Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada
(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.
- Approximating the expectations: rough-cut formulas versus simulation. Replacing the exact problem by a sample problem (with fixed ω).

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.
- Approximating the expectations: rough-cut formulas versus simulation. Replacing the exact problem by a sample problem (with fixed ω).
- A general solution algorithm based on cut generation. Concavity issues.

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.
- Approximating the expectations: rough-cut formulas versus simulation. Replacing the exact problem by a sample problem (with fixed ω).
- A general solution algorithm based on cut generation. Concavity issues.
- Numerical illustrations.

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.
- Approximating the expectations: rough-cut formulas versus simulation. Replacing the exact problem by a sample problem (with fixed ω).
- A general solution algorithm based on cut generation. Concavity issues.
- Numerical illustrations.
- The blend case.

Staffing and Scheduling in Multiskill and Blend Call Centers

Pierre L'Ecuyer
GERAD and DIRO, Université de Montréal, Canada

(Joint work with Tolga Cezik, Éric Buist, and Thanos Avramidis)

- The multiskill setting: staffing and scheduling problems and their formulation as integer programs.
- Approximating the expectations: rough-cut formulas versus simulation. Replacing the exact problem by a sample problem (with fixed ω).
- A general solution algorithm based on cut generation. Concavity issues.
- Numerical illustrations.
- The blend case.
- Ongoing and future work. Conclusion.

Multiskill Agents and Skill-Based Routing

K types of calls.
I types of agents (or skill groups).

Multiskill Agents and Skill-Based Routing

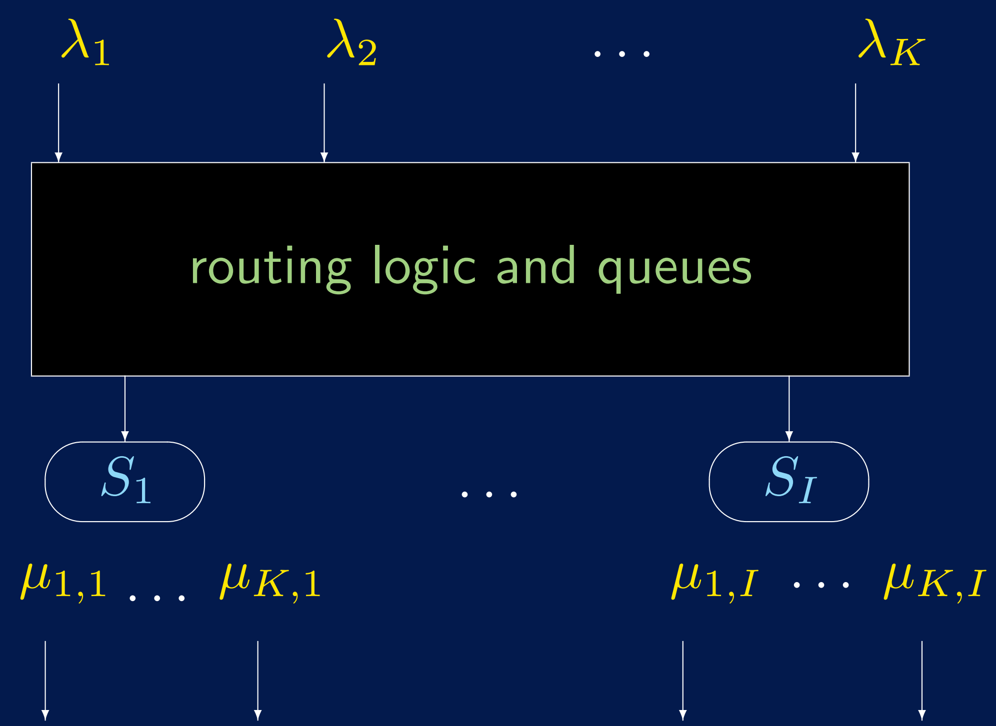
K types of calls.
I types of agents (or skill groups).

Each agent type has the skills to handle certain call types.
Some agents may be better than others for a given call type.

Multiskill Agents and Skill-Based Routing

K types of calls.
I types of agents (or skill groups).

Each agent type has the skills to handle certain call types.
Some agents may be better than others for a given call type.



Arrival processes can be nonstationary, doubly stochastic, dependent, etc. Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc. Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc. Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc. Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

If each agent has a single skill: K single queues in parallel.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc.
Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

If each agent has a single skill: K single queues in parallel.

If each agent has all skills: one single queue.
More efficient (less wait, fewer abandonments), but more costly.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc.
Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

If each agent has a single skill: K single queues in parallel.

If each agent has all skills: one single queue.
More efficient (less wait, fewer abandonments), but more costly.

Typically, each additional skill adds 10–20% to the cost of an agent.

Arrival processes can be nonstationary, doubly stochastic, dependent, etc.
Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

If each agent has a single skill: K single queues in parallel.

If each agent has all skills: one single queue.
More efficient (less wait, fewer abandonments), but more costly.

Typically, each additional skill adds 10–20% to the cost of an agent.

One or two skills per agent often gives a performance almost as good as all skills for all agents (e.g., Wallace and Whitt 2004).

Arrival processes can be nonstationary, doubly stochastic, dependent, etc.
Examples for single type: nonstationary Poisson with random inflation factor each day; total number gamma with multinomial distribution across periods; NORTA distribution.

Abandonments: callers may abandon after a random patience time.

Callbacks: Not considered here.

If each agent has a single skill: K single queues in parallel.

If each agent has all skills: one single queue.
More efficient (less wait, fewer abandonments), but more costly.

Typically, each additional skill adds 10–20% to the cost of an agent.

One or two skills per agent often gives a performance almost as good as all skills for all agents (e.g., Wallace and Whitt 2004).

Performance measures and constraints:
Total cost of agents, quality of service (QoS) (e.g., fraction of calls answered within 20 seconds, abandonment ratio, etc.).

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Static routing: Each call type has an ordered list of agent types.
If all are busy, the call joins a queue.

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Static routing: Each call type has an ordered list of agent types.
If all are busy, the call joins a queue.
Could be one queue per call type (usually), or one queue per agent type (not so good), or a mixture.

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Static routing: Each call type has an ordered list of agent types.
If all are busy, the call joins a queue.
Could be one queue per call type (usually), or one queue per agent type (not so good), or a mixture.

Each agent type also has an ordered list of queues to dig from when becoming available (priorities)

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Static routing: Each call type has an ordered list of agent types.
If all are busy, the call joins a queue.
Could be one queue per call type (usually), or one queue per agent type (not so good), or a mixture.

Each agent type also has an ordered list of queues to dig from when becoming available (priorities)

Mixed: Could use thresholds (state-dependent), or weighted FIFO rule, etc.

Skill-based routing (SBR) strategies:

Dynamic routing: Decision may depend on the entire state of the system.
An optimal policy is generally too complicated and hard to implement.

Static routing: Each call type has an ordered list of agent types.
If all are busy, the call joins a queue.
Could be one queue per call type (usually), or one queue per agent type (not so good), or a mixture.

Each agent type also has an ordered list of queues to dig from when becoming available (priorities)

Mixed: Could use thresholds (state-dependent), or weighted FIFO rule, etc.

Here, we assume that the routing rules are fixed; we do not optimize them.

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

These constraints can be per customer type, per period, or aggregated.

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

These constraints can be per customer type, per period, or aggregated.

Caveat: it is usually not possible to match exactly the optimal staffing by scheduling a set of agents whose working shifts are admissible (i.e., satisfy the constraints determined by Union rules, etc.).

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

These constraints can be per customer type, per period, or aggregated.

Caveat: it is usually not possible to match exactly the optimal staffing by scheduling a set of agents whose working shifts are admissible (i.e., satisfy the constraints determined by Union rules, etc.).

Scheduling problem: Determine a set of agents, each with its working shift for the day, so that the QoS constraints are met, at minimal cost.

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

These constraints can be per customer type, per period, or aggregated.

Caveat: it is usually not possible to match exactly the optimal staffing by scheduling a set of agents whose working shifts are admissible (i.e., satisfy the constraints determined by Union rules, etc.).

Scheduling problem: Determine a set of agents, each with its working shift for the day, so that the QoS constraints are met, at minimal cost.

Scheduling and rostering problem: In practice, we do not have an infinite supply for each agent type!
For a given set of agents and a given set of admissible shifts, assign a shift to each agent, to meet the QoS constraints at minimal cost.

Staffing problem: Divide the day into periods (e.g, half hours) and determine how many agents of each type to have in the center for each period in order to meet the QoS constraints, at minimal cost.

These constraints can be per customer type, per period, or aggregated.

Caveat: it is usually not possible to match exactly the optimal staffing by scheduling a set of agents whose working shifts are admissible (i.e., satisfy the constraints determined by Union rules, etc.).

Scheduling problem: Determine a set of agents, each with its working shift for the day, so that the QoS constraints are met, at minimal cost.

Scheduling and rostering problem: In practice, we do not have an infinite supply for each agent type!
For a given set of agents and a given set of admissible shifts, assign a shift to each agent, to meet the QoS constraints at minimal cost.
We do not address this one here.

Integer Programming Formulation

Call types	$k = 1, \ldots, K$;
Agent types (or skill groups)	$i = 1, \ldots, I$;
Periods	$p = 1, \ldots, P$;
Shift types	$q = 1, \ldots, Q$.

Integer Programming Formulation

Call types	$k = 1, \dots, K$;
Agent types (or skill groups)	$i = 1, \dots, I$;
Periods	$p = 1, \dots, P$;
Shift types	$q = 1, \dots, Q$.

The shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

Integer Programming Formulation

Call types	$k = 1, \ldots, K$;
Agent types (or skill groups)	$i = 1, \ldots, I$;
Periods	$p = 1, \ldots, P$;
Shift types	$q = 1, \ldots, Q$.

The shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

Costs: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})$ where
$c_{i,q}$ = cost of an agent of type i having shift q.

Integer Programming Formulation

Call types	$k = 1, \ldots, K$;
Agent types (or skill groups)	$i = 1, \ldots, I$;
Periods	$p = 1, \ldots, P$;
Shift types	$q = 1, \ldots, Q$.

The shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

Costs: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})$ where
$c_{i,q}$ = cost of an agent of type i having shift q.

Decision variables: $\mathbf{x} = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})$ where
$x_{i,q}$ = number of agents of type i having shift q.

Integer Programming Formulation

Call types	$k = 1, \ldots, K$;
Agent types (or skill groups)	$i = 1, \ldots, I$;
Periods	$p = 1, \ldots, P$;
Shift types	$q = 1, \ldots, Q$.

The shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

Costs: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})$ where
$c_{i,q}$ = cost of an agent of type i having shift q.

Decision variables: $\mathbf{x} = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})$ where
$x_{i,q}$ = number of agents of type i having shift q.

Auxiliary variables: $\mathbf{y} = (y_{1,1}, \ldots, y_{1,P}, \ldots, y_{I,1}, \ldots, y_{I,P})$ where
$y_{i,p}$ = number of agents of type i in period p.

Integer Programming Formulation

Call types	$k = 1, \dots, K$;
Agent types (or skill groups)	$i = 1, \dots, I$;
Periods	$p = 1, \dots, P$;
Shift types	$q = 1, \dots, Q$.

The shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

Costs: $\mathbf{c} = (c_{1,1}, \dots, c_{1,Q}, \dots, c_{I,1}, \dots, c_{I,Q})$ where
$c_{i,q}$ = cost of an agent of type i having shift q.

Decision variables: $\mathbf{x} = (x_{1,1}, \dots, x_{1,Q}, \dots, x_{I,1}, \dots, x_{I,Q})$ where
$x_{i,q}$ = number of agents of type i having shift q.

