In cooperation with:
ProjectsOne
De Corridor 12L
3621 ZB Breukelen

For the degree of Master of Science in Applied Mathematics at the Department of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology

Thesis Committee:
Chair: Prof. dr. ir. G. Jongbloed, Faculty EEMCS (Statistics), TU Delft
University supervisor: dr. ir. Frank van der Meulen, Faculty EEMCS (Statistics), TU Delft
University member of a different department: dr. D.C. Gijswijt, Faculty EEMCS (Optimization)
Company supervisor: Melvin Harteveld, ProjectsOne
Abstract

In this thesis, a model for a staff scheduling problem within a multi skilled environment is constructed. It is shown that a cost optimal planning can be obtained by Simulated Annealing.

The resulting algorithm is analyzed and applied to a toy example and a scheduling problem of a Dutch railway construction company.

Contents

| 1 Introduction | 5 |
| 2 A Model for multi skilled staff scheduling | 5 |
| 2.1 Staff Members and Skills | 5 |
| 2.2 Tasks and Required Skill | 6 |
| 2.3 Assignment of Staff Members to Tasks | 7 |
| 3 Planning Costs | 11 |
| 4 Cost Function | 12 |
| 5 The Assignment Problem | 13 |
| 5.1 Problem formulation — Assignment Problem | 13 |
| 5.2 Special Case where $I = 1$ — Maximum Weight Independent Set | 14 |
| 5.3 Special Case — Set Covering Problem | 15 |
| 5.3.1 Set Covering Problem | 15 |
| 5.3.2 Associated Assignment Problem for a SCP | 15 |
| 5.4 General Case | 17 |
| 6 Simulated Annealing | 17 |
| 6.1 The Metropolis-Hasting Algorithm | 18 |
| 6.2 Metropolis-Hasting with $\exp(-g)$ | 21 |
| 6.3 Simulated Annealing Algorithm | 22 |
| 6.4 Simulated Annealing to generate a Markov Chain | 24 |
| 6.4.1 Convergence Speed and Conditions | 25 |
| 6.5 SA and Constrained Minimization | 29 |
| 7 SA applied on the Assignment Problem | 30 |
| 7.1 Neighborhood Structure | 30 |
| 7.1.1 Spray Neighborhood Outline | 31 |
| 7.1.2 The Spray Neighborhood Structure | 32 |
| 7.1.3 Suggestion set $\mathcal{J}$ | 32 |
| 7.1.4 Weights for the suggestions set $\mathcal{J}$ | 33 |
| 7.2 Adjustments for the (SI) and (WR) condition | 37 |
| 7.2.1 Clean-and-Spray Neighborhood structure | 38 |
| 7.2.2 Mixed-Spray Neighborhood structure | 39 |
7.3 Parameters which define a SA run for the assignment problem

8 Test Case 1 — Simple Example
8.1 Problem Formulation
  8.1.1 Work matrix and Available skill matrix
  8.1.2 Cost function
  8.1.3 A global minimum
8.2 SA parameters which found global minima
8.3 Variations in the Cooling Schedule
8.4 Variations in the Initial State
8.5 Variations in the Stopping Criteria
8.6 Improve the result for \( N_p = 10 \)

9 Global recommendations for setting the SA parameters
9.1 Initial State
9.2 \( N_p, \alpha, T_0 \) and \( T_{stop} \)
9.3 Weights and \( k \)

10 Test Case 2 — Computer (SA) versus Humans
10.1 Problem Formulation
  10.1.1 Required Skill matrix and Available Skill matrix
  10.1.2 Cost function
10.2 Results

11 Test Case 3 — The Railway case
11.1 Current Situation
  11.1.1 Constraints
11.2 Railway Test Case Planning Setup and Modifications
11.3 Soft and Hard constraints
  11.3.1 Travel-Overlap Matrix \( T \)
  11.3.2 Deal with the **internal/external balance**
  11.3.3 The neighborhood structure modifications
  11.3.4 Weights and vacation requests
  11.3.5 Weights
11.4 Railway Case SA parameters
11.5 Quality of output planning
11.6 200 Re-Runs
  11.6.1 Results with all internal staff available
11.7 Impact on working with 10 less staff members
11.8 Implementation in Planning Software

12 Conclusions & Recommendations for Future Work

A Matrices for Test Case 1
B Matrices for Test Case 2
Notation Conventions

For vector $x, y \in \mathbb{R}^N$, and $I \times J$ matrices $A = [a_{i,j}], B = [b_{i,j}]$, we use the following notation conventions:

- $x_n$ -or- $x[n]$ \hspace{1cm} $n$th component of vector $x$.
- $x'$ \hspace{1cm} The transpose of vector $x$.
- $A'$ \hspace{1cm} The transpose of matrix $A$.
- $A_{i,j}$ -or- $A[i,j]$ \hspace{1cm} Element $i, j$ of matrix $A$.
- $x \preceq y$ \hspace{1cm} If $x_n \leq y_n$, for all $n = 1, 2, \ldots, N$.
- $x \prec y$ \hspace{1cm} If $x_n < y_n$, for all $n = 1, 2, \ldots, N$.
- $A \preceq B$ \hspace{1cm} If $a_{i,j} \leq b_{i,j}$, for all $1 \leq i \leq I, 1 \leq j \leq J$. (component wise)
- $A \prec B$ \hspace{1cm} If $a_{i,j} < b_{i,j}$, for all $1 \leq i \leq I, 1 \leq j \leq J$. (component wise)
- $\preceq$ and $\prec$ defines a partial ordering on $\mathbb{R}^n$.
- $\text{Col}_j(A)$ \hspace{1cm} $j$th column of matrix $A$.
- $\text{Row}_i(A)$ \hspace{1cm} $i$th row of matrix $A$, as a column vector.
- $\|A\|_{\text{max}}$ \hspace{1cm} The matrix max-norm, i.e. $\max_{i,j} |a_{i,j}|$.
- $\|x\|_1$ \hspace{1cm} The vector 1-norm, i.e. $\sum_{i=1}^N |x_i|$.
- $0_{I \times J}$ \hspace{1cm} $I \times J$ zero matrix.
- $1_{I \times I}$ \hspace{1cm} $I \times I$ identity matrix.
- $\text{diag}(x)$ \hspace{1cm} The diagonal matrix, where $\text{diag}(x)[n,n] = x_n$.
- $\text{Col-max}(A)$ \hspace{1cm} A vector of length $J$, where element $j = \|\text{Col}_j(A)\|_{\text{max}}$.
- $\text{Row-max}(A)$ \hspace{1cm} A vector of length $I$, where element $i = \|\text{Row}_i(A)\|_{\text{max}}$.
- $\lceil \cdot \rceil$ \hspace{1cm} The ceiling function, e.g. $\lceil 3/4 \rceil = 1$.
- $\lfloor \cdot \rfloor$ \hspace{1cm} The floor function, e.g. $\lfloor 3/4 \rfloor = 0$.

Table 1: Notation Conventions
1 Introduction

Staff scheduling, where staff members are assigned to tasks, is a crucial part of staff management. Overstaffing is costly as is understaffing. The right persons, at the right place, at the right time, is the key for successful staff scheduling.

The importance of staff scheduling is reflected in the many reported investigations in the last decades in the literature [3] [1] [11].

In this thesis we focus on staff scheduling in a multi skilled environment where not necessarily each staff member is suitable for every task. Examples of multi skilled environments are: schools and universities where teachers/professors only lecture within their own field, and hospitals where each staff member has his/her own specialization.

In such planning environments, planners often assign the staff manually. The available planning software is mostly of managing/visual aid.

In this paper we try to mathematically formalize a staff scheduling situation, within a multi skilled environment. Subsequently, we try to answer the following question:

- Given a group of staff members, and a set of tasks, how can we assign each staff member to tasks most efficiently?

Here, ‘most efficiently’ could be with the lowest cost, or with the preferences of staff members taken into consideration. If the above question can be answered, we can present planners with a tool to help them improving the quality of their planning.

For a Dutch railway construction company, where planners often struggle scheduling the construction staff, the resulting planning tool will be tested and compared against their current working method.

2 A Model for multi skilled staff scheduling

2.1 Staff Members and Skills

In staff scheduling, planners assign staff members to tasks or shifts. Inside a multi skilled environment, each staff member can possess a different set of skills. We assume a total of $I$ staff members, and enumerate the staff by $1, \ldots, I$. Each of the $I$ staff members can possess some of the $N \in \mathbb{N}$ skills.

To each staff member, we can therefore attach a skill vector.

**Definition.** For the $i^{th}$ staff member, we define its **skill vector** $s_i \in \mathbb{R}^N$ to be a vector with each element either equal to 0 or 1. Let $n \in \{1, \ldots, N\}$. If the $n^{th}$ component of $s_i$ equals one, this means that the $i^{th}$ staff member possesses skill $n$, where a zero would mean the staff member does not possess skill $n$. 


The skills of all staff members can conveniently be summarized in an available skill matrix.

**Definition.** The available skill matrix $S$ is the $N \times I$ matrix defined by

$$S = [s_1, s_2, \ldots, s_I],$$

so the $i^{\text{th}}$ column of $S$ corresponds to the skill vector of the $i^{\text{th}}$ staff member.

