Routing by Ranking: A Link Analysis Method for the Constrained Dial-A-Ride Problem

Lauri Häme*,[1]

Aalto University School of Science, PL 14100, 00076 Aalto, Finland

Harri Hakula

Aalto University School of Science, PL 14100, 00076 Aalto, Finland

Abstract

Using a modified version of hyperlink-induced topic search (HITS), we characterize hubs as nodes in a directed graph with many out-links to other hubs and calculate a hub score for each node. Ranking the nodes by hub score gives guidance to a dynamic programming algorithm for efficiently finding feasible solutions to the dial-a-ride problem.

Key words: Dial-a-Ride Problem, Link Analysis, HITS, Topological Ordering

Introduction

Several web information retrieval (IR) methods have been developed for finding the most appropriate web pages corresponding to queries given to search engines. The most sophisticated methods, such as HITS [1], PageRank [2] and SALSA [3] in use today make use of the hyperlinked structure of the web, since the goodness of a web page and the position of the page with respect to other web pages seem to have a certain connection. For example, a web page may be considered good if there are many other web pages linking to that page. In other words, web pages are ranked by search engines not only by means of the content of the page, but also by exploiting information regarding the hyperlink-induced relationships between pages.

The bringing of hyperlinks to bear on the ordering of web pages has given rise to a mathematical analysis related to hyperlink-induced web IR methods, such as in [4–7], in which the behaviour of several IR methods is studied from the computational point of view.

In this work, we focus on the HITS (Hyperlink-Induced Topic Search) algorithm, which defines *authorities* (web pages with several inlinks) and *hubs* (several outlinks). The HITS thesis is that *good hubs point to good authorities and good authorities are pointed to by good hubs*. Based on this thesis, HITS assigns both a hub score and authority score to each web page [5]. In this paper, we use the term ”hub” similarly as in the above context and not in the context of travel dispatch centers.

We present an application of HITS on the dial-a-ride problem (DARP) [8–20], in which the goal is to construct a set of vehicle routes, serving a set of customers, satisfying the given time, capacity and precedence constraints. We examine the DARP as a *constraint satisfaction problem*, in which the goal is to find a set of m feasible vehicle routes that serve all customers, where m is the number of vehicles, or to prove that such a set of routes does not exist, as in [20]. In this reference, it is noted that an algorithm for checking the feasibility of a DARP instance has two main applications: 1) Determining the feasibility can be the first phase in an optimization algorithm in a *static* setting, where all trip requests are known, for example, one day in advance. 2) In *dynamic* services, a constraint satisfaction algorithm can be used for deciding whether to accept or reject incoming user requests. In contrast to most works related to vehicle routing and dial-a-ride problems, no specific cost function is considered. For approaches to vehicle routing and dial-a-ride problems with cost functions, we refer to [21–23] and [13, 19].

In the context of the dial-a-ride problem, we define links between nodes as feasible transitions with respect to the constraints of the problem: If node j can be visited after i, a link from i to j is formed. Thus, a good hub score of a pick-up or drop-off node i means that many nodes can be reached in time from i. Thus, in order to efficiently find feasible solutions to the dial-a-ride problem, we suggest that nodes with large hub scores should be visited first since there are many nodes that can be visited after such nodes.

This work is partially motivated by a dynamic demand-responsive transport (DRT) service currently being planned to operate in Helsinki. Helsinki Region Transport board has approved a plan under which the trial period of the service takes place from 2012 to 2014. Similarly as the current service routes [24], the new DRT service is designed to operate on a demand-responsive basis, that is, each trip is booked in advance and the vehicle routes are modified according to the trips. The main difference to existing services is that no pre-order times for trips are required and the trips can be booked ”on the fly” by means of an interactive user interface.

1. The Ranking Method

In the following sections we introduce a ranking method based on the HITS algorithm (Sections 1.1 and 1.2) and show

*Corresponding author

Email addresses: Lauri.Hame@tkk.fi (Lauri Häme), Harri.Hakula@tkk.fi (Harri Hakula)

[1]Tel.: +358 40 576 3585, Fax: +358 9 470 23016

how it can be used to solve the dial-a-ride problem (Section 1.3).