Auxiliary variables: $\mathbf{y} = (y_{1,1}, \dots, y_{1,P}, \dots, y_{I,1}, \dots, y_{I,P})$ where
$y_{i,p}$ = number of agents of type i in period p.

We have $\mathbf{y} = \mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is block diagonal with i blocks $\tilde{\mathbf{A}}$, and element (p, q) of $\tilde{\mathbf{A}}$ is 1 if shift q covers period p, 0 otherwise.

$g_{k,p}(\mathbf{y})$ = long-run QoS for call type k in period p.
E.g., fraction of calls answered within 20 seconds, or $s_{k,p}$ seconds:

$$g_{k,p}(\mathbf{y}) = \frac{E[\text{num. of calls answered within the limit in period } p]}{E[\text{num. of calls in period } p]}.$$

$g_{k,p}(\mathbf{y})$ = long-run QoS for call type k in period p.
E.g., fraction of calls answered within 20 seconds, or $s_{k,p}$ seconds:

$$g_{k,p}(\mathbf{y}) = \frac{E[\text{num. of calls answered within the limit in period } p]}{E[\text{num. of calls in period } p]}.$$

$g_p(\mathbf{y})$ = aggregated long-run QoS over period p.
$g_k(\mathbf{y})$ = aggregated long-run QoS for call type k.
$g(\mathbf{y})$ = aggregated long-run QoS overall.

$g_{k,p}(\mathbf{y})$ = long-run QoS for call type k in period p.
E.g., fraction of calls answered within 20 seconds, or $s_{k,p}$ seconds:

$$g_{k,p}(\mathbf{y}) = \frac{E[\text{num. of calls answered within the limit in period } p]}{E[\text{num. of calls in period } p]}.$$

$g_p(\mathbf{y})$ = aggregated long-run QoS over period p.
$g_k(\mathbf{y})$ = aggregated long-run QoS for call type k.
$g(\mathbf{y})$ = aggregated long-run QoS overall.

These functions $g_\bullet$ (ratios of expectations) are unknown, but they can be either

- approximated via simplified queueing models, or
- estimated by simulation.

$g_{k,p}(\mathbf{y})$ = long-run QoS for call type k in period p.
E.g., fraction of calls answered within 20 seconds, or $s_{k,p}$ seconds:

$$g_{k,p}(\mathbf{y}) = \frac{E[\text{num. of calls answered within the limit in period } p]}{E[\text{num. of calls in period } p]}.$$

$g_p(\mathbf{y})$ = aggregated long-run QoS over period p.
$g_k(\mathbf{y})$ = aggregated long-run QoS for call type k.
$g(\mathbf{y})$ = aggregated long-run QoS overall.

These functions $g_\bullet$ (ratios of expectations) are unknown, but they can be either

- approximated via simplified queueing models, or
- estimated by simulation.
  We are developing a Java library for the simulation of call centers.

Scheduling problem

$$
\begin{array}{ll}
\min & \mathbf{c}^{\mathsf{t}}\mathbf{x} = \sum_{i=1}^{I}\sum_{q=1}^{Q} c_{i,q}x_{i,q} \\
\text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& g_{k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k,p, \\
& g_{p}(\mathbf{y}) \geq l_{p} \qquad \text{for all } p, \\
& g_{k}(\mathbf{y}) \geq l_{k} \qquad \text{for all } k, \\
& g(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \ \text{and integer.}
\end{array}
$$

Staffing Problem

Relaxation: forget about the admissibility of schedules; just assume that any staffing is admissible.

Staffing Problem

Relaxation: forget about the admissibility of schedules; just assume that any staffing is admissible.

Costs: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,P}, \ldots, c_{I,1}, \ldots, c_{I,P})$ where
$c_{i,p}$ = cost of an agent of type i in period p.

Staffing Problem

Relaxation: forget about the admissibility of schedules; just assume that any staffing is admissible.

Costs: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,P}, \ldots, c_{I,1}, \ldots, c_{I,P})$ where
$c_{i,p}$ = cost of an agent of type i in period p.

$$
\begin{aligned}
\min \quad & \mathbf{c}^{\mathsf{t}}\mathbf{y} = \textstyle\sum_{i=1}^{I} \sum_{p=1}^{P} c_{i,p} y_{i,p} \\
\text{subject to} \quad & g_{k,p}(\mathbf{y}) \ge l_{k,p} \qquad \text{for all } k, p, \\
& g_p(\mathbf{y}) \ge l_p \qquad \text{for all } p, \\
& g_k(\mathbf{y}) \ge l_k \qquad \text{for all } k, \\
& g(\mathbf{y}) \ge l, \\
& \mathbf{y} \ge 0, \text{ and integer.}
\end{aligned}
$$

Single period, steady-state approximation

Take one period at a time and assume that the system is in steady-state.

Single period, steady-state approximation

Take one period at a time and assume that the system is in steady-state.

$\mathbf{c} = (c_1, \ldots, c_I)$ where $c_i =$ cost of an agent of type i.

Single period, steady-state approximation

Take one period at a time and assume that the system is in steady-state.

$\mathbf{c} = (c_1, \ldots, c_I)$ where c_i = cost of an agent of type i.

$\mathbf{y} = (y_1, \ldots, y_I)$ where y_i = number of agents of type i.

Single period, steady-state approximation

Take one period at a time and assume that the system is in steady-state.

$\mathbf{c} = (c_1, \ldots, c_I)$ where c_i = cost of an agent of type i.

$\mathbf{y} = (y_1, \ldots, y_I)$ where y_i = number of agents of type i.

$$
\begin{array}{lll}
\min & \mathbf{c}^{\mathrm{t}}\mathbf{y} = \sum_{i=1}^{I} c_i y_i & \\
\text{subject to} & g_k(\mathbf{y}) \geq l_k & \text{for all } k, \\
& g(\mathbf{y}) \geq l, & \\
& \mathbf{y} \geq 0, \ \text{and integer.} &
\end{array}
$$

Scheduling by a two-step method

First step: solve the staffing problem (easier).
Let $\mathbf{y}^*$ be an optimal solution.

Scheduling by a two-step method

First step: solve the staffing problem (easier).
Let $\mathbf{y}^*$ be an optimal solution.

Second step: find an admissible schedule that covers the staffing $\mathbf{y}^*$, by solving:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{y}^*, \\ & \mathbf{x} \geq 0, \text{ and integer.} \end{array}$$

Scheduling by a two-step method

First step: solve the staffing problem (easier).
Let $\mathbf{y}^*$ be an optimal solution.

Second step: find an admissible schedule that covers the staffing $\mathbf{y}^*$, by solving:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{y}^*, \\ & \mathbf{x} \geq 0, \text{ and integer.} \end{array}$$

This does not give an optimal solution to the original scheduling problem.
And the gap could be quite significant.

Steady-state approximations

Koole and Talim (2000): static overflow routing, loss system.

Steady-state approximations

Koole and Talim (2000): static overflow routing, loss system.

Call type k has a Poisson arrival process, with rate λ_k, and ordered list of agent types to look for.

Steady-state approximations

Koole and Talim (2000): static overflow routing, loss system.

Call type k has a Poisson arrival process, with rate λ_k, and ordered list of agent types to look for.

Other simplifying assumptions:

1. If all these agent types are busy, the call is lost.

Steady-state approximations

Koole and Talim (2000): static overflow routing, loss system.

Call type k has a Poisson arrival process, with rate λ_k, and ordered list of agent types to look for.

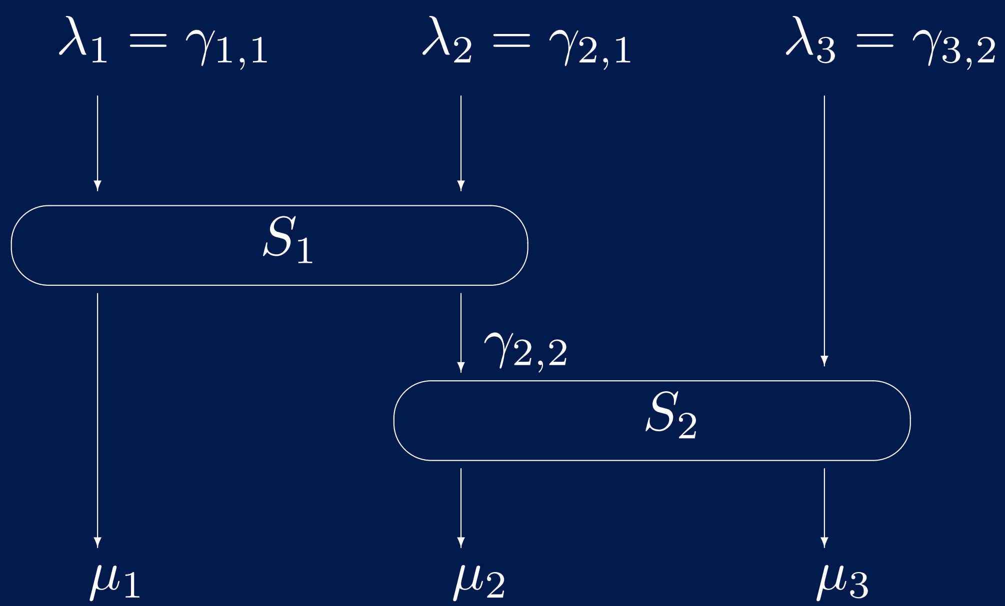
Other simplifying assumptions:
1. If all these agent types are busy, the call is lost.
2. The net arrival process at each agent type is Poisson.
3. Quality of service $g(\mathbf{y})$ is approximated via a relation between the loss (or blocking) probability in an Erlang B system and the delay probability in an Erlang C system (valid for a single skill).

Steady-state approximations

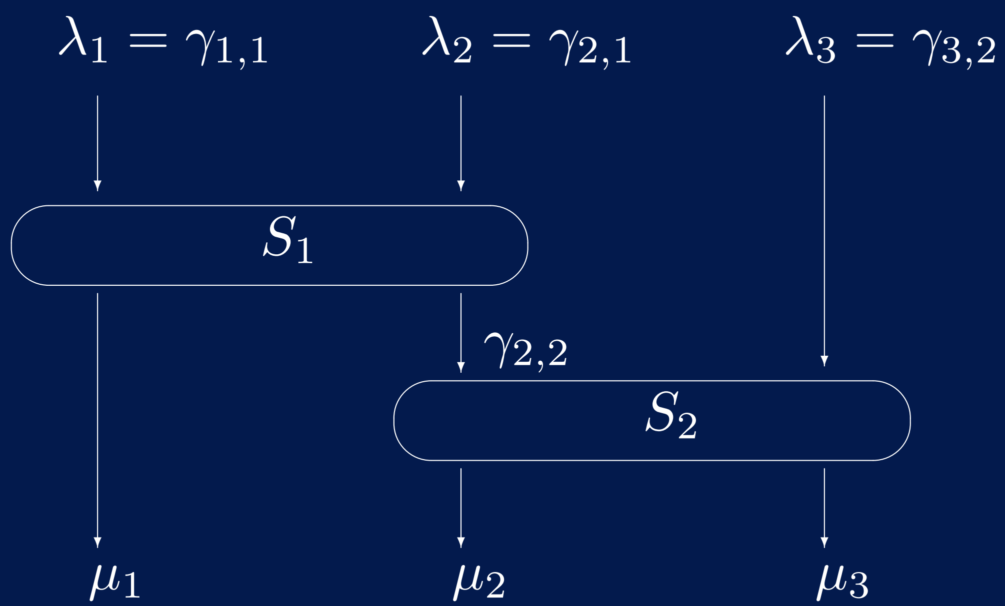
Koole and Talim (2000): static overflow routing, loss system.

Call type k has a Poisson arrival process, with rate λ_k, and ordered list of agent types to look for.

