Math Meth Oper Res
DOI 10.1007/s00186-017-0583-3

ORIGINAL ARTICLE

Scheduling for a processor sharing system with linear slowdown

Liron Ravner[1] · **Yoni Nazarathy**[2]

Received: 21 July 2016

Abstract We consider the problem of scheduling arrivals to a congestion system with a finite number of users having identical deterministic demand sizes. The congestion is of the processor sharing type in the sense that all users in the system at any given time are served simultaneously. However, in contrast to classical processor sharing congestion models, the processing slowdown is proportional to the number of users in the system at any time. That is, the rate of service experienced by all users is linearly decreasing with the number of users. For each user there is an ideal departure time (due date). A centralized scheduling goal is then to select arrival times so as to minimize the total penalty due to deviations from ideal times weighted with sojourn times. Each deviation penalty is assumed quadratic, or more generally convex. But due to the dynamics of the system, the scheduling objective function is non-convex. Specifically, the system objective function is a non-smooth piecewise convex function. Nevertheless, we are able to leverage the structure of the problem to derive an algorithm that finds the global optimum in a (large but) finite number of steps, each involving the solution of a constrained convex program. Further, we put forward several heuristics. The first is the traversal of neighbouring constrained convex programming problems, that is guaranteed to reach a local minimum of the centralized problem. This is a form of a "local search", where we use the problem structure in a novel manner. The second is a one-coordinate "global search", used in coordinate pivot iteration. We then merge these two heuristics into a unified "local–global" heuristic, and numerically illustrate the effectiveness of this heuristic.

✉ Yoni Nazarathy
y.nazarathy@uq.edu.au

[1] Department of Statistics and the Federmann Center for the Study of Rationality, The Hebrew University of Jerusalem, Jerusalem, Israel

[2] School of Mathematics and Physics, The University of Queensland, Brisbane, Australia

Published online: 20 March 2017

Keywords Scheduling · Road traffic · Global optimization

1 Introduction

Users of shared resources are frequently faced with the decision of when to use the resource with a view of trying to avoid rush hour effects. Broad examples include, workers taking their lunch break and attending a cafeteria; people entering and vacating sporting events; and commuters using transportation networks. In many such situations the so called rush-hour game is played by all users acting individually. On the one hand, each user typically has an ideal arrival/departure time, while on the other hand, users often wish to avoid rush hour so as to minimise congestion costs. These general types of scenarios have received much attention through the transportation community (Arnott et al. 1993), the queueing community (see Glazer and Hassin 1983 or p. 84 of Hassin 2016 for a review) and more specifically within the setting we consider in this paper (Ravner et al. 2016).

While understanding social strategic (game) behaviour is important, a complementary analysis is with regards to the social optimum (centralised scheduling decisions). These types of situations occur often in manufacturing, appointment scheduling, education and service. Most of the research on scheduling methodology does not consider processor sharing but rather focuses on the situation where resources are dedicated, see Pinedo (2008). In this paper, we put forward a novel scheduling model, that offers a simple abstraction of a common scenario: Jobs may be scheduled simultaneously, yet slow each other down when sharing the resource. In this respect our model is related to the study of scheduling problems with batch processing, see Potts and Kovalyov (2000). However, from a mathematical perspective, our model, results and methods do not involve the classical discrete approaches but rather rely on piecewise affine dynamics with breakpoints. This type of behaviour resembles Separated Continuous Linear Programs, as in Weiss (2008), and is often used to solve optimization problems associated with fluid multi-class queueing networks (cf. Avram et al. 1995; Nazarathy and Weiss 2009).

A standard way of modelling resource sharing phenomena, is the so-called processor sharing queue, see for example Harchol-Balter (2013). In such a model, given that at time t there are $q(t)$ users in the system, the total fixed service capacity, $\beta > 0$, is allocated, such that each user receives an instantaneous *service rate*,

$$v\big(q(t)\big) = \frac{\beta}{q(t)}. \tag{1.1}$$

Such a model then captures the relationship of the arrival time of a user, a, the departure time of a user, d and the service demand, ℓ through

$$\ell = \int_a^d v\big(q(t)\big)dt.$$

The *aggregate throughput* with q users in the system is the product $q\, v(q)$. For the processor sharing model (1.1), this is obviously the constant β. However, in practice,

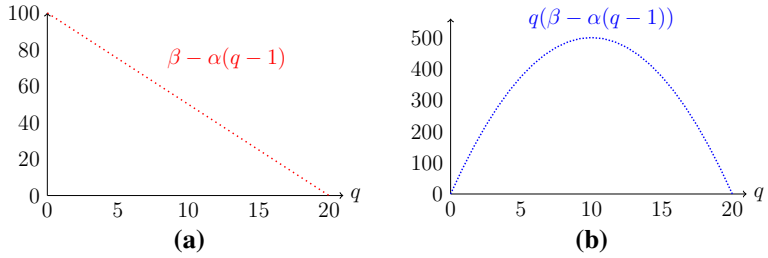

Fig. 1 The service rate and aggregate throughput as a function of the number of users in the system. Parameter values: $\beta = 100$ and $\alpha = 5$. **a** Service rate, **b** aggregate throughput

the aggregate throughput is not necessarily constant with respect to $q(t)$. In many situations, most notably in traffic and transportation scenarios, users inter-play in a complicated manner. In particular, in the classic Greenshield fluid model (see for example Henderson 1974 or Mahmassani and Herman 1984), the aggregate throughput is not monotone in the number of users and even exhibits a traffic jam effect. The simplest model, describing such a phenomenon is

$$v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \tag{1.2}$$

which is a discrete variation of Greenshield's model.[1] With a single user in the system, (1.2) yields the *free flow rate* β which coincides with (1.1). Then for each additional user, there is a linear slowdown of $\alpha > 0$ units in the rate. See Fig. 1 for a simple illustration. Note that in road networks, much research has focused on the so-called fundamental diagram for networks, such as in Daganzo (2007). Indeed Fig. 1b resembles a fundamental diagram.

Our scheduling problem is to centrally choose arrival times $\mathbf{a} = (a_1, \ldots, a_N)'$ in an effective manner, where N is the number of users. In this paper we assume that all users share the same service demand, ℓ. In our objective, user i incurs a cost of

$$(d_i - d_i^*)^2 + \gamma\,(d_i - a_i),$$

where d_i is his departure time and d_i^* is the ideal departure time (due date) and γ captures tradeoff between meeting the due date and sojourn time costs. The total costs incurred by all users is then the sum of individual user costs.

If there was no congestion (say due to d_i^* being well separated), an ideal choice is $a_i = d_i^* - \ell/\beta$. But in general, users interact, so the scheduling decision needs to take this interaction into account. If, for example, $\gamma = 0$ and $d_i^* = d^*$ for all i, then the problem is trivially solved with zero cost by setting

[1] Note that in queueing theory, situations where $v(\cdot)$ is not as in (1.1) but is rather some other function are sometimes referred to as generalized processor sharing. See for example Cohen (1979). Generalized processor sharing has also taken other meanings over the years, so sometimes there is confusion about the term.

$$a_i = d^* - \frac{\ell}{\beta - \alpha(N-1)}.$$

Here since sojourn time does not play a role, sending all users simultaneously will imply they arrive simultaneously after being served together at the slowest possible rate. Continuing with the case of $\gamma = 0$, if now users do not have the same d_i^*, then attaining zero costs is still possible. In fact, we show in the sequel, that in this specific case ($\gamma = 0$) the optimal schedule can be computed efficiently (in polynomial time).

At the other extreme consider the case where minimising sojourn times is prioritised over minimisation of due dates (e.g. if fuel costs are extremely high). This corresponds to $\gamma \approx \infty$. While for any finite γ, it is possible that an optimal schedule allows overlap of users, an approximation for the case of large γ is obtained by enforcing a schedule with no overlap ($q(t) \leq 1 \ \forall t$). This is because overlaps have a very large sojourn time cost relative to the possible reduction in quadratic deviation from desired departure times. Now with such a constraint, the problem resembles a single machine scheduling problem with due date penalties. This problem has been heavily studied (see for example Baker and Scudder 1990 or Sen and Gupta 1984). In our case, in which users have identical demand, finding the optimal schedule is a convex quadratic program and can thus be solved in polynomial time. We spell out the details in the sequel.

Setting aside the extreme cases of $\gamma = 0$ or $\gamma \approx \infty$, the problem is more complicated. While we do not have an NP-hardness proof, we conjecture that finding the optimal $\mathbf{a}$ is a computationally challenging problem. In the current paper we handle this problem in several ways. First we show that departure times depend on arrival times in a piecewise affine manner. We find an efficient algorithm for calculating $d_i(\mathbf{a})$. We then show that the total cost is a piecewise convex quadratic function but generally not convex, i.e. there is a large (but finite) number of polytopes in $\mathbb{R}^N$ where within each polytope, it is a convex quadratic function of $\mathbf{a}$. This is a similar formulation to that of the piecewise-linear programming problem presented in Vielma et al. (2010), which is known to be NP-hard. The structure of the total cost yields an *exhaustive search* scheduling algorithm which terminates in finite time.

We then put forward heuristics. The first heuristic, which we refer to as the *local search*, operates by solving a sequence of neighbouring quadratic problems until finding a local minimum with respect to the global optimization. The second heuristic performs a *global search* over one coordinate (arrival time of a single user), keeping other coordinates fixed. This is done in a provably efficient manner. In particular, we bound the number of steps in each coordinate search by a polynomial. It then repeats over other coordinates, cycling over all coordinates until no effective improvement in the objective function is possible. In case of smooth objectives, it is known that such coordinate pivot iterations (CPI) schemes converge to local minima (see for example Bertsekas 1999, p. 272). Further, in certain special cases of non-smooth objectives, it is also known that CPI schemes converge to local minima (see for example Tseng 2001). But in our case, the non-separable piecewise structure of the objective often causes our heuristic to halt at a point that is not a local minimum. Nevertheless, the global search heuristic is fruitful when utilized in a *combined local–global search* heuristic. This heuristic performs global searches with different initial points, each followed by

a local search. We present numerical evidence, illustrating that it performs extremely well. Often finding the global optimum in very few steps.

The structure of the sequel is as follows. In Sect. 2 we present the model and basic properties. In Sect. 3 we focus on arrival departure dynamics, showing a piecewise affine relationship between the arrival and departures times. We give an efficient algorithm for calculating the departure times given arrival times or vice-versa. This also solves the scheduling problem for the special case $\gamma = 0$. In Sect. 4 we characterise the constraints associated with quadratic programs which make up the piecewise quadratic cost. These are then used in the *exhaustive search* algorithm. We then present the *local search* algorithm and prove it always terminates at a local minimum (of the global objective). In Sect. 5 we present our global search method based on CPI. We utilize the structure of the problem to obtain an efficient single coordinate search within the CPI. Then in Sect. 6, the local search and global searches are combined into a unified heuristic. We further illustrate the power of our heuristic through numerical examples. We conclude in Sect. 7. Some of the proofs are deferred to the "Appendix".

Notation We denote $x \wedge y$ and $x \vee y$ to be the minimum and maximum of x and y, respectively. We define any summation with initial index larger than the final index to equal zero (e.g. $\sum_{i=2}^{1} a_i = 0$). Vectors are taken as columns and are denoted in bold. $\mathbf{1} \in \mathbb{R}^N$ denotes a vector of 1's and $\mathbf{e_i} \in \mathbb{R}^N$ denotes a vector of zeros in all but the i'th coordinate, which equals 1. The indicator function is denoted by $\mathbb{1}$.

2 Model

Our model assumes that there is a fixed user set $\mathcal{N} = \{1, \ldots, N\}$ where the service requirement of each user, ℓ, is the same and is set to 1 without loss of generality (this can be accounted for by changing the units of β and α). Then the equations determining the relationship between the arrival times vector $\mathbf{a} = (a_1, \ldots, a_N)'$ and the departure times vector $\mathbf{d} = (d_1, \ldots, d_N)'$ are

$$1 = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad \text{where} \quad q(t) = \sum_{j \in \mathcal{N}} \mathbb{1}\{t \in [a_j, d_j]\}. \tag{2.1}$$

Using the linear slowdown rate function, (1.2), the equations are represented as,

$$1 = \int_{a_i}^{d_i} \beta - \alpha \left(\sum_{j \in \mathcal{N}} \mathbb{1}\{t \in [a_j, d_j]\} - 1 \right) dt, \quad i = 1, \ldots, N. \tag{2.2}$$

These N equations can be treated as equations for the unknowns $\mathbf{d}$, given $\mathbf{a}$ or vice-versa. We assume $N < \beta/\alpha + 1$ so that it always holds that $v\big(q(t)\big) > 0$.

