# Response-Time-Optimised Service Deployment:

## MILP Formulations of Piece-wise Linear Functions Approximating Non-linear Bivariate Mixed-integer Functions

Matthias Keller and Holger Karl

**Abstract**—A current trend in networking and cloud computing is to provide compute resources at widely dispersed places; this is exemplified by developments such as Network Function Virtualisation. This paves the way for wide-area service deployments with improved service quality: e.g., a nearby server can reduce the user-perceived response times. But always using the nearest server can be a bad decision if that server is already highly utilised. This paper formalises the two related problems of *allocating* resources at different locations and *assigning* users to them with the goal of minimising the response times for a given number of resources to use – a non-linear capacitated facility location problem with integrated queuing systems. To efficiently handle the non-linearity, we introduce five linear problem approximationsand adapt the currently best heuristic for a similar problem to our scenario. All six approaches are compared in experiments for solution quality and solving time. Surprisingly, our best optimisation formulation outperforms the heuristic in both time and quality. Additionally, we evaluate the influence ot resource distributions in the network on the response time: Cut by half for some configurations. The presented formulations are applicable to a broader optimisation domain.

**Index Terms**—cloud computing; next generation networking, virtual network function; network function virtualisation; resource management; placement; facility location; queueing model; piecewise linear approximation; optimisation; mixed integer function; derivative; erlang delay formula;

## 1 Introduction

Providing resources at widely dispersed places is a new trend in networking and cloud computing known under different labels: for example, Carrier Clouds [3], [9], [39], Distributed Cloud Computing [2], [16], [34], or In-Network Clouds [20], [37], [38]. These In-Network resources are often geared towards specific network services, e.g., firewalls or load balancers – a development popularised as Network Function Virtualisation [17]. They have an important advantage: Their resources are closer to end users than those of conventional clouds, have smaller latency between user and cloud resource, and are therefore suitable for running highly interactive services. Examples for such services are streaming applications [4], [41], user-customised streaming [5], [23], or cloud gaming [29]. For such applications, the crucial quality metric is the user-perceived *response time* to a request. Large response times impede usability, increase user frustration [10], or prevent commercial success.

As detailed in a prior publication [27], these response times comprise three parts: A request's *round trip time* (RTT), its *processing time* (PT), and its *queuing delay* (QD) when it has to wait for free resources (Figure 1). A first attempt to provide small response times would be to allocate some resources at many sites so that each user has one resource with enough spare capacity nearby. This, however, is typically infeasible as each used resource incurs additional costs. We hence have to decide where user requests shall be processed – the *assignment* decision. Extending our prior work [27], we here additionally decide how many resources shall process the requests at each site – the *allocation* decision. Both decisions are mutually dependent as exemplified in the next section; the resulting problem is called the *assignment and allocation problem*.

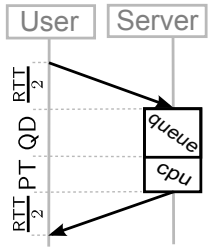

Figure 1: RT = RTT + QD + PT.

The assignment and allocation decisions change the queuing delay in two ways: Allocation modifies the number of resources $y$ at a site; assignment changes the resource utilisation at a site (for a fixed $y$). However, the queuing delays approximately grow exponentially for increasing utilisation. For simple services with short processing times (e.g. static content web server) the delays are still small enough to be ignored. But for computation-intensive services, the long processing time and significant longer queuing delay are negatively perceived by customers. Only little research (Section 2) on server allocation has yet integrated these queuing delays.

This paper investigates deploying computation-intensive *services* at *resources dispersed* through the net. Service requests are issued by users from many locations. The service deployment objective is to minimise the response time while using exactly $p$ resources.

Ignoring the queuing delay for now, assigning user requests to nearby allocated resources reduces the round trip time. By additionally restricting the number of allocated resources to $p$ the problem becomes hard to solve. Ignoring the response time, on the other hand, while minimising the queuing delay, all requests are assigned to a single site to which *all* $p$ resources are allocated [8]. Hence, the queuing delay and round trip time, are minimised by two

- *Manuscript received January xx,yy; revised January xx,yy. This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).*
- *The authors are with the University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany. E-mail: mkeller@upb.de, holger.karl@upb.de*

conflicting allocation schemes, nearby and consolidated. To find an optimal solution *for the sum* of QD and RTT, the two allocation schemes have to be balanced to find assignments and allocations that minimises the response time.

Details on how queuing delay affects response time are exemplified in an extended version of this paper [26]. It also elaborates further the mutual dependency between assignment and allocation decision.

This paper casts the above problem as a queue-extended $p$-median facility location problem (Section 3). To efficiently solve this non-linear problem, five approximate formulations present different modelling approaches and with different trade-offs between accuracy and search space size (Section 4). Additionally, a known-to-be-good heuristic for a similar problem is adjusted to our problem (Section 5). Finally, two aspects are evaluated: First, the five approximations and the heuristic are compared for two metrics, accuracy and solving time (Section 6.1). Second, an additional evaluation discusses how resource distribution in the network reduces response times (Section 6.2). For the last two evaluations, $111{,}600$ problems are solved.

This paper extends our previous work [25] by generalising from M/M/1 to M/M/k queuing models. To do so, the resource allocation has to be decided as well, resulting in a complex problem. Neither problem formulations, approximation approaches, nor evaluation results presented here have been published before.

## 2 Related work

Similar assignment and allocation problems with integrated queuing systems have been investigated before. While this related work focuses on a broader overview, we point interested readers to our detailed comparison of structure and simplifying assumptions of these problems [27].

### 2.1 Assignment and Allocation with Queues

Queuing systems have been integrated in FLP problems before [1], [6], [13], [30], [31], [32], [35], [40], [42] with different objectives. All of them either have a non-linear objective function or non-linear constraints, but no previous work has utilised a non-linear solver. In our previous work [27], we solved a queueing system extended facility location problem by utilising a convex solver and compared its solutions with solutions obtained by solving a linear problem approximation using Gurobi. The approximation was solved magnitudes faster than using a convex solver with only a marginal optimality gap in our experiments. Another, complexer technique uses a cutting plane technique to improve linearisation accuracy [40] but has an unknown optimality gap.

The mentioned papers solve their problems via greedy heuristics [1], [6], [13], [35], [42] or via full enumeration [1] for small instances, e.g., five facilities. The present paper contributes to those papers by describing how to obtain near-optimal solutions for larger input. We also adapt the genetic heuristic of Aboolian et al. [1] to our problem. The heuristic was chosen among the references mentioned above because it is well justified, explained, and showed good results.

Most mentioned work copes with simpler problems than ours. Usually, a smaller search space is obtained, e.g., by predefining assignments [1], [6], [13], [42], [43] or by replacing the non-linear queuing delay part by a constant upper bound [42], [43]. Both simplifications prevent load balancing between facilities that could further reduce the average queuing delay. We do not make such simplifications.

### 2.2 Linearisation

Function linearisation is a technique to approximate a non-linear function over a finite interval by several line segments. All segments together are the approximation and are called in short a piece-wise linear function (PWL function). Applying this technique to the objective function or the constraints any non-linear optimisation problem can be approximated by a linear problem [15], [18]. Section 4.1 provides technical details.

We consider solving time and accuracy simultaneously, and, hence, face the trade-off between few segments and small error. Imamoto et al.'s algorithm [21], [22] (improved by us [27]) obtains segments' start and end basepoints having a very small error for convex, univariate functions. This algorithm is applied to obtain basepoints in this paper.

Rebennack et al. [36] linearised multi-variate functions by first decomposing them into separate independent functions and then recombining their approximations. This approach is limited to separable functions and our function of interest is Erlang-C-based (Section 3.1 Eq. 1), which is not separable.

Vidyarthi et al. [40] refines the piece-wise linear function iteratively by adding basepoints while solving. In contrast, we first compute a tight function approximation which also involves modifying existing basepoints. Then, these basepoints are integrated into the problem to solve. This two-step approach is much simpler to implement and to solve than changing the search space dynamically during solving.

## 3 Problem

This section starts with the scenario description. Then, the corresponding model and optimisation problem are presented.

### 3.1 Model

The scenario (Section 1) is cast as a capacitated $p$-median facility location problem [14]. A bipartite graph $G = (C \cup F,\ E)$ has two types of nodes: *Clients* ($c \in C$) and *facilities* ($f \in F$). Clients correspond to locations where customer requests enter the network. Facilities represent candidate locations with resources executing the service, e.g., data centres. Figure 4 (p. 4) shows such a graph. The round trip time $l_{cf}$ is the time to send data from $c$ to $f$ and back.

The geographically[1] distributed demand is modelled by the request arrival rate $\lambda_c$ for each client $c$. Each facility $f$ has $k_f$ resources available and each resource can process requests at service rate $\mu_f$ – the resource capacity. Modelling heterogeneous facilities can be easily approximated by using two facilities $f_1$ and $f_2$ with $\mu_{f_1} \neq \mu_{f_2}$ but $\forall c: l_{cf_1} = l_{cf}2$; this

1. More precisely, the request arrival and service locations are topologically distributed; the round trip time of a path between two locations only roughly matches its geographically distance. We use "geographical" as an intuitive shorthand.

**Optimisation Problem 1** $\mathsf{QP}(G,\, p,\, \mathsf{T}_*)$, the reference

$$\min_{x,y} \underbrace{\frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c}}_{\text{avg. RTT}} + \underbrace{\frac{\sum_f (\sum_c x_{cf})\, \mathsf{T}_{\mu_f}(\sum_c x_{cf}, y_f)}{\sum_c \lambda_c}}_{\text{avg. time in system}} \tag{4}$$

$$\text{s.t.} \quad \sum_f x_{cf} = \lambda_c \qquad \forall c \quad \text{(demand)} \tag{5}$$

$$\left(\sum_c x_{cf}, y_f\right) \in \mathrm{dom}(T_\mu) \qquad \forall f \quad \text{(capacity)} \tag{6}$$

$$y_f \le k_f \qquad \forall f \quad \text{(count)} \tag{7}$$

$$\sum_f y_f = p \qquad \text{(limit)} \tag{8}$$

introduces inaccuracy by modelling two queues where the precise model would had one. Table 1 lists all variables.

The expected time for computing an answer is obtained by utilising a queuing system. Such a system is modelled at each facility with the usual assumptions: The service times are exponentially distributed and independent. The request arrivals at each client $c$ are described by a Poisson process; client requests can be assigned to different facilities. At one facility, requests arrive from different clients and the resulting arrival process is also a Poisson process, because splitting and joining a Poisson processes results in a Poisson processes. Therefore, we have an M/M/k-queuing model at each facility.

