Math Meth Oper Res (2013) 77:371–380
DOI 10.1007/s00186-012-0403-8

ORIGINAL ARTICLE

An adaptive routing approach for personal rapid transit

Kaspar Schüpbach · Rico Zenklusen

Published online: 11 September 2012

Abstract Personal Rapid Transit (PRT) is a public transportation mode, in which small automated vehicles transport passengers on demand. Central control of the vehicles leads to interesting possibilities for optimized routings. The complexity of the involved routing problems together with the fact that routing algorithms for PRT essentially have to run in real-time often leads to the choice of fast greedy approaches. The most common routing approach is arguably a sequential one, where upcoming requests are greedily served in a quickest way without interfering with previously routed vehicles. The simplicity of this approach stems from the fact that a chosen route is never changed later. This is as well the main drawback of it, potentially leading to large detours. It is natural to ask how much one could gain by using a more adaptive routing strategy. This question is the main motivation of this article. In this paper, we first suggest a simple mathematical model for PRT, and then introduce a new adaptive routing algorithm that repeatedly uses solutions to an LP as a guide to route vehicles. Our routing approach incorporates new requests in the LP as soon as they appear, and reoptimizes the routing of all currently used vehicles, contrary to sequential routing. We provide preliminary computational results that give first evidence of the potential gains of an adaptive routing strategy, as used in our algorithm.

Keywords Personal rapid transit · Adaptive routing · Online algorithms · Decision support models

K. Schüpbach
ETH Zurich, Institut für Operations Research, HG E 65.1, Rämistrasse 101, 8092 Zurich, Switzerland
e-mail: kaspar.schuepbach@ifor.math.ethz.ch

R. Zenklusen (✉)
MIT, Department of Mathematics, Room 2-332, Cambridge, MA 02139, USA
e-mail: ricoz@math.mit.edu

1 Introduction

Personal Rapid Transit (PRT) is a rather novel mode of public transportation, where small automated vehicles, called *pods*, transport passengers on demand on a track network. Passengers can choose the destination of the ride when entering the pod at a station. Then a central automatic system guides the pod through the network to the desired destination. PRT is intended to complement public transportation systems, as subways and trains, by offering a quick means of transportation for short distances, thus addressing the well-known *last mile problem*. The goal of PRT is hence to improve public transportation in general and increase its share, also by providing an alternative to the automobile. Even though the general idea of PRT is several decades old [see Fichter (1964)], only recently first PRT projects were deployed and many more entered concrete planning and decision phases [see Request for proposal for San José automated transit network FFRDC development services (2009), Feasibility of PRT in Ithaca, New York—executive summary (2010), Sukayna (2011), Heathrow T5 (2011)].

The fact that the routing can be done centrally in a PRT system is a major advantage compared to road traffic, and opens up exciting new possibilities. However, the central routing comes with two major challenges. First, the resulting routing problems are likely to be computationally hard. In particular, it was shown in Schüpbach and Zenklusen (2011) that even a simple offline model for automated vehicle routing [a slight variation of which was previously considered in Krishnamurthy et al. (1993), Spenke (2006), Stenzel (2008)], already leads to NP hard problems even on a network consisting of a single path. For the same model, only very recently approximation algorithms were found with sublinear approximation guarantee with respect to the number of vehicles to be routed (Schüpbach and Zenklusen 2011). Second, online routing algorithms for PRT must be real-time, to avoid that vehicles stand idle while waiting for routing instructions.

The arguably most common routing paradigm for similar online routing problems is a sequential scheme, that we simply call *sequential routing*. Here, whenever a request is revealed, a quickest way to fulfill it is determined (avoiding conflicts with previously fixed routes for requests revealed earlier). In particular, once a routing for a particular request is fixed, it is never changed later. The simplicity of sequential routing is appealing for applications and, despite its simplicity, it performs very well in a variety of settings (Kim and Tanchoco 1991; Möhring et al. 2005; Rivera-Vega et al. 1990; Stenzel 2008).

Nevertheless, the policy of never changing a fixed route in the future is a restriction of sequential routing that comes at its price. Pods routed later might have to take large detours, even though it may be possible that a slight rerouting of previously routed pods could free the desired tracks for new requests. In this paper, we are interested in addressing this potential drawback.