The cost incurred by user i is,

$$c_i(a_i, d_i) = (d_i - d_i^*)^2 + \gamma\,(d_i - a_i), \tag{2.3}$$

and the total cost function, which we seek to minimise, is

$$c(\mathbf{a}) = \sum_{i \in \mathcal{N}} c_i\big(a_i, d_i(\mathbf{a})\big). \tag{2.4}$$

We assume (without loss of generality) that the ideal departure times, $\mathbf{d}^* = (d_1^*, \ldots, d_N^*)'$ are ordered, i.e. $d_1^* \leq \cdots \leq d_N^*$.

Remark For clarity of the exposition we choose the cost, (2.3) to be as simplistic as possible. Practical straightforward generalizations to the cost and to the associated algorithms and heuristics are discussed in the conclusion of the paper. These include other convex penalty functions, ideal arrival times and a potentially different penalty for early and late departures. Our algorithms, can all be adapted for such cost functions.

We first have the following elementary lemmas:

Lemma 2.1 *Assume that the arrivals,* $\mathbf{a}$*, are ordered:* $a_1 \leq a_2 \leq \cdots \leq a_N$*, then the departures,* $\mathbf{d}$*, follow the same order:* $d_1 \leq d_2 \leq \cdots \leq d_N$*.*

Lemma 2.2 *For any* $\mathbf{a}$ *there is a unique* $\mathbf{d}$ *and vice-versa.*

As a consequence of the assumed order of $\mathbf{d}^*$ and of the above lemma we assert that an optimal schedule can only be attained with an ordered $\mathbf{a}$ whose individual coordinates lie in a compact interval, as shown in the following lemma.

Lemma 2.3 *An optimal arrival schedule satisfies* $\underline{a} \leq a_1 \leq \cdots \leq a_N \leq \overline{a}$*, where*

$$\underline{a} = d_1^* - \frac{N}{\beta - \alpha(N-1)}, \quad \overline{a} = d_N^* + \frac{N}{\beta - \alpha(N-1)}.$$

We may thus define the search region for the optimal schedule:

$$\mathcal{R} = \left\{ \mathbf{a} \in \mathbb{R}^N : \underline{a} \leq a_1 \leq \cdots \leq a_N \leq \overline{a} \right\},$$

and take our scheduling problem to be $\min_{\mathbf{a} \in \mathcal{R}} c(\mathbf{a})$.

No strict condition on the joint order of a_i and d_i can be imposed except for the requirement that $a_i < d_i$ for any i (the sojourn time of all users is strictly positive). We are thus motivated to define the following for $i \in \mathcal{N}$:

$$k_i := \max\left\{k \in \mathcal{N} : a_k \leq d_i\right\} = \min\left\{k \in \mathcal{N} : a_{k+1} > d_i\right\}, \tag{2.5}$$

$$h_i := \min\left\{h \in \mathcal{N} : d_h \geq a_i\right\} = \max\left\{h \in \mathcal{N} : d_{h-1} < a_i\right\}. \tag{2.6}$$

The variable k_i specifies the interval $[a_{k_i}, a_{k_i+1})$ in which d_i resides. Similarly the variable h_i specifies that a_i lies in the interval $(d_{h_i-1}, d_{h_i}]$. Note that we define $a_0,\ d_0 := -\infty$ and $a_{N+1},\ d_{N+1} := \infty$. The sequences k_i and h_i satisfy some basic properties: (1) They are non-decreasing and are confined to the set $\mathcal{N}$. (2) From the fact that $a_i < d_i$ we have that $i \leq k_i$. (3) Since $\mathbf{d}$ is an ordered sequence and also $a_i < d_i$

we have $h_i \leq i$. (4) We have $h_1 = 1$ and $k_N = N$. (5) Each sequence determines the other:

$$k_i = \max\left\{k \in \mathcal{N} : h_k \leq i\right\}, \quad \text{and} \quad h_i = \min\left\{h \in \mathcal{N} : k_h \geq i\right\}.$$

Thus given either the sequence $k_i,\ i \in \mathcal{N}$ or the sequence $h_i,\ i \in \mathcal{N}$ or both, the ordering of the $2N$ tuple $(a_1, \ldots, a_N, d_1, \ldots, d_N)$ is fully specified as long as we require that a_i's and d_i's are ordered so as to be consistent with Lemmas 2.1 and 2.3.

We denote the set of possible $\mathbf{k} = (k_1, \ldots, k_N)'$ by

$$\mathcal{K} := \left\{\mathbf{k} \in \mathcal{N}^N : k_N = N,\ k_i \leq k_j\ \forall i \leq j\right\}. \tag{2.7}$$

Similarly, we denote the set of possible $\mathbf{h} = (h_1, \ldots, h_N)'$ by $\mathcal{H}$. We have that,

$$|\mathcal{K}| = |\mathcal{H}| = \frac{\binom{2N}{N}}{N+1}.$$

This follows (for example) by observing that the elements of $\mathcal{K}$ correspond uniquely to lattice paths in the $N \times N$ grid from bottom-left to top-right with up and right movements without crossing the diagonal. The number of such elements is the Nth Catalan number, see for example p. 259 in Koshy (2009).

The following example illustrates the dynamics of the model (without optimization) and shows the role of $\mathbf{k}$, or alternatively $\mathbf{h}$, in summarizing the piecewise affine dynamics.

Example 2.1 Take $\beta = 1/2$, $\alpha = 1/6$ and $N = 3$. This 3 user system exhibits rates that are either $1/2$, $1/3$ or $1/6$ depending on the number of users present. The free flow sojourn time is $1/\beta = 2$. Assume $a_1 = 0$, $a_2 = 1$ and $a_3 = 3$. We now describe the dynamics of the system. See also Fig. 2.

During the time interval $[0, 1)$, $q(t) = 1$ and the first user is being served at rate $1/2$. By time $t = 1$ the remaining service required by that user is $1/2$. At time $t = 1$, the number of users in the system, $q(t)$, grows to 2 and the rate of service to each user is reduced to $1/3$. This means that without a further arrival causing further slowdown, user 1 is due to leave at time $t = 2.5$. Since $2.5 < a_3$, this is indeed the case. At $t = 2.5$, $q(t)$ changes from 2 to 1. By that time, the remaining service required by user 2 is $1/2$. Then during the time interval $[2.5, 3)$ user 2 is served at rate $1/2$ reducing the remaining service of that user to $1/4$. At time $t = 3$, user 3 joins, increasing $q(t)$ back to 2 and reducing the service rate again to $1/3$. User 2 then leaves at time $t = 3.75$ and as can be verified using the same types of simple calculations, user 3 finally leaves at time $t = 5.25$.

Observe that for this example, the order of events is:

$$a_1 \leq a_2 \leq d_1 \leq a_3 \leq d_2 \leq d_3.$$

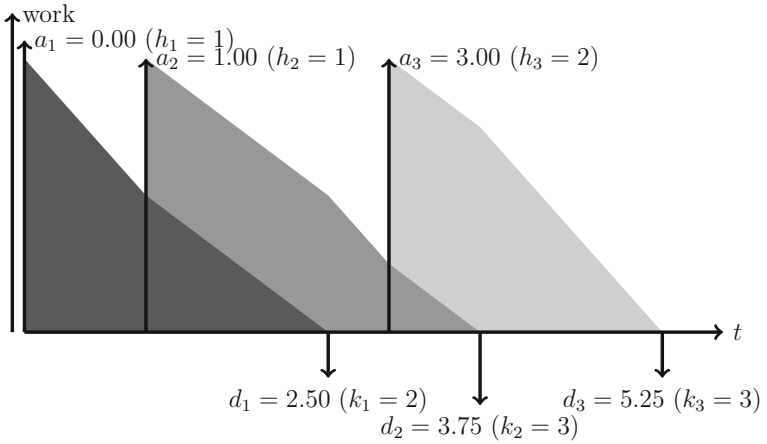

Fig. 2 An illustration of the dynamics of a three user example. The *shaded gray areas* show the remaining work for each individual user. Work is depleted at rate $\frac{1}{2}$ when only one user is present and is depleted at the slower rate of $\frac{1}{3}$ when two users are present

This then implies that for this schedule,

$$k_1 = 2,\ k_2 = 3,\ k_3 = 3, \quad \text{and} \quad h_1 = 1,\ h_2 = 1,\ h_3 = 2.$$

3 Arrival departure dynamics

We now investigate the relationship between arrivals and departures, induced by the linear slowdown dynamics.

Proposition 3.1 *Equation* (2.2) *can be expressed as*

$$(\beta - \alpha(k_i - i))\, d_i - \alpha \sum_{j=h_i}^{i-1} d_j - (\beta - \alpha(i - h_i))\, a_i + \alpha \sum_{j=i+1}^{k_i} a_j = 1, \quad i \in \mathcal{N}, \tag{3.1}$$

or alternatively,

$$D\,\mathbf{d} - A\,\mathbf{a} = \mathbf{1}, \tag{3.2}$$

with the matrices $A \in \mathbb{R}^N$ *and* $D \in \mathbb{R}^N$ *defined as follows:*

$$A_{ij} := \begin{cases} \beta - \alpha(i - h_i), & i = j, \\ -\alpha, & i+1 \le j \le k_i, \\ 0, & o.w. \end{cases} \qquad D_{ij} := \begin{cases} \beta - \alpha(k_i - i), & i = j, \\ -\alpha, & h_i \le j \le i-1, \\ 0, & o.w. \end{cases}$$

Proof We manipulate (2.2) to get,

$$\begin{aligned}
1 &= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{N} \int_{a_i}^{d_i} \mathbb{1}\{t \in [a_j, d_j]\} dt \\
&= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{N} \left(d_i \wedge d_j - a_i \vee a_j\right)^+ \\
&= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{i-1} \left(d_i \wedge d_j - a_i \vee a_j\right)^+ - \alpha(d_i - a_i) \\
&\quad - \alpha \sum_{j=i+1}^{N} \left(d_i \wedge d_j - a_i \vee a_j\right)^+ . \\
&= \beta(d_i - a_i) - \alpha \sum_{j=1}^{i-1} \left(d_i \wedge d_j - a_i \vee a_j\right)^+ - \alpha \sum_{j=i+1}^{N} \left(d_i \wedge d_j - a_i \vee a_j\right)^+ .
\end{aligned}$$

where in the third step we have used the fact that $a_i < d_i$ for the term corresponding to $j = i$. We now use the fact that $\mathbf{a}$ and $\mathbf{d}$ are both ordered to get,

$$\begin{aligned}
1 &= \beta(d_i - a_i) - \alpha \sum_{j=1}^{i-1} \left(d_j - a_i\right)^+ - \alpha \sum_{j=i+1}^{N} \left(d_i - a_j\right)^+ \\
&= \beta(d_i - a_i) - \alpha \sum_{j=1}^{i-1} (d_j - a_i \wedge d_j) - \alpha \sum_{j=i+1}^{N} (d_i - a_j \wedge d_i) \\
&= -\beta a_i + (\beta - \alpha(N-i)) d_i - \alpha \sum_{j=1}^{i-1} d_j + \alpha \sum_{j=1}^{i-1} (a_i \wedge d_j) + \alpha \sum_{j=i+1}^{N} (a_j \wedge d_i).
\end{aligned}$$

Now the summations $\sum_{j=1}^{i-1}(a_i \wedge d_j)$ and $\sum_{j=i+1}^{N}(a_j \wedge d_i)$ can be broken up as follows:

$$\begin{aligned}
\sum_{j=1}^{i-1} (a_i \wedge d_j) &= \sum_{j=1}^{i-1} \mathbb{1}\{d_j < a_i\} d_j + \sum_{j=1}^{i-1} \mathbb{1}\{d_j \geq a_i\} a_i \\
&= \sum_{j=1}^{h_i - 1} d_j + \sum_{j=h_i}^{i-1} a_i = \sum_{j=1}^{h_i - 1} d_j + (i - h_i) a_i , \\
\sum_{j=i+1}^{N} (a_j \wedge d_i) &= \sum_{j=i+1}^{N} \mathbb{1}\{a_j > d_i\} d_i + \sum_{j=i+1}^{N} \mathbb{1}\{a_j \leq d_i\} a_j \\
&= \sum_{j=k_i+1}^{N} d_i + \sum_{j=i+1}^{k_i} a_j = (N - k_i) d_i + \sum_{j=i+1}^{k_i} a_j .
\end{aligned}$$