**Example 1** (A music band with 5 band members). Suppose a band consists of $I = 5$ (staff) members, where each of the members can possess one of more of the $N = 4$ possible skills: drum$^{(1)}$, guitar$^{(2)}$, vocal$^{(3)}$, and keyboard$^{(4)}$. With the skill vectors defined as follows,

$$s_1 = (1, 0, 0, 0)^t$$
$$s_2 = (0, 1, 0, 1)^t$$
$$s_3 = (0, 1, 1, 0)^t$$
$$s_4 = (0, 0, 1, 1)^t$$
$$s_5 = (1, 0, 1, 0)^t,$$

staff member 1 is can play drums, staff member 2 can play guitar and keyboard. For the other staff members one can similarly read of their skills. The available skill matrix is given by

$$S = [s_1, s_2, s_3, s_4, s_5] = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}. $$

### 2.2 Tasks and Required Skill

**Definition.** For fixed $N \in \mathbb{N}$, a task $\bar{w}$ is defined by a 3-tuple, $\bar{w} = (a, b, w)$. $a, b \in \mathbb{R}^+$ describes the start and end time of the task respectively, with $a < b$. Vector $w \in \{0, 1, \ldots\}^N$, the required skill vector, describes the required skills for task $\bar{w}$, in type and quantity. If $w_n = \ell$, then the task requires $\ell$ staff members possessing the $n^{\text{th}}$ skill.

**Definition.** Let $w_1, w_2, \ldots, w_J$ be $J$ required skill vectors. Define the $N \times J$ required skill matrix $W$ by

$$W = [w_1, w_2, \ldots, w_J].$$

The required skill matrix $W$ does not contain any information about the length and starting times of the tasks. The task-time vector, contains the lengths of the $J$ tasks, and is defined by

$$t = (b_1 - a_1, \ldots, b_J - a_J).$$
Example 2 (Tasks and the required skill matrix, continuation of example 1). Define the 4 tasks as

\[
\bar{w}_1 = (9, 12, (1, 0, 0, 0)') \\
\bar{w}_2 = (9, 12, (0, 1, 1, 0)') \\
\bar{w}_3 = (9, 12, (0, 1, 0, 0)') \\
\bar{w}_4 = (9, 12, (0, 0, 1, 1)')
\]

These tasks corresponds to playing certain music from time step 9 till 12. Task 1 requires a drummer. Task 2 and 3 requires a guitar player and a singer. The last task requires a keyboard player and a singer. The required skill matrix is given by

\[
W = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{pmatrix},
\]

and the task-time vector is \( t = (3, 3, 3, 3)' \).

2.3 Assignment of Staff Members to Tasks

The assignment of staff members to tasks is done with the help of the assignment matrix.

Definition. The assignment matrix \( A = [a_{i,j}] \) is an \( I \times J \) matrix, with components in \( \{0, 1\} \). Elements of the assignment matrix have the interpretation:

\[
a_{i,j} = \begin{cases} 
1 & \text{if staff member } i \text{ is assigned to task } j. \\
0 & \text{if staff member } i \text{ is not assigned to task } j.
\end{cases}
\]

In staff scheduling, each staff member can only work on one task at a time. This motivates the following definition:

Definition. We say the tasks \( \bar{w}_j = (a_j, b_j, w_j) \) and \( \bar{w}_k = (a_k, b_k, w_k) \) have overlap, if

\[
[a_j, b_j] \cap [a_k, b_k] \neq \emptyset.
\]

All overlapping tasks can easily be read of from the overlap matrix, which we now define.

Definition. Define the \( J \times J \) overlap matrix \( P = [p_{j,k}] \) as

\[
p_{j,k} = \begin{cases} 
J & \text{if } j = k, \\
1 & \text{if task } j \text{ and } k \text{ overlap, for } j \neq k \\
0 & \text{else.}
\end{cases}
\]
The choice of the value $J$ on the diagonal is not immediately clear, but this will turn out to be a useful choice later on.

**Example 3** (Overlapping tasks). Suppose we are given the tasks

$\bar{w}_1 = (0, 2, w_1)$  
$\bar{w}_2 = (3, 6, w_2)$  
$\bar{w}_3 = (4, 9, w_3)$

then tasks $\bar{w}_2$ and $\bar{w}_3$ overlap. These are the only overlapping tasks.

The overlap matrix $P$ for this example is

$$P = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 1 \\ 0 & 1 & 3 \end{pmatrix}.$$

**Definition.** An assignment matrix $A$ is called **consistent** if no staff member $1 \leq i \leq I$ is assigned to overlapping tasks.

Consistency of an assignment matrix can easily be checked with help of the overlap matrix $P$, as shown by the following lemma.

**Lemma 1.** The assignment matrix $A$ is consistent for overlap matrix $P$ if and only if

$$\| AP \|_{\max} \leq J.$$

For matrix $B = [b_{i,j}]$, the max norm $\|B\|_{\max} = \max_{i,j} |b_{i,j}|$.

**Proof.** ($\Leftarrow$) First assume $A$ is not consistent. Then there exists a staff member $i$ which is assigned to at least 2 tasks with overlap, say task $j$ and $k$. Therefore by definition we have $\text{Row}_i(A) = (a_1, a_2, \ldots, a_J)'$, with $a_j = 1$ and $a_k = 1$. Furthermore we have $\text{Col}_j(P) = (p_1, p_2, \ldots, p_J)'$, with $p_j = J$ and $p_k = 1$ by definition of the overlap matrix $P$. For element $i, j$ of the matrix product $AP$ we get

$$(AP)_{i,j} = \sum_{m=1}^{J} a_m \cdot p_m \geq a_j \cdot p_j + a_k \cdot p_k = J + 1.$$  

($\Rightarrow$) If $\| AP \|_{\max} > J$, there exists $i, j$ such that

$$(AP)_{i,j} = \text{Row}_i(A) \cdot \text{Col}_j(P)' = \sum_{m=1}^{J} a_m \cdot p_m > J,$$

Figure 1: Graphical representation of tasks $w_1, w_2,$ and $w_3$. 

The overlap matrix $P$ for this example is
where \( p_j = J \). Note that for each term \( a_m \cdot p_m \) we have

\[
0 \leq a_m \cdot p_m \leq \begin{cases} 
1 & \text{for } m \neq j \\
\frac{J}{m = j}. 
\end{cases}
\]

The sum of \( J \) terms can only exceed \( J \) if at least one term is greater than 1. Therefore \( a_j p_j = J \), and \( a_k p_k = 1 \) for some \( k \).

In other words, staff member \( i \) is assigned to overlapping tasks \( j \) and \( k \), and thus matrix \( A \) is not consistent.

\( AP \) provides us more useful information. When \( (AP)_{i,j} = J \), staff member \( i \) is assigned to task \( j \), but not to any overlapping tasks. If \( (AP)_{i,j} > J \), staff member \( i \) is assigned to task \( j \), and to at least one task that causes overlap with task \( j \).

**Example 4** (Consistency of \( A \) in terms of the overlap matrix \( P \)). Let \( \bar{w}_1, \bar{w}_2, \bar{w}_3 \) be as in example 3 (fig. 1), with overlap matrix

\[
P = \begin{pmatrix}
3 & 0 & 0 \\
0 & 3 & 1 \\
0 & 1 & 3
\end{pmatrix}.
\]

Define the assignment matrix \( A \) as,

\[
A = \begin{pmatrix}
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1
\end{pmatrix}.
\]

Here, we have

\[
\|AP\|_{\max} = \begin{bmatrix}
3 & 3 & 1 \\
3 & 4 & 4 \\
3 & 1 & 3
\end{bmatrix} = 4.
\]

Since \( \|AP\|_{\max} > 3 \), matrix \( A \) is inconsistent. Since \( AP_{2,2} > J \) and \( AP_{2,3} > J \), staff member 2 causes overlap by being assigned to task 2 and 3.

The following properties of the overlap matrix \( P \) are easily verified:

- \( P \) is symmetric, since ‘overlapping’ defines an equivalence relation.
- If \( P = J \cdot \mathbb{1}_{J \times J} \), no tasks overlap, and \( A \) is always consistent.

\( SA \) is called the **assigned skill matrix**, and is a \( N \times J \) matrix.

\[
\text{Col}_j(SA) = SA_j = \sum_{0 \leq i \leq I} s_i a_{ij}.
\]

The \( j \)th column of \( SA \) is the sum of the skill vectors corresponding to the staff members which are assigned to task \( j \), and interprets as the sum of workforce on task \( j \).
The $N \times J$ assigned skill matrix $SA$, can be compared to the $N \times J$ required skill matrix $W$. If for $n, j$ we have

$$SA[n, j] < W[n, j],$$

task $j$ requires more staff members possessing the $n$th skill. If for task $j$ we have

$$SA[n, j] \geq W[n, j] \text{ for } 1 \leq n \leq N,$$

we say we meet all the required skills for task $j$.

**Example 5** (Assigned skill matrix, continuation of example 2). With $S$ and $W$ as in example 1, 2, we define the assignment matrix as

$$A = \begin{pmatrix}
1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix}.$$ 

The assigned skill matrix becomes

$$SA = \begin{pmatrix}
1 & 1 & 1 & 0 \\
0 & 2 & 0 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0
\end{pmatrix}.$$ 

The required skill matrix from example 2 reads

$$W = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1
\end{pmatrix}.$$ 

For task 1 and 2 all the required skills are met, but not for task 3 and 4. This can be verified by comparing the columns.

All the introduced notation is given in the planning setup below.
Planning Setup

$I$ - The number of staff members.

$J$ - The number of tasks.

$N$ - The number of skills, each staff member can possibly possess.

$s_i \in \{0, 1\}^N$ - The skill vector of the $i^{th}$ staff member, indicating which of the $N$ skills he/she possess.

$S_{N \times I} = [s_1, \ldots, s_I]$ - The available skill matrix, containing the $I$ skill vectors.

$\bar{w}_j = (a_j, b_j, w_j)$ - The $j^{th}$ task, where real numbers $a_j < b_j$ indicate the start/end times, and vector $w_j \in \mathbb{N}^N$ is the required skill vector.

$t = (b_1 - a_1, \ldots, b_J - a_J)$ - the task-time vector, containing the lengths of the $J$ tasks.

