Branch and Bound Algorithms for the Bus Evacuation Problem*

Marc Goerigk, Bob Grün, and Philipp Heßler

Fachbereich Mathematik, Technische Universität Kaiserslautern, Germany

February 20, 2013

Abstract

The Bus Evacuation Problem (BEP) is a vehicle routing problem that arises in emergency planning. It models the evacuation of a region from a set of collection points to a set of capacitated shelters with the help of buses, minimizing the time needed to bring the last person out of the endangered region.

In this work, we describe multiple approaches for finding both lower and upper bounds for the BEP, and apply them in a branch and bound framework. Several node pruning techniques and branching rules are discussed. In computational experiments, we show that solution times of our approach are significantly improved compared to a commercial integer programming solver.

1 Introduction

Recent events like hurricanes over North America (see [Lit06]), or tsunamis in the Indian Ocean remind us that evacuating whole regions may become necessary in case of an emergency; and in such a situation, operations research is able to save both lives and expenses. For a survey on models and challenges in this area of research, see, e.g., [HT01], [AG06], or [YAM08].

In this work, we consider the problem of evacuating an urban region to a set of emergency shelter locations with the help of available public transport infrastructure. In particular, we assume that evacuees that do not travel on their own, be it due to age, sickness, the lack of a private car or any other reason, are gathered at few collection points, where they are brought on buses. The arising optimization problem is to determine a set of bus routes along with their timetable that minimizes the evacuation time, i.e., the time needed for the last evacuee to reach a shelter.

The problem we consider is closely related to the one discussed in [Bis11], where mixed-integer programming models and an iterative local search heuristic

*Partially supported by the Federal Ministry of Education and Research Germany, grant DSS_Evac_Logistic, FKZ 13N12229.

is presented. Making use of public transport in emergency evacuation is also considered in [SE10], incorporating traffic flow dynamics. Recently, [GG12] considered the uncertain bus evacuation problem, where the exact number of evacuees is not available at the beginning of the evacuation planning.

Contributions Naturally, planning in emergency situations has strict computation time limitations, and the usage of sub-optimal heuristics stands to reason. We discuss several approaches to construct feasible solutions to the Bus Evacuation Problem, and to calculate lower bounds on the evacuation time that help the planner to assess the situation.

However, being useful on their own, we show that these upper and lower bounds can be easily integrated into a branch and bound framework that aims at finding an optimal solution. In computational experiments we show that such an algorithm can solve most realistically sized instances in reasonable time; furthermore, the resulting computation times are significantly smaller than when a commercial IP solver is used.

Overview In Section 2 we describe the Bus Evacuation Problem in detail. We then present upper bounds for the problem in Section 3, and lower bounds in Section 4. In Section 5, we discuss branching rules and node reduction techniques. Computational results are presented in Section 6. Finally, Section 7 concludes the paper.

2 Problem Description

In this section we formalize the problem we consider, and model it with the help of a linear integer programming formulation.

The evacuation scenario we consider is the following: A densely populated region needs to be evacuated, and not everybody is able to leave the region on its own. An example for such a situation is the defusal of a bomb within a city center, as is frequently necessary in cities bombed during World War II. Emergency shelters are prepared for the evacuees, and buses of the local transport agency are used.

The collection points $[S] = \{1, \dots, S\}$ where evacuees gather will be referred to as *sources*, and the shelters $[T] = \{1, \dots, T\}$ where they are transported to as *sinks*. We assume that the number of evacuees at every source $i \in [S]$ is known, and given in terms of integer multiples of bus loads denoted by l_i. Furthermore, every sink $j \in [T]$ can only shelter a limited number of evacuees, and is thus given a capacity u_j. We will refer to the number of evacuees and the shelter capacities as *lower* and *upper* bounds, or as *supply* and *demand*. We denote the total number of evacuees with $L = \sum_{i \in [S]} l_i$. All buses start from a depot that is not necessarily a source or a sink. Formally, the bus evacuation problem (BEP) is now given as:

The Bus Evacuation Problem (BEP):

Input: The number of buses B, of sources S, and of sinks T. A matrix $(d_{ij})_{i \in [S], j \in [T]}$ of source-sink-distances, a vector $(d_i^{start})_{i \in [S]}$ of depot-source-distances, a vector $(l_i)_{i \in [S]}$ of numbers of evacuees, and a vector $(u_j)_{j \in [T]}$ of sink capacities.

Find: Find a bus schedule minimizing the maximum travel time over all buses such that all evacuees are transported to the sinks, and sink capacities are respected.

The problem was first described in [Bis11], where both sources and sinks are nodes in a transportation network, and buses are allowed to pick up and drop off amounts of evacuees up to a given bus capacity. Here we consider a slightly simplified problem version; however, all described algorithms can also be applied to the problem definition of [Bis11]. We explain the problem using a small example instance.

Example 1. *Figure 1 illustrates a BEP instance. There are three sources and three sinks given. The sources have a supply of* $l = (1, 3, 3)$*, while the sinks have capacities of* $u = (4, 4, 1)$*. The distance from the depot to the source is given by* $d^{start} = (7, 4, 9)$*, and the distances between* S *and* T *is given by*

$$d = \begin{pmatrix} 6 & 7 & 8 \\ 10 & 9 & 2 \\ 6 & 3 & 7 \end{pmatrix}.$$

The number of buses is $B = 3$*.*

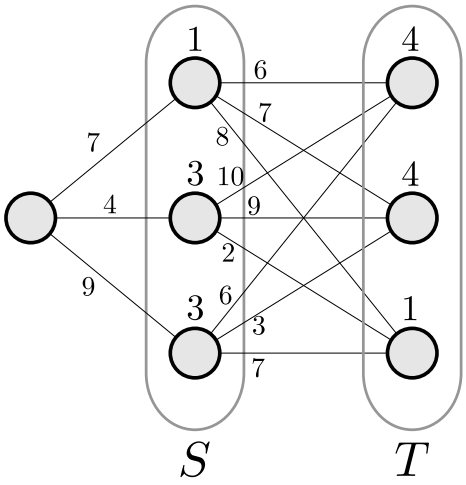


Figure 1: Example BEP instance.

Table 1 represents a feasible solution to the presented instance; in fact, it is even optimal. The first bus travels from source 1 to sink 1, and then from source 3 to sink 2. Its total travel time is thus given by $d_1^{start} + d_{11} + d_{31} + d_{32} = 7 + 6 + 6 + 3 = 22$*. For bus 2 we calculate a travel time of* 23*, and for bus 3 a travel time of* 23 *again, resulting in a total evacuation time of* 23*.*

We shall refer to a pair (i, j) of source and sink node as a *tour*, and to a list of tours as a *tourplan*.