Having this model at hand, the probability that an arriving request gets queued is computed by the Erlang-C formula EC given in (1) [7]. Each facility has to be in steady state with resource utilisation $\rho{=}\lambda/k\mu{<}1$. Derived from EC, the expected *number of requests in the system* (NiS) is $\mathsf{N}(a,\, k)$ (2) with shorthand $a{=}\lambda/\mu$. Similarly, the expected *time a request spends in the system* (TiS) is $\mathsf{T}_\mu(\lambda,\, k)$ (3).

$$\mathrm{EC}(a,\, k) = \frac{\frac{ka^k}{k!\,(k-a)}}{\frac{ka^k}{k!\,(k-a)} + \sum_{i=0}^{k-1} \frac{a^i}{i!}} \qquad \text{(Erlang-C)} \tag{1}$$

$$\mathrm{N}(a,\, k) = \frac{a}{(k-a)}\, \mathrm{EC}(a,\, k) + a \qquad \text{(NiS)} \tag{2}$$

$$\mathrm{T}_\mu(\lambda,\, k) = \frac{1}{\lambda}\mathrm{N}(\lambda/\mu,\, k) = \frac{\mathrm{EC}\,(\lambda/\mu,\, k)}{k\mu - \lambda} + \frac{1}{\mu} \qquad \text{(TiS)} \tag{3}$$

### 3.2 Formulation

The core complexity of our problem comes from two mutually depending decisions made at the same time: Request assignment $x_{cf}$ and resource allocations $y_f$ Table 1. Both are chosen to minimise the average response time. The queuing delay non-linearly depends on both the assigned request and allocated resources . The formulation QP (Problem 1) extends the $p$-median facility location problem P [27] by having additional costs, the queueing delays, at each facility. Constraint (5) ensures that all demand is assigned. The assignments at each facility $f$, $\sum_c x_{cf}$, must not exceed $f$'s processing capacity $y_f \mu_f$ (6), which is the processing capacity per resource $\mu_f$ times the number of allocated resources $y_f$. The allocated resources $y_f$ do not exceed the local (7) and global (8) limit.

Table 1: Model variables

| | |
|---|---|
| Input: | |
| $G = (V, E, l_{**}, \lambda_*, \mu_*, k_*)$ | Bipartite Graph with $V = C \cup F$, $C \cap F = \varnothing$ with client nodes $c \in C$ and facility nodes $f \in F$ |
| $l_{cf} \in \mathbb{R}_{\ge 0}$ | Round trip time between $c$ and $f$ |
| $\mu_f \in \mathbb{R}_{>0}$ | Service rate as capacity at $f$ |
| $k_f \in \mathbb{N}_{>0}$ | Number of servers available at $f$ |
| $\lambda_c \in \mathbb{R}_{\ge 0}$ | Arrival rate as demand at $c$ |
| $\alpha_i;\ \beta_i$ | $i$-th basepoint $g(\alpha_i){=}\beta_i$ of PWL $\tilde{g}$ |
| $p = \sum_f y_f$ | Limit on maximal resources to allocation |
| Decision variables: | |
| $x_{cf} \in \mathbb{R}_{\ge 0}$ | Assignment in demand units |
| $y_f \in \mathbb{N}_{>0}$ | Number of allocated resources at $f$ |
| $\dot{y}_f;\ \dot{y}_{fj} {\in} \{0,1\}$ | Indicator: Open fac. $f$; with $j$-th curve active |
| $z_{fi};\ z_{fji} {\in} [0,1]$ | Weight: $i$-th basepoint at $f$; of $j$-th curve |
| $h^u_{fji};\ h^l_{fji} {\in} \{0,1\}$ | Indicator: $j, i$-th triangle at $f$ |

Notation shorthand: $k$ or $k_*$ refers to the tuple/vector $(k_1, .., k_{|F|})$; and $x_{c*}$ refers to matrix slice $(x_{c1}, .., x_{c|F|})$.

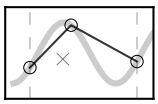

Figure 2: 1D-PWL example.

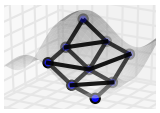

Figure 3: 2D-PWL example.

## 4 Linearisation

The problem QP is non-linear and could be solved by a non-linear solver. In previous work [27], a simpler problem was successfully approximated and was solved by a linear solver fast and without substantial quality loss. Because of these encouraging results, we also follow the linearisation approach in this paper. QP is complexer than our previous problem: QP additionally decides on the number of allocated resources at each site. This turns $\mathsf{T}_\mu$ into a bivariate function (with two instead of one parameter). In addition, the second parameter is integer, making it difficult to apply standard linearisation formulations.

The remaining section describes the linearisation technique in general (Section 4.1) and reformulates QP by linearising either several curves separately (Section 4.2, Section 4.3) or together as a surface (Section 4.4, Section 4.5, Section 4.6). The extended version of this paper [26] additionally discusses potential problem simplifications only allowed for convex cost functions. Additionally, it contains technical details on efficient computation techniques for Erlang-C-based functions.

### 4.1 Piece-wise linear

Any non-linear, univariate function $g$ can be approximated over a finite interval by multiple line segments. A function composed of such segments is called a piece-wise linear(PWL) function [18]; we denote the PWL approximation of $g$ as $\tilde{g}$ (strictly speaking it is a piece-wise affine function). As an example, Figure 2 shows $g(x){=}\sin(x)$ and two segments approximating $g(x)$. For $m$ basepoints with coordinates $(\alpha_i,\, \beta_i{=}g(\alpha_i))$, $0{\le}i{<}m$, a continuous PWL function can be defined (9) by linearly interpolating between two adjacent

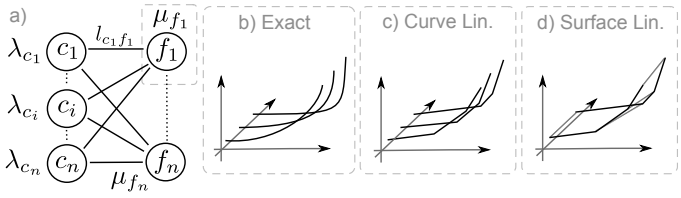


Figure 4: Bipartite graph of a facility location problem with queuing systems at each facility (a) and zoomed bivariate time in system version: exact (b), curve-based (c) and surface-based (d).

basepoints (black circles). The interval is implicitly defined by the outer basepoints, $[\alpha_0, \alpha_{m-1}]$.

$$\tilde{g}(x) = \begin{cases} (\alpha_1{-}\alpha_0)(x{-}\alpha_0) + \beta_0 & \text{if } \alpha_0{\leq}x{<}\alpha_1 \\ (\alpha_2{-}\alpha_1)(x{-}\alpha_1) + \beta_1 & \text{if } \alpha_1{\leq}x{<}\alpha_2 \\ \quad \dots \end{cases} \quad (9)$$

The approximation error $\epsilon$ measures the approximation accuracy as the maximal difference between $g$ and $\tilde{g}$, $\epsilon{=}\max_x |g(x) - \tilde{g}(x)|$. Because the PWL function definition in (9) is not directly understood by an MILP solver it has to be transformed. We use the convex-combination method [12] to model continuous, univariate PWL functions. The cases of (9) are substituted by a convex combination of the basepoint coordinates (10) with weights $z_i$.

$$\tilde{y}{=}\tilde{g}(x) \Leftrightarrow \sum_i z_i\alpha_i{=}x,\ \sum_i z_i\beta_i{=}\tilde{y},\ \sum_i z_i{=}1,\ \mathsf{SOS2}(z_*) \quad (10)$$

In addition, at most two adjacent basepoint weights are allowed to be non-zero, $z_i, z_{i+1}{>}0$, $z_j{=}0$, $j{<}i \vee i{+}1{<}j$. Most optimisation solvers support such restriction through specifying special order sets[2]; with such additional structure information the branch and bound methods can explore the search space more intelligently. Otherwise, coordinates could be expressed that are not located on line segments, e.g., the cross in Figure 2.

Similar to univariate functions, bivariate functions $g(x, y){=}z$ can be approximated by triangles instead of segments (Figure 3). Then, the convex combination is extended by one parameter (11) and weights for only three vertices of a single triangle are non-zero instead of the two segment vertices in the univariate version [18].

$$\begin{aligned} \tilde{w}{=}\tilde{g}(x,y) \ \Leftrightarrow & \sum_i z_i\alpha_i{=}x,\ \sum_i z_i\beta_i{=}y, \\ & \sum_i z_i\theta_i{=}\tilde{w},\ \sum_i z_i{=}1,\ \mathsf{SOS2}(z_*) \end{aligned} \quad (11)$$

The particular challenge here is to deal with an objective function having bivariate cost functions with one integer parameter (the number of resources). The following two sections present different linearisation strategies for this challenge.

### 4.2 Curves

Linearising QP boils down to linearising the non-linear part $\mathsf{T}_\mu(\lambda, k)$ (3) of the objective function (4). However, linearising $\mathsf{T}_\mu$ is not obvious as the second parameter is integer. So, $\mathsf{T}_\mu$ is reformulated as $n$ separate univariate functions $\mathsf{T}_{\mu,j}(\lambda){=}\mathsf{T}_\mu(\lambda, j)$, $j = 1..k_f$ (Figure 4b).[3] If a facility is allocated with exactly $j$ resources, we call the corresponding function $\mathsf{T}_{\mu,j}$ a *curve*. Linearising all these functions (Figure 4c) also obtains a linearisation of the mixed integer function $\mathsf{T}_\mu$ (3).

Each function $\mathsf{T}_{\mu,j}$ is convex, so we can use our algorithm [27] to obtain the PWL function $\tilde{\mathsf{T}}_{\mu,j}$ with high approximation accuracy with few basepoints. Assuming $m$ basepoints are used for one function, one facility needs $mk_f$ basepoints. Technically, the problem allows that all facilities have different service rates so that all basepoint coordinates are different, yielding a total of $m\sum_f k_f$ basepoints[4] The coordinates of a single basepoint $(\alpha_{fji}, \beta_{fji})$ belongs to facility $f$, curve $j$, and $i$-th basepoint of the function $\tilde{\mathsf{T}}_{\mu,j}$.

For the problem formulation, two groups of decision variables are introduced. Firstly, we reformulate the integer variable $y_f$ of decision problem QP into a vector of binary decision variables $\dot{y}_{fj}$ exactly one of which is 1 ($\forall f{:}\mathsf{SOS1}(\dot{y}_{f*})$ ),[5] each one representing that facility $f$ uses $j$ resources if and only if $\dot{y}_{fj}{=}1$. Technically, $\dot{y}_{fj}$ is also used to select the $i$th curve in the computation of the time in system function. Formally: $y_f{=}\sum_j j\ \dot{y}_{fj}$ and the time in queuing system becomes then $\mathsf{T}_{\mu_f}(\lambda, k){=}\sum_j \dot{y}_j\mathsf{T}_{\mu,j}(\lambda)$.

Secondly, continuous weights $z_{fi}$ are used to express the convex combination of basepoints (Section 4.1), representing the utilisation and the corresponding TiS at each facility. For each facility $f$, we need $m$ weights since only a single curve is selected by the decision variables $\dot{y}_{fj}$ (and not $mk_f$ weights per facility $f$, as one might suspect). For the $j$th curve, the linearisation of $\tilde{\mathsf{T}}_{\mu,j}(\lambda)$ is then formulated as $\sum_i z_{fi}\beta_{fji}$ with $\lambda$ as $\sum_i z_{fi}\alpha_{fji}$.

In the resulting problem formulation cQP3[6] (Problem 2), QP's objective function (4) is transformed as described above, replacing (4) by (12) and by (18). : The new constraint (15) ensures a convex combination only of neighbouring weights. The capacity constraint (14) replaces (6). The local and global limits are adjusted using $\dot{y}$; constraints (7), (8) are replaced by (16), (17).

The right term of the objective function (12) multiplies three decision variables, $x\dot{y}z$, making the problem cubic. Transforming it into a linear problem is explained in the remainder of this section.

To do so, at first, the factor of cQP3's time-in-system function (12) is integrated in the basepoint coordinates. To understand this modification, QP's objective function (4) has to be revisited. In this function, the time-in-system function $\mathsf{T}_\mu$ has the same factor $\sum_c x_{cf}{=}w$. Integrating $w$ yields $w\mathsf{T}_\mu(w, y){=}\mathrm{N}(w/\mu, y)$ with function $\mathrm{N}(\cdot)$ being already defined in (2). The problem QP can be reformulated by replacing constraints (4), (6) with (19), (20).