$W_{N \times J} = [w_1, \ldots, w_J]$ - the required skill matrix, containing the $J$ required skill vectors.

$P_{J \times J}$ - the overlap matrix, where $P_{j,k} = 1$ if task $j$ and $k$ overlap, and 0 otherwise.

$A_{I \times J}$ - the assignment matrix, where $A_{i,j} = 1$ if staff member $i$ is assigned to task $j$, and 0 otherwise.

$S_{A_{N \times J}}$ - the assigned skill matrix, where column $j$ of $SA$, is the sum of the assigned skill to task $j$.

3 Planning Costs

There is not always the right staff available, or enough. Consider the following example, where not enough skills are available to meet all the required skill vectors.

Example 6. Consider a staff with a total of $I = 4$ employees, with available skill matrix

$$ S = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, $$

and required skill matrix

$$ W = \begin{pmatrix} 5 \\ 5 \end{pmatrix} . $$

Assume the tasks don’t have overlap, hence $P = 2 \cdot \mathbb{I}_{2 \times 2}$. Assume both tasks have a length of 1, so $t = (1, 1)'$. 

11
The best a planner could possibly do to meet the required skill, is to assign
the whole staff to both tasks, e.g.

\[
A = \begin{pmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{pmatrix},
\]

which is a consistent assignment matrix, resulting in the following difference
between the assigned skill and required skill:

\[
SA = \begin{pmatrix}
3 & 3 \\
3 & 3 \\
3 & 3 \\
\end{pmatrix} \preceq \begin{pmatrix}
5 & 5 \\
5 & 5 \\
5 & 5 \\
\end{pmatrix} = W.
\]

For \( I \times J \) matrix \( A, B \) we have \( A \preceq B \) if \( a_{i,j} \leq b_{i,j} \), for all \( 1 \leq i \leq I, 1 \leq j \leq J \). This give a difference between the assigned and required vector of \((2, 2, 2)'\) for both tasks.

If in real planning situations the work is overwhelming like in the previous
example, the planner could choose to hire extra staff, temporary staff, or
make a concession in the quality of work. The extra costs of hiring the
temporary staff, or the quality reduction, will be modelled in a cost function.
Depending on the skill, the cost for hiring staff can vary. Also, hiring staff
for a longer period of time makes costs more. The cost function models
these concepts, and uses the difference between required skill and assigned
skill, and the length of the task, to define the planning costs.

4 Cost Function

Definition. Define the skill penalty vector as \( p = (p_1, p_2, \ldots, p_N)' \). For
assignment matrix \( A \), we define the cost function as

\[
C(A) = p' \max(W - SA, 0_{J \times N})t,
\]

where \( \max(\cdot, \cdot) \) is taken element-wise, and maps to a \( J \times N \) matrix in the
above expression.

Some justification for the above formula; matrix \( \max(W - SA, 0_{J \times N}) \) con-
tains positive elements on column \( j \), if for task \( j \) there are not enough skills
assigned. Vector \( p \) weights all these not met skills. Time vector \( t \) finally
scales the result, proportional to the corresponding task times.

It is easy to verify that for assignment matrices \( A_1 \) and \( A_2 \) with \( A_1 \preceq A_2 \),
we have

\[
C(A_1) \geq C(A_2).
\]
Example 7 (Cost Function). In example 6, we found the following difference between required skill and assigned skill:

\[ W - SA = \begin{pmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{pmatrix}. \]

With \( t = (1, 1)' \) and \( p = (10, 10, 20)' \), the costs become

\[
p' \max(W - SA, 0_{J \times N}) t = (10, 10, 20) \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} (1, 1)' \]
\[ = 160 \]

These costs could represent hiring 2 extra staff members possessing skill vector \( (1, 1, 1)' \), for both tasks with a total duration of 2 time steps. With the penalty vector \( p = (10, 10, 20)' \), the hire costs for each skill can be read, and skill vector \( (1, 1, 1)' \) would cost 40 per time step. Hiring these 2 extra staff members, each paying 40 a time step, for a total of 2 time steps results in a total extra cost of 160.

5 The Assignment Problem

In the assignment problem, we try to assign a set of staff members to a set of tasks such that the costs are minimized.

In terms of the model, for fixed \( W \) and \( S \), a consistent assignment matrix \( A \) has to be found, such that the cost are minimized.

5.1 Problem formulation — Assignment Problem

The assignment problem is formulated as:

Find an \( I \times J \) binary assignment matrix \( A \), such that

\[ p' \max(W - SA, 0_{J \times N}) t \]

is minimized under the condition

\[ \|AP\|_{\text{max}} \leq J. \]

Since there are finitely many \( I \times J \) assignment matrices \( A \), a minimizing matrix \( A^* \) always exists. The minimum \( A^* \) may not be unique. Without the condition, we minimize over a space containing \( 2^{IJ} \) elements. The subset that satisfies the condition, is never empty, since it always contains the \( I \times J \) zero matrix.

Before embarking the general problem, we first consider a simple case of the stated optimization problem, where only one staff member is scheduled.
5.2 Special Case where $I = 1$ — Maximum Weight Independent Set

Firstly, we consider the case with one staff member ($I = 1$). The problem simplifies to find a subset $L \subset \{1, \ldots, J\}$ of none-overlapping tasks, that minimizes the cost. The assignment matrix $A$ becomes a binary vector of length $J$.

The maximum cost $M$ is obtained with the zero assignment vector $A = 0_{1 \times J}$. This is obtained by not assigning the staff member to any task, where the cost becomes

$$M = C(0_{1 \times J}) = p'Wt.$$ 

If we assign the staff member only to task $j$, we can calculate the resulting cost reduction. That is, the difference between $M$ and the cost with the staff member assigned to task $j$. Call this positive cost reduction $d_j$ and define the non-negative vector $d$ as:

$$d = (d_1, d_2, \ldots, d_J)'$$

For more than one staff members, this reduction can depend on the assignment of the other staff members.

Minimizing the cost is maximizing the cost reduction $d'A$. Since the cost cannot be negative, we can rewrite the costs as

$$C(A) = \max(M - d'A, 0).$$

The problem can be now formulated as

$$\begin{align*}
\text{maximize} & \quad d'A \\
\text{subject to} & \quad PA \leq J \\
& \quad \text{and } A \text{ is a binary vector}
\end{align*}$$

This problem is known in graph-theory as finding the 'maximum weight independent set' (MWIS). In the corresponding undirected graph each node represents a task, where edges between nodes indicates that the corresponding tasks overlap. For arbitrary graphs, the MWIS is a NP-hard optimization problem [7]. The graph induced by the assignment problem is an interval graph. On interval graphs, polynomial time algorithms exists which solves the corresponding optimalisation problem[10].

For $I > 1$ staff members, the MWIS problem also raises if we ask ourselves how to optimally assign one staff member, given that the assignment of the other $I - 1$ staff members stay fixed. Indeed, the cost reduction $d_j$ for each tasks $j$ remains constant.

[10] provides an algorithm which can be used to optimally assign a single staff member. For more than 1 staff members, different solving methods are required, since the problem is of a different kind as shown in the next section.
5.3 Special Case — Set Covering Problem

5.3.1 Set Covering Problem

Let $U = \{1, 2, 3, \ldots, D\} \subset \mathbb{N}$ (universe), and $V_1, V_2, \ldots, V_I \subset U$ be a family of $I$ subsets, which cover $U$:

$$\bigcup_{1 \leq i \leq I} V_i = U.$$ 

In the Set Covering Problem (SCP), one tries to find a subset of indices $Z \subset \{1, 2, \ldots, I\}$ which cover $U$:

$$\bigcup_{i \in Z} V_i = U.$$ 

The corresponding (unweighted) optimization problem, where the number of covering sets $|Z|$ is minimized, is NP-hard [9]. Each optimization SCP corresponds to an assignment problem with 2 tasks, showing the assignment problem is a NP-hard scheduling problem. We assume $V_i \neq \emptyset$ for $1 \leq i \leq I$. We now proceed showing how to construct an assignment problem, given a SCP.

5.3.2 Associated Assignment Problem for a SCP

Consider $I$ staff members, each possessing some of the $\{1, 2, \ldots, D, D + 1\}$ skills ($N = D + 1$). Staff member $i$ possess skills $V_i \cup \{N\}$, and corresponds to subset $V_i$ in the associated SCP. We define $J = 2$ overlapping tasks. $w_1$, the required skill vector for task 1 has its first $D$ elements 1, and its last element 0. $w_2$, the required skill vector for task 2 has its first $D$ elements 0, and its last element $I$.

$$w_1 = (1, 1, \ldots, 1, 1, 0),$$

$$w_2 = (0, 0, \ldots, 0, 0, I).$$

Let the penalty vector be such that the first $D$ skills are weighted more than the last one, e.g. $p = (2, 2, \ldots, 2, 2, 1)$ and task-time vector $t = (1, 1)$. Let $A^*$ be a solution for the assignment problem. The subset of staff members assigned to task 1 in $A^*$, corresponds to the subsets in the SCP, which cover $U = \{1, 2, 3, \ldots, D\}$. If in $A^*$, $l$ staff members are assigned to task 1, then
the minimum number of subsets needed to cover $U$ is $l$. If in the SCP $l$ subsets is the minimum number of subsets which can cover $U$, then the $l$ subsets corresponds to the $l$ staff members, assigned to task 1 for some minimizing assignment matrix.