1.1. The HITS algorithm [1]

Given a web graph $G = (V, A)$ consisting of pages V and links A between pages, the authority and hub scores a_i and h_i are computed for each page i as follows. Letting (i, j) represent a link from page i to page j, given that each page has been assigned an initial authority score $a_i(0)$ and hub score $h_i(0)$, HITS successively refines these scores by computing

$$a_i(k) = \sum_{(j,i)\in A} h_j(k-1)$$
$$h_i(k) = \sum_{(i,j)\in A} a_j(k-1)$$

for $k \in \{1, 2, \ldots\}$. By using matrix notation, these equations can be written the form $a(k) = L^T h(k-1)$ and $h(k) = La(k-1)$, where $a(k)$ is the authority vector containing the authority scores of each of the pages at step k, $h(k)$ is the corresponding hub vector and L is the adjacency matrix of the graph with elements $L_{ij} = 1$ if $(i, j) \in A$ and $L = 0$ otherwise [5].

It has been shown in [4] that the authority and hub vectors describing the authority and hub scores of nodes of a given graph are given by the dominant eigenvectors of the matrices $L^T L$ and LL^T (or equivalently, dominant singular vectors of L), where L is the adjacency matrix of the graph.

1.2. Modified HITS

For constrained routing problems, we present a modified version of the HITS algorithm, in which only the hub scores are considered. More precisely, our thesis is that *good hubs point to good hubs*. This formulation is motivated by Theorem 1, which states that for a specific class of graphs, the hub score of a node i corresponds to the number of self-avoiding paths from i to a given destination node. When attempting to construct a path that visits all nodes, or to maximize the number of visited nodes, the modified HITS idea induces the following intuitive policy: Good hubs are visited first, since many nodes can be reached from good hubs.

The hub scores are calculated as follows. Let L denote the adjacency matrix of a directed graph G. Similarly as in the HITS algorithm, the hub vector containing the *hub scores* of nodes is first initialized, $h(0) = (1, 1, \ldots, 1)$ and the hub vector is successively updated by means of the power method

$$h'(k) = Lh(k-1). \tag{1}$$

Similarly as in the original HITS algorithm, the hub vector converges to a dominant eigenvector of L.

Theorem 1 characterizes the hub scores produced by the modified HITS algorithm for *sink graphs* defined as follows.

Definition. *Let $G = (V, A)$ be a directed acyclic graph and let $s \in V$ be a node such that $(s, i) \notin A$ for all $i \in V$. The graph $G_s = (V, A \cup (s, s))$ is called a* sink graph.

In other words, a sink graph is a directed acyclic graph (V, A) with the exception that one node $s \in V$ with zero outdegree is associated with a loop (s, s).

Theorem 1. *Let L denote the adjacency matrix of a sink graph $G_s = (V, A)$, where $V = \{1, \ldots, |V|\}$, let h_i denote the number of self-avoiding paths from i to s for $i \in V \setminus \{s\}$ and let $h_s = 1$. Then, $h = (h_1, \ldots, h_{|V|})^T$ is a unique dominant eigenvector of L.*

Proof. Since the adjacency matrix of a directed acyclic graph is an upper triangular matrix with zeros on the diagonal, the adjacency matrix of a sink graph is an upper triangular matrix with diagonal elements $L_{ii} = 0$ except for the sink node s, for which we have $L_{ss} = 1$. The eigenvalues of an upper triangular matrix are equal to the diagonal elements [25] and thus there exists a unique dominant eigenvalue 1. Let us show that $h = (h_1, \ldots, h_{|V|})^T$ is the corresponding eigenvector of L.

Since all paths in a directed acyclic graph are self-avoiding and by definition we have $h_s = 1$, the number of self-avoiding paths h_i from node i to the sink node s in the sink graph G_s satisfies $h_i = \sum_{j\in V} L_{ij} h_j$ for $i \in V \setminus \{s\}$ and h_s satisfies $h_s = 1 = L_{ss} h_s = \sum_{j\in V} L_{sj} h_j$. In matrix form, we have $h = Lh$ and thus h is the unique eigenvector corresponding to eigenvalue 1. □

Note that since $h_i \leq h_j$ for all $i, j \in V$ for which $L_{ij} = 1$, the vector h defines a *topological ordering* [26] of the nodes for which $h_i > 0$. Although Theorem 1 considers a special class of graphs[2], the result gives us an idea of the behaviour of the modified HITS method: There are many paths beginning from nodes with high hub scores.

In the following section, we show how the hub scores are used to give indicative guidance to a dynamic programming algorithm for the dial-a-ride problem.