Other simplifying assumptions:
1. If all these agent types are busy, the call is lost.
2. The net arrival process at each agent type is Poisson.
3. Quality of service $g(\mathbf{y})$ is approximated via a relation between the loss (or blocking) probability in an Erlang B system and the delay probability in an Erlang C system (valid for a single skill).

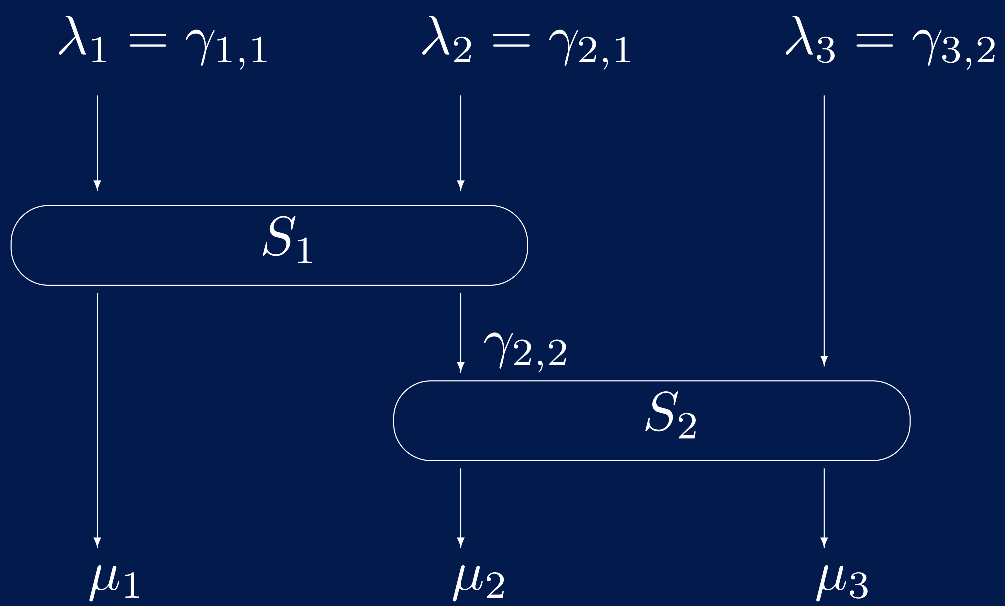


$\gamma_{k,i}$ = net arrival rate of call type k at skill group i;



$\gamma_{k,i}$ = net arrival rate of call type k at skill group i;

$\mu_{k,i}$ = service rate of call type k at skill group i;



$\gamma_{k,i}$ = net arrival rate of call type k at skill group i;

$\mu_{k,i}$ = service rate of call type k at skill group i;

$a_i = \sum_k \gamma_{k,i}/\mu_{k,i}$ = load at skill group i;

System of nonlinear equations:

$$\begin{aligned}
\gamma_{k,\mathrm{first}(k)} &= \lambda_k, \\
\gamma_{k,\mathrm{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \mathrm{succ}(k,i) \text{ exists}, \\
B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad \text{(blocking probability for skill group } i\text{)}.
\end{aligned}$$

System of nonlinear equations:

$$\begin{aligned}\gamma_{k,\mathrm{first}(k)} &= \lambda_k, \\ \gamma_{k,\mathrm{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \mathrm{succ}(k,i) \text{ exists}, \\ B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad \text{(blocking probability for skill group } i\text{).}\end{aligned}$$

$\gamma_{k,\mathrm{last}(k)} B(y_{\mathrm{last}(k)}, a_{\mathrm{last}(k)}) =$ loss rate for call type k.

System of nonlinear equations:

$$
\begin{aligned}
\gamma_{k,\mathrm{first}(k)} &= \lambda_k, \\
\gamma_{k,\mathrm{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \mathrm{succ}(k,i) \text{ exists,} \\
B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad \text{(blocking probability for skill group } i\text{).}
\end{aligned}
$$

$\gamma_{k,\mathrm{last}(k)} B(y_{\mathrm{last}(k)}, a_{\mathrm{last}(k)}) =$ loss rate for call type k.

Delay probability (one more heuritic): Suppose aggregated arrival rate λ, service rate μ, load a, and y servers.

System of nonlinear equations:

$$\begin{aligned}\gamma_{k,\text{first}(k)} &= \lambda_k, \\ \gamma_{k,\text{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \mathrm{succ}(k,i) \text{ exists}, \\ B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad (\text{blocking probability for skill group } i).\end{aligned}$$

$\gamma_{k,\text{last}(k)} B(y_{\text{last}(k)}, a_{\text{last}(k)})$ = loss rate for call type k.

Delay probability (one more heuritic): Suppose aggregated arrival rate λ, service rate μ, load a, and y servers.

$P[\text{delay} > 0] \approx yB(y,a)/[y - a(1 - B(y,a))]$.

System of nonlinear equations:

$$\begin{aligned}\gamma_{k,\mathrm{first}(k)} &= \lambda_k, \\ \gamma_{k,\mathrm{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \mathrm{succ}(k,i) \text{ exists}, \\ B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad \text{(blocking probability for skill group } i\text{)}.\end{aligned}$$

$\gamma_{k,\mathrm{last}(k)} B(y_{\mathrm{last}(k)}, a_{\mathrm{last}(k)}) =$ loss rate for call type k.

Delay probability (one more heuritic): Suppose aggregated arrival rate λ, service rate μ, load a, and y servers.

$P[\text{delay} > 0] \approx yB(y,a)/[y - a(1 - B(y,a))]$.

To approximate $P[\text{delay} > \tau]$, use $P[\text{delay} > \tau | \text{delay} > 0] P[\text{delay} > 0]$, with exponential conditional distribution as in $M/M/s$ queue.

System of nonlinear equations:

$$\begin{aligned}\gamma_{k,\text{first}(k)} &= \lambda_k, \\ \gamma_{k,\text{succ}(k,i)} &= \gamma_{k,i} B(y_i, a_i) \qquad \text{whenever } \text{succ}(k,i) \text{ exists}, \\ B(y_i, a_i) &= \frac{a_i^{y_i}/y_i!}{1 + a_i + \cdots + a_i^{y_i}/y_i!} \qquad \text{(blocking probability for skill group } i).\end{aligned}$$

$\gamma_{k,\text{last}(k)} B(y_{\text{last}(k)}, a_{\text{last}(k)}) =$ loss rate for call type k.

Delay probability (one more heuritic): Suppose aggregated arrival rate λ, service rate μ, load a, and y servers.

$P[\text{delay} > 0] \approx yB(y,a)/[y - a(1 - B(y,a))]$.

To approximate $P[\text{delay} > \tau]$, use $P[\text{delay} > \tau | \text{delay} > 0] P[\text{delay} > 0]$, with exponential conditional distribution as in $M/M/s$ queue.

There are also better approximations for the loss rate.

Estimation and optimization via simulation

We simulate n independent operating days (could also be weeks, etc.) of the center, to estimate the functions $g_\bullet$.

Estimation and optimization via simulation

We simulate n independent operating days (could also be weeks, etc.) of the center, to estimate the functions $g_\bullet$.

Let ω represent the source of randomness, i.e., the sequence of independent uniform r.v.'s underlying the entire simulation (n runs).

Estimation and optimization via simulation

We simulate n independent operating days (could also be weeks, etc.) of the center, to estimate the functions $g_\bullet$.

Let ω represent the source of randomness, i.e., the sequence of independent uniform r.v.'s underlying the entire simulation (n runs).

For a ω, the empirical QoS's over the n simulation runs are:

$G_{n,k,p}(\mathbf{y}, \omega)$ for call type k in period p;
$G_{n,p}(\mathbf{y}, \omega)$ aggregated over period p;
$G_{n,k}(\mathbf{y}, \omega)$ aggregated for call type k;
$G_n(\mathbf{y}, \omega)$ aggregated overall.

Estimation and optimization via simulation

We simulate n independent operating days (could also be weeks, etc.) of the center, to estimate the functions $g_\bullet$.

Let ω represent the source of randomness, i.e., the sequence of independent uniform r.v.'s underlying the entire simulation (n runs).

For a ω, the empirical QoS's over the n simulation runs are:

$G_{n,k,p}(\mathbf{y}, \omega)$ for call type k in period p;
$G_{n,p}(\mathbf{y}, \omega)$ aggregated over period p;
$G_{n,k}(\mathbf{y}, \omega)$ aggregated for call type k;
$G_n(\mathbf{y}, \omega)$ aggregated overall.

For a fixed ω, these are deterministic functions of $\mathbf{y}$.

Estimation and optimization via simulation

We simulate n independent operating days (could also be weeks, etc.) of the center, to estimate the functions $g_\bullet$.

Let ω represent the source of randomness, i.e., the sequence of independent uniform r.v.'s underlying the entire simulation (n runs).

For a ω, the empirical QoS's over the n simulation runs are:

$G_{n,k,p}(\mathbf{y},\omega)$ for call type k in period p;
$G_{n,p}(\mathbf{y},\omega)$ aggregated over period p;
$G_{n,k}(\mathbf{y},\omega)$ aggregated for call type k;
$G_{n}(\mathbf{y},\omega)$ aggregated overall.

For a fixed ω, these are deterministic functions of $\mathbf{y}$.

To compute them at different values of $\mathbf{y}$, we simply use simulation with common random numbers.

Empirical scheduling optimization problem (sample version of the problem):

$$\begin{aligned}
\min \quad & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \textstyle\sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& G_{n,k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k, p, \\
& G_{n,p}(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\
& G_{n,k}(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\
& G_n(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \text{ and integer.}
\end{aligned}$$

Empirical scheduling optimization problem (sample version of the problem):

$$\begin{aligned}
\min \quad & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \textstyle\sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& G_{n,k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k, p, \\
& G_{n,p}(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\
& G_{n,k}(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\
& G_n(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \text{ and integer.}
\end{aligned}$$

Similar formulation for the staffing problem.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

For simplicity, assume a finite number of solutions.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

For simplicity, assume a finite number of solutions.

Let $\mathcal{Y}^*$ be the set of optimal solutions of the exact problem.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

For simplicity, assume a finite number of solutions.

Let $\mathcal{Y}^*$ be the set of optimal solutions of the exact problem.

Suppose no QoS constraint is satisfied exactly for these solutions.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

For simplicity, assume a finite number of solutions.

Let $\mathcal{Y}^*$ be the set of optimal solutions of the exact problem.

Suppose no QoS constraint is satisfied exactly for these solutions.

Let $\mathcal{Y}_n^*$ be the set of optimal solutions of the sample problem.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \longrightarrow \infty$.

For simplicity, assume a finite number of solutions.

Let $\mathcal{Y}^*$ be the set of optimal solutions of the exact problem.

Suppose no QoS constraint is satisfied exactly for these solutions.

Let $\mathcal{Y}_n^*$ be the set of optimal solutions of the sample problem.

Theorem (follows from Vogel 1994; see also Atlason et al. 2004).
(a) W.p.1, there is an integer $N_0 < \infty$ such that for all $n \geq N_0$, $\mathcal{Y}_n^* = \mathcal{Y}^*$.

We know that $G_{n,k,p}(\mathbf{y})$ converges to $g_{k,p}(\mathbf{y})$ for each (k,p) and each $\mathbf{y}$. Thus, the empirical problem converges to the exact problem when $n \to \infty$.

For simplicity, assume a finite number of solutions.

Let $\mathcal{Y}^*$ be the set of optimal solutions of the exact problem.

Suppose no QoS constraint is satisfied exactly for these solutions.

Let $\mathcal{Y}_n^*$ be the set of optimal solutions of the sample problem.

