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; virtual network function; network function virtualisation; resource management; placement; facility location; queueing model; linearisation; optimization; mixed integer function; derivative; erlang delay formula;

1 Introduction

1.1 Opportunities and Problem Statement

Providing resources at widely dispersed places is a new trend in networking and cloud computing known under different labels: for example, Carrier Clouds [4], [11], [46], Distributed Cloud Computing [2], [19], [40], or In-Network Clouds [23], [44], [45]. These In-Network resources are often geared towards specific network services, e.g., firewalls or load balancers – a development popularised as Network Function Virtualisation [20]. 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 [5], [50], user-customised streaming [6], [28], or cloud gaming [35]. For such applications, the crucial quality metric is the user-perceived *response time* to a request. Large response times impede usability, increase user frustration [12], or prevent commercial success.

As detailed in a prior publication [32], 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 [32], 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*.

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.

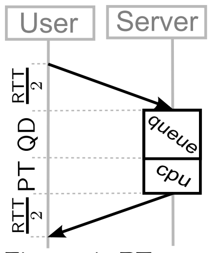


Figure 1: RT = RTT + QD + PT.

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 [9]. Hence, the queuing delay and round trip time, are minimised by two conflicting allocation schemes, nearby and consolidated. To find an optimal solution *for the sum* of QD and RTT, the 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*

allocation schemes have to be balanced to find assignments and allocations that minimises the response time.

Similarly, the assignment should prefer faster resources, even if their round trip time is a *bit* longer: Using faster resources reduces the processing time. This time reduction can compensate for the longer round trip time, in total reducing the response time.

1.2 Queuing Delay Effects

Consider graph G below Table 1 as an example: Two clients c_a, c_b are connected to three facilities $f_{1..3}$ having 10 resources available, $k_{1..3}{=}10$, such as very large data centres. All resources within one facility are homogeneous but they differ among the facilities, having service rates $\mu_{1..3}{=}60; 120; 60\,{}^{\mathrm{req.}}/_{\mathrm{s}}$ for facility $1; 2; 3$, respectively. The assignable requests are limited to 98% of the service rate to avoid very high queuing delays due to its asymptotic behaviour. This way, the queuing systems are also in steady state. The number of resources to use are limited to $p{=}5$. Round trip times between clients and facilities are $l_{a,1..3}{=}l_{b,3..1}{=}40; 70; 100\,{}^{\mathrm{req.}}/_{\mathrm{s}}$. Client c_a's arrival rate is slightly larger than c_b's arrival rate; this slight distortion leads to unique solutions. Each row in Table 1 corresponds to one level of arrival rates. In the columns, two solutions are compared which are obtained by either ignoring (P) or considering (QP) the queuing delays when deciding allocation and assignment. To compare them directly, for both solutions the following measurements were listed: The average response times ∅RT computed with queuing delays in both cases and the times in queuing systems at each facility. When comparing the average response times of both solutions in the middle of the table; stars mark the superior solution. All QP's solutions are at least equal to *optP*'s solutions; most are superior.

This superiority has two causes: When allocating resources, problem P only minimises the round trip time and, thus, assign as many requests as possible nearby. Consequently, if all resources are used at the nearest facility, additional resources are allocated somewhere. In contrast, in our example QP allocates additional resources where the queuing delay is reduced the most; cf. $t_{1..3}$ for $\lambda_a{\le}60$. Similarly, the small resource capacity μ_1 forces P to assign more assignments to f_2 when increasing $\lambda_a{\ge}200$, $\mu_1{<}\mu_2$. In contrast, QP assignments are shifted at lower arrival rate to f_2 *as the queuing time reduction compensates for the higher round trip time*. Even with the same allocations $y_{1..3}$ in $\lambda_a{=}120$, QP balances the assignments better than P which results in shorter queueing delays $t_{1..3}$.

Table 1 supports another observation: The response time differences between P and QP are small at low or very high system utilisation; *system utilisation* is the fraction of the total arrival rate to the total capacity, $\sum_f \lambda_f / \sum_f k_f \mu_f$. A high system utilisation enforces one solution where all resources are fully utilised (1). At low system utilisation, the gain of additionally considering the queuing delay is small, $\lambda_a{\le}100$. However, the tide is turning for medium utilisation, where QP's response times can be significantly smaller than P's response time, e.g., for $\lambda_a{=}180$ the ∅RT$_{\mathrm{QP}}$ is 26.41% of ∅RT$_{\mathrm{P}}$.

$$\underbrace{0.98\,p\,\mu_1{=}294}_{\text{cap } f_1} < \underbrace{550}_{\lambda_a+\lambda_b} < \underbrace{588{=}0.98\,p\,\mu_2}_{\text{cap } f_2} \qquad (1)$$

Assignment and allocation are mutually dependent; making a simple separation into subproblems and solving them sequential is suboptimal. When assigning demand first, one facility f could exist whose amount of assigned requests requires allocating more resources as globally allowed (limit p); as a simple example imagine 10 customers are assigned to 10 nearby facilities but only $p{=}8$ resources are allowed to be used. They only way to fix this problem is to adjust the assignment to the need of the allocation. Allocating resources first without assigning requests beforehand is always feasible. But without knowing and assigning the user requests at the same time, the round trip times between user and facilities with resources are most likely longer than otherwise. Either way, the solutions are better when assignment and allocation are obtained together. This in turn significantly increases the complexity of solving the problem to optimality.

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, three aspects are evaluated: First, different basepoint sets for the linear approximations are discussed to reduce approximation error and the number of basepoints – which are two conflicting goals (Section 6.1). Second, the five approximations and the heuristic are compared for two metrics, accuracy and solving time (Section 6.2). Third, an additional evaluation discusses how resource distribution in the network reduces response times (Section 6.3). For the last two evaluations, 111,600 problems are solved.

This paper extends our previous work [30] 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

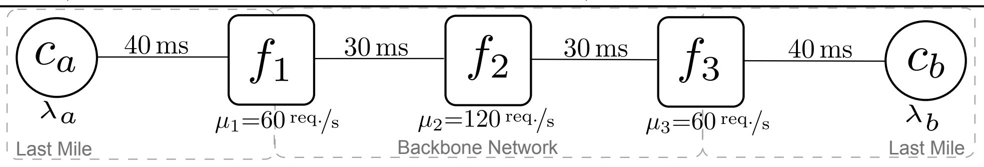
Similar assignment and allocation problems with integrated queuing systems have been investigated before. While this related work focuses on a broader overview, we point interested readers to our detailed comparison of structure and simplifying assumptions of these problems [32].

2.1 Assignment and Allocation with Queues

Queuing systems have been integrated in FLP problems before [1], [7], [15], [36], [37], [38], [41], [49], [51] 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 [32], 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

Table 1: Compares response times considering (QP) and ignoring (P) QDs for different arrival rates.

arrival rate [req./s]		# resources			time in system [ms]			∅RT$_{\text{QP}}$	∅RT$_{\text{P}}$	# resources			time in system [ms]		
λ_a	λ_b	f_1	f_2	f_3	f_1	f_2	f_3	[ms]	[ms]	f_1	f_2	f_3	f_1	f_2	f_3
20	10	3	0	2	11.1	0.0	5.6	56.7 *	62.2	1	1	3	16.7	0.0	5.6
40	30	3	0	2	9.7	0.0	7.6	57.3 *	58.3	2	1	2	10.7	0.0	7.6
60	50	3	0	2	9.5	0.0	9.2	58.7 *	61.3	2	1	2	12.1	0.0	9.2
80	70	3	0	2	9.9	0.0	11.8	61.6 *	64.3	2	0	3	16.0	0.0	8.3
100	90	3	0	2	10.7	0.0	18.0	68.8	68.8	3	0	2	10.7	0.0	18.0
120	110	3	0	2	15.8	0.0	22.0	79.3 *	102.5	3	0	2	12.6	0.0	49.9
140	130	5	0	0	42.1	0.0	0.0	96.5 *	248.9	3	0	2	24.2	0.0	183.3
160	150	0	2	3	0.0	7.9	18.4	97.6 *	379.1	2	1	2	159.7	5.3	159.7
180	170	3	2	0	18.9	14.5	0.0	107.2 *	405.9	3	1	1	143.1	63.1	140.0
200	190	0	3	2	0.0	13.4	13.0	111.3 *	223.3	3	2	0	128.5	22.0	0.0
220	210	0	4	1	0.0	12.6	10.6	116.3 *	216.4	2	3	0	115.1	17.7	0.0
240	230	0	5	0	0.0	12.4	0.0	112.4 *	291.3	2	3	0	105.3	101.0	0.0
260	250	0	5	0	0.0	15.6	0.0	115.6 *	223.3	1	4	0	96.1	34.1	0.0
280	270	0	5	0	0.0	24.3	0.0	124.3	124.3	0	5	0	0.0	24.3	0.0



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 [49] but has an unknown optimality gap.

The mentioned papers solve their problems via greedy heuristics [1], [7], [15], [41], [51] 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], [7], [15], [51], [53] or by replacing the non-linear queuing delay part by a constant upper bound [51], [53]. 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 [18], [21]. 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 [26], [27] (improved by us [32]) 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. [43] 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. 2), which is not separable.

Vidyarthi et al. [49] 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 [16]. 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. 5) 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 λ_c for each client c. Each facility f has k_f resources available and each resource can process requests at service rate μ_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 $\forall c{:} l_{cf_1} = l_{cf2}$; this introduces inaccuracy by modelling two queues where the precise model would had one. Table 2 lists all variables.

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

$$\min_{x,y} \underbrace{\frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c}}_{\text{avg. RTT}} + \underbrace{\frac{\sum_f (\sum_c x_{cf})\, \mathsf{T}_{\mu_f}(\sum_c x_{cf}, y_f)}{\sum_c \lambda_c}}_{\text{avg. time in system}} \quad (5)$$

$$\text{s.t.} \sum_f x_{cf} = \lambda_c \qquad \forall c \quad \text{(demand)} \quad (6)$$

$$\left(\sum_c x_{cf}, y_f\right) \in \operatorname{dom}(T_\mu) \quad \forall f \quad \text{(capacity)} \quad (7)$$

$$y_f \le k_f \qquad \forall f \quad \text{(count)} \quad (8)$$

$$\sum_f y_f = p \qquad \text{(limit)} \quad (9)$$

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 (2) [8]. 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 $\mathsf{N}(a,\ k)$ (3) with shorthand $a{=}\lambda/\mu$. Similarly, the expected *time a request spends in the system* (TiS) is $\mathsf{T}_\mu(\lambda,\ k)$ (4).

$$\mathrm{EC}(a,\ k) = \frac{\frac{ka^k}{k!\,(k-a)}}{\frac{ka^k}{k!\,(k-a)} + \sum_{i=0}^{k-1} \frac{a^i}{i!}} \qquad \text{(Erlang-C)} \quad (2)$$

$$\mathrm{N}(a,\ k) = \frac{a}{(k-a)}\,\mathrm{EC}(a,\ k) + a \qquad \text{(NiS)} \quad (3)$$

$$\mathrm{T}_\mu(\lambda,\ k) = \frac{1}{\lambda}\mathrm{N}(\lambda/\mu,\ k) = \frac{\mathrm{EC}\,(\lambda/\mu,\ k)}{k\mu - \lambda} + \frac{1}{\mu} \qquad \text{(TiS)} \quad (4)$$

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 2. Both are chosen to minimise the average response time. The queuing delay non-linearly depends on both the assigned request and allocated resources – a well-known fact later recapitulated in Theorem 5. The formulation QP (Problem 1) extends the p-median facility location problem P [32] by having additional costs, the queueing delays, at each facility. Constraint (6) ensures that all demand is assigned. The assignments at each facility f, $\sum_c x_{cf}$, must not exceed f's processing capacity $y_f\mu_f$ (7), which is the processing capacity per resource μ_f times the number of allocated resources y_f. The allocated resources y_f do not exceed the local (8) and global (9) limit.

4 Linearisation

The problem QP is non-linear and could be solved by a non-linear solver. In previous work [32], a simpler problem

Table 2: 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 $(x_{c1}, .., x_{c|F|})$.

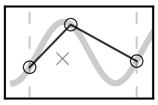


Figure 2: 1D-PWL example.

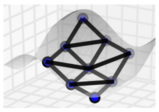


Figure 3: 2D-PWL example.

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 T_μ 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).

Then, Section 4.7 further discusses potential problem simplifications. Finally, Section 4.8 presents technical details on how Erlang-C-base functions are efficiently implemented.