First, we introduce a simple model for PRT capturing some central features of the problem. The introduced model is deadlock-free, i.e., no matter how pods are routed up to some time τ, it is always still possible to fulfill all requests. We then suggest an online routing algorithm, that iteratively re-optimizes the routing choices of all involved pods. Our algorithm first solves an LP relaxation, corresponding to a node capacitated multi-commodity flow problem on a time-expanded graph, and then locally (in terms of time) rounds the obtained solution. Finally, we provide preliminary computational results to compare our algorithm against the sequential routing approach.

2 The model

The PRT model we suggest is essentially a directed online version of the offline model to route automated guided vehicles presented in (Schüpbach and Zenklusen 2011). We are given a directed graph $G = (V, A)$ representing the track network and consider a discretisation of the time into time steps. Pods reside on nodes, and at each time step each pod can either remain on its current node or move along an arc to an adjacent node. No two pods are allowed to reside at the same node during the same time step.

The unit travel time between any two adjacent nodes in G is obtained by subdividing physical tracks with higher travel times through the introduction of extra nodes. The fact that any two pods in our model must be separated by at least one time step is used to ensure that pods keep some minimum temporal separation for safety reasons. This is a typical requirement in many conflict-free routing applications. Hence, the time unit would be chosen to be roughly this desired headway time. Our model does not deal with precise acceleration patterns for pods, but essentially assumes that the speed of any pod depends only on its location. The faster a pod can move on some track, the less often the track is subdivided to obtain the desired headway time.

At each time step, new *requests* may appear. Each request π is described by a triple (s_π, t_π, τ_π), where $s_\pi, t_\pi \in V$, $s_\pi \neq t_\pi$ are the *start point* and *end point* (or *origin* and *destination*) of the request, and $\tau_\pi \in \mathbb{Z}_+$ is the *release* time of the request. In other words, (s_π, t_π, τ_π) is a request that was filed at time τ_π by passengers who want to go from s_π to t_π. We do not impose any hard constraints on how many requests can appear at any time step, but only assume that the total number of requests will be finite. For the computational results in Sect. 4, we discuss a probabilistic model for the appearances of request. For any time τ, a request π is *open* if $\tau_\pi \leq \tau$, and the request was not served yet to completion. We make a rather strong assumption about the availability of empty pods. Namely, whenever a new request (s_π, t_π, τ_π) appears, we assume that there is an empty pod immediately available at s_π at time step τ_π. The main reason for this assumption is that we are primarily interested in the problem of finding strong routes between origin-destination pairs. The above assumption allows us to isolate this question from other challenges interesting in their own right, like a possible shortage of pods, or the navigation of spare empty pods to places where they are ready to quickly serve upcoming requests (see Lees-Miller (2011) and references therein for more information). Essentially, our setting can be loosely interpreted as having no shortage of pods, and such that at any time there is only very little time needed to navigate an empty pod to any fixed node. Due to this assumption, we call—with slight abuse of notation—the pod serving request (s_π, t_π, τ_π) simply π.

As a relaxation of the conflict definition at a node, we assume that pods can only be conflicting while in transit, i.e., a pod π cannot cause a conflict before departure and after arrival. The departure time of π is the last time step that the pod is still at its origin s_π, and the arrival time is the earliest time when π is at its destination t_π. We call this relaxation, which was as well used in Schüpbach and Zenklusen (2011), the *parking assumption*. The motivation for the parking assumption stems from the fact that in typical PRT settings, the stations are on separate side tracks, and thus do not interfere with the remaining network. We call a pod/request π *active* if it is open and π is not anymore at its parking position at the origin.

To simplify the exposition we concentrate on one natural objective function, namely the average delay of the requests. Once a pod π arrives at its destination at some time step τ, we define its *delay* del_π—or the delay of the corresponding request—as

$$del_\pi := \tau - \tau_\pi - d(s_\pi, t_\pi),$$

where $d(s_\pi, t_\pi)$ is the length of a shortest path (in number of arcs) from s_π to t_π. Hence, the delay is the difference between the time needed to complete a request and the shortest possible time to fulfill the request if there were no other pods on the network which could cause conflicts. Our goal is thus to minimize $\frac{1}{|\Pi|}\sum_{\pi\in\Pi} del_\pi$, where Π is the set of all requests, which will be revealed online.