Theorem (follows from Vogel 1994; see also Atlason et al. 2004).
(a) W.p.1, there is an integer $N_0 < \infty$ such that for all $n \geq N_0$, $\mathcal{Y}_n^* = \mathcal{Y}^*$.
(b) Under mild assumptions, there exist positive numbers α and β such that

$$P[\mathcal{Y}_n^* = \mathcal{Y}^*] \geq 1 - \alpha e^{-\beta n}.$$

Solving the optimization problems

We solve the staffing and scheduling problems via IP with cut generation, as suggested by Atlason, Epelman, and Henderson (2004).

Solving the optimization problems

We solve the staffing and scheduling problems via IP with cut generation, as suggested by Atlason, Epelman, and Henderson (2004).

In their paper, they consider a single call type and a single agent type. They solve an example with 5 periods and 6 shift types.

Solving the optimization problems

We solve the staffing and scheduling problems via IP with cut generation, as suggested by Atlason, Epelman, and Henderson (2004).

In their paper, they consider a single call type and a single agent type.
They solve an example with 5 periods and 6 shift types.

Our aim is to:
— extend the methodology to the blend and multiskill settings;
— explore the difficulties and make it work for larger problems;
— compare approximation formulas with detailed simulation.

Solving the optimization problems

We solve the staffing and scheduling problems via IP with cut generation, as suggested by Atlason, Epelman, and Henderson (2004).

In their paper, they consider a single call type and a single agent type.
They solve an example with 5 periods and 6 shift types.

Our aim is to:
— extend the methodology to the blend and multiskill settings;
— explore the difficulties and make it work for larger problems;
— compare approximation formulas with detailed simulation.

We explain the algorithm for the exact functions $g_\bullet$, but it works the same way if they are replaced by approximations or estimation $G_{n,\bullet}$.

Idea: relax QoS constraints and solve the IP problem.

Idea: relax QoS constraints and solve the IP problem.

Let $\bar{\mathbf{y}}$ denote the current solution (optimal for the relaxed problem).

Idea: relax QoS constraints and solve the IP problem.

Let $\bar{\mathbf{y}}$ denote the current solution (optimal for the relaxed problem).
If all constraints are satisfied at $\bar{\mathbf{y}}$, eureka!

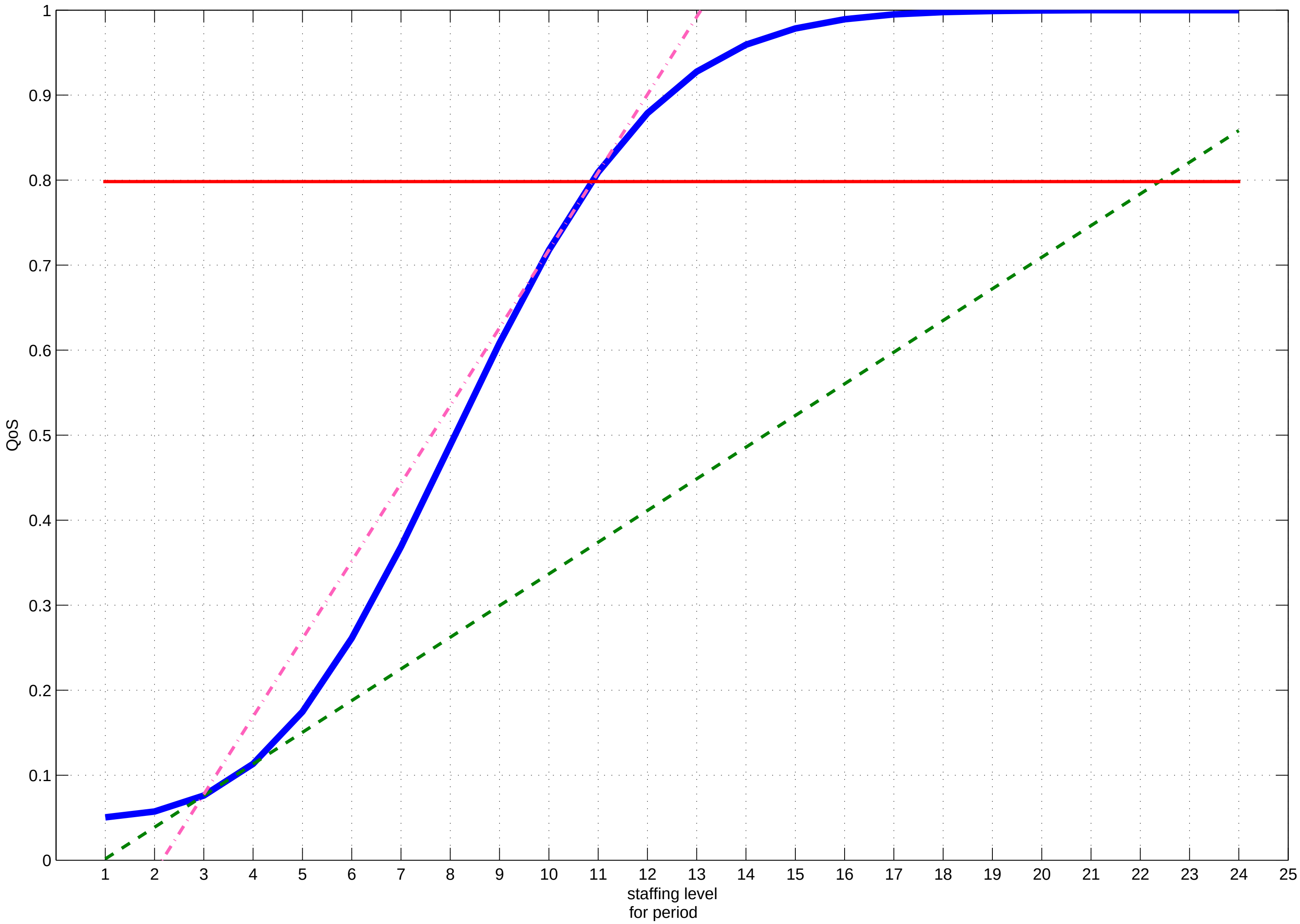
Idea: relax QoS constraints and solve the IP problem.

Let $\bar{\mathbf{y}}$ denote the current solution (optimal for the relaxed problem).
If all constraints are satisfied at $\bar{\mathbf{y}}$, eureka!

Otherwise, for each violated constraint, say $g(\bar{\mathbf{y}}) < l$, suppose that g is concave in $\mathbf{y}$ for $\mathbf{y} \geq \bar{\mathbf{y}}$. Then

$$g(\mathbf{y}) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}})$$

where $\bar{\mathbf{q}}$ is any subgradient of g at $\bar{\mathbf{y}}$.


QoS
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
staffing level
for period

Otherwise, for each violated constraint, say $g(\bar{\mathbf{y}}) < l$, suppose that g is concave in $\mathbf{y}$ for $\mathbf{y} \geq \bar{\mathbf{y}}$. Then

$$g(\mathbf{y}) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}})$$

where $\bar{\mathbf{q}}$ is any subgradient of g at $\bar{\mathbf{y}}$.

But we want $g(\mathbf{y}) \geq l$, so we must have

$$l \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}}),$$

i.e.,

$$\bar{\mathbf{q}}^{\mathsf{t}}\mathbf{y} \geq \bar{\mathbf{q}}^{\mathsf{t}}\bar{\mathbf{y}} + l - g(\bar{\mathbf{y}}).$$

Otherwise, for each violated constraint, say $g(\bar{\mathbf{y}}) < l$, suppose that g is concave in $\mathbf{y}$ for $\mathbf{y} \geq \bar{\mathbf{y}}$. Then

$$g(\mathbf{y}) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}})$$

where $\bar{\mathbf{q}}$ is any subgradient of g at $\bar{\mathbf{y}}$.

But we want $g(\mathbf{y}) \geq l$, so we must have

$$l \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}}),$$

i.e.,

$$\bar{\mathbf{q}}^{\mathsf{t}}\mathbf{y} \geq \bar{\mathbf{q}}^{\mathsf{t}}\bar{\mathbf{y}} + l - g(\bar{\mathbf{y}}).$$

Adding this linear cut inequality to the constraints removes $\bar{\mathbf{y}}$ from the current set of feasible solutions.

How to obtain a subgradient?

How to obtain a subgradient?

Heuristic: simply use forward finite differences.
Compute g at $\mathbf{y}$ and at $\mathbf{y} + \mathbf{e}_j$ for $j = 1, \ldots, IP$.
Component j of $\bar{\mathbf{q}}$ is $g(\mathbf{y} + \mathbf{e}_j) - g(\mathbf{y})$ for all j.

How to obtain a subgradient?

Heuristic: simply use forward finite differences.
Compute g at $\mathbf{y}$ and at $\mathbf{y} + \mathbf{e}_j$ for $j = 1, \ldots, IP$.
Component j of $\bar{\mathbf{q}}$ is $g(\mathbf{y} + \mathbf{e}_j) - g(\mathbf{y})$ for all j.

Unfortunately, this is not necessarily a subgradient, even if g is concave.

How to obtain a subgradient?

Heuristic: simply use forward finite differences.
Compute g at $\mathbf{y}$ and at $\mathbf{y} + \mathbf{e}_j$ for $j = 1, \ldots, IP$.
Component j of $\bar{\mathbf{q}}$ is $g(\mathbf{y} + \mathbf{e}_j) - g(\mathbf{y})$ for all j.

Unfortunately, this is not necessarily a subgradient, even if g is concave.

Heuristic univarite "concavity check": For each j check if

$$g(\bar{\mathbf{y}} + 2\mathbf{e}_j) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(2\mathbf{e}_j).$$

These are only necessary concavity conditions.
Proving joint concavity appears much too difficult and unpractical.

How to obtain a subgradient?

Heuristic: simply use forward finite differences.
Compute g at $\mathbf{y}$ and at $\mathbf{y} + \mathbf{e}_j$ for $j = 1, \ldots, IP$.
Component j of $\bar{\mathbf{q}}$ is $g(\mathbf{y} + \mathbf{e}_j) - g(\mathbf{y})$ for all j.

Unfortunately, this is not necessarily a subgradient, even if g is concave.

Heuristic univarite "concavity check": For each j check if

$$g(\bar{\mathbf{y}} + 2\mathbf{e}_j) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(2\mathbf{e}_j).$$

These are only necessary concavity conditions.
Proving joint concavity appears much too difficult and unpractical.

Danger: if g not concave, or if $\bar{\mathbf{q}}$ is not a subgradient, we can cut out a large piece of the feasible set of solutions, including the optimal one.

How to obtain a subgradient?

Heuristic: simply use forward finite differences.
Compute g at $\mathbf{y}$ and at $\mathbf{y} + \mathbf{e}_j$ for $j = 1, \ldots, IP$.
Component j of $\bar{\mathbf{q}}$ is $g(\mathbf{y} + \mathbf{e}_j) - g(\mathbf{y})$ for all j.

Unfortunately, this is not necessarily a subgradient, even if g is concave.