1.3. The dial-a-ride problem

The dial-a-ride problem is defined as follows [20]. Let $G = (V, A)$ be a complete and directed graph with node set $V = \{0\} \cup P$, where node 0 represents the depot, and P represents the set of pick-up and drop-off nodes, where $(|P| = 2n)$. The set P is partitioned into sets P^+ (pick-up nodes) and P^- (drop-off nodes). Each arc $(i, j) \in A$ has a non-negative travel time T_{ij}. The travel times satisfy the triangle equation, that is, $T_{ij} + T_{jk} \geq T_{ik}$ for all $i, j, k \in R$. With each node $i \in V$ are associated a time window $[E_i, L_i]$, a service duration D_i and a load q_i, where $D_0 = 0$ and $q_0 = 0$. Let $H = \{1, \ldots, n\}$ be the set of customers and let $T^{\max}$ be the maximum ride time for any customer. With each customer i is associated a pickup node $i^+ \in P^+$, a delivery node $i^- \in P^-$ and a load $q_{i^+} = q_{i^-}$. The above parameters are illustrated in Figure 1.

Let $K = \{1, \ldots, m\}$ be the set of available vehicles, each with capacity Q. A *route* is a circuit over a set of nodes in P, starting and finishing at the depot 0. The goal is to construct m vehicle routes such that: (i) for every customer i, the pick-up node and the drop-off node are visited by the same route and the pick-up node is visited before the drop-off node; (ii) the load of the vehicles does not exceed the capacity Q at any time; (iii) the ride time of each customer is at most $T^{\max}$; (iv) the service at node i begins within the interval $[E_i, L_i]$.

[2] The calculation of self-avoiding paths in graphs with cycles is discussed in [27].

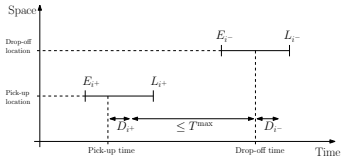

Figure 1: Time pick-up and drop-off time windows. The pick-up point of customer i is denoted by i^+ and the drop-off point is denoted by i^-. The customer should be picked up at i^+ within the time window $[E_{i^+}, L_{i^+}]$ and the customer should be dropped off at i^- within the time window $[E_{i^-}, L_{i^-}]$. The service times needed for the customer to get on the vehicle and get off the vehicle are denoted by D_{i^+} and D_{i^-}. The time between the drop-off and the pick-up (excluding D_{i^+}) should not exceed the maximum ride time $T^{\max}$.

1.3.1. Single-vehicle solution

In this section, we propose an exact constraint programming method for the single-vehicle DARP ($m = 1$) that produces a feasible solution whenever one exists or proves that the problem is infeasible. In the latter case, the algorithm returns a route that maximizes the number of served customers. The main challenge is to find a sequence of pick-up and drop-off nodes for which the time and precedence constraints are satisfied. While capacity constraints are also considered, the focus is on problems, in which time limits are more restrictive. The solution is extended to the multi-vehicle case in Section 1.4.

Letting node $0 \in V$ be the location of the vehicle at $t = 0$, we aim to find a feasible path $(0, p_1, \ldots, p_{2n}, 0)$ consisting of the pick-up and drop-off nodes of n customers. The number of permutations is $(2n)!$ and the number of feasible permutations with respect to precedence constraints is $(2n)!/2^n$ [17].

Briefly, our approach to the problem is a *depth-first* search, in which the remaining nodes are ranked by means of hub scores at each step and the order of the depth-first search is determined by the ranking. The search begins from node 0 by ranking the pick-up and drop-off nodes P of all customers. Then, the node p^* with the highest ranking is added to the sequence and the ranking procedure is repeated for the remaining nodes $P \setminus \{p^*\}$.

Formally, the search is executed by means of the recursion $\mathrm{REC}(S, R_S)$ shown in Algorithm 1, where S denotes a sequence of nodes (initially $S = (0)$), R_S denotes the set of remaining nodes at S (initially $R_S = \{1^+, 1^-, \ldots, n^+, n^-\}$) and $S_{\max}$ denotes the sequence that maximizes the number of served customers (initially $S_{\max} = (0)$). $C(S)$ denotes the number of served customers in sequence S, which is equal to the number of drop-off points in S.