2. $n$-Special Ordered Sets $\mathsf{SOS}n(V)$ are special constraints limiting decisions variables of a list $V$ so that at most $n$ adjacent variables are non-zero. Notation: $\mathsf{SOS2}(z_*)$ is a shorthand for $\mathsf{SOS2}(z_1, .., z_m)$ (Table 1).

3. This paper uses the notation $f_a(x)$ to indicate that $f$ is a function with constants $a$ and variables $x$, e.g., $f_1(x), f_2(x)$ are separate functions of $x$.

4. Basepoint coordinates can be shared among facilities with same service rates. If all resources are homogeneous, only $m\,max_f k_f$ different basepoints exists.

5. Notation: Shorthand $\dot{y}_{f*}$ for $\dot{y}_{f1}, .., \dot{y}_{fn}$ (Table 1)

6. We use the following naming convention for optimisation problems: "3" indicates the cubic nature of this problem; the prefix "c" indicates the curve-based approximation of the time-on-system function, later to be complemented by "t" for triangular formulations and "q" for quadrilateral formulations.

**Optimisation Problem 2** cQP3($G$, $p$, $\tilde{\mathsf{T}}_{**}$)
separate curves, cubic

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\sum_f (\sum_c x_{cf}) \overbrace{\sum_j \dot{y}_{fj} \sum_i z_{fi} \beta_{fji}}^{\text{time in system } f}}{\sum_c \lambda_c} \quad (12)$$

$$\text{s.t.} \quad \sum_f x_{cf} = \lambda_c \qquad \forall c \quad \text{(demand)} \quad (13)$$

$$\sum_c x_{cf} \leq \sum_j \dot{y}_{fj} \sum_i z_{fi} \alpha_{fji} \quad \forall f \quad \text{(capacity)} \quad (14)$$

$$\sum_i z_{fi} = 1,\ \mathsf{SOS2}(z_{f*}) \qquad \forall f \quad \text{(weights)} \quad (15)$$

$$\sum_j j\, \dot{y}_{fj} \leq k_f \qquad \forall f \quad \text{(count)} \quad (16)$$

$$\sum_{fj} j\, \dot{y}_{fj} = p \qquad \text{(limit)} \quad (17)$$

$$\mathsf{SOS1}(\dot{y}_{f*}) \qquad \forall f \quad \text{(curve flip)} \quad (18)$$

$$\min_{x,y} \sum_{cf} x_{cf} l_{cf} + \sum_f \mathsf{N}(\frac{\sum_c x_{cf}}{\mu_f},\, y_f) \quad (19)$$

$$\text{s.t.} \quad \left(\frac{\sum_c x_{cf}}{\mu_f}, y_f\right) \in \text{dom}(\mathsf{N}) \quad \forall f \quad \text{(capacity)} \quad (20)$$

Then, the adjusted QP is linearised as before: Function $\mathsf{N}$ is also mixed-integer and separate functions $\tilde{\mathsf{N}}_j$ are linearised. And to integrate the inner function $w/\mu$ of $\tilde{\mathsf{N}}_j$, the convex combination of basepoints are adjusted from (21) to (22).

$$\tilde{\mathsf{N}}_k(a = w/\mu) \Leftrightarrow \sum_i \alpha_i z_{fi} = a, \quad \sum_i z_{fi} \beta_i = \mathsf{N}_k(a) \quad (21)$$

$$\tilde{\mathsf{N}}_k(a = w/\mu) \Leftrightarrow \mu \sum_i \alpha_i z_{fi} = w, \sum_i z_{fi} \beta_i = \mathsf{N}_k(a) \quad (22)$$

The resulting linearisation cQP2 (now quadratic, hence a "2") results from replacing constraints (12),(14) with (23), (24):

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\sum_{fj} \dot{y}_{fj} \sum_i z_{fi} \beta_{ji}}{\sum_c \lambda_c} \quad (23)$$

$$\sum_c x_{cf} \leq \mu_f \sum_j \dot{y}_{fj} \sum_i \alpha_{ji} z_{fi} \ \forall f \quad \text{(capacity)} \quad (24)$$

In consequence, cQP2 needs fewer basepoints than cQP3, because $\tilde{\mathsf{N}}_j$ depends only on the number of allocated resources $j$ but is now independent of $\mu$; in total $m \max_f k_f$ basepoints are necessary.

The objective function of cQP2 (23) is quadratic ($\dot{y}z$). We further simplified the problem by two modifications to derive a linear problem formulation: Firstly, more weights $z$ are used; previously only $m$ weight per facility $f$ is modelled whereas now $mn$ weights are modelled; $z_{fji}$ for facility $f$, curve $j$, basepoint $i$. Doing so yields an equivalent problem with increased search space. Secondly, $\dot{y}$ is turned into a constraint enforcing that only those weights $z_{fji}$ are non-zero which correspond to the $j$-th active curve at facility $f$, $\dot{y}_{fj}$. Several test runs showed that the resulting linear problem is solved faster than its quadratic counterpart despite its larger search space.

**Optimisation Problem 3** cQP1($G$, $p$, $\tilde{\mathsf{N}}_*$)
separate curves, linear

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\overbrace{\sum_{fji} z_{fji} \beta_{ji}}^{\text{total time in systems}}}{\sum_c \lambda_c} \quad (25)$$

$$\text{s.t.} \quad \sum_f x_{cf} = \lambda_c \qquad \forall c \quad \text{(demand)} \quad (26)$$

$$\sum_c x_{cf} \leq \mu_f \sum_{ji} z_{fji} \alpha_{ji} \quad \forall f \quad \text{(capacity)} \quad (27)$$

$$\sum_i z_{fji} = 1,\ \mathsf{SOS2}(z_{fj*}) \ \forall f, j \quad \text{(weights)} \quad (28)$$

$$\sum_i z_{fji} = \dot{y}_{fj} \qquad \forall f, j \quad \text{(sync)} \quad (29)$$

Constraints (16), (17), (18)

The resulting linear formulation cQP1 (Problem 3) has a new constraint (29), ensuring that only the relevant weights are allowed to be non-zero. Having this constraint in place, the new objective function (25) equals the old objective function (23). Constraints (27), (28) now support the new $j$ index for weights $z$. This way the new linear problem cQP1 computes the same solution as the previous quadratic problem cQP2.

### 4.3 Thinned Curves

Problem cQP1 uses $mn$ basepoints at each facility. For example, for 100 facilities with 40 resources available, cQP1 has $n$=40 separate curves at each facility. When using $m$=10 basepoints per curve, the search space contains 40,000 weight decision variables. The search space can be reduced by reducing $m$ or $n$, but this lowers accuracy. Reducing the number of basepoints is a straightforward trade-off of accuracy against search space size, but simply reducing the number of resources $n$ is not adequate as this modifies the problem instance. Hence, to obtain a similar trade-off for the resources, we need to find a way to reduce the amount of resources to look at – one option would be to allow gaps in the sequence of number of allocated resources, with appropriate rounding, $J$=$[1, .., n]$. For example, with 40 available resources at a facility, we could remove the option to use, say, 26 resources, forcing the solution to either use 25 or 27 resources instead, $J$=$[1, .., 25, 27, .., n]$. In the example with 100 facilities and $m$=10 basepoints this removes 1000 decision variables. The sequence $J$ specifies which options $j \in J$ for the number of allocated resources at one facility are available. As an example, Figure 5 shows $\mathsf{N}(a, j)$=$t$ from the top with contour lines for $t$. Basepoints are plotted as crosses for $\tilde{\mathsf{N}}_k(a)$, $j \in J = [1, 2, 4, 8, 16, 32, 40]$ with $n$=7, $m$=6. With $|F|$=100 facilities, only $|F|mn$=4200 weight decision variables are needed as opposed to 40,000 variables with $n$=40 and $m$=10.

Problem cQP1 (Problem 4) is adjusted by adding parameter $J$ and replacing the constraints (16), (17) by (30), (31). By dropping a particular $j$ from $J$, two issues can arise: a) In special cases the problem becomes infeasible. b) The number of allocated resources $y_f \in J$, $y_f = \sum_{j \in J} j\, \dot{y}_{fj}$ cannot satisfy $\sum_f y_f = p$; this is the reason why Constraint (31) is relaxed to

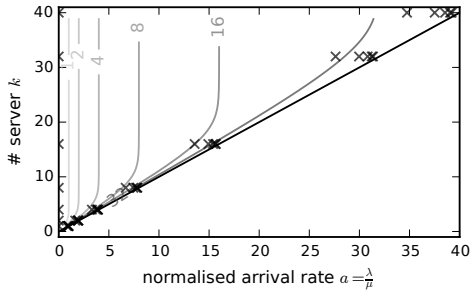


Figure 5: Contour plot for $\mathsf{N}_j(a)$ with basepoints for $j{\in}J{=}[1,2,4,8,16,32,40]$ of $\tilde{\mathsf{N}}_J(a,k)$

**Optimisation Problem 4** $\mathsf{cQP}(G,\ p,\ \tilde{\mathsf{N}}_*,\ J)$, thinned curves

Obj. func./Constr. (25),(18),(26),(27),(28),(29)

$$\text{s.t.} \quad \sum_{j\in J} j\ \dot{y}_{fj} \le k_f \quad \forall f \qquad \text{(count)} \tag{30}$$

$$\sum_{f,\,j\in J} j\ \dot{y}_{fj} \le p \qquad \text{(limit)} \tag{31}$$

an upper bound. To illustrate both issues, consider the following example: Resources should be allocated at 6 facilities deciding values for $y_{1..6}$. The facilities have $k_f{=}30$ resources available and all resources are homogeneous. The demand should be handled by $p{=}18$ resources and needs at least 15 resources, $\lceil\frac{\sum_c \lambda_c}{\mu}\rceil{=}15$. Basepoints exist for $\forall j{\in}J\ :\ \tilde{\mathsf{N}}_j(a)$. Starting with all curves available, $J{=}[1,...,30]$, the optimal allocation is $y_{1..6}=3{\in}J$. A thinned $J{=}[1,30]$ renders cQP infeasible. Allocating 30 resources at one location violates the upper limit, $30{\neq}18{=}p$. Similarly, allocating one resource at each facility violates the limit, $6{\neq}18{=}p$. Removing fewer $j$ from $J$ reduces the likeliness of this special case.

To illustrate issue b) from above, consider cQP with $J{=}[1,10,20,30]$. Under constraint (31), this problem is feasible, e.g., with allocation $y_{1..6}{=}10,1,1,1,1,1$ which satisfies the limit $\sum_f y_f{=}16 \le p{=}18$ and also satisfies all demand (it would not be feasible if constraint (31) stipulated equality). However, the downside is that not all $p$ resources are used and queuing delay could be reduced further: Adding a resource at any facility always reduces that facility's queuing delay. So adding the remaining resources, $\sum_f y_f{-}p$, will always improve the solution. The algorithmsALLOC and MAXCOSTDROP increase resources at those facilities where the queuing delay is reduced the most. The auxiliary algorithm ALLOC merely recasts many QP variables into a formulation suitable for the generic greedy algorithm MAXCOSTDROP. This algorithm considers each facility as a bucket into which $n=\sum_f y_f{-}p$ tokens are to be distributed – the available resources. Placing a token into a bucket represents adding a resource to that facility and hence a reduction of the queueing delay. MAXCOSTDROP then looks for a token distribution that maximizes the reduction in cost/queuing delay. The remaining section proves the correctness.

**Theorem 1.** MAXCOSTDROP$(n,\ c_*(y),\ y^{max})$ (Algorithm 2) maximises $\sum_f c_f(y_f)$ ensuring $\forall b{:}y_f < y_f^{max}$, $\sum_f y_f{=}n$ for any decreasing and convex cost function $c_f(x)$.