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 [21]; 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 (10) by linearly interpolating between two adjacent basepoints (black circles). The interval is implicitly defined by the outer basepoints, $[\alpha_0, \alpha_{m-1}]$.

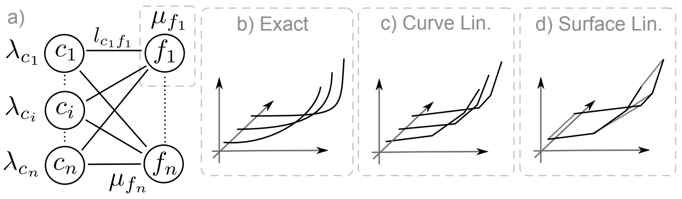


Figure 4: Bipartite graph of a facility location problem with queuing systems at each facility (a) and zoomed bivariate time in system version: exact (b), curve-based (c) and surface-based (d).

$$\tilde{g}(x) = \begin{cases} (\alpha_1{-}\alpha_0)(x{-}\alpha_0) + \beta_0 & \text{if } \alpha_0{\le}x{<}\alpha_1 \\ (\alpha_2{-}\alpha_1)(x{-}\alpha_1) + \beta_1 & \text{if } \alpha_1{\le}x{<}\alpha_2 \\ \dots \end{cases} \tag{10}$$

The approximation error ϵ 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 (10) is not directly understood by an MILP solver it has to be transformed. We use the convex-combination method [14] to model continuous, univariate PWL functions. The cases of (10) are substituted by a convex combination of the basepoint coordinates (11) with weights z_i.

$$\tilde{y}{=}\tilde{g}(x) \Leftrightarrow \sum_i z_i\alpha_i{=}x,\ \sum_i z_i\beta_i{=}\tilde{y},\ \sum_i z_i{=}1,\ \mathsf{SOS2}(z_*) \tag{11}$$

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 (12) and weights for only three vertices of a single triangle are non-zero instead of the two segment vertices in the univariate version [21].

$$\begin{aligned}\tilde{w}{=}\tilde{g}(x,y)\ \Leftrightarrow &\sum_i z_i\alpha_i{=}x,\ \sum_i z_i\beta_i{=}y, \\ &\sum_i z_i\theta_i{=}\tilde{w},\ \sum_i z_i{=}1,\ \mathsf{SOS2}(z_*)\end{aligned} \tag{12}$$

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 $\mathsf{T}_\mu(\lambda, k)$ (4) of the objective function (5). However, linearising T_μ is not obvious as the second parameter is integer. So, T_μ 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 T_μ (4).

Each function $\mathsf{T}_{\mu,j}$ is convex, so we can use our algorithm [32] 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 ith 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 jth curve, the linearisation of $\tilde{\mathsf{T}}_{\mu,j}(\lambda)$ is then formulated as $\sum_i z_{fi}\beta_{fji}$ with λ as $\sum_i z_{fi}\alpha_{fji}$.

In the resulting problem formulation cQP3[6] (Problem 2), QP's objective function (5) is transformed as described above, replacing (5) by (13) and by (19). : The new constraint (16) ensures a convex combination only of neighbouring weights. The capacity constraint (15) replaces (7). The local and global limits are adjusted using $\dot{y}$; constraints (8), (9) are replaced by (17), (18).

The right term of the objective function (13) 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 (13) is integrated in the basepoint coordinates. To understand this modification, QP's objective function (5) has to be revisited. In this function, the time-in-system function T_μ 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 (3). The problem QP can be reformulated by replacing constraints (5), (7) with (20), (21).

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 2).

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 2)

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

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\sum_f (\sum_c x_{cf}) \overbrace{\sum_j \dot{y}_{fj} \sum_i z_{fi} \beta_{fji}}^{\text{time in system } f}}{\sum_c \lambda_c} \tag{13}$$

$$\text{s.t.} \quad \sum_f x_{cf} = \lambda_c \qquad \forall c \qquad \text{(demand)} \tag{14}$$

$$\sum_c x_{cf} \leq \sum_j \dot{y}_{fj} \sum_i z_{fi} \alpha_{fji} \qquad \forall f \qquad \text{(capacity)} \tag{15}$$

$$\sum_i z_{fi} = 1, \ \mathsf{SOS2}(z_{f*}) \qquad \forall f \qquad \text{(weights)} \tag{16}$$

$$\sum_j j \, \dot{y}_{fj} \leq k_f \qquad \forall f \qquad \text{(count)} \tag{17}$$

$$\sum_{fj} j \, \dot{y}_{fj} = p \qquad \text{(limit)} \tag{18}$$

$$\mathsf{SOS1}(\dot{y}_{f*}) \qquad \forall f \qquad \text{(curve flip)} \tag{19}$$

$$\min_{x,y} \sum_{cf} x_{cf} l_{cf} + \sum_f \mathsf{N}(\frac{\sum_c x_{cf}}{\mu_f}, \, y_f) \tag{20}$$

$$\text{s.t.} \quad \left(\frac{\sum_c x_{cf}}{\mu_f}, y_f \right) \in \operatorname{dom}(\mathsf{N}) \quad \forall f \quad \text{(capacity)} \tag{21}$$

Then, the adjusted QP is linearised as before: Function 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 (22) to (23).

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

$$\tilde{\mathsf{N}}_k(a = {}^w/_\mu) \Leftrightarrow \mu \sum_i \alpha_i z_{fi} = w, \sum_i z_{fi} \beta_i = \mathsf{N}_k(a) \tag{23}$$

The resulting linearisation cQP2 (now quadratic, hence a "2") results from replacing constraints (13),(15) with (24), (25):

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\sum_{fj} \dot{y}_{fj} \sum_i z_{fi} \beta_{ji}}{\sum_c \lambda_c} \tag{24}$$

$$\sum_c x_{cf} \leq \mu_f \sum_j \dot{y}_{fj} \sum_i \alpha_{ji} z_{fi} \, \forall f \quad \text{(capacity)} \tag{25}$$

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 μ; in total $m \max_f k_f$ basepoints are necessary.

The objective function of cQP2 (24) 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

$$\min_{x,y,z} \frac{\sum_{cf} x_{cf} l_{cf}}{\sum_c \lambda_c} + \frac{\overbrace{\sum_{fji} z_{fji} \beta_{ji}}^{\text{total time in systems}}}{\sum_c \lambda_c} \tag{26}$$

$$\text{s.t.} \quad \sum_f x_{cf} = \lambda_c \qquad \forall c \qquad \text{(demand)} \tag{27}$$

$$\sum_c x_{cf} \leq \mu_f \sum_{ji} z_{fji} \alpha_{ji} \qquad \forall f \qquad \text{(capacity)} \tag{28}$$

$$\sum_i z_{fji} = 1, \ \mathsf{SOS2}(z_{fj*}) \, \forall f, j \qquad \text{(weights)} \tag{29}$$

$$\sum_i z_{fji} = \dot{y}_{fj} \qquad \forall f, j \qquad \text{(sync)} \tag{30}$$

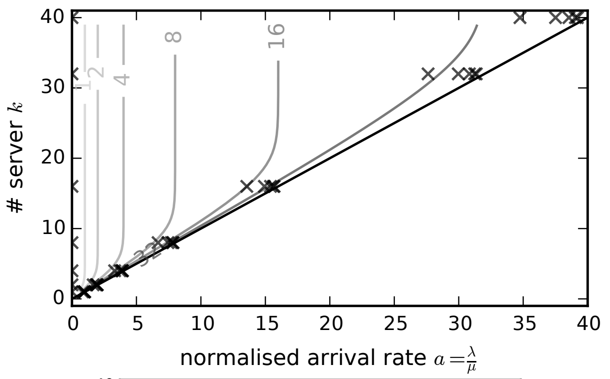
Constraints (17), (18), (19)

The resulting linear formulation cQP1 (Problem 3) has a new constraint (30), ensuring that only the relevant weights are allowed to be non-zero. Having this constraint in place, the new objective function (26) equals the old objective function (24). Constraints (28), (29) 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 (17), (18) by (31), (32). 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 (32) is relaxed to



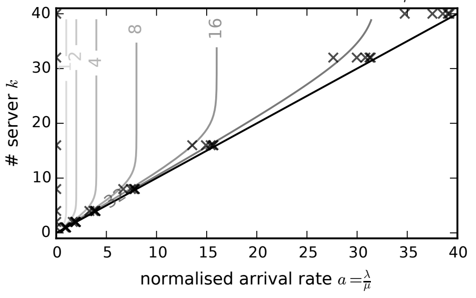


Figure 5: Contour plot for $\mathsf{N}_j(a)$ with basepoints for $j{\in}J{=}[1,2,4,8,16,32,40]$ of $\tilde{\mathsf{N}}_J(a,k)$

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

Obj. func./Constr. (26),(19),(27),(28),(29),(30)

$$\text{s.t.}\quad \sum_{j\in J} j\ \dot{y}_{fj} \le k_f \quad \forall f \qquad \text{(count)} \qquad (31)$$

$$\sum_{f,\,j\in J} j\ \dot{y}_{fj} \le p \qquad \text{(limit)} \qquad (32)$$

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 (32), 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 (32) 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.

Algorithm 1 ALLOC$(p,\, \lambda',\, y{=}0,\, F'{=}F) \;\rightarrow\; y, T$
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:     $x \leftarrow$ MAXCOSTDROP$(n,\, c(x),\, x^{\max})$
               $\forall f : c_f(x) := \mathsf{N}(a_f, y_f{+}x)$
               $\forall f : x_f^{\max} \leftarrow k_f{-}y_f^{\min}$
5:     $\forall f : y_f = y_f^{\min} + x_f$
6: else
7:     $\forall f : y_f = y_f^{\min}$
8: return $y,\, \sum_f \mathsf{N}(a_f, y_f)$
```

Algorithm 2 MAXCOSTDROP$(n,\, c_*(y'),\, y^{\max}) \;\rightarrow\; y$
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: $S \leftarrow \{y_{bi} \mid 1{\le}b{\le}m, 1{\le}i{\le}y_b^{\max}\}$
       $I \leftarrow \{S'{\subseteq}S \mid n{\ge}|S'|\}$
       $w(y_{bi}) := c_b(i) - c_b(i{-}1)$ if $y{>}1$ else $c_b(0)$
2: $A \leftarrow \emptyset$
3: for $y \in S$, sorted by non-increasing weights do
4:     if $|A| < n$ then $|A| \leftarrow A \cup \{y\}$
5: $\forall f{:}x_f \leftarrow |\{y_{bi}{\in}A \mid 1{\le}i{\le}y_b^{\max}\}|$
6: return $x_*$
```

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 $c_f(x)$.

Proof. For a weighted matroid $M{=}(S,I)$, algorithm GREEDY ([13], 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, λ', 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 λ'_f having $n{\geq}0$ resources still to be allocated ($n{<}0$ contradicts λ'_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 MAX-COSTDROP is used, where dropping tokens into buckets correspond to increasing resource at facilities. The token cost 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 $1{\leq}k{\leq}40$ 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, $g(\alpha_{ji},\beta_{ji}){=}\theta_{ji}$, $0{\leq}j{<}n$, $0{\leq}i{<}m$. Using the convex combination MILP formulation (33), each basepoint has a weight z_{ji} as the decision variable. Only the three edges of exactly one triangle form the convex combination (SOS3) [18], [21]. Since current solvers do not support SOS3 constraints, D'Ambrosio et al. [18] presented an equivalent formulation (34) 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 (34) corresponds to triangle enumeration and orientation in Figure 6a. Integrating the triangle formulation from (33), (34) in cQP results in tQP$_{-}$ (Problem 5).

