1 Appendix

1.1 Solution to small example

Shift					Agent 1	Agent 2
length	shift start	break 1 delay	lunch start	break 3 delay		
7:30	8:00	1:30	12:30	1:45	2	0
7:30	8:00	1:45	13:00	1:30	2	0
7:30	8:00	1:45	12:00	2:00	1	0
7:30	9:30	1:30	12:00	2:00	2	0
7:30	9:30	2:00	12:00	2:00	0	2
7:45	9:15	1:30	12:00	2:00	1	0
7:45	9:15	1:30	12:30	1:45	0	1
8:00	9:00	1:30	12:30	1:45	1	0
8:00	9:00	1:30	13:00	1:30	1	0
8:00	9:00	1:45	13:00	1:30	1	0
8:15	8:45	1:30	13:00	1:30	2	0
8:30	8:30	1:30	12:30	1:45	0	1
8:30	8:30	1:30	12:30	2:00	1	0
9:00	8:00	3:00	14:00	1:45	0	1
9:00	8:00	3:15	13:30	1:15	0	1
9:00	8:00	3:15	14:00	1:15	0	1
9:00	8:00	3:15	14:00	1:30	2	0
9:00	8:00	3:30	13:30	1:30	2	0
9:00	8:00	3:45	13:30	1:15	2	1
6:30	10:00	1:45	14:00	0:45	2	0
6:30	10:00	2:00	14:00	0:45	1	0
All others					0	0

Table 1: Solution to small example

1.2 Skill sets for the large example

Agent type	Skill set	Agent type	Skill set
1	1,2,5,11,16	19	4
2	3,5,9,13,15,17,19	20	5
3	1,6,18,20	21	6
4	3,4,9,12,18	22	7
5	1,5,7,8,9,10,11,13,16	23	8
6	3,8,9,10,15,17,20	24	9
7	1,6,13,15	25	10
8	3,6,11,14,19,20	26	11
9	1,7,10,12,14	27	12
10	3,6,8,9,11,13,16	28	13
11	2,5,12,17	29	14
12	4,7,8,11,16,19,20	30	15
13	2,7,10,14,18,20	31	16
14	4,8,10,12,15,18,20	32	17
15	2,8,12,13,14,19	33	18
16	1	34	19
17	2	35	20
18	3		

Table 2: Skill sets for the large example.

1.3 CP-LP algorithm: Pseudocode

We will refer to the following linear integer problem:

$$
\begin{array}{ll}
\min & \mathbf{c}^{\mathrm{t}}\mathbf{x} \\
\text{s.t.} & \\
& \mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{z} \geq \mathbf{y} \\
& \text{[skill-supply constraints]} \\
& \mathbf{G}_u\mathbf{y} \geq \mathbf{g}_u \\
& \mathbf{x} \geq 0,\ \mathbf{z} \geq 0,\ \mathbf{y} \geq 0 \qquad \text{and integer}
\end{array}
\tag{1}
$$

where: $\mathbf{G}_u\mathbf{y} \geq \mathbf{g}_u$ denotes the set of cuts generated up to iteration u. The skill-supply constraints (discussed in the last paragraph of Section 3 of the paper) are not shown explicitly.
Algorithm 1 shows the complete algorithm, making reference to functions BinarySearch and LocalSearch. LocalSearch uses functions Phase1, Phase2, and Phase3 . In function BinarySearch, for any vector $\mathbf{x}$ of real numbers and $0 \leq \delta \leq 1$, $\mathbf{x}(\delta)$ denotes the vector whose i-th component is

$$
x_i(\delta) := \begin{cases} \lceil x_i \rceil, & x_i - \lfloor x_i \rfloor > \delta \\ \lfloor x_i \rfloor, & \text{otherwise} \end{cases}
\tag{2}
$$

where x_i denotes the i-th component of $\mathbf{x}$. In function Phase1, ArrivalRate(p,k) is the Poisson arrival rate of type-k calls during period p; and Occupancy$(\mathbf{y},a,p)$ is the simulated time-average number

of type-a busy agents divided by the number of type-a agents for period p under staffing $\mathbf{y}$. In function Phase3, Rand(S) denotes an element of the finite set S sampled randomly from the uniform distribution.