The main idea of the recursion is that at each step, we have the current sequence S and the set of remaining nodes R_S (nodes that can possibly be added to the sequence). At first, we check the time, capacity and precedence constraints of S. If S is feasible and all customers have been served, that is, if $C(S) = n$, a feasible solution is found and the recursion is terminated. Otherwise, the remaining nodes R_S are ranked by hub scores and the infeasible nodes are removed from R_S (see Section 1.3.2). Then, the recursive function $\mathrm{REC}((S, i), R \setminus \{i\})$ is called for all sequences (S, i) in ranked order. In the following, we describe the ranking of remaining nodes in more detail.

```
Check time, capacity and precedence constraints;
if S is infeasible then
    Return;
end if
if C(S) > C(S_max) then
    S_max ← S;                          (Store the current sequence S.)
    if C(S) = n
     Terminate recursion;                (Feasible route.)
    end if
end if
Rank the remaining nodes i ∈ R_S and remove infeasible nodes from R_S;
(See Section 1.3.2.)
for all i ∈ R_S (in ranked order) do
    REC((S, i), R_S \ {i});
end for
```

Algorithm 1: A recursive solution $\mathrm{REC}(S, R_S)$ to the single-vehicle DARP. S denotes a sequence of nodes (initially $S = (0)$), R_S denotes the set of remaining nodes (initially $R_S = \{1^+, 1^-, \ldots, n^+, n^-\}$) and $S_{\max}$ denotes the sequence that maximizes the number of served customers (initially $S_{\max} = (0)$). $C(S)$ denotes the number of served customers in sequence S, which is equal to the number of drop-off points in S.

1.3.2. Ranking and pruning

The ranking of the remaining nodes R_S is determined in two phases: First, the adjacency matrix of the remaining nodes is determined by studying feasible transitions between the nodes. Then, the hub vector is calculated by means of Equation (1).

Adjacency matrix

Given that the vehicle is at the last node s of sequence S, we study which sequences (S, i, j), where $i, j \in R_S$, are feasible with respect to the time and precedence constraints. Let t_s denote the departure time and Q_s denote the load of the vehicle at s, and $t_i = t_s + T_{si}$ denote the arrival time at $i \in R_S$. For clarity, service times are assumed equal to zero. However, the modification of the following equations for non-zero service times is straightforward. In addition, we consider fixed time windows $[E_i, L_i]$ for each node. However, maximum ride time constraints [12] are taken into account by updating the upper bounds L_i of the time windows during run-time as described in [28].

First, all nodes that are seen to be non-reachable from S are removed from the set R_S of remaining nodes.

Definition. *A node $i \in R_S$ is said to be infeasible if there does not exist a feasible sequence $(S, \ldots, i)$ ending at i.*

Clearly, if the time of arrival t_i staisfies $t_i > L_i$, the node i is infeasible. The nodes $i \in I$ for which $t_i > L_i$ are removed from R_S, that is, $R_S \leftarrow R_S \setminus I$. This elimination criteria is similar to the one presented in [29].

For nodes that are not seen to be infeasible, we define feasible transitions as follows.

Definition. *The transition $i \rightarrow j$ from node $i \in R_S$ to node $j \in R_S$ is said to be* feasible*, if* $\max(t_i, E_i) + T_{ij} \leq L_j$.

The above definition is similar to the one studied in [30], in which the set of *admissible arcs* between the pick-up and delivery nodes is constructed as a preprocessing step in the pickup and delivery problem. The set of admissible arcs is made up of arcs which a priori satisfy the precedence, capacity and time constraints of the problem. The difference in our formulation is that we define the feasibility of transitions as a function of the state of the vehicle (see Figure 2).

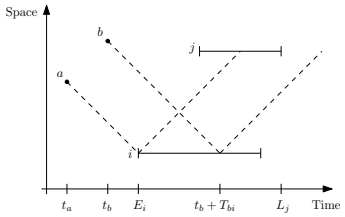

Figure 2: Feasible transitions. The figure shows the locations and time windows of two nodes i and j. If the vehicle left from a at the instant t_a, visited node i and moved directly to node j, the time constraint L_j would be satisfied. However, assuming that at the instant t_b the vehicle is located at b, the transition $i \rightarrow j$ is seen to be infeasible.

The elements of the adjacency matrix are defined for all $i, j \in R_S$ by $L_{ij} = 1$, if $i \neq j$ and $i \rightarrow j$ is feasible, $L_{ij} = 0$ otherwise.

Link analysis

After the adjacency matrix L has been determined, the hub vector $h = (h_1, \ldots, h_{|R_S|})$ of L is determined by Equation (1) (here the remaining nodes $i \in R_S$ are numbered from 1 to $|R_S|$ for clarity). In our formulation, a large hub score of node i means that many nodes can be visited after i. The hub ranking method is defined as follows.

Definition (Hub ranking)**.** *The remaining nodes $i \in R_S$ are sorted in descending order of the hub scores h_i: The branches are evaluated in the order $i_1, \ldots, i_{|R_S|}$, where $h_{i_1} \geq h_{i_2} \geq \ldots \geq h_{i_{|R_S|}}$.*

The hub scores are used to give guidance to the algorithm regarding the order in which the depth-first search visits the nodes. Since the goal is to maximize the number of served customers, the sequences that can not improve the maximum number of served customers, can be discarded by studying the neighborhoods of the remaining nodes.