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

$$\min_{\substack{x,y,\\z,h}} \frac{1}{\sum_c \lambda_c}\sum_{cf} x_{cf} l_{cf} + \underbrace{\frac{1}{\sum_c \lambda_c}\sum_{fji} z_{fji}\theta_{ji}}_{\text{sum of all TiS}} \tag{35}$$

$$\text{s.t. } \sum_f x_{cf} = \lambda_c \qquad \forall c \quad \text{(demand)} \tag{36}$$

$$\sum_c x_{cf} {\leq} \underbrace{\mu_f \sum_{ji} z_{fji}\alpha_{ji}}_{\text{capacity at } f} \qquad \forall f \quad \text{(capacity)} \tag{37}$$

$$\underbrace{\sum_{ji} z_{fji}\beta_{ji}}_{\#\text{server used at } f} \ \leq k_f \qquad \forall f \quad \text{(count)} \tag{38}$$

$$\sum_{fji} z_{fji}\beta_{ji} = p \qquad \text{(limit)} \tag{39}$$

$$\sum_{ji} z_{fji} = y_f \qquad \forall f \quad \text{(open)} \tag{40}$$

$$\sum_{j'i'} h^*_{fj'i'} = y_f \qquad \forall f \quad \text{(o-sync)} \tag{41}$$

$$\sum_{j'i'} h^u_{fj'i'} + h^l_{fj'i'} = 1, \quad \mathsf{SOS1}(h^*_{f**}) \qquad \forall f \quad \text{(single-tri)} \tag{42}$$

$$h^*_{f0*}{=}h^*_{f*0}{=}h^*_{fm*}{=}h^*_{f*n}{=}0 \qquad \forall f \quad \text{(tri-corner)} \tag{43}$$

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

with $0{\leq}j'{\leq}n$, $0{\leq}i'{\leq}m$

$$\sum_{ji}\alpha_{ji}z_{ji}{=}x, \sum_{ji}\beta_{ji}z_{ji}{=}y, \sum_{ji}\theta_{ji}z_{ji}{=}z = g(x,y),$$

$$\mathsf{SOS3}(z_{**}),\ \sum_{ij} z_{ij} = 1 \tag{33}$$

$$\sum_{ji} h^u_{ji}{+}h^l_{ji}{=}1,\ \mathsf{SOS1}(h^*_{**}),\ h^*_{0*}{=}h^*_{*0}{=}h^*_{m*}{=}h^*_{*n}{=}0$$

$$\forall ji : z_{ji} \leq h^u_{ji}{+}h^l_{ji}{+}h^u_{\substack{i+1\\j}}{+}h^l_{\substack{i+1\\j+1}}{+}h^u_{\substack{i+1\\j+1}}{+}h^l_{\substack{i\\j+1}} \tag{34}$$

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 (35)–(43) and replaces constraint (44) by (45).

$$z_{ji} \leq h^u_{fji} + h^l_{f\,j\,i+1} + h^u_{f\,j\,i+1} + h^l_{f\,j+1\,i+1} +$$

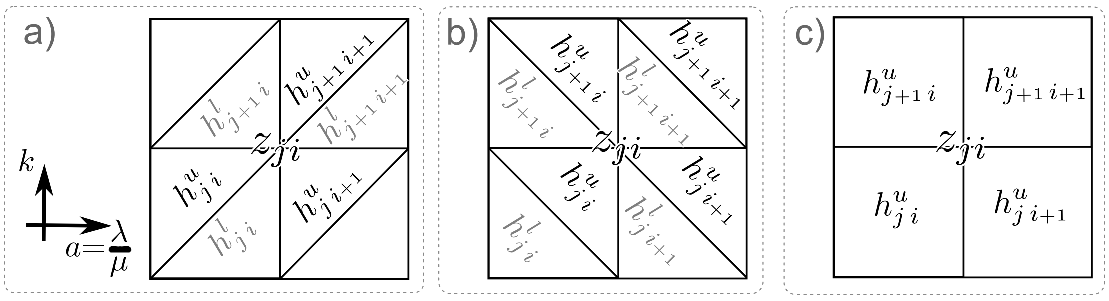


Figure 6: Surface approximations with MILP formulation relationship; a) tQP$_-$, b) tQP$_+$, and c) qQP.

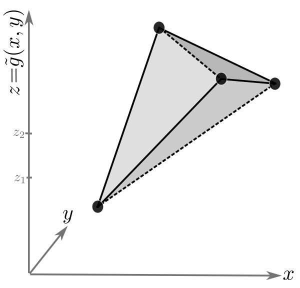


Figure 7: Triangle surface between four basepoints.

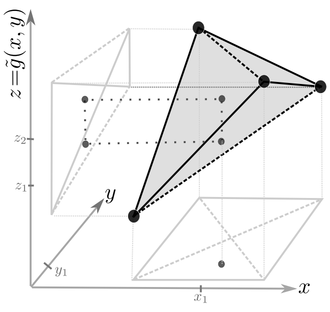


Figure 8: Quadrilateral surface; convex hull of convexly combining four basepoints.

$$h^u_{f\,j+1\,i} + h^l_{f\,j+1\,i} \quad \forall ji \quad \text{(tri-sync)} \tag{45}$$

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. Taking this fact into account, Section 6.1 discusses the generation of basepoint sets while maximising accuracy.

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 (46). Similarly, four basepoints of a quadrilateral can be used (47). However, while equations (46) form a linear system having a unique solution, the corresponding equations (47) form an underdetermined linear system having more unknowns (z_i) than equations.

$$\sum_{i=0}^{2} \alpha_i w_i = x, \ \sum_{i=0}^{2} \beta_i w_i = x, \ \sum_{i=0}^{2} \theta_i w_i = x \tag{46}$$

$$\sum_{i=0}^{3} \lambda_i w_i = x, \ \sum_{i=0}^{3} \beta_i w_i = x, \ \sum_{i=0}^{3} \theta_i w_i = x \tag{47}$$

This results in the following issue: For given x, y-coordinates, there are infinitely many solutions for z-coordinates admissible under equations (47). 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 (35) 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 qQP$(G, p, \tilde{\mathsf{N}})$ has objective function and constraints from (35)–(43) and replaced constraints (44) by (48).

$$\begin{aligned} z_{ji} \leq {} & h_{fji} + h_{f\,j\,i+1} + \\ & h_{f\,j+1\,i+1} + h_{f\,j+1\,i} \quad \forall ij \quad \text{(rec-sync)} \end{aligned} \tag{48}$$

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 (44) vs. (48).

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 tQP$_-$, tQP$_+$, or 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

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

```
1:  $p' \leftarrow p$
2:  while $p' \le p$ do
3:      if $p'$ is considered the first time then
4:          feasible, $x, y \leftarrow$ xQP$(G, p', \tilde{\mathsf{N}})$
5:          if feasible then
6:              $\forall f : y_f \leftarrow \lceil y_f \rceil,\ \lambda'_f = \sum_c x_{cf}$
7:              $\Delta \leftarrow \sum_f y_f - p$     // valid solution for $\Delta \le 0$
8:              if $\Delta = 0$ then return $x$, $y$     // direct hit
9:              if $\Delta < 0$ then     // add remaining resources
10:                 $y \leftarrow$ ALLOC($p$, $\lambda'$, $y$)
11:                 return $x$, $y$
12:             if $\Delta > 0$ then     // too many resources
13:                 $y \leftarrow$ DEALLOC($p$, $\lambda'$, $y$)
14:                 if DEALLOC$(\cdot)$ was feasible then
15:                     return $x$, $y$
16:                 else
17:                     decrease $p'$ by $\Delta$
18:         else     // infeasible with current $p'$
19:             increase $p'$ by 1
20:     else // $p'$ was considered in previous loop iterations
21:         increase $p'$ by 1
22: return SEARCH is infeasible
```

$\sum_f y_f \le p$. 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 $p, p-1, \dots, p_\downarrow$, 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 Δ resources; DEALLOC (Algorithm 4) is basically a twin of ALLOC removing resources one by one; details on DEALLOC are provided laterthis paper's extended version provides details on DEALLOC. Often removing all Δ resources will not be feasible because the current assignment distributes the demand improperly among the facilities. Then, the only adjusting the assignments

Algorithm 4 DEALLOC(p, λ', $y{=}k_f$) $\rightarrow$ y, T
Optimally reduce over-allocation, specified in vector y.

```
    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: $n \leftarrow \sum_f y_f - p$
3: if $n > 0$ then
4:     $y^{\text{sub}} \leftarrow$ MAXCOSTDROP($n$, $c_*$, $y^{\max}$)
           $\forall f : c_f(y') := \mathfrak{M} - \mathsf{N}(a_f, y_f + y')$
           with $\forall y = \mathsf{N}(.,1) : \mathfrak{M} > y$
           $\forall f : y_f^{\max} \leftarrow k_f - y_f^{\min}$
5:     $\forall f : y_f = k_f - y_f^{\text{sub}}$
6: return $y$, $\sum_f \mathsf{N}(a_f, y_f)$
```

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, $\sum_f \lambda_f / \sum_f k_f\mu_f \ge 0.96$ – 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 6.2.

Lemma 2. DEALLOC(p, λ', k) (Algorithm 4) removes p resources at fully allocated facilities $f \in F'$ so

Table 3: Properties of Erlang-C-based functions

name	function		property
Erlang-C	$\mathrm{EC}_\rho(k)$		decreasing [52], convex [24]
	$\mathrm{EC}_k(\rho), \mathrm{EC}_k(a)$		strictly increasing (str.inc.) [34], [52], convex [34]
# req. in queue	$\mathrm{N}^Q_\rho(k)$		strictly decreasing (str.dec.) and convex [24]
# req. in system	$\mathrm{N}_\rho(k)$		str.inc. and convex [24]
	$\mathrm{N}_{\mu,k}(\lambda)$	*	is convex (Theorem 2)
	$\mathrm{N}_\mu(\lambda, k)$	*	is not convex (Section 8.1)
∅ wait in queue	$\mathrm{T}^Q_{\lambda,\rho}(k)$		str.dec. and strictly convex [24]
	$\mathrm{T}^Q_{\mu,\rho}(k)$		str.dec. and strictly convex [24]
∅ wait in system	$\mathrm{T}_{\lambda,\mu}(k)$		decreasing and convex [17]
	$\mathrm{T}_{\lambda,\rho}(k)$		str.inc. and strictly convex [24]
	$\mathrm{T}_{\mu,\rho}(k)$		str.dec. and strictly convex [24]
	$\frac{1}{\mathrm{T}_{\mu,k}(\rho)}$		strictly concave [25]
	$\mathrm{T}_k(\lambda,\mu), \mathrm{T}_{k,\mu}(\lambda)$		strictly convex [25]
	$\mathrm{T}_\lambda(\mu, k)$		is convex [48]
	$\mathrm{T}_\mu(\lambda, k)$		is not convex (Section 8.1)

that $\sum_f \mathsf{N}({}^{\lambda'_f}/_{\mu_f}, y_f){=}T$ is minimised while ensuring $\lambda'_f{<}y_f\mu_f, \sum_f y_f{=}p$.

Proof. Similar to ALLOC, DEALLOC uses MAXCOSTDROP, but while ALLOC interprets dropped tokens as added resources, DEALLOC interprets dropped tokens as removed resources; this make adjustements and a new minimality deduction necessary. With $n{=}0$ no resources need to be removed $\sum_f k_f{=}p$ having one solution with $T{=}\sum_f \mathsf{N}(a_f, k_f)$ minimal; done. With $n{>}0$, algorithm MAXCOSTDROP computes how many resources are removed at which facility. After removing the resources, T increases by $\sum_f \mathsf{N}(a_f, k_f){-}\mathsf{N}(a_f, k_f{-}y_f) = T_\triangle$; so minimising $T_\triangle$ will also minimise T. As MAXCOSTDROP maximises the token costs (Theorem 1), we need to flip them around (line (4)). This way, $T_\triangle$ is minimal and so is T; done. □

4.7 Exploiting Problem Structure

This section discusses redundant constraints and decision variables for problems P, whose objective function f is linearised in their linear counterparts $\tilde{P}$ modelled by through a convex combinations[7]. Those counterparts usually use constraints like SOS-constraints to enforce that only at least two adjacent basepoints are active. According to [3], if P is to be minimised and f is convex then the basepoint constraint is redundant. Consequently, if presented objective functions are convex, their linearisation's constraints can be dropped, search space is simplified, and solving time is shortened.

To check convexity, the objective functions are split in two groups for using either the uni- or the bivariate PWL function $\tilde{\mathsf{N}}$ (respectively, Section 4.2 and Section 4.4, Section 4.5). For both groups, at first, the convexity of $\tilde{\mathsf{N}}$ and, then, the objective function itself is investigated. The literature has some results (Table 3) on convexity properties of Erlang-C-based functions (to which N belongs). Rows marked by a star correspond to presented $\tilde{\mathsf{N}}$ functions.

7. Describing $\tilde{f}$ through a convex combination has no relationship to $f/\tilde{f}$ being convex.

4.7.1 Univariate Group

For the first, univariate group with objective function (26) of cQP, Theorem 2 shows $\mathsf{N}_{\mu,k}(\lambda)$'s convexity and Theorem 3 shows the objective function's convexity. Consequently, the SOS2-constraint in (29) can be dropped.

Lemma 3 (Convexity of univariate functions)**.** $g(x)$ is convex ⇔ the second derivative $\bigtriangledown^2 f(x) \geq 0$ with domain $x{\in}\operatorname{dom}(g)$ being convex [10].