Algorithm 1: CP-LP

input : Call center and optimization parameters, simulator
output: Scheduling solution $\mathbf{x}$

begin
  $u \leftarrow 0$.
  Initialize problem (1) with $\mathbf{G}_u \leftarrow \emptyset$ and $\mathbf{g}_u \leftarrow \emptyset$.
  finished $\leftarrow$ **false**.
  while finished = **false do**
    Solve the relaxed linear problem (1) and **retrieve** solutions $\mathbf{x}^u$, $\mathbf{z}^u$ and $\mathbf{y}^u$.
    if *vector* $\mathbf{y}^u$ *is converging* **then** `// to avoid loop`
      finished = **true**.
    Round $\mathbf{x}^u$ and $\mathbf{y}^u$ using a fixed threshold δ and **round** down $\mathbf{z}^u$ (see section 1.3.1 for motivation).
    Correct solution $(\mathbf{x}^u, \mathbf{z}^u)$ such that $\bar{\mathbf{y}}^u = \mathbf{A}\mathbf{x}^u + \mathbf{B}\mathbf{z}^u$ has no negative values.
    Compute the service levels $\hat{g}_n(\mathbf{y}^u)$, $\hat{g}_{n,k}(\mathbf{y}^u)$, $\hat{g}_{n,p}(\mathbf{y}^u)$, and $\hat{g}_{n,k,p}(\mathbf{y}^u)$, for all k and p. **Compute** also for $\bar{\mathbf{y}}^u$. It is possible that $\mathbf{y}^u \neq \bar{\mathbf{y}}^u$ or $\mathbf{y}^u$ has no feasible corresponding solution $(\mathbf{x}^u, \mathbf{z}^u)$.
    if *the service levels with* $\mathbf{y}^u$ *OR* $\bar{\mathbf{y}}^u$ *satisfy all their targets* l, l_k, l_p *and* $l_{k,p}$ **then**
      finished $\leftarrow$ **true**.
    if finished = **false then**
      if *global service level :* $\hat{g}_n(\mathbf{y}^u) < l$ **then**
        Add associated cut to $\mathbf{G}_u$ and $\mathbf{g}_u$.
      else if *for some* k *:* $\hat{g}_{n,k}(\mathbf{y}^u) < l_k$ **then**
        Add associated cuts to $\mathbf{G}_u$ and $\mathbf{g}_u$.
      else if *for some* p *:* $\hat{g}_{n,p}(\mathbf{y}^u) < l_p$ **then**
        Add associated cuts to $\mathbf{G}_u$ and $\mathbf{g}_u$.
      else
        Add associated cuts to $\mathbf{G}_u$ and $\mathbf{g}_u$ (for some $(k,p) : \hat{g}_{n,k,p}(\mathbf{y}^u) < l_{k,p}$).
      Let $\mathbf{G}_{u+1} \leftarrow \mathbf{G}_u$, $\mathbf{g}_{u+1} \leftarrow \mathbf{g}_u$ and $u \leftarrow u+1$
  $(\mathbf{x}, \mathbf{z}) \leftarrow$ BinarySearch$(\mathbf{x}^u, \mathbf{z}^u)$.
  $\mathbf{x} \leftarrow$ LocalSearch$(\mathbf{x}, \mathbf{z})$.
end

1.3.1 Applying skill transfers

Because the algorithm rounds $\mathbf{x}$ and $\mathbf{z}$ independently, we may need to apply some correction (also in the local search where $\mathbf{z}$ remains fixed from the last **Solve** call). The application of skill transfers is limited to the condition that $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z}$ has no negative values.
The definitions of the supersets and subsets avoid the problem of looping (where an agent can be downgraded and re-upgraded) when applying the skill transfers. We present a simple algorithm to

retrieve the staffed agents vector $\mathbf{y}$ from the scheduling and skill transfer solutions $(\mathbf{x}, \mathbf{z})$.

