Sampling for Personal Rapid Transit Empty Vehicle Redistribution

John D. Lees-Miller
Department of Engineering Mathematics,
University of Bristol, Queen's Building, University Walk,
Bristol, BS8 1TR, United Kingdom
enjdlm@bristol.ac.uk
(corresponding author)

R. Eddie Wilson
Department of Engineering Mathematics,
University of Bristol, Queen's Building, University Walk,
Bristol, BS8 1TR, United Kingdom
RE.Wilson@bristol.ac.uk
Tel.: +44(0) 117 331 5627, Fax.: +44(0) 117 331 5606

6,217 words + 4 figures

ABSTRACT

A Personal Rapid Transit (PRT) system uses compact, computer-guided vehicles running on dedicated guideways to carry individuals or small groups directly between pairs of stations. PRT vehicles operate on demand, when a passenger requests service at his/her origin station. Because the number of trips requested from a station need not equal the number of trips ending there, some vehicles must move empty to balance the flows. The empty vehicle redistribution (EVR) problem is to decide which empty vehicles to move, either reactively, in response to known requests, or proactively, in anticipation of future requests. This paper develops a new algorithm for the EVR problem called Sampling and Voting (SV). SV chooses reactive movements using a simple nearest-neighbor rule, and it chooses proactive movements by generating an ensemble of possible sequences of future passenger requests, solving a deterministic optimization problem for each sequence individually, and then finding the empty vehicle movements that are common among the sequences. Moving vehicles proactively is essential for providing low passenger waiting times. The new SV algorithm is tested in simulation with several case study systems, and it produces significantly lower passenger waiting times than existing EVR algorithms. Variants of the SV method developed here for PRT are also applicable to conventional taxi systems and emergency response systems.

1. INTRODUCTION

Personal Rapid Transit (PRT) is an emerging urban transport mode. The world's first PRT system is currently undergoing final testing at Heathrow Airport in London, England (*1*). The basic idea of a PRT system is that it provides on-demand, non-stop travel with compact, computer-guided vehicles running on a dedicated network of unidirectional guideways. Each PRT vehicle carries either an individual or a small party traveling together by choice. The vehicle begins its trip *on demand*, when a passenger arrives at a PRT station and makes a *request* for service. Once the passenger is ready to depart, his vehicle takes the quickest path to the chosen destination station: it does not stop at intermediate stations to let other passengers on or off. Hence, a PRT system is similar to a taxi system, except that PRT vehicles are constrained to start and end their trips at stations.

In general, the number of trips requested from a particular station need not equal (on average or instantaneously) the number of trips ending at that station, so that some PRT vehicles must move *empty*, that is, without passengers. The question of which vehicles to move, and where to move them, is known as the empty vehicle redistribution (EVR) problem.

In (*2*), we used a fluid limit to analyze the EVR problem in terms of the overall capacity of the PRT system – that is, the maximum demand for travel that can be met without diverging queues of waiting passengers. This fluid limit approach is useful for benchmarking proposed EVR algorithms in terms of capacity, but it has two important limitations:

1. It yields only average flows of empty vehicles and does not suggest an operational algorithm for the movement of vehicles at an individual level.
2. In practice, PRT systems that are presently planned will run far short of the capacity limit, and the main objective will be to minimize passenger waiting time, either at the mean or at some percentile; the fluid-limit cannot be used for this analysis.

The focus of this paper is on operational EVR algorithms, which must move individual empty vehicles in response to passenger requests or in anticipation of future requests. It is assumed that the average request rate between each pair of stations is known from historical data in the form of an origin-destination *demand matrix*, but that the individual requests are revealed only while the system is operating. Historically, EVR algorithms have been based on decision rules (*3*, *4*) to be executed when a new passenger arrives or a vehicle becomes empty. For example, if a station is short of empty vehicles, a vehicle is chosen according to the rules and sent there; this may simply be the nearest empty vehicle at a station that is not itself short of vehicles (*5*), or it may be determined by solving a transportation problem (*6*).

This paper describes a new EVR algorithm, here called the Sampling and Voting (SV) algorithm. SV moves empty vehicles in anticipation of future passenger requests by analyzing artificial ensembles that represent possible sequences of future passenger requests over a given finite horizon. Each sequence defines a deterministic optimization problem whose (approximate) solution suggests a plan of empty vehicle movements. Features of these plans which are common across the ensemble are then extracted to determine which empty vehicles should actually be moved. Passenger waiting times with the proposed SV algorithm are significantly lower than those for existing methods, in simulation tests.

The terminology and approach used here are based on the literature for vehicle routing problems (VRPs). A classical example (*7*) of a VRP is the distribution of goods from a warehouse to a set of stores; each day, stores request deliveries for the next day, routes are planned over night, and vehicles are dispatched on these routes in the morning. The planning

objective is typically to minimize the cost of delivering the goods. This problem is *static*, because all customer requests are known when the routes are being planned. A *dynamic* VRP (*8*, *9*) arises if `same-day' requests are to be served, because routes must be modified while the system is operating. When probabilistic information about same-day requests is known, the problem is said to be *stochastic*.

The EVR problem is both dynamic (because requests arrive as the system is operating) and stochastic (because the mean rates are known from the demand matrix). The corresponding static problem, in which all passenger requests are known in advance, arises in the SV algorithm, because each sequence in the ensemble generated by SV defines an instance of the static EVR problem. The static problem also provides a means of benchmarking EVR algorithms in terms of passenger waiting time, because performance in the *perfect information* situation always matches or exceeds that where the information is uncertain.

The sampling approach that we adopt has been successfully applied to several dynamic VRP variants related to the EVR problem (*10–12*). The use of the static problem to benchmark the performance of algorithms on the dynamic problem is known as *competitive analysis*, and this has also been done with dynamic VRPs (*13*, *14*). The most relevant VRP variants are the *multivehicle truckload pickup and delivery* problems, where ‘truckload’ means that each vehicle can serve at most one request at a time (*15*), and ‘pickup and delivery’ means that each request has a particular origin and destination (*14*, *16*). The objective in most VRPs is to minimize travel cost, but waiting times may also be considered (*10*); the objective of minimizing average passenger waiting time is known as a *minimum latency* objective (*17*, *18*). Some dynamic problems with minimum latency objectives have also been studied (*19*, *20*). However, none of the VRP variants described above is a perfect match for the EVR problem.