Lemma 4 (Convexity of nonnegative weighted sum)**.** If $g(x)$ is convex, then $wg(x)$,$w{\geq}0$, is convex. If $g_1(x)$ and $g_2(x)$ is convex, then $g_1(x){+}g_2(x)$ is convex [10]. Combining both, $g(x) = \sum_i w_i g_i(x)$, $w_i{\geq}0$, is convex.

Lemma 5 (Convexity of bivariate functions)**.** $g(x, y)$ is convex ⇔ the second derivative is positive semi-definite $\bigtriangledown^2 g(x, y){=}H \succeq 0$ with domain $xy{\in}\operatorname{dom}(f)$ being convex [10]; with definitions:
Hessian matrix $H{=}\nabla^2 g(x, y) = \begin{bmatrix} \frac{\partial^2}{\partial x^2} g(x,y) & \frac{\partial^2}{\partial x \partial y} g(x,y) \\ \frac{\partial^2}{\partial y \partial x} g(x,y) & \frac{\partial^2}{\partial y^2} g(x,y) \end{bmatrix}$
$n{\times}n$ matrix H is pos. semi-definite ⇔ $H{\succeq}0$ ⇔ $v^T H v{\geq}0$, $v{\in}\mathbb{R}^n_{\geq 0}$

Lemma 6 (Convexity of vector compositions)**.** Functions $g_1(x), .., g_n(x)$, $g_i{:}\mathbb{R}^m{\to}\mathbb{R}$, are convex and nondecreasing ∨ function $h(x)$, $h{:}\mathbb{R}^n{\to}\mathbb{R}$, is convex ⇔ $h(g_1(x), .., g_n(x)) = f(x)$, $f{:}\mathbb{R}^m{\to}\mathbb{R}$ is convex [10].

Theorem 2. *Function $\mathsf{N}_{\mu,k}(\lambda)$ is convex, $\lambda, \mu{\in}\mathbb{R}_{>0}$, $k{\in}\mathbb{Z}_{>0}$, $\lambda{<}\mu k$.*

Proof. Function $\mathsf{N}_{\mu,k}$ can be rewritten as (49):

$$\mathsf{N}_{\mu,k}(\lambda) = \underbrace{\frac{\lambda}{k\mu - \lambda}}_{=A} \underbrace{EC_k(\frac{\lambda}{\mu})}_{=B} + \underbrace{\frac{\lambda}{\mu}}_{=C} = AB + C \tag{49}$$

$$\bigtriangledown^2 \frac{x}{a{-}x} {=} \frac{d^2}{dx^2} \frac{x}{a{-}x} {=} \frac{2\,x}{(a{-}x)^3} {+} \frac{2}{(a{-}x)^2} > 0, x{<}a \tag{50}$$

$$h(x, y) < h(x{+}\Delta_x, y{+}\Delta_y),\ \Delta_x, \Delta_y > 0 \tag{51}$$

$$H{=}\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]{\succeq}0 \ \Leftrightarrow\ \forall v_1, v_2{\in}\mathbb{R}_{\geq 0}{:}\ [v_1\ v_2] H_h [v_1\ v_2]^T{=}2v_1 v_2{\geq}0 \tag{52}$$

C is linear (fix μ) and, thus, convex. B is convex [34] with ${}^{\lambda}/_{\mu}{=}a$ being linear and, thus, convex. A is convex because of relationship (50) and Lemma 3. AB can be treated as a function composition $h(A, B){=}h(x, y){=}xy$. $h(x, y)$ is nondecreasing (51) and h is convex (52). Then AB is convex (Lemma 6), $AB{+}C$ is convex (Lemma 4), and $\mathsf{N}_{\mu,k}$ is convex (49). □

Theorem 3. *The objective function (26) is convex.*

Proof. The function (26) can be rewritten as (53). The sum $\sum_i z_{fji}\beta_{ji}$ is the MILP formulation of $\tilde{\mathsf{N}}_{\mu,k}$ for $j{\in}J$, where only one j (at each facility) is active (19) and (30); parameter μ correspond to factor μ in (28).

$$f(x) := \overbrace{\frac{1}{\sum_c \lambda_c}}^{\mathrm{A}} \left(\overbrace{\sum_{cf} x_{cf} l_{cf}}^{\mathrm{B}} + \overbrace{\sum_f \tilde{\mathsf{N}}_{\mu,k} \left(\underbrace{\sum_c x_{cf}}_{\mathrm{D}} \right)}^{\mathrm{C}} \right) \tag{53}$$

Term C is convex (Lemma 6) because $\mathsf{N}_{\mu,k}$ is convex (Theorem 2) and term D is linear/convex. Term B is linear/convex and term A is a constant weight, hence $f(x)$ is convex (Lemma 4). □

4.7.2 Bivariate Group

The second, bivariate group contains problems tQP$_-$, tQP$_+$, qQP sharing the same objective function (35). The corresponding bivariate functions $\mathsf{N}(a,k)$ and $\mathsf{EC}(a,k)$ has integer parameter k which commonly cannot be differentiated. However, Tokgöz et al. [47] develop a differentiability extension for mixed functions (and they also used it to prove convexity of their bivariate mixed function, similar to our attempt).

Lemma 7 (Derivative of univariate integer functions)**.** For integer function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ the first $\nabla(f)$ and second derivative $\nabla^2(f)$ is defined by: $\nabla(f) = f(x{+}1) - f(x)$, $\nabla^2(f) = f(x{+}2) - 2f(x{+}1) + f(x)$ [48].

Lemma 8 (Hessian of bivariate mixed function)**.** For mixed function $f : \mathbb{R}_{\geq 0}{\times}\mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ the second derivative is defined by Hessian matrix:

$$H = \nabla^2 f(a,k) = \begin{bmatrix} \nabla_{kk} f(a,k) & \nabla_k \frac{\partial}{\partial a} f(a,k) \\ \frac{\partial}{\partial a}\nabla_k f(a,k) & \frac{\partial^2}{\partial a \partial a} f(a,k) \end{bmatrix}$$

So in order to discard the linearisation constraint, the objective function, function $\mathsf{N}(\lambda,k)$, $\mathsf{N}(a,k)$, and $\mathsf{EC}(a,k)$ have to be convex; but we show they are not convex, so no constraint is discarded.

Theorem 4. Function $\mathsf{EC}(a,k)$, $\mathsf{N}(a,k)$ is not convex, $a{\in}\mathbb{R}_{>0}$, $k{\in}\mathbb{N}_{>1}$, $a{<}k$.

Proof. Assume $\mathsf{EC}(a,k)$ is convex then (54) holds with (Lemma 5). However, for some values detailed in Section 8.1 the term $v^T H v$ is negative and, by this, contradicting (54). As a result $\mathsf{EC}(a,k)$ is not generally convex in both parameters a, k and so $\mathsf{N}(a,k)$ is not. (Section 8.1 contains detailed equations.)

$$\nabla^2 \mathsf{EC}(a,k){=}H = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \succeq 0$$
$$\Leftrightarrow \forall v_1, v_2{\in}\mathbb{R}_{\geq 0}\colon v^T H v{=}v_1^2 A{+}v_1 v_2 B{+}v_1 v_2 C + v_2^2 D \geq 0 \quad (54)$$

□

Theorem 5. Function $\mathsf{N}_\mu(\lambda,k)$ is not convex, $\lambda{\in}\mathbb{R}_{>0}$, $k{\in}\mathbb{N}_{>1}$, $\lambda{<}\mu k$.

Proof. Function N_μ can be rewritten as (55).

$$\mathsf{N}_\mu(\lambda,k) = \underbrace{\frac{\lambda}{k\mu-\lambda}}_{=A} \underbrace{EC(\frac{\lambda}{\mu},k)}_{=B} + \underbrace{\frac{\lambda}{\mu}}_{=C} = AB + C \quad (55)$$

Assuming N_μ is convex, then the argumentation follows the same pattern as in proof of Theorem 2. However, since B is non-convex (Theorem 4) but have to be, N_μ is non-convex. □

Theorem 6. The objective function (35) of tQP$_-$, tQP$_+$, qQP is not convex.

Proof. If (35) were convex, $\mathsf{N}_\mu(\lambda,k)$ should be also convex but it is not (Theorem 5). □

While objective function (35) is not convex (mixed integer convexity) is to not safe to discard the linearisation constraints from tQP$_-$, tQP$_+$, qQP.

4.8 Linearise Erlang-C Formula

This section describes our approach to linearise the Erlang-C function EC (2). Our algorithm based on Imamoto et al. [27] obtains basepoints with small approximation error [32]. This small error is achieved by successively changing the basepoints and estimating their error. To compute basepoint changes, our algorithm needs the first derivative of the function to be approximated. This section proposes an efficient, vectorised recursive functions to compute EC, its derivative, and Erlang-C-related functions fast. This reduces the runtime for obtaining the basepoints.

Beside the efficiency needs, EC becomes not computable with a plain implementation[8] for $k{>}145$ and a near k. For such a, k, the term a^k or $k!$ becomes large enough to overflow[9] and further processing is distorted. Alternatively, algebra systems such as Maxima resolves expressions symbolically allowing a greater accuracy paid by a longer runtime and memory usage[10]. A better alternative is proposed by Pasternack et al. [42]: The recursive function V (56) replaces the computationally challenging part.

For an M/M/k-queuing system with arrival rate λ, service rate μ, and k servers available, the probability of no jobs $P_{0\,\text{jobs}}$ (57) can be rewritten as (58), $a{=}\lambda/\mu{<}k$. As part of the Erlang-C formula (59), it can be rewritten similarly using V (60). Other queue systems' performance measures use EC. Examples are the expected number of requests in the system $\mathsf{N}(a,k)$ (61) or in queue $\mathrm{N}^Q(\lambda,\mu,k)$ (62), the expected time of request spent in system $\mathrm{T}(\lambda,\mu,k)$ (63) or in queue $\mathrm{T}^Q(\lambda,\mu,k)$ (64) [8].

$$\begin{aligned}
\mathsf{V}(a,k) &= \sum_{i=0}^{k-1} \frac{k!}{i!} a^{i-k} \\
&= \begin{cases} \frac{1}{a} & \text{if } k{=}1 \\ \frac{k}{a}(\mathsf{V}(a,k{-}1){+}1) & \text{else} \end{cases} && (56) \\
P_{0\,\text{jobs}} &= \left(\frac{k a^k}{k!\,(k-a)} + \sum_{i=0}^{k-1} \frac{a^i}{i!} \right)^{-1} && (57) \\
&= (\frac{a^k}{k!}\frac{k}{k-a} + \frac{a^k}{k!} \underbrace{\sum_{i=0}^{k-1} \frac{a^i}{i!}\frac{k!}{a^k}}_{=V(a,k)})^{-1} \\
&= \left(\frac{a^k}{k!} \frac{k+(k-a)V(a,k)}{k-a} \right)^{-1} \\
&= \frac{k!}{a^k} \frac{k-a}{k+(k-a)V(a,k)} && (58) \\
\mathrm{EC}(a,\,k) &= \frac{k a^k}{k!\,(k-a)} P_{0\,\text{jobs}} && (59)
\end{aligned}$$

8. Implementation in Python 2.7, Scipy 1.8.0, 64 bit-floats; computing EC(139, 140) on our laptop lasts 16 ms.

9. The number of floating-point bits determines the largest number which can be encoded, in our case $\approx 1.81{\cdot}10^{308}$ with 64 bit. If an operation results in a value larger than this limit the result is replaced by a special value, `inf`, indicating the overflow.

10. Computing EC(500, 1000) with Maxima 5.26.0 took 0.6 s and consumed 80 MiB. EC(1000, 2000) took 3.8 s and consumed 374 MiB.