Combining the above we obtain:

$$1 = -(\beta - \alpha(i - h_i))a_i + (\beta - \alpha(k_i - i))d_i - \alpha\left(\sum_{j=1}^{i-1} d_j - \sum_{j=1}^{h_i-1} d_j\right) + \alpha \sum_{j=i+1}^{k_i} a_j$$
$$= -(\beta - \alpha(i - h_i))a_i + (\beta - \alpha(k_i - i))d_i - \alpha\left(\sum_{j=h_i}^{i-1} d_j - \sum_{j=i+1}^{k_i} a_j\right).$$

Rearranging we obtain (3.1). □

The following observations are a consequence of Proposition 3.1:

1. Consider some user i arriving at time a_i to an empty system, and departing at time d_i to leave an empty system. In this case there are no other users effecting his sojourn time or rate. For such a user $k_i = h_i = i$. In this case (3.1) implies that $d_i = a_i + 1/\beta$ as expected.
2. The matrices A and D are lower and upper triangular, respectively, with a non-zero diagonal, and are therefore both non-singular.
3. For the special cases $i = 1$ and $i = N$ (using the fact $h_1 = 1$ and $k_N = N$):

$$\big(\beta - \alpha(k_1 - 1)\big)d_1 - \beta\, a_1 + \alpha \sum_{j=2}^{k_1} a_j = 1, \quad \text{and}$$
$$\beta\, d_N - \alpha \sum_{j=h_N}^{N-1} d_j - \big(\beta - \alpha(N - h_N)\big)a_N = 1.$$

i.e.,

$$d_1 = \frac{1 + \beta a_1 - \alpha \sum_{j=2}^{k_1} a_j}{\beta - \alpha(k_1 - 1)}, \quad a_N = \frac{\beta d_N - \alpha \sum_{j=h_N}^{N-1} d_j - 1}{\beta - \alpha(N - h_N)}.$$

The above structure suggests iterative algorithms for either determining $\mathbf{a}$ based on $\mathbf{d}$ or vice-versa. In both cases, $\mathbf{k}$ and $\mathbf{h}$ are found as bi-products. As an aid to describing these algorithms, define for $i, k, h \in \mathcal{N}$ and for a given $\mathbf{a}$ (respectively $\mathbf{d}$), the functions $\tilde{d}_{i,k,h}(\cdot \mid \mathbf{a}), \tilde{a}_{i,k,h}(\cdot \mid \mathbf{d}) \colon \mathbb{R}^N \to \mathbb{R}$ as follows,

$$\tilde{d}_{i,k,h}\Big(\tilde{\mathbf{d}} \,\Big|\, \mathbf{a}\Big) := \frac{1 + \big(\beta - \alpha(i - h)\big)a_i + \alpha\Big(\sum_{j=h}^{i-1} \tilde{d}_j - \sum_{j=i+1}^{k} a_j\Big)}{\beta - \alpha(k - i)},$$
$$\tilde{a}_{i,k,h}\Big(\tilde{\mathbf{a}} \,\Big|\, \mathbf{d}\Big) := \frac{\big(\beta - \alpha(k - i)\big)d_i - \alpha\Big(\sum_{j=h}^{i-1} d_j - \sum_{j=i+1}^{k} \tilde{a}_j\Big) - 1}{\beta - \alpha(i - h)}.$$

Observe that in the evaluation of these functions, the arguments, $\tilde{\mathbf{d}}$ or $\tilde{\mathbf{a}}$ are only utilized for the coordinates indexed $h, \ldots, i-1$ or $i+1, \ldots, k$ respectively (if $i = 1$

or respectively $i = N$ these index lists are empty). Further observe that stated in terms of $\tilde{d}(\cdot)$ or $\tilde{a}(\cdot)$ and given $\mathbf{k} \in \mathcal{K}$ and $\mathbf{h} \in \mathcal{H}$, Eq. (3.1) can be represented as,

$$d_i = \tilde{d}_{i,k_i,h_i}\Big((d_1,\ldots,d_N)' \mid (a_1,\ldots,a_N)'\Big), \quad i \in \mathcal{N},$$

or alternatively,

$$a_i = \tilde{a}_{i,k_i,h_i}\Big((a_1,\ldots,a_N)' \mid (d_1,\ldots,d_N)'\Big), \quad i \in \mathcal{N}.$$

Given the above we have two (dual) algorithms for determining the network dynamics. Algorithm 1a finds the departure times based on arrival times. Algorithm 1b finds the arrival times given the departure times.

Proposition 3.2 *Algorithm* 1*a finds the unique solution* $\mathbf{d}$ *to Eq.* (2.2)*, given* $\mathbf{a}$*. Similarly Algorithm* 1*b finds a unique solution* $\mathbf{a}$ *to the equations, given* $\mathbf{d}$*. Both algorithms require at most* $2N$ *steps in each of which* (3.1) *is evaluated.*

Algorithm 1a: Determination of network dynamics with given arrival times

Input: $\mathbf{a} \in \mathbb{R}^N$ such that $a_1 \le a_2 \le \cdots \le a_N$
Output: $\mathbf{d} = (d_1, ..., d_N)$, $\mathbf{k} = (k_1, ..., k_N)$ and $\mathbf{h} = (h_1, ..., h_N)$
  init $\mathbf{k} = \mathbf{h} = (1, 2, 3, \ldots, N)$
  init $\mathbf{d} = \emptyset$
  for $i = 1, \ldots, N$ **do**
    set $k = i \vee k_{i-1}$ (taking $k_0 := 1$)
    compute $\tilde{d}_i(k, h_i, \mathbf{d} \mid \mathbf{a})$
    while $\tilde{d}_i(k, h, \mathbf{d} \mid \mathbf{a}) \le a_{k+1}$ **do**
      increment k
      compute $\tilde{d}_i(k, h_i, \mathbf{d} \mid \mathbf{a})$
    end while
    set $k_i = k$
    set $d_i = \tilde{d}_i(k, h_i, \mathbf{d} \mid \mathbf{a})$
    set $h_{i+1} = \max\{h \in \{1, \ldots, i+1\} : k_h \ge i+1\}$
  end for
  return $(\mathbf{d}, \mathbf{k}, \mathbf{h})$

3.1 Optimizing for extreme cases of γ

As described in the introduction, optimizing (2.4) when $\gamma = 0$ or $\gamma \approx \infty$ can be done efficiently. For the case $\gamma = 0$, all that is needed is to schedule arrivals so that each departure time, d_i is exactly at d_i^*. This achieves zero costs. Such a schedule is simply obtained by running Algorithm 1b with input $\mathbf{d} = \mathbf{d}^*$. This immediately leads to the following corollary of Proposition 3.2:

Corollary 3.3 *For the special case* $\gamma = 0$ *there is an efficient polynomial time algorithm that finds the unique optimal schedule,* $\mathbf{a}^0$*, achieving* $c(\mathbf{a}^0) = 0$*.*

For the case of large γ it is sensible to consider a classic schedule where users do not overlap:

Algorithm 1b: Determination of network dynamics with given departure times

Input: $\mathbf{d} \in \mathbb{R}^N$ such that $d_1 \le d_2 \le \cdots \le d_N$
Output: $\mathbf{a} = (a_1, ..., a_N)$, $\mathbf{k} = (k_1, ..., k_N)$ and $\mathbf{h} = (h_1, ..., h_N)$
  init $\mathbf{k} = \mathbf{h} = (1, 2, 3, \ldots, N)$
  init $\mathbf{d} = \emptyset$
  for $i = N, \ldots, 1$ **do**
    set $h = i \wedge h_{i+1}$ (taking $h_{N+1} := N$)
    compute $\tilde{a}_i(k_i, h, \mathbf{a} \,|\, \mathbf{d})$
    while $\tilde{a}_i(k_i, h, \mathbf{a} \,|\, \mathbf{d}) \ge d_{h-1}$ **do**
      decrement h
      compute $\tilde{a}_i(k_i, h, \mathbf{a} \,|\, \mathbf{d})$
    end while
    set $h_i = h$
    set $a_i = \tilde{a}_i(k_i, h, \mathbf{a} \,|\, \mathbf{d})$
    set $k_{i-1} = \min\left\{k \in \{i-1, \ldots, N\} : h_k \le i-1\right\}$
  end for
  return $(\mathbf{a}, \mathbf{k}, \mathbf{h})$

$$a_i + \frac{1}{\beta} = d_i \le a_{i+1}, \quad i = 1, \ldots, N-1. \tag{3.3}$$

This poses the problem as a classic single machine scheduling problem with due dates (see for example Baker and Scudder 1990 or Sen and Gupta 1984). This implies that the total costs due to sojourn times is at the minimal possible value $\gamma N/\beta$ and the costs due to deviations from ideal departure times is,

$$\sum_{i \in \mathcal{N}} (a_i + 1/\beta - d_i^*)^2.$$

For any finite γ this does not necessarily minimize (2.4), but as $\gamma \to \infty$ it is a sensible approximation. I.e. for large γ the optimal schedule is approximated by the solution of the following convex quadratic program:

$$\begin{aligned} \min_{(a_1,...,a_N)' \in \mathbb{R}^N} \quad & \sum_{i=1}^{N} (a_i + 1/\beta - d_i^*)^2 \\ \text{s.t.} \quad & a_i - a_{i+1} \le -\frac{1}{\beta}, \quad i = 1, \ldots, N-1. \end{aligned} \tag{3.4}$$

The above quadratic program can be efficiently solved using any standard convex quadratic programming method. Denote the optimizer by $\mathbf{a}^\infty$.

3.2 A linear approximation

Having the schedules $\mathbf{a}^0$ and $\mathbf{a}^\infty$ for the cases $\gamma = 0$ and $\gamma = \infty$ respectively, we are motivated to suggest a set of potential (initial) guesses for the optimal schedule for arbitrary γ. Let $M \ge 1$ be some integer specifying the number of initial guesses. Then the set of initial guesses lie on the segment interpolating $\mathbf{a}^0$ and $\mathbf{a}^\infty$:

$$\mathcal{A} = \left\{ \mathbf{a}^0 \frac{m}{M-1} + \mathbf{a}^\infty \left(1 - \frac{m}{M-1}\right) : m = 0, \ldots, M-1 \right\}, \tag{3.5}$$

when $M \geq 2$ or equals $\{\mathbf{a}^0\}$ if $M = 1$. We shall use the M points of $\mathcal{A}$ as initial guess points for the optimization heuristics that we present in the sequel. This is a sensible choice since every set of due dates $d_1^*, \ldots, d_N^*$ exhibits some contour in $\mathcal{R}$, parametrized by γ, corresponding to the optimal schedules (for each γ). The end points of this contour are $\mathbf{a}^0$ and $\mathbf{a}^\infty$ which we can efficiently find. Thus for $\alpha \in [0, 1]$, the points $\mathbf{a}^0 \alpha + \mathbf{a}^\infty (1 - \alpha)$ constitute a linear approximation of this contour. In cases where the contour is almost not curved we have that the optimal value lies very near to the linear approximation. In other cases, this is simply a set of initial guesses, yet possibly a sensible one. Note that the values of M do not need to be excessively large because initial points that are close are likely to yield the same local solutions. The numerical analysis of Sect. 6 reinforces this observation.

4 Piecewise quadratic formulation

Our key observation in this section is that the search region $\mathcal{R}$ can be partitioned into polytopes indexed by $\mathbf{k} \in \mathcal{K}$, where over each such polytope, the objective is of a convex quadratic form. This yields $|\mathcal{K}|$ convex quadratic problems, each of which (individually) can be solved efficiently. An immediate exhaustive-search algorithm is then to solve all of the problems so as to find the minimising one. This yields a finite-time exact solution and is a sensible choice for small N (e.g. $N \leq 15$). But since,

$$|\mathcal{K}| \sim \frac{4^N}{N^{3/2}\sqrt{\pi}},$$

solving all convex problems is not a viable method for non-small N. We thus also specify a *local-search* algorithm which searches elements of $\mathcal{K}$ by moving across neighbouring polytopes until finding a local optimum.