Definition. *The neighborhood of node $i \in R_S$ is defined by $N(i) = \{i\} \cup \{j \in R_S \mid L_{ij} = 1\}$.*

Suboptimal sequences are detected as follows.

Theorem 2. *Let $S_{\max}$ denote the best found solution. The sequence S is suboptimal if*

$$C(S) + \max_{i \in R_S} C(N(i)) \leq C(S_{\max}),$$

where $C(S), C(N(i))$ and $C(S_{\max})$ denote the number of drop-off points in $S, N(i)$ and $S_{\max}$, respectively.

Proof. Suppose there exists a feasible sequence $(S, i_1, \ldots, i_h)$ for which $C((i_1, \ldots, i_h)) > \max_{i \in R_S} C(N(i))$. Then, since $\{i_1, \ldots, i_h\} \subset N(i_1)$, we have $C(N(i_1)) \geq C((i_1, \ldots, i_h)) > \max_{i \in R_S} C(N(i))$, which is a contradiction. □

1.3.3. A heuristic extension

Algorithm 1 describes an exact procedure for maximizing the number of served customers in a single route. In order to find the exact solution, the algorithm goes through all branches until no further nodes can be added to the current sequence or the sequence is seen to be suboptimal.

The effort of the algorithm can be controlled by limiting the number of branches that are evaluated by means of a positive parameter J. For example, if $J = 1$, the algorithm constructs a single sequence and stops when the set of remaining nodes in empty. By increasing J the search space is expanded and if $J = (2n)!/2^n$ (the number of permutations that satisfy precedence constraints), the heuristic coincides with the exact algorithm.

1.4. Multi-vehicle solution

In this section, we propose an exact approach to the multi-vehicle DARP as a constraint satisfaction problem. By using Algorithm 1 as a subroutine, the method produces a feasible solution to any instance or proves that the instance is infeasible. The main idea is that the vehicle routes are constructed one by one, each maximizing the number of served customers in the set of remaining customers. Customers are denoted by numbers $1, \ldots, n$ and vehicles are denoted by numbers $1, \ldots, m$.

First, a route is constructed for vehicle 1, serving as many customers as possible. Then, the process is repeated with vehicle 2 for the set of customers that were not served by vehicle 1 and so forth (see Figure 3). Letting X denote a set of customers, we denote by $\mathrm{SV}(X)$ the single-vehicle algorithm (Algorithm 1) that returns a route that maximizes number of served customers in set X and the corresponding set of served customers. This set will be referred to as a maximum cluster of X, which is formally defined as follows.

Definition. *Let X be a set of customers and $M \subset X$ be a subset of X, for which there exists a feasible route serving all customers in M. If $|M| \geq |Y|$ for all sets $Y \subset X$ for which there exists a feasible route serving all customers in Y, then M is a* maximum cluster *of X.*

Note that if there exists a feasible route serving all customers in X, we have $M(X) = X$. The solution to the multi-vehicle case is outlined in Algorithm 2.

for all $k \in \{1, \ldots, m\}$ **do**
  $[S_k, X'_k] \leftarrow \mathrm{SV}(X_k)$; ($S_k$ = route, X'_k = set of served customers)
  $X_{k+1} \leftarrow X_{k+1} \cup (X_k \setminus X'_k)$, where $X_{m+1} = X_1$;
end for

Algorithm 2: Outline of the multi-vehicle algorithm. X_k denotes the set of customers assigned to vehicle k and $\mathrm{SV}(X_k)$ denotes the single-vehicle subroutine presented in Algorithm 1 that returns a maximum cluster X'_k of X_k and the corresponding route S_k. Initially, all customers are assigned to the first vehicle, that is, $X_1 = \{1, \ldots, n\}$ and $X_k = \emptyset$ for all $k \in \{2, \ldots, m\}$.

If a feasible solution is not found directly by using the procedure described in Algorithm 2, the process is repeated. The approach is to find a set of customer-vehicle assignments such that for each vehicle there exists a feasible route serving all customers assigned to the vehicle or to prove infeasibility by going through all possible sets of customer-vehicle assignments [28].