The paper is laid out as follows. In section 2, we introduce the mathematical representation that we use as a simplified model of PRT operations, and we introduce two basic EVR algorithms (one for the dynamic case, one for the static case) that will be incorporated into the SV algorithm, which is described in detail in section 3. Then section 4 compares SV to other EVR algorithms in simulations with two test networks. Finally, section 5 presents conclusions and sets the scene for future work.

2. PRT SYSTEM MODEL AND BASIC EVR ALGORITHMS

We now develop a mathematical representation of PRT operations and use it to introduce two basic EVR algorithms: the Bell and Wong (*21*) Nearest-Neighbors algorithm (BWNN), which operates in a dynamic context, and the Static Nearest-Neighbors algorithm (SNN), which operates in a static context. There are two objectives. Firstly, the BWNN and SNN algorithms are components of the more complicated Sampling and Voting (SV) algorithm that we develop in section 3. Secondly, they are in themselves useful EVR algorithms which can be used to benchmark and evaluate other EVR algorithms.

Since we are mainly concerned with modeling passenger waiting time and how it may be minimized with the use of EVR algorithms, we develop a mathematical model which simplifies and neglects some of the details of real-world PRT system operations. These simplifications have two flavors:

1. Congestion on the guideway is ignored, so that travel times between stations are constant – thus vehicles may always depart upon the quickest path without any delay due to slot-booking processes (*22*, pp. 92–94) etc.

2. PRT system processes at stations are simplified. For example, we neglect spontaneous ride sharing (*23*), and we neglect passenger loading and unloading times. Finally, we assume that the maneuvering order of individual vehicles within stations is unconstrained. See (*2*) for a more detailed discussion of these points.

These assumptions lead to a PRT system model that is essentially the urban taxi model of (*21*). Let S denote the set of stations and K denote the set of vehicles, indexed $1,2,\ldots,n_K$. Let D_{ij} denote the demand in requests per unit time between stations $i \in S$ and $j \in S$, with $D_{ij} = 0$ if $i = j$, and let $T(i,j)$ be the travel time from station i to station j, with $T(i,j) = 0$ if $i = j$.

Each vehicle in the system maintains a list of stations which it must visit in order. Each *trip* thus specified by consecutive elements of the list is either *occupied* (carrying passengers, thus satisfying a request) or *empty* (without passengers, thus specified by the EVR algorithm). Each vehicle's list evolves in time: as each trip is completed, the head of the list is deleted, and more stations are added to the tail of the list as new occupied or empty movements are assigned. If a vehicle completes all of the trips in its list, it becomes *idle* and remains at the last station of its final movement waiting for new trips.

The key discrete events are passenger *requests*. These occur when a passenger (or a preformed group of passengers who intend to travel together in a single vehicle) arrive at station i and request a vehicle to take them to station j. If an empty vehicle is idle at station i, then clearly it should be assigned to the new request. Otherwise, the allocation of a vehicle is more complicated: it may so happen that there are already vehicles en route to station i, but if not, a new empty vehicle trip is required. However, the ideal situation is to move idle vehicles *proactively*, in anticipation of future requests, so that future passengers do not have to wait.

In this model, the chief principle is that trips are completed in order, and vehicles are never rerouted from either empty or occupied movements that have previously been assigned to them. Particularly when the system is busy, rerouting might be beneficial, but it introduces further complications, such as the “indefinite deferment” of requests at outlying stations (*9*); as such, models that allow rerouting are the subject of ongoing research. Because the model used here does not allow rerouting, only the tail of each vehicle's list is important: specifically, we need to consider only the destination station $d[k] \in S$ of the last trip assigned to each vehicle $k \in K$ and the time $a[k]$ at which this trip will be completed. When a vehicle becomes idle, we simply freeze these values so that $a[k]$ denotes the (past) time at which the vehicle became idle and $d[k]$ denotes the station at which it is idle. When a new trip is appended to a vehicle's list, $d[k]$ is updated, and $a[k]$ is recomputed according to when the new trip will be completed.

Bell and Wong Nearest Neighbors (BWNN) Algorithm

This algorithm is the simplest of several algorithms presented by (*21*). It does not make proactive moves, but instead only moves empty vehicles in response to requests. Specifically, when a request is received at time t with origin i and destination j, the vehicle

$$k^* = \underset{k}{\operatorname{argmin}} \left[\max\{0, a[k] - t\} + T(d[k], i) \right] \tag{1}$$

is assigned. In intuitive terms, this choice is locally optimal, because it minimizes the time that the new request waits before a vehicle is made available to serve it. Here the terms on the right-hand side incorporate the time until vehicle k becomes idle and the empty vehicle trip time to reach the new request's origin station, i. This formulation includes the cases where there are vehicles that are either already idle or will become idle at station i. To complete the specification,

a tie-breaking rule is required for when the argmin in (1) is non-unique: for simplicity we choose the minimum such vehicle index *k*.

Figure 1 illustrates the handling of vehicle request lists and the operation of the BWNN algorithm. It also shows that passenger waiting times could be reduced if future requests were known, as is the case with the SNN algorithm described below.