In order to model the problem as a mixed-integer linear program, we fix a maximum number of rounds R the evacuation process might possibly take (a trivial upper bound on R is given by $\sum_{i \in [S]} l_i$). We introduce variables $x_{ij}^{br} \in \mathbb{B}$ that represent whether bus b travels from source i to sink j in round r. The

Trip nr.	1	2	3
Bus 1	$(1,1)$	$(3,2)$	–
Bus 2	$(2,1)$	$(3,2)$	–
Bus 3	$(2,3)$	$(2,2)$	$(3,2)$

Table 1: Feasible solution.

variables t_{to}^{br} and t_{back}^{br} measure the travel time for bus b in round r from the source to the sink, and from the sink to the next source, respectively. Finally, the variable t_{max} denotes the maximum total travel distance over all buses.

$$\min\ t_{max} \tag{1}$$

$$\text{s.t. } t_{max} \geq \sum_{r\in[R]} \left(t_{to}^{br} + t_{back}^{br}\right) + \sum_{i\in[S]}\sum_{j\in[T]} d_i^{start} x_{ij}^{b1} \qquad \forall b \in [B] \tag{2}$$

$$t_{to}^{br} = \sum_{i\in[S]}\sum_{j\in[T]} d_{ij} x_{ij}^{br} \qquad \forall b \in [B], r \in [R] \tag{3}$$

$$t_{back}^{br} \geq d_{ij}\left(\sum_{k\in[S]} x_{kj}^{br} + \sum_{l\in[T]} x_{il}^{b,r+1} - 1\right) \tag{4}$$
$$\forall b \in [B], r \in [R-1], i \in [S], j \in [T]$$

$$\sum_{i\in[S]}\sum_{j\in[T]} x_{ij}^{br} \leq 1 \qquad \forall b \in [B], r \in [R] \tag{5}$$

$$\sum_{i\in[S]}\sum_{j\in[T]} x_{ij}^{br} \geq \sum_{i\in[S]}\sum_{j\in[T]} x_{ij}^{b,r+1} \qquad \forall b \in [B], r \in [R-1] \tag{6}$$

$$\sum_{j\in[T]}\sum_{b\in[B]}\sum_{r\in[R]} x_{ij}^{br} \geq l_i \qquad \forall i \in [S] \tag{7}$$

$$\sum_{i\in[S]}\sum_{b\in[B]}\sum_{r\in[R]} x_{ij}^{br} \leq u_j \qquad \forall j \in [T] \tag{8}$$

$$x_{ij}^{br} \in \mathbb{B} \qquad \forall i \in [S], j \in [T], b \in [B], r \in [R] \tag{9}$$

$$t_{to}^{br}, t_{back}^{br} \in \mathbb{R} \qquad \forall b \in [B], r \in [R] \tag{10}$$

$$t_{max} \in \mathbb{R} \tag{11}$$

Constraint (2) ensures that t_{max} is as large as the maximal travel time of all buses. Constraints (3) and (4) are used to measure the travel time, while Constraint (5) makes sure a bus can only travel from one source to one sink per round. Constraint (6) models that bus tours are connected and can stop whenever they like, while Constraints (7) and (8) ensure that all evacuees are transported and shelter capacities are respected.

It is shown in [GG12] that BEP is NP-complete, even in the case of $d_i^{start} = 0$ and $d_{ij} = d_{i'j}$ for all $i, i' \in S$.

3 Upper Bounds: Algorithms for Constructing Feasible Solutions

We now develop four greedy approaches to construct a feasible solution to the BEP. All algorithms are able to make use of solutions that are partially fixed, which will be of importance for the usage within a branch and bound framework. In particular, we may assume that for each bus $i \in [B]$ we are given a tourplan x_i consisting of a number of sequential tours, beginning from the first tour of the bus (i.e., a partial solution does not have "gaps" in the tourplan). When no solution is given, we can simply set $x_i = \emptyset$ for every bus. Furthermore, we assume that lower and upper bounds l and u are modified according to the given partial solution.

3.1 Algorithms with Precomputed Tourlists

3.1.1 UB 1.

Our first algorithm is based on a greedy distribution of tours to buses. It consists of two parts: First, we generate a set of tours that transport the remaining evacuees to the shelters. Then, we assign these tours to buses.

For each source node i and unit of positive supply l_i, we generate a tour from i to the closest sink with positive capacity u_j, and reduce l_i and u_j by one. We add this tour to a list and repeat the process until $l = 0$, i.e., all evacuees are transported. We then add a randomly chosen tour from the list to one of the buses with smallest total travel time, remove the tour from the list, and repeat until the list is empty. The result is a heuristic feasible solution calculated in polynomial time.

Algorithm 1 describes this procedure in more detail.

Algorithm 1 (UB 1)

Require: An instance of BEP with partial tour plans x_b, $i = b, \dots, B$.

1: $tourlist \leftarrow \emptyset$
2: **for** $i \in [S]$ **do**
3: **while** $l_i > 0$ **do**
4: $j \leftarrow \arg\min\{d_{ij'} : u_{j'} > 0, j' \in [T]\}$
5: $tourlist \leftarrow tourlist + (i, j)$
6: $l_i \leftarrow l_i - 1$
7: $u_j \leftarrow u_j - 1$
8: **end while**
9: **end for**
10: **while** $|tourlist| > 0$ **do**
11: Choose any tour (i, j) at random from $tourlist$.
12: Let $b \in [B]$ be a bus with minimum total travel time.
13: $x_b \leftarrow x_b + (i, j)$
14: Remove (i, j) from $tourlist$.
15: **end while**
16: **return** Feasible solution x.

3.1.2 UB 2.

In our second greedy heuristic, we use a similar approach as before by computing a set of tours in a first step, and then assigning the tours to buses in a second step. We sort the tours (i, j) by non-decreasing cost. Then we start with the cheapest tour and check if the supply l_i and the capacity u_j are positive. In this case we add the tour to our set of tours until the supply or the capacity reaches 0. After all tours have been considered we have a set of tours that we are going to assign to the available buses in the next step.

For the assignment we start with the tour that has been added to the set last, i.e., the tour with the highest cost. We assign this tour to the first available bus and continue with the next expensive tour. This way of assignment is inspired by the longest-processing-time-first rule (LPT rule) known in scheduling theory (see, e.g., [Pin08]) and tries to accomplish a balanced load on all buses.

Algorithm 2 (UB 2)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

$tourlist \leftarrow \emptyset$
while $l \neq 0$ **do**
  $(i, j) \leftarrow \arg\min\{d_{i'j'} : l_{i'} > 0, u_{j'} > 0, i' \in [S], j' \in [T]\}$
  $times \leftarrow \min\{l_i, u_j\}$
  Add $times$ tours (i, j) to $tourlist$
  $l_i \leftarrow l_i - times$
  $u_j \leftarrow u_j - times$
end while
while $|tourlist| > 0$ **do**
  Choose the last tour (i, j) from $tourlist$.
  Let $b \in [B]$ be a bus with minimum total travel time.
  $x_b \leftarrow x_b + (i, j)$
  Remove (i, j) from $tourlist$.
end while
return Feasible solution x.

