>13</td>
<td>2326</td>
<td>2310</td>
<td>2181</td>
<td>2050</td>
</tr>
<tr>
<td>27.7.03</td>
<td>12</td>
<td>2375</td>
<td>2312</td>
<td>2312</td>
<td>2260</td>
</tr>
<tr>
<td>3.8.03</td>
<td>12</td>
<td>3289</td>
<td>2820</td>
<td>2820</td>
<td>2693</td>
</tr>
<tr>
<td>10.8.03</td>
<td>14</td>
<td>3293</td>
<td>3077</td>
<td>3081</td>
<td>3067</td>
</tr>
<tr>
<td>15.8.03</td>
<td>13</td>
<td>1684</td>
<td>1472</td>
<td>1472</td>
<td>1472</td>
</tr>
<tr>
<td>17.8.03</td>
<td>11</td>
<td>1448</td>
<td>1448</td>
<td>1448</td>
<td>1341</td>
</tr>
<tr>
<td>24.8.03</td>
<td>10</td>
<td>2908</td>
<td>2698</td>
<td>2698</td>
<td>2676</td>
</tr>
<tr>
<td>31.8.03</td>
<td>14</td>
<td>2160</td>
<td>1998</td>
<td>1992</td>
<td>1743</td>
</tr>
<tr>
<td>7.9.03</td>
<td>14</td>
<td>3224</td>
<td>2705</td>
<td>2705</td>
<td>2598</td>
</tr>
</tbody>
</table>

Average deviation (%) 13.26 3.47 3.18

Let us first consider the base case where no additional pickups are allowed, i.e. for each campaign event the minimum number of required pickups applies. The results in terms of solution quality and computation time are given in Table 1. Note that the computation times for the heuristics are in the range of milliseconds and are thus omitted from the table. For the branch-and-bound procedure we provide both times needed to prove optimality and in parentheses the time needed to find the optimal solution. Table 1 shows for each of the 22 days the total distance travelled in the solutions obtained by the three heuristics and the branch-and-bound algorithm, as well as the computation times required by the latter. Finally, in the last row, the average deviation (in percent) of the results obtained by the three heuristics over the optimal results are presented.

The first observation to be made from Table 1 is that the Greedy variant of the heuristic is clearly outperformed by the other two versions of the heuristic, Savings and Savings+. In fact, for 21 out of 22 instances it finds strictly worse solutions, while only for one instance all three variants produce equal solutions. Among the latter two versions, the Savings+ algorithm performs better than the Savings version. However this difference is small. More precisely, for 16 out of the 22 instances the two versions produce identical results, for 4 instances Savings+ is superior and for 2 instances Savings works best.

The second observation relates to the average deviation from the optimal solution. The inferiority of the Greedy algorithm is confirmed by its average deviation of 13.26% from the optimum. On the other hand, Savings and Savings+ are on average only 3.47% and 3.18% above the optimum. Moreover, for five problem instances these two versions of the algorithm indeed find the optimal solutions as shown by the bold entries in Table 1.

Finally, the third observation is that the branch-and-bound algorithm finds the exact solutions very fast. In fact only in one case it takes 8 s to terminate, while in all other cases it finishes within 1 s. Moreover, the optimal solution is always found within 2 s.

Overall, the results from Table 1 suggest that for this base case the branch-and-bound algorithm is efficient. However, the small deviation produced by the Savings and Savings+ version of the heuristic obtained in negligible time are both interesting as upper bounds and promising for cases where the branch-and-bound algorithm may not be efficient.
Having analyzed the results for the base case, we found that for some instances pickup combinations of events that looked appealing from the geographical point of view seemed to be infeasible due to the narrow time windows implied by the restriction to only consider for each event the minimum number of pickups.

Thus, we performed a kind of scenario analysis to investigate the merit of increasing the number of pickups. More specifically, we considered to increase the total number of pickups over all events by up to one, two, three and four pickups. First, we tried one additional pickup at each possible event and then selected the best solution. The same was done for the other three cases, with the restriction that at each event at most one additional pickup is possible.

The results for the four different cases are given in Tables 2 and 3 and in Fig. 3. In 18 out of 22 cases allowing additional pickups leads to improved optimal solutions, whereas in 4 cases no improvements are possible. Further, (as expected) the benefit from adding pickups decreases as the number of additional pickups increases. However, in five cases even adding three or four pickups yields improvements.

To get a better picture of this effect let us look at the average and worst percentage deviation from the optimum due to increasing the number of pickups. This information is shown in Fig. 4, once again for all five cases. Note that the optimum is the best optimal solution over the five cases for each instance. The results clearly show the decreasing gains from increasing the number of pickups. In fact with three additional pickups the worst deviation is already less than 1% from the optimum.