We show this by reason as follows; for a minimizing assignment matrix $A^*$ each staff member is either assigned to task 1 or 2, but not to both since these are overlapping tasks. If for any consistent assignment matrix, staff member $i$ is assigned to task 2, re-assigning him to task 1 would only result in a reduction of cost, if some of his skills ($V_i \cup \{N\}$) are required by task 1, but are not yet meet. Only then assigning him to task 1 has a positive impact on the cost, by design of the penalty vector which weights the first $D$ elements more than the last. This shows a minimizing $A^*$ associated to a SCP must have all required skills met for task 1.

If each required skill for task 1 is met, any additional assignments of staff members to task 1 won’t reduce the cost anymore. The remaining staff members can better be assigned to task 2, which reduces the cost by 1 for each staff members assigned to it.

By minimizing the number of staff members needed to meet all the required skills for task $w_1$, one maximizes the number of staff members which still can be assigned to task $w_2$, each reducing the total cost by 1 with our defined $p$.

**Example 8 (SCP and associated Assignment Problem).** Let the universe be $U = \{1, 2, 3, 4, 5\}$, and

$$V_1 = \{1, 2, 3\},$$

$$V_2 = \{1, 2\},$$

$$V_3 = \{3, 4\},$$

$$V_4 = \{4, 5\},$$

$$V_5 = \{5\}.$$

$V_1, V_4$ is the only pair which can cover $U$, making 2 the smallest number of subsets covering $U$. Consider 5 staff members with available skill matrix, associated with the above SCP;

$$S = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}.$$ 

Indeed the first skill vector $s_1 = (1, 1, 1, 0, 0, 1)'$ corresponds to the subset $V_1$
with an additional 6\textsuperscript{th} skill, etc. . . With the following two overlapping tasks:

\[ w_1 = (1, 1, 1, 1, 0), \]
\[ w_2 = (0, 0, 0, 0, 5), \]

penalty vector \( p = (2, 2, 2, 2, 2, 1) \), and task-time vector \( (1, 1) \), the minimizing consistent assignment matrix \( A^* \) is

\[
A^* = \begin{pmatrix}
1 & 0 \\
0 & 1 \\
0 & 1 \\
1 & 0 \\
0 & 1
\end{pmatrix}.
\]

All required skills for the first task are meet, and 2 is the minimum number of staff members needed to assign to task 1 to accomplish this. For this \( A^* \) the cost become \( C(A^*) = 2 \). Staff members 1 and 4 which are assigned to task 1 corresponds to subsets \( V_1 \) and \( V_4 \), which was the only pair of subsets covering \( U \).

5.4 General Case

For the general case, we use a statistical method called Simulated Annealing (SA), explained in the next section.

6 Simulated Annealing

Simulated Annealing (SA) is a Monte Carlo optimization technique, based on the Metropolis Hasting (MH) algorithm. The Metropolis Hasting algorithm is a Markov Chain Monte Carlo method to sample from a probability distribution.

The SA Algorithm is the analogy of the physical phenomenal called annealing, where a substance is slowly cooled down, where eventually it will reach a state of minimum energy.

It was first used on minimization problems by Kirkpatrick, Gelett, and Vecchi\[12\] (1983), and Cerny\[4\] (1985). Here we will apply the algorithm to the assignment problem formulated in 5.1. We are not the first to apply SA on planning problems. A notable implementation of SA on planning, is the case of Willis\[16\] and Terrill (1994), who applied the algorithm to plan the Australian State Cricket Season (with success).

After briefly discussing the Metropolis-Hasting algorithm, simulated annealing will be explained. Simulated annealing and the Metropolis-Hasting algorithm are built upon the same concepts.
6.1 The Metropolis-Hasting Algorithm

Given a finite set $S$, and a probability mass function $f : S \to \mathbb{R}$, the Metropolis-Hasting algorithm describes a method to obtain (dependent) samples from $f$. This is achieved by constructing an ergodic Markov Chain $X^{(t)}$, which has $f$ as its stationary distribution.

The ingredients for the Metropolis-Hasting Algorithm (as we describe it here) are:

- A finite discrete set $S$ (states).
- A probability mass function $f : S \to \mathbb{R}$ (target function), with $f(x) > 0$ for $x \in S$.
- A proposal matrix $Q = [q_{xy}]$, for $x, y \in S$. $Q$ is assumed to be a transition matrix. For fixed $x \in S$, we call $q_{xy}$ the proposal distribution for state $x$. The Markov Chain associated with transition matrix $Q$, is assumed to be ergodic (irreducible and aperiodic). In addition, we will assume that if $q_{xy} > 0$ then $q_{yx} > 0$, for all $x, y \in S$.

The MH-algorithm constructs a Markov Chain $X^{(t)}$ as follows:

<table>
<thead>
<tr>
<th>Metropolis-Hasting Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose an initial state $X^{(0)} = x_0 \in S$. If the current state is $X^{(t)} = x$, the next state $X^{(t+1)}$ is determined with the following steps:</td>
</tr>
<tr>
<td>1. Draw $y \sim q_{xy}$.</td>
</tr>
<tr>
<td>2. Calculate the acceptance probability $\alpha_{xy} = \min \left(1, \frac{f(y) \cdot q_{yx}}{f(x) \cdot q_{xy}} \right)$.</td>
</tr>
<tr>
<td>3. Set $X^{(t+1)} = \begin{cases} y &amp; \text{with probability } \alpha_{xy} \text{ (accept)} \ x &amp; \text{with probability } 1 - \alpha_{xy} \text{ (reject)} \end{cases}$</td>
</tr>
</tbody>
</table>

The Markov Chain $X^{(t)}$, has transition probabilities

$$P_{xy} = \mathbb{P} \left( X^{(t+1)} = y | X^{(t)} = x \right) = q_{xy} \cdot \min \left(1, \frac{f(y) \cdot q_{yx}}{f(x) \cdot q_{xy}} \right).$$

Since we assume that if $q_{xy} > 0$ then $q_{yx} > 0$, and $f(x), f(y) > 0$, the above expression is well defined.

From the transition probabilities we see that if $q_{xy} > 0$, then $P_{xy} > 0$. Since $S$ is finite, the aperiodic and ergodic properties of $Q$ are inherited by
Aperiodic ergodic Markov Chains over a finite space have a unique stationary distribution.

We show that \( f \) is this unique stationary distribution by showing that detailed balance holds. For \( x, y \in S \), we have:

\[
P_{xy} f(x) = q_{xy} \cdot \min \left( 1, \frac{f(y) \cdot q_{yx}}{f(x) \cdot q_{xy}} \right) f(x)
\]

\[
= \min (f(x) \cdot q_{yx}, f(y) \cdot q_{xy})
\]

\[
= P_{yx} f(y) \text{ (by symmetry)}
\]

Summing over \( y \) gives

\[
f(x) = \sum_y P_{yx} f(y),
\]

proving that \( f \) is the stationary distribution for Markov chain \( X^{(t)} \).

Ideally, one would like to sample the initial state \( X(0) \) from \( f \). In practice this is often not possible, and one neglects the first generated samples (burn-in period), to mimic sampling the initial state from \( f \).

**Example 9 (Metropolis-Hasting Simulation).** Suppose

\[
f(k) = \begin{cases} 
C_\lambda \frac{k^{k-\lambda}}{k!}, & k = 0, 1, 2, \ldots, 15 \\
0 & \text{else}
\end{cases}
\]

for some normalization constant \( C_\lambda \). This is a truncated Poisson distribution. To illustrate the MH-algorithm we simulate 1000 realizations from \( f \) using \( \lambda = 5 \).

We define the proposal transition matrix \( Q = [q_{xy}] \) for \( 0 < x < 15 \) as:

\[
q_{xy} = \begin{cases} 
0.5 & \text{if } |x - y| = 1 \\
0 & \text{otherwise}
\end{cases}
\]

And for \( x = 0 \) and \( x = 15 \):

\[
q_{xy} = \begin{cases} 
1 & \text{if } |x - y| = 1, \text{ for } x \in S \\
0 & \text{otherwise}
\end{cases}
\]

This is a very typical ‘neighborhood’-like way to define the proposal distribution. Indeed \( Q \) is irreducible and aperiodic, and if \( q_{xy} > 0 \), then \( q_{yx} > 0 \).

The chain is initialized by setting \( X^{(0)} = 1 \), where the first 1000 samples are neglected. A bar-chart of 1000 subsequently generated states after the burn-in period is shown in fig. 3.

The bar-chart indeed shows a Poisson like distribution shape, what one would expect.
Figure 3: Computer Simulations with the Metropolis-Hasting algorithm. 1000 samples are generated with the Metropolis-Hasting algorithm for $f(k) = C_\lambda \times \frac{\lambda^k}{k!} e^{-\lambda}$, for $k = 0, 1, \ldots, 15$, and $\lambda = 5$. 
Figure 4: A plot of $g(x) = x^2$, and 3 plots of $f$ for different $T$ (10, 1, 1/10).

### 6.2 Metropolis-Hasting with $\exp(-g)$

For the Metropolis-Hasting algorithm to work, it suffices to know the target function $f$ up to a certain constant. Hence any function $f$ proportional to some (unknown) distribution function, with $f(x) > 0$, for $x \in S$, can be used as a target function in the Metropolis-Hasting algorithm.

In particular, for an arbitrary real valued function $g : S \to \mathbb{R}$, defined on a finite discrete set $S$, we can apply the Metropolis-Hasting algorithm on a function $f$, defined by

$$f(x) = \exp\left(-\frac{g(x)}{T}\right), x \in S \ T > 0.$$ 

$f$ is positive, over a finite set $S$, and therefore proportional to some density. If $x^* \in S$ is a minimum for $g$, then $x^*$ is a maximum for $f$, as $\exp(-x)$ is decreasing. As $T$ gets smaller the function $f$ gets more spiked, with a maximum point at $x^*$. Figure 4 contains a plot of $g(x) = x^2$, and three plots of $f$ for $T = 10, 1, 1/10$.