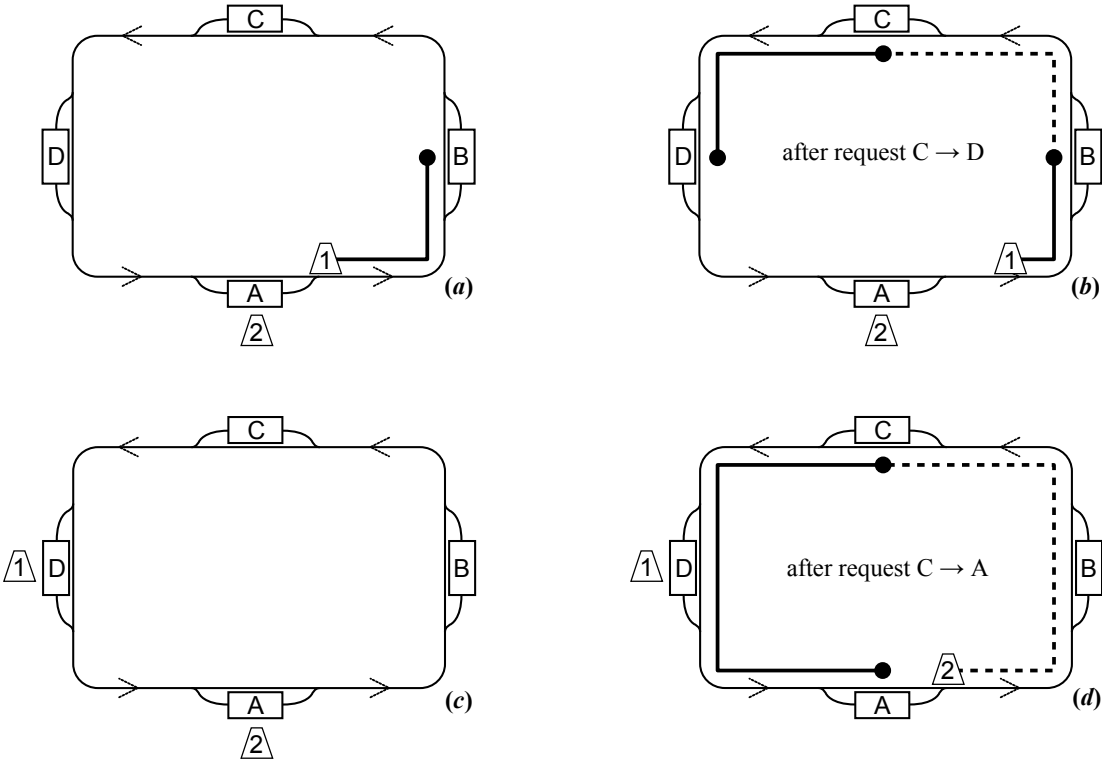

FIGURE 1 Illustration of the BWNN algorithm. There are four off-line stations (labeled A – D) in a ring and two vehicles (labeled 1 and 2). Traffic flow is counter-clockwise. (*a*) Vehicle 1 is initially moving to station B, and vehicle 2 is idle at station A. (*b*) When a request for travel from C to D is received, vehicle 1 is assigned, because it gives a smaller waiting time than vehicle 2. The new request is appended to vehicle 1's request list; this requires an empty vehicle trip (dashed line) from B to C and an occupied trip (solid line) from C to D. Note that, while vehicle 1 stops at station B and station C (filled circles), it does not become *idle* at either station, because it has not finished with its request list. (*c*) However, vehicle 1 does become idle at D, in this case, because no further requests are assigned to it. (*d*) When another request is received from C to A, vehicle 2 is assigned, and it begins an empty trip to C. Vehicle 2 was idle at A, so it could have moved to C proactively, if the request from C had been anticipated; this would have reduced the passenger's waiting time.

Static Nearest Neighbors (SNN) Algorithm

We now modify BWNN to create an algorithm for the *static* EVR problem, in which all requests are known in advance. Let R be a set of requests, indexed $1,2,\ldots,n_R$, and for each request $r \in R$ let i_r and j_r be the origin and destination respectively, and let t_r be the time at which the request is received, with $t_1 < t_2 < \ldots < t_{n_R}$. Like the BWNN algorithm, we step through the requests in ascending order in $r \in R$, but in contrast to BWNN there is no notion of future or past because all of the requests are known simultaneously. For request r we assign the vehicle

$$k^* = \underset{k}{\operatorname{argmin}} \max\{0, a[k] + T(d[k], i_r) - t_r\} \tag{2}$$

and update $a[k^*]$ and $d[k^*]$ accordingly. Like the BWNN rule (1), the SNN rule (2) chooses a vehicle k to minimize the request's waiting time, but with the difference that the empty trip for vehicle k can begin as early as $a[k]$, even if the request is received much later. In other words, the SNN algorithm can move vehicles in advance of requests so that passengers are not kept waiting.

Ties in (2) are common, because there are often several vehicles that could reach i_r before the request at t_r, and all such vehicles would give zero waiting time. To break ties, we first try to select the vehicle with the minimum empty travel time $T(d[k], i_r)$. However, if (for example) there is more than one vehicle inbound to station i_r, they will all have zero empty time and so a further tie-breaking rule is required. To this end, we select the vehicle with latest arrival time $a[k]+T(d[k], i_r)$ at i_r. The idea is that this choice allows better moves as one progresses through the list of requests, because the vehicles that arrive earlier at i_r have more flexibility to serve requests further down the list. Finally, if these measures are equal, we break ties by choosing the minimum such vehicle index k.

With perfect information about future arrivals, SNN tends to yield average waiting times less than those for the dynamic case; quantitative comparison follows in section 4. However, because the routes produced by the SNN algorithm are not provably optimal, other algorithms may outperform SNN for generic patterns of passenger requests. In separate unpublished work, we have found that standard methods for proving the optimality of solutions to the static EVR problem have not been effective for usefully large instances. It has, however, been found (*17*, *18*, *20*) that nearest neighbor heuristics often produce good solutions for other minimum latency routing problems.

3. NEW EVR ALGORITHM: SAMPLING AND VOTING (SV)

The main challenge addressed by the proposed SV algorithm is to move idle vehicles proactively in a dynamic context. The BWNN algorithm moves vehicles reactively in a dynamic context, and the SNN algorithm is effective at proactive movements in a static context, in which all future requests are known. The approach taken here aims to combine the best features of these two algorithms.