The following proposition is key:

Proposition 4.1 *The region $\mathcal{R}$ can be partitioned into polytopes indexed by $\mathbf{k} \in \mathcal{K}$, and denoted*

$$\mathbb{P}_\mathbf{k} := \left\{ \mathbf{a} \in \mathcal{R} : a_{k_i} \leq \left[\Theta_\mathbf{k} \mathbf{a} + \boldsymbol{\eta}_\mathbf{k}\right]_i \leq a_{k_i+1},\ i \in \mathcal{N} \right\},$$

where $\Theta_\mathbf{k} = D^{-1}A$ and $\eta_\mathbf{k} = D^{-1}\mathbf{1}$ with A and D based on $\mathbf{k}$ are specified by Proposition 3.1. Then for $\mathbf{a} \in \mathbb{P}^\mathbf{k}$ the objective function is convex and is given by,

$$c_\mathbf{k}(\mathbf{a}) = \mathbf{a}' Q_\mathbf{k} \mathbf{a} + \mathbf{b}_\mathbf{k}\, \mathbf{a} + \tilde{b}_\mathbf{k},$$

with,

$$\begin{aligned} Q_\mathbf{k} &= \Theta_\mathbf{k}'\, \Theta_\mathbf{k}, \\ \mathbf{b}_\mathbf{k} &= 2\big(\boldsymbol{\eta}_\mathbf{k} - \mathbf{d}^*\big)'\Theta_\mathbf{k} + \gamma \mathbf{1}'(\Theta_\mathbf{k} - I), \\ \tilde{b}_\mathbf{k} &= \big(\boldsymbol{\eta}_\mathbf{k} - \mathbf{d}^*\big)'\big(\boldsymbol{\eta}_\mathbf{k} - \mathbf{d}^*\big) + \boldsymbol{\eta}_\mathbf{k}. \end{aligned}$$

Proof The results of Proposition 3.1 show that every $\mathbf{k} \in \mathcal{K}$ specifies matrices D and A such that, $\mathbf{d} = D^{-1}A\,\mathbf{a} + D^{-1}\mathbf{1} = \Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}}$. This holds with constant $\Theta_{\mathbf{k}}$ and $\boldsymbol{\eta}_{\mathbf{k}}$ for all $\mathbf{a}$ and $\mathbf{d}$ for which $\mathbf{k}$ as defined in (2.5) is fixed. The polytope $\mathbb{P}_{\mathbf{k}}$ specifies this exactly by describing the set of arrival points for which the specific ordering of departures within arrivals is given by $\mathbf{k}$.

Since for all $\mathbf{a} \in \mathbb{P}_{\mathbf{k}}$ the affine relationship between $\mathbf{a}$ and $\mathbf{d}$ holds with the same $\Theta_{\mathbf{k}}$ and $\boldsymbol{\eta}_{\mathbf{k}}$ the cost, (2.4), can be explicitly represented in terms of $\mathbf{a}$:

$$\begin{aligned} c(\mathbf{a}) &= \sum_{i\in\mathcal{N}} \left(d_i - d_i^*\right)^2 + \gamma\,(d_i - a_i) \\ &= \left(\mathbf{d} - \mathbf{d}^*\right)' \left(\mathbf{d} - \mathbf{d}^*\right) + \gamma \mathbf{1}'\,(\mathbf{d} - \mathbf{a}) \\ &= \left(\mathbf{a}'\Theta_{\mathbf{k}}' + \left(\boldsymbol{\eta}_{\mathbf{k}} - \mathbf{d}^*\right)'\right) \left(\Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}} - \mathbf{d}^*\right) + \gamma \mathbf{1}'\,\left(\Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}} - \mathbf{a}\right) \\ &= \mathbf{a}'\Theta_{\mathbf{k}}'\Theta_{\mathbf{k}}\mathbf{a} + \left(2\left(\boldsymbol{\eta}_{\mathbf{k}} - \mathbf{d}^*\right)'\Theta_{\mathbf{k}} + \gamma \mathbf{1}'\left(\Theta_{\mathbf{k}} - I\right)\right)\mathbf{a} \\ &\quad + \left(\boldsymbol{\eta}_{\mathbf{k}} - \mathbf{d}^*\right)'\left(\boldsymbol{\eta}_{\mathbf{k}} - \mathbf{d}^*\right) + \gamma \mathbf{1}'\boldsymbol{\eta}_{\mathbf{k}}. \end{aligned}$$

This yields $Q_{\mathbf{k}}$, $\mathbf{b}_{\mathbf{k}}$ and the constant term, $\tilde{b}_{\mathbf{k}}$. Finally, since $Q_{\mathbf{k}}$ is a Gram matrix, it is positive semi-definite. Hence the objective is convex. □

4.1 Exhaustive search

We are now faced with a family of convex quadratic programs. For each $\mathbf{k} \in \mathcal{K}$, denote $c_{\mathbf{k}}(\cdot)$ to be the cost associated with $\mathbf{k}$ then,

$$QP(\mathbf{k}): \quad \min_{\mathbf{a}\in\mathbb{P}_{\mathbf{k}}} c_{\mathbf{k}}(\mathbf{a}). \tag{4.1}$$

Note that while the constant term $\tilde{b}_{\mathbf{k}}$ is not required for finding the solution of $QP(\mathbf{k})$, it is needed for comparing the outcomes of the quadratic programs associated with different elements of $\mathcal{K}$. Indeed the most basic use of $QP(\mathbf{k})$ is for an exhaustive search algorithm which finds the global optimal schedule in finite time. This is summarised in Algorithm 2.

The virtue of Algorithm 2 is that it finds the optimal schedule in finite time. But this is done by solving an exponential (in N) number of convex $QP(\cdot)$ problems, so for non-small N it is not a sensible algorithm. Hence we now introduce a search heuristic.

4.2 Neighbour search

In this section we introduce a heuristic search aimed at finding a local minimum by searching on neighbouring regions. The search procedure solves the QP (4.1) over neighbouring elements of $\mathcal{K}$ by changing a single coordinate of $\mathbf{k}$ at a time. We prove that this procedure converges to a local minimum; yet this may possibly take an exponential number of steps in the worst case.

Algorithm 2: Exhaustive search for global optimum

Input: Model parameters only ($N, \alpha, \beta, \mathbf{d}^*$ and γ)
Output: $\mathbf{a}^*$ (global optimum)
  init $m^* = \infty$
  for $\mathbf{k} \in \mathcal{K}$ **do**
    solve $QP(\mathbf{k})$ with optimiser $\mathbf{a}$ and optimum m
    if $m < m^*$ **then**
      set $\mathbf{a}^* = \mathbf{a}$
      set $m^* = m$
    end if
  end for
  return $(\mathbf{a}^*,\ m^*)$

Given a solution $\boldsymbol{a}$ of $QP(\boldsymbol{k})$ we define the following two sets of indices:

$$\mathcal{I}_1(\mathbf{a},\ \mathbf{k}) := \left\{ j \in \mathcal{N} \colon \left[\Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}}\right]_j = a_{k_j+1} \right\},$$
$$\mathcal{I}_2(\mathbf{a},\ \mathbf{k}) := \left\{ j \in \mathcal{N} \colon a_{k_j} = \left[\Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}}\right]_j \right\}.$$

Noting that $d_i = [\Theta_{\mathbf{k}}\mathbf{a} + \boldsymbol{\eta}_{\mathbf{k}}]_i$, and recalling that k_i is index of the maximal arrival time that is less than or equal to d_i we have that if $i \in \mathcal{I}_1(\mathbf{a},\ \mathbf{k})$ then the optimal solution of $QP(\boldsymbol{k})$ exhibits $d_i = a_{k_i+1}$ as an active constraint. Hence a neighbouring region to the constraint set $\mathbb{P}_{\mathbf{k}}$ is $\mathbb{P}_{\mathbf{k}^{(\mathbf{i})}}$ where $\mathbf{k}^{(i)} = \mathbf{k}$ on all coordinates except for i where it is equal to $k_i + 1$. Similarly if $i \in \mathcal{I}_2(\mathbf{a},\ \mathbf{k})$ then $a_{k_i} = d_i$ as an active constraint. In this case, $\mathbf{k}^{(i)}$ is set to equal $\mathbf{k}$ on all co-ordinates except for i where it is set to equal $k_i - 1$. Thus for every element of $\mathcal{I}_1(\mathbf{a},\ \mathbf{k})$ and $\mathcal{I}_2(\mathbf{a},\ \mathbf{k})$ we have a well defined neighbouring region. Defining now the sets of neighbouring regions to $\mathbb{P}_{\mathbf{k}}$ by

$$\mathcal{K}_\ell\left(\mathcal{I}_\ell(\mathbf{a},\ \mathbf{k})\right) := \left\{ \mathbf{k}^{(i)} \colon i \in \mathcal{I}_\ell(\mathbf{a},\ \mathbf{k}) \right\}, \quad \ell = 1, 2,$$

we have the following local search algorithm:

Algorithm 3: Neighbour search for local optimum (local search)

Input: $\mathbf{k}$
Output: $\mathbf{a}^*$ and m^*
  solve $QP(\mathbf{k})$ with optimiser $\mathbf{a}$ and optimum m
  init $m^* = m$
  init $\mathbf{a}^* = \mathbf{a}$
  for $i \in \mathcal{K}_1\left(\mathcal{I}_1(\mathbf{a}, \mathbf{k})\right)$ **do**
    solve $QP(\mathbf{k}^{(i)})$ with optimiser $\mathbf{a}$ and optimum m
    if $m < m^*$ **then** restart algorithm with $\mathbf{k} = \mathbf{k}^{(\mathbf{i})}$
  end for
  for $i \in \mathcal{K}_2\left(\mathcal{I}_2(\mathbf{a}, \mathbf{k})\right)$ **do**
    solve $QP(\mathbf{k}^{(i)})$ with optimiser $\mathbf{a}$ and optimum m
    if $m < m^*$ **then** restart algorithm with $\mathbf{k} = \mathbf{k}^{(\mathbf{i})}$
  end for
  return $(\mathbf{a}^*,\ m^*)$

Proposition 4.2 *Algorithm* 3 *converges to a local minimum for any initial vector* $\mathbf{k}$.

Proof Every step of the algorithm can only improve the objective function, since $m < m^*$ is the condition for the change of $\mathbf{k}$, hence the algorithm cannot go back to a region which it has already visited. Furthermore, there is a finite number of regions which means the algorithm terminates in a finite number of steps. If for some $\mathbf{a}$ which is the solution of $QP(\mathbf{k})$ there are no improvements in any of the neighbouring regions the algorithm stops at a local minimum. This can be either due to no active constraints to $QP(\mathbf{k})$ (an interior point) or due to the fact that the neighbouring quadratic programs do not improve on the solution of $QP(\mathbf{k})$. □

5 Global search over single coordinates

In this section we put forward Algorithms 4 and 5 that together form a coordinate pivot iteration procedure. We first describe how the dynamics presented in Sects. 2 and 4 can be used to find a global minimum with respect to a single coordinate $r \in \mathcal{N}$ (user) when all other coordinates are fixed. We call this procedure a *global search over a single coordinate* r.

The computational complexity of such a procedure is shown to be at most $O(N^5)$. We then utilise this procedure to define a coordinate pivot iteration algorithm, that performs optimization cycles on all of the coordinates until no improvement can be made.

To understand the main idea consider Fig. 3a. This figure corresponds to an example with $N = 4$, $\alpha = 1.5$ and $\beta = 5$. Here the arrival times a_2, a_3, a_4 are fixed at $(0.05, 0.15, 0.45)$ and the arrival time of user $r = 1$ (denoted also x) is allowed to vary. The (horizontal) blue dotted lines denote the fixed arrival times a_2, a_3, a_4. The thin blue curves correspond to the departure times d_2, d_3, d_4. The thick green dotted and solid curves correspond to the arrival and departure time of user 1 respectively. When x is small enough or large enough, it is seen that user 1 does not affect the other users. But otherwise, user 1 interacts with the other users and potentially modifies their departure times.

As is further evident from Fig. 3a, the dynamics of the departure times are piecewise affine with breakpoints as marked by the vertical lines in the figure. In between these lines, the effect of changing x on other quantities is affine. In between these breakpoints, the objective function is piecewise convex (quadratic). This property is illustrated in Fig. 3b where the objective is plotted as a function of x. This property allows us to optimise globally over a single coordinate, utilizing the problem structure. The desired departure times used for the cost function in (b) were $d_i^* = 0.5$ for $i = 1, \ldots, 4$.

The global search over a single coordinate works by varying x from $\underline{a}$ to $\overline{a}$ and in the process searches for the one-coordinate optimum. This is done with a finite number of steps because of the piecewise-affine dynamics. Our algorithm incrementally computes the piecewise-affine coefficients within these steps. We call each step a "breakpoint". The following types of breakpoints may occur:

Type 1a The arrival of r overtakes the next arrival of any i (solid black line).