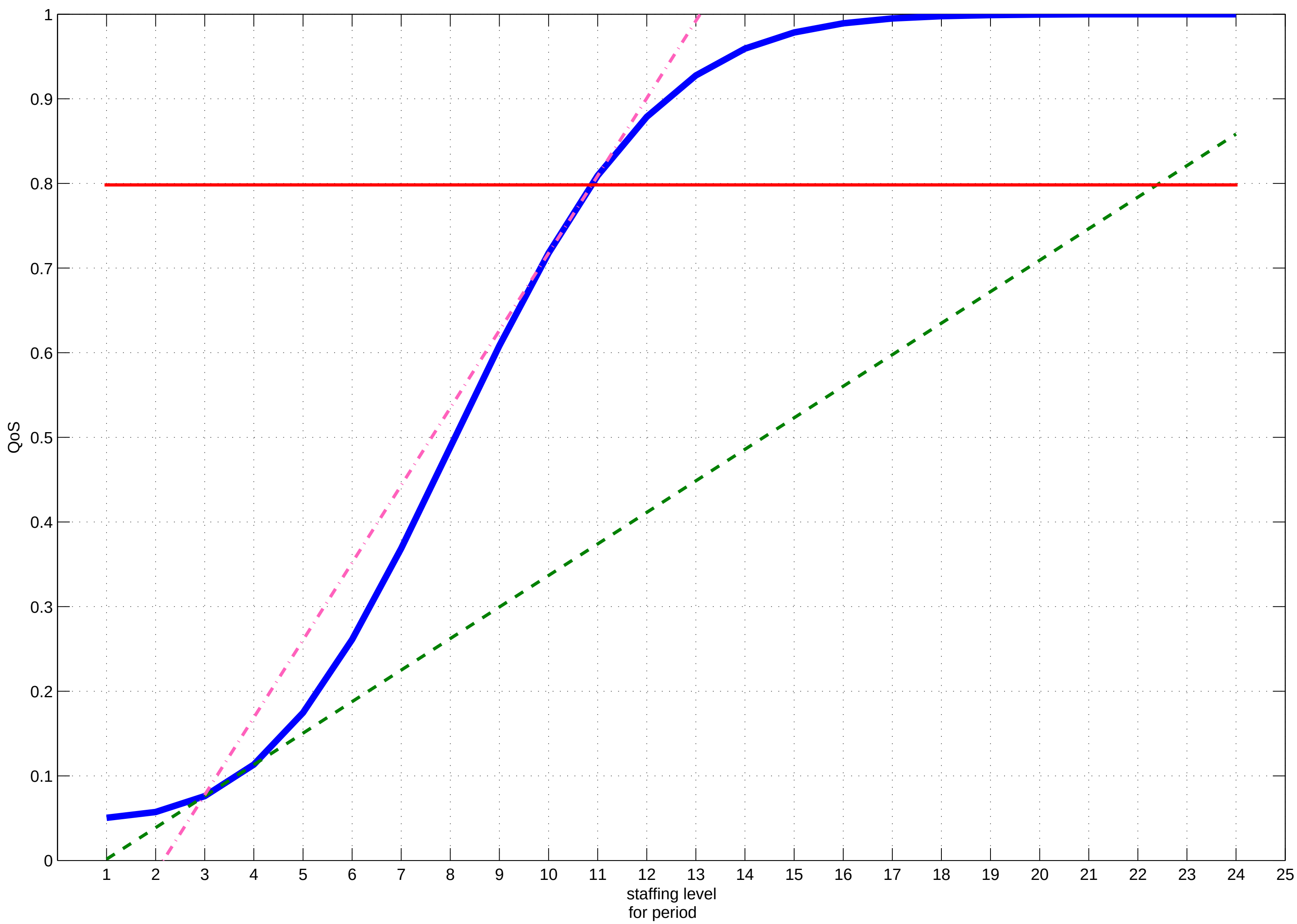
Heuristic univarite "concavity check": For each j check if

$$g(\bar{\mathbf{y}} + 2\mathbf{e}_j) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathrm{t}}(2\mathbf{e}_j).$$

These are only necessary concavity conditions.
Proving joint concavity appears much too difficult and unpractical.

Danger: if g not concave, or if $\bar{\mathbf{q}}$ is not a subgradient, we can cut out a large piece of the feasible set of solutions, including the optimal one.

What should we do when we detect non-concavity?



1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
QoS
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
staffing level
for period

Initial constraints.
For the full relaxation (no constraints), the optimal solution is $\mathbf{y} = \mathbf{0}$ and the functions $g_\bullet$ are usually convex at $\mathbf{0}$.

Initial constraints.
For the full relaxation (no constraints), the optimal solution is $\mathbf{y} = \mathbf{0}$ and the functions $g_\bullet$ are usually convex at $\mathbf{0}$.

Therefore, we cannot start the algorithm from there.

Initial constraints.

For the full relaxation (no constraints), the optimal solution is $\mathbf{y} = \mathbf{0}$ and the functions $g_\bullet$ are usually convex at $\mathbf{0}$.

Therefore, we cannot start the algorithm from there.

Heuristic: to remove areas of high non-concavity, restrict the set of admissible solutions a priori, by imposing (extra) initial constraints.

Initial constraints.

For the full relaxation (no constraints), the optimal solution is $\mathbf{y} = \mathbf{0}$ and the functions $g_\bullet$ are usually convex at $\mathbf{0}$.

Therefore, we cannot start the algorithm from there.

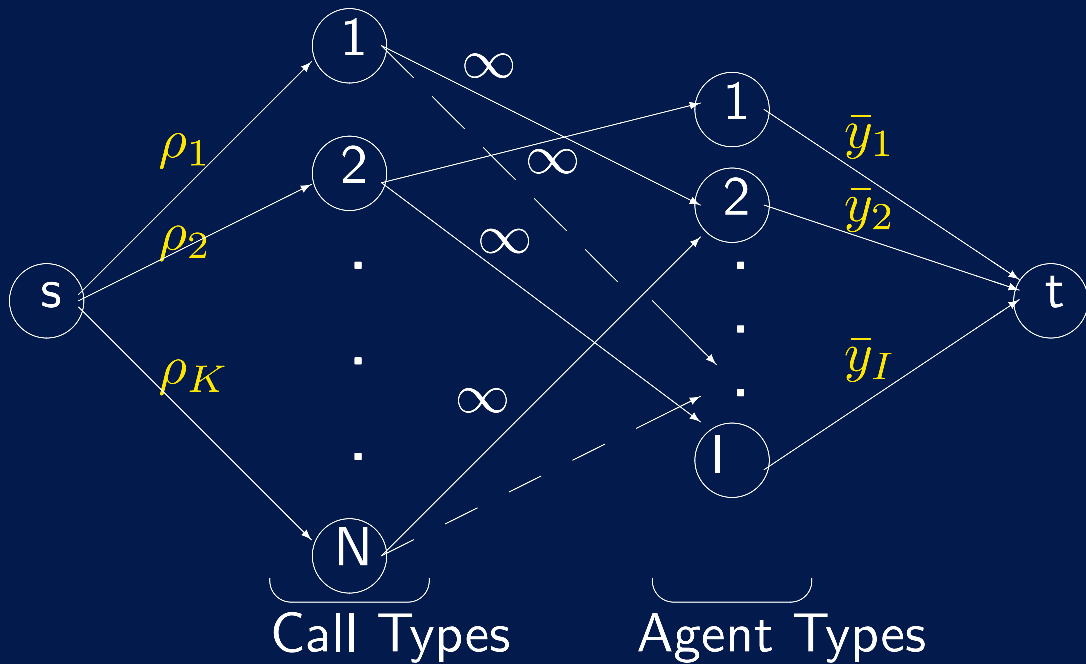
Heuristic: to remove areas of high non-concavity, restrict the set of admissible solutions a priori, by imposing (extra) initial constraints.

That is, impose $\mathbf{y} \in \mathcal{Y}$, in which the $g_\bullet$ are more likely to be concave.

What we do: impose that for each period p, the skill supply of the available agents can cover the total load for each call type.

What we do: impose that for each period p, the skill supply of the available agents can cover the total load for each call type.

That is, if $\rho_k = \lambda_k/\mu_k$ is the load for call type k (assuming the service rate depends on call type only), the maximum flow in the following graph must be $\sum_{k=1}^{K} \rho_k$:



In this graph, there is an arc from call type k to agent type i if and only if agent type i can handle call type k.

This "stability" condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

This "stability" condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

If the load cannot be handled by the agent mix, the max-flow algorithm finds a minimum cut that provides a linear inequality (constraint) on $\mathbf{y}$ needed to reach the required max flow, but violated by the current solution.

This "stability" condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

If the load cannot be handled by the agent mix, the max-flow algorithm finds a minimum cut that provides a linear inequality (constraint) on $\mathbf{y}$ needed to reach the required max flow, but violated by the current solution.
We add this constraint and find a new solution $\bar{\mathbf{y}}$.

This “stability” condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

If the load cannot be handled by the agent mix, the max-flow algorithm finds a minimum cut that provides a linear inequality (constraint) on $\mathbf{y}$ needed to reach the required max flow, but violated by the current solution.
We add this constraint and find a new solution $\bar{\mathbf{y}}$.
This is repeated until the load is covered by the current solution.

This "stability" condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

If the load cannot be handled by the agent mix, the max-flow algorithm finds a minimum cut that provides a linear inequality (constraint) on $\mathbf{y}$ needed to reach the required max flow, but violated by the current solution.
We add this constraint and find a new solution $\bar{\mathbf{y}}$.
This is repeated until the load is covered by the current solution.

Also repeated each time we have a new solution $\bar{\mathbf{y}}$, after adding QoS constraints.

This "stability" condition is not strictly necessary when there are losses (or abandonments). And if all calls must be served, it may not be sufficient for stability (depending, e.g., on priority rules). This is only a heuristic to cut out areas of high non-concavity of the functions g_*.

If the load cannot be handled by the agent mix, the max-flow algorithm finds a minimum cut that provides a linear inequality (constraint) on $\mathbf{y}$ needed to reach the required max flow, but violated by the current solution.
We add this constraint and find a new solution $\bar{\mathbf{y}}$.
This is repeated until the load is covered by the current solution.

Also repeated each time we have a new solution $\bar{\mathbf{y}}$, after adding QoS constraints.

Cut-generation method with queueing approx. can be problematic because we must handle low-QoS solutions.

Computational Experiments

Optimization with simulation (sample problem).
Single period, assume steady-state (start the system full and use batch means with 20 batches to estimate variance + 1 additional batch for warmup).

Computational Experiments

Optimization with simulation (sample problem).
Single period, assume steady-state (start the system full and use batch means with 20 batches to estimate variance + 1 additional batch for warmup).
Simulation time = 500 operating hours = 20 batches of 25 hours.

Computational Experiments

Optimization with simulation (sample problem).
Single period, assume steady-state (start the system full and use batch means with 20 batches to estimate variance + 1 additional batch for warmup).
Simulation time = 500 operating hours = 20 batches of 25 hours.

Routing: each call type has an ordered list of agents types, and goes in a queue (per call type) if all these agents are busy.
Each agent type has an ordered list of call types, for when he becomes free.

Example 1a (partly from Koole and Talim 2000).
5 call types, 12 agent types.

Example 1a (partly from Koole and Talim 2000).
5 call types, 12 agent types.

Skill groups and routing:

k	agent type i											
1	1		3	4	5		7	8	9		11	12
2			3			6	7	8			11	12
3		2		4		6	7		9	10	11	12
4					5					10		12
5								8	9	10	11	12

Example 1a (partly from Koole and Talim 2000).
5 call types, 12 agent types.

Skill groups and routing:

k	agent type i											
1	1		3	4	5		7	8	9		11	12
2			3			6	7	8			11	12
3		2		4		6	7		9	10	11	12
4					5					10		12
5								8	9	10	11	12

An agent that becomes free looks at call queues in numeric order.

Example 1a (partly from Koole and Talim 2000).
5 call types, 12 agent types.

Skill groups and routing:

k	agent type i											
1	1		3	4	5		7	8	9		11	12
2			3			6	7	8			11	12
3		2		4		6	7		9	10	11	12
4					5					10		12
5								8	9	10	11	12

An agent that becomes free looks at call queues in numeric order.

This makes the system highly unbalanced $\Rightarrow$ generally more difficult to solve.

Example 1a (partly from Koole and Talim 2000).
5 call types, 12 agent types.