3.1.3 UB 3.

Observe that after the last tour we do not need to travel back. Therefore, it might pay off to assign a long tour to the last round of a bus, because the back tour will most likely also be long.

Inspired by this fact we modify the upper bound presented in section 3.1.2 by reversing the added tours for each bus in the end. Note that this is not guaranteed to improve the solution since we cannot be sure that a long tour will result in a long back tour.

3.2 An Iterative Algorithm: UB 4

The fourth upper bound is also a greedy approach – however, instead of computing a set of tours in the first step, and then assigning these tours to buses, we generate tours on-the-fly. In each step, we consider the best possibility to bring one evacuee from a sink to a source. In order to simplify the description, offset_b

Algorithm 3 (UB 3)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

1: Run Algorithm 2 to obtain feasible solution x.
2: Reverse the set of tours in x_b that have been added during the run of Algorithm 2 for each $b \in [B]$.
3: **return** Feasible solution x.

denotes the distance of bus $b \in [B]$ which is already planned and t_i^b denotes the distance from the current position of bus b to the source $i \in \mathcal{S}$. In this case $\mathcal{S} = \{i \in [S] : l_i > 0\}$ denotes the available sources and $\mathcal{T} = \{j \in [T] : u_j > 0\}$ denotes the available sinks.

Since all the buses are in the depot at the beginning, we initialize offset_b by 0 and t_i^b by d_i^{start}. At the beginning, the best possibility is the minimum distance of a bus $b \in [B]$ from the depot over a source $i \in \mathcal{S}$ to a sink $t \in \mathcal{T}$. This is attained by the minimum of $t_i^b + d_{ij}$. Therefore, we add the tour (i, j) to the tourlist of bus b and increase offset_b by $t_i^b + d_{ij}$. If we reduce the number of evacuees l_i and u_j by 1 and update t_i^b, we can compute the next step in a similar way. Algorithm 4 describes the idea in more detail.

Algorithm 4 (UB 4)

Require: An instance of BEP with partial tour plans x_b, offset_b and t_i^b, $i = 1, \ldots, S$, $b = 1, \ldots, B$

1: $\mathcal{S} = \{i \in [S] : l_i > 0\}$
2: $\mathcal{T} = \{j \in [T] : u_j > 0\}$
3: **while** $|\mathcal{S}| > 0$ **do**
4: $\quad (i', j', b') \leftarrow \arg\min\{\mathit{offset}_b + t_i^b + d_{ij} : i \in \mathcal{S}, j \in \mathcal{T}, b \in [B]\}$
5: $\quad l_{i'} \leftarrow l_{i'} - 1$
6: $\quad$ **if** $l_{i'} = 0$ **then**
7: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \setminus \{i'\}$
8: $\quad$ **end if**
9: $\quad u_{j'} \leftarrow u_{j'} - 1$
10: $\quad$ **if** $u_{j'} = 0$ **then**
11: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \{j'\}$
12: $\quad$ **end if**
13: $\quad \mathit{offset}_{b'} \leftarrow \mathit{offset}_{b'} + t_{i'}^{b'} + d_{i'j'}$
14: $\quad t_i^{b'} = d_{ij'}\ \forall i \in [S]$
15: $\quad x_{b'} \leftarrow x_{b'} + (i', j')$
16: **end while**
17: **return** Feasible solution x.

4 Lower Bounds

In the following, we present three lower bounds on the evacuation time of an instance of BEP: The first one underestimates the travel time for every necessary trip, the second one is based on a network flow formulation, and the third one a simplification of the model formulation. As is the case for the upper bounds, all algorithms can be computed in polynomial time, and can be applied if a partial

solution is given. As before, we assume that l_i denotes the residual supply at sources in the current solution, u_j the residual capacity, and x_b, $b \in [B]$ denotes the partial tour plans. Furthermore, let offset_b denote the travel time in the current solution for bus $b \in [B]$.

4.1 LB 1

This approach is an adapted and slightly improved version of LB2 in [GG12]. We estimate the total travel time from sources to sinks (todistance), and the total travel time from sinks to sources (backdistance) separately.

Algorithm 5 presents the details for estimating the backdistance. We proceed as follows: We assume that for every residual supply at the sources $i \in [S]$, a bus will enter this source from the sink with the smallest distance. However, those buses that do not yet have any tours assigned, may also enter a source by an arc from the depot. Thus, we assume that we may substitute up to C trips, where C is the number of buses without any tours, with the minimum distance from the depot to the sources.

Algorithm 5 (Backdistance estimate)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

$backlist \leftarrow \emptyset$
$lb \leftarrow 0$
for $i \in [S]$ **do**
  $mindist \leftarrow \min_{j \in [T]} d_{ij}$
  for $k \in 1, \ldots, l_i$ **do**
    $backlist$.pushback($mindist$)
  end for
end for
sort $backlist$
$minstart \leftarrow \min_{i \in [S]} d_i^{start}$
for $b \in [B]$ **do**
  if $x_b = \emptyset$ and $minstart < backlist$.end **then**
    $lb \leftarrow lb + minstart$
    $backlist$.popback()
  end if
end for
for $v \in backlist$ **do**
  $lb \leftarrow lb + v$
end for
return lb

Algorithm 6 estimates the todistance, which we calculate for every source node separately. We assume that the evacuees at each source node can be sent to the closest sink node, respecting capacities.

Finally, Algorithm 7 combines these two estimates to a lower bound on the residual problem. We assume that the residual travel time can be distributed equally over all buses, taking the current travel times into account.

Algorithm 6 (Todistance estimate)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

$lb \leftarrow 0$
for $i \in [S]$ **do**
  $tolist \leftarrow \emptyset$
  for $j \in [T]$ **do**
    $tolist$.pushback((d_{ij}, u_j))
  end for
  sort $tolist$ lexicographically
  while $l_i > 0$ **do**
    $l_i \leftarrow l_i - 1$
    while $tolist$.front.second $= 0$ **do**
      $tolist$.popfront()
    end while
    $lb \leftarrow lb +$ $tolist$.front.first
    $tolist$.front.second $\leftarrow$ $tolist$.front.second - 1
  end while
end for
return lb

Algorithm 7 (LB 1)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

$lb_1 \leftarrow$ output of Algorithm 5
$lb_2 \leftarrow$ output of Algorithm 6
$lb \leftarrow lb_1 + lb_2$
$maxoffset \leftarrow \max_{b \in [B]} offset_b$
for $b \in [B]$ **do**
  $lb \leftarrow lb - maxoffset + offset_b$
end for
if $lb \leq 0$ **then**
  return $maxoffset$
end if
$lb \leftarrow \lceil \frac{lb}{B} \rceil$
return $lb + maxoffset$

4.2 LB 2

A lower bound for the maximum travel time is the average travel time. Therefore, we will have a look at a version of the BEP where the objective is to minimize the sum of the travel times. If we have a solution to this problem and divide the value by the number of buses we obtain a lower bound on the original BEP. An IP formulation for this problem can easily be constructed from problem formulation (1) - (11) by replacing (1) and (2) by

$$\min \sum_{b\in[B]} \sum_{r\in[R]} \left(t^{br}_{to} + t^{br}_{back}\right) + \sum_{i\in[S]} \sum_{j\in[T]} d^{start}_i x^{b1}_{ij} \tag{12}$$

Fortunately, a relaxation of this problem can be written as a pure minimum cost flow problem. For a given BEP instance, we construct a directed graph $G_{ave} = (V, E)$. The graph contains the following nodes:

- For every source $s \in [S]$, two nodes $v^{to}_{S,s}$ and $v^{back}_{S,s}$,
- for every sink $t \in [T]$, two nodes $v^{to}_{T,t}$ and $v^{back}_{T,t}$,
- a start depot node v^{start}_d, and
- an end depot node v^{end}_d.