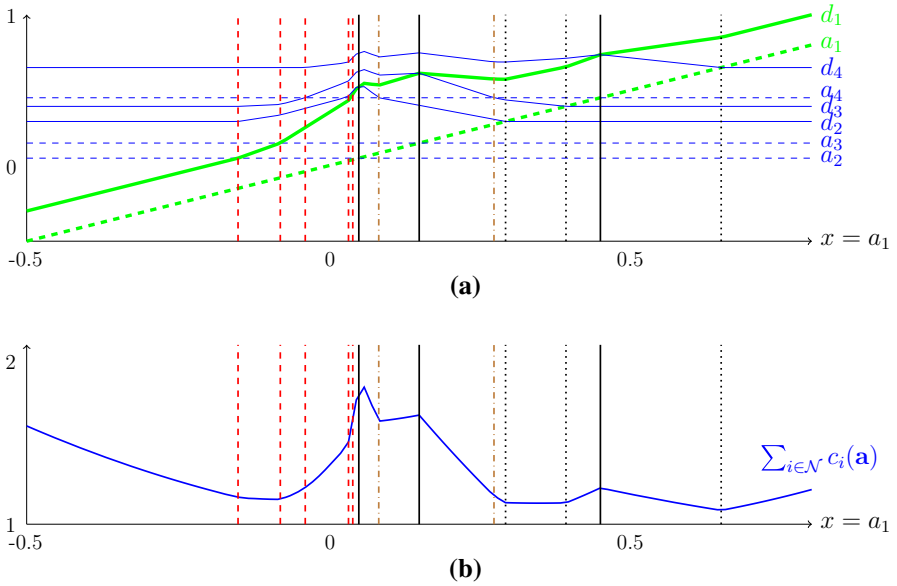

Fig. 3 **a** Arrival (*horizontal dotted*) and departure (*horizontal solid*) profiles obtained by changing the arrival time of user 1, **b** cost function obtained by changing the arrival time of user 1. Break points are marked in both **a** and **b** by *vertical lines* as follows: *solid black lines* mark Type 1a points (note there are exactly $N - 1 = 3$ such breakpoints). *Dotted black lines* mark Type 1b breakpoints (note that there are exactly $N - 1 = 3$ such breakpoints as well). Type 2a breakpoints are marked by *dashed red lines* and Type 2b breakpoints are marked by *brown dashed-dotted lines*. (Color figure online)

Type 1b The departure of any i is overtaken by the arrival of r (dotted black line).
Type 2a The departure of any i overtakes any arrival (dashed red line).
Type 2b The departure of any i is overtaken by an arrival of $j \neq r$ (brown dashed-dotted line).

Observe that in varying x, breakpoints of type 1a and 1b occur exactly $N - 1$ times each. Less trivially, we have a bound on the number of type 2a and 2b breakpoints:

Proposition 5.1 *In executing the global search over a single coordinate r, the total number of breakpoints is $O(N^3)$.*

Before presenting the proof, we present the details of the piecewise-affine dynamics and the details of the global search over a single coordinate r algorithm.

5.1 Algorithm details

In carrying out the global search over a single coordinate r, we remove the restriction that arrival times are ordered. That is, the search region is extended from $\mathcal{R}$ to a set not requiring such order $\widetilde{\mathcal{R}} := [\underline{a}, \overline{a}]^N$. This allows us to carry out a full search for the optimum with respect to a single user r without the restriction $a_r \in [a_{r-1}, a_{r+1}]$. This broader search potentially enables bigger gains in the objective when integrating

the algorithm within a search heuristic. Further, any point $\mathbf{a} \in \widetilde{\mathcal{R}}$ can be mapped into a unique point $\mathcal{O}(\mathbf{a}) \in \mathcal{R}$ where $\mathcal{O}(\cdot)$ is an ordering operator. By Lemma 2.3 we have that $c\big(\mathcal{O}(\mathbf{a})\big) \leq c\big(\mathbf{a}\big)$.

Take $\widetilde{\mathbf{a}} \in \widetilde{\mathcal{R}}$ as an initial arrival vector and suppose that we are optimising over user r. Let $x \in [\underline{a},\, \overline{a}]$ be the immediate search value of a_r (keeping the other arrival times fixed). For any such x we define a corresponding permutation $\boldsymbol{\pi}(\widetilde{\mathbf{a}},\, x)$ indicating the current order of arrivals, as well as the ordered arrival vector

$$\mathbf{a}(\widetilde{a},\, x) := \mathcal{O}\big(a_{\pi_1(x)},\, \ldots,\, a_{\pi_r(x)-1},\, x,\, a_{\pi_r(x)+1},\, \ldots,\, a_{\pi_N(x)}\big). \tag{5.1}$$

This vector can serve as input to Algorithm 1a yielding a corresponding $\mathbf{d}(\widetilde{a},\, x)$, $\mathbf{k}(\widetilde{a},\, x)$ and $\mathbf{h}(\widetilde{a},\, x)$. Furthermore, using (3.1) we have the local piecewise-affine relationship,

$$d_i(\widetilde{a},\, x) = x\, \theta_{i\,|r,\boldsymbol{\pi}(\widetilde{\mathbf{a}},\, x),\mathbf{k}(\widetilde{\mathbf{a}},\, x)} + \eta_{i\,|r,\boldsymbol{\pi}(\widetilde{\mathbf{a}},\, x),\mathbf{k}(\widetilde{\mathbf{a}},\, x)},\ i \in \mathcal{N}, \quad x \in [\underline{a},\, \overline{a}].$$

That is, the coefficients of the departures between breakpoints depend on the permutation of the users as well as on the current order of their arrivals and departures. For brevity we omit the dependencies on x, $\widetilde{\mathbf{a}}$, $\boldsymbol{\pi}$ and $\mathbf{k}$. Manipulating (3.1) we obtain,

$$(\theta_i,\, \eta_i) = \begin{cases} \left(0,\ \frac{1+a_i(\beta-\alpha(i-h_i))-\alpha\left(\sum_{j=i+1}^{k_i} a_j - \sum_{j=h_i}^{i-1} \eta_j\right)}{\beta-(k_i-i)\alpha}\right), & i < \pi_r,\ k_i < \pi_r, \\ \left(-\frac{\alpha\left(1-\sum_{j=h_i}^{i-1} \theta_j\right)}{\beta-\alpha(k_i-i)},\ \frac{1+a_i(\beta-\alpha(i-h_i))-\alpha\left(\sum_{j=i+1}^{k_i} a_j \mathbb{1}\{j\neq\pi_r\} - \sum_{j=h_i}^{i-1} \eta_j\right)}{\beta-\alpha(k_i-i)}\right), & i < \pi_r,\ k_i \geq \pi_r, \\ \left(\frac{\beta+\alpha\sum_{j=h_i}^{i-1}(\theta_j-1)}{\beta-\alpha(k_i-i)},\ \frac{1-\alpha\left(\sum_{j=i+1}^{k_i} a_j - \sum_{j=h_i}^{i-1} \eta_j\right)}{\beta-\alpha(k_i-i)}\right), & i = \pi_r, \\ \left(\frac{\alpha\sum_{j=h_i}^{i-1} \theta_j}{\beta-\alpha(k_i-i)},\ \frac{1+a_i(\beta-\alpha(i-h_i))-\alpha\left(\sum_{j=i+1}^{k_i} a_j - \sum_{j=h_i}^{i-1} \eta_j\right)}{\beta-\alpha(k_i-i)}\right), & i > \pi_r. \end{cases} \tag{5.2}$$

On every interval, the departure times d_i are all affine and continuous w.r.t x with the above coefficients, until a breakpoint (of type 1a, 1b, 2a or 2b) occurs. Computing the time of the next breakpoint is easily done by considering the piecewise affine dynamics. Potential breakpoints of types 1a and 1b are to occur at times t where $x + t = a_{\pi_r+1}$ and $t\,\theta_i + d_i = a_r + t$, respectively. Potential breakpoints of types 2a and 2b involving user i are to occur at times $t\,\theta_i + d_i = a_{k_i+1}$ and $t\,\theta_i + d_i = a_{k_i}$ respectively. Observing now that type 2a breakpoints may occur only when $\theta_i > 0$ and type 2b breakpoints may occur only when $\theta_i < 0$ we have that the next breakpoint occurs at,

$$\tau = \min\{t_0, t_1, \ldots, t_N, t_{N+1}\}, \tag{5.3}$$

where $t_0 = a_{\pi_r+1} - x$ (type 1a breakpoints), $t_{N+1} = \bar{a} - x$ (termination) and for $1 \leq i \leq N$:

$$t_i = \begin{cases} \frac{a_{k_i} - \theta_i x - \eta_i}{\theta_i}, & \theta_i < 0,\ k_i \neq r, \\ \frac{a_{k_i} - \theta_i x - \eta_i}{\theta_i - 1}, & \theta_i < 0, k_i = r, \\ \frac{a_{k_i+1} - \theta_i x - \eta_i}{\theta_i}, & \theta_i > 0, \\ \infty, & \theta_i = 0. \end{cases}$$

Considering the time interval until the next breakpoints, $[x, \tau]$ we have that the total cost as a function of the arrival time $\hat{x} \in [x, \tau]$ of user r is

$$\tilde{c}(\hat{x}; \boldsymbol{\pi}) := \sum_{j \in \mathcal{N}} \left(\left(\theta_{\pi_j} \hat{x} + \eta_{\pi_j} - d_j^* \right)^2 + \gamma \left(\theta_{\pi_j} \hat{x} + \eta_{\pi_j} - a_{\pi_j} \right) \right),$$

with derivative $\partial \tilde{c}(\hat{x}; \boldsymbol{\pi}) = \sum_{j \in \mathcal{N}} \theta_{\pi_j} \left(2(\eta_{\pi_j} - d_j^*) + \gamma \right) + 2\hat{x} \sum_{j \in \mathcal{N}} \theta_{\pi_j}^2$, and with the root $x_0 \geq x$, solving $\partial \tilde{c}(x_0; \boldsymbol{\pi}) = 0$ (and often not lying within the interval $[x, \tau]$):

$$x_0 = \frac{- \sum_{j \in \mathcal{N}} \theta_{\pi_j} \left(2 \left(\eta_{\pi_j} - d_j^* \right) + \gamma \right)}{2 \sum_{j \in \mathcal{N}} \theta_{\pi_j}^2}.$$

Note that it is crucial to keep track of $\boldsymbol{\pi}$ at every step in order to associate the correct ideal departure time to every user. In iterating over intervals we search for the minimal $\tilde{c}(\hat{x}; \boldsymbol{\pi})$ (denoted m^*) as follows: if $\partial \tilde{c}(\hat{x}; \boldsymbol{\pi}) > 0$ for all $\hat{x} \in [x, \tau]$, then we continue to the next interval. Otherwise, if $x_0 - x \leq \tau$ and $m^* > \tilde{c}(x_0; \boldsymbol{\pi})$, then set $m^* = \tilde{c}(x_0; \boldsymbol{\pi})$ and $\left(\mathbf{a}^* \right)_r = x_0$, and if $x_0 - x > \tau$ and $m^* > \tilde{c}(x + \tau; \boldsymbol{\pi})$, then set $m^* = \tilde{c}(x + \tau; \boldsymbol{\pi})$ and $\left(\mathbf{a}^* \right)_r = x + \tau$.

In this way, x updates over intervals, of the form $[x, \tau]$. Prior to moving to the next interval we need to update the permutation variables π, $\mathbf{k}$, and $\mathbf{h}$. Denote the minimizing set of (5.3) by $\mathcal{T} := \operatorname{argmin}\{t_0, t_1, \ldots, t_N\}$ and sequentially for every $i \in \mathcal{T}$:

- If $i = 0$, then we update π by changing the order between user r and the next user j: $\pi_j = \pi_r + 1$, i.e. set $\pi_r = \pi_r + 1$ and $\pi_j = \pi_j - 1$. In this case, there is no change in $\mathbf{k}$ or $\mathbf{h}$ (type 1a breakpoints).
- If $i \in \{1, \ldots, N\}$, then the order $\boldsymbol{\pi}$ does not change, but we update $\mathbf{k}$ and $\mathbf{h}$: If $\theta_i < 0$, then update $h_{k_i} = h_{k_i} + 1$, followed by $k_i = k_i - 1$. If $\theta_i > 0$, then update $k_i = k_i + 1$, followed by $h_{k_i} = h_{k_i} - 1$ (all other types of breakpoints).
- If $i = N + 1$, then the iteration is complete and no changes are required.

Remark For any convex and differentiable cost functions, the first order condition yielding x_0 can be solved. For some elaborate functions this may also require a numerical procedure. If the late and early cost functions are not strictly convex (for example affine), then computing x_0 can be skipped. If the cost function is piecewise affine, then only the sign of $\partial \tilde{c}$ needs to be computed, and if it is negative check if the next point $x + \tau$ is a new minimum point or not.