**Algorithm 1** ALLOC$(p,\ \lambda',\ y{=}0,\ F'{=}F)\ \to\ y, T$
Optimal allocation for known assignment.

**requires** $\lambda'_f{<}k_f\mu_f, \sum_f\lceil\lambda'_f/\mu_f\rceil{\le}p$
**ensures** $\lambda'_f{<}y_f\mu_f, \sum_f y_f{=}p$
minimises $\sum_f \mathsf{N}(\lambda'_f/\mu_f, y_f)$
1: $\forall f{\in}F' : a_f{\leftarrow}\lambda'_f/\mu_f;\ \ y_f^{\min}{\leftarrow}\max\{y_f,\ \lceil a_f\rceil\}$
2: $n \leftarrow p{-}\sum_f y_f^{\min}$
3: **if** $n{>}0$ **then**
4: $\quad x \leftarrow$ MAXCOSTDROP$(n,\ c(x),\ x^{\max})$
$\quad\quad\quad \forall f : c_f(x) := \mathsf{N}(a_f, y_f{+}x)$
$\quad\quad\quad \forall f : x_f^{\max} \leftarrow k_f{-}y_f^{\min}$
5: $\quad \forall f : y_f = y_f^{\min} + x_f$
6: **else**
7: $\quad \forall f : y_f = y_f^{\min}$
8: **return** $y,\ \sum_f \mathsf{N}(a_f, y_f)$

**Algorithm 2** MAXCOSTDROP$(n,\ c_*(y'),\ y^{\max})\ \to\ y$
Drops $n$ tokens in $m$ buckets while minimising total drop costs under bucket capacity constraint.

**requires** $\forall 1{\le}b{\le}m : c_b(x)$ is decreasing and convex
**ensures** $\forall b : y_b{<}y_b^{max}$, $\sum_b y_b{=}n$
maximises $\sum_b c_b(y_b)$
1: $S \leftarrow \{y_{bi}\,|\,1{\le}b{\le}m, 1{\le}i{\le}y_b^{\max}\}$
$\quad I \leftarrow \{S'{\subseteq}S\,|\,n{\ge}|S'|\}$
$\quad w(y_{bi}) := c_b(i) - c_b(i{-}1)$ **if** $y{>}1$ **else** $c_b(0)$
2: $A \leftarrow \emptyset$
3: **for** $y \in S$, sorted by non-increasing weights **do**
4: $\quad$ **if** $|A| < n$ **then** $|A| \leftarrow A \cup \{y\}$
5: $\forall f{:}x_f \leftarrow |\{y_{bi}{\in}A\,|\,1{\le}i{\le}y_b^{\max}\}|$
6: **return** $x_*$

*Proof.* For a weighted matroid $M{=}(S,I)$, algorithm GREEDY ( [11], Theorem 16.10, p. 348ff) computes a subset $A$ with maximal weight. Line 1 defines a weighted matroid $M{=}(S,I)$: $S$ is non-empty; $I$ is hereditary meaning $\forall B{\in}I, A{\subseteq}B\ :\ A{\in}B$; $M$ satisfies the exchange property meaning $\forall A,B{\in}I, |A|{<}|B| : x{\in}B{-}A\ \wedge\ A{\cup}\{y\}{\in}I$; $w(y)$ is positive. The lines (2)–(4) are Corman's GREEDY casted to our $M$. By adding $y_{bi}$ to subset$A$, one token is added to bucket $b$; so the number of tokens in each bucket $x_b$ can be aggregated as in line (5). In particular, the following property holds: $\forall b,i{:}y_{bi}$ was added before $y_{b(i+1)}$. This is ensured by the weight sorting (line (3)) and $c_b(y)$ being decreasing and convex; then $\forall b,i{:}w_{bi}{>}w_{b(i+1)}$. From the same property the costs can be derived as $\forall b{:}c_b(y_b){=}\sum_{y_{bi}\in A} w(y_{bi})$. From Theorem 16.10, the computed subset $A{\subset}S$ maximises $\sum_{y_{bi}\in A} w(y_{bi})$ which also maximises $\sum_b c_b(y_b)$. □

**Lemma 1.** ALLOC$(G,\ \lambda',\ p)$ (Algorithm 1) allocates $p$ resources at facilities $f{\in}F'$ so that $\sum_f \mathsf{N}(\lambda'_f/\mu_f, y_f) = T$ is minimised while ensuring $\lambda'_f{<}y_f\mu_f$ and $\sum_f y_f{=}p$.

*Proof.* Line (1) allocates the resources $y_f^{\min}$ at least necessary to handle assigned demand $\lambda'_f$ having $n{\ge}0$ resources still to be allocated ( $n{<}0$ contradicts $\lambda'_f$'s requirements). With $n{=}0$ the computed minimal allocation is the only one and, hence, $T$ is minimal; done. With $n{>}0$ greedy algorithm MAXCOSTDROP is used, where dropping tokens into buckets correspond to increasing resource at facilities. The token cost

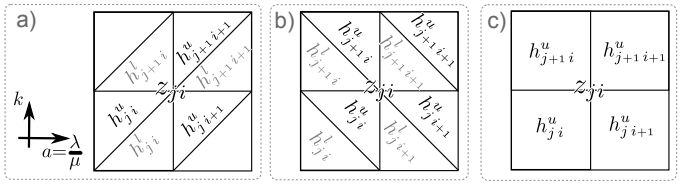


Figure 6: Surface approximations with MILP formulation relationship; a) tQP$_-$, b) tQP$_+$, and c) qQP.

function $c_f(y)$ and $N_a(y)$ is decreasing and convex and $y_f^{max}$ specifies the capacity of bucket $f$. The $n$ tokens dropped in buckets, resulting in distribution vector $y$, maximises the total token cost $\sum_f c_f(y_f)$ (Theorem 1). The token costs correspond to the reduction, $\forall_f N_a(y_f^{min}) - N_a(y_f^{min}+y'_f) = c_f(y'_f)$, incurred by allocating $n$ additional resources at $f$. Coming from the minimal allocation ($y^{\min}$, line (1)) with minimal $T$ and adding resources that maximal reduces $T$, then the new $T$ is also minimal; done. □

### 4.4 Surface with Triangles

As discussed in the previous section, dropping separate curves from $J$ limits feasibility. In order to overcome this issue, the previously separate univariate PWL functions are now joined into one bivariate PWL function. This is done by creating a mesh of triangles over all basepoints of all curves in $J$ (Section 4.1); such a mesh defines also the surface of the approximation (Figure 4d). Doing so, the problem of missing curves goes away as they are implicitly represented by interpolating neighbouring curves. Hence, the set of basepoints can be thinned out more aggressively to further reduce the search space without jeopardising feasibility.

This section discusses three arising questions: 1) Using neighouring curves' basepoints for interpolation introduces inaccuracy – how large is the drop in accuracy? 2) Modelling triangles is complexer than modelling line segments – will a smaller search space compensate for a more complex optimisation problem? The convex combination MILP formulation introduces continuous weights rendering $y_f$ to be continuous – how to treat fractional server allocations?

The approximation surface $\tilde{S}$ (the mesh of triangles between basepoints of curves $J$) linearly approximates the surface $S$ of the original function $g$, which is in our case $\mathsf{N}(a,k)$ ;Figure 5 shows its contour for $1{\le}k{\le}40$ and corresponding basepoints of an example $J$. The triangle mesh (Figure 6a) has $mn$ basepoints. As before, the basepoints $\alpha_{ji}, \beta_{ji}, \theta_{ji}$ touching the surface, $g(\alpha_{ji},\beta_{ji}){=}\theta_{ji}$, $0{\le}j{<}n$, $0{\le}i{<}m$. Using the convex combination MILP formulation (32), each basepoint has a weight $z_{ji}$ as the decision variable. Only the three edges of exactly one triangle form the convex combination (SOS3) [15], [18]. Since current solvers do not support SOS3 constraints, D'Ambrosio et al. [15] presented an equivalent formulation (33) only using SOS1 constraints: A new miscellaneous, binary decision variable $h$ for each triangle indicates whether the triangle is active, only one $h$ and its triangle weights are allowed to be non-zero while all other weights are forced to zero. The formulation (33) corresponds to triangle enumeration and orientation in Figure 6a. Integrating the triangle formulation from (32), (33) in cQP results in tQP$_-$ (Problem 5).

$$\sum_{ji}\alpha_{ji}z_{ji}{=}x,\ \sum_{ji}\beta_{ji}z_{ji}{=}y,\sum_{ji}\theta_{ji}z_{ji}{=}z=g(x,y),$$
$$\mathsf{SOS3}(z_{**}),\ \sum_{ij}z_{ij}=1 \tag{32}$$

$$\sum_{ji}h^u_{ji}{+}h^l_{ji}{=}1,\ \mathsf{SOS1}(h^*_{**}),\ h^*_{0*}{=}h^*_{*0}{=}h^*_{m*}{=}h^*_{*n}{=}0$$
$$\forall ji: z_{ji} \le h^u_{ji}{+}h^l_{ji}{+}h^u_{i+1\,\,j}{+}h^l_{i+1\,\,j+1}{+}h^u_{i+1\,\,j+1}{+}h^l_{i\,\,j+1} \tag{33}$$

**Optimisation Problem 5** tQP$_-(G,\ p,\ \tilde{\mathsf{N}})$, with triangles

$$\min_{\substack{x,y,\\ z,h}} \frac{1}{\sum_c\lambda_c}\sum_{cf}x_{cf}l_{cf}+\underbrace{\frac{1}{\sum_c\lambda_c}\sum_{fji}z_{fji}\theta_{ji}}_{\text{sum of all TiS}} \tag{34}$$

s.t.
$$\sum_f x_{cf}=\lambda_c \qquad \forall c \quad \text{(demand)} \tag{35}$$
$$\sum_c x_{cf} \le \underbrace{\mu_f\sum_{ji}z_{fji}\alpha_{ji}}_{\text{capacity at } f} \qquad \forall f \quad \text{(capacity)} \tag{36}$$
$$\underbrace{\sum_{ji}z_{fji}\beta_{ji}}_{\#\text{ server used at } f} \le k_f \qquad \forall f \quad \text{(count)} \tag{37}$$
$$\sum_{fji}z_{fji}\beta_{ji}=p \qquad \text{(limit)} \tag{38}$$
$$\sum_{ji}z_{fji}=y_f \qquad \forall f \quad \text{(open)} \tag{39}$$
$$\sum_{j'i'}h^*_{fj'i'}=y_f \qquad \forall f \quad \text{(o-sync)} \tag{40}$$
$$\sum_{j'i'}h^u_{fj'i'}+h^l_{fj'i'}=1,$$
$$\mathsf{SOS1}(h^*_{f**}) \qquad \forall f \quad \text{(single-tri)} \tag{41}$$
$$h^*_{f0*}{=}h^*_{f*0}{=}h^*_{fm*}{=}h^*_{f*n}{=}0 \qquad \forall f \quad \text{(tri-corner)} \tag{42}$$
$$z_{ji} \le h^u_{fji}+h^l_{fji}+h^u_{f\,j\,i+1}+h^l_{f\,j+1\,i+1}+$$
$$h^u_{f\,j+1\,i+1}+h^l_{f\,j+1\,i} \qquad \forall ji \quad \text{(tri-sync)} \tag{43}$$
with $0{\le}j'{\le}n$, $0{\le}i'{\le}m$

While D'Ambroisio et al. focus on tight approximation using many basepoints, we want to reduce solving time in addition which depends on the number of used decision variables and basepoints. So in a nutshell, we aim for large triangles with an approximation error as small as possible; the error is the maximal difference between the original surface $S$ and the triangle's surface $\tilde{S}$. Beside good basepoints, the remaining section discusses first a changed triangle orientation and afterwards replacing the triangles with quadrilaterals.