Algorithm 5 Vectorised function $\mathsf{V}(a,k)$ (56)

```
1: function V($\bar{a}$, $\bar{k}$)
2:     $\bar{r} \leftarrow {}^{1}/_{\bar{a}}$ with same length as $\bar{a}$ and $\bar{k}$
3:     for $i = 2..max\{\bar{k}\}$ do
4:         mask$\leftarrow \bar{k} > i$
5:         $r[\text{mask}] \leftarrow \frac{k[\text{mask}]}{a[\text{mask}]}(r[\text{mask}]+1)$
       return $r$
```

With array arithmetic $\bar{a}{=}a_1,...,a_m$: element-wise operations $\bar{a}\circledast\bar{b} \Leftrightarrow \forall i{:}a_i\circledast b_i$ and masking $\bar{a}[1,0,...,0,1]{=}a_1,a_m$.

$$= \frac{ka^k}{k!\ (k-a)}\ \frac{k!}{a^k}\frac{k-a}{k+(k-a)V(a,k)}$$

$$= \frac{k}{k+(k-a)V(a,k)} \tag{60}$$

$$\mathsf{N}(a,k) = a + \frac{a}{k-a}\ \mathsf{EC}(a,k)$$

$$= a + \frac{a}{k-a}\ \frac{k}{k+(k-a)V(a,k)}$$

$$= a + \frac{ak}{(k-a)k+(k-a)^2V(a,k)} \tag{61}$$

$$\mathsf{N}^Q(\lambda,\mu,k) = \frac{\lambda}{\mu k-\lambda}\ \frac{k}{k+\frac{\mu k-\lambda}{\mu}V(\frac{\lambda}{\mu},k)}$$

$$= \frac{\lambda k}{k(\mu k-\lambda)+\frac{(\mu k-\lambda)^2}{\mu}V(\frac{\lambda}{\mu},k)} \tag{62}$$

$$\mathsf{T}(\lambda,\mu,k) = \frac{1}{\mu} + \frac{k}{(\mu k-\lambda)k+\mu(k-\frac{\lambda}{\mu})^2V(\frac{\lambda}{\mu},k)} \tag{63}$$

$$\mathsf{T}^Q(\lambda,\mu,k) = \frac{k}{k(\mu k-\lambda)+\frac{(\mu k-\lambda)^2}{\mu}V(\frac{\lambda}{\mu},k)} \tag{64}$$

Two technical performance improvements are worth mentioning: Pasternack et al. [42]'s recursive function is implemented as a loop that not only avoids time needed to push local variables on the stack for each recursive call but also evades the limited stack depth also limiting k. Implementing this improvement results in computation magnitudes faster[11] than the other alternatives. The function EC with V becomes not computable for $k{\gtrsim}100{,}000$, sufficient enough for our scenario. In addition a vectorised version of $\mathsf{V}(a,k)$ allows computing several values $\mathsf{V}(a_0,...,a_m;\ k_0,...,k_m){=}t_0,...,t_m$ in a row (Algorithm 5). By masking the array fields, different loop iterations depths $k_0,...,k_m$ are processed in one loop (Line 5). This vectorises our linearisation algorithm and, thus, is much faster than a simple implementation.

Finally, the first derivation of N is needed. The basepoints are obtained for parameter a with fix k, $\mathsf{N}_k(a)$. As a result, $\mathsf{N}_k(a)$ is differentiated only for a; details in Section 8.2. The first derivative $\frac{d}{da}\mathsf{N}_k(a)$ (72) is rewritten with V (73).

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 $\mathsf{QP_A}$ differs from QP in three points: Firstly, $\mathsf{QP_A}$ minimises the *maximal* response time whereas QP minimises the *average* response time. Secondly, assignments in $\mathsf{QP_A}$ are predefined whereas QP also decides the assignments. Thirdly, $\mathsf{QP_A}$ has no resource limit per facility, which QP has. These three differences necessitate adjustments to $\mathsf{QP_A}$'s heuristic.

11. Computing EC(139, 140) via V took 0.6 ms (Python 2.7).

Algorithm 6 Combines local solutions to find new ones.

```
1: function Genetic(G , p)
2:     /* Create a random population of solutions */
3:     $P \leftarrow \emptyset,\ l \leftarrow \lfloor\sqrt{|F|}\rfloor$
4:     while not enough solutions in $P$ do
5:         $F_s \leftarrow l$ random facilities from $F$
6:         $F_s, y_f, t \leftarrow$ Descent($G,\ F_s,\ p,\ F,\ \emptyset$)
7:         if solution not found then increase $l$
8:         else
9:             if $F_s{\notin}P$ then add $(F_s, y_f, t)$ to $P$
10:    while not enough merge steps are done do
11:        /* merge two solutions */
12:        $F_s, F'_s \leftarrow$ two random $F_s \in P$
13:        $F_\mathrm{U} \leftarrow F_s{\cup}F'_s;\ F_\mathrm{I} \leftarrow F_s{\cap}F'_s;$
14:        $F_\mathrm{M} \leftarrow$ three random $f \in F \setminus F_\mathrm{U}$          // Mutation
15:        $F_\mathrm{D} \leftarrow (F_\mathrm{U} \setminus F_\mathrm{I}) \cup F_\mathrm{M}$
16:        $F_\mathrm{N} \leftarrow F_\mathrm{I}$ with one $f{\in}F_\mathrm{D}$ added          // Mutation
17:        $F_s, y_f, t \leftarrow$ Descent($G, F_\mathrm{N}, p, F_\mathrm{D}, F_\mathrm{I}$)
18:        if solution found $\wedge\ F_s{\notin}P \wedge t <$ largest $t$ in P then
19:            replace worst solution in $P$ with current
20:    return $F_s, y_f, t$ from $P$ with smallest $t$
```

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 8) 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 7), 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 6) 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 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_I of both solutions' subsets F_s, F'_s is part of the new offspring facility set F_N. The domain F_D is the union of F_s, F'_s with three random new facilities F_M; F_D; it limits the following explorations by Descent. The offspring F_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

Algorithm 7 Refines solution towards local optimum

```
1: function DESCENT($G$, $F_s$, $p$, $F_\text{D}$, $F_\text{I}$)
2:     $F_s^\text{min}, y_f^\text{min}, t^\text{min} \leftarrow F_s$, SOLVE($F_s'$, $p$)
3:     while smaller $t^\text{min}$ was found do
4:         for $F_s' \in$ NEIGH($F_s^\text{min}$, $F_\text{D}$, $F_\text{I}$) do
5:             $y_f'$, $t'$ ←SOLVE($G$, $F_s'$, $p$)
6:             if $t' < t^\text{min}$ then
7:                 $F_s^\text{min}, y_f^\text{min}, t^\text{min} \leftarrow F_s', y_f', t'$
8:     return $F_s^\text{min}, y_f^\text{min}, t^\text{min}$
```

Algorithm 8 Approximate assignment and optimal allocation

```
1: function SOLVE($G$, $F_s$, $p$)
2:     $\forall c, f{\in}C{\times}F : x_{cf} \leftarrow 0$
3:     for $c, f{\in}C{\times}F_s$ sorted by increasing $l_{cf}$ do
4:         dem ← $\lambda_c - \sum_{f'} x_{cf'}$
5:         cap ← $k_f \mu_f - \sum_{c'} x_{c'f}$
6:         if dem $> 0 \wedge$ cap $> 0$ then
7:             $x_{cf} \leftarrow \min\{$ dem , cap $\}$
8:     return ALLOC($G$, $\forall f : \sum_c x_{cf}$, $p$)
```

consisting of facility subsets constructed by a) removing one facility , b) adding one facility, or c) doing both (65).

$$\begin{aligned} \text{NEIGH}(F_s, F_\text{D}, F_\text{I}) = \{F_s \cup \{f\} \mid \forall f{\in}F_\text{D} \setminus F_s\} && \text{(a)} \\ \cap \{F_s \setminus \{f\} \mid \forall f{\in}F_s \setminus F_\text{I}\} && \text{(b)} \\ \cap \{F_s \cup \{f\} \setminus \{f'\} \mid \forall f{\in}F_\text{D} \setminus F_s, \forall f'{\in}F_s \setminus F_\text{I}\} && \text{(c)} \\ \text{with } F_\text{I} \subseteq F_s \subseteq F_\text{D} \subseteq F && (65) \end{aligned}$$

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_D, F_I restrict the cardinality of NEIGH and thus the number of instances solved by SOLVE. This is done by ensuring that F_I – the intersection of the two parents – is always part of F_s and that F_D – the modified union of the two parents – is the subset of F to which F_s can grow.

SOLVE (Algorithm 8) computes the assignment and allocation for a given facility subset F_s using p resources. The first part assigns the demand λ_c to the closest facilities. This assignment is complexer than QP$_\text{A}$'s assignments, which neither considers resource capacities (μ_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 (QP$_\text{A}$) 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 (Δ) 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, it does not compute the total average RTT in all cases. Let us consider two clients c, c' with demand, $\lambda{=}1$, and facilities f, f' with one capacity, $\mu{=}1$, having the following latencies $l_{cf}{=}1$, $l_{cf'}{=}l_{c'f}{=}2$, $l_{c'f'}{=}4$. The assignment heuristic (Line 2–7) computes $x_{cf}{=}x_{c'f'}{=}1$. First the c, f pair is visited, then $c'f$ is not possible as capacity is exceeded, and finally $c'f'$ is paired. These assignments result in total RTT 5 but the optimum is 4 with assignment $x_{cf}{=}x_{c'f'}{=}1$.

Table 4: Investigated values for J and m.

J's shorthand	$J = n$	n	m
k100	$1, ..., 100$	100	30
fib	$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 100$	11	15
2^i	$1, 2, 4, 8, 16, 32, 64, 100$	8	8
3^i	$1, 3, 9, 27, 81, 100$	6	6
4^i	$1, 4, 16, 64, 100$	5	4
k3	$1, 50, 100$	3	

One way to handle this case, is to check for total distance reduction when swapping assignments in a post-processing. The swapping procedure is very time consuming as considering all pairs only once is not sufficient to obtain the optimal solution. The described special case rarely occurred in our test instances and we had favoured short runtime, so that we omitted such a post-processing.

The presented assignment heuristic applies an extended nearest-facility assignment rule and, hence, is more restrictive than QP. We tested an alternative implementation for SOLVE: Assignment and allocation are obtained by our fastest optimisation problem qQP for facility subset F_s. However, SOLVE as the basic function of H is called very often and the runtimes were unsatisfactory longer than the presented implementation, much longer than any other presented approach to solve QP discussed in Section 6. Because of this, we entirely skipped this variant.

6 EVALUATION

6.1 Obtaining the Basepoints

The linearised formulations' quality and solving time depend on the choice of basepoints. No matter if cQP considers separate basepoint sets for each curve or the other formulations form a mesh of triangles or quadrilaterals, more basepoints improve the linearisation accuracy and support a good solution quality. But more basepoints also increases the search space size and extends solving time.

Four control factors influences the accuracy: a) The number of basepoints m for one curve $\tilde{\text{N}}_j(a)$; b) the number of curves/allocated resources n, $1{<}n{\leq}k_f$; c) the linearisation interval's upper bound for a and j; d) the basepoint positions itself.

For factor (c), we allow $0{<}a{<}0.98j$ similar to our previous work [32] and $1{<}j{\leq}100$. Setting the maximal number of available resources $\forall f : k_f{=}100$ is large enough for our evaluation. Our findings can be applied for larger value of k_f that decrease either the linearisation accuracy (m fix) or the number of basepoints m (when accuracy is fixed) [32]. For factor (c), our algorithm [32] is applied obtaining basepoint sets with high linearisation accuracy. For factor (a) and (b), different configurations for m and J are investigated. Table 4 shows these configurations. For example, $J{=}[1, 50, 100]$

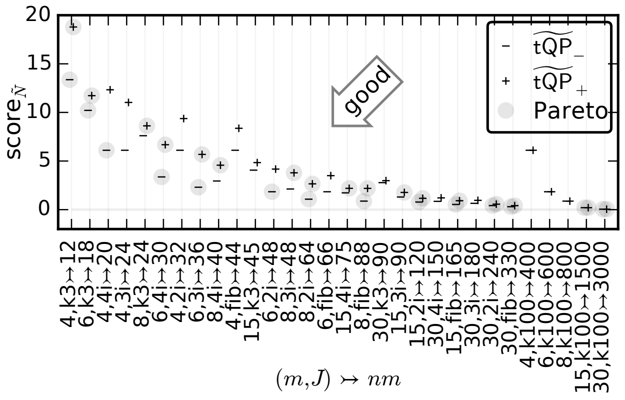


Figure 9: Surface approximation accuracy $\epsilon_{\tilde{N}}$ (66).

means $3{=}n$ curves for $y{=}1$, 5, or 100 are considered and has the shorthand k3. For each list J, the corresponding $\tilde{\mathrm{N}}_j(a)$, $j{\in}J$ are either approximated by 4, 6, 8, 15, or 30 basepoints. The table describes 30 different configurations (J, m).

The concrete values for m result from following thoughts: When adding a basepoint to a PWL function (for one curve), the accuracy increment successively decreases [32], e.g., adding a basepoint to 4 basepoints helps more than adding one to 8 basepoints. So, many low values for m covers the interval that the accuracy is significantly changed and few large values would provide hints towards the asymptotic behaviour. The cases for 2 or 3 basepoints are dropped; $m{=}2$ results in one line segment and with $m{=}3$, having two line segments, approximating our convex function is still results in low accuracy.