The edge set E is partitioned into 5 subsets: E^{to}, E^{hold}, E^{back}, E^{start}, and E^{end} with the following edges:

- For every pair $(i, j) \in [S] \times [T]$ of source and sink nodes, E^{to} contains an edge $(v^{to}_{S,i}, v^{to}_{T,j})$ and E^{back} contains an edge $(v^{back}_{T,j}, v^{back}_{S,i})$ both with infinite capacity and cost d_{ij}.
- For every sink $j \in [T]$, E^{hold} contains an edge $(v^{to}_{T,j}, v^{back}_{T,j})$ with capacity u_j and cost 0.
- For every source $i \in [S]$, E^{start} contains an edge $(v^{start}_d, v^{back}_{S,i})$ with infinite capacity and cost d^{start}_i.
- For every sink $j \in [T]$, E^{end} contains an edge $(v^{back}_{T,j}, v^{end}_d)$ with infinite capacity and cost 0.

The supply of nodes $v^{to}_{S,i} \in V, i \in S$, and the demand of nodes $v^{back}_{S,i} \in V, i \in S$, are both l_i. The supply of node $v^{start}_d \in V$ and the demand of node v^{end}_d are both B.

Example 2. *The resulting graph for our example is shown in Figure 2. If we solve the minimum cost flow we obtain the flow drawn in gray with a minimal cost of 62. Since we have 3 buses the average bus travel time would be $\frac{62}{3}$ which gives a lower bound of 21 since the objective value must be integer.*

If we solve a minimum cost flow problem in this graph and divide the obtained cost by the number of available buses we get a lower bound on BEP.

However, note that we do not solve the relaxed problem as stated above: We do not exclude subtours. Imagine the following situation: There are two sources

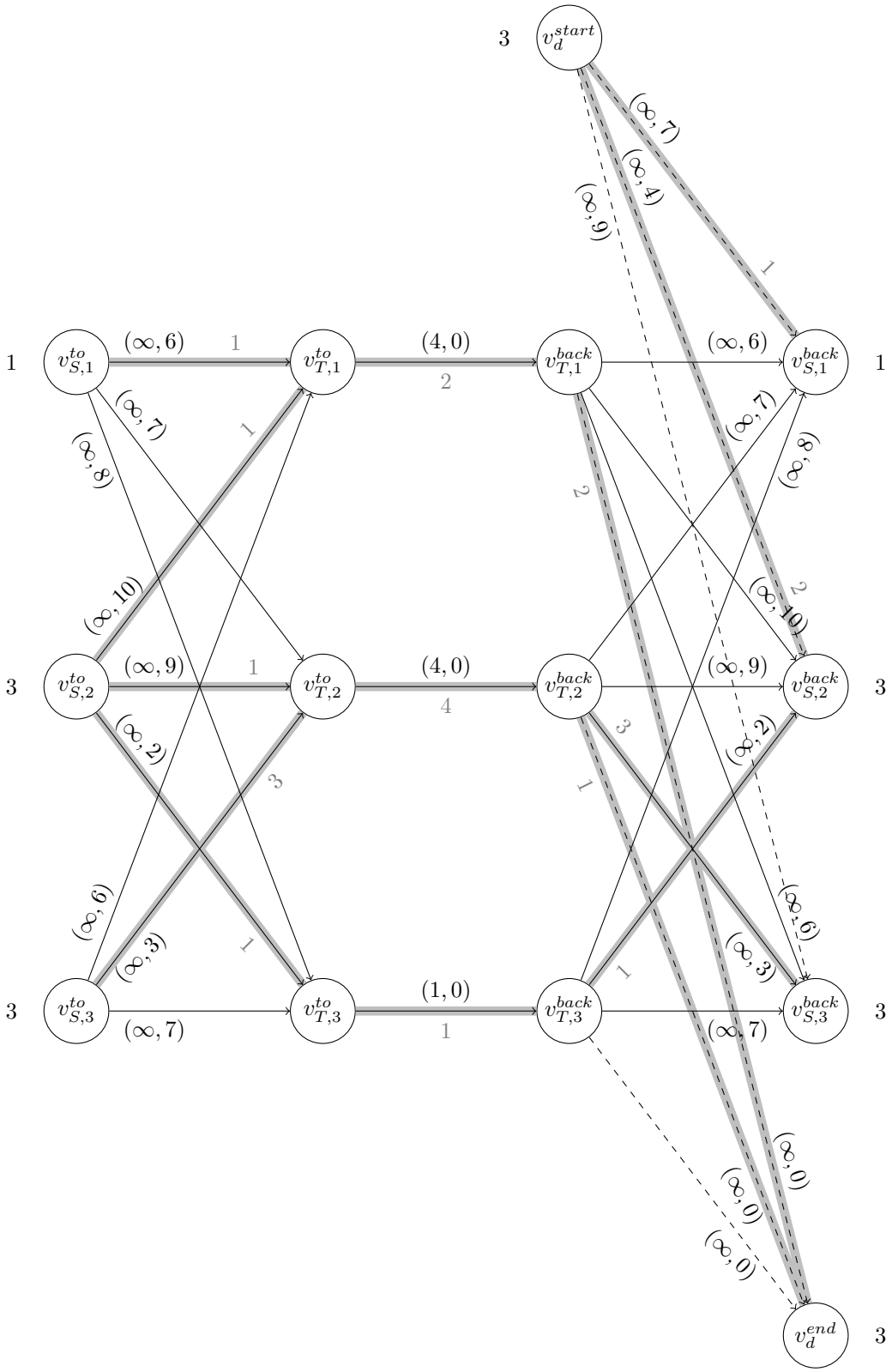


Figure 2: Example for an Average Bus Time Network

with one evacuee each and two sinks with capacity 1. Travelling from source i to sink i is cheap but travelling from source i to sink j with $i \neq j$ is expensive. Furthermore we are only given a single bus and the start distances are low for the first source and high for the second source. The optimal flow will send one unit of flow from the start depot to the first source ($v_d^{start} - v_{S,1}^{back}$), one unit from the first source to the first sink to the end depot ($v_{S,1}^{to} - v_{T,1}^{to} - v_{T,1}^{back} - v_d^{end}$), and one unit from the second source to the second sink to the second source ($v_{S,2}^{to} - v_{T,2}^{to} - v_{T,2}^{back} - v_{S,2}^{back}$). Obviously this is not a valid solution to the BEP no matter what the objective function looks like.