Four groups of decision variables form the search space: a) The request assignment $x_{cf}$; b) binary $y_f$ activates facility $f$ and considers time in system only for active facilities; c) the weights $z_{fji}$ of the approximated surfaces at each $f$; d)the miscellaneous variable $h^u_{fji}$, $h^l_{fji}$ for the upper and lower triangles ensure that weights $z_{fji}$ are non-zero if exactly one triangle at each facility is active.

As a variant, the triangle direction can be flipped (Figure 6b). The adjusted formulation tQP$_+(G,\ p,\ \tilde{\mathsf{N}})$ uses objective function and constraints from (34)–(42) and replaces constraint (43) by (44).

$$z_{ji} \le h^u_{fji}+h^l_{f\,j\,i+1}+h^u_{f\,j\,i+1}+h^l_{f\,j+1\,i+1}+$$
$$h^u_{f\,j+1\,i}+h^l_{f\,j+1\,i} \quad \forall ji \quad \text{(tri-sync)} \tag{44}$$

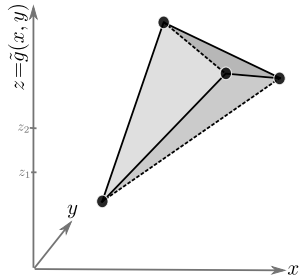


Figure 7: Triangle surface between four basepoints.

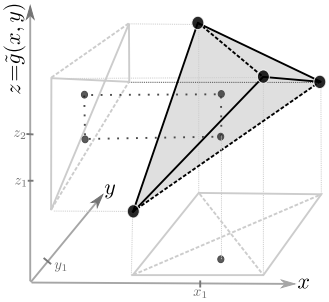


Figure 8: Quadrilateral surface; convex hull of convexly combining four basepoints.

Comparing the approximations of both triangle orientations, the resulting surfaces $\tilde{S}^+, \tilde{S}^-$ are obviously different. Zooming into two adjacent triangles, Figure 7 shows four example basepoints and the two triangles of each of the two orientations. The four triangles are differently grey shaded (one is completely overlapped in the back). Two triangles connected by the dashed diagonals either form an upper $\tilde{S}^+$ or lower surface $\tilde{S}^-$, depending on the orientation and on the differences of the basepoint coordinates.

All basepoints lie on the approximated surface $S$. At any other points, at a given $(x, y)$ coordinate, there are two points $(x, y, z_s) \in S$ (the original point) and $(x, y, z_{\tilde{s}}) \in \tilde{S}$ (its approximation); typically, $z_S \neq z_{\tilde{S}}$, inducing some approximation inaccuracy. In general, this inaccuracy depends on the triangle orientation, the number of used basepoints, and the basepoint coordinates itself. We discuss the generation of basepoints to maximise accuracy in the extended version.

### 4.5 Surface with Quadrilaterals

Triangle-based approximations can be found in the literature. We explore here an alternative, namely an approximation where the basepoints form quadrilaterals rather than triangles. The hope is again to reduce the search space.

For approximating a function $g(x, y){=}z$ a single triangle can be described by three basepoints and all points within the triangle by convexly combining the basepoints (45). Similarly, four basepoints of a quadrilateral can be used (46). However, while equations (45) form a linear system having a unique solution, the corresponding equations (46) form an underdetermined linear system having more unknowns ($z_i$) than equations.

$$\sum_{i=0}^{2} \alpha_i w_i{=}x,\ \sum_{i=0}^{2} \beta_i w_i{=}x,\ \sum_{i=0}^{2} \theta_i w_i{=}x \tag{45}$$

$$\sum_{i=0}^{3} \lambda_i w_i{=}x,\ \sum_{i=0}^{3} \beta_i w_i{=}x,\ \sum_{i=0}^{3} \theta_i w_i{=}x \tag{46}$$

This results in the following issue: For given $x, y$-coordinates, there are infinitely many solutions for $z$-coordinates admissible under equations (46). For a geometric illustration, Figure 8 shows four basepoints. For a fixed $(x_1, y_1)$, the admissible values for $z$ are visualised by a vertical grey line. This line intersects with the basepoints' convex hull at $(x_1, y_1, z_1)$ and $(x_1, y_1, z_2)$, limiting admissible $z$ to the interval $[z_1, z_2]$.

In general, this formulation is difficult to integrate in MILP because $z$ is not unique. However, for problems minimising $z$, the optimum is $z_1$; maximising $z$ results in $z_2$. For such problems $z$ values are uniquely defined.

The relevant term of our objective function (34) is $\sum_f \sum_{ji} w_{fji}\theta_{ji}$ The term is to be minimised as the objective function is, so that the quadrilateral formulation is viable to apply.

The corresponding problem formulation $\mathsf{qQP}(G,\ p,\ \tilde{\mathsf{N}})$ has objective function and constraints from (34)–(42) and replaced constraints (43) by (47).

$$z_{ji} \leq h_{fji} + h_{f\,j\,i+1} + h_{f\,j+1\,i+1} + h_{f\,j+1\,i} \quad \forall ij \quad \text{(rec-sync)} \tag{47}$$

The advantage of this approach is that it needs fewer decision variables: we need only half the number of quadrilaterals to cover an area than triangles (Figure 6), and each triangle or each quadrilateral needs its own decision variable to control whether its basepoints contribute to the convex combination. Moreover, the constraints in the quadrilateral case are simpler than in the triangle case – compare (43) vs. (47).

**Algorithm 3** SEARCH($G$, $\mathsf{xQP}$, $p$, $\tilde{\mathsf{N}}$) $\rightarrow$ $x$, $y$)
Testing $p' \leq p$ to obtain integer $y$.

```
p' ← p
while p' ≤ p do
    if p' is considered the first time then
        feasible, x, y ← xQP(G, p', N̂)
        if feasible then
            ∀f : y_f ← ⌈y_f⌉,  λ'_f = Σ_c x_cf
            Δ ← Σ_f y_f − p      // valid solution for Δ≤0
            if Δ = 0 then return x, y          // direct hit
            if Δ < 0 then      // add remaining resources
                y ← ALLOC(p, λ', y)
                return x, y
            if Δ > 0 then          // too many resources
                y ← DEALLOC(p, λ', y)
                if DEALLOC(·) was feasible then
                    return x, y
                else
                    decrease p' by Δ
        else                       // infeasible with current p'
            increase p' by 1
    else // p' was considered in previous loop iterations
        increase p' by 1
return SEARCH is infeasible
```

### 4.6 Post-processing Surface's Results

As the final building block, the algorithm SEARCH (Algorithm 3) overcomes the inherent drawback of all surface approximations: Because linear interpolation along $J$ is allowed, allocations obtained by $\mathsf{tQP}_-$, $\mathsf{tQP}_+$, or $\mathsf{qQP}$ are continuous and not necessarily integer values. However, our resources are not splitable, e.g., VM instance size, thus allocations have to be rounded. Rounding down allocations could result in overutilised resources and infeasible solutions. Hence, fractional allocations have to be rounded up, potentially resulting in more allocating resources than allowed, violating $\sum_f y_f{\leq}p$. Hence, rounding up *all* allocations is safe from a utilisation perspective but might incorrectly use too many

resources. Could it be possible to round up only some of these allocations and round down some others? SEARCH tries to answer this question. If this is not possible, another solution is obtained with smaller limit $p'$. This way, the algorithm conceptually tries all $p' \in [p_\downarrow, .., p]$ where $p_\downarrow$ is the smallest number of resources at which the problem is still feasible.

In fact, SEARCH is a bit more complicated. While it would be correct to iterate over the sequence $p, p-1, \ldots, p_\downarrow$, this has unacceptable runtime, in particular owing to the frequent calls to solve xQP. But solutions for any $p'$ and $p'-1$ are most likely to be very similar anyway: If any $p'$ had an inappropriate assignments most likely $p'-1$'s solution is similarly inappropriate; so we could skip $p-1$ to save runtime. Hence, SEARCH modifies $p'$ in larger steps, trying to find a suitable solution more quickly. The intuition for the step size is the amount of number of excess allocations due to rounding.

In detail, SEARCH invokes solving tQP$_-$, tQP$_+$, or qQP. The parameter xQP points to the function solving the concrete problem, e.g. tQP$_-$. Solving xQP results in one of three cases: a) Exactly $p$ resources are allocated; done. b) Fewer resources than requested are allocated ($\Delta<0$), then ALLOC (Algorithm 1) distributes the remaining $|\Delta|$ resources; done. c) More resources than requested are allocated ($\Delta>0$), then DEALLOC removes $\Delta$ resources; DEALLOC is basically a twin of ALLOC removing resources one by one; this paper's extended version provides details on DEALLOC. Often removing all $\Delta$ resources will not be feasible because the current assignment distributes the demand improperly among the facilities. Then, the only adjusting the assignments itself can help. This is done by reinvoking xQP with a lower $p'$. The resulting solution is processed as the original limit $p$ in one of the three cases. The basic idea is to reduce $p'$ and find a good assignment for which the allocation can be adjusted to use $p$ resources. At some point $p'$ is so small, $p_\downarrow \leftarrow p'$, and xQP becomes infeasible. The algorithm finishes in two cases: i) The allocation matches (a) or could be fixed (b,c). ii) After considering all $p' \in [p_\downarrow, .., p]$ without success the algorithm stops and did not find a solution; even if one would exists and, e.g., is found by cQP. Finish case (ii) occurs if xQP with $p_\downarrow+1$ had a solution whose over-allocation could not be fixed by DEALLOC due to the assignments while xQP with $p_\downarrow$ is infeasible. In our evaluations this case occurred more frequently with inputs having a very high system utilisation, $\sum_f \lambda_f / \sum_f k_f \mu_f \geq 0.96$ – in such cases, the minimal necessary allocation matches the limit.

The runtime of SEARCH is determined by xQP's solving time and the number of tested $p'$ values, e.g.., for $p$=100 and $p_\downarrow$=70 the optimisation xQP is done 31 times. The different resource limits $p'$ are tested in order to obtain a suitable assignment for which the allocation could be fixed (c).

To realise this idea, SEARCH examines $p, .., p_\downarrow$ as follows ($p_\downarrow$ is initially unknown): It starts with $p$ and jumps down to $p' \leftarrow p-\Delta$ (Line 17), skipping as many potential resource limits as resources were over-allocated. The idea is that for large over-allocations $p-1$, a larger step is necessary than for slight over-allocations ($\Delta$=1) to change the assignment. Additionally, this stepping depends on the input such as $|F|$, $k$, $p$,... It continues jumping down, $p' \leftarrow p'-\Delta$, until the new $p'=p_\downarrow$ is infeasible and then increases $p' \leftarrow p'+1$. Then, if $p'$ was already considered it is further increased until $p$ is reached and the search terminates without success. The first stage where $p'$ jumps down allows to find quickly a small $p$ values for which the assignment is appropriate. If not, the second stage ensures that all $p' \in [p^i, .., p]$ are tried.

In our evaluations, only for inputs with very high system utilisation testing many $p'$ limits results in long runtimes, but the remaining input had only 3–7 iterations and independent xQP solvings. However, this raises the question if the runtime of these multiple solving times can still compete with runtimes of a single solving time, e.g., cQP? An answer is provided by our evaluation in Section 6.1.