Algorithm 2: GetStaffedAgents

input : $\mathbf{x}, \mathbf{z}$ and $\mathscr{S}^-$ (contains the strict maximum subsets of each agent group)
output: $\mathbf{y}$ (the actual number of staffed agents per group and per period)

```
1  begin
2      y ← Ax.
3      for each period p do
4          continue ← true.
5          while continue = true do
6              continue ← false.
7              for each agent group g do
8                  for each s ∈ 𝒮_g^- do
9                      if y_{g,p} > 0 and z_{g,s,p} > 0 then
10                         size ← min(y_{g,p}, z_{g,s,p}).
11                         y_{g,p} ← y_{g,p} − size.
12                         y_{s,p} ← y_{s,p} + size.
13                         z_{g,s,p} ← z_{g,s,p} − size.
14                         continue ← true.
15 end
```

Because $z_{g,s,p} = z_{s,g,p}$, we only modelize the skill transfer variables $z_{i,j,p}$ such that $j \in \mathscr{S}_i^-$ in our algorithm. A conservative estimation of the complexity is $O(IU)$, where I is the number of agent groups and $U = \sum_{p=1}^{P} \sum_{i=1}^{I} y_{i,p}$ is the sum of agents of each period.
We may better balance the rounding errors by implementing a randomized round-robin selection of "**for each** $s \in \mathscr{S}_g^-$ **do**" and a constant size = 1 (this would require an additional loop).

Note: Using such simple straightforward skill transfer correction might not be practical in certain cases. For example, suppose the staffing of a M-design call center composed of 2 call types, the specialist agent group 1 and 2, and a generalist agent group 3. Suppose the continuous solution is $y_1 = y_2 = 0, y_3 = 0.8$ and $z_{3,1} = z_{3,2} = 0.4$. Assuming a rounding threshold $\delta < 0.4$, we obtain the rounded values : $\bar{y}_3 = 1$ and $\bar{z}_{3,1} = \bar{z}_{3,2} = 1$. Applying *plainly* the skill transfers $\bar{\mathbf{z}}$, we obtain the solution $\hat{y}_1 = \hat{y}_2 = 1$ and $\hat{y}_3 = -1$, which is not feasible. If we correct using Algorithm 2, we would obtain $\hat{y}_1 = 1$ and $\hat{y}_2 = \hat{y}_3 = 0$. But this solution has *no agent* to serve calls of type 2! For this reason, we are more *conservative* when applying skill transfers by *rounding down* $\mathbf{z}$.
Better algorithms can be designed to further balance the skill transfers.

1.3.2 Generating the subgradient cut

At each call to **Solve** in algorithm 1, we retrieve the continuous solutions $\bar{\mathbf{x}}, \bar{\mathbf{z}}$ and $\bar{\mathbf{y}}$. In our experiments, we found it was more efficient to generate cuts based on $\bar{\mathbf{y}}$ instead of the staffed solution $\tilde{\mathbf{y}} = \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{z}} \geq \bar{\mathbf{y}}$.
Let $\hat{\mathbf{y}} = \mathit{Round}(\bar{\mathbf{y}})$ and $\mathbf{q}(\hat{\mathbf{y}})$ be a subgradient at point $\hat{\mathbf{y}}$ (calculated using forward finite difference method). The linear cut to add is :

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

We can see this approach as solving the relaxed version of the problem where we borrow the service levels and subgradients from the integer neighbors. This approach has the advantage of having lower chance to end in a loop as opposed to $\mathbf{q}(\hat{\mathbf{y}})^{\mathsf{t}}\mathbf{y} \geq \mathbf{q}(\hat{\mathbf{y}})^{\mathsf{t}}\hat{\mathbf{y}} + l - g(\hat{\mathbf{y}})$, but requires possibly more cuts (iterations).

1.3.3 Avoiding infinite loop

On rare occasions, it is possible that the algorithm enters a loop situation. When solving and rounding the solutions of the relaxed problem, it is possible to enter a loop (unless maybe if we always round up) because the subgradient cut may not actually *cut* the current solution. The next iteration may return the same solution and a loop is started.
Another possibility to loop is when the service level is within ε to the target and the solution converges *very slowly* (or sometimes indefinitely because of rounding errors of the machine).
We propose two easy solutions :

1. Detect converging solution. After each **Solve** call in Algorithm 1, check if every element $y_{i,p}^{u+1}$ has changed less than ε from $y_{i,p}^{u}$. A shortcut is to perform the check only if $|cost(\mathbf{x}^{u+1}) - cost(\mathbf{x}^{u})| < \varepsilon$. If convergence is detected, stop the algorithm and continue with binary search and local search.