Due to the parking assumption it is obvious that there always exists a feasible way to route the pods to their destinations: one could simply serve the requests one-by-one, thus only having a single pod on the tracks at any time, while the pods for all other active requests are waiting at the corresponding start points. However, the parking assumption (at the destinations) implies something much stronger, namely that it is impossible to create deadlocks, no matter how one routes the pods. A *deadlock* is a constellation of pods and corresponding requests such that it is not anymore possible to route all pods to their destinations since pods on the tracks block each other.

Theorem 1 *No deadlocks are possible in the suggested PRT model, i.e., for any time τ, any set of open requests and position of pods serving the open requests, it is possible to fulfill all requests.*

Proof It clearly suffices to show that it is always possible to serve all active requests, since one can wait with serving non-started and future requests until the active ones are completed, and then use e.g. a simple serial one-by-one routing strategy as mentioned above. Hence, let Π all active requests, and for $\pi \in \Pi$ let $v_\pi \in V$ be the current position of pod π. We prove the theorem by showing that during the next time step, the current pods can be routed such that they reach positions $v'_\pi \in V$ for $\pi \in \Pi$ satisfying

$$\sum_{\pi\in\Pi} d(v'_\pi, t_\pi) < \sum_{\pi\in\Pi} d(v_\pi, t_\pi), \tag{1}$$

i.e., the total distance of the current pods to their destinations strictly decreases. For each $\pi \in \Pi$ we fix an arbitrary arc a_π outgoing of v_π that would bring pod π closer to its destination. Let $A_\Pi = \{a_\pi \mid \pi \in \Pi\}$. Consider the subgraph (V, A_Π) of G. In (V, A_Π) each vertex has at most one outgoing arc, since each vertex contains at most one pod. If A_π contains a directed cycle C, then we can move each pod π on a vertex of C along its arc a_π, without moving any other pods. This clearly does not create a conflict and fulfills (1). Otherwise, if (V, A_Π) does not contain any directed cycle, then each pod $\pi \in \Pi$ can simultaneously be moved along a_π without creating a conflict and by bringing them closer to their destinations.

3 Online algorithms for the PRT problem

In this section, we give a short summary of *sequential routing* and introduce our new routing approach. Both algorithms do not assume any particular distribution describing how new requests appear, and therefore base their routing decisions at each time step only on the currently open requests.

Time-expanded graphs, a well-known algorithmic tool to deal with network problems with an inherent time component that was already studied by Ford and Fulkerson (1958), are used for both sequential routing and our approach. We therefore start by introducing time-expanded graphs as used in our context, before discussing sequential routing and our routing approach.

3.1 Time-expanded graphs

A time-expanded graph can be used to describe all routing possibilities during a given time frame. To describe all routing options for the open requests Π at some time step $\overline{\tau}$, we construct a time-expanded graph $H = (W, F)$ for the time frame $\{\overline{\tau}, \overline{\tau}+1, \ldots, T\}$, where $T \in \mathbb{N}$ is a sufficiently large value ensuring that all open requests can be served. The nodes and arcs of $H = (W, F)$ can each be partitioned into two groups $W = W_1 \cup W_2$ and $F = F_1 \cup F_2$, where

$$\begin{aligned}
W_1 &= \{v^\tau \mid v \in V, \tau \in \{\overline{\tau}, \overline{\tau}+1, \ldots, T\}\},\\
F_1 &= \{(v_1^\tau, v_2^{\tau+1}) \mid v_1, v_2 \in V, v_1 = v_2 \text{ or } (v_1, v_2) \in A, \text{and } \tau \in \{\overline{\tau}, \ldots, T-1\}\},\\
W_2 &= \{\overline{s}_\pi \mid \pi \in \Pi\} \cup \{\overline{t}_\pi \mid \pi \in \Pi\},\\
F_2 &= \{(\overline{s}_\pi, s_\pi^\tau) \mid \tau \in \{\overline{\tau}, \ldots, T\}\} \cup \{(t_\pi^\tau, \overline{t}_\pi) \mid \tau \in \{\overline{\tau}, \ldots, T\}\}
\end{aligned}$$