In the branch and bound algorithm we will have to deal with constrained instances where some tours are already assigned to buses. Assume bus b is requested to travel from source i to sink j in its first round. Then we decrease the supply of v_d^{start} and $v_{S,i}^{to}$, the demand of $v_{S,i}^{back}$ and the capacity of $(v_{T,j}^{to}, v_{T,j}^{back})$ by one and increase the supply at $v_{T,j}^{back}$ by one, i.e., we reduce the number of people at source i and the capacity at sink j, and force one bus to start at sink j instead of the depot.

For a given partial solution where most of the tours are already fixed, we may be able to improve this lower bound even further. Assume there are only few people left at the sources. Then the above version would balance the total time for these few people among all buses. This bound might be small; imagine only two people being left in a scenario with 5 buses. Then every bus would do approximately 0.4 trips (ignoring the fixed tours). A better lower bound in this case would be two buses doing one trip each. To approach this idea we introduce a threshold for each bus: $thres_b$ shall denote the minimal time that bus b would travel, if he continues travelling at all. This threshold can be computed by taking the time that bus b has travelled so far and add the minimal time from its current location (a sink or the depot) to a source with people waiting to a sink with positive residual capacity. If a total evacuation time of less than $thres_b$ should be achieved, bus b must not continue after the so far fixed tours.

For each bus b we compute those buses that have a threshold of at most $thres_b$. We distribute the remaining time computed by the above minimum cost flow formulation among these buses such that they are all travelling the same amount of time lb_b. We have to ensure that each of these lb_b is greater or equal to $thres_b$ and the offset of all buses. The minimal of the lb_b then yields an improved lower bound. The pseudo code for this computation is shown in Algorithm 8.

4.3 LB 3

In this section we consider an approach which is motivated by the fact that in real world evacuation scenarios, the sources lie in a close environment and the sinks are normally far away. Therefore, the distances between different sources are neglected such that all the sources can be collapsed to a super node S_0 with $l_{S_0} = \sum_{i \in [S]} l_i$. The distance between the super node S_0 and the sinks $j \in [T]$ will be denoted by $d_j := \min_{i \in [S]} d_{ij}$ and the distance from the depot to the source S_0 is $d^{start} = \min_{i \in [S]} d_i^{start}$. Since the depot-source distance is the same for all buses, this can be neglected in the optimization step.

We can formulate the following IP:

Algorithm 8 (LB 2)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

construct G_{ave}
$t_{rem} \leftarrow$ min cost flow value in G_{ave}
for $b \in [B]$ **do**
  if b currently ends at sink k **then**
    $\mathit{thres}_b \leftarrow \min_{i \in S, j \in T}\{\mathit{offset}_b + d_{ik} + d_{ij}\}$
  else
    $\mathit{thres}_b \leftarrow \min_{i \in S, j \in T}\{\mathit{offset}_b + d_i^{start} + d_{ij}\}$
  end if
end for
for $i \in [B]$ **do**
  $B_{cur} \leftarrow \{b \in B : \mathit{thres}_b \leq \mathit{thres}_i\}$
  distribute t_{rem} among B_{cur} such that a minimal time t_i is achieved
  $t_i \leftarrow \max\{t_i, \mathit{maxoffset}, \mathit{thres}_i\}$
end for
return $\min_{i \in [B]}\{t_i\}$

$$\min\ t_{max} \tag{13}$$

$$\text{s.t. } t_{max} \geq \sum_{j \in T} d_j \left(x_j^b + y_j^b\right) \qquad \forall b \in [B] \tag{14}$$

$$\sum_{b \in [B]} \sum_{j \in [T]} x_j^b \geq l_{S_0} \tag{15}$$

$$\sum_{b \in [B]} x_j^b \leq u_j \qquad \forall j \in [T] \tag{16}$$

$$\sum_{j \in [T]} y_j^b = \sum_{j \in [T]} x_j^b - 1 \qquad \forall j \in [T] \tag{17}$$

$$x_j^b, y_j^b \in \mathbb{N} \qquad \forall b \in [B], j \in [T] \tag{18}$$

$$t_{max} \in \mathbb{R} \tag{19}$$

In this forumlation x_j^b is the number of tours that bus b drives from the super node to sink j and y_j^b is the number of tours that bus b drives back from sink j to the super node.

Since the computation of this IP is too complex as a lower bound, in the computational test, we will relax the variables x_j^b, y_j^b in the way that $x_j^b, y_j^b \in \mathbb{R}$.

Algorithm 9 (LB 3)

Require: An instance of BEP with partial tour plans x_b, $b = 1, \ldots, B$.

Solve the LP relaxation of the program (13)–(19). Let lb be its objective value.
return $\lceil lb \rceil$

5 Branching

Having described algorithms to compute both lower and upper bounds for a partial solution, we explain in the following how to include them in a branch and bound framework. We begin with several branching rules, and proceed with additional node pruning strategies.

5.1 Rules

In the following we present several branching rules. In order to minimize the number of branches we additionally propose some checks on the created branches. In the following a trip (i, j) is feasible if the residual capacity of sink j and the number of people at source i are at least one.

5.1.1 B1: Full Branching.

The first branching rule we consider is the naïve approach of creating one node for each bus, source, and sink with positive residual capacity.

As an example, reconsider the problem instance of Example 1, and assume that the tours $(1, 1)$ for bus 1, $(2, 1)$ for bus 2, and $(2, 3)$ for bus 3 are fixed. As there are two sources and two sinks with positive residual capacity, the partial solution is branched into $2 \cdot 2 \cdot 3 = 12$ new partial solutions, i.e., new branch and bound nodes.

Note that we may generate up to $S \cdot T \cdot B$ new subproblems with each branching step, which may be more than we actually need to compute. As an example, consider the partial solution where bus 1 drives a tour a, and bus 2 drives a tour b. This node can be reached by both fixing the tour a first, and then tour b, or vice versa. To reduce such unnecessary node duplication, we further improve the full branching rule in the following sections.

5.1.2 B2: First Buses First.

The first buses first rule branches those buses with smallest index first, and sets a flag for those buses that do not get branched further. Given a partial tour plan, we find the bus b with minimal index which has not yet been marked as done. For this bus create the following new partial tour plans:

- the old branch and bound node, where b is marked as done, and
- a new branch and bound node for each feasible trip (i, j), $i \in [S]$ and $j \in [T]$.