The concrete values for J result from following thoughts: As discussed in Section 4.3, dropping larger $j{\in}J$ from J results in less inaccuracy than dropping small j, so we choose sequences like $a_n{=}2^n$ favouring many small values and few large values. Additionally, $J{=}k100$ is added as a baseline comparison using all curves.

Figure 9 shows the error $\epsilon_{\tilde{N}}$ defined in equation (66) for each (m, J) configuration (Table 2) plotted as a function of number of basepoints nm; its labels encode the different configurations for m, J (using J's shorthand). Two symbols correspond to the two triangle orientations (Figure 6) used to define the approximating surface $\tilde{S}$.

$$\begin{aligned}&\epsilon_{\tilde{N}} := max_{a,k} |\, \tilde{N}(a,k) - N(a,k)\, | \\ &\tilde{N} \text{ is paremeterised by } (m,\ J,\ \text{triangle orientation} + /-) \qquad (66)\end{aligned}$$

Figure 9 also illustrates the trade-off between two conflicting metrics: Few basepoints nm, shown on the left, and low error $\epsilon_{\tilde{N}}$, shown on the right. Pareto optimal solutions (for each triangle orientation) are shown grey shaded; those solutions cannot be improved in one metric without degrading the other.

For our evaluation we choose the basepoint sets as follows: Since a high number of basepoints nm increases the search space size, large $nm{>}330$ were skipped; the lowest error of these skipped configurations is marginally smaller than the lowest error of the remaining configurations. The remaining nm values were split in three partitions. In each partition, one configuration was chosen with the lowest error $\epsilon_{\tilde{N}}$ satisfying the Pareto property: $(m, J){=}(15, 2^i), (8, 3^i), (6, 4^i)$. In additionally, configuration (8,k100) from the baseline basepoint sets (k100) were added. The number of basepoints $m{=}8$ provides a good trade-off between low error and few basepoints, e.g., with 25% less basepoints the error nearly doubles (6,k100) and with 88% more basepoints the error is only divided by four (15,k100).

6.2 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[12] [39], topology zootable 5 [33], and kingtrace[13] [22]. The topology sources offer 534 candidate topologies from which 8 were selected (Table 5) by three properties: Firstly, the number of nodes increases the problem size quadratically. We selected three small (marked as n-) (with at least 20 nodes), three medium (n/), and one[14] large (n+) topologies. Secondly, the round trip times contribute to the objective function. The selected topologies have low (l-), medium (l/), or high (l+) average round trip times, $\varnothing\text{RTT}{=}{}^{1}\!/_{|E|} \sum_{vv'} l_{vv'}$. 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 (marked by d) having different quartiles of node degrees as the other topologies (Table 5).

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

The second factor $\hat{D}$ describes how the individual Poisson processes' arrival rates λ_c are assigned to nodes c. We conceive of λ_c as random variables with expected value $\hat{\lambda}$, distributed according to three different distributions: a) $\hat{D}\hat{=}\mathrm{N}(\text{mean, std.dev.}){=}\mathrm{N}(\hat{\lambda}, {}^{\hat{\lambda}}\!/_{20}){=}\mathrm{N}_1$ reflecting small variation around the mean, b) $\hat{D}\hat{=}\mathrm{N}(\hat{\lambda}, \hat{\lambda}){=}\mathrm{N}_2$ reflecting large variations around the mean, c) $\hat{D}\hat{=}\mathrm{Exp}(\hat{\lambda})$ with even stronger variations to reflect local hot spots. We ignore nodes with negative arrival rates, hence the random variables λ_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[15] 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

12. Round trip times were approximated by geographical distances [29]. Nodes without geolocation positions were removed and their neighbours were directly connected.

13. 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.

14. Only one of three large topologies were solved within the time limit, see extended version [31].

15. 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, $\sum_f \lambda_f / \sum_f \mu_f y_f {<} \sum_f \lambda_f / p \ \max_f \mu_f$.

Table 5: Topology overview.

Name$_{\text{source}}$ $\hat{G}$	Mark	Size $\lvert N\rvert$	Density $2^{\lvert N\rvert}/\lvert E\rvert(\lvert E\rvert-1)$	∅RTT [ms]	diam$_{\text{RTT}}$ [ms]	Degree Perc. [min 25 % median 75 % max]
Columbus $_{\text{zoo}}$	d	31	0.211	326.5	1203.8	[1.0 3.0 4.0 11.0 12.0]
Cesnet20 $_{\text{zoo}}$	n- l-	38	0.064	189.5	387.3	[1.0 1.0 1.0 3.0 15.0]
giul39 $_{\text{snd}}$	d	39	0.116	447.9	852.1	[3.0 3.0 4.0 5.0 8.0]
pioro40 $_{\text{snd}}$	n- l+	40	0.114	545.6	1140.8	[4.0 4.0 4.0 5.0 5.0]
Colt $_{\text{zoo}}$	n/ l/	149	0.016	546.8	1338.5	[1.0 1.0 2.0 3.0 18.0]
UsCarrie $_{\text{zoo}}$	n/ l/	152	0.016	732.7	2076.6	[1.0 2.0 2.0 3.0 6.0]
Cogentco $_{\text{zoo}}$	n/ l/	186	0.014	865.9	2260.1	[1.0 2.0 2.0 3.0 9.0]
Kdl $_{\text{zoo}}$	n+ l+	726	0.003	1476.4	4317.1	[1.0 2.0 2.0 3.0 10.0]
intercon* $_{\text{zoo}}$	n+ l/	1108	0.002	595.8	1758.4	[1.0 1.0 2.0 3.0 54.0]
kingtrace*	n+ l-	1819	0.180	47.9	726.4	[3.0 220.0 242.0 467.0 559.0]

(*) Marked topologies were too large to be solved efficiently.

instead of 1 reflects the linearisation bound α_m=0.98k (factor (d) in Section 6.1) 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 (Table 5), 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. From Section 6.1, $\hat{B}$ is $(m, J) \in \{(15, 2^i), (8, 3^i), (6, 4^i), (8, k100)\}$.

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 (cQP, tQP$_-$, tQP$_+$, qQP) use 3 basepoint sets $\hat{B}$=$\{(15, 2^i), (8, 3^i), (6, 4^i)\}$. In addition, cQP is solved with basepoint set $(8, k100)$ (Section 6.1) 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[16] is used and is configured to stop solving after one hour. Especially for larger topologies, this often causes optimality gaps up to 20%.

The remaining section first discusses example results (Section 6.2.1) and then presents aggregated statistics revealing the core findings.

6.2.1 Detailed Results

Figure 10a shows two plots with either the solution quality as the average response time and the corresponding average solving time with 95%−confidence intervals. They compare all 14 solving approaches, represented by different symbols, for different utilisation levels $\hat{p}$. Figure 10a shows results for medium topology `Colt` with exponential demand distribution. Below, Figure 10b shows results for the same topology but with normal demand distribution N$_1$. At last, Figure 10c shows results for smaller topology `Pioro` with demand distribution N$_2$.

In the shown plots, the baseline approach cQP$_{8,k100}$ computes the best solution (smallest avg. resp. time) among the other approaches but is the slowest approach, except for small topologies (e.g. Figure 10c) where other approaches are significant slower. Section 6.2.2 describes how cQP$_{8,k100}$ performs among the other topologies.

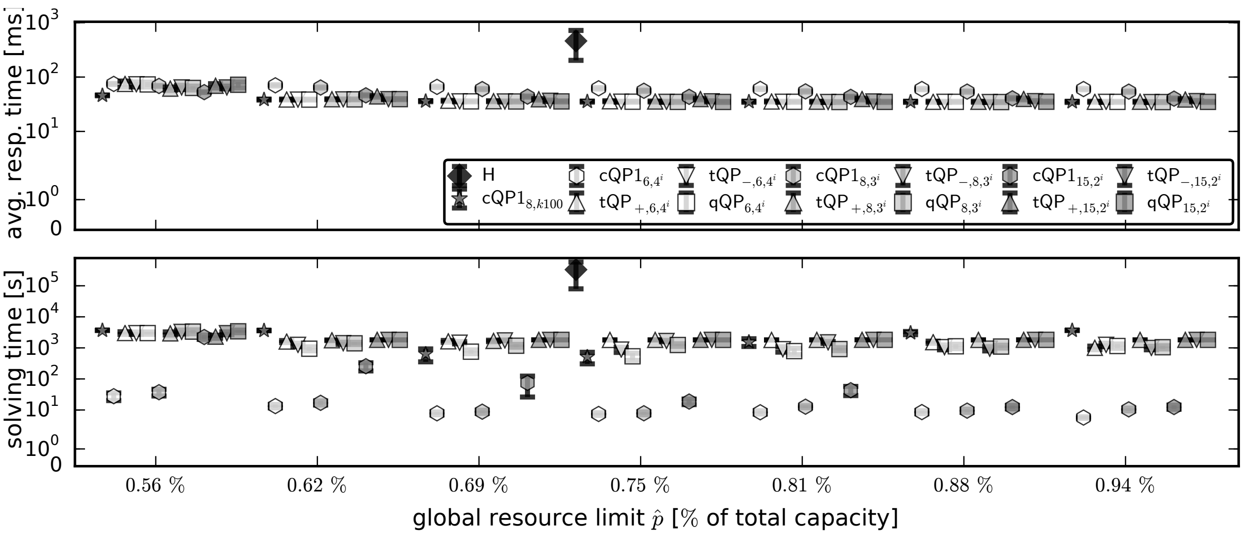
The different formulations cQP$_j$, tQP$_{+,j}$, tQP$_{-,j}$, and qQP$_j$ are grouped together and the three basepoint sets $j \in \{(6, 4^i), (8, 3^i), (15, 2^i)\}$ form three groups distinguished in Figure 10a-c by different grey levels. Among these four approaches, the thinned curve formulation cQP$_j$ is the simplest and the fastest in each group, sometimes magnitudes faster, while the quality is a bit worse than the other approaches. When comparing the surface approximations, the quadrilateral-based formulation qQP is as good as the other two formulations but is solved faster in some cases. Section 6.2.3 describes how the surface approximations perform among all configurations.

Finally, the last solving approach uses the genetic algorithm as an heuristic. For medium and large topology, the heuristics's solving time exceed a maximum runtime of six hours in most cases; for those cases no measurements are available. Section 6.2.5 describes how the heuristic performs among all configurations.

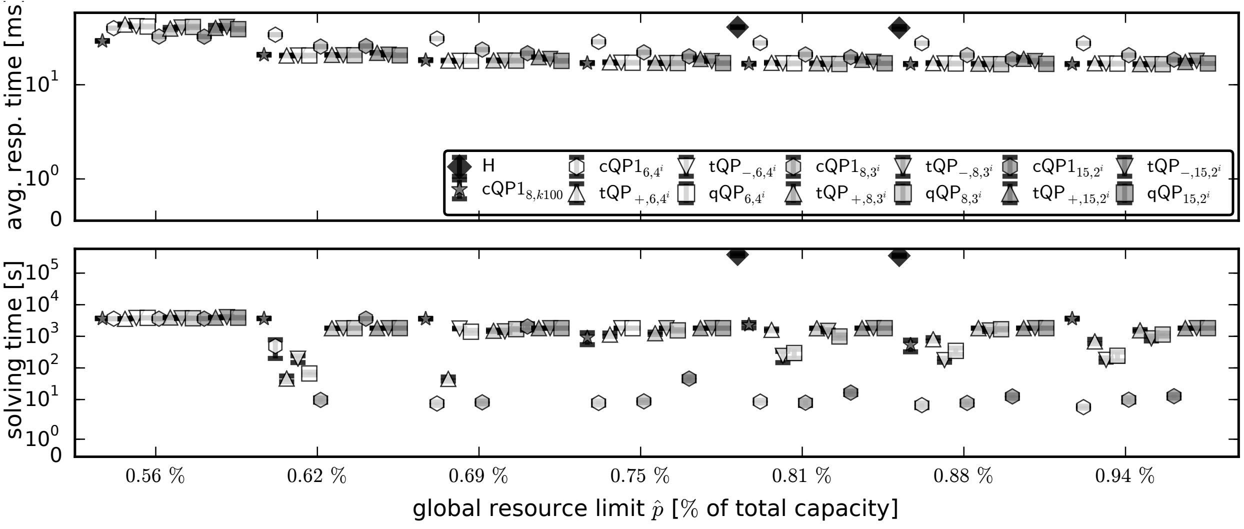
6.2.2 Baseline Algorithm

The approach 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 11.