The nodes and arcs given by W_1 and F_1 give a representation for each vertex in V at each time step in $\{\overline{\tau}, \ldots, T\}$, and how vehicles can move between them. The extra nodes and arcs W_2 and F_2 are used to model the parking assumptions. A routing for a pod in Π can now simply be represented by a single path in H. If a pod $\pi \in \Pi$ is still at its parking position at time step $\overline{\tau}$, then a routing of π corresponds to a path from $a_\pi = \overline{s}_\pi$ to $\overline{t}_\pi$. Otherwise, if π is active and its current position is v_π, then a routing of π corresponds to a path from $a_\pi = v_\pi^{\overline{\tau}}$ to $\overline{t}_\pi$. One can easily observe that a simultaneous conflict-free routing of all vehicles in Π corresponds to choosing for each $\pi \in \Pi$, a path P_π from a_π to $\overline{t}_\pi$ in H such that the paths $\{P_\pi \mid \pi \in \Pi\}$ are *node-disjoint*. Furthermore, the delay of a vehicle $\pi \in \Pi$ only depends on the last arc on P_π, which is of the form $(t_\pi^\tau, \overline{t}_\pi)$, and τ then corresponds to the arrival time of π when using the routing given by P_π.

As a simple means to associate the corresponding delay when routing π over a path P_π in H, we assign costs c to the arcs F. Namely, all arcs in F_1 and arcs of the form $(\overline{s}_\pi, s_\pi^\tau)$ have a cost of 0. Furthermore, an arc $f = (t_\pi^\tau, \overline{t}_\pi)$ has a cost of

$c(f) = \tau - \tau_\pi - d(s_\pi, t_\pi)$, which is precisely the delay of any routing for π given by a path P_π in H which uses $(t_\pi^\tau, \bar{t}_\pi)$ as its last arc. Therefore, the delay of π is equal to $c(P_\pi)$, the cost of the path P_π with respect to c. In particular, when assuming that no new requests appear, finding the optimal way to route the open requests Π corresponds to finding node-disjoint paths between a_π and $\bar{t}_\pi$ for $\pi \in \Pi$ of minimum total length with respect to c.

3.2 Sequential routing

Sequential routing is a standard routing paradigm for vehicle routing problems [see e.g. Kim and Tanchoco (1991), Möhring et al. (2005)]. In sequential routing, once a request π appears, a routing is fixed for π that will not be changed later, no matter what other requests are revealed in the future. For a time step τ, the routing of the requests Π_τ revealed to τ is determined as follows. We go through the requests Π_τ in any order, determining routings one-by-one. The routing of a pod π is chosen to be the quickest possible without creating conflicts with previously fixed routes, i.e., this includes requests with reveal time $<\tau$ and those considered before π at time step τ. Such a quickest route can be obtained by computing a shortest a_π—$\bar{t}_\pi$ path in H over all nodes not used by any previously fixed routes.

3.3 An adaptive LP-based routing approach

To describe our approach we consider an arbitrary constellation of open requests Π at some time step τ and specify how to route the pods between τ and $\tau + 1$. Based on the time-expanded graph $H = (W, F)$, we consider an LP relaxation for the best way to complete all open requests, assuming that no new requests appear. More precisely, $H = (W, F)$ spans the time steps from τ up to some sufficiently large time T. We will explain soon what “sufficiently large” means and how the choice of the right T can be handled implicitly. For each open request $\pi \in \Pi$, let $a_\pi \in W$ be the node corresponding to the current position of pod π at time step τ. Our goal is to solve a minimum cost multicommodity flow on H with unit node capacities and costs given by c, where each request $\pi \in \Pi$ corresponds to a different commodity, and the goal is to send for each $\pi \in \Pi$ a unit-flow from a_π to $\bar{t}_\pi$. Unit node capacities enforce that no node has more than one unit of flow entering nor leaving.

Notice that since no deadlocks are possible in our PRT model due to Theorem 1, there is a time $\overline{T}$ such that all open pods can be routed to their destinations, and hence also the multicommodity LP will be feasible for $T = \overline{T}$. Furthermore, by choosing T to be larger than the minimum value which leads to LP feasibility, LP solutions of smaller cost may result. However, it is also not hard to argue that there is a *sufficiently large* value $\widetilde{T}$ such that the LP solution for $T = \widetilde{T}$ does not get even cheaper for any $T > \widetilde{T}$. We are interested in finding a minimum cost multicommodity flow for some $T \geq \widetilde{T}$. Since the way we find such a minimum cost multicommodity flow uses a combination of rather standard procedures, we only give a brief overview of the details.