Algorithm 4: Global search over a single coordinate

Input: $\tilde{\mathbf{a}} \in \widetilde{\mathcal{R}}$, $r \in \mathcal{N}$, $\boldsymbol{\pi}$
Output: $\mathbf{a}^*$ and m^*

```
init x = ã_r = a_
init a = O(ã)
run Alg.1a(a) → (d, k, h)
init a* = ã
init m* = c̃(x; π)
set π_r = 1
for i < r do
    set π_i = π_i + 1
end for
while x ≤ ā do
    set a = O(ã)
    compute: θ, η, τ, T, and ∂c̃(x; π)
    if ∂c̃(x; π) < 0 then
        compute x_0 and c̃(x_0; π)
        if x_0 < x + τ then
            if c̃(x_0; π) < m* then
                set a*_r = x_0
                set m* = c̃(x_0; π)
            end if
        else if c̃(x + τ; π) < m* then
            set a*_r = x + τ
            set m* = c̃(x + τ; π)
        end if
    end if
    set x = x + τ
    for i ∈ T do
        if i = 0 then
            set π_j = π_j − 1 where j satisfies π_j = π_r + 1
            set π_r = π_r + 1
        end if
        if i ∈ {1, ..., N} then
            if θ_i < 0 then
                set h_{k_i} = h_{k_i} + 1 and k_i = k_i − 1
            else if θ_i > 0 then
                set k_i = k_i + 1 and h_{k_i} = h_{k_i} − 1
            end if
        end if
    end for
end while
return (a*, m*)
```

5.2 Computational complexity

In the following series of lemmata we analyse the complexity of Algorithm 4. In particular, we prove Proposition 5.1, establishing bounds for the number of breakpoints of each type. Throughout the analysis we continue denoting the coordinate being optimised by r and the respective value by $x = \tilde{a}_r$. Keep in mind that π_i, θ_i, k_i, and h_i are functions of x and the initial unordered vector $\tilde{\mathbf{a}}$ for every $i \in \mathcal{N}$. We treat $\mathbf{a}$ as the ordered vector (5.1) as before.

Lemma 5.2 *For any $i \in \mathcal{N}$ such that $i \neq r$, the coefficient $\theta_i \leq 0$ and as a consequence $d_i(x)$ is monotone non-increasing for every $x > a_i$.*

Lemma 5.3 *For any permutation $\boldsymbol{\pi}$ at the start of the global search on r, the coefficient θ_i of any $i \in \boldsymbol{\pi}$ changes sign from strictly positive to strictly negative or vice versa at most $i - 1$ times during the search.*

We now prove Proposition 5.1:

Proof For any $2 \leq i \leq N$ in the original permutation $\boldsymbol{\pi}$, the type 2a and 2b breakpoints occur at most $N - i$ times for every change of sign. This is because their departure time can only cross arrival times of later arrivals. According to Lemma 5.3, the number of sign changes for any $2 \leq i \leq N$ is at most $i - 1$. Thus, the total number of breakpoints of type (2a or 2b) is at most

$$\sum_{i=2}^{N}(N-i)(i-1) = \frac{N(2-3N+N^2)}{6}.$$

Thus, adding up all types of breakpoints, we get that the search domain $[\underline{a}, \overline{a}]$ is broken up to at most $\left(\frac{1}{3}N^3 - N^2 + \frac{8}{3}N - 2\right)$ intervals. □

Furthermore, we have the following bound for the complexity of Algorithm 4.

Corollary 5.4 *The computation complexity of Algorithm* 4 *is at most $O(N^5)$.*

Proof In every interval step of a global search on a single coordinate there is a need to compute the coefficient vectors $\boldsymbol{\eta}$ and $\boldsymbol{\theta}$. This is equivalent to calculating the departure times recursively using Algorithm 1a. In Proposition 3.2 it was shown that the recursion requires at most $2N$ steps. On top of this, in every one of these steps the actual computation requires summation of up to N variables. Now since the number of breakpoints intervals is bounded by $O(N^3)$ we conclude the result. □

5.3 Coordinate pivot iteration optimization

In this subsection we illustrate how Algorithm 4 can be applied to carry out standard coordinate pivot iteration (CPI), see Bertsekas (1999, p. 272). In every iteration of the CPI algorithm, the total cost function is minimized with respect to the arrival time of one user, when all other arrival times are fixed. This is then repeated for all users; we call the iteration over all N users a CPI cycle. The CPI algorithm stops when the total improvement in a cycle is smaller than some specified tolerance parameter, $\epsilon > 0$. Note that in non-smooth CPI (such as our case), CPI often stops when the total improvement is in-fact exactly 0. That is, ϵ is often not a significant parameter. A further comment is that our CPI algorithm utilizes Algorithm 4 searching over the broader space, $\widetilde{\mathcal{R}}$. We can thus improve the objective (see Lemma 2.3) by incorporating the ordering operator, $\mathcal{O}$, at the end of each CPI cycle.

We add the following notations for the optimization procedure: Let $n = 0, 1, \ldots$ be the cycle number, $c^{(n)}$ the total cost at end of cycle n, m^* the global minimal total cost, and $\mathbf{a}^*$ the global optimal arrival vector.

Algorithm 5: Coordinate pivot iteration (global search)

Input: $\mathbf{a}^{(0)}$ and ϵ
Output: $\mathbf{a}^*$ and m^*
  init $n = 0$
  init $\Delta = \epsilon + 1$
  init $\mathbf{a}^* = \mathbf{a}^{(0)}$
  init $c^{(0)} = c(\mathbf{a}^*)$
  while $\Delta > \epsilon$ **do**
    set $n = n + 1$
    set $\tilde{\mathbf{a}} = \mathbf{a}^*$
    for $r \in \mathcal{N}$ **do**
      run **Alg. 4**$(r, \tilde{\mathbf{a}}) \to \tilde{\mathbf{a}}$,
    end for
    set $\mathbf{a}^* = \mathcal{O}(\tilde{\mathbf{a}})$
    set $c^{(n)} = c(\mathbf{a}^*)$
    set $\Delta = c^{(n-1)} - c^{(n)}$
  end while
  set $m^* = c^{(n)}$
  return $(\mathbf{a}^*,\ m^*)$

Hinging upon the results of the previous section, we have:

Corollary 5.5 *The computation complexity of a single* CPI cycle, i.e. conducting a line search on all coordinates, is at most $O(N^6)$.

Proof In Proposition 5.4 we established that for a single coordinate the complexity is at most $O(N^5)$. It is therefore immediate that the complexity of running the algorithm for every coordinate is at most $O(N^6)$. □

Note that while we have a polynomial time CPI algorithm, there is no guarantee that it converges to a local minimum since the objective function is not smooth. In fact, numerical experimentation suggests that this is typically the case when the number of users is not very small, i.e., $N \geq 4$. Nevertheless, experimentation has shown that CPI algorithm generally outputs an arrival vector that lies in the vicinity of the optimum. This motivates combining it with the neighbour search, Algorithm 3 as discussed in the next section.

6 A combined heuristic and numerical results

We now utilise the problem structure and aforementioned algorithms to produce a combined heuristic. We use $\mathcal{A}$ as in (3.5) for initial points. For each of these M initial points we run a CPI (global) search followed by neighbour (local) search. The core principal is to use the CPI method in order to find a "good" initial polytope, or equivalently an arrival-departure permutation, and then to seek a local minimum using the neighbour search.

Algorithm 6: Combined global and local search heuristic

Input: Model parameters only ($N, \alpha, \beta, \mathbf{d}^*$ and γ)
Output: $\mathbf{a}^*$ (local optimum)
  init $m^* = \infty$
  for $\mathbf{a} \in \mathcal{A}$ **do**
    run **Alg.5**$(\mathbf{a}, \ldots) \to (\hat{\mathbf{a}}, \hat{m})$
    set $\hat{\mathbf{k}} = \mathbf{k}(\hat{\mathbf{a}})$
    run **Alg.3**$(\hat{\mathbf{k}}, \ldots) \to (\hat{\mathbf{a}}, \hat{m})$
    if $\hat{m} < m^*$ **then**
      set $\mathbf{a}^* = \hat{\mathbf{a}}$ and $m^* = \hat{m}$
    end if
  end for
  return $(\mathbf{a}^*,\ m^*)$

We tested the combined heuristic Algorithm 6 on a variety of problem instances and it appears to perform very well both in terms of running time and in finding what we believe is a global optimum. Here we illustrate these results for one such problem instance. We take $\beta = 1$ and $\alpha = 0.8/N$ (in this case the maximal slowdown is of the order of 80% independently of N). We set $\mathbf{d}^*$ as the N quantiles of a normal distribution with mean 0 and standard deviation $1/2$. That is, there is an ideal departure profile centred around 0. It is expected that when using optimal schedules, more congestion will occur as N increases and/or γ decreases.

Figure 4 illustrates the dynamics of the obtained schedules as generated by the heuristic (using $M = 3$ and $\epsilon = 0.001$). In these plots arrival times of individual users are plotted on the top axis, marked by blue dots, shifted to the right by the free flow time ($1/\beta = 1$). Departure times are plotted on the bottom axis. Users that do not experience any delay are then represented by lines that are exactly vertical. Further, the more slanted the line, the more slowdown that the user experiences. The ideal departure times are marked by green stars. Hence ideally the stars are to align with the red dots. This occurs exactly when $\gamma = 0$, and approximately occurs for small γ, for instance $\gamma = 0.1$ as in (a) and (d). Then as γ is increased, the optimal schedule is such that there is hardly any delay (almost perfectly vertical lines), but in this case, users experience major deviations between departure times and the ideal values.

For $N = 15$, as presented in (a)–(c), we were indeed able to verify optimality using the exhaustive search Algorithm 2. For $N = 50$, as presented in (d)–(f) we are not able to use the exhaustive search algorithm in any reasonable time. Nevertheless, in this case, in addition to seeing qualitatively sensible results, experimentation showed that increasing M does not modify the results. Hence we believe that the obtained schedules are also optimal.

For $N \leq 15$, we were not able to find a case where the heuristic did not find the optimal schedule. This was tested on a wide range of parameter values by varying α and γ and randomly generating multiple due date vectors. Further for large N (up to 500) we see insensitivity with respect to M (the number of initial points) as well as to other randomized initial points. This result was also robust to changes in all of the parameter values (α, β, γ, and $\mathbf{d}^*$). This leads us to believe that our heuristic performs very well.

Table 1 Running time in seconds and computational steps of the combined heuristic (Algorithm 6 with $M = 1$) and the exhaustive search (Algorithm 2)

N	3	5	10	11	12	14	15	20	30	50
Combined heuristic										
CPI cycles	3	2	4	5	4	3	3	3	3	4
Total breakpoints	24	65	306	382	441	642	727	1383	3260	8636
NS QPs solved	2	2	9	6	8	14	10	29	34	29
Running time (s)	0.05	0.15	1.17	1.67	1.99	3.09	3.28	6.38	18.31	85.33
Global opt.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	NA
Exhaustive search										
$\lvert\mathcal{K}\rvert$ QPs solved	5	42	16,796	58,786	208,012	2.6×10^6	9.7×10^6	6.6×10^9	3.8×10^{15}	2×10^{27}
Running time (s)	0.00	0.05	25.25	162.41	509	8206	39,454	NA	NA	NA