When a new request is received at time t, SV assigns a vehicle using the BWNN algorithm. Immediately after a vehicle has been assigned to serve the request, SV may then move idle vehicles proactively. To decide which idle vehicles to move, an *ensemble* of n_E possible sequences of n_R future requests each is generated from the demand matrix. Each sequence in the ensemble, together with the current state of the system, defines an instance of the static EVR problem. Each of these instances is solved approximately using the SNN algorithm, and each resulting solution prescribes a sequence of empty vehicle trips, which constitutes `advice' on which idle vehicles the system should actually move. However, because each solution is computed for a (probably) different sequence of requests, they may offer conflicting advice.

To determine which action should actually be taken, a *voting* system is used. The system adopted here is that at most one idle vehicle at each station may be moved. So, each solution casts one vote on the best destination (as defined below) for an idle vehicle at each station *i* with idle vehicles; note that it may vote for *i*, which means that it votes *not* to move any idle vehicles from *i* at this decision point. If the destination with the most votes is not *i*, an idle vehicle at *i* is selected (breaking ties on minimum vehicle index) and moved.

The destinations to vote for are determined as follows. The solution for each static EVR instance maps each input request to an empty vehicle trip, so each solution yields an ordered sequence of exactly n_R empty trips. Empty trip $p \in \{1,\ldots,n_R\}$ is described by the tuple (i_p, j_p, k_p), where $k_p \in K$ is the index of the vehicle used for trip *p*, and $i_p \in S$ and $j_p \in S$ are the stations at which the empty trip begins and ends, respectively. Trips with $i_p = j_p$ are trivial, in that no actual empty vehicle movement is required. For each station *i* with idle vehicles, let $K_i = \{k \in K : d[k] = i \text{ and } a[k] \le t\}$ be the set of vehicles that are currently idle at *i*, and hence eligible to be moved at the current decision point. The following rules then determine the destination to vote for.

i. vote for *i* if all vehicles in K_i were used for trips *p* with $j_p = i$, or
ii. vote for j_p for the first trip *p* with $k_p \in K_i$ and $j_p \neq i$, if one exists, or
iii. vote for j_p for the first trip *p* with $i_p = i$ and $j_p \neq i$, if one exists, or
iv. vote for *i*.

Rule (i) votes to leave all idle vehicles at *i* if they were all needed there. If an idle vehicle at *i* was moved to another station, rule (ii) votes to perform this trip. If none of the idle vehicles were moved, as is common when demand is light, rule (iii) looks at all trips for a hint at where it should send one of these idle vehicles. If there were no trips from station *i*, rule (iv) leaves them where they are.

The concept of planning with an ensemble is very general, and many variants on the SV algorithm are possible, such as (a) using algorithms other than BWNN and SNN to assign vehicles to requests; (b) running the ensemble generation and voting system at different decision points; (c) using different voting systems; (d) using different rules to choose destinations from each sequence in the ensemble. The SV algorithm described here gave the lowest mean passenger waiting times among several such variants that we have tested in simulation.

4. RESULTS

In this section, the proposed algorithm is evaluated in simulations of two case study systems. The required input data are a network, an origin-destination demand matrix and a fleet size. The main outputs are passenger waiting times and empty vehicle use. The steady state distributions of these outputs are estimated by running long simulations with the demand matrix held constant for each run. Passenger requests from station *i* to station *j* are generated from a Poisson process with rate D_{ij}. For convenience, passenger arrival and travel times are rounded to the nearest integer second.

The observed waiting times strongly depend on how ‘heavy’ the demand is, relative to the network size and the available vehicle fleet. This is made precise using the *intensity* measure from (*2*). Given a network and a demand matrix, the method in (*2*) computes the minimum number of vehicles needed to serve the demand based on the required average flows of both occupied and empty vehicles. The intensity of the demand on the system is defined as the ratio of

the number of vehicles needed to the number of vehicles actually available (n_K). When the intensity is larger than one, passenger requests are arriving faster than they can possibly be served, regardless of the EVR algorithm used, because there are not enough vehicles. The focus here is therefore on demands with intensity less than one. To measure the effects of intensity, an initial demand matrix is scaled up and down to produce a family of demand matrices with the same spatial distribution but different intensities between zero and one.

The input data used here are the 'Grid' and 'Corby' networks and their corresponding demand matrices from (*2*) (for the Grid network, the demand matrix with dispersion parameter $\theta = 0.01$ is used). The fleet size is set at $n_K = 200$ vehicles. Intensity one corresponds to a total demand of 1414 requests/hour for the Corby system and 2035 requests/hour for the Grid system. Simulations have been conducted on nine systems in total, with between 15 and 60 stations, between 50 and 600 vehicles, and total demand at intensity one between 360 and 5050 requests/hour, and the results and conclusions are consistent with those presented here.

For comparison, another EVR algorithm, here called the Surplus / Deficit (SD) algorithm, is also evaluated. It is an algorithm for the dynamic EVR problem that moves vehicles proactively. The general approach in SD is similar to several other published EVR algorithms (*4*); it is most similar to that of (*5*). Each station i has an associated *call time* l_i, which is the cumulative average of all previous empty vehicle trip times to that station (simulations with exponential moving averages gave the same results). The *surplus* of vehicles at station i is the number of inbound vehicles (with $d[k] = i$) minus the expected number of requests over the call time, namely $l_i \sum_j D_{ij}$. When a new request is received, a vehicle is assigned using BWNN. Immediately afterward, SD may move idle vehicles proactively, as follows. For each station i with idle vehicles, in descending order by number of idle vehicles, if the surplus of vehicles at i is greater than or equal to one, an idle vehicle at i is sent to the nearest station with surplus less than zero (if any). Additionally, when a vehicle becomes idle at station i, the above actions are taken for station i only.