**Algorithm 4** Combines local solutions to find new ones.

```
1:  function GENETIC(G , p)
2:      /* Create a random population of solutions */
3:      P ← ∅, l ← ⌊√|F|⌋
4:      while not enough solutions in P do
5:          F_s ← l random facilities from F
6:          F_s, y_f, t ← DESCENT(G, F_s, p, F, ∅)
7:          if solution not found then increase l
8:          else
9:              if F_s ∉ P then add (F_s, y_f, t) to P
10:     while not enough merge steps are done do
11:         /* merge two solutions */
12:         F_s, F'_s ← two random F_s ∈ P
13:         F_U ← F_s ∪ F'_s; F_I ← F_s ∩ F'_s;
14:         F_M ← three random f ∈ F \ F_U          // Mutation
15:         F_D ← (F_U \ F_I) ∪ F_M
16:         F_N ← F_I with one f ∈ F_D added        // Mutation
17:         F_s, y_f, t ← DESCENT(G, F_N, p, F_D, F_I)
18:         if solution found ∧ F_s ∉ P ∧ t < largest t in P then
19:             replace worst solution in P with current
20:     return F_s, y_f, t from P with smallest t
```

## 5 HEURISTIC

This section discusses an adoption of the most related (Section 2) heuristic proposed by Aboolian et al. [1]. While their work also combines the facility location problem with M/M/k-queuing systems at each facility, their problem QP$_A$ differs from QP in three points: Firstly, QP$_A$ minimises the *maximal* response time whereas QP minimises the *average* response time. Secondly, assignments in QP$_A$ are predefined whereas QP also decides the assignments. Thirdly, QP$_A$ has no resource limit per facility, which QP has. These three differences necessitate adjustments to QP$_A$'s heuristic.

The resulting heuristic H consists of four major parts. ALLOC (Algorithm 1) computes the optimal allocation for a given assignment to a subset of facilities $F_s \subseteq F$. SOLVE (Algorithm 6) first assigns requests to the closest facilities in a given facility subset $F_s$ and computes the corresponding allocation with ALLOC. SOLVE is used by DESCENT (Algorithm 5), which iteratively varies the facility subset to find better subsets. These variations are limited by two additional facility subsets $F^I$ and $F^D$, so that only a local minimum is found. To find new local minima, GENETIC (Algorithm 4) randomly combines already found solutions.

The meta-heuristic GENETIC maintains the finite set of currently best solutions $P$. At first an initial set of solutions is randomly generated (Line 3–Line 9). This is influenced by two factors: The number of maintained solutions, $|P|$, is predefined; maintaining many solutions increases the chance to

**Algorithm 5** Refines solution towards local optimum

```
1: function DESCENT(G, F_s, p, F_D, F_I)
2:     F_s^min, y_f^min, t^min ← F_s, SOLVE(F'_s, p)
3:     while smaller t^min was found do
4:         for F'_s ∈ NEIGH(F_s^min, F_D, F_I) do
5:             y'_f, t' ← SOLVE(G, F'_s, p)
6:             if t' < t^min then
7:                 F_s^min, y_f^min, t^min ← F'_s, y'_f, t'
8:     return F_s^min, y_f^min, t^min
```

**Algorithm 6** Approximate assignment and optimal allocation

```
1: function SOLVE(G, F_s, p)
2:     ∀c, f∈C×F : x_cf ← 0
3:     for c, f∈C×F_s sorted by increasing l_cf do
4:         dem ← λ_c − Σ_f' x_cf'
5:         cap ← k_f μ_f − Σ_c' x_c'f
6:         if dem > 0 ∧ cap > 0 then
7:             x_cf ← min{ dem , cap }
8:     return ALLOC(G, ∀f : Σ_c x_cf, p)
```

find different maxima but also increases runtime. The size of facility subsets $l$ starts from $\sqrt{|F|}$ as in the original heuristic; if no solution is found with these facilities, $I$ is increased. An initial solution is generated by invoking DESCENT (Line 6) with $l$ randomly chosen facilities. DESCENT returns a new subset $F_s$, potentially different from the chosen facilities, the allocation $y_f$, and corresponding average response time $t$ as the measure of solution quality.

In GENETIC's major part (Line 10–Line 19) two random solutions from $P$ are combined to find a new facility subset $F_s$. Three new subsets are computed: The intersection $F_\mathrm{I}$ of both solutions' subsets $F_s, F'_s$ is part of the new offspring facility set $F_\mathrm{N}$. The domain $F_\mathrm{D}$ is the union of $F_s, F'_s$ with three random new facilities $F_\mathrm{M}$; $F_\mathrm{D}$; it limits the following explorations by DESCENT. The offspring $F_\mathrm{N}$ construction is the same as in the original heuristic. A local solution found by DESCENT replaces the worst solution in $P$ if the newly found solution is better than the replaced one. After a finite number of merge operations, the best solution found so far is returned.

DESCENT starts with a given facility subset $F_s$ and searches *neighbouring* subsets for better solutions. Aboolian et al. define $F_s$'s neighbourhood (NEIGH) as a superset consisting of facility subsets constructed by a) removing one facility , b) adding one facility, or c) doing both (48).

$$\begin{aligned}\mathrm{NEIGH}(F_s, F_\mathrm{D}, F_\mathrm{I}) = \{F_s \cup \{f\} \mid \forall f \in F_\mathrm{D} \setminus F_s\} &\quad \text{(a)}\\ \cap \{F_s \setminus \{f\} \mid \forall f \in F_s \setminus F_\mathrm{I}\} &\quad \text{(b)}\\ \cap \left\{F_s \cup \{f\} \setminus \{f'\} \mid \forall f \in F_\mathrm{D} \setminus F_s,\, \forall f' \in F_s \setminus F_\mathrm{I}\right\} &\quad \text{(c)}\\ \text{with } F_\mathrm{I} \subseteq F_s \subseteq F_\mathrm{D} \subseteq F &\quad (48)\end{aligned}$$

If at least one better solution is found in the *subset neighbourhood*, the search continues for the best found solution. As a better solution implies a shorter response time, the algorithm descends towards a local minimum.

The facility sets $F_\mathrm{D}, F_\mathrm{I}$ restrict the cardinality of NEIGH and thus the number of instances solved by SOLVE. This is done by ensuring that $F_\mathrm{I}$ – the intersection of the two parents – is always part of $F_s$ and that $F_\mathrm{D}$ – the modified union of the two parents – is the subset of $F$ to which $F_s$ can grow.

SOLVE (Algorithm 6) computes the assignment and allocation for a given facility subset $F_s$ using $p$ resources. The first part assigns the demand $\lambda_c$ to the closest facilities. This assignment is complexer than $\mathsf{QP}_\mathrm{A}$'s assignments, which neither considers resource capacities ($\mu_f$) nor limits ($k_f$). When a facility's resources are exhausted but demand still exists, it is served by another facility. What was a simple binary assignment to the nearest facility in Aboolian's problem ($\mathsf{QP}_\mathrm{A}$) becomes an optimisation problem to minimise the total assignment distances, $\min_x l_{cf} x_{cf}$ under capacity and serve-all-demand constraints. To solve this problem efficiently, the assignment is computed by the greedy heuristic in Line 2–7: Demand is served by the nearest facility $f$ with remaining capacity. To do so, $c, f$ pairs with the shortest distance are handled first. If demand remains to be distributed, $\lambda_c - \sum_{f'} x_{cf'} > 0$, and facility $f$ has enough capacity left, $\mu_f k_f - \sum_{c'} x_{c'f} > 0$, then as much demand as possible ($\Delta$) is assigned from $c$ to $f$. This way, the demand is shoved to free facilities in a way similar to a multi-source breath-first-search.

While this heuristic is in most cases sufficient, in rare cases swapping the assignments $x_{cf}$ would further improve the solution. This paper's extended version [26] discusses these cases and proposes an extension to SOLVE.

## 6 Evaluation

### 6.1 Approach Comparison

Which presented approaches (optimisation problem with solver or heuristic) solves QP best, where *best* is described by two conflicting metrics: Quality and solving time. We consider here examples in which we vary four factors: Topology $\hat{G}$, demand distribution $\hat{D}$, basepoint set $\hat{B}$, and resource limit $\hat{p}$.

Candidate topologies were collected from different sources: sndlib[7] [33], topology zoo[7] [28], and kingtrace[8] [19]. The topology sources offer 534 candidate topologies from which 8 were selected by three properties: Firstly, the number of nodes increases the problem size quadratically. We selected three small (with at least 20 nodes), three medium, and one[9] large topologies. Secondly, the round trip times contribute to the objective function. The selected topologies have low, medium, or high average round trip times, $\varnothing\mathrm{RTT} = {}^1/_{|E|} \sum_{vv'} l_{vv'}$. Thirdly, resource distributions will be degree-dependent, e.g., few resources on poorly connected nodes and many resources on well connected nodes. Two topologies were explicitly selected having different quartiles of node degrees as the other topologies.

The available resources, in total $\sum_f k_f = 5|N|$, are assigned to nodes weighted by degree. All resources are homogeneous with $\hat{\mu} = 100\,{}^{\mathrm{req.}}/_{\mathrm{s}} = \mu_f,\ \forall f$.

The second factor $\hat{D}$ describes how the individual Poisson processes' arrival rates $\lambda_c$ are assigned to nodes $c$. We

7. Round trip times were approximated by geographical distances [24]. Nodes without geolocation positions were removed and their neighbours were directly connected.

8. A sparse matrix specifies point-to-point latencies. Some were only available in one direction. We assume the same latency for the opposite direction; otherwise those nodes would have to be discarded.

9. Only one of three large topologies were solved within the time limit, see extended version [26].

conceive of $\lambda_c$ as random variables with expected value $\hat{\lambda}$, distributed according to three different distributions: a) $\hat{D}\hat{=}\text{N}(\text{mean}, \text{std.dev.})=\text{N}(\hat{\lambda}, \hat{\lambda}/20)=\text{N}_1$ reflecting small variation around the mean, b) $\hat{D}\hat{=}\text{N}(\hat{\lambda}, \hat{\lambda})=\text{N}_2$ reflecting large variations around the mean, c) $\hat{D}\hat{=}\text{Exp}(\hat{\lambda})$ with even stronger variations to reflect local hot spots. We ignore nodes with negative arrival rates, hence the random variables $\lambda_c$ are defined as follows: $\forall c{:}\lambda_c{=}\max\{0, X\}$ with $X{\sim}\hat{D}$.

The mean arrival rate is $\hat{\lambda}{=}0.98/2\sum_f k_f\mu_f$. This way, on average half of the resources are needed to handle the demand. In such cases, the optimisation potential is large. If system utilisation[10] is large, all resources need to be used anyway and there is no freedom of choice. If it is small, on the other hand, queuing delays become unimportant and the problem basically degenerates into a simple RTT optimization problem. Technically, the reduced factor $0.98$ instead of $1$ reflects the linearisation bound $\alpha_m{=}0.98k$ and ensures the queuing systems' steady state.

The third factor is the resource limit $\hat{p}{=}\sum_f y_f$. It influences the average resource utilisation and, hence, the average queuing delay. The higher the resource limit, the lower the resource utilisation. Since the total number of available resources ($\sum_f k_f$) differs among the topology sizes, the resource limit is cast as a function of this total number of resources, $\hat{p}{=}\lceil a\ \sum_f k_f\rceil$; for values $a{<}0.5$ the input becomes infeasible due to the selection of $\hat{\lambda}$, where half of the resources are needed to handle the demand. For the evaluation, factor $\hat{p}$ uses $a{=}0.5625; 0.625; 0.6875; 0.75; 0.8125; 0.875; 0.9375$; where $a = 0.56$ allows slightly more resources as needed for the demand and $a = 0.94$ allows using nearly all resources.