5.1.3 B3: Minimal Offset Bus First.

The minimal offset bus first rule branches those buses with smallest offset first, and sets a flag for those buses that do not get branched further as before. Given a partial tour plan compute the bus b with minimal offset which has not yet been marked as done. The offset is the time that a bus needs for travelling the already fixed tours. As is the case for rule B2, we create the following new partial tour plans:

- the old branch and bound node, where b is marked as done, and

- a new branch and bound node for each feasible trip (i, j), $i \in [S]$ and $j \in [T]$.

5.2 Tree Reduction

5.2.1 TR1: Lexicographic Pruning.

During a general branch and bound, equivalent branches would be created several times due to the problem symmetry in the buses. To circumvent this problem we suggest to do a check for lexicographic bus assignment. The tourplans x_{b_1} and x_{b_2} of two buses b_1 and b_2 are lexicographically assigned ($x_{b_1} \leq x_{b_2}$) if the following holds:

- The tourplans for b_1 and b_2 are identical, or
- the tourplans for b_1 and b_2 are identical for the first $r-1$ rounds and for round r it holds that
  - Either x_{b_1} is not defined for round r, or
  - the source index for round r and bus b_1 is smaller than the source index for round r and bus b_2, or
  - the source indices for round r and bus b_1 and b_2 are equal and the sink index for round r and bus b_1 is smaller than the sink index for round r and bus b_2.

If in a branch there exist $b_i \leq b_j$ with $b_i, b_j \in [B]$ for which $x_{b_i} \leq x_{b_j}$ does not hold, we discard the branch.

5.2.2 TR2: Subtour Pruning.

In some optimal solutions there is only one tour – the critical tour – that actually takes the maximal evacuation time and most others take shorter time. Then there might be several solutions with this optimal evacuation time but different assignments of the tours that are not part of the critical tour. To avoid checking all of these solutions yielding the same objective value, we would like to enforce the tours of each bus to be assigned in an optimal way. We say that the tours of a bus are assigned in an optimal way if there is no other assignment of tours to this bus that visits the same sources and the same sinks and ends at the same sink as the original assignment which takes less time.

This means that we need to solve a restricted BEP with only one bus. For the sake of runtime, we only compute a heuristic solution based on one of the above upper bounds, instead of checking if the subtours are optimal. If the given subtour takes longer than the heuristic solution we discard the branch.

We use UB4 as a basis for the heuristic solution. First we run the upper bound algorithm on a BEP instance with a single bus where the demand and supply of the sources and sinks is given by the number of visits by the given subtour. If the computed solution does not end at the correct sink, we move the last tour that ends at the correct sink to the end of the assigned tours. Finally, we compare the time needed for the given subtour with the time needed for the computed subtour and discard the branch if the computed subtour takes less time.

6 Experiments

In this section we analyze the numerical performance of the lower bounds, upper bounds, branching rules, and tree reduction checks presented in this paper. We give an overview of these algorithms in Table 2, where TR0 corresponds to applying no tree reduction rule.

LBs		UBs		Bs		TRs	
Nr.	Sec.	Nr.	Sec.	Nr.	Sec.	Nr.	Sec.
1	4.1	1	3.1.1	1	5.1.1	0	-
2	4.2	2	3.1.2	2	5.1.2	1	5.2.1
3	4.3	3	3.1.3	3	5.1.3	2	5.2.2
		4	3.2				

Table 2: Algorithm overview.

As comparing all possible combinations would result in $3 \cdot 4 \cdot 3 \cdot 3 = 108$ different branch and bound algorithms, we need to proceed sequentially: We will compare only the lower bounds in Experiment 1, only the upper bounds in Experiment 2, and the branching and tree reduction rules in Experiment 3. We present our results as plots for the sake of intuition; tables showing detailed results can be found in the appendix.

Environment All experiments were conducted on a compute server with a 16-core Intel Xeon E5-2670 processor, running at 2.60 GHz (up to 3.3 GHz with with turbo boost) with 20MB cache, 32 GB RAM and Ubuntu 12.04. We wrote our code in C++ using gcc v. 4.5.4. with compile flag -O3, and used the commercial MIP solver CPLEX v. 12.4 ([ILO12]). with OPLRUN for comparison. For the minimum cost flow computations we used the network simplex implemented by the LEMON library ([COI12]). All algorithms, including CPLEX, were pinned to one core.

Datasets We generated 9 instance sets of varying size, which consisted of 10 instances each; in total, 90 instances were used. Table 3 gives an overview of the respective values for the number of sources S, the number of sinks T, and the number of buses B. Values d_{ij}, d_i^{start}, and u_j where drawn randomly from $\{1, \ldots, 10\}$, and l_i from $\{1, \ldots, 5\}$. Infeasible instances were dropped and regenerated.

6.1 Experiment 1: Lower Bounds

Setup In our first experiment, we are interested in analyzing the quality-time-tradeoff of the lower bounds presented in Section 4. We use the following setup:

1. We solve each instance three times, using either LB 1, LB 2, or LB 3 as lower bound.

2. As an upper bound we use UB 3. Furthermore, we use B 3 as branching rule and apply both TR 1 and TR 2 for tree reduction.

set name	S	T	B
$\mathcal{I}_2$	2	2	2
$\mathcal{I}_3$	3	3	2
$\mathcal{I}_4$	4	4	3
$\mathcal{I}_5$	5	5	3
$\mathcal{I}_6$	6	6	4
$\mathcal{I}_7$	7	7	4
$\mathcal{I}_8$	8	8	5
$\mathcal{I}_9$	9	9	5
$\mathcal{I}_{10}$	10	10	6

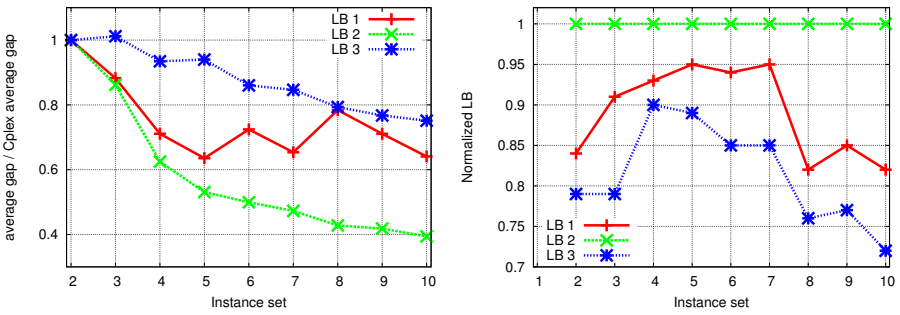
Table 3: Instance sizes.

3. We impose a timelimit of 15 minutes per instance and algorithm, and a memory limit of 10 GB. If either is reached before the instance is solved to optimality, the algorithm is aborted. We measure the resulting gap UB/LB for each instance (note that we do not subtract 1).

4. Additionally, we solve each instance computing all three lower bounds, and in each step using the best of these three. We measure the relative quality of the lower bounds, i.e., we compute $LB_i / \max_{j=1,2,3} LB_j$ for $i = 1, 2, 3$ in each iteration.