Second, let us consider the average times needed to find and prove the optimal solutions by the branch-and-bound algorithm. In Fig. 5 this information is shown and compared with the average time required for the heuristic Savings+ for the case with four additional pickups. Clearly, the times needed to find the optimal solutions goes up as the number of additional pickups increases and hence the search space grows. However, the time needed to prove optimality increases much more severely. In fact, for the case of four additional pickups the branch-and-bound algorithm did not always terminate within the time limit of 100 h. Rather, for three instances this time limit was reached and in Table 3 these results are printed in bold. Note that we have not performed runs with five or more additional pickups due to these

<table>
<thead>
<tr>
<th>Date</th>
<th>No. of locations</th>
<th>One additional pickup</th>
<th>Two additional pickups</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Greedy</td>
<td>Savings</td>
</tr>
<tr>
<td>Opt. Time (s)</td>
<td>Opt. Time (s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.5.01</td>
<td>14</td>
<td>2854</td>
<td>2498</td>
</tr>
<tr>
<td>3.6.01</td>
<td>9</td>
<td>1808</td>
<td>1733</td>
</tr>
<tr>
<td>9.6.01</td>
<td>12</td>
<td>1232</td>
<td>1088</td>
</tr>
<tr>
<td>16.6.01</td>
<td>6</td>
<td>465</td>
<td>431</td>
</tr>
<tr>
<td>17.6.01</td>
<td>15</td>
<td>2196</td>
<td>1889</td>
</tr>
<tr>
<td>24.6.01</td>
<td>13</td>
<td>2885</td>
<td>2646</td>
</tr>
<tr>
<td>1.7.01</td>
<td>15</td>
<td>3598</td>
<td>3518</td>
</tr>
<tr>
<td>3.3.02</td>
<td>13</td>
<td>3002</td>
<td>2809</td>
</tr>
<tr>
<td>23.3.02</td>
<td>8</td>
<td>1183</td>
<td>1127</td>
</tr>
<tr>
<td>24.3.02</td>
<td>9</td>
<td>1713</td>
<td>1645</td>
</tr>
<tr>
<td>29.6.03</td>
<td>11</td>
<td>2848</td>
<td>2852</td>
</tr>
<tr>
<td>6.7.03</td>
<td>13</td>
<td>3791</td>
<td>3521</td>
</tr>
<tr>
<td>13.7.03</td>
<td>13</td>
<td>2311</td>
<td>2196</td>
</tr>
<tr>
<td>20.7.03</td>
<td>13</td>
<td>2509</td>
<td>2284</td>
</tr>
<tr>
<td>27.7.03</td>
<td>12</td>
<td>2322</td>
<td>2284</td>
</tr>
<tr>
<td>3.8.03</td>
<td>12</td>
<td>3264</td>
<td>2702</td>
</tr>
<tr>
<td>10.8.03</td>
<td>14</td>
<td>3265</td>
<td>2921</td>
</tr>
<tr>
<td>15.8.03</td>
<td>13</td>
<td>1684</td>
<td>1420</td>
</tr>
<tr>
<td>17.8.03</td>
<td>11</td>
<td>1405</td>
<td>1380</td>
</tr>
<tr>
<td>24.8.03</td>
<td>10</td>
<td>2672</td>
<td>2670</td>
</tr>
<tr>
<td>31.8.03</td>
<td>14</td>
<td>2096</td>
<td>1927</td>
</tr>
<tr>
<td>7.9.03</td>
<td>14</td>
<td>3224</td>
<td>2705</td>
</tr>
</tbody>
</table>

Average deviation (%) 13.56 4.63 3.34 14.24 4.59 3.41
Table 3
Results of the heuristics and the branch-and-bound algorithm for three and four additional pickups

<table>
<thead>
<tr>
<th>Date</th>
<th>No. of locations</th>
<th>Three additional pickups</th>
<th>Four additional pickups</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.5.01</td>
<td>14</td>
<td>2801</td>
<td>2498</td>
</tr>
<tr>
<td>3.6.01</td>
<td>9</td>
<td>1808</td>
<td>1698</td>
</tr>
<tr>
<td>9.6.01</td>
<td>12</td>
<td>1238</td>
<td>1088</td>
</tr>
<tr>
<td>16.6.01</td>
<td>6</td>
<td>465</td>
<td>431</td>
</tr>
<tr>
<td>17.6.01</td>
<td>15</td>
<td>2099</td>
<td>1889</td>
</tr>
<tr>
<td>24.6.01</td>
<td>13</td>
<td>2833</td>
<td>2637</td>
</tr>
<tr>
<td>1.7.01</td>
<td>15</td>
<td>3598</td>
<td>3359</td>
</tr>
<tr>
<td>3.3.02</td>
<td>12</td>
<td>2832</td>
<td>2792</td>
</tr>
<tr>
<td>23.3.02</td>
<td>8</td>
<td>1183</td>
<td>1127</td>
</tr>
<tr>
<td>24.3.02</td>
<td>9</td>
<td>1713</td>
<td>1645</td>
</tr>
<tr>
<td>29.6.03</td>
<td>11</td>
<td>2857</td>
<td>2852</td>
</tr>
<tr>
<td>6.7.03</td>
<td>13</td>
<td>4001</td>
<td>3436</td>
</tr>
<tr>
<td>13.7.03</td>
<td>13</td>
<td>2311</td>
<td>2196</td>
</tr>
<tr>
<td>20.7.03</td>
<td>13</td>
<td>2483</td>
<td>2279</td>
</tr>
<tr>
<td>27.7.03</td>
<td>13</td>
<td>2332</td>
<td>2239</td>
</tr>
<tr>
<td>3.8.03</td>
<td>12</td>
<td>3286</td>
<td>2611</td>
</tr>
<tr>
<td>10.8.03</td>
<td>14</td>
<td>3281</td>
<td>2921</td>
</tr>
<tr>
<td>15.8.03</td>
<td>13</td>
<td>1615</td>
<td>1416</td>
</tr>
<tr>
<td>17.8.03</td>
<td>11</td>
<td>1406</td>
<td>1315</td>
</tr>
<tr>
<td>24.8.03</td>
<td>10</td>
<td>2741</td>
<td>2480</td>
</tr>
<tr>
<td>31.8.03</td>
<td>14</td>
<td>2124</td>
<td>1927</td>
</tr>
<tr>
<td>7.9.03</td>
<td>14</td>
<td>3224</td>
<td>2705</td>
</tr>
</tbody>
</table>