2. Set a minimal improvement when generating a subgradient cut : $\mathbf{q}(\hat{\mathbf{y}})^{\mathsf{t}}\mathbf{y} \geq \mathbf{q}(\hat{\mathbf{y}})^{\mathsf{t}}\bar{\mathbf{y}} + \Delta$, where $\Delta = \max\{l - g(\hat{\mathbf{y}}), \varepsilon\}$ and ε is a small number, e.g.: 0.005.

1.3.4 Binary search

After completing the subgradient cut phase, we correct the rounding threshold parameter δ with binary search.

Algorithm 3: BinarySearch

```
   input  : x, z
   output: x, z
 1 begin
 2     δ_L ← 0, δ_U ← 1, δ_Best ← δ_L.
 3     while δ_U − δ_L > 0.01 do
 4         δ ← (δ_U + δ_L)/2.
 5         Round (x, z, δ).
 6         y ← GetStaffedAgents (x, z). (see Algorithm 2)
 7         Simulate solution y.
 8         if y is feasible then
 9             δ_Best ← δ.
10             δ_L ← δ + 0.01.
11         else
12             δ_U ← δ- 0.01.
13         Round (x, z, δ_Best).
14 end
```

The procedure Round does floor rounding on $\mathbf{z}$ (it does not depend on parameter δ).

1.3.5 Local search

A local search procedure is executed at the end of CP-LP. The same local search is used for CP-IP without the rounding steps. A time-limit is used which terminates the algorithm at any step once this limit is reached.

Algorithm 4: LocalSearch

input : $\mathbf{x}, \mathbf{z}$
output: $\mathbf{x}$

$n_1 \leftarrow n_2$.
$p \leftarrow 0$.
begin
  do
    $n_1 \leftarrow \min\{n, \mathsf{Round}((1+p)n_2)\}$.
    $\mathbf{x} \leftarrow$ `Phase1` $(\mathbf{x}, \mathbf{z}, n_1)$.
    $\mathbf{x} \leftarrow$ `Phase2` $(\mathbf{x}, \mathbf{z}, n_1)$.
    $\mathbf{x} \leftarrow$ `Phase3` $(\mathbf{x}, \mathbf{z}, n_1)$.
    $p \leftarrow p + 0.5$.
    $\mathbf{y} \leftarrow$ `GetStaffedAgents` $(\mathbf{x}, \mathbf{z})$.
    Simulate $\mathbf{y}$ with n days.
  while $n_1 < n$ *AND* $\mathbf{y}$ *is infeasible for* $SP0_n$.
end

In **Phase1** (Algorithm 5), sets are used to treat ties simultaneously.

1.4 Quality of IP solutions in CP-IP and TS

IP instances for CP-IP and TS are solved using CPLEX 9.0. CPLEX uses branching methods to solve the IP. For practical reasons, a time limit is set and the solver usually stops before claiming optimality. The quality of the returned integer solution can be measure by its *MIP gap*, defined as :

$$\frac{c_{\text{INT}} - c_{\text{NODE}}}{c_{\text{INT}} + \varepsilon} \times 100,$$

where c_{INT} is the cost of the returned integer solution, c_{NODE} is the cost of the best node in the branches and ε is a very small positive number. The MIP gap measures the *potential* cost improvement over the returned integer solution.
In summary,

- For the small call center, the solutions of TS have a MIP gap smaller than 0.3% and the IP instances of CP-IP generally have a MIP gap of less than 1.5%.
- For the medium-sized problems, the solutions of TS have a MIP gap smaller than 1.5% and IP instances of CP-IP can sometimes have a MIP gap up to 10% because of the short CPLEX time limit.
- For the large-sized problems, the solutions of TS have a MIP gap smaller than 0.03% for L_{36} and 0.001% for L_{52}. The problems were too large to be solve by CP-IP.