5. We use CPLEX to solve each instance with the same time- and memory limitations as before, and measure the resulting gap.

Results We present the average gap over the instance sets $\mathcal{I}_1$ to $\mathcal{I}_{10}$, divided by the average gap of CPLEX, in Figure 3(a). Note that the lower the resulting value, the better performs the respective lower bound compared to CPLEX.



(a) Average gap divide by CPLEX average gap. (b) Average normalized quality of bounds.

Figure 3: Experiment 1: Evaluation of lower bounds.

We find that all three approaches clearly outperform CPLEX, where LB 2 turns out to be the best choice, with a gap that is on average only 40% of CPLEX' gap on the instance set $\mathcal{I}_{10}$. Next comes LB 1, and LB 3 shows the worst performance. The average normalized values show a similar behavior, as can be seen in Figure 3(b): The value of LB 2 is the largest on average, with

LB 1 and LB 3 following behind. Note the performance gap for instances of size 8 and larger for LB 2. The reason for this behavior is the increased memory consumption, which leads to an early abortion of the algorithm.

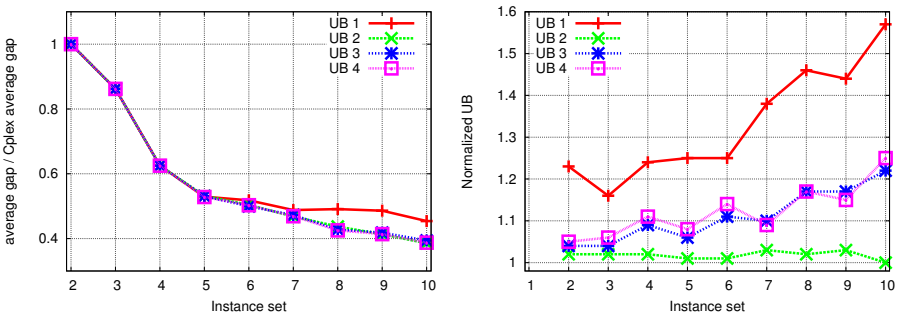
As a note on the comparison to CPLEX, we chose the definition of gap to be UB/LB instead of $UB/LB - 1$ for an easier normalization. However, in the "classic" definition of a branch and bound gap, LB 2 has an average gap of 18% on the instances $\mathcal{I}_{10}$, while CPLEX has a gap of 200% (see appendix).

6.2 Experiment 2: Upper Bounds

Setup In our second experiment, we consider the differences between the upper bounds presented in Section 3. We use the following setup, similar to the previous experiment:

1. We solve each instance four times, using either UB 1, UB 2, UB 3, or UB 4.
2. As a lower bound we use LB 2. As before, we use the branching rule B 3 and both TR 1 and TR 2 for tree reduction.
3. We impose a timelimit of 15 minutes per instance and algorithm, and a memory limit of 10 GB.
4. We solve each instance again, computing all 4 upper bounds, and in each step using the best of these four. We measure the relative quality of the upper bounds, i.e., we compute $UB_i / \max_{j=1,2,3,4} UB_j$ for $i = 1, 2, 3, 4$ in each iteration.

Results We present the average normalized gap in Figure 4(a), and the normalized bound quality in Figure 4(b). Even though the average gap is similar for all four approaches, we see that the quality of UB 1 is worse than that of the other three, and its relatively good performance is due to its fast computability. However, this becomes increasingly disadvantageous for larger instances, where the size of the branch and bound tree forces an early abort. Furthermore, UB 2 tends to yield the strongest bounds.



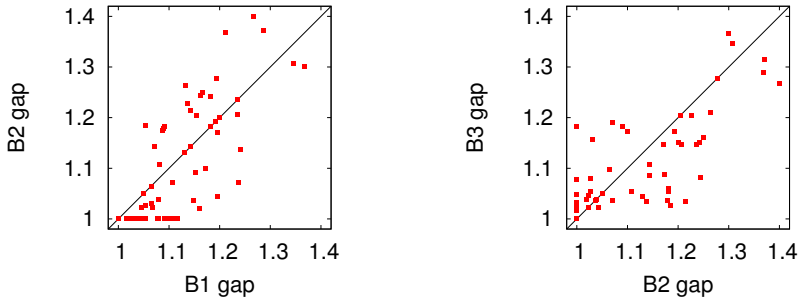
(a) Average gap divide by CPLEX average gap. (b) Average normalized quality of bounds.

Figure 4: Experiment 2: Evaluation of upper bounds.

6.3 Experiment 3: Branching Rules

Setup In our final experiment, we compare both the branching rules and the tree reduction approaches. We solve each instance using branching rules B 1, B 2 and B 3, with the same time- and memory limits as before, and UB 3, LB 2 and both TR 1 and TR 2 for tree reduction. Furthermore, we solve each instance using the tree reductions TR 0, TR 1 and TR 2, using branching rule B 3.

Results For each of the 90 instances, we present the direct comparison of the resulting gaps of B1 and B2, as well as B2 and B3, in Figures 5(a) and 5(b), respectively. A point below the diagonal line means that the algorithm on the vertical scale performs better than the algorithm on the horizontal scale. We find that B 2 is able to find more optimal solutions (i.e., a gap of 1) than both B 1 and B 3, and clearly outperforms B 1. However, on average, B 3 results in a better gap than B 2.



(a) B1-B2 gap.

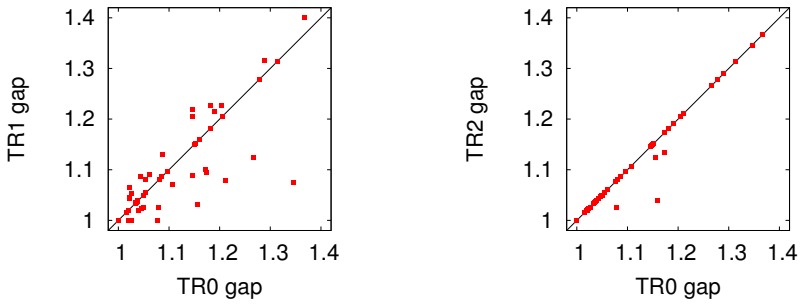
(b) B2-B3 gap.

Figure 5: Experiment 3: Comparison of branching rules.

This comparison is not so distinct for the tree reduction rules, see Figures 6(a) and 6(b). While TR 2 seldom has an impact on the resulting gap, it does not worsen it; TR 1 on the other hand has a larger impact on the solution quality, but sometimes for the better, sometimes for the worst. We are not aware of any instance property that would predict if it pays off to use TR 1.

7 Conclusion and Further Research

We considered the bus evacuation problem, which optimizes the transport of transit-dependent people with the help of public transport infrastructure in the case of an emergency. We presented four greedy algorithms to construct a feasible solution, and three algorithms to find lower bounds. We discussed how to use them in a branch and bound framework, and described additional branching and pruning rules. In an extensive computational study, we found that our algorithms can improve the branch and bound gap from 200% for CPLEX to about 20% over the same time horizon. Hence we hope that the presented algorithms will bring operations research methodology one step closer to practical applicability in disaster management.