Skill groups and routing:

k	agent type i											
1	1		3	4	5		7	8	9		11	12
2			3			6	7	8			11	12
3		2		4		6	7		9	10	11	12
4					5					10		12
5								8	9	10	11	12

An agent that becomes free looks at call queues in numeric order.

This makes the system highly unbalanced $\Rightarrow$ generally more difficult to solve.

Agent's costs: $1 + \kappa\cdot$(number of skills -1), where $\kappa = 0.10$, 0.15, 0.20.

Arrival rate $\lambda_k = 240$ per hour for all k;
Service rate $\mu_k = 12$ per hour for all k.

Arrival rate $\lambda_k = 240$ per hour for all k;
Service rate $\mu_k = 12$ per hour for all k.

No abandonment.

Arrival rate $\lambda_k = 240$ per hour for all k;
Service rate $\mu_k = 12$ per hour for all k.

No abandonment.

Minimal QoS overall: $l = 0.80$.

Example 1a. $\ell = .80$ and $\ell_k = 0$ for all k.

κ cuts	obj. CPU (sec)	QoS QoS KT	QoS per call type staffing vector
500 hours			
.10 10	119.6 746	$.803 \pm .002$ .540	(.98, .97, .99, .79, .00) (0, 18, 0, 0, 20, 20, 27, 0, 0, 21, 0, 0)
.15 16	127.7 1387	$.802 \pm .002$ .624	(.98, .99, .99, .82, .00) (1, 0, 0, 17, 20, 37, 11, 0, 0, 21, 0, 0)
.20 17	131.8 1500	$.800 \pm .002$ .491	(.98, .96, .99, .80, .00) (0, 15, 0, 4, 20, 22, 23, 0, 0, 21, 0, 0)

Example 1a. $\ell = .80$ and $\ell_k = 0$ for all k.

κ cuts	obj. CPU (sec)	QoS QoS KT	QoS per call type staffing vector
500 hours			
.10 10	119.6 746	$.803 \pm .002$ .540	(.98, .97, .99, .79, .00) (0, 18, 0, 0, 20, 20, 27, 0, 0, 21, 0, 0)
.15 16	127.7 1387	$.802 \pm .002$ .624	(.98, .99, .99, .82, .00) (1, 0, 0, 17, 20, 37, 11, 0, 0, 21, 0, 0)
.20 17	131.8 1500	$.800 \pm .002$ .491	(.98, .96, .99, .80, .00) (0, 15, 0, 4, 20, 22, 23, 0, 0, 21, 0, 0)
50 hours			
.10 10	125.7 68	$.800 \pm .005$ .844	(1.00, .98, 1.00, .88, .00) (6, 24, 0, 9, 20, 22, 12, 0, 0, 21, 0, 0)
.15 19	142.7 125	$.827 \pm .025$ .959	(1.00, .80, 1.00, .98, .35) (0, 29, 21, 18, 26, 9, 0, 0, 0, 22, 0, 0)
.20 13	141.6 85	$.806 \pm .028$ .906	(1.00, .99, .99, .80, .25) (0, 22, 0, 8, 32, 20, 12, 21, 1, 0, 0, 0)

Constraints per call type: $\ell = .80$ and $\ell_k = 0.5$ for all k.

Constraints per call type: $\ell = .80$ and $\ell_k = 0.5$ for all k.

Difficulty: If the QoS for a given call type is almost zero (see above) and we want to generate a corresponding cut, we run into a non-concavity problem.

Constraints per call type: $\ell = .80$ and $\ell_k = 0.5$ for all k.

Difficulty: If the QoS for a given call type is almost zero (see above) and we want to generate a corresponding cut, we run into a non-concavity problem.

The "subgradient" is often nearly flat (zero or almost zero) and the corresponding cut would remove all the good solutions!

Constraints per call type: $\ell = .80$ and $\ell_k = 0.5$ for all k.

Difficulty: If the QoS for a given call type is almost zero (see above) and we want to generate a corresponding cut, we run into a non-concavity problem.

The “subgradient” is often nearly flat (zero or almost zero) and the corresponding cut would remove all the good solutions!

So in this case (which typically happens for highly unbalanced systems), we need a different type of heuristic.

Constraints per call type: $\ell = .80$ and $\ell_k = 0.5$ for all k.

Difficulty: If the QoS for a given call type is almost zero (see above) and we want to generate a corresponding cut, we run into a non-concavity problem.

The “subgradient” is often nearly flat (zero or almost zero) and the corresponding cut would remove all the good solutions!

So in this case (which typically happens for highly unbalanced systems), we need a different type of heuristic.

We add a non-subgradient-type cut as follows:
Choose k for which $G_{n,k}(\bar{\mathbf{y}}) - \ell_k$ is smallest and replace the constraint

$$\sum_{i=1}^{I} y_i \mathcal{I}[\text{agent } i \text{ has skill } k] \geq \rho_k$$

by

$$\sum_{i=1}^{I} y_i \mathcal{I}[\text{agent } i \text{ has skill } k] \geq (1+\delta)\rho_k$$

for some $\delta > 0$.

Example 1a. $\ell = .80$ and $\ell_k = 0.50$ for all k, 500 hours.

κ cuts	obj. CPU (sec)	QoS QoS KT	QoS per call type staffing vector
.10 24	124.5 222	$.809 \pm .006$ .701	(.99, .79, .86, .75, .63) (0, 5, 0, 0, 21, 0, 0, 43, 10, 24, 2, 1)

Example 1a. $\ell = .80$ and $\ell_k = 0.50$ for all k, 500 hours.

κ cuts	obj. CPU (sec)	QoS QoS KT	QoS per call type staffing vector
.10 24	124.5 222	$.809 \pm .006$ .701	(.99, .79, .86, .75, .63) (0, 5, 0, 0, 21, 0, 0, 43, 10, 24, 2, 1)
.15 18	133.0 207	$.800 \pm .003$ .633	(.97, .72, .92, .79, .58) (0, 6, 0, 3, 17, 0, 0, 45, 0, 35, 0, 0)

Example 1a. $\ell = .80$ and $\ell_k = 0.50$ for all k, 500 hours.

κ cuts	obj. CPU (sec)	QoS QoS KT	QoS per call type staffing vector
.10 24	124.5 222	$.809 \pm .006$ .701	(.99, .79, .86, .75, .63) (0, 5, 0, 0, 21, 0, 0, 43, 10, 24, 2, 1)
.15 18	133.0 207	$.800 \pm .003$ .633	(.97, .72, .92, .79, .58) (0, 6, 0, 3, 17, 0, 0, 45, 0, 35, 0, 0)
.20 22	142.2 220	$.818 \pm .007$ .677	(.98, .70, .90, .84, .66) (9, 2, 0, 0, 16, 0, 0, 39, 0, 41, 0, 0)

Example 1b. Same, except: Abandonment with prob. 0.01, then at rate 10.0.
Arrival rates $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$. $\ell = .80$.
Simulate 500 operating hours.

κ cuts	obj. CPU (sec)	QoS	QoS per call type staffing vector
$\ell_k = 0$			
.10 13	218.5 1521	$.800 \pm .003$	(.99, .90, .97, .90, .15) (34, 31, 15, 0, 45, 37, 0, 12, 0, 27, 0, 0)

Example 1b. Same, except: Abandonment with prob. 0.01, then at rate 10.0.
Arrival rates $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$. $\ell = .80$.
Simulate 500 operating hours.

κ / cuts	obj. / CPU (sec)	QoS	QoS per call type / staffing vector
$\ell_k = 0$			
.10 / 13	218.5 / 1521	$.800 \pm .003$	(.99, .90, .97, .90, .15) / (34, 31, 15, 0, 45, 37, 0, 12, 0, 27, 0, 0)
.15 / 13	227.25 / 1521	$.802 \pm .002$	(.99, .90, .97, .89, .18) / (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)

Example 1b. Same, except: Abandonment with prob. 0.01, then at rate 10.0. Arrival rates $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$. $\ell = .80$.
Simulate 500 operating hours.

κ cuts	obj. CPU (sec)	QoS	QoS per call type staffing vector
$\ell_k = 0$			
.10 13	218.5 1521	$.800 \pm .003$	(.99, .90, .97, .90, .15) (34, 31, 15, 0, 45, 37, 0, 12, 0, 27, 0, 0)
.15 13	227.25 1521	$.802 \pm .002$	(.99, .90, .97, .89, .18) (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
.20 13	236.0 1530	$.800 \pm .003$	(.99, .90, .97, .89, .18) (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
$\ell_k = 0.5$			
.10 24	221.6 1474	$.801 \pm .006$	(0.99, 0.94, 0.86, 0.67, 0.52) (20, 31, 36, 0, 42, 4, 0, 50, 0, 17, 0, 0)

Example 1b. Same, except: Abandonment with prob. 0.01, then at rate 10.0.
Arrival rates $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$. $\ell = .80$.
Simulate 500 operating hours.

κ / cuts	obj. / CPU (sec)	QoS	QoS per call type / staffing vector
$\ell_k = 0$			
.10 / 13	218.5 / 1521	$.800 \pm .003$	(.99, .90, .97, .90, .15) / (34, 31, 15, 0, 45, 37, 0, 12, 0, 27, 0, 0)
.15 / 13	227.25 / 1521	$.802 \pm .002$	(.99, .90, .97, .89, .18) / (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
.20 / 13	236.0 / 1530	$.800 \pm .003$	(.99, .90, .97, .89, .18) / (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
$\ell_k = 0.5$			
.10 / 24	221.6 / 1474	$.801 \pm .006$	(0.99, 0.94, 0.86, 0.67, 0.52) / (20, 31, 36, 0, 42, 4, 0, 50, 0, 17, 0, 0)
.15 / 24	232.5 / 1475	$.803 \pm .007$	(0.99, 0.94, 0.88, 0.66, 0.52) / (25, 33, 34, 0, 40, 2, 0, 50, 0, 17, 0, 0)

Example 1b. Same, except: Abandonment with prob. 0.01, then at rate 10.0.
Arrival rates $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$. $\ell = .80$.
Simulate 500 operating hours.