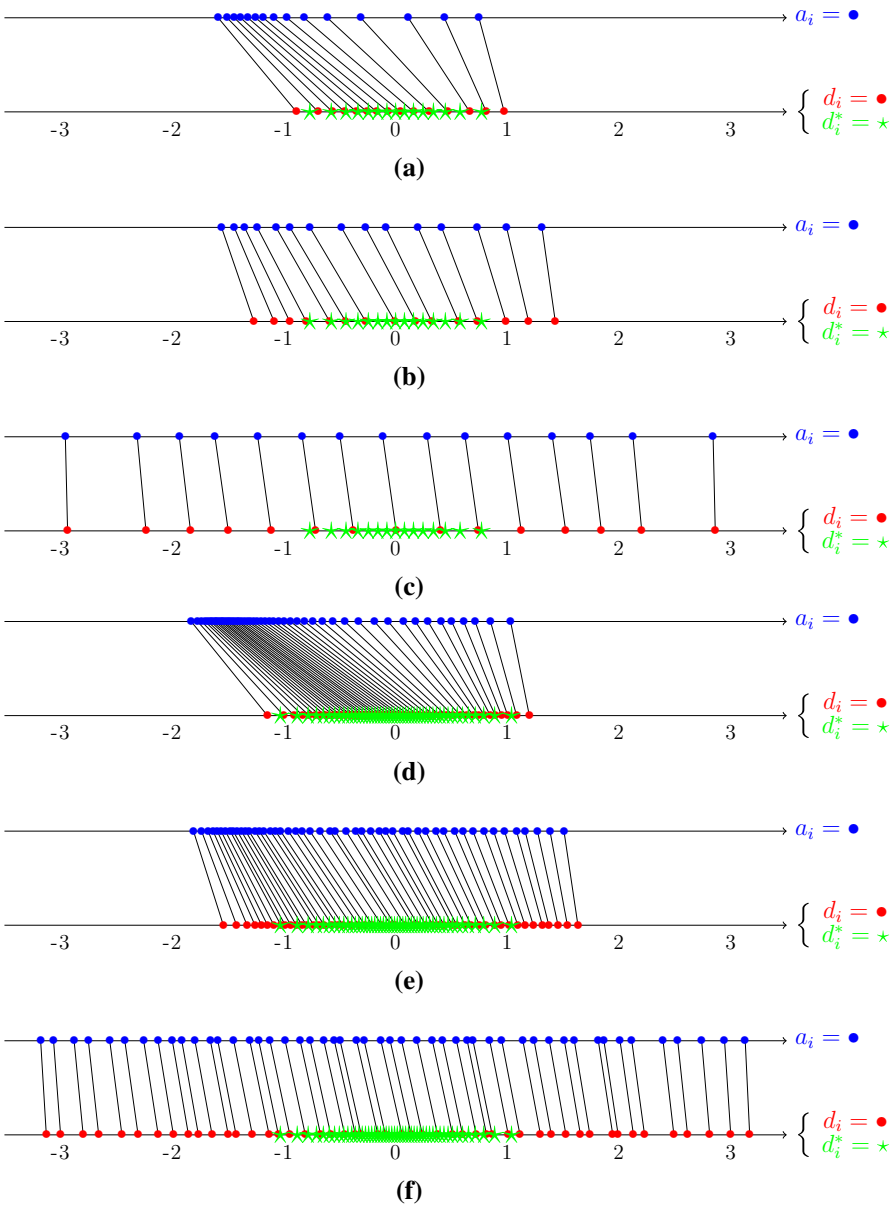

Fig. 4 Optimal arrival-departure diagram for $\alpha = 0.8/N$, $\beta = 1$, and $\mathbf{d}^*$ the N quantiles of a normal distribution with mean 0 and standard deviation 1/2. **a** $N = 15, \gamma = 0.1$, **b** $N = 15, \gamma = 1$, **c** $N = 15, \gamma = 20$, **d** $N = 50, \gamma = 0.1$, **e** $N = 50, \gamma = 1$, **f** $N = 50, \gamma = 20$. (Color figure online)

Results, comparing running times are reported in Table 1 where we consider the algorithm with a single initial point $\mathbf{a}^0$ ($M = 1$), and compare it to the exhaustive search given by Algorithm 2. For this table, we use the same problem data as described above,

but scale the standard deviation by N to be $0.04N$. For $N \leq 15$ the combined heuristic converged to the global optimum as verified by the exhaustive search with a negligible number of computations. For example, for $N = 15$ the heuristic method made $\sim$737 core computations, i.e. solving a QP for a single CPI interval or NS polytope, in 3.28 s, while the exhaustive search had to solve $\sim 10^7$ quadratic programs and required about 11 h.[2] Clearly, for larger N it is not feasible to run the exhaustive search while the combined heuristic is still very quick, as seen for up to $N = 50$ in Table 1.

To further investigate our combined heuristic, in Fig. 5 we illustrate the number CPI cycles and breakpoints, along with the respective number of quadratic programs solved by the neighbour search, until convergence of Algorithm 6. The problem data was scaled as in the previous example. For every N the initial points given by $\mathcal{A}$ with $M = 5$ distinct initial points. The figure displays the minimum and maximum values out of the 5 initial points. Note that for every N the algorithm converged to the same local minimum for all initial points in $\mathcal{A}$.

We can see that the number of required CPI cycles was small and stabilized on 2 regardless of the number of users. However, we should take into account that the number of coordinate iterations in every cycle is N, and that the complexity of each iteration also grows with N. Specifically, Proposition 5.4 shows that the number of breakpoints for every coordinate in the CPI is at most N^3, but in the example we see the growth is in effect linear ($\sim 3N$). Furthermore, the number of required quadratic programs solved in the neighbour search also grows linearly ($\sim \frac{1}{3}N$). This hints that the CPI does indeed find a point that is very "close" to a local minimum. The widening gap between the minimum and maximum number of NS iterations suggests that some of the initial points are better than others, and thus it is worthwhile trying several of them. The last point is important when solving for even larger values of N as the algorithm becomes more sensitive to "bad" initial points and may require setting a maximum number of iterations parameter for every initial point. Roughly, when γ and α are both small, starting closer to $\mathbf{a}^0$ is better and when they are both large, starting closer to $\mathbf{a}^\infty$ is better. However, for most combinations of parameters there seems to be no a-priori indication of what is a "good" starting point. Thus it is still beneficial to do the full search on $\mathcal{A}$. Again we stress that the behaviour displayed in Fig. 5 was robust with respect to changes in the model parameters.

7 Conclusion and outlook

We presented a model for a discrete-user deterministic processor sharing system, and addressed the problem of scheduling arrivals to such a system with the goal of minimizing congestion and tardiness costs. A full characterisation of the congestion dynamics and an efficient method for computing them was provided. It was further shown that the optimal arrival schedule can be computed in a finite, but exponentially large, number of steps. Several heuristics were therefore developed with the goal of an efficient computation of the optimal schedule. A combined global and local search

[2] These computation times are using an AMD computer with 4 *Phenom II 955 3.2 GHz* processors, with our algorithms implemented in version 3.1.2 of the R software.

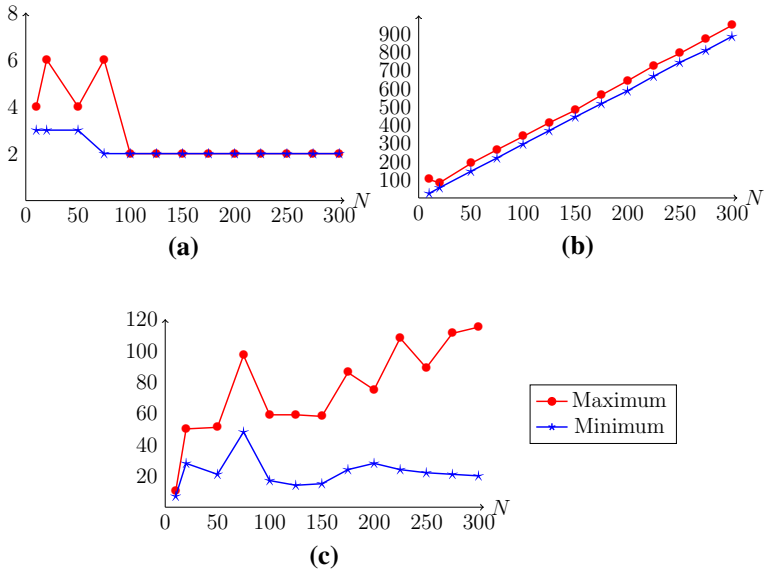

Fig. 5 Number of iterations in each component of Algorithm 6 as a function of N. **a** CPI cycles, **b** CPI breakpoints, **c** NS iterations. (Color figure online)

heuristic was presented and numerically analysed. This method was shown to be efficient in numerical examples for a large population of users.

The essential parts of our analysis and results applies for a much more general cost formulation, as we shall next detail. Given that user i enters the system at time a_i and leaves at time $d_i > a_i$, a plausible cost incurred by the user is the following:

$$\begin{aligned} c_i(a_i, d_i) = {} & g_i^{(1)}\Big((d_i - d_i^*)^+\Big) + g_i^{(2)}\Big((d_i^* - d_i)^+\Big) \\ & + g_i^{(3)}\Big((a_i - a_i^*)^+\Big) + g_i^{(4)}\Big((a_i^* - a_i)^+\Big) \\ & + g_i^{(5)}\Big(d_i - a_i\Big), \end{aligned} \tag{7.1}$$

where $(x)^+ := \max(x, 0)$, and $g_i^{(j)}(\cdot)$, $j \in \{1, \ldots 5\}$, $i \in \mathcal{N}$ are some convex functions.

The first and third terms of (7.1) capture the penalty for being late to the ideal departure and arrival times d_i^* and a_i^*, respectively. The second and fourth terms are the user's cost for arriving and departing early. The fifth term is the user's cost for travel/usage of the system. Our algorithm and results in this paper hold with slight technical modifications for arbitrary convex $g_i^{(j)}(\cdot)$. For purpose of exposition, we focused on, $g_i^{(1)}(x) = g_i^{(2)}(x) = x^2$, $g_i^{(3)}(x) = g_i^{(4)}(x) = 0$ and $g_i^{(5)}(x) = \gamma\, x$. If adapted to the more general formulation, The exhaustive and neighbour search algorithms of Sect. 4 will generally require solving a constrained convex program, instead of convex quadratic, for every region. If $g_i^{(1)}(x) \neq g_i^{(2)}(x)$ and/or $g_i^{(3)}(x) \neq$

$g_i^{(4)}(x)$, namely there are different penalties for arriving/departing later and early, then the CPI algorithm of Sect. 5.3 will require some refinement of the definition of the piecewise segments. The complexity will not change as for every single coordinate there will be an addition of at most three segments, corresponding for these new points of discontinuity. Moreover, the root of the first order condition in every continuous segment will be given by the general form of the functions, instead of the quadratic root.

An interesting generalization is considering a system with users who have heterogeneous service demand. If this is the case then the order of departures is no longer identical to the order of arrivals. This means that the characterisation of Proposition 3.1 is no longer valid.

A natural complementary model to this work is considering a decentralized decision framework in which the users choose their own arrival time. Namely, a non cooperative game with the individual arrival times are the actions of the players. This game is formulated and analysed in Ravner et al. (2016).

Finally, there is the challenge of characterising the computational complexity of our scheduling problem. We believe that finding the optimal $\mathbf{k}^* \in \mathcal{K}$ is an NP-complete problem but we still do not have a proof for this. Our belief is motivated (but not supported) by the fact that there are a number of related optimization problems which are known to be NP hard. Our problem is equivalent to a special case of one of them, namely non-linear integer programming.

As we have shown, our goal is to minimize a non-convex piecewise quadratic objective function, subject to piecewise linear constraints. It is known that non-convex quadratic programs and non-convex piecewise linear optimization are both NP hard (see Keha et al. 2006; Murty and Kabadi 1987). In Vielma et al. (2010) it is shown that piecewise linear optimization problems can be modelled as linear mixed integer programs, where the definition of a piecewise linear program relies on different coefficients for different polytopes, in a similar manner to our piecewise quadratic formulation in Sect. 4. It may be possible to apply similar methods with modifications for the piecewise convex instead of linear objective. However, there is a more natural construction for our case. Let $\tilde{\mathbf{a}}(\mathbf{k})$ be the solution to $QP(\mathbf{k})$, i.e., the solution to the local convex QP of a polytope $\mathbf{k} \in \mathcal{K}$. But this can also be viewed as a function of the integer vector $\mathbf{k}$ which we can compute in polynomial time. Hence, solving our problem in polynomial time is equivalent to solving the non-linear integer program:

$$\min_{\mathbf{k}\in\mathcal{K}} \tilde{\mathbf{a}}(\mathbf{k})' Q_{\mathbf{k}} \tilde{\mathbf{a}}(\mathbf{k}) + \mathbf{b}_{\mathbf{k}}\, \tilde{\mathbf{a}}(\mathbf{k}) + \tilde{b}_{\mathbf{k}}.$$

Recall that $\mathcal{K} = \left\{\mathbf{k} \in \mathcal{N}^N \,:\, k_N = N,\ k_i \le k_j\ \forall i \le j\right\}$ defines a set of linear constraints on the integer decision variables. Clearly the objective is not linear with respect to $\mathbf{k}$, as $\tilde{\mathbf{a}}(\mathbf{k})$ itself is already not necessarily linear. Such problems are known to be NP hard. See for example, De Loera et al. (2006) and Pia et al. (2016). Although we could not find a straightforward reduction of the problem to a known NP hard problem, we have shown that our problem can be formulated as an (rather cumbersome) instance of a polynomial integer program, and have no reason to believe that the specific model comes with significant simplification of the general form.

As a closing note we mention that it is generally of interest to compare our heuristics to potential integer programming methods. One may either discretize time and solve integer programs, or alternatively seek related integer programming formulations. It remains an open problem to compare our heuristics to such potential methods both in terms of accuracy and computation time.