(a) TR0-TR1 gap.

(b) TR0-TR2 gap.

Figure 6: Experiment 3: Comparison of tree reduction rules.

Future research includes the extension of the bus evacuation problem by further planning problems arising during the evacuation process; in particular, the logistics surrounding the provisioning of evacuees should be taken into account, as well as the location planning of the collection points and shelters.

References

[AG06] Nezih Altay and Walter G. Green III. OR/MS research in disaster operations management. *European Journal of Operational Research*, 175(1):475 – 493, 2006.

[Bis11] D. R. Bish. Planning for a bus-based evacuation. *OR Spectrum*, 33:629–654, 2011.

[COI12] COIN-OR. LEMON Graph Library 1.2.3, 2012. See http://lemon.cs.elte.hu.

[GG12] M. Goerigk and B. Grün. The robust bus evacuation problem. Technical report, Fachbereich Mathematik, Technical University of Kaiserslautern, 2012.

[HT01] H. W. Hamacher and S. A. Tjandra. Mathematical modelling of evacuation problems: a state of the art. In *Pedestrian and Evacuation Dynamics*, pages 227–266. Springer, Berlin, 2001.

[ILO12] ILOG. CPLEX 12.4, 2012. See http://www.cplex.com.

[Lit06] T. Litman. Lessons from katrina and rita: What major disasters can teach transportation planners. *Journal of Transportation Engineering*, 132(1):11–18, 2006.

[Pin08] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008.

[SE10] Fatemeh Sayyady and Sandra D. Eksioglu. Optimizing the use of public transit system during no-notice evacuation of urban areas. *Computers & Industrial Engineering*, 59(4):488 – 495, 2010.

[YAM08] M. Yusoff, J. Ariffin, and A. Mohamed. Optimization approaches for macroscopic emergency evacuation planning: A survey. In *Information Technology, ITSim, International Symposium, IEEE*, pages 1–7, 2008.

A Detailed results of Experiment 1

$\mathcal{I}$	LB 1	LB 2	LB 3	CPLEX
2	1.00	1.00	1.00	1.00
3	1.02	1.00	1.17	1.16
4	1.14	1.00	1.50	1.60
5	1.21	1.01	1.79	1.90
6	1.49	1.03	1.77	2.06
7	1.47	1.06	1.90	2.25
8	1.99	1.09	2.01	2.54
9	1.98	1.16	2.13	2.78
10	1.92	1.18	2.25	2.99

Table 4: Experiment 1: Average gap.

$\mathcal{I}$	LB 1			LB 2			LB 3		
	OPT	TIME	MEM	OPT	TIME	MEM	OPT	TIME	MEM
2	10	0	0	10	0	0	10	0	0
3	8	0	2	10	0	0	5	5	0
4	3	1	6	10	0	0	0	10	0
5	0	2	8	7	2	1	0	10	0
6	0	0	10	3	5	2	0	10	0
7	0	1	9	1	5	4	0	10	0
8	0	0	10	0	1	9	0	10	0
9	0	0	10	0	2	8	0	10	0
10	0	0	10	0	2	8	0	10	0

Table 5: Experiment 1: Distribution.

B Detailed results of Experiment 2

$\mathcal{I}$	UB 1	UB 2	UB 3	UB 4	CPLEX
2	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.16
4	1.00	1.00	1.00	1.00	1.60
5	1.01	1.01	1.01	1.00	1.90
6	1.07	1.04	1.03	1.04	2.06
7	1.10	1.05	1.06	1.06	2.25
8	1.25	1.11	1.09	1.08	2.54
9	1.35	1.15	1.16	1.15	2.78
10	1.36	1.15	1.18	1.16	2.99

Table 6: Experiment 2: Average gap.

$\mathcal{I}$	UB 1			UB 2			UB 3			UB 4		
	OPT	TIME	MEM	OPT	TIME	MEM	OPT	TIME	MEM	OPT	TIME	MEM
2	10	0	0	10	0	0	10	0	0	10	0	0
3	10	0	0	10	0	0	10	0	0	10	0	0
4	10	0	0	10	0	0	10	0	0	10	0	0
5	8	1	1	8	1	1	7	2	1	8	2	0
6	1	3	6	2	5	3	3	5	2	4	3	3
7	2	4	4	2	7	1	1	5	4	2	5	3
8	0	1	9	0	0	10	0	1	9	0	2	8
9	0	0	10	0	4	6	0	2	8	0	1	9
10	0	0	10	0	2	8	0	3	7	0	1	9

Table 7: Experiment 2: Distribution.

C Detailed results of Experiment 3

$\mathcal{I}$	B 1	B 2	B 3	CPLEX
2	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.16
4	1.03	1.00	1.00	1.60
5	1.04	1.00	1.01	1.90
6	1.07	1.04	1.03	2.06
7	1.12	1.09	1.06	2.25
8	1.17	1.17	1.09	2.54
9	1.21	1.19	1.16	2.78
10	1.26	1.27	1.18	2.99

Table 8: Experiment 3: Average gap.

$\mathcal{I}$	B 1			B 2			B 3		
	OPT	TIME	MEM	OPT	TIME	MEM	OPT	TIME	MEM
2	10	0	0	10	0	0	10	0	0
3	10	0	0	10	0	0	10	0	0
4	4	4	2	10	0	0	10	0	0
5	1	4	5	9	1	0	7	2	1
6	0	7	3	4	5	1	3	5	2
7	0	2	8	2	5	3	1	5	4
8	0	2	8	0	4	6	0	1	9
9	0	1	9	0	3	7	0	2	8
10	0	5	5	0	2	8	0	3	7

Table 9: Experiment 3: Distribution.

$\mathcal{I}$	TR 0	TR 1	TR 2	CPLEX
2	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.16
4	1.00	1.00	1.00	1.60
5	1.01	1.01	1.01	1.90
6	1.04	1.03	1.04	2.06
7	1.09	1.07	1.07	2.25
8	1.10	1.09	1.10	2.54
9	1.15	1.16	1.15	2.78
10	1.20	1.18	1.20	2.99

Table 10: Experiment 3: Average gap.

$\mathcal{I}$	TR 0			TR 1			TR 2		
	OPT	TIME	MEM	OPT	TIME	MEM	OPT	TIME	MEM
2	10	0	0	10	0	0	10	0	0
3	10	0	0	10	0	0	10	0	0
4	10	0	0	10	0	0	10	0	0
5	6	3	1	7	2	1	6	3	1
6	1	5	4	3	5	2	1	7	2
7	1	5	4	1	5	4	1	6	3
8	0	3	7	0	1	9	0	3	7
9	0	2	8	0	1	9	0	2	8
10	0	2	8	0	3	7	0	2	8

Table 11: Experiment 3: Distribution.