More precisely, the way we deal with the issues of not knowing $\widetilde{T}$ and the potentially large resulting LP, is by using a column generation approach to solve the LP. We use the classical path formulation of the multicommodity flow problem [see e.g. Korte and Vygen (2008)], and generate new paths step by step, by finding for each commodity $\pi \in \Pi$ a minimum weighted a_π-$\bar{t}_\pi$ path, whose weight is the sum of node weights given by the dual variables of the LP and arc weights given by c [see Schrijver (2003) for more details on this standard column generation technique]. With this approach, we can use Dijkstra's algorithm on a time-expanded graph to find new paths for $\pi \in \Pi$. Notice that for Dijkstra's algorithm, we do not have to fix a precise value of T, but we can just add additional time layers during the algorithm as needed until Dijkstra's algorithm reaches $\bar{t}_\pi$. After having added new paths to the LP, we use CPLEX 12.3 to solve the LP with the extended set of variables[1]. Once an optimal multicommodity flow is obtained, we are only interested in flow between time step τ and $\tau + 1$. Let

$$\overline{V} = V \cup \{\bar{s}_\pi \mid \pi \in \Pi\} \cup \{\bar{t}_\pi \mid \pi \in \Pi\},$$

i.e., $\overline{V}$ represents all possible locations where pods can be (at time $\tau + 1$), where $\bar{s}_\pi, \bar{t}_\pi$ correspond to the pod $\pi \in \Pi$ being parked at the start and end, respectively. For $\pi \in \Pi$ and $v \in \overline{V}$, let $\mathbf{x}_\pi(v) \in [0, 1]$ be the flow value of commodity π through location v at time step $\tau + 1$. E.g., if $v \in V$, then $\mathbf{x}_\pi(v)$ is the total flow in H of commodity π going through the node v^τ; furthermore $\mathbf{x}_\pi(\bar{s}_\pi), \mathbf{x}_\pi(\bar{t}_\pi)$ is the total amount of flow in H of commodity π that is at time step τ at node $\bar{s}_\pi$ and $\bar{t}_\pi$, respectively, i.e., this corresponds to the amount of flow of commodity π that will leave the parking position after τ or reached the destination until time τ, respectively. Hence, for each request $\pi \in \Pi$ we have a distribution $\mathbf{x}_\pi \in [0, 1]^{\overline{V}}$ over $\overline{V}$.

Based on $\mathbf{x}_\pi$, we want determine through a randomized rounding approach to which vertex $u_\pi \in \overline{V}$ pod π will be sent in time step $\tau + 1$. The rounding has of course to satisfy that no conflicts occur, i.e., it must hold $u_\pi \neq u_{\pi'}$ for any $\pi, \pi' \in \Pi$ with $\pi \neq \pi'$. Furthermore, we want to round such that the marginal probabilities are preserved, i.e., $\Pr[u_\pi = v] = \mathbf{x}_\pi(v)\ \forall v \in \overline{V}$. Our problem can easily be translated into the well-known problem of rounding fractional matchings in bipartite graphs. More precisely, the vectors $\mathbf{x}_\pi$ for $\pi \in \Pi$ can be represented as a fractional matching in a (complete) bipartite graph $(\Pi, \overline{V})$, where the weight of the edge $\{\pi, v\}$ is equal to $\mathbf{x}_\pi(v)$. We thus seek to round this fractional matching to an integral matching by preserving marginal probabilities, a well-studied setting discussed e.g. in Gandhi et al. (2006), Chekuri et al. (2010). A conceptually simple rounding technique would be to use a constructive version of Carathéodory's theorem to express the fractional matching as a convex combination of integral matchings, and then pick randomly one of the integral matchings with probability equal to the corresponding coefficient in the convex combination. However, we use the specialized technique as presented e.g. in Gandhi et al. (2006), since it provides a very efficient way to round the fractional matching by doing simple local rounding steps on fractional cycles. After having obtained the

[1] As a feasible starting solution to the multicommodity LP, we introduced very expensive arcs from each origin to each destination and sent flow only over those arcs.

locations u_π for $\pi \in \Pi$ through the rounding, we send pod π to location u_π between time step τ and $\tau + 1$. Next time step, the same procedure is repeated to determine how to send the pods between $\tau + 1$ and $\tau + 2$, and so on.