We could construct a discrete time Markov Chain, that has $f$ as its stationary distribution. This is the main idea of Simulated Annealing, where $T$ is slowly decreased towards zero, hopefully yielding samples from a density with a spike at the minimum of $h$. 

21
6.3 Simulated Annealing Algorithm

Annealing is the physical process where a material is cooled down in such a way, that its physical properties are altered. Eventually, the material will reach a state of minimum energy. Simulated Annealing mimics this process, where one tries to reach a state with minimum energy, i.e. a minimizing state.

With the help of a non-increasing function $T(t) : \mathbb{N} \rightarrow (0, \infty)$, with

$$\lim_{t \to \infty} T(t) = 0,$$

called the cooling schedule, a time-depended Markov Chain $X(t)$ is formed. With starting temperature $T_0$ and cooling factor $\alpha$, we used the following cooling schedule for our SA implementation:

$$T(t) = T_0 \alpha^{\lfloor \frac{t}{N_p} \rfloor}.$$

The above cooling schedule was adopted from [15], but many others suitable cooling schedules can be defined [13], and studies have been done comparing several cooling schedules [14] [2].
Simulated-Annealing Algorithm Implementation

Given:
- $Q$ a transition matrix on $S$
- $g : S \to \mathbb{R}$ (objective function)
- $x_0 \in S$ (initial state)
- $T_0 \in \mathbb{R}$, with $0 < T_0$ (starting temperature)
- $T_{\text{stop}} \in \mathbb{R}$, with $0 < T_{\text{stop}} < T_0$ (stopping temperature)
- $\alpha \in \mathbb{R}$, with $0 < \alpha < 1$ (cooling factor)
- $N_p \in \mathbb{N}$ (number of repetitions)

$T \leftarrow T_0$ (initialize the starting temperature)
$X^{(0)} \leftarrow x_0$ (set the initial state)
$t \leftarrow 0$

while $T \geq T_{\text{stop}}$ (stopping criterion) do

repeat $N_p$ times

- $x \leftarrow X^{(t)}$,
- Draw $y \sim q_{xy}$, (proposal state)
- Calculate the acceptance probability
  \[ \alpha_{xy} \leftarrow \min \left( 1, \exp \left[ \frac{g(x) - g(y)}{T} \right] \right) \]
- Set
  \[ X^{(t+1)} = \begin{cases} 
  y & \text{with probability } \alpha_{xy}. \text{ (accept)} \\
  x & \text{with probability } 1 - \alpha_{xy} \text{ (reject)}
  \end{cases} \]
- $t \leftarrow t + 1$

end of $N_p$ repetitions

$T \leftarrow \alpha T$ (lower the temperature)

end

Return: $X^{(\mu)}$, where $\mu = \max \{ 0 \leq k \leq t : g(X^{(k)}) = \min_{0 \leq \tau \leq t} g(X^{(\tau)}) \}$
(the last found state which obtained the minimum.)

For fixed $T$, like shown in previous section, the stationary distribution will be concentrated around the global minimum of function $g(x)$, and acts as a Metropolis Hastings Markov Chain. If the cooling schedule is cooling ‘slowly enough’, it sounds plausible that the Markov Chain will ultimately end up in a global minimum of $g$. This will be made more precise in section 6.4.1.

For higher temperatures ($T \gg 0$), proposal states with a greater objective function than the current state $x$ still have an acceptance probability big enough to be accepted. This occasionally ascends in the algorithm prevent getting stuck inside a local minimum. As the temperature reaches 0, ascends
are less likely to happen (as their probability reaches 0). The strength of SA is exactly that it allows these occasional ascends in the search for the minimum, unlike greedy algorithms which could fall into local minima. SA is closely related to the MH-algorithm. With the time dependent function \( f_t : S \rightarrow \mathbb{R} \) as
\[
 f_t(x) = \exp \left[ \frac{-g(x)}{T(t)} \right],
\]
the transition probabilities of SA become identical to the transition probabilities of the MH-algorithm, if one applies the MH-algorithm on \( f_t \) with acceptance probability:
\[
 \alpha_{xy} = \min \left( 1, \frac{f_t(y)}{f_t(x)} \right) = \min \left( 1, \exp \left[ \frac{g(x) - g(y)}{T(t)} \right] \right).
\]
Note that the above acceptance probability is slightly different than that of the MH-algorithm. Absence of the proposal probabilities \( q_{xy}, q_{yx} \) in the acceptance probability for SA, will cause that for fixed \( T \), the stationary distribution is not necessary identical to \( f_T(x) = \exp(-g^2(x)) \). This is not of our concern, since we are ultimately interested in reaching a state of minimum energy, and not in the particular stationary distribution for fixed \( T \). In practice, the proposal state is often randomly chosen, without knowledge of the values \( q_{xy}, q_{yx} \). Therefore they are often omitted in the acceptance equation, though the algorithm would still work if they would be included.

### 6.4 Simulated Annealing to generate a Markov Chain

For our SA implementation we fixed the cooling schedule \( T(t) \), choose a stopping criterion, and defined an output state. For general cooling schedules and without the stopping criterion, SA generates a Markov Chain \( X^{(t)} \) for \( t = 0, 1, 2, \ldots \), for some cooling schedule \( T(t) \).
SA-Algorithm for general $T(t)$ to generate $X^{(t)}$, for $t = 0, 1, 2, \ldots$

Given:
- $Q$ a transition matrix on $S$
- $g : S \to \mathbb{R}$ (objective function)
- $x_0 \in S$ (initial state)
- $T(t) : \mathbb{N} \to \mathbb{R}$ (cooling schedule)

Set $X^{(0)} = x_0$, and apply the MH-algorithm on time-depended $f_t$ (proportional to a density):

$$f_t(x) = \exp \left[-\frac{g(x)}{T(t)}\right],$$

with the following acceptance probability

$$\alpha_{xy} = \min \left(1, \frac{f(y)}{f(x)} \right) = \min \left(1, \exp\left[\frac{g(x) - g(y)}{T(t)}\right] \right).$$

### 6.4.1 Convergence Speed and Conditions

Within this section, if we refer to the SA algorithm, we refer to the “SA-Algorithm for general $T(t)$ to generate $X^{(t)}$, for $t = 0, 1, 2, \ldots$” of section 6.4.

For the resulting Markov Chain on objective function $g : S \to \mathbb{R}$, define $S^* \subset S$ as the set of global minima. For $x \in S^*$,

$$g(x) \leq g(y), \text{ for all } y \in S.$$ 

We say the SA converges if

$$\lim_{t \to \infty} \mathbb{P} \left(X^{(t)} \in S^*\right) = 1.$$ 

The following example taken from [6] shows that if $Q$ is irreducible, SA not necessarily converges.

**Example 10.** Let $S = \{1, 2, 3\}$, and define $g$ as $g(x) = x(x = 1, 2, 3)$. Let the proposal matrix be

$$Q = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$ 

$Q$ is irreducible. For fixed $T$, each transition probability, can be calculated with help of the acceptance probabilities. For states $x, y \in \{1, 2, 3\}$, where $x \neq y$, we have the following transition probability

$$\mathbb{P}_{xy} = q_{xy} \cdot \min \left(1, \exp\left[\frac{g(x) - g(y)}{T}\right] \right).$$
resulting in the following transition matrix $M$:

$$
M = \begin{pmatrix}
1 - e^{-1/T} & e^{-1/T} & 0 \\
0 & 1 - e^{-1/T} & e^{-1/T} \\
1 & 0 & 0
\end{pmatrix}.
$$

Figure 6.4.1 shows the 3 states, with arcs where $q_{xy} > 0$, and returning arcs to each state representing not accepting the next proposal state.

$M$ is irreducible, and aperiodic. Solving $\pi(T) = \pi(T)M$ to get the stationary probabilities gives

$$
\pi_2(T) = \frac{1}{2 + e^{-1/T}}.
$$

Taking the limit $\lim_{T \to \infty} \pi_2(T) = \frac{1}{2} \neq 0$, shows we have a positive stationary probability for state 2 which is not a global minimum. Hence for this irreducible $Q$, SA does not converge.

The examples shows us that irreducibility on $Q$ (and therefore $M$) is not sufficient for convergence. Hajek[8] presented necessary and sufficient conditions on $Q$ and $T(t)$ for convergence, we will now present.

**Definition.** We say state $y$ is reachable from state $x$ at **height** $h$ if there is a sequence of states $x = x_0, x_1, \ldots, x_m = y$ such that

$$
q_{x_k,x_{k+1}} > 0, \text{ for } 0 \leq k < m,
$$

and

$$
g(x_k) \leq h, \text{ for } 0 \leq k \leq m.
$$
Figure 6.4.1 contains a figure of states, with their heights shown on the vertical axis. In this figure, state 4 is reachable from state 1 at height 12. State 4 is not reachable from state 5 at height 8, but is at height 10.

For the \((S, Q, g)\) triplet, we define the following two properties:

**Property 1 (Strong Irreducibility (SI)), on \((S, Q, g)\).** For any \(x, y \in S\), \(x\) is reachable from \(y\) for some height \(h\).

This is a way of stating that all states communicate under \(Q\). If the proposal probabilities \(Q\), are such that its corresponding Markov Chain is irreducible, \((S, Q, g)\) always meets the (SI) property.

**Property 2 (Weak Reversibility (WR)), on \((S, Q, g)\).** For any \(x, y \in S\), and any \(h > 0\), \(x\) is reachable at height \(h\) from \(y\), if and only if \(y\) is reachable at height \(h\) from \(x\).