Comparison of Algorithms

Figure 2(*a*) compares the mean waiting times observed for the four algorithms on the Corby system. An important observation is that waiting times increase rapidly as intensity approaches one, regardless of which EVR algorithm is used, as is expected based on the definition of intensity. However, the SV algorithm produces the lowest mean waiting times at all intensities tested. In practice, we are most interested in the system's performance at around 70% to 90% intensity, because in this range the system is well-utilized, but acceptably low passenger waiting times may still be obtained. At intensity 0.8, for example, mean waiting times are 355s for BWNN, 41s for SD and 15s for SV, so SV reduces mean waiting times by 96% from BWNN and by 63% from SD. The relative reduction decreases as intensity increases, however, and in fact the SD and SV algorithms become increasingly similar to the BWNN algorithm at higher intensities, because there are fewer idle vehicles to redistribute, as shown in Figure 2(*c*). Figure 2(*c*) also shows that the reduction in passenger waiting times comes from a modest increase in the average number of moving empty vehicles, or equivalently in empty vehicle travel time. The largest increase occurs at intensity 0.91, and this is from 47 concurrently moving empty vehicles with BWNN to 51 with SV (out of 200 vehicles). With perfect information about future arrivals, SNN finds routes with average waiting times less than those for the dynamic case, as expected; at intensity 0.8, the mean waiting time for SNN is 3s. Only mean waiting times are reported, but

the ranking of the four algorithms is the same at the 90th percentile of the waiting distribution; at intensity 0.8, 90% of passengers wait less than 51s with SV and less than 106s with SD.

Results for the Grid network are qualitatively similar, as shown in Figure 2(*b, d*). However, it is notable that for all three algorithms the number of concurrently moving vehicles reaches its maximum (200 = n_K) at around intensity 0.95, which is below the theoretical maximum (intensity 1). It thus appears that nearest neighbor algorithms of the type studied here do not deliver maximum possible throughput; it is not yet known whether there is any practical algorithm that does.

Computation time for the SV algorithm is larger than that for the other algorithms, but SV is still fast enough for real time use with the case study networks. The mean computing time per passenger request for the SV results in Figure 2 is 0.04s (user plus system CPU time on an Intel Xeon E5405 at 2.0GHz with Red Hat g++ 4.1.2 and -O3). The largest system so far tested with SV has 60 stations and 600 vehicles. When n_E = 50 sequences and n_R = 750 generated requests / sequence, SV produces lower waiting times than the SD and BWNN methods on this system, and it uses 1.2s of computing time per passenger request. For this system, the demand at intensity one is 5050 requests/hour, or 0.7s per request, so the *sequential* SV algorithm used for testing is not fast enough for real time use at high intensity, in this case. However, SV is easy to parallelize, because each of the n_E sequences can be generated and processed independently; for example, real elapsed time per request could in this case be reduced to near 0.6s if two processors were used. The main limit on the scalability of the SV algorithm is thus the time to process a single sequence. The SNN minimization (2) takes $O(n_K)$ time, so processing a single sequence takes $O(n_K n_R)$ time.