4 Preliminary computational results

We present some preliminary computational results in this section. As a simple first setting, we consider a grid topology, as used in similar contexts for analyzing the routing of automated guided vehicles (Stenzel 2008). More precisely, we consider a grid of size 8×8, whose edges are oriented in an alternating way as shown in Fig. 1. Requests are randomly generated as follows. Over $1,000$ time steps, the number of new requests at each time step is an independent Poisson random variable with some parameter λ, which we call *release rate*. Hence, a release rate of e.g. $\lambda = 3$ means that three new requests appear at each of the first $1,000$ time steps in average. For each request, origin and destination is chosen uniformly at random among all pairs of two different vertices.

The left plot of Fig. 2 shows, on this grid instance, the average delay for the sequential approach and our adaptive approach depending on the release rate. As expected, for low release rates the two approaches perform similarly, since due to the low traffic there are many good routes to handle new requests, even if conflicts with previously routed pods have to be avoided. However, for higher release rates this restriction of the sequential algorithms turns out to be a serious issue and the adaptive approach is considerable stronger.

The right plot of Fig. 2 shows the number of open requests at each time step for the four release rates 5.4, 5.5, 6.2, 6.3. These release rates were chosen such that one can observe when the routing algorithms reach an unstable mode, i.e., in average more

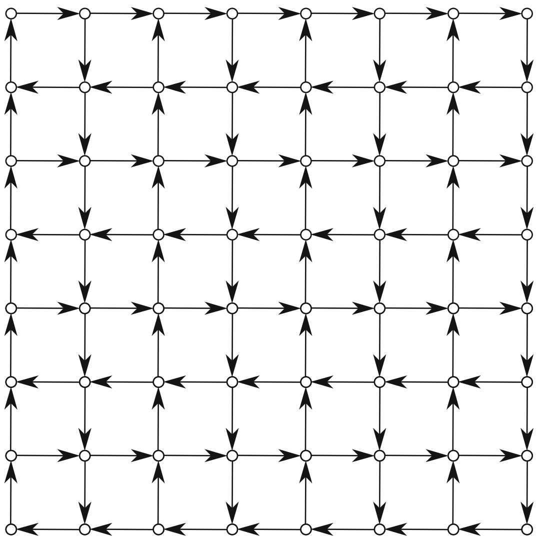
Fig. 1 Topology of the grid instance we use for computational results

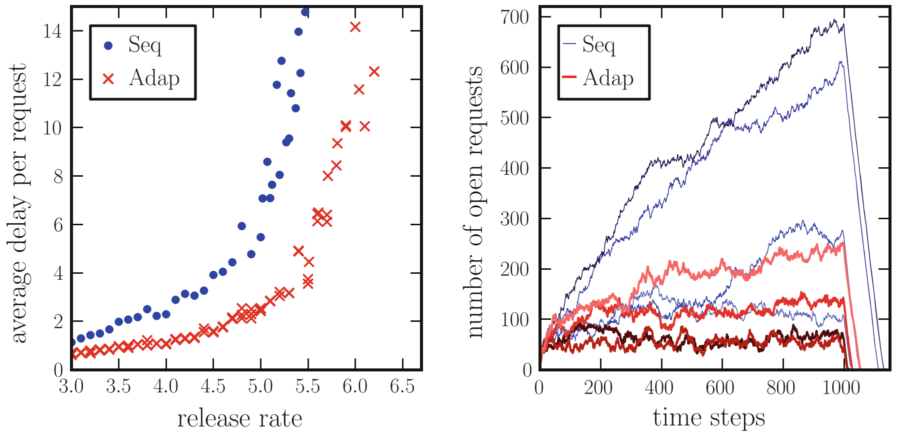

Fig. 2 *Left* average delay per request for different release rates for sequential routing (Seq) and our adaptive approach (Adap). *Right* number of open requests over time for four different release rates: 5.4, 5.5, 6.2, 6.3

new requests appear than the number of requests served. For each algorithm, plots further on top correspond to higher release rates. In this example, sequential routing is stable for a release rate of 5.4 and starts to accumulate open requests for release rates from 5.5 on. Our approach is stable for release rates up to roughly 6.2 and becomes unstable for release rate 6.3. One can also observe that the adaptive approach leads to a much more efficient use of the infrastructure for the same release rate.