While the first three factors vary the topology, the final basepoint set factor $\hat{B}$ varies the linearised problems. Each basepoint set is described by two parameters $m$ and $J$ (Section 4.3), where $m$ is the number of basepoints used for a single curve and $J$ specifies which curves $\mathsf{N}_j$, $j{\in}J$ are used. Numerous combinations are possible but considering all of them is impractical. Instead, we performed a pre-evaluation and compared values for $m$ and reasonable sequences for $J$ for their linearisation accuracy; the extended version [26] describes this in detail. Four sets of different sizes with high corresponding accuracy are selected $\hat{B}{=}(m, J){\in}\{(15, 2^i), (8, 3^i), (6, 4^i), (8, k100)\}$, where $J{=}a^i$ refers to the sequence $J{=}[a^1, a^2, ..100]$ and $k100$ refers $J{=}[1, .., 100]$.

Putting all factors together, $168$ factor combinations ($\hat{G}$, $\hat{D}$, $\hat{p}$) are considered and for each one, $50$ random demand realisations are generated. This results in $8400$ different *configurations* to be solved by either the optimization solver or the heuristic. The linearised problems ($\mathsf{cQP}, \mathsf{tQP}_-, \mathsf{tQP}_+, \mathsf{qQP}$) use 3 basepoint sets $\hat{B}{=}\{(15, 2^i), (8, 3^i), (6, 4^i)\}$. In addition, cQP is solved with basepoint set $(8, k100)$ and serves as the baseline comparison; this case considers all curves and, hence, has the highest linearisation accuracy and the best solution quality. The randomised heuristic solves each configuration up to $15$ times to achieve statistically meaningful results. However, for medium or large topologies the heuristic did not compute a solution in reasonable time, because after $6$ hours it still builds up its initial population. As an optimisation solver, Gurobi[11] is used and is configured to stop solving after one hour. Especially for larger topologies, this often causes optimality gaps up to $20\%$.

10. The *system utilisation* is the total arrival rate divided by the total available capacity, $\sum_f \lambda_f/\sum_f \mu_f k_f$. Another metric is the *average resource utilisation*, the total arrival rate divided by the allocated resource capacity, $\sum_f \lambda_f/\sum_f \mu_f y_f{<}\sum_f \lambda_f/p\ \max_f \mu_f$.

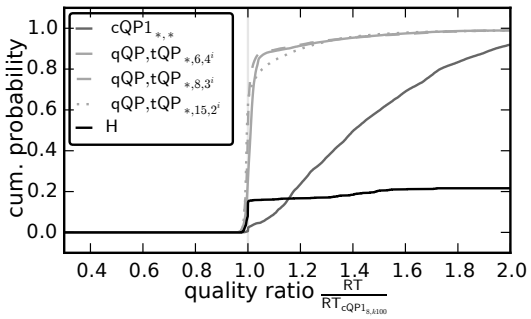


Figure 9: Comparing the quality of baseline approach $\mathsf{cQP}_{8,k100}$ with the other approaches as a CDF plot.

While this paper's extended version [26] compares solution quality (average response time) for single factor combinations, this paper provides aggregated information revealing the core findings.

#### 6.1.1 Baseline Algorithm

The approach $\mathsf{cQP}_{8,k100}$ is the baseline formulation. We compare solution alternatives to the baseline by computing the ratio of the alternative's quality to the baseline's quality individually for each of the $8400$ configurations (ratio $> 1$ means the baseline algorithm performs better). As this produces many individual results, we group similar solution alternatives together (see below for details). The ratios in these groups are then jointly described by empirical cumulative density functions (ECDFs), e.g. Figure 9.

We identify five groups of similar solution alternatives. The first group contains all thinned curves approaches ($\mathsf{cQP}_{6,4^i}$,$\mathsf{cQP}_{8,3^i}$,$\mathsf{cQP}_{15,2^i}$); for this group, $99.4\%$ of all configurations are solved better by the baseline approach. For the remaining configurations, the lowest ratio is $0.97$; the better solutions are caused by ALLOC which uses the exact queuing delay function.

The second, third, and fourth group contain all surface approximations with basepoint sets $\hat{B} = (6, 4^i)$, $(8, 3^i)$, and $(15, 2^i)$. For these groups, the configuration are solved similarly well as with the baseline approach. Surprisingly, $26\%$, $51\%$, and $63\%$ of all configurations result in ratios below $1$, with the lowest ratios being $0.95$, $0.96$, and $0.96$. This is caused by algorithm ALLOC called by the post-processing SEARCH.

The fifth group contains all heuristic solutions, which are very good for small topologies but for the other topologies the quality was magnitudes worse. Section 6.1.4 has more details about the heuristic.

To conclude, solutions obtained by $\mathsf{cQP}_{8,k100}$ *are good references for the expected solution quality* and *solutions obtained by* tQP, qQP *are similarly good*. Better solution qualities than the baseline algorithm's quality involved using ALLOC; the difference to the baseline is always small and results from using the exact instead of the linearised queuing delay. The difference relates to the linearisation accuracy.

11. Gurobi version 5.6.3, Python version 2.7.9

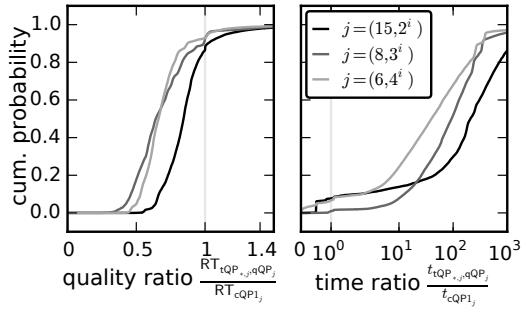


Figure 10: Comparing the quality and solving time ratios of $\mathsf{cQP}_j$ with $\mathsf{tQP}_{*,j}$ and $\mathsf{qQP}_j$.

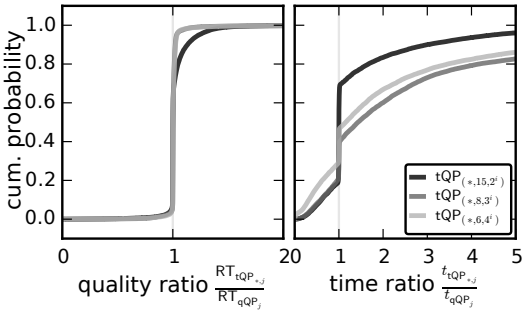


Figure 11: Comparing the quality and solving time ratios of $\mathsf{cQP}_j$ with $\mathsf{tQP}_{*,j}$ and $\mathsf{qQP}_j$.

#### 6.1.2 Thinned Curves vs. Surface Approximations

One of the questions of this paper was to compare univariate vs. bivariate linearisations of the time in system function. To do so, we compare here the quality ratios obtained from either thinned curve linearisation (Section 4.3) vs. surface approximations (Sections 4.4 and 4.5). For the comparison, we fix the basepoint sets $j \in \{(6,4^i), (8,3^i), (15,2^i)\}$. For each basepoint set, we compute the quality ratio by dividing the solution quality of $\mathsf{tQP}_{+,j}$, $\mathsf{tQP}_{-,j}$, or $\mathsf{qQP}_j$ by that of $\mathsf{cQP}_j$. The resulting ratios again give raise to three ECDFs, one per basepoint set. We do the same thing for the solving times, obtaining three more ECDFs.

Figure 10 shows ECDFs of these quality and time ratios for each of the basepoint sets. Most ($92.6\%$, $90\%$, $86.7\%$) solutions quality ratios where below 1. However, solving configurations with surface-base approaches takes magnitudes longer than with $\mathsf{cQP}_j$; $32\%$, $48\%$, and $70\%$ of the surface approaches took $100$ times longer than the thinned curves approach.

The cause of this solving time difference is two-fold: First, $\mathsf{cQP}_j$ is much simpler and has a fast post-processing (ALLOC) while tQP or qQP is invoked multiple times by SEARCH. A better but complexer post-processing adjusting an over-provisioned solution obtained by tQP or qQP could reduce their solving times; this is for further study. Second, Gurobi stopped improving the solution after one hour; so without this limit, the time factor likely increases. To conclude, *among the linearised problems, the thinned curve approach* cQP *showed good quality with the shortest solving time in most of the cases.*

#### 6.1.3 Triangle vs. Qudrilateral Surfaces

Similar to the comparison of curves vs. surfaces, we are interested in characterizing the behaviour of the different surface approaches. To this end, we compute quality and solving time ratios of the triangles divided by the quadrilateral approaches (Figure 11).

For each group, in $65\%$, $72\%$, and $62\%$ of all configurations qQP– the quadrilateral version – is faster than the other formulations. But in $27\%$, $18\%$, and $17\%$ of all configurations, qQP is $10\%$ slower than the other formulations. For each group, $87\%$, $87\%$, and $88\%$ of all configurations are solved equally good or better with qQP than with the other formulations and in the remaining configurations qQP is worse than the other formulations. This confirms the structural arguments for the similarity between these two approaches. The first SEARCH iteration results in different over-utilised solutions' when using qQP or $\mathsf{tQP}_*$. Then, SEARCH continues solving problems with different limit $p$ and that results in different solutions. To conclude, *the problem* qQP *embedded in algorithm* SEARCH *obtains in most cases a very good solution faster than the triangle-base problems.*

In addition, we solved a subset of $200$ configurations directly with $\mathsf{tQP}_*$ and qQP without using SEARCH. While this results in ignoring limit $p$ and in over-utilised allocations, the solutions show the following pattern: (a) The objective value of $\mathsf{tQP}_-$'s solutions were always smaller/better than $\mathsf{tQP}_+$'s objective value, supporting the necessity of considering the triangle orientation; (b) qQP's objective value was always the same as $\mathsf{tQP}_-$'s objective values, which supports the argumentation for quadrilaterals.

#### 6.1.4 Heuristic

The heuristic has a long solving time[12], so a six-hour time limit was imposed; solving times beyond this limit are entirely impracticable for our scenario. Each configuration was solved $15$ times as the heuristic GENETIC itself is randomised. Some of these solving attempts were successful while others failed for the same configuration.

In total, $85.1\%$ of all attempts were not solved in time and most of them do not advance beyond GENETIC's initial phase. The size of the topology corresponds directly to a large neighbourhood visited several times in DECENT (Section 5). This causes the dramatically long solving time. Concluding, *using the genetic heuristic* (in its current version) *is impractical with its significantly worse quality and solving time.*

### 6.2 Scenario Variants

This section investigates how application performance and resource distribution influence the average response times. The same setup as in the previous section was used with different factors.