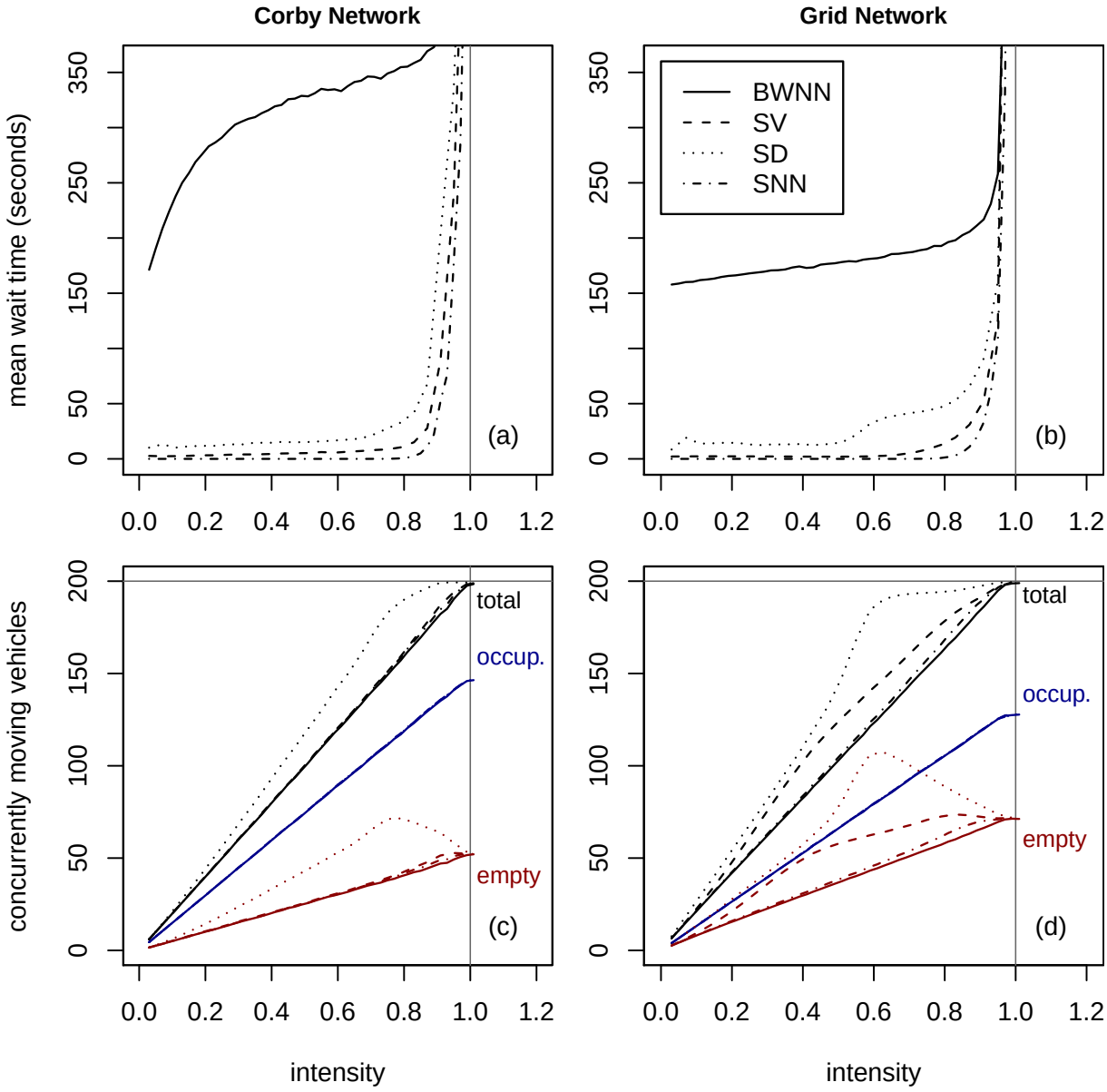

FIGURE 2 Mean passenger waiting times (*a*, *b*) and vehicle fleet utilization (*c*, *d*) for the Grid and Corby networks for the four algorithms. The proposed SV algorithm moves idle vehicles in anticipation of future requests, which reduces waiting times significantly below the BWNN baseline and the SD algorithm. The SNN algorithm operates with perfect information about future requests in order to estimate how much further waiting times might be reduced. Here there are n_E = 50 sequences, each with n_R = 300 requests for SV. Each point is averaged over 10 independent runs of 50,000 simulated passengers each.

Effect of Number and Length of Ensemble Sequences on SV

The SV algorithm has two free parameters. Figure 3 shows the effects of varying the number of sequences n_E, and Figure 4 shows the effects of varying the number of requests, n_R, in each sequence.

Figures 3(*a, b*) show that waiting times are not very sensitive to the number of sequences used. On the Corby system, a single sequence is sufficient to reduce waiting times well below the BWNN baseline. However, Figures 3(*c, d*) show that using more sequences produces some further reductions in waiting time and also significant reductions in empty vehicle travel. For example, on the Grid system at intensity 0.4 with a single sequence, 65% of moving vehicles are empty on average, but this falls to 45% when 50 sequences are used. This is because a small ensemble is less likely to be representative of the actual demand, and also because the actions recommended by a small ensemble are more likely to change at each decision point. So, while a small ensemble will sometimes make good recommendations that reduce passenger waiting times, it is also likely to make bad recommendations that result in wasted empty vehicle trips.

Figures 4(*a, b*) show that waiting times decrease for all intensities as n_R increases. A sequence with more requests generates more occupied and empty vehicle trips, which makes it more likely that the sequence will vote on what to do with an idle vehicle, rather than just leaving it idle. This results in more empty vehicle movement, but also lower waiting times. For example, at intensity 0.8 on the Grid network, increasing n_R from 100 to 200 increases the fraction of moving vehicles that are empty from 36% to 40%, and it decreases mean passenger waiting times from 46s to 18s.