We note that since the number of possible partitions of n customers into m sets is equal to the Stirling number of the second kind, that is, $|\mathcal{P}| = S_2(n, m) = \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{i} (m - i)^n$, going through all possible partitions is computationally taxing. However, some instances are rendered infeasible by studying the feasibility of routes involving two customers: If there exist no feasible routes involving customers $\{i, j\}$, an arc between i and j is formed (customers i and j can not be assigned to the

same vehicle). Then, we find the *maximum clique* C within the set of customers, that is, the largest set of customers for which there is an arc between all pairs of nodes $i, j \in C$, see [31]. If the size $|C|$ of the maximum clique is greater than the number m of vehicles, the instance is infeasible.

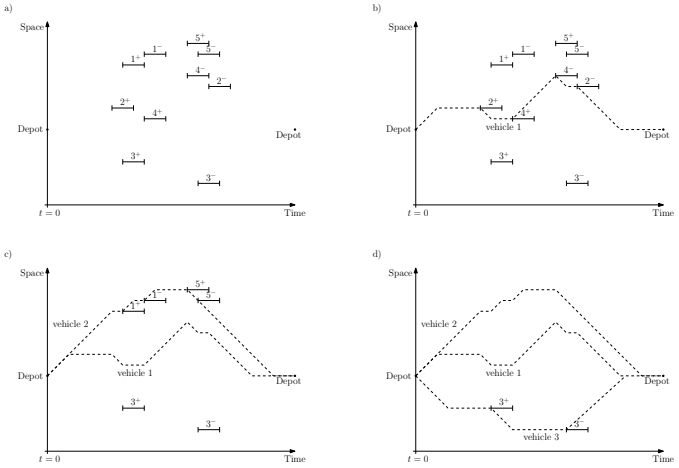

Figure 3: A one-dimensional example of the approach to the multi-vehicle problem involving five customers and three vehicles. The first route is constructed by maximizing the number of customers that can be served by a single vehicle (Figure b). Then, the customers 2 and 4 served by the first vehicle are removed from the set of remaining customers and the process is repeated for the second vehicle (Figure c). Finally, a route is construced for the third vehicle, serving the last remaining customer 3 (Figure d).

2. Structure and complexity

The structure of the single vehicle algorithm is as follows. Briefly described, each recursion step involves checking the the feasibility of sequences (S, i, j), where $i, j \in R_S$. Clearly, this process has a complexity of order $O(|R_S|^2)$, where $|R_S|$ is the number of remaining nodes. After the adjacency matrix L of the remaining nodes has been determined, the complexity of calculating k steps of the power method (Equation (1)) is of order $O(k|R_S|^2)$.

The overall complexity of finding a feasible solution in the single vehicle case is strongly dependent on the number of recursion steps needed to find the solution. Finding a feasible solution to a problem with n customers involves at least $2n$ recursion steps: At first, the number of remaining nodes is $2n$ and each time a feasible node is added to the end of the service sequence, the number of remaining nodes is decreased by one (see Figure 3). In the best case, no infeasible states are encountered and thus there are $2n$ recursion steps (one for each node added to the sequence). At step $k \in \{1, \ldots, 2n\}$, the number of remaining nodes is $2n+1-k$ and thus the complexity is of order $O(\sum_{k=1}^{2n}(2n+1-k)^2) = O(n^3)$. Roughly, the computational work is linearly increased with the number of dead ends M encountered during the recursion. The overall complexity of the single vehicle algorithm is thus $O(n^3 + M)$.

The multi-vehicle algorithm involves solving the single vehicle case for each vehicle in r iterations. Thus the complexity is bounded above by $O(rmn^3)$. Note that the complexity is in general lower since the size of the set of customers is n only the first vehicle. For the other vehicles, the number of customers is smaller since the customers that are already included in another vehicle route are not a part of the subproblem. For example, if the customers were divided equally among vehicles a priori, the complexity would be of order $O(rm(n/m)^3) = O(r(n^3/m^2))$.

3. Numerical experiments

In the following, we present extracts of the results of computational tests reported in [28]. An algorithm using the hub ranking idea is compared with two existing solution methods, namely, a tabu search algorithm [13] and a constraint programming (CP) algorithm [20].

The hub algorithm was implemented in Matlab and the tests were performed on a 2.2 GHz Dual Core Intel PC. The tabu and CP algorithms were tested on a 2.5 GHz Dual Core AMD Opteron computer [18].

In the studied instances the pick-up and drop-off points are located in a 20×20 square and the ride times between points (in minutes) are equal to Euclidean distances. The time windows have 15 minutes of length. In the first set of instances marked with a, the capacity of a vehicle equals $Q = 3$ and the load of each customer is 1. In the second set marked with b, we have $Q = 6$ and the load associated to each customer is chosen randomly according to a uniform distribution on the set $\{1, \ldots, Q\}$. In both sets, the maximum ride time is equal to $R = 22$ and the service time is proportional to the number of passengers, namely $d_i = d_{n+i} = q_i$. The instances are described in more detail in [16, 20, 32].