Average deviation (%) 14.61 4.64 3.48 14.39 4.66 3.51

Fig. 3. Number of optima found with different numbers of additional pickups.

runtime observations and the fact that improvements are already quite small with three and four additional pickups. While there may be some further improvements possible by allowing even more additional pickups our results hint that these improvements should be negligible.

Third, let us turn to the performance of the heuristics. In Fig. 5 we have shown the computation time needed by Savings+ for the case with four additional pickups. Even in this case with the largest search space, the algorithm takes on average 1 s. Note that the runtimes for all three heuristics are almost identical. However, at least in the base case the solution quality differed significantly particularly between the sequential Greedy heuristic and the Savings and Savings+ heuristics. In Fig. 6 this comparison is shown again, this time for all five cases. More precisely, for each algorithm and each case Fig. 6 shows the percentage deviation from the optimum in the respective case. These results
Fig. 4. Average percentage deviation from optimum with different numbers of additional pickups.

Fig. 5. Computation times needed to find and prove optimality.

Fig. 6. Performance comparison of heuristics for different numbers of additional pickups.
clearly indicate the performance differences between the three algorithms. Savings+ produces a deviation of less than 4%, while the deviations of Savings range from 4% to 5%. The Greedy approach is not competitive with average deviations of more than 13%.

Besides that, we also observe that the performance of the Greedy heuristic seems to deteriorate with an increasing number of additional pickups, whereas the two Savings do not really show this effect. This is mainly due to the fact that the Greedy heuristic does not gain from large additional savings values, particularly if they relate to combinations to be realized later during the planning horizon day. On the other hand, the performance of the Savings+ heuristic is almost completely unaffected by the increase in the number of additional pickups, thus showing its robustness.

6. Conclusion and outlook

In this paper we have proposed a mixed-integer programming model, three variants of a heuristic solution approach and an optimal branch-and-bound procedure for a real-world scheduling problem faced by the Austrian Red Cross. Particularly, the heuristics are capable of solving the problem in negligible time even if we consider the number of pickups at each customer as decision variables. The branch-and-bound algorithm while finding the optimal solutions quite fast also may need excessive time to prove optimality. Our computational results also indicate that allowing additional pickups at locations may improve solution quality significantly, however, only up to a certain point after which no further improvements are possible.

The model and approaches are currently in a prototype phase, used by the Austrian Red Cross to evaluate their current practice and to calibrate the underlying time and distance information supplied by the GIS. Typical improvements to hand-made solutions range from 6% to 12% depending on the complexity of the problem.

Starting from the approach presented in this paper we currently investigate the integration of a rendezvous option where two vehicles can exchange loads in order to save unnecessary travelling. Apart from that considering the connection of the problem described in this paper with the delivery of blood to hospitals (which is based on stochastic and dynamic requirements by the hospitals) should be an interesting direction for future research as these tasks are currently performed by the same fleet. For these enhanced models we plan to integrate a metaheuristic approach based on artificial ants (e.g. [21,22]).

Acknowledgments

Financial support by Grant #11187 from the Oesterreichische National Bank (OeNB) and from the Ministry of Infrastructure and Technology (BMVIT) is gratefully acknowledged. We are grateful to experts of the Austrian Red Cross such as Markus Hnatek, Maria Jahl, Franz Jelinek, Helmut Kallinger, Michael Niemeck, Gerhart Svoboda, Werner Terkola for providing us with detailed information about the processes of the blood bank. We would also like to thank Karin Leithner and Kerstin Zisser for processing the data and developing a first draft of the model for this problem.

References