Acknowledgements We thank Hai Vu and Moshe Haviv for useful discussions and advice. We are grateful to two anonymous reviewers for their helpful comments. We thank The Australia-Israel Scientific Exchange Foundation (AISEF) for supporting Liron Ravner's visit to The University of Queensland. Yoni Nazarathy's research is supported by ARC Grants DE130100291 and DP130100156.

Appendix: Proofs

Proof of Lemma 2.1 Consider two arrivals $a_i \le a_j$. During the time interval $[a_i, a_j]$, user i has received some service,

$$\int_{a_i}^{a_j} v\big(q(t)\big)dt,$$

while user j has not. Then during the time interval $[a_j, d_i \wedge d_j]$ both users receive the same service, $\int_{a_j}^{d_i \wedge d_j} v\big(q(t)\big)dt$. Then if $d_i > d_j$ we have that $\int_{a_j}^{d_i \wedge d_j} v\big(q(t)\big)dt = 1$, which in turn would imply that,

$$\int_{a_i}^{d_i} v\big(q(t)\big)dt = \int_{a_i}^{a_j} v\big(q(t)\big)dt + \int_{a_j}^{d_i \wedge d_j} v\big(q(t)\big)dt + \int_{d_i \wedge d_j}^{d_i} v\big(q(t)\big)dt > 1,$$

a contradiction. Hence $d_i \le d_j$. □

Proof of Lemma 2.2 Without loss of generality assume $a_1 \le \cdots \le a_N$ and hence by the previous lemma, $\mathbf{d}$ is ordered. Assume now that there exists a $\tilde{\mathbf{d}} \ne \mathbf{d}$ and define $i = \min\{i : \tilde{d}_i \ne d_i\}$. Without loss of generality, assume that $d_i < \tilde{d}_i$. Using (2.1) it holds that,

$$\int_{a_i}^{d_i} v\big(q(t)\big)\,dt = 1 = \int_{a_i}^{d_i} v\big(\tilde{q}(t)\big)\,dt + \int_{d_i}^{\tilde{d}_i} v\big(\tilde{q}(t)\big)\,dt.$$

Now since for all $t \le d_i$ it holds that $q(t) = \tilde{q}(t)$, then,

$$\int_{d_i}^{\tilde{d}_i} v\big(\tilde{q}(t)\big)\,dt = 0.$$

A contradiction.

Now there exists a full symmetry between $\mathbf{a}$ and $\mathbf{d}$, hence going in the opposite direction (for every $\mathbf{d}$ there exists a unique $\mathbf{a}$) follows a similar argument to the above. □

Proof of Lemma 2.3 We first argue that an optimal arrival must be ordered ($a_1 \leq \cdots \leq a_N$) by means of an interchange argument. Assume this is not the case, i.e. $\mathbf{a}$ is an optimal arrival schedule such that $a_i > a_j$ for some $i < j$ (such that $d_i^* < d_j^*$). If we switch between the arrival times of users i and j: $\tilde{a}_i = a_j$ and $\tilde{a}_j = a_i$, while not changing any other arrival time, then because all users have the same service demand the departure times of all other users do not change. Consequently, the departure times are also switched: $\tilde{d}_i = d_j$ and $\tilde{d}_j = d_i$. Therefore, the only change in the total cost function is the change in the cost incurred by i and j themselves. The change in the cost incurred by user i is given by (2.3):

$$\begin{aligned} c_i(\tilde{a}_i, \tilde{d}_i) - c_i(a_i, d_i) &= \big(\tilde{d}_i - d_i^*\big)^2 + \gamma\big(\tilde{d}_i - \tilde{a}_i\big) - (d_i - d_i^*)^2 - \gamma(d_i - a_i) \\ &= (d_j - d_i^*)^2 + \gamma(d_j - a_j) - (d_i - d_i^*)^2 - \gamma(d_i - a_i), \end{aligned} \tag{7.2}$$

and for user j:

$$\begin{aligned} c_j(\tilde{a}_j, \tilde{d}_j) - c_j(a_j, d_j) &= \big(\tilde{d}_j - d_j^*\big)^2 + \gamma\big(\tilde{d}_j - \tilde{a}_j\big) - (d_j - d_j^*)^2 - \gamma(d_j - a_j) \\ &= (d_i - d_j^*)^2 + \gamma(d_i - a_i) - (d_j - d_j^*)^2 - \gamma(d_j - a_j). \end{aligned} \tag{7.3}$$

Summing (7.2) and (7.3) we obtain that the total change in cost is

$$2\big(d_i^* - d_j^*\big)\big(d_i - d_j\big).$$

From Lemma 2.1 we know that if $a_i > a_j$ then $d_i > d_j$, and that by definition $d_j^* > d_i^*$, hence the change in the total cost function is negative which contradicts the assumption that the schedule is optimal. In conclusion, any unordered schedule can be improved by a simple interchange of a pair of unordered coordinates, and therefore an optimal schedule must be ordered.

The slowest service rate occurs when all N users are present in the system, and therefore the longest possible sojourn time is $\frac{1}{\beta - \alpha(N-1)}$. The total time required to clear all users from the system is then coarsely upper bounded by $\frac{N}{\beta - \alpha(N-1)}$. A schedule such that $a_1 < \underline{a}$ is clearly not optimal, since a trivial improvement can always be achieved by setting $a_1 = \underline{a}$ and shifting to the right the arrival times of any user that overlap due to the change in a_1. We are guaranteed this is possible by the fact that all users can arrive and leave the system in the interval $[\underline{a}, d_1^*]$, without any overlaps. Clearly, the deviation from ideal times can only decrease when making this change, while the sojourn times remain unchanged. The coarse upper bound $\overline{a}$ holds for the same reasons. □

Proof of Proposition 3.2 The proof is for Algorithm 1a. The argument for Algorithm 1b follows the same arguments. For every user i, iterating on all possible values of $k_i \in \mathcal{N}$ ensures that every possible departure interval $[a_k, a_{k+1})$ is checked. In a sense, this is an exhaustive search on all solutions that satisfy the dynamics given by Proposition 3.1. Therefore, the algorithm will always converge to the unique solution.

Given a vector of arrivals $\mathbf{a} \in \mathbb{R}^N$, for every $i \in \mathcal{N}$, the departure time d_i occurs in one of the above defined partitions $[a_k, a_{k+1})$, $k \in \mathcal{N}$. The total number of steps

will include the number of "correct" computations, that is for every i and $k_i = k$ the resulting d_i will indeed be in the interval $[a_k, a_{k+1})$. In total there will be exactly N correct computations. However, there will also be steps which will turn out to be false: for a given k the departure time d_i will not be in the interval $[a_k, a_{k+1})$. If $k_j = k$ for some j, then for every $i > j$: $k_i \geq k$. Therefore, if for some i and $k \geq i$ the computation will yield $d_i \notin [a_k, a_{k+1})$, then this interval will not be attempted by any later arrival $j > i$ in the following steps. As a result, every interval will yield at most one false computation. Since there are exactly N intervals this completes the proof. □

Proof of Lemma 5.2 Since $x > a_i$ it holds that $i < \pi_r$, and thus using (5.2) if $k_i < \pi_r$ then $\theta_i = 0$, and if $k_i \geq \pi_r$ then

$$\theta_i = -\frac{\alpha\left(1 - \sum_{j=h_i}^{i-1} \theta_j\right)}{\beta - \alpha(k_i - i)}.$$

Since $N < \beta/\alpha + 1$, the denominator is always positive. We next show that the numerator is non-negative by induction on $h_{\pi_r} \leq i < \pi_r$. Recall that $h_i = \min\{h : k_h \geq i\}$, and so $k_i \geq \pi_r$ is equivalent to $i \geq h_{\pi_r}$. Thus for $j < h_{\pi_r}$: $\theta_j = 0$ and the denominator in the case $i = h_{\pi_r}$ equals $\alpha(1 - 0) > 0$. The induction step is then immediate because the sum $\sum_{j=h_i}^{i-1} \theta_j$ is non-negative for all $h_{\pi_r} < i < \pi_r$. □

Proof of Lemma 5.3 Without loss of generality assume that $\underline{a} + \frac{1}{\beta} < a_i < \overline{a} - \frac{1}{\beta}$, $\forall i \in \mathcal{N}$. If this were not the case we could always extend the search range by $\frac{1}{\beta}$ in both directions. Hence, $\theta_i = 0$ at $x = \underline{a}$ and at $x = \overline{a}$ for any $i \in \pi$. Furthermore, from Lemma 5.2 we have that $\theta_i \leq 0$ for $x > a_i$. Clearly, there is some x such that $\theta_i > 0$ for the first time. So far we have established that θ_i starts at zero, is positive at some point and negative at some back to zero, for every $i \in \pi$. We are left with finding the number of possible sign changes prior to $x = a_i$. For any $x < a_i$ it follows that $i > \pi_r$, and from (5.2) we have that:

$$\theta_i = \frac{\alpha \sum_{j=h_i}^{i-1} \theta_j}{\beta - \alpha(k_i - i)}$$

Note that θ_i can only be negative when there is at least one $j < i$ such that $\theta_j < 0$. We use this to complete the proof by induction on the initial order π. We start the induction at $i = 2$ because $\pi_r = 1$ in the initial permutation and $\theta_{\pi_r} \geq 0$ for all values of x. For $i = 2$ and $x < a_2$: $\theta_2 = \frac{\alpha\theta_1 \mathbb{1}\{h_2=1\}}{\beta - \alpha(k_2 - 2)} \geq 0$. Together with Lemma 5.2 we have established that θ_2 changes sign exactly once. Now let us assume that the claim is correct for all $j \leq i - 1$. From (5.2) we see that for $x < a_i$, θ_j can only change sign when one of the previous $j \in \{h_i, \ldots, i - 1\}$ changes sign. If θ_{i-1} changed sign exactly $i - 2$ times then θ_i can potentially change at all these times and additionally when $x = a_i$, and therefore there are indeed at most $i - 1$ changes of sign. □

References

Arnott R, de Palma A, Lindsey R (1993) A structural model of peak-period congestion: a traffic bottleneck with elastic demand. Am Econ Rev 83(1):161–79

Avram F, Bertsimas D, Ricard M (1995) Fluid models of sequencing problems in open queueing networks; an optimal control approach. Inst Math Appl 71:199

Baker K, Scudder GD (1990) Sequencing with earliness and tardiness penalties: a review. Oper Res 38(1):22–36

Bertsekas DP (1999) Nonlinear programming, 2nd edn. Athena Scientific, Belmont

Cohen J (1979) The multiple phase service network with generalized processor sharing. Acta Inf 12(3):245–284

Daganzo CF (2007) Urban gridlock: macroscopic modeling and mitigation approaches. Transp Res Part B Methodol 41(1):49–62

De Loera JA, Hemmecke R, Kppe M, Weismantel R (2006) Integer polynomial optimization in fixed dimension. Math Oper Res 31(1):147–153

Glazer A, Hassin R (1983) ?/M/1: on the equilibrium distribution of customer arrivals. Eur J Oper Res 13(2):146–150

Harchol-Balter M (2013) Performance modeling and design of computer systems: queueing theory in action. Cambridge University Press, Cambridge

Hassin R (2016) Rational queueing. CRC Press, Boca Raton

Henderson J (1974) Road congestion. J Urban Econ 1(3):346–365

Keha AB, de Farias IR, Nemhauser GL (2006) A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. Oper Res 54(5):847–858

Koshy T (2009) Catalan numbers with applications. Oxford University Press, New York

Mahmassani H, Herman R (1984) Dynamic user equilibrium departure time and route choice on idealized traffic arterials. Transp Sci 18(4):362–384

Murty KG, Kabadi SN (1987) Some NP-complete problems in quadratic and nonlinear programming. Math Program 39(2):117–129

Nazarathy Y, Weiss G (2009) Near optimal control of queueing networks over a finite time horizon. Ann Oper Res 170(1):233–249

Pia AD, Dey SS, Molinaro M (2016) Mixed-integer quadratic programming is in NP. Math Program 162(1):225–240

Pinedo ML (2008) Scheduling: theory, algorithms, and systems. Springer, Berlin

Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. Eur J Oper Res 120(2):228–249

Ravner L, Haviv M, Vu HL (2016) A strategic timing of arrivals to a linear slowdown processor sharing system. Eur J Oper Res 255(2):496–504

Sen T, Gupta SK (1984) A state-of-art survey of static scheduling research involving due dates. Omega 12(1):63–76

Tseng P (2001) Convergence of a block coordinate descent method for nondifferentiable minimization. J Optim Theory Appl 109(3):475–494

Vielma JP, Ahmed S, Nemhauser GL (2010) Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions. Oper Res 58(2):303–315

Weiss G (2008) A simplex based algorithm to solve separated continuous linear programs. Math Program 115:151–198