**Note 1.** If \(Q\) is chosen such that if \(q_{xy} > 0\), then \(q_{yx} > 0\), for all \(x, y \in S\), this implies the (WR) condition on \(Q\), and therefore is a stronger proposition. Indeed, for such \(Q\), if \(x\) is reachable at height \(h\) from \(y\), there is a sequence of states from \(y\) to \(x\), with height equal or lower then \(h\). Simply reversing the sequence shows \(y\) is reachable at height \(h\) from \(x\), and therefore (WR) holds.

**Definition.** We call \(s \in S\) a Q-local minimum if \(g(s) \leq g(x)\), for each \(x \in S\) which is reachable from \(s\) at height \(g(s)\) for some \(Q\).

For \(s \in S\) to be a Q-local minimum, depends on the choice of \(Q\).

**Definition.** The depth of a Q-local minimum \(s\), which is not a global minimum, is defined as the smallest number \(d > 0\), such that some state \(x\) with \(g(x) < g(s)\) can be reached from state \(s\) at height \(g(s) + d\). For a global minimum \(s\), we define the depth as \(d = \infty\).

When a SA realization accepts a Q-local minimum at temperature \(T\), the depth of the Q-local minimum is related to the chance of escaping this Q-local minimum. Q-local minima with smaller depths, have a higher probability to be escape from.

**Definition.** Let \(C \subset S\). We call the subset \(C\) a cup, if for some \(h > 0\), we have for each \(c \in C\):

\[
C = \{ s : s \text{ can be reached at height } h \text{ from } c\}.
\]

Moreover, for a cup \(C\), let

\[
\bar{g}(C) = \max_{c \in C} g(c),
\]

and

\[
g(C) = \min_{c \in C} g(c).
\]
Definition. For cup $C$, the **bottom** $B \subset C$ is defined as

$$B = \{ b \in B : g(b) = g(C) \}.$$  

Definition. For cup $C$, define its **depth** $d$ as

$$d = \bar{g}(C) - \underline{g}(C).$$

Each Q-local minimum with depth $d$, which is not a global minimum, is an element of bottom $B$ for some cup of depth $d$. For a Q-local minimum $s$ of depth $d$, the following set $C$ will be a cup of depth $d$:

$$C : \{ x \in S : x \text{ is reachable at height smaller than } g(s) + d \text{ from } s \}.$$  

With these definitions, we can state Hajek’s theorem.

**Theorem 1** (Hajek’s Theorem for escaping Q-local minima). Let $X(t)$ be a Markov Chain, produced by the SA-algorithm with proposition matrix $Q$, objective function $g$, and cooling schedule $T(t)$. Assume that (SI) and (WR) holds, and that the cooling schedule $T(t)$ is a non-increasing function with

$$\lim_{t \to \infty} T(t) = 0,$$

Let $B$ be the bottom of some cup with depth $d$, and a set of Q-local minima of depth $d$. Then

$$\lim_{t \to \infty} \mathbb{P}[X(t) \in B] = 0,$$
if and only if
\[ \sum_{t=0}^{\infty} \exp \left( -\frac{d}{T(t)} \right) < \infty. \]

Hajek showed these are necessary and sufficient conditions for escaping a Q-local minima.
For example, for the following cooling schedule the above theorem can be applied if and only if \( c \geq d \):
\[ T(t) = \frac{c}{\log(t)}, \quad t > 0. \]

With the above cooling schedule, for cups \( B \) with depth \( d \) we have \( \lim_{t \to \infty} \mathbb{P}[X(t) \in B] = 0 \).
Let \( d^* \) be the maximum of all the finite depth’s of Q-local minima. For this \( d^* \), there exists a cooling schedule such that for the set of Q-local minima with finite depth, \( S_{\text{Q-local minima, finite depth}} \), we have
\[ \lim_{t \to \infty} \mathbb{P} \left[ X^{(t)} \in S_{\text{Q-local minima, finite depth}} \right] = 0. \]
Without proof, Hajek also conjectured that for a state \( s \in S \) which is not a Q-local minimum we have
\[ \lim_{t \to \infty} \mathbb{P} \left[ X^{(t)} = s \right] = 0. \]
Combining these results give rise to a logarithmic cooling schedule for which SA will converge.
The logarithmic cooling scheme is at this moment only of theoretical use. Implementation would require endless generation of states, with a slow logarithmic cooling schedule. On top, the constant \( d^* \) is not known a priori, and can be difficult to calculate. But there is good news, heuristics have shown SA produces even good results with faster cooling schemes.

6.5 SA and Constrained Minimization

When applying SA on constrained minimization, the minimizing point \( x \) is required to be an element of some subset \( \tilde{S} \subset S \) which satisfies all conditions. \( \tilde{S} \) is called the feasible region.
SA can handle constraints in two ways. (1) Never propose/accept states which fall outside the feasible region \( \tilde{S} \). (2) Add a penalty to the object function, when a state outside \( \tilde{S} \) is proposed, making it possible to be accepted.
The first method can require extra calculations to ensure proposals stay within \( \tilde{S} \). The second method can use any proposal method. Here one hopes the penalty is big enough to (often) converge within the feasible region,
but small enough to be able to 'search through' the entire set $S$. Walking
though forbidden areas sometimes leads to shortcuts. The preferred method
is problem dependent.
For our assignment problem, using the overlap matrix $P$ makes it easy to
check whether we stay the feasible region, so we choose to never propose
states which fall outside the feasible region.

7 SA applied on the Assignment Problem

SA has several properties which makes it a suitable algorithm for planning
problems:

- SA is an iterative algorithm, making it possible to use valid plannings
  as a input which can be (partially) improved.
- SA has no requirements on the cost function (linear / non-linear).
- Overlap and other constraints can be implemented by avoiding the
corresponding states.
- For most practical planning problems, 'close to' the minimum is good
  enough. In other words, a very good Q-local minimum which is not a
global minimum, translates to a very good planning, but not the best.

For SA applied on the assignment problem, each state $x$ corresponds to an
assignment matrix $A_x$. State and planning are used interchangeably from
now on. For an implementation of SA on the assignment problem we
need to choose:

1. A suitable matrix $Q$, which preferably holds the (SI) and (WR) prop-
   erty, and stays within the feasible region;
2. An initial state $x_0$;
3. A cooling factor $\alpha$, and start/stoping temperature $T_0/T_{stop}$, and the
   number of repetitions $N_p$.

As pointed out by Faigle and Kern [6], the proposal probabilities are of much
greater impact than the cooling schedule for good results. This is confirmed
by our experiments.

7.1 Neighborhood Structure

The proposal matrix $Q$ defines the probabilities to draw a proposal state $y$,
for the current state $x$. Recall SA doesn’t require knowledge of the value
$q_{xy}$, once the proposal stated is chosen. Also, Hajek’s conditions for con-
vergence don’t rely on the size of the proposal probabilities, only on which
proposal states have positive probabilities. Any 'Neighborhood Structure' which stochastically defines a new state depending on the current will therefore suffice, if (SI) and (WR) can be checked. For our purpose, we also require for the neighborhood structure to stay within the feasible region.

7.1.1 Spray Neighborhood Outline

Our Neighborhood structure assumes the current planning $A_x$ lies in the feasible region. $A_x$ is altered on one row $i$, where $i$ is picked randomly, leading to a new state $A_y$. We illustrate the main idea by means of a small example. Suppose planning $A_x$ for staff member $i$ is as visually presented in fig. 7. The figure shows all $J = 6$ tasks proportional to their length. Staff member $i$ is currently assigned to 3 of them, indicated by thick lines.

![Figure 7: Planning $A_x$. Staff member $i$ is currently assigned to 3 tasks, indicated by thick lines. There are a total of 6 tasks.](image)

First, we randomly select a set of $k$ non-overlapping tasks $\mathcal{J}$, which we call a suggestion set. The selection process will be explained later in detail. For an example of a suggestion set $\mathcal{J}$, see fig. 8, where $k = 1$.

![Figure 8: Planning $A_x$ and set $\mathcal{J}$. Tasks in set $\mathcal{J}$ are indicated by dashed lines, and do not overlap with each other, but could overlap with tasks in $A_x$ (continuation of fig. 7).](image)

Now we re-assign staff member $i$ to the tasks in $\mathcal{J}$ by setting the corresponding elements in $A_x$ to 1. This may cause overlap, as is our case in fig. 8, where 2 tasks overlap. We handle this by de-assigning staff member $i$ from all the tasks in his original planning $A_x$, if they cause overlap with $\mathcal{J}$ (see the example in fig. 9).

![Figure 9: Planning $A_y$. Staff member $i$ is re-assigned, such that he is assigned to all the tasks in $\mathcal{J}$, and his original tasks from $A_x$, if they do not overlap with any task in $\mathcal{J}$ (continuation of fig. 8).](image)
The de-assignment of tasks guarantees that we stay in the feasible region. We call this neighborhood structure the Spray neighborhood structure, since it sprays a new suggestion of tasks over the old ones.

### 7.1.2 The Spray Neighborhood Structure

**Spray Neighborhood Structure — Construct proposal planning** \( A_y, \text{ given } A_x \).

Assume current state/planning \( A_x \) has no overlapping tasks. Fix \( k \in \{1, 2, \ldots \} \). Proposal \( A_y \) is constructed by:

1. Uniformly select a staff member \( i \in I \).

2. Construct **suggestion set** \( J \subset \{1, 2, \ldots, J\} \) for staff member \( i \). This is a set of at most \( k \) non-overlapping tasks, which is chosen in a specific randomized way, depending on the current planning \( A_x \). The dependence of \( J \) on \( k \), \( i \), and \( A_x \) is suppressed in the notation.

3. Define \( N_J \subset \{1, 2, \ldots, J\} \), the set of tasks which have overlap with tasks in \( J \) by

   \[
   N_J = \{ j : P[j, l] = 1, \text{ for some } l \in J \}.
   \]

   \( N_J \) depends on \( A_x \), \( i \), and constant \( k \) as well.