The results of the tests are shown in Table 3. The first column shows the instance labels of the form *am-n* or *bm-n*, where m indicates the number of vehicles and n corresponds to the number of customers. The other columns show the average time (in seconds) needed to solve the instances and the corresponding modifications by using the different solution methods, calculated over ten runs. The results obtained by the first two algorithms, tabu and CP, have been reported previously in [18, 20]. and the results for the hub algorithm have been reported in [28]. A number in parentheses indicates that the instance was proven to be infeasible, a dash indicates that a solution was not found in three minutes computing time and a star indicates that results for the instance have not been reported. The Matlab-implementations of the hub algorithm are available at `http://math.tkk.fi/~lehame/exact/`. The heuristic version of the algorithm with $J = 1$ was used (see Section 1.3.3).

By looking at the results obtained by the hub algorithm we see that most instances were solved within a fraction of a second. The CPU times obtained by the hub algorithm are typically of order ten times smaller compared to the results of the tabu and CP algorithms. Although the algorithms were tested on different platforms, the results seem to justify the efficiency of the hub algorithm on the test instances. We acknowledge that there are instances in wich tabu and CP produced a feasible solution or proved infeasibility faster than the hub algorithm. However, the results suggest that the multi-vehicle hub algorithm may have practical importance since it is capable of handling large problems in short computation times.

Instance	Tabu	CP	Hub
a4-40	0.5	0.3	0.05
a4-48	1.3	0.4	0.05
a5-40	0.3	0.3	0.02
a5-50	0.6	0.6	0.03
a5-60	1.5	0.9	0.05
a6-48	0.4	0.7	0.03
a6-60	-	(1.1)	(0.63)
a6-72	-	(1.7)	(0.77)
a7-56	-	(0.6)	(0.40)
a7-70	1.4	7.6	0.06
a7-84	3.0	4.1	0.08
a8-64	-	(1.9)	(0.84)
a8-80	1.8	6.0	0.07
a8-96	-	(3.7)	(1.72)

Instance	Tabu	CP	Hub
b4-40	0.4	0.3	0.05
b4-48	-	(0.1)	(0.32)
b5-40	-	(0.1)	(0.27)
b5-50	1.1	0.9	0.18
b5-60	1.6	1.2	0.04
b6-48	*	*	0.02
b6-60	1.0	1.4	0.04
b6-72	2.4	2.4	0.05
b7-56	0.5	1.5	0.04
b7-70	1.3	2.7	0.05
b7-84	-	(0.1)	(0.64)
b8-64	0.8	1.9	0.04
b8-80	-	(0.5)	(1.28)
b8-96	3.9	56.1	0.15

Table 1: Comparison between a tabu search algorithm [13], a constraint programming algorithm [18] and the hub algorithm. The results are given in [18, 20, 28]. The upper table shows the average time (in seconds) needed to solve the instance of the dial-a-ride problem in the first column by using the different solution methods, calculated over ten runs. The times without parentheses indicate that a feasible solution was found and the times in parentheses indicate that the instance was proven to be infeasible. A star (*) indicates that the computing time has not been reported.

4. Conclusions

In this work, an efficient dynamic programming method for the time constrained dial-a-ride problem is suggested. The purpose of the method is to produce routes that satisfy the constraints of the problem, since finding such routes is seen to be a major challenge in highly constrained problems. The main result is an exact constraint satisfaction algorithm for the multi-vehicle case.

Our approach is motivated by HITS, a link-analysis algorithm originally developed for web information retrieval. We define links between the pick-up and drop-off nodes as feasible transitions with respect to the constraints of the problem, *hubs* as nodes with several out-links. Ranking the nodes by hub score gives guidance to a depth-first search used to maximize the number of served customers in a single vehicle route. The method is extended to solve the multi-vehicle case. Numerical experiments suggest that the proposed ranking method is capable of producing feasible solutions to large problems in relatively small computation times.

This work is an example on how recommendation-type link analysis can be used to solve combinatorial problems. Although we have focused on the Dial-A-Ride Problem (DARP), we note that the DARP is a generalization of the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP) and thus the proposed hub ranking method could be useful in solving other constrained routing problems as well.

References

[1] J. Kleinberg, Authoritative sources in a hyperlinked environment, in: Proceedings of the Ninth Annual ACM-SIAM Symposium of Discrete Algorithms, ACM Press, New York, 1998, pp. 668–677.