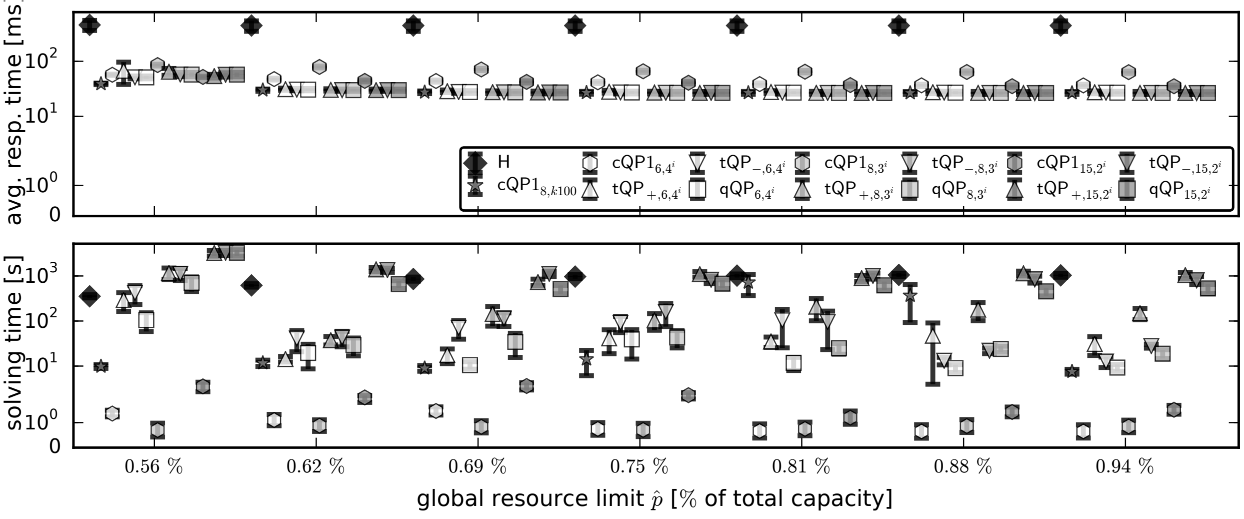
16. Gurobi version 5.6.3, Python version 2.7.9



(a) For $\hat{G} = \texttt{Colt}$, $\hat{D} = \text{Exp}$



(b) For $\hat{G} = \texttt{Colt}$, $\hat{D} = \text{N}_1$



(c) For $\hat{G} = \texttt{Pioro}$, $\hat{D} = \text{N}_2$

Figure 10: Picked detail comparison of algorithms – quality and runtime.

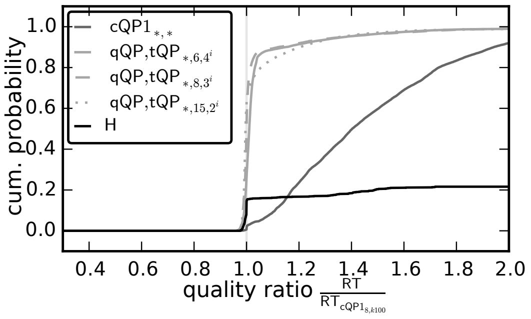


Figure 11: Comparing the quality of baseline approach cQP$_{8,k100}$ with the other approaches as a CDF plot.

We identify five groups of similar solution alternatives. The first group contains all thinned curves approaches (cQP$_{6,4^i}$,cQP$_{8,3^i}$,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 6.2.5 has more details about the heuristic.

To conclude, solutions obtained by 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.

6.2.3 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 tQP$_{+,j}$, tQP$_{-,j}$, or qQP$_j$ by that of 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 12 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 cQP$_j$; 32%, 48%, and 70% of the surface approaches took 100 times longer than the thinned curves approach.

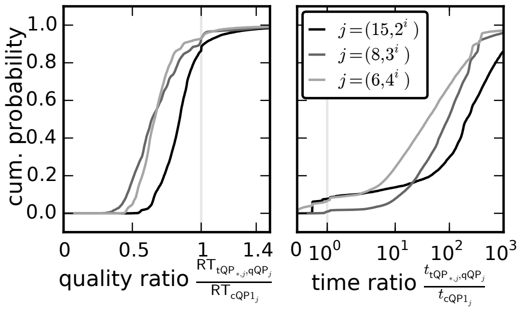


Figure 12: Comparing the quality and solving time ratios of cQP$_j$ with tQP$_{*,j}$ and qQP$_j$.

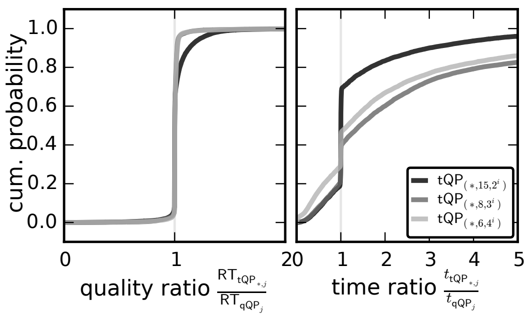


Figure 13: Comparing the quality and solving time ratios of cQP$_j$ with tQP$_{*,j}$ and qQP$_j$.

The cause of this solving time difference is two-fold: First, 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*.

6.2.4 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 13).

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 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 tQP$_*$ and qQP without using SEARCH. While

Table 6: Obtained solutions of all configurations with heuristic.

Name, G	Columbus	Cesnet20	guil39	pioro
Size, $\|N\|$	31	38	39	40
# Sol. %	30	39.7	40	89.3
Name, G	Colt	UsCarrier	Cogento	Kdl
Size, $\|N\|$	149	152	186	726
# Sol. %	0.2	0	0.7	0

this results in ignoring limit p and in over-utilised allocations, the solutions show the following pattern: (a) The objective value of tQP_'s solutions were always smaller/better than tQP$_+$'s objective value, supporting the necessity of considering the triangle orientation; (b) qQP's objective value was always the same as tQP_'s objective values, which supports the argumentation for quadrilaterals.

6.2.5 Heuristic

The heuristic has a long solving time[17], 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. Table 6 shows how many attempts were successful, grouped by topologies.

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. Improvements are possible, such as caching a history of DECENT calls avoiding visiting same neighbourhoods again, but a first implementation showed high memory demand of the cache and high runtime due to many cache checks. Concluding, *using the genetic heuristic* (in its current version) *is impractical with its significantly worse quality and solving time.*

6.3 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 having the top 5 degrees[18]; (b) $\hat{S}$=d, resources are degree-weighted distributed across all nodes – this was used by the previous evaluation; (c) $\hat{S}$=d^2, the degrees are squared, which amplifies the effect of having more resources at better

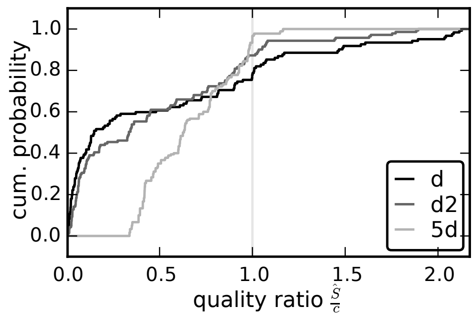


Figure 15: Comparison of quality factors for three $\hat{S}$ against $\hat{S}$=p

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)

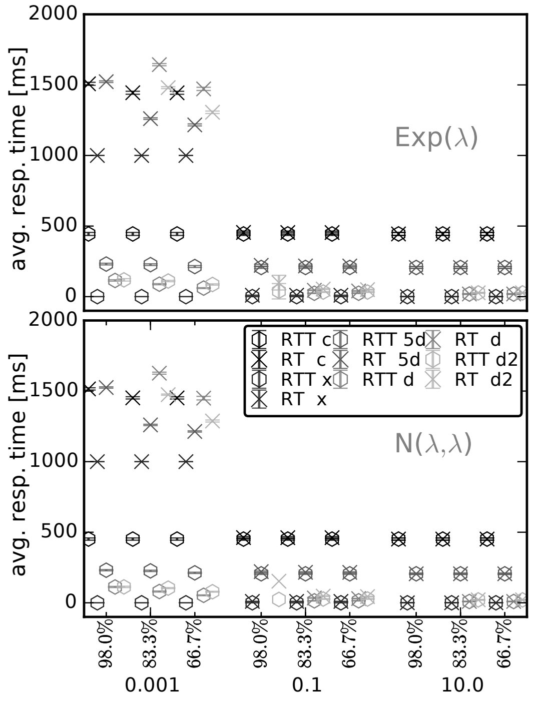
The previous factor values are now limited to fewer values: The demand distribution follows only two mathematical distributions, $\hat{D}\in\{\mathrm{N}_2, \mathrm{Exp}\}$. The resource limit $\hat{p}$ is restricted to low and high facility utilisation, $\hat{p}=\lceil 5a|N|\rceil$, a=0.51; 0.6; 0.75. The same topologies are considered as in the previous section (Table 5). 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 $\hat{\mu}, \hat{p}, \hat{D}$. All plots in Figure 14 compares average round trip times, response times, and queuing delay (the different between response times and round trip times) as a function of resource limit $\hat{p}$, service rate $\hat{\mu}$, demand distribution $\hat{D}$ for the five groups of resource distributions $\hat{S}$.

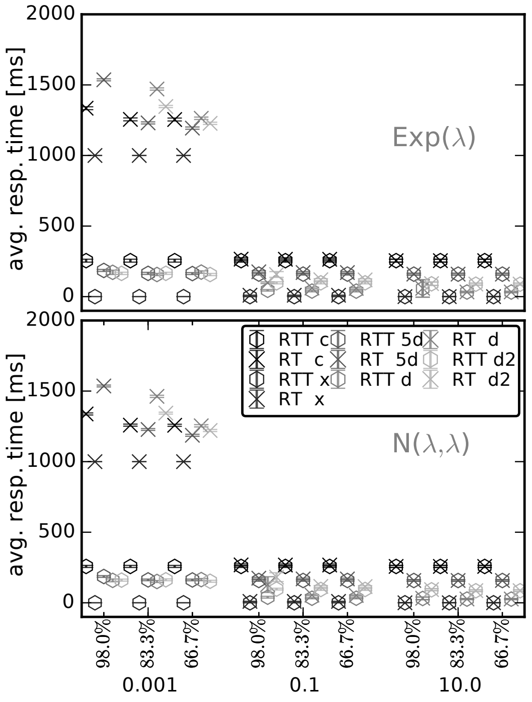
How much a distributed deployment reduces the response time compared to a single-site deployment, the quality of different resource distributions $\hat{S}$= d, d2, 5d are compared against $\hat{S}$=c by computing quality factors. Figure 15 shows ECDFs for quality factors for each resource distribution. For these resource distributions, 78%, 87%, and 97% of the configurations yield better response times than $\hat{S}$=c (the quality ratio being smaller than 1) and 61%, 61%, 35% have half or shorter response times (the quality ratios being smaller than 0.5). In conclusion, *deploying application across multiple sites can (at least) halve response times.*

17. 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.

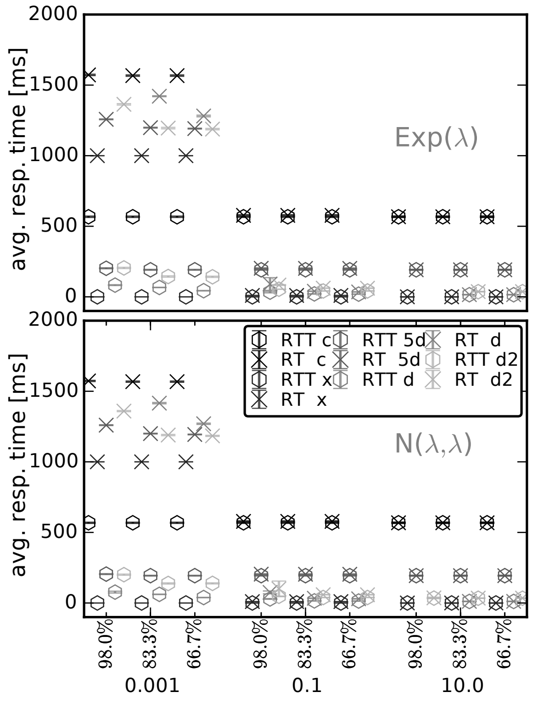
18. For all degree-based selections, the node ID was the tiebreaker.