For the application performance, we consider short, medium, and long request processing times resulting in high, medium, and low service rates; $\hat{\mu}$=$1; 100; 10.000\,^{\text{req.}}/_{\text{s}}$. The same total number of resources $\sum_f k_f$=$5|N|$ were geographically distributed in four different ways (factor $\hat{S}$): (a) $\hat{S}$=d5, $|N|$ available resources are assigned to $5$ nodes

12. The heuristic was implemented in Python and utilised NumPy to speed up computations. In contrast, Gurobi is highly optimised. This biases the solving time comparison a bit, but the results reveal that even a highly optimised heuristic will be outperformed by Gurobi for large topologies.

having the top 5 degrees[13]; (b) $\hat{S}$=d, resources are degree-weighted distributed across all nodes – this was used by the previous evaluation; (c) $\hat{S}$=$\mathrm{d}^2$, the degrees are squared, which amplifies the effect of having more resources at better connected nodes. Finally, as a baseline case, all available resources are placed at a single node having the lowest total latency to all other nodes, $\hat{S}$=c. This baseline case minimises the average queuing delay by forcing all resources to be allocated at one node. Additionally, with $\hat{S}$=$x$, all nodes have 100 available resources eliminating effects of capacity limits ($k_f$)

The previous factor values are now limited to fewer values: The demand distribution follows only two mathematical distributions, $\hat{D}\in\{\mathrm{N}_2, \mathrm{Exp}\}$. The resource limit $\hat{p}$ is restricted to low and high facility utilisation, $\hat{p}$=$\lceil 5a|N| \rceil$, $a$=$0.51; 0.6; 0.75$. The same topologies are considered as in the previous section. The resulting 720 factor combinations each have 50 realisations of demand distribution, resulting in 36,000 configurations. These configurations are solved by qQP with basepoint set $\hat{B}$=$(6, 4^i)$.

At first, the configurations are grouped into combinations of factors $(\hat{\mu}, \hat{p}, \hat{D})$. Within one group the resource distribution factors $\hat{S}$ are compared: For all groups, configurations with 100 resources everywhere ($\hat{S}$=$x$) have, as expected, the shortest round trip times and response times. Configurations with resources at a single site ($\hat{S}$=$c$) have, as expected, the longest round trip times in all groups but only have, surprisingly, in some groups the shortest response time. In the other groups, the resource consolidation at one site causes the queuing delay to drop significantly. So, *using resources at multiple sites does not always decrease response times compared to a single-site deployment*. Which factor $\hat{S}$ results in the second-shortest response time depends on the topology $\hat{G}$ but was independent of $\hat{\mu}, \hat{p}, \hat{D}$. This paper's extended version [26] provides detail plots on the queuing delays and response times for these variants.

How much a distributed deployment reduces the response time compared to a single-site deployment, the quality of different resource distributions $\hat{S}$= d, d2, 5d are compared against $\hat{S}$=c by computing quality factors.For these resource distributions, 78%, 87%, and 97% of the configurations yield better response times than $\hat{S}$=$c$ and 61%, 61%, 35% have half or shorter response times. In conclusion, *deploying application across multiple sites can (at least) halve response times*.

## 7 Conclusion

This paper investigated the problem of allocating resources at multiple sites in order to minimise the user perceived request response time. Such a distributed deployment scheme reduces response time by half or more compared to a single-site deployment (Section 6.2). Five different formulations were presented, trading off quality against solving time. One of these techniques – thinned curves – seems particularly attractive as it vastly outperforms the alternatives at only marginally reduced service response times. This technique, however, is somewhat sensitive to an improper choice of basepoints; the surface techniques are much more robust against a small number of basepoints.

From the considered factors, the topology has – as expected – a strong influence on the optimal solution and its quality. Also, severely limited resources make the problem hard to solve well; the common wisdom of queuing theory to provide ample spare capacity is reinforced here in a distributed setting. Moreover, our results indicate that jointly considering queuing delay and RTT is indeed crucial when these two times are roughly of the same order of magnitude – otherwise, when one of the two times dominates, simpler optimisation models suffice to obtain good solutions. In fact, we found scenarios where single-site deployments outperformed distributed deployments.

The presented formulation techniques are not limited to the paper's problem. Beyond this paper, any optimisation problem having a univariate or bivariate, non-linear cost function can be linearised by the presented approaches. We extended known approaches to mixed-integer objective functions which have not been treated in the literature so far.

Our formulations are not restricted to convex/concave cost functions. In particular, a newly presented surface linearisation based on quadrilaterals instead of commonly used triangles turned out to be a promising alternative that is sometimes faster than the triangle-based formulations and has the same solution quality (Section 6.1.2).

We introduced three greedy algorithms MaxWeight, Alloc, Dealloc for allocation problems where $n$ tokens are placed in $m$ buckets so that the costs are minimised. If the cost functions $c(x)$ for the number of buckets $c$ is decreasing and convex, these algorithms are optimal.

In summary, this paper not only improves service response times by optimally allocating resources but also presents optimisation techniques and greedy algorithms applicable beyond this scenario.

## Acknowledgment

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

## References

[1] R. Aboolian, O. Berman, and Z. Drezner. The multiple server center location problem. *Annals of Operations Research*, 167:337–352, 3 2009.

[2] S. Agarwal, J. Dunagan, and N. Jain. Volley: Automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10)*, 2010.

[3] M. Bagaa, T. Taleb, and A. Ksentini. Service-aware network function placement for efficient traffic handling in carrier cloud. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2402–2407. IEEE, 4 2014.

[4] S. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *1st annual conference on Multimedia systems (MMSys)*, page 35. ACM Press, 2010.

[5] M. Bauer, S. Braun, and P. P. Domschitz. Media processing in the future internet. In *Proceedings of the 11th Würzburg Workshop on IP: Visions of Future Generation Networks*, pages 113–115, 2011.

[6] O. Berman and Z. Drezner. The multiple server location problem. *Journal of the Operational Research Society*, 58:91–99, 1 2006.

[7] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi. *Queueing networks and Markov chains*. Wiley-Interscience, 2005.

[8] S. Borst, A. Mandelbaum, and M. I. Reiman. Dimensioning large call centers. *Operations Research*, 52:17–34, 1 2004.

13. For all degree-based selections, the node ID was the tiebreaker.

[9] D. Cai and S. Natarajan. The evolution of the carrier cloud networking. In *Seventh International Symposium on Service-Oriented System Engineering*, pages 286–291. IEEE, 3 2013.

[10] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM - Entertainment networking*, 49:40–45, 2006.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 9 2009.

[12] G. B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28:30–44, 1960.

[13] T. Drezner and Z. Drezner. The gravity multiple server location problem. *Computers & Operations Research*, 38:694–701, 3 2011.

[14] Z. Drezner and H. W. Hamacher. *Facility location: applications and theory*. Springer, 2004.

[15] C. D'Ambrosio, A. Lodi, and S. Martello. Piecewise linear approximation of functions of two variables in MILP models. *Operations Research Letters*, 38:39–46, 1 2010.

[16] P. Endo, A. de Almeida Palhares, N. Pereira, G. Goncalves, D. Sadok, J. Kelner, B. Melander, and J.-E. Mangs. Resource allocation for distributed cloud: concepts and research challenges. *IEEE Network*, 25:42–46, 2011.

[17] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, PP:1–19, 2013.

[18] B. Geißler, A. Martin, A. Morsi, and L. Schewe. *Using Piecewise Linear Functions for Solving MINLPs*, pages 287–314. Springer New York, 2013.

[19] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King : Estimating latency between arbitrary internet end hosts. In *IMW '02 Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages p. 5–18, 2002.

[20] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song. Enhancing dynamic cloud-based services using network virtualization. *1st ACM workshop on Virtualized infrastructure systems and architectures (VISA)*, 40:37, 2009.

[21] A. Imamoto and B. Tang. Optimal piecewise linear approximation of convex functions. In *World Congress on Engineering*, pages 22–25, 2008.

[22] A. Imamoto and B. Tang. A recursive descent algorithm for finding the optimal minimax piecewise linear approximation of convex functions. In *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 287–293. IEEE, 10 2008.

[23] A. Ishii and T. Suzumura. Elastic stream computing with clouds. In *4th International Conference on Cloud Computing*, pages 195–202. IEEE, 7 2011.

[24] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, and R. Steinmetz. Modelling the internet delay space based on geographical locations. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 301—-310. IEEE, 2009.

[25] M. Keller and H. Karl. Response time-optimized distributed cloud resource allocation. In *Workshop on Distributed Cloud Computing (DCC)*, 2014.

[26] M. Keller and H. Karl. Response-Time-Optimised Service Deployment: MILP Formulations of Piece-wise Linear Functions Approximating Non-linear Bivariate Mixed-integer Functions. report, http://arxiv.org/abs/1507.08834, 2015.

[27] M. Keller and H. Karl. Response time-optimized distributed cloud resource allocation (preprint). *tba*, tba:tba, 2015.

[28] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29:1765–1775, 10 2011.

[29] Y. Lee, K. Chen, H. Su, and C. Lei. Are all games equally cloud-gaming-friendly? an electromyographic approach. In *Proceedings of IEEE/ACM NetGames 2012*, pages 4–9, 2012.

[30] V. Marianov and D. Serra. Probabilistic maximal covering location-allocation models with constrained waiting time or queue length for congested systems. *Journal of Regional Science*, 38(3):401–424, 1996.

[31] V. Marianov and D. Serra. Location – allocation of multiple-server service centers. *Annals of Operations Research*, 111:35–50, 2002.

[32] M. Moghadas and T. Kakhki. Maximal covering location-allocation problem with m/m/k queuing system and side constraints. *Iranian Journal of Operations Research*, 2:1–16, 2011.

[33] S. Orlowski, M. Pioro, A. Tomaszewski, and R. Wessaly. Sndlib survivable network design library. report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2007.

[34] S. Pandey, A. Barker, K. K. Gupta, and R. Buyya. Minimizing execution costs when using globally distributed cloud services. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 222–229. Ieee, 2010.

[35] S. H. R. Pasandideh and S. T. A. Niaki. Genetic application in a facility location problem with random demand within queuing framework. *Journal of Intelligent Manufacturing*, 23:651–659, 5 2010.

[36] S. Rebennack and J. Kallrath. Continuous piecewise linear delta-approximations for bivariate and multivariate functions. *Journal of Optimization Theory and Applications*, 163:1–16, 2014.

[37] M. Scharf, T. Voith, W. Roome, B. Gaglianello, M. Steiner, V. Hilt, and V. K. Gurbani. Monitoring and abstraction for networked clouds. In *16th International Conference on Intelligence in Next Generation Networks*, pages 80–85. IEEE, 10 2012.

[38] A. L. Stolyar and B. Labs. Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud. In *INFOCOM, 2013. 32nd IEEE International Conference on Computer Communications*, pages 644–652, 2013.

[39] T. Taleb. Toward carrier cloud: Potential, challenges, and solutions. *IEEE Wireless Communications*, 21:80–91, 6 2014.

[40] N. Vidyarthi, S. Elhedhli, and E. Jewkes. Response time reduction in make-to-order and assemble-to-order supply chain design. *IIE Transactions*, 41:448–466, 3 2009.

[41] Z. Wan. Cloud computing infrastructure for latency sensitive applications. In *12th International Conference on Communication Technology*, pages 1399–1402. IEEE, 11 2010.

[42] Q. Wang, R. Batta, and C. M. Rump. Algorithms for a facility location problem with stochastic customer demand and immobile servers. *Annals of Operations Research*, 111:17–34, 3 2002.

[43] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba. Dynamic service placement in geographically distributed clouds. In *2nd International Conference on Distributed Computing Systems*, pages 526–535. IEEE, 6 2012.


**Matthias Keller** received his diploma degree in computer science from the University of Paderborn, Germany. He is currently working as a research associate in the Computer Networks group in the University of Paderborn. He focuses on adaptive resource allocation across wide area networks, designed a framework, and created a prototype testbed. Previously, he worked at the Paderborn Centre for Parallel Computing as a research associate and as a software engineer in the industry.

**Holger Karl** received his PhD in 1999 from the Humboldt University Berlin; afterwards he joined the Technical University Berlin. Since 2004, he is Professor for Computer Networks at the University of Paderborn. He is also responsible for the Paderborn Centre for Parallel Computing and has been involved in various European and national research projects. His main research interests are wireless communication and architectures for the Future Internet.