[2] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: Proceedings of the Seventh International Conference on World Wide Web, Vol. 7, 1998, pp. 107–117.

[3] R. Lempel, S. Moran, The stochastic approach for link-structure analysis (salsa) and the tkc effect, in: The 9th Intl. WWW Conference, 2000.

[4] A. Farahat, T. LoFaro, J. C. Miller, G. Rae, L. Ward, Authority rankings from hits, pagerank, and salsa: existence, uniqueness, and effect of initialization, SIAM Journal on Scientific Computing 27 (2006) 1181–1201.

[5] A. Langville, C. Meyer, A survey of eigenvector methods of web information retrieval, SIAM Review 47 (2005) 135–161.

[6] A. Y. Ng, A. X. Zheng, M. I. Jordan, Stable algorithms for link analysis, SIGIR01, New Orleans, Louisiana, USA.

[7] M. Agosti, L. Pretto, A theoretical study of a generalized version of kleinberg's hits algorithm, Information Retrieval 8 (2005) 219–243.

[8] H. Psaraftis, An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows, Transportation Science 17 (1983) 351–357.

[9] J. Jaw, A. Odoni, H. Psaraftis, N. Wilson, A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows, Transportation Research Part B 20 (1986) 243–257.

[10] O. Madsen, H. Ravn, J. Rygaard, A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives, Annals of Operations Research 60 (1995) 193–208.

[11] P. Toth, D. Vigo, Heuristic algorithms for the handicapped persons transportation problem, Transportation Science 31 (1997) 60–71.

[12] B. Hunsaker, M. W. P. Savelsbergh, Efficient feasibility testing for dial-a-ride problems, Operations Research Letters 30 (2002) 169–173.

[13] J.-F. Cordeau, G. Laporte, A tabu search heuristic for the static multi-vehicle dial-a-ride problem, Transportation Research B 37 (2003a) 579–594.

[14] M. Diana, M. Dessouky, A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows, Transportation Research Part B 38 (2004) 539–557.

[15] K. Wong, M. Bell, Solution of the dial-a-ride problem with multi-dimensional capacity constraints, International Transactions in Operational Research 13 (2006) 195–208.

[16] J.-F. Cordeau, A branch-and-cut algorithm for the dial-a-ride problem, Operations Research 54 (2006) 573–586.

[17] L. Häme, An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows, European Journal of Operational Research. 209 (1) (2011) 11 – 22.

[18] G. Berbeglia, Complexity Analyses and Algorithms for Pickup and Delivery Problems, Ph.D. Thesis, HEC Montreal, 2009.

[19] G. Berbeglia, J.-F. Cordeau, G. Laporte, A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem, forthcoming (2011).

[20] G. Berbeglia, G. Pesant, L.-M. Rousseau, Checking feasibility of the dial-a-ride problem using constraint programming, forthcoming (2011).

[21] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, part i: Route construction and local search algorithms, Transportation Science 39 (1) (2005) 104118.

[22] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, part ii: Metaheuristics, Transportation Science 39 (1) (2005) 119139.

[23] C. Groër, B. Golden, E. Wasil, A library of local search heuristics for the vehicle routing problem, Mathematical Programming Computation 2 (2).

[24] Helsinki Region Traffic, Jouko neighbourhood routes and service routes (Oct. 2010).
URL `http://www.hsl.fi/EN/passengersguide/`

[25] S. Axler, Linear Algebra Done Right, Springer-Verlag, ISBN 0-387-98258-2, 1996.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, 2001.

[27] J. Ponstein, Self-avoiding paths and the adjacency matrix of a graph 14 (3) (1966) 600–609.

[28] L. Häme, H. Hakula, A maximum cluster algorithm for checking the feasibility of dial-a-ride instances, manuscript submitted to Transportation Science.

[29] J. Desrosiers, Y. Dumas, F. Soumis, A dynamic programming solution of the large-scale singlevehicle dial-a-ride problem with time windows, American Journal of Mathematical and Management Sciences 6 (1986) 301–325.

[30] Y. Dumas, J. Desrosiers, F. Soumis, The pickup and delivery problem with time windows, European Journal of Operational Research 54 (1991) 7–22.

[31] D. R. Wood, An algorithm for finding a maximum clique in a graph, Operations Research Letters 21 (5) (1997) 211–217.

[32] S. Ropke, J.-F. Cordeau, G. Laporte, Models and branch-and-cut algorithm for pickup and delivery problems with time windows, Networks 49 (2007) 258–272.