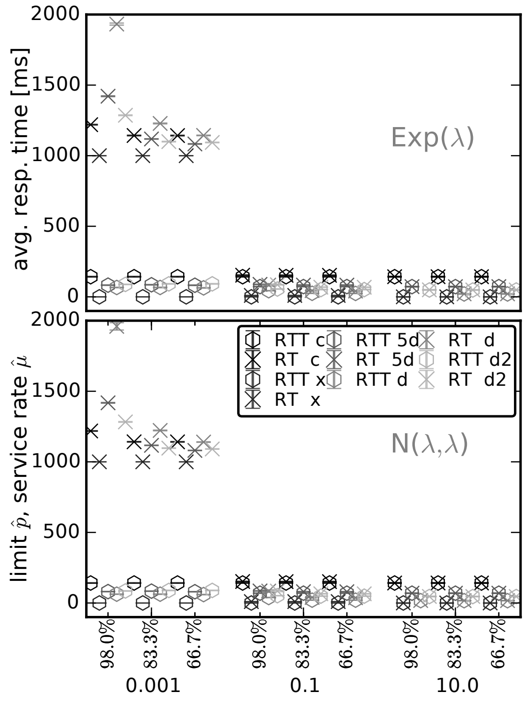
(a) $\hat{G} =$ `guil39`



(b) $\hat{G} =$ `Columbus`



(c) $\hat{G} =$ `Colt`



(d) $\hat{G} =$ `Cesnet20`

Figure 14: Comparison of RT and RTT as a function of $\hat{\mu}$ and $\hat{p}$ grouped by different $\hat{S}$.

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 6.3). 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 optimization 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 6.2.3).

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.

An advanced queuing system approximating a facility, data centre, or rack of compute nodes, (heterogeneous resources) could improve accuracy; but by how much is the question?

The following unproven thought could improve all approaches using the algorithm Alloc: When calling Alloc without initial allocation the result is equal to but most likely better as providing an initial allocation as proposed in this paper. Even if the provided initial allocation is optimal for the used problem approximation, the optimal allocation of Alloc might performs better. Similarly, calling Dealloc can be replaced by calling Alloc without initial allocation instead.

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] AIIMS. *Integer Linear Programming Tricks*, pages 83–85. 2009.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 6 2004.

[11] 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.

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

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms, third edition. 9 2009.

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

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

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

[17] M. E. Dyer and L. G. Proll. On the validity of marginal analysis for allocating servers in m/m/c queues. *Management Science*, 23:1019–1022, 1977.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] A. Harel. Convexity results for the erlang delay and loss formulae when the server utilization is held constant. *Operations Research*, 59:1420–1426, 11 2011.

[25] A. Harel and P. H. Zipkin. Strong convexity results for queueing systems. *Operations Research*, 35:405–418, 5 1987.

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

[27] 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.

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

[29] 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.

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

[31] M. Keller and H. Karl. Response-time-optimised service deployment (preprint). *tba*, 2015.

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

[33] 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.

[34] H. L. Lee and M. A. Cohen. A note on the convexity of performance measures of m/m/c queueing systems. *Journal of Applied Probability*, 20:920–923, 1983.

[35] 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.

[36] 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.

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

[38] 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.

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

[40] 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.

[41] 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.

[42] B. A. Pasternack and Z. Drezner. A note on calculating steady state results for an m/m/k queuing when the ratio of the arrival rate to the service rate is large. *Applied Mathematics and Decision Science (JAMDS)*, 2:201–203, 1998.

[43] 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.

[44] 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.

[45] 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.

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

[47] E. Tokgöz. Algorithms for mixed convexity and optimization of 2-smooth mixed convex functions. *International Journal of Pure and Applied Mathematics*, 57:103–110, 2009.

[48] E. Tokgoz and H. Kumin. A mixed integer convexity result with an application to an m/m/s queueing system. *International Journal of Open Problems in Computer Science and Mathematics*, 3:48–61, 2010.

[49] 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.

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

[51] 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.

[52] G. Zeng. Two common properties of the erlang-b function, erlang-c function, and engset blocking function. *Mathematical and Computer Modelling*, 37:1287–1296, 6 2003.

[53] 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.

8 Appendix

8.1 Non-convexity of Erlang-C

This section provides full equations for Theorem 2. The proof by contradiction assumes (67) holds. We obtain the derivatives with Maxima 5.26.0 and simplify the terms as much as possible. Numerically testing equations B for many tested values $0<a<k$ found B always $B<0$. By comparing the simplified terms, we found B and C to be equal in general. Then, equation (67) can be transformed into T (68). The equation T is for most values of v_1, v_2, a, k larger 0, $T\geq 0$. But for some values exemplified in Figure 16, T contradicts (67). As a result, in general function $\mathrm{EC}(a,k)$ is not convex in both parameters a, k.

Figure 16: Value table

a	k	v_1	v_2	T
0.1	1	1	1	-1.72
0.99	1	1	1	-0.47
6.27	10	1	1	$-7.94\cdot 10^{-5}$
6.27	10	0	1	$+0.06$
6.27	10	1	0	$+0.03$
9.99	10	1	1	-0.009

$$\nabla^2\mathrm{EC}(a,k)=H=\begin{bmatrix}A & B\\ C & D\end{bmatrix}\succeq 0$$

$$\Leftrightarrow\quad \forall v_1,v_2\in\mathbb{R}_{\geq 0}\colon v^T H v=v_1^2A+v_1v_2B+v_1v_2C+v_2^2D=T\geq 0 \tag{67}$$

$$T=v_1^2A+2v_1v_2B+v_2^2D\geq 0 \tag{68}$$

$$\begin{aligned}A &= \nabla_{kk}\mathrm{EC}(a,k)=\mathrm{EC}(a,k+2)-2\mathrm{EC}(a,k+1)+\mathrm{EC}(a,k)\\
&=\frac{a^{k+2}\,(k+2)}{(k-a+2)\,(k+2)!\left(S+\frac{a^{k+2}\,(k+2)}{(k-a+2)\,(k+2)!}+\frac{a^{k+1}}{(k+1)!}+\frac{a^k}{k!}\right)}\\
&\quad-\frac{2\,a^{k+1}\,(k+1)}{(k-a+1)\,(k+1)!\left(S+\frac{a^{k+1}\,(k+1)}{(k-a+1)\,(k+1)!}+\frac{a^k}{k!}\right)}+\frac{a^k\,k}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)},\end{aligned}$$

$$S=\sum_i^{k-1}\frac{a^i}{i!} \tag{69}$$

$$\begin{aligned}B,C &= \nabla_k\frac{\partial}{\partial a}\mathrm{EC}(a,k)=\frac{\partial}{\partial a}\mathrm{EC}(a,k+1)-\frac{\partial}{\partial a}f(a,k)=\frac{\partial}{\partial a}\nabla_k\mathrm{EC}(a,k)\frac{\partial}{\partial a}(\mathrm{EC}(a,k+1)-\mathrm{EC}(a,k))\\
&=\frac{a^k\,(k+1)^2}{(k-a+1)\,(k+1)!\left(S+\frac{a^{k+1}\,(k+1)}{(k-a+1)\,(k+1)!}+\frac{a^k}{k!}\right)}+\frac{a^{k+1}\,(k+1)}{(k-a+1)^2\,(k+1)!\left(S+\frac{a^{k+1}\,(k+1)}{(k-a+1)\,(k+1)!}+\frac{a^k}{k!}\right)}\\
&\quad-\frac{a^{k-1}\,k^2}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)}-\frac{a^k\,k}{(k-a)^2\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)}\\
&\quad-\frac{a^{k+1}\,(k+1)\left(S+\frac{a^k\,(k+1)^2}{(k-a+1)\,(k+1)!}+\frac{a^{k+1}\,(k+1)}{(k-a+1)^2\,(k+1)!}\right)}{(k-a+1)\,(k+1)!\left(S+\frac{a^{k+1}\,(k+1)}{(k-a+1)\,(k+1)!}+\frac{a^k}{k!}\right)^2}+\frac{a^k\,k\left(S+\frac{a^{k-1}\,k^2}{(k-a)\,k!}+\frac{a^k\,k}{(k-a)^2\,k!}-\frac{a^{k-1}}{(k-1)!}\right)}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)^2}\end{aligned} \tag{70}$$

$$\begin{aligned}D &= \frac{\partial^2}{\partial a\partial a}\mathrm{EC}(a,k)\\
&=\frac{2\,a^k\,k\left(S+\frac{a^{k-1}\,k^2}{(k-a)\,k!}+\frac{a^k\,k}{(k-a)^2\,k!}-\frac{a^{k-1}}{(k-1)!}\right)^2}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)^3}+\frac{a^{k-2}\,(k-1)\,k^2}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)}+\frac{2\,a^{k-1}\,k^2}{(k-a)^2\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)}\\
&\quad+\frac{2\,a^k\,k}{(k-a)^3\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)}-\frac{a^k\,k\left(a^2\,S+\frac{a^{k-2}\,(k-1)\,k^2}{(k-a)\,k!}+\frac{2\,a^{k-1}\,k^2}{(k-a)^2\,k!}+\frac{2\,a^k\,k}{(k-a)^3\,k!}-a^3-a^2\right)}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)^2}\\
&\quad-\frac{2\,a^{k-1}\,k^2\left(S+\frac{a^{k-1}\,k^2}{(k-a)\,k!}+\frac{a^k\,k}{(k-a)^2\,k!}-\frac{a^{k-1}}{(k-1)!}\right)}{(k-a)\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)^2}-\frac{2\,a^k\,k\left(S+\frac{a^{k-1}\,k^2}{(k-a)\,k!}+\frac{a^k\,k}{(k-a)^2\,k!}-\frac{a^{k-1}}{(k-1)!}\right)}{(k-a)^2\,k!\left(S+\frac{a^k\,k}{(k-a)\,k!}\right)^2}\end{aligned} \tag{71}$$

8.2 $\mathsf{N}_k(a)$ Derivative and Transformation

This sections provide the transformation of first derivative N of a with fix k into a term using recursive $\mathsf{V}(a,k)$ (56).

$$\begin{aligned}\frac{d}{da}\mathsf{N}_k(a)=&\frac{a^k\,k\,(k+1)}{(k-a)\left(\left(\sum_{i=0}^{k-1}\frac{a^i}{i!}\right)(k-a)\,k!+a^k\,k\right)}+1+\frac{a^{k+1}\,k}{(k-a)^2\left(\left(\sum_{i=0}^{k-1}\frac{a^i}{i!}\right)(k-a)\,k!+a^k\,k\right)}\\
&-\frac{a^{k+1}\,k\left(\left(\sum_{i=0}^{k-1}\frac{a^{i-1}\,i}{i!}\right)(k-a)\,k!-\left(\sum_{i=0}^{k-1}\frac{a^i}{i!}\right)k!+a^{k-1}\,k^2\right)}{(k-a)\left(\left(\sum_{i=0}^{k-1}\frac{a^i}{i!}\right)(k-a)\,k!+a^k\,k\right)^2}\end{aligned} \tag{72}$$

$$= \frac{k\ (k+1)}{(k-a)\left((k-a)\sum_{i=0}^{k-1}\frac{a^i\,k!}{i!\,a^k}+k\right)} + 1 + \frac{a\,k}{(k-a)^2\left((k-a)\sum_{i=0}^{k-1}\frac{a^i\,k!}{i!\,a^k}+k\right)}$$

$$- \frac{a\,k\left(\sum_{i=0}^{k-1}\frac{a^{(i-k)}\ i\ (k-1)!}{i!}(k-a)\frac{k}{a}) + \sum_{i=0}^{k-1}\frac{a^i\,k!}{i!\,a^k} + \frac{k^2}{a}\right)}{(k-a)\ \left(\sum_{i=0}^{k-1}\frac{a^i\,k!}{i!\,a^k}(k-a)+k\right)^2}$$

$$= \frac{k\ (k+1)}{(k-a)^2\,\mathrm{V}\,(a,k)+k\ (k-a)} + 1 + \frac{a\,k}{(k-a)^3\,\mathrm{V}\,(a,k)+(k-a)^2\,k}$$

$$- \frac{\mathrm{V}\,(a,k-1)\ k^2\ (k-a) - \mathrm{V}\,(a,k)\ a\,k + k^3}{(k-a)\ (\mathrm{V}\,(a,k)\ (k-a)+k)^2} \quad (73)$$

$$V(a,k-1) = \sum_{i=0}^{k-2}\frac{(k-1)!}{i!}a^{i-k+1} = \sum_{i=1}^{k-1}\frac{(k-1)!}{i!}\,i\,a^{i-k} \quad (74)$$