For the above instance, the runtime of the proposed approach was in the range of about 10 seconds per time step for release rates around 6, and considerable lower for smaller release rates. The computations were performed on a computer with an Intel Core i7 2.7 GHz dual-core processor and 4GB RAM. Whereas these preliminary computational results are obviously limited, they provide good reason to hope that also in more general settings an adaptive route choice as in our routing approach can lead to considerable shorter delays.

5 Concluding remarks

We introduced a simple model for PRT routing, and presented an LP-based adaptive routing algorithm. Our goal was to suggest an alternative to sequential routing—a standard routing approach used in many similar settings—that does not suffer from bad routing choices that cannot be adapted at a later point in time. At each time step our algorithm locally rounds an LP solution to determine how to route pods from this time step to the next one. First computational results show potential gains that can be achieved with this more adaptive routing approach, and within reasonable computational time.

A natural direction for further research would be to extend our preliminary computational results in general, but in particular by considering different network topologies. Also a more general PRT model could be of interest for cases where the parking assumption does not holds, and where only a limited number of pods is considered, such that the movement of empty pods becomes relevant. Furthermore, we believe that the study of alternative objective functions is of interest. In particular, instead of minimizing average delay, one could try to reach a particular well-defined service level that penalizes more strongly very high delays.

Acknowledgments Both authors wish to express their gratitude to Prof. Hans-Jakob Lüthi, former head of the Institute for Operations Research at ETH Zurich, whose commitment and vision for interdisciplinary research was a strong inspiration for the authors, and in particular led to this article. The second author was supported by NSF grants CCF-1115849 and CCF-0829878, and by ONR grants N00014-11-1-0053 and N00014-09-1-0326.

References

Chekuri C, Vondrák J, Zenklusen R (2010) Dependent randomized rounding via exchange properties of combinatorial structures. In: Proceedings of the 51st IEEE symposium on foundations of computer science (FOCS), pp 575–584

Fichter D (1964) Individualized automatic transit and the city. B. H. Sikes, 1430 East 60th Place, Chicago, IL 60637

Ford LR, Fulkerson DR (1958) Constructing maximal dynamic flows from static flows. Oper Res 6(3):419–433

Gandhi R, Khuller S, Parthasarathy S, Srinivasan A (2006) Dependent rounding and its applications to approximation algorithms. J. ACM 53(3):324–360 doi:10.1145/1147954.1147956

Kim CW, Tanchoco JMA (1991) Conflict-free shortest-time bidirectional AGV routing. Int J Prod Res

Korte B, Vygen J (2008) Combinatorial optimization, theory and algorithms. 4. Springer, Berlin

Krishnamurthy NN, Batta R, Karwan MH (1993) Developing conflict-free routes for automated guided vehicles. Oper Res 41(6):1077–1090

Lees-Miller JD (2011) Empty vehicle redistribution for personal rapid transit. Ph.D. thesis, University of Bristol

Möhring RH, Köhler E, Gawrilow E, Stenzel B (2005) Conflict-free real-time AGV routing. In: Proceedings of operations research, pp 18–24

Rivera-Vega P, Varadarajan R, Navathe S (1990) Scheduling data redistribution in distributed databases. In: Proceedings of sixth international conference on data engineering, pp 166–173. doi:10.1109/ICDE.1990.113466

Request for proposal for San José automated transit network FFRDC development services (2009) Tech. rep., Department of Transportation of the City of San José

Feasibility of PRT in Ithaca, New York—executive summary (2010) Tech. rep., New York State Department of Transportation

Schrijver A (2003) Combinatorial optimization, polyhedra and efficiency. Springer, Berlin

Schüpbach K, Zenklusen R (2011) Approximation algorithms for conflict-free vehicle routing. In: Proceedings of the 18th Annual European Symposium on Algorithms (ESA), pp 640–651

Spenke I (2006) Complexity and approximation of static k-splittable flows and dynamic grid flows. Ph.D. thesis, Technische Universität Berlin

Stenzel B (2008) Online disjoint vehicle routing with application to agv routing. Ph.D. thesis, Technische Universität Berlin

Sukayna S (2011) Gurgaon eyes elevated pods. Times of India, Mumbai

Heathrow T5. Website (2011). http://www.ultraglobalprt.com/wheres-it-used/heathrow-t5/