Algorithm 5: Phase1

input : $\mathbf{x}, \mathbf{z}, n$
output: $\mathbf{x}$

begin
  $\mathbf{y} \leftarrow$ `GetStaffedAgents` $(\mathbf{x}, \mathbf{z})$.
  Simulate $\mathbf{y}$ with n days.
  while $\mathbf{y}$ *is not feasible for* $SP0_n$ **do**
    $V \leftarrow \{k | \hat{g}_{n,k}(\mathbf{y}) < l_k\}$.
    if $V = \emptyset$ **then**
      $(a,p) \leftarrow \arg\max_{a,p}\{\texttt{Occupancy}(\mathbf{y}, a, p)\}$. (in case of ties : take cheapest agent)
      $L \leftarrow \arg\min_k\{\hat{g}_{n,k}(\mathbf{y})\}$. ($L$ is a set)
    else
      $a \leftarrow \arg\max_a\{|S_a \cap V|\}$.
      $L \leftarrow \arg\min_k\{\hat{g}_{n,k}(\mathbf{y}) - l_k\}$. ($L$ is a set of call types)
    $PV \leftarrow \cup_{k \in L} \arg\max_p\{\texttt{ArrivalRate}(p,k)\}$. ($PV$ is a set of periods)
    $Q' \leftarrow \arg\max_s \sum_{p \in PV} \mathbf{A}_0(p,s)$. ($Q'$ is a set of shifts)
    $s \leftarrow$ `Rand` (Q').
    $x_{a,s} \leftarrow x_{a,s} + 1$.
    $\mathbf{y} \leftarrow$ `GetStaffedAgents` $(\mathbf{x}, \mathbf{z})$.
    Simulate $\mathbf{y}$ with n days.
end

1.5 The indexing of shifts

In all the examples with 285 shifts, the order followed for sorting them is: the ones 30 periods long; the ones 31 periods long; the ones 32 periods long; the ones 33 periods long; the 34 periods long; the 36 periods long and finally the ones 26 periods long. For the shift definition for the ones with the same length, the order is: the shift 1 has the first element in S, the first in D_1, the first in P_2 and the first in D_3; the shift 2 has the first in S, the first in D_1, the first in P_2 and the second in D_3; the shift 3 has the first in S, the first in D_1, the first in P_2 and the third in D_3 and so on.

Algorithm 6: Phase2

input : $\mathbf{x}, \mathbf{z}, n$
output: $\mathbf{x}$

begin
  for each *agent group* $a \in I$ **do**
    for each *shift* $s \in Q$ **do**
      if $x_{a,s} > 0$ **then**
        $x_{a,s} \leftarrow x_{a,s} - 1$.
        $\mathbf{y} \leftarrow$ `GetStaffedAgents` $(\mathbf{x}, \mathbf{z})$.
        Simulate y with n days.
        if y *is infeasible for* $SP0_n$ **then**
          $x_{a,s} \leftarrow x_{a,s} + 1$.
end

Algorithm 7: Phase3

input : $\mathbf{x}, \mathbf{z}, n$
output: $\mathbf{x}$

begin
  $i \leftarrow 0$.
  maxTries $\leftarrow 40$.
  while $i <$ maxTries **do**
    do
      $a1 \leftarrow$ `Rand` $(\{1, 2, \ldots, K\})$. (group to remove agent)
      $s1 \leftarrow$ `Rand` $(\{1, 2, \ldots, Q\})$.
      $a2 \leftarrow$ `Rand` $(\{1, 2, \ldots, K\})$. (group to add agent)
      $s2 \leftarrow$ `Rand` $(\{1, 2, \ldots, Q\})$.
    while $c_{a1,s1} < c_{a2,s2}$ *OR* $x_{a1,s1} = 0$
  $x_{a1,s1} = x_{a1,s1} - 1$.
  $x_{a2,s2} = x_{a2,s2} + 1$.
  $\mathbf{y} \leftarrow$ `GetStaffedAgents` $(\mathbf{x}, \mathbf{z})$.
  Simulate y with n days.
  if y *is infeasible for* $SP0_n$ **then**
    $x_{a1,s1} = x_{a1,s1} + 1$.
    $x_{a2,s2} = x_{a2,s2} - 1$.
    $i \leftarrow i + 1$.
  else
    $i = 0$.
end