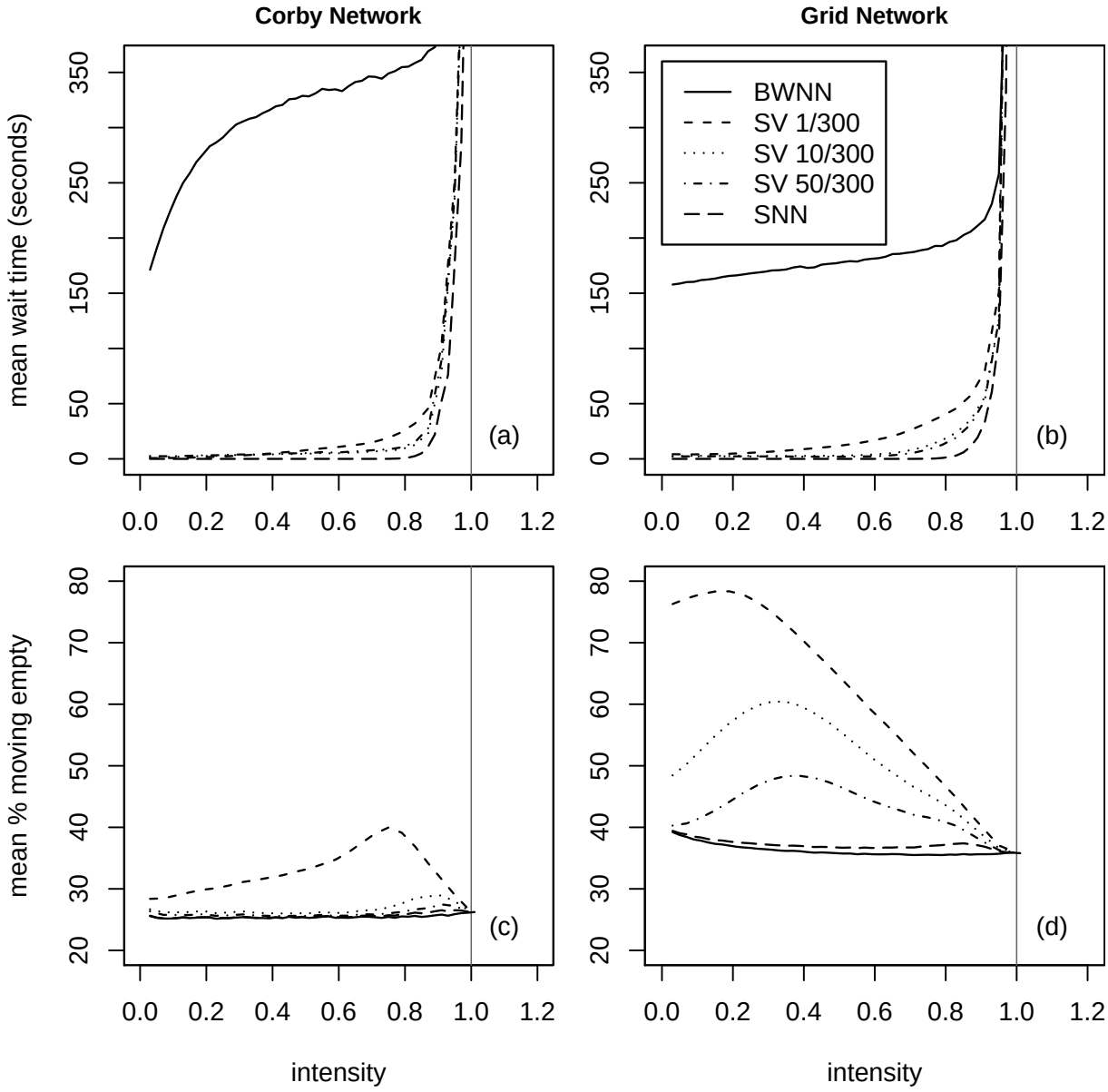

FIGURE 3 Effect of the number of sequences (n_E) on mean passenger waiting times (*a, b*) and the fraction of moving vehicles which are empty (*c, d*). Even a single sequence ($n_E = 1$) is sufficient to significantly reduce waiting times below the BWNN baseline, but using more sequences reduces the amount of empty vehicle travel required. Here $n_R = 300$ requests per sequence. Each point is averaged over 10 independent runs of 50,000 simulated passengers each.

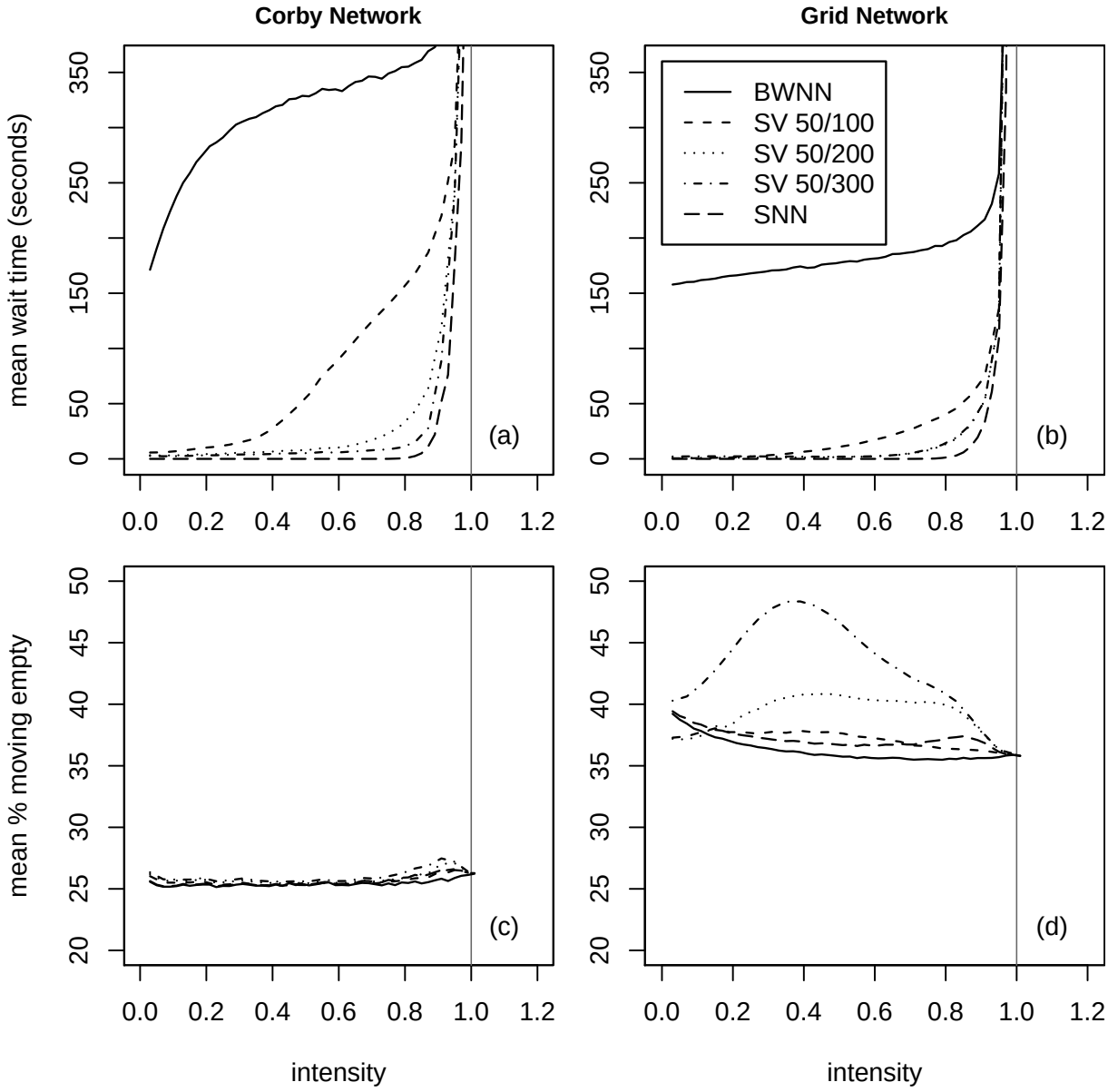

FIGURE 4 Effect of sequence length (n_R) on mean waiting times (*a, b*) and the fraction of moving vehicles which are empty (*c, d*). Longer sequences (with more requests) reduce passenger waiting times at the cost of modest increases in empty vehicle running. Here n_E = 50 sequences. Each point is averaged over 10 independent runs of 50,000 simulated passengers each.

5. CONCLUSIONS

The Sampling and Voting (SV) algorithm developed in this paper is a new algorithm for solving the dynamic Empty Vehicle Redistribution (EVR) problem for Personal Rapid Transit systems. Variants of SV might also be applied to manage fleets of conventional taxis or emergency response vehicles. Average passenger waiting times must diverge as the demand approaches capacity, regardless of the EVR algorithm used. However, the SV algorithm allows the system to maintain a good service level through a larger part of the intensity range than other algorithms in the literature, albeit with some additional computational cost.