4. Set \( A_y = A_x \), and next:

   \[
   A_y[i, j] = \begin{cases} 
   1 & \text{if } j \in J \\
   0 & \text{if } j \in N_J \setminus J 
   \end{cases}
   \]

   In this way, staff member \( i \) is assigned to the tasks in \( J \), and is de-assigned from original tasks that cause overlap.

With the help of the overlap matrix \( P \) — the overlapping tasks subject to de-assignment can be quickly be calculated (see example 3). Next we explain how the suggestion set \( J \) is constructed.

### 7.1.3 Suggestion set \( J \)

The suggestions set \( J \), containing \( k \) tasks is constructed by the following scheme with help of the weight vector \( \eta \).
Construction of suggestion set $\mathcal{J}$ of size $k$, given $A_x$ and $i$.

For $k \in \mathbb{N}$, weight vector $\eta \in \mathbb{R}^J$ we construct a set of $k$ non-overlapping tasks $\mathcal{J}$, by

1. Set $\mathcal{J} = \emptyset$.
2. Draw $j \sim \eta_j$, and set
   \[ \mathcal{J} = \mathcal{J} \cup \{j\} \]
3. Update weight vector $\eta$, by setting
   \[ \eta_j = 0 \text{ for all } j \in \mathcal{N}_\mathcal{J} \]
4. Return $\mathcal{J}$ if one of the following holds:
   (a) $\|\eta\| = 0$. (This means we cannot assign more tasks that are non-overlapping to $\mathcal{J}$.)
   (b) $|\mathcal{J}| = k$. ($k$ overlapping tasks have been selected.)

Else, repeat from step 3.

In line with the recommendations of Eglese [5], $k$ should not be chosen too big. Eglese stated: “In general, a neighborhood structure which imposes a 'smooth' topology, where the local optima are shallow is preferred to a 'bumpy' topology where there are many deep local minima.”

Next we discuss possible choices for weights $\eta$.

7.1.4 Weights for the suggestions set $\mathcal{J}$

The most simple choice for weights $\eta$ would be to set for each $i$

\[ \eta = (1, 1, \ldots, 1), \]

implying each task $j$ has an equal probability to be picked from. One downside is that a staff member can be assigned to tasks, for which he doesn’t possess any of required skills. Since these tasks could overlap with tasks where he is useful, these simple weights are not efficient.

Other poor choices are weight vectors which leads, with a high probability, to a proposal state $A_y$ which is the same as the current state $A_x$. This happens when the suggestion set is a subset of already assigned tasks. This is not necessarily wrong, but results in a waste of computation time. Especially with small $k$ this can be the case.

To ensure each proposal state is different than the current state, one could set the weights corresponding to already assigned tasks to 0.

33
Each weight vector we present comes in two flavors; one plain and one where the tasks for which staff member $i$ is already assigned are set to 0. The latter will prevent proposals identical to the current planning. To discriminate between the two variants, we mark the weights with currently assigned tasks set to zero by a bar.

For a fixed staff member $i$, the following vector $v_{ij} \in \mathbb{R}^N$ was shown to be helpful for the construction of the weight vectors:

$$v_{ij} = \text{diag}(s_i)w_j.$$  

If element $k$ of $v_{ij}$ is positive, staff member $i$ has the $k^{th}$ skill, and task $j$ requires skill $k$ at least ones. Also, for planning $A_x$, and staff member $i$, and task $j$, define the following two $I \times J$ matrices:

$$A_{x=0}^j[q, r] = \begin{cases} 0 & \text{if } (q, r) = (i, j) \\ A_x & \text{Otherwise.} \end{cases}$$

$$A_{x=1}^j[q, r] = \begin{cases} 1 & \text{if } (q, r) = (i, j) \\ A_x & \text{Otherwise.} \end{cases}$$

We fix staff member $i$, and assume the current state $A_x$ lies in the feasible region, and define the following weight vectors of length $J$.

**CAN-DO weights** Set $\overline{\text{CAN-DO}}_j = 1$ if the staff member has any of the required skill for task $j$, and zero otherwise.

$$\overline{\text{CAN-DO}}_j = \begin{cases} 1 & \text{If } \|v_{ij}\| > 0 \\ 0 & \text{Otherwise.} \end{cases}$$

**IS-USEFUL weights** Sometimes, assigning an extra staff member to task $j$ is of no use, since the current planning $A_x$ already has enough skills assigned to task $j$. In other words, assigning him to task $j$ doesn’t have effect in the cost function.

$$\overline{\text{IS-USEFUL}}_j = \begin{cases} 1 & \text{If } C(A_x^j=1) < C(A_x^j=0) \\ 0 & \text{Otherwise.} \end{cases}$$
**CAN-FINISH weights** If assigning the staff member to task $j$, would have the effect that only then all the required skills are met, with respect to $A_x$, we say the staff member can ‘finish’ task $j$.

If staff member $i$ is assigned to task $j$, all required skill for task $j$ are met if

$$\text{Col}_j(W - SA_x^{j=1}) \preceq 0_{1 \times J}. \quad (M_j)$$

**CAN-FINISH weights**

$$\eta_j = \begin{cases} 
1 & \text{if } \text{CAN-FINISH}_j = 1 \text{ and } A_x[i, j] = 0 \\
0 & \text{otherwise}.
\end{cases}$$

**SCARCITY weights** We define the following quantity to indicate the scarcity for each skill $k$ (given $I$ staff members):

$$\mathcal{S}_k = 1 - \frac{\# \text{ staff members possessing skill } k}{I + 1},$$

so that $\frac{1}{I+1} \leq \mathcal{S}_k \leq 1$. When closer to 1, it indicates a high scarcity for the skill. Let $B_{ij} = \{k \in \{1, 2, \ldots, N\} : v_{ij}[k] > 0\}$. Then

$$\text{SCARCITY}_j = \begin{cases} 
\max_{n \in B_{ij}} \mathcal{S}_n & \text{if } B_{ij} \neq \emptyset \\
0 & \text{otherwise}.
\end{cases}$$

**CAN-DO** and **SCARCITY** weights do not depend on the current planning $A_x$ and therefore remain constant during a SA run. All other weights need to be re-calculated for each proposal. These computation costs are about as expensive as the object function itself.

For fixed $A_x, i$, we have

$$\text{CAN-DO} \succeq \text{IS-USEFUL} \succeq \text{CAN-FINISH},$$

and

$$\text{CAN-DO} \succeq \text{SCARCITY}.$$
SCARCITY weights reflect the principle, that staff members with rare skills, can be best assigned to tasks which require this rare skill. Planners know they should not waste its specialists on common tasks, the scarcity weights try to mimic this principle.

Example 11. For a planning setup with $N = 2$ skills, and $I = 3$ staff members, and $J = 4$ tasks, let available skill matrix $S$, and required skill matrix $W$ be:

$$S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad W = \begin{pmatrix} 2 & 3 & 3 & 2 \\ 1 & 1 & 0 & 0 \end{pmatrix}. $$

We don’t define the start and end times, but assume none of the 4 task overlap. Let assignment matrix $A_x$ and its resulting assigned skill be:

$$A_x = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad SA_x = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 0 \end{pmatrix}. $$

The CAN-DO weights for staff member $i = 1$ become:

$$\text{CAN-DO} = (1, 1, 1, 1),$$

and

$$\text{CAN-DO} = (0, 1, 0, 1).$$

Indeed, to calculated the first element of CAN-DO, we need to look at vector

$$v_{1,j=1} = (2, 1)^t,$$

which has a positive norm. Since staff member 1 is already assigned to task 1, the CAN-DO weight has its first element 0.

To calculate the IS-USEFUL weight, we first de-assign $i = 1$ from any task, to get the following assignment matrix and assigned skill:

$$A_x^* = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad SA_x^* = \begin{pmatrix} 1 & 2 & 1 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}. $$

By comparing the above assigned skill without staff member $i$, and the required skill matrix, we can easily see if $s_1 = (1, 1)^t$ is useful to assign to each task. Check that he is useful for each task, except the last task, where already sufficient skills are assigned:

$$\text{IS-USEFUL} = (1, 1, 1, 0).$$

The same matrices can be used to deduct if assigning staff member 1 only then has the effect that all the required skills are met. This is only the case for the first and second tasks:

$$\text{CAN-FINISH} = (1, 1, 0, 0),$$

36
To calculate the scarcity weights, first we calculate:

\[ \mathcal{S}_1 = 1 - \frac{3}{4} = \frac{1}{4} \]
\[ \mathcal{S}_2 = 1 - \frac{1}{4} = \frac{3}{4} \]

For the first 2 tasks skill \( n = 2 \) is required, and possessed by \( i = 1 \). The latter two tasks only require the first skill, which is also possessed by \( i = 1 \).

\[ \text{SCARCITY} = (\frac{3}{4}, \frac{3}{4}, \frac{1}{4}, \frac{1}{4}) \]

All linear combinations with positive scalars are valid weights too, like

\[ A_{ij} = \text{CAN-DO}_{ij} + 5 \text{IS-USEFUL}_{ij}, \]
and the \text{SCARCITY} weights can be used as multipliers for other weights, e.g:

\[ B_{ij} = \text{diag}(\text{SCARCITY}_{ij}) \cdot \text{IS-USEFUL}_{ij}. \]

The latter will prioritize tasks for which the staff member possesses a rare skill, if the corresponding staff member is still useful.

### 7.2 Adjustments for the (SI) and (WR) condition

With \text{CAN-DO} weights, problems exists for which the (SI) condition is not satisfied with the Spray neighborhood structure. Indeed, a staff member cannot be de-assigned from all tasks, since \( J \neq \emptyset \) for every proposal. Therefore, the zero-matrix is not reachable from any state. Besides the zero-matrix, there are many more sparse assignment matrices which are not necessarily reachable from all states.