κ / cuts	obj. / CPU (sec)	QoS	QoS per call type / staffing vector
$\ell_k = 0$			
.10 / 13	218.5 / 1521	$.800 \pm .003$	(.99, .90, .97, .90, .15) / (34, 31, 15, 0, 45, 37, 0, 12, 0, 27, 0, 0)
.15 / 13	227.25 / 1521	$.802 \pm .002$	(.99, .90, .97, .89, .18) / (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
.20 / 13	236.0 / 1530	$.800 \pm .003$	(.99, .90, .97, .89, .18) / (33, 34, 30, 0, 43, 20, 0, 15, 0, 26, 0, 0)
$\ell_k = 0.5$			
.10 / 24	221.6 / 1474	$.801 \pm .006$	(0.99, 0.94, 0.86, 0.67, 0.52) / (20, 31, 36, 0, 42, 4, 0, 50, 0, 17, 0, 0)
.15 / 24	232.5 / 1475	$.803 \pm .007$	(0.99, 0.94, 0.88, 0.66, 0.52) / (25, 33, 34, 0, 40, 2, 0, 50, 0, 17, 0, 0)
.20 / 24	242.6 / 1410	$.800 \pm .006$	(0.99, 0.96, 0.81, 0.67, 0.53) / (25, 34, 41, 0, 45, 4, 0, 46, 0, 8, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)
1. QoS cut	236.0	(.99, .94, .97, .87, .12)	.795	(26, 31, 8, 0, 52, 46, 0, 17, 0, 20, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)
1. QoS cut	236.0	(.99, .94, .97, .87, .12)	.795	(26, 31, 8, 0, 52, 46, 0, 17, 0, 20, 0, 0)
11. QoS cut	236.2	(.99, .94, .98, .88, .12)	.798	(29, 28, 5, 0, 48, 52, 0, 11, 0, 27, 0, 0)

How the algorithm behaves:

Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)
1. QoS cut	236.0	(.99, .94, .97, .87, .12)	.795	(26, 31, 8, 0, 52, 46, 0, 17, 0, 20, 0, 0)
11. QoS cut	236.2	(.99, .94, .98, .88, .12)	.798	(29, 28, 5, 0, 48, 52, 0, 11, 0, 27, 0, 0)
12. QoS cut	236.2	(.99, .92, .97, .90, .13)	.798	(27, 30, 8, 0, 51, 46, 0, 13, 0, 25, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)
1. QoS cut	236.0	(.99, .94, .97, .87, .12)	.795	(26, 31, 8, 0, 52, 46, 0, 17, 0, 20, 0, 0)
11. QoS cut	236.2	(.99, .94, .98, .88, .12)	.798	(29, 28, 5, 0, 48, 52, 0, 11, 0, 27, 0, 0)
12. QoS cut	236.2	(.99, .92, .97, .90, .13)	.798	(27, 30, 8, 0, 51, 46, 0, 13, 0, 25, 0, 0)
13. QoS cut	236.2	(.99, .87, .97, .93, .14)	.795	(25, 32, 15, 0, 56, 34, 0, 15, 0, 23, 0, 0)

How the algorithm behaves:
Example 1b, 500 hours, $\kappa = 0.20$, only overall QoS constraint.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS cut	233.0	(.99, .96, .94, .50, .18)	.721	(36, 36, 16, 0, 45, 30, 0, 37, 0, 0, 0, 0)
3. QoS cut	233.0	—	—	(36, 36, 0, 0, 45, 46, 0, 19, 0, 18, 0, 0)
flow-cuts	233.0	(.99, .90, .98, .84, .08)	.773	(36, 36, 1, 0, 45, 45, 0, 19, 0, 18, 0, 0)
4. QoS cut	235.8	(.99, .97, .95, .75, .14)	.772	(36, 23, 0, 0, 45, 58, 0, 26, 0, 12, 0, 0)
5. QoS cut	235.8	(.99, .94, .97, .89, .10)	.793	(36, 22, 1, 0, 45, 59, 0, 9, 0, 28, 0, 0)
6. QoS cut	235.8	(.99, .86, .97, .93, .11)	.786	(34, 24, 1, 0, 47, 57, 0, 0, 0, 37, 0, 0)
7. QoS cut	236.0	(.99, .96, .96, .78, .14)	.778	(22, 35, 3, 0, 58, 45, 0, 33, 0, 4, 0, 0)
8. QoS cut	236.0	(.99, .95, .98, .86, .12)	.795	(28, 29, 5, 0, 49, 52, 0, 15, 0, 22, 0, 0)
9. QoS cut	236.0	(.99, .91, .97, .91, .12)	.796	(22, 35, 12, 0, 59, 35, 0, 23, 0, 14, 0, 0)
1. QoS cut	236.0	(.99, .94, .97, .87, .12)	.795	(26, 31, 8, 0, 52, 46, 0, 17, 0, 20, 0, 0)
11. QoS cut	236.2	(.99, .94, .98, .88, .12)	.798	(29, 28, 5, 0, 48, 52, 0, 11, 0, 27, 0, 0)
12. QoS cut	236.2	(.99, .92, .97, .90, .13)	.798	(27, 30, 8, 0, 51, 46, 0, 13, 0, 25, 0, 0)
13. QoS cut	236.2	(.99, .87, .97, .93, .14)	.795	(25, 32, 15, 0, 56, 34, 0, 15, 0, 23, 0, 0)
14. QoS cut	**236.2**	(.99, .90, .97, .92, .16)	**.803**	(28, 29, 11, 0, 50, 44, 0, 8, 0, 30, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)
11. QoS	241.0	(.99, .97, .66, .68, .49)	.769	(22, 25, 41, 0, 48, 12, 0, 45, 0, 7, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)
11. QoS	241.0	(.99, .97, .66, .68, .49)	.769	(22, 25, 41, 0, 48, 12, 0, 45, 0, 7, 0, 0)
12. QoS type 5	241.2	(.99, .97, .49, .77, .48)	.758	(18, 25, 44, 0, 53, 11, 0, 46, 0, 3, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)
11. QoS	241.0	(.99, .97, .66, .68, .49)	.769	(22, 25, 41, 0, 48, 12, 0, 45, 0, 7, 0, 0)
12. QoS type 5	241.2	(.99, .97, .49, .77, .48)	.758	(18, 25, 44, 0, 53, 11, 0, 46, 0, 3, 0, 0)
13. QoS typ. 3,5	241.2	(.99, .97, .54, .73, .50)	.761	(22, 24, 39, 0, 51, 12, 0, 47, 0, 5, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)
11. QoS	241.0	(.99, .97, .66, .68, .49)	.769	(22, 25, 41, 0, 48, 12, 0, 45, 0, 7, 0, 0)
12. QoS type 5	241.2	(.99, .97, .49, .77, .48)	.758	(18, 25, 44, 0, 53, 11, 0, 46, 0, 3, 0, 0)
13. QoS typ. 3,5	241.2	(.99, .97, .54, .73, .50)	.761	(22, 24, 39, 0, 51, 12, 0, 47, 0, 5, 0, 0)
14. QoS	241.2	(.99, .98, .82, .55, .53)	.779	(24, 24, 39, 0, 44, 22, 0, 43, 0, 5, 0, 0)

Same example, with QoS contraint of $\ell_k = 0.50$ per call type.

iteration	obj.	QoS per call type	QoS	staffing vector
1. flow-cuts	233.0	(.99, .97, .60, .55, .25)	.685	(36, 36, 45, 0, 45, 1, 0, 37, 0, 0, 0, 0)
2. QoS type 5	234.2	—	—	(36, 36, 45, 1, 38, 1, 0, 43, 0, 0, 0, 0)
flow-cuts	235.4	(.99, .95, .62, .54, .41)	.711	(36, 36, 34, 0, 45, 0, 0, 48, 1, 0, 0, 0)
3. QoS type 5	235.4	—	—	(36, 33, 40, 0, 43, 2, 0, 44, 2, 0, 0, 0)
flow-cuts	235.8	(.99, .96, .63, .51, .48)	.723	(31, 36, 41, 0, 45, 1, 0, 46, 0, 0, 0, 0)
4. QoS type 5	235.8	(.99, .97, .48, .57, .47)	.706	(36, 30, 37, 0, 45, 7, 0, 45, 0, 0, 0, 0)
5. QoS typ. 3,5	235.8	(.99, .97, .53, .53, .48)	.711	(33, 32, 41, 0, 45, 5, 0, 44, 0, 0, 0, 0)
6. QoS type 5	236.0	—	—	(33, 34, 40, 0, 43, 3, 2, 45, 0, 0, 0, 0)
flow-cuts	236.0	(.99, .97, .55, .52, .48)	.710	(33, 33, 39, 0, 45, 4, 0, 46, 0, 0, 0, 0)
7. QoS type 5	236.0	(.99, .97, .61, .51, .44)	.713	(25, 36, 52, 0, 45, 1, 0, 41, 0, 0, 0, 0)
8. QoS type 5	236.2	(.99, .97, .55, .53, .50)	.718	(30, 32, 45, 0, 45, 5, 0, 43, 0, 0, 0, 0)
9. QoS	239.8	(.99, .97, .88, .37, .57)	.753	(18, 36, 34, 0, 45, 14, 0, 53, 0, 0, 0, 0)
10. QoS type 4	240.0	(.99, .95, .77, .55, .53)	.763	(20, 36, 32, 0, 49, 7, 0, 56, 0, 0, 0, 0)
11. QoS	241.0	(.99, .97, .66, .68, .49)	.769	(22, 25, 41, 0, 48, 12, 0, 45, 0, 7, 0, 0)
12. QoS type 5	241.2	(.99, .97, .49, .77, .48)	.758	(18, 25, 44, 0, 53, 11, 0, 46, 0, 3, 0, 0)
13. QoS typ. 3,5	241.2	(.99, .97, .54, .73, .50)	.761	(22, 24, 39, 0, 51, 12, 0, 47, 0, 5, 0, 0)
14. QoS	241.2	(.99, .98, .82, .55, .53)	.779	(24, 24, 39, 0, 44, 22, 0, 43, 0, 5, 0, 0)
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)
19. QoS	242.0	(.99, .96, .83, .61, .52)	.789	(31, 32, 39, 0, 41, 5, 0, 44, 0, 11, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)
19. QoS	242.0	(.99, .96, .83, .61, .52)	.789	(31, 32, 39, 0, 41, 5, 0, 44, 0, 11, 0, 0)
20. QoS	242.2	(.99, .95, .87, .61, .53)	.796	(27, 34, 36, 0, 41, 4, 0, 48, 0, 12, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)
19. QoS	242.0	(.99, .96, .83, .61, .52)	.789	(31, 32, 39, 0, 41, 5, 0, 44, 0, 11, 0, 0)
20. QoS	242.2	(.99, .95, .87, .61, .53)	.796	(27, 34, 36, 0, 41, 4, 0, 48, 0, 12, 0, 0)
21. QoS	242.4	(.99, .95, .88, .61, .54)	.798	(26, 33, 37, 0, 41, 6, 0, 47, 0, 12, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)
19. QoS	242.0	(.99, .96, .83, .61, .52)	.789	(31, 32, 39, 0, 41, 5, 0, 44, 0, 11, 0, 0)
20. QoS	242.2	(.99, .95, .87, .61, .53)	.796	(27, 34, 36, 0, 41, 4, 0, 48, 0, 12, 0, 0)
21. QoS	242.4	(.99, .95, .88, .61, .54)	.798	(26, 33, 37, 0, 41, 6, 0, 47, 0, 12, 0, 0)
22. QoS	242.6	(.99, .95, .84, .63, .53)	.794	(27, 32, 39, 0, 41, 3, 0, 46, 0, 14, 0, 0)

iteration	obj.	QoS per call type	QoS	staffing vector
⋮				
15. QoS	241.8	(.99, .96, .85, .59, .52)	.787	(26, 38, 42, 0, 46, 3, 0, 46, 1, 2, 0, 0)
16. QoS	241.8	(.99, .96, .85, .59, .55)	.791	(26, 31, 42, 0, 43, 6, 0, 42, 5, 7, 0, 0)
17. QoS	242.0	(.99, .94, .87, .62, .51)	.791	(26, 33, 38, 0, 39, 2, 0, 47, 0, 16, 0, 0)
18. QoS	242.0	(.99, .96, .82, .62, .52)	.789	(23, 30, 38, 0, 44, 9, 0, 47, 0, 10, 0, 0)
19. QoS	242.0	(.99, .96, .83, .61, .52)	.789	(31, 32, 39, 0, 41, 5, 0, 44, 0, 11, 0, 0)
20. QoS	242.2	(.99, .95, .87, .61, .53)	.796	(27, 34, 36, 0, 41, 4, 0, 48, 0, 12, 0, 0)
21. QoS	242.4	(.99, .95, .88, .61, .54)	.798	(26, 33, 37, 0, 41, 6, 0, 47, 0, 12, 0, 0)
22. QoS	242.6	(.99, .95, .84, .63, .53)	.794	(27, 32, 39, 0, 41, 3, 0, 46, 0, 14, 0, 0)
23. QoS	**242.6**	(.99, .96, .81, .67, .53)	**.801**	(25, 34, 41, 0, 45, 4, 0, 46, 0, 8, 0, 0)

Example 2a. 20 call types. 15 agent types.
Arrival rates λ_k vary between 125 and 260 (total = 4080); $\mu_k = 12$ for all k.
No abandonment; $\ell = .80$; simulate 500 hours.