Simulation results show that moving idle vehicles proactively is key to providing low waiting times, because this is what the SV algorithm does that the baseline Bell and Wong Nearest Neighbors (BWNN) algorithm does not. This entails some risk of making wasted empty vehicle trips, but, in the case studies considered here, SV yields large relative reductions (up to 96%) in waiting time with smaller relative increases in empty vehicle travel time, particularly at high intensities (around 22% at intensity 0.8 on the Grid network, for example). This increase in empty vehicle travel contributes to network congestion, which is not modeled here. In a system operating near its congested limit, other algorithms (or variants of SV that do model congestion) might perform better than SV as described here.

The Static Nearest Neighbors (SNN) algorithm gives an approximate solution to the static EVR problem, in which there is perfect information of all future passenger requests. We have thus used the SNN algorithm to give a rough estimate of the potential for further waiting time reductions for the dynamic EVR problem, in which only the mean request rates are known. For example, such reductions might be obtained by allowing empty vehicles to be rerouted, or by the use of empty vehicle sidings (or *car barns* (*4*)), which act as buffers for empty vehicles. SV (as currently defined) cannot deal with empty vehicle sidings, because they are effectively stations with no requests, which thus receive no vehicles. Finally, improvements may be possible in the details of the sampling algorithm, as outlined in section 3.

REFERENCES

1. ULTra PRT. ULTra at London Heathrow Airport. www.ultraprt.com/applications/existing-systems/heathrow. Accessed July 31, 2010.

2. Lees-Miller, J. D., J. C. Hammersley and R. E. Wilson. Theoretical Maximum Capacity as a Benchmark for Empty Vehicle Redistribution in Personal Rapid Transit. In *Transportation Research Record: Journal of the Transportation Research Board*. No. 2146, Transportation Research Board of the National Academies, Washington, D.C., 2010.

3. Andréasson, I. Vehicle Distribution in Large Personal Rapid Transit Systems. In *Transportation Research Record: Journal of the Transportation Research Board*. No. 1451, Transportation Research Board of the National Academies, Washington, D.C., 1994, pp. 95–99.

4. Anderson, J. E. Control of personal rapid transit systems. *Journal of Advanced Transportation*, vol. 32, no. 1, 1998.

5. Andréasson, I. Quasi-Optimum Redistribution of Empty PRT Vehicles. In *Automated People Movers VI: Creative Access for Major Activity Centers*, W. J. Sproule, E. S. Neumann and S. W. Lynch, eds. American Society of Civil Engineers, Reston, VA, April 1998, pp. 541–550.

6. Andréasson, I. Reallocation of empty personal rapid transit vehicles en route. In *Transportation Research Record: Journal of the Transportation Research Board*. No. 1838, Transportation Research Board of the National Academies, Washington, D.C., 2003, pp. 36–41.

7. Toth, P. and D. Vigo. *The vehicle routing problem*. SIAM monographs on discrete mathematics and applications, Society for Industrial and Applied Mathematics, 2002.

8. Gendreau, M. and J.-y. Potvin. Dynamic Vehicle Routing and Dispatching. In *Fleet management and logistics*, T. G. Crainic and G. Laporte, eds., Kluwer, Boston, chap. 5, 1998, pp. 115–126.

9. Psaraftis, H. N. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, vol. 61, no. 1, December 1995, pp. 143–164.

10. Yang, J., P. Jaillet and H. Mahmassani. Real-Time Multivehicle Truckload Pickup and Delivery Problems. *Transportation Science*, vol. 38, no. 2, May 2004, pp. 135–148.

11. Bent, R. W. and P. Van Hentenryck. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Operations Research*, vol. 52, no. 6, 2004, pp. 977–987.

12. Hvattum, L. M., A. Løkketangen and G. Laporte. Solving a Dynamic and Stochastic Vehicle Routing Problem with a Sample Scenario Hedging Heuristic. *Transportation Science*, vol. 40, no. 4, November 2006, pp. 421–438.

13. Mitrovic-Minic, S., R. Krishnamurti and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, vol. 38, no. 8, September 2004, pp. 669–685.

14. Berbeglia, G., J.-F. Cordeau and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, vol. 202, no. 1, April 2010, pp. 8–15.

15. Powell, W. B. A Stochastic Formulation of the Dynamic Assignment Problem, with an Application to Truckload Motor Carriers. *Transportation Science*, vol. 30, no. 3, August 1996, pp. 195–219.

16. Berbeglia, G., J.-F. Cordeau, I. Gribkovskaia and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP*, vol. 15, no. 1, July 2007, pp. 131.

17. Fischetti, M., G. Laporte and S. Martello. The Delivery Man Problem and Cumulative Matroids. *Operations Research*, vol. 41, no. 6, 1993, pp. 1055–1064.

18. Larsen, A., O. B. G. Madsen and M. M. Solomon. The A Priori Dynamic Traveling Salesman Problem with Time Windows. *Transportation Science*, vol. 38, no. 4, 2004, pp. 459–472.

19. Bertsimas, D. J. and D. S. Levi. A New Generation of Vehicle Routing Research: Robust Algorithms, Addressing Uncertainty. *Operations Research*, vol. 44, no. 2, 1996, pp. 286–304.

20. Swihart, M. R. and J. D. Papastavrou. A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, vol. 114, no. 3, May 1999, pp. 447–464.

21. Bell, M. G. H. and K. I. Wong. A Rolling Horizon Approach to the Optimal Dispatching of Taxis. In *Transportation and traffic theory : Flow, dynamics and human interaction : proceedings of the 16th International Symposium on Transportation and Traffic Theory*, H. S. Mahmassani, ed. Elsevier, Amsterdam, July 2005, pp. 629–648.

22. Irving, J. H., H. Bernstein, C. L. Olson and J. Buyan. *Fundamentals of Personal Rapid Transit*. Lexington Books, D.C. Heath and Company, 1978.

23. Lees-Miller, J., J. Hammersley and N. Davenport. Ride Sharing in Personal Rapid Transit Capacity Planning. In *Automated People Movers 2009*, R. R. Griebenow, ed.