Example 2a. 20 call types. 15 agent types.
Arrival rates λ_k vary between 125 and 260 (total = 4080); $\mu_k = 12$ for all k.
No abandonment; $\ell = .80$; simulate 500 hours.

κ	obj.	QoS	QoS KT	cuts	CPU (sec)
QoS per call type					
$\ell_k = 0$					
.20	627.0	$.802 \pm .002$	.084	6	1239
(1.0, .99, .76, .91, .99, .99, .98, .80, .00, .89, .98, .98, .97, .94, .92, .93, .91, .51, .21, .00)					
$\ell_k = 0.50$					
.20	667.8	$.841 \pm .006$	.689	50	6312
(.99, .99, .98, .66, .97, .99, .97, .98, .54, .89, .96, .97, .94, .99, .51, .83, .53, .87, .76, .72)					

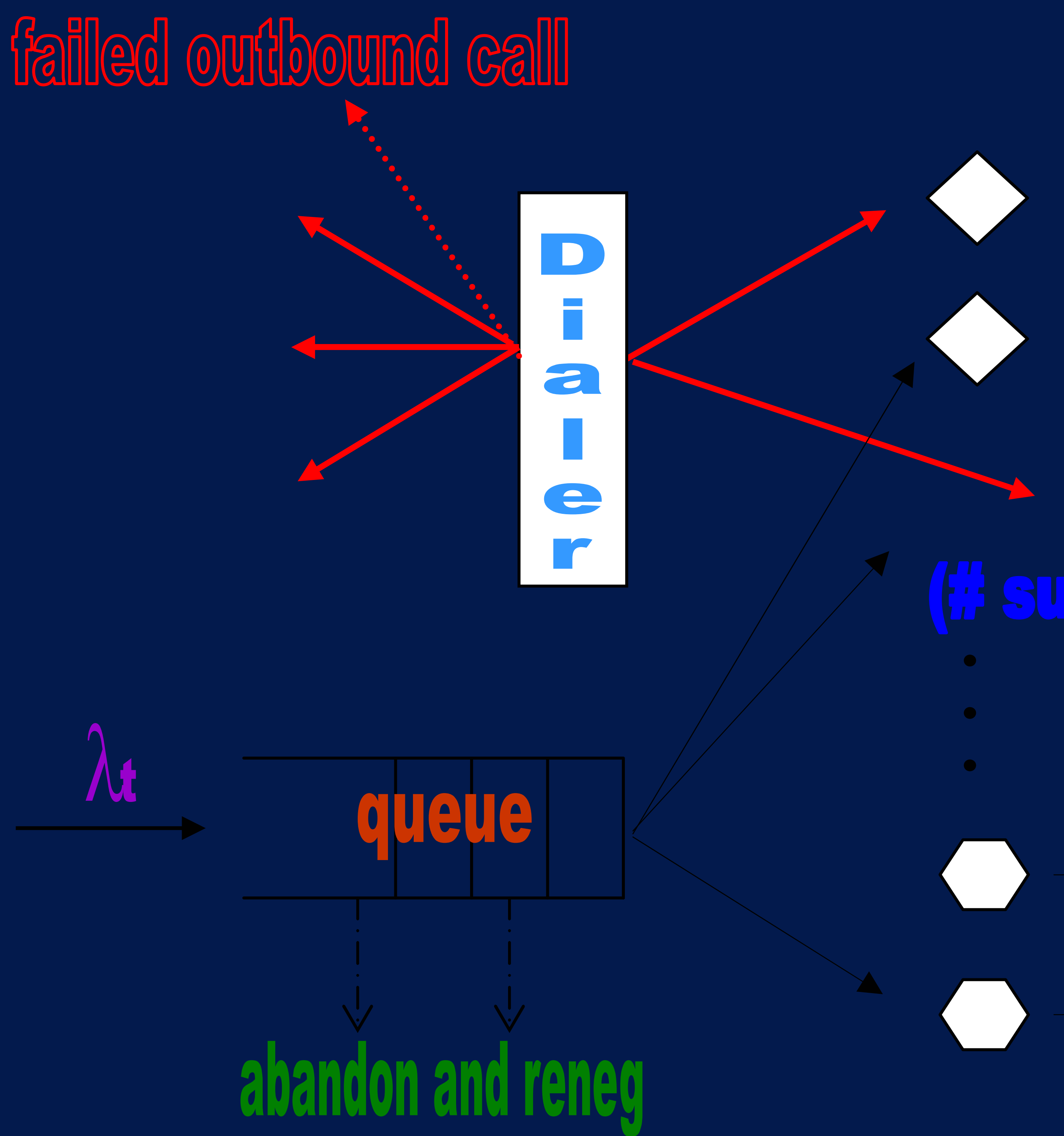
Example 2b. Same, except: abandonment with prob. 0.01, then at rate 10.0.

κ	obj.	QoS	cuts	CPU (sec)
QoS per call type				
$\ell_k = 0$				
.20	551.0	$.800 \pm .002$	1	107
(.99, .99, .60, .33, .99, .98, .88, .77, .34, .84, .79, .99, .98, .96, .97, .68, .99, .32, .79, .70)				
$\ell_k = 0.50$				
.20	553.8	$.801 \pm .002$	6	305
(.99, .98, .68, .50, .98, .99, .57, .80, .53, .77, .76, .99, .99, .87, .99, .66, .98, .74, .50, .81)				

Example 2c. Same, except: Abandonment with prob. 0.01, then at rate 10.0. Arrival rates vary between 225 and 360 (total = 5750) (more traffic).

κ	obj.	QoS	cuts	CPU (sec)
QoS per call type				
$\ell_k = 0$				
.20	859.6	$.800 \pm .002$	11	2186
(.99, .99, .77, .27, .99, .99, .92, .94, .01, .66, .88, .99, .99, .98, .98, .51, .98, .88, .61, .57)				
$\ell_k = 0.50$				
.20	916.2	$.854 \pm .003$	15	1197
(.99, .99, .98, .95, .99, .99, .95, .93, .87, .52, .99, .94, .89, .93, .59, .99, .84, .76, .50, .51)				

A Blend System


failed outbound call
Dialer
(# su
λ_t
queue
abandon and reneg

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Can use detailed simulation or simplified approximations.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Can use detailed simulation or simplified approximations.

We have developed steady-state CTMC models of varying complexities (Deslauriers, L'Ecuyer, Pichitlamken, Ingolfsson, Avramidis 2004):

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Can use detailed simulation or simplified approximations.

We have developed steady-state CTMC models of varying complexities (Deslauriers, L'Ecuyer, Pichitlamken, Ingolfsson, Avramidis 2004):

M1. All blend agents, $\mu_1 = \mu_2$, one dial at a time. Birth-death.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Can use detailed simulation or simplified approximations.

We have developed steady-state CTMC models of varying complexities (Deslauriers, L'Ecuyer, Pichitlamken, Ingolfsson, Avramidis 2004):

M1. All blend agents, $\mu_1 = \mu_2$, one dial at a time. Birth-death.
M3. Two agent types, $\mu_1 \neq \mu_2$, several dials at a time.

Problem: want to select a collection of shifts for agents to minimize costs subject to a QoS constraint on inbound calls.

Perhaps another QoS constraint on the expected volume of successful outbound calls, or they can be incorporated in the cost function.

An agent can be inbound-only or blend.

Can use detailed simulation or simplified approximations.

We have developed steady-state CTMC models of varying complexities (Deslauriers, L'Ecuyer, Pichitlamken, Ingolfsson, Avramidis 2004):

M1. All blend agents, $\mu_1 = \mu_2$, one dial at a time. Birth-death.
M3. Two agent types, $\mu_1 \neq \mu_2$, several dials at a time.
Etc.

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

if wait > 0, joins queue with probability γ;
then abandonment rate ν;

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

if wait > 0, joins queue with probability γ;
then abandonment rate ν;

service rate μ;

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

if wait > 0, joins queue with probability γ;
then abandonment rate ν;

service rate μ;

n identical blend agents;

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

if wait > 0, joins queue with probability γ;
then abandonment rate ν;

service rate μ;

n identical blend agents;

Outbound dialing: one call iff no more than $\dot{n}$ busy agents (threshold policy);
exponential connection delay; successful with probability κ.

M1. Birth-death model (simplest).

Poisson arrivals, rate λ;

if wait > 0, joins queue with probability γ;
then abandonment rate ν;

service rate μ;

n identical blend agents;

Outbound dialing: one call iff no more than $\dot{n}$ busy agents (threshold policy);
exponential connection delay; successful with probability κ.

Finite queue length c.

Optimization:

1. With M1 in each period.
2. Simulation with 400 runs of a 25-period (12.5 hours) day.
   Nonstationary, but otherwise similar to M1 model.

Optimization:

1. With M1 in each period.
2. Simulation with 400 runs of a 25-period (12.5 hours) day.
   Nonstationary, but otherwise similar to M1 model.

QoS: (80, 20) rule.

Optimization:

1. With M1 in each period.
2. Simulation with 400 runs of a 25-period (12.5 hours) day.
   Nonstationary, but otherwise similar to M1 model.

QoS: (80, 20) rule.

Number of admissible shifts: 110.
Each shift: 7 working hours + 1 hour lunch (movable).

Period	Start	λ_i	κ_i	$1/\nu_i$	$1/\mu_{1,i}$	$1/\mu_{2,i}$	agents	
							simul.	M1
1	8.0	26.1	0	400	595.6	440.2	8	9
2	8.5	38.3	0	400	595.6	440.2	13	16
3	9.0	50.9	0	400	595.6	440.2	20	22
4	9.5	58.0	0	700	595.6	440.2	24	25
5	10.0	61.4	0	700	595.6	440.2	24	25
6	10.5	62.2	0	600	595.6	440.2	24	25
7	11.0	62.6	0	600	595.6	440.2	27	25
8	11.5	60.5	0	600	595.6	440.2	27	29
9	12.0	56.8	0	600	575.1	440.2	26	26
10	12.5	56.7	0	600	575.1	440.2	23	25
11	13.0	59.0	0.30	500	575.1	440.2	24	23
12	13.5	57.3	0.33	500	575.1	440.2	27	32
13	14.0	56.5	0.27	500	541.5	440.2	26	23
14	14.5	55.4	0.27	500	518.2	440.2	31	23
15	15.0	56.5	0.28	500	509.2	440.2	27	31
16	15.5	59.0	0.29	500	506.0	440.2	27	31
17	16.0	59.9	0.29	500	512.8	440.2	28	28
18	16.5	54.7	0.30	500	505.7	440.2	23	21
19	17.0	43.5	0.33	500	505.2	440.2	16	15
20	17.5	40.0	0.37	500	491.0	440.2	12	12
21	18.0	33.3	0.40	500	494.9	440.2	12	12
22	18.5	29.7	0.38	500	471.9	440.2	12	12
23	19.0	25.0	0.41	500	468.4	440.2	9	12
24	19.5	22.5	0.41	100	462.6	440.2	9	8
25	20.0	17.9	0.41	50	460.4	440.2	5	8

Optim. with	CPU (sec)	num. shifts	QoS with M1	QoS with simul.
M1	18	37	**.803**	.815
simulation	2349	36	.785	**.803**

Ongoing and Future Work in Our Group

- Develop a Java library for simulation of call centers.
- Integrate with optimization algorithms (CPLEX-based IP, perhaps Lagrangian relaxation, neighborhood search, etc.) for staffing, scheduling, rostering, routing, . . .
- Integrate with approximation formulas and CTMC models.
- Better models for inbound traffic and other aspects.
- Currently, our work is strongly driven by the interests of Bell Canada.