In addition, any assignment matrix which has staff members assigned to tasks for which he doesn’t possess any of the required skill are also not reachable with the \text{CAN-DO}, \text{IS-USEFUL}, \text{CAN-FINISH}, or \text{SCARCITY} weights. To solve the latter, we restrict the problem to the subset \( \tilde{S} \subset S \) where only staff members are assigned to tasks for which he possess at least 1 of the required skills.

\[ \tilde{S} = \{ A_x \in S : A_x[i,j] = 0 \text{ if } ||v_{ij}|| = 0 \} . \]

To make the neighborhood structure satisfy the (SI) and (WR) condition within \( \tilde{S} \), we introduce a variant on the Spray neighborhood, the \text{Clean-and-Spray}. In this variant, all previous assigned tasks for the selected worker \( i \) are de-assigned, where after he will be assigned to all tasks in the suggestions set \( \tilde{J} \), where \( \tilde{J} \) could be the empty set.
7.2.1 Clean-and-Spray Neighborhood structure

Clean-and-Spray Neighborhood Structure — Construct proposal planning $A_y$, given $A_x$.

Assume current state/planning $A_x \in \tilde{S}$ has no overlapping tasks. Proposal $A_y$ is constructed by:

1. Uniformly select a staff member $i \in I$.
2. Pick $k \in \{0, 1, 2, \ldots, J\}$, uniformly.
3. Construct a suggestion set $J \subset \{1, 2, \ldots, J\}$. $J$, using the CAN-DO weights.
4. Set $A_y = A_x$, except for row $i$ where:

$$A_y[i, j] = \begin{cases} 1 & \text{if } j \in J \\ 0 & \text{else.} \end{cases}$$

With this neighborhood structure, for each state $x \in \tilde{S}$, all states $y$ become reachable within $\tilde{S}$ (within the feasible region), hence the (SI) condition is now satisfied.

Let $q_{xy}$ be the proposal probability for state $y$ from $x$. For the Clean-and-Spray we have that if $q_{xy} > 0$, then $q_{yx} > 0$ for $x, y \in \tilde{S}$. This symmetric property of the proposal scheme implies the (WR) condition.

We have shown that the (SI) and (WR) conditions hold, but it is not necessarily an improvement for the speed of convergence. But the comfort of having a method which converges in theory, makes us want to use it anyhow. To still benefit from our smarter Spray neighborhood structure, we apply the Clean-and-Spray in combination with the Spray neighborhood structure.

We can create a mix by simply using one of the two methods, after having selected staff member $i$, we proceed as follows: With probability 95%, we apply the Spray neighborhood structure with a weight vector of our choice, and with probability 5% we apply the Clean-and-Spray neighborhood structure with the CAN-DO weight vector.

The resulting mix still has that if $q_{xy} > 0$, then $q_{yx} > 0$, for all $x, y \in \tilde{S}$ and therefore still satisfies the (SI) and (WR) condition.
7.2.2 Mixed-Spray Neighborhood structure

Mixed-Spray Neighborhood Structure — Construct proposal planning \( A_y \), given \( A_x \), and weight vector \( \eta \).

1. Uniformly select a staff member \( i \in I \).
2. Draw \( m \sim \text{uniformly in } [0, 1] \).
3. If \( m \leq 0.95 \), apply the \text{Spray} neighborhood structure to construct \( A_y \), with weight vector \( \eta \).
4. Else, apply the \text{Clean-and-Spray} neighborhood structure to construct \( A_y \).

7.3 Parameters which define a SA run for the assignment problem

The following parameters/functions completely defines a SA run, applied for the assignment problem. Due the stochastic nature of the algorithm, the reader cannot reproduce outputs, but must get similar results when using the same parameters.

Simulated Annealing Parameters for the Assignment Problem (Mixed Spray Neighborhood)

Given a Planning Setup, the following parameters set the SA procedure, using the \text{Mixed-Spray} neighborhood structure:

- \( T_0 \) - Start temperature for the \text{cooling schedule}.
- \( T_{\text{stop}} \) - Stopping temperature for the \text{cooling schedule}.
- \( N_p \) - The number of repetitions before the temperature is lowered by factor \( \alpha \).
- \( \alpha \) - The \text{Cooling factor}.
- \( A_0 \) - Initial state.
- \( k \) - number of tasks in the \text{suggestion set} \( J(A_x, i, \eta, k) \) (see 11.3.3), used in the \text{Spray} neighborhood structure.
- \( \eta \) - weight vector for the \text{suggestion set} \( J(A_x, i, \eta, k) \) (see 7.1.4), used in the \text{Spray} neighborhood structure.

8 Test Case 1 — Simple Example

We will now apply SA with the Mixed-Spray neighborhood structure on a problem, where the reader can easily verify the optimal solution. We will run experiments with different parameters to see the effect of each.
For the planning setup, we use \( N = 4 \) skills, \( I = 8 \) staff members, and \( J = 14 \) tasks. Each task takes place between time unit 0 and 60 (minutes). Tasks are chosen such that they can be grouped into 4, where each group contains non-overlapping tasks of equal length.

### 8.1 Problem Formulation

#### 8.1.1 Work matrix and Available skill matrix

The first 2 tasks each have a length of 30 time-steps;

\[
\bar{w}_1 = (0, 30, (1, 1, 1, 1)')
\]
\[
\bar{w}_2 = (30, 60, (1, 1, 1, 1)')
\]

The next 3 tasks each have a length of 20 time-steps;

\[
\bar{w}_3 = (0, 20, (1, 1, 1, 1)')
\]
\[
\bar{w}_4 = (20, 40, (1, 1, 1, 1)')
\]
\[
\bar{w}_5 = (0, 60, (1, 1, 1, 1)')
\]

The next 4 tasks each have a length of 15 time-steps;

\[
\bar{w}_6 = (0, 15, (1, 1, 1, 1)')
\]
\[
\bar{w}_7 = (15, 30, (1, 1, 1, 1)')
\]
\[
\bar{w}_8 = (30, 45, (1, 1, 1, 1)')
\]
\[
\bar{w}_9 = (45, 60, (1, 1, 1, 1)')
\]

The last 5 tasks each have a length of 12 time-steps;

\[
\bar{w}_{10} = (0, 12, (1, 1, 1, 1)')
\]
\[
\bar{w}_{11} = (12, 24, (1, 1, 1, 1)')
\]
\[
\bar{w}_{12} = (24, 36, (1, 1, 1, 1)')
\]
\[
\bar{w}_{13} = (36, 48, (1, 1, 1, 1)')
\]
\[
\bar{w}_{14} = (48, 60, (1, 1, 1, 1)')
\]

\( W = [w_1, \ldots, w_{14}] \). Fig. 10 contains a graphical representation of the tasks. The overlap matrix \( P \), and task-time vector \( t \) are given in the appendix.

Figure 10: Graphical representation of the tasks for example 1.
We have chosen the 8 staff members such that — the sum of the skill vector of each two subsequent staff members is equal to $(1,1,1,1)'$.

$$S = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}.$$

### 8.1.2 Cost function

With penalty vector $p = (1,1,1,1)'$, and task-time vector as in the appendix, the cost function becomes

$$C(A) = p' \max(W - SA, 0_{J \times N})t.$$

### 8.1.3 A global minimum

By the structure of the constructed planning problem, humans can easily form groups of tasks, and create efficient pairs of staff members. I.e. put the first 2 staff members on the first 2 tasks, the second 2 staff members on the next 3 tasks, etc.

The following assignment matrix $A_{\text{min}}$ has cost 0, but is not unique:

$$A_{\text{min}} = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{pmatrix}.$$

### 8.2 SA parameters which found global minima

Table 2 contains the parameters which are used to try finding global minima with SA. In 100 runs, 100 times a global minimum was found, with cost equal to zero. Fig. 11 contains a graph of the cost (object function) in one of these runs. In the graph one can see that in the early stage where the temperature is high, ascends are accepted more than later-on.
Each of the 100 runs was finished within 50 seconds, on an Intel I3 CPU, 4GB ram, with an implementation in the software package R. The total number of possibilities is $2^{14 \times 8} = 2^{112} \approx 5 \times 10^{33}$, including states outside the feasible region.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>chosen value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_0$</td>
<td>$0_{I\times J}$ (zero matrix)</td>
</tr>
<tr>
<td>$T_0$</td>
<td>500</td>
</tr>
<tr>
<td>$T_{stop}$</td>
<td>0.5</td>
</tr>
<tr>
<td>$N_p$</td>
<td>1000</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>.95</td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
</tr>
</tbody>
</table>

weights for 3 CAN-DO

Table 2: SA parameters used to find the global minima, where the Mixed-Spray neighborhood structure was used.

Next, we vary different parameters of table 2 and re-do the experiment, to
show their impact on the speed of convergence, by counting the amount of times a global minimum was returned.

8.3 Variations in the Cooling Schedule

Table 3 contains the amount of times a global minimum was returned out of 100 runs, for different starting temperatures $T_0$. All other parameters are as in table 2. As one would expect, lowering the starting temperature $T_0$ decreases the computation time, but increases the amount of not returning a global minimum.

<table>
<thead>
<tr>
<th>Variation on $T_0$</th>
<th># returning a global minimum out of 100 runs</th>
<th>duration of 1 run (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no variations ($T_0 = 500$)</td>
<td>100</td>
<td>41</td>
</tr>
<tr>
<td>$T_0 = 100$</td>
<td>100</td>
<td>34</td>
</tr>
<tr>
<td>$T_0 = 50$</td>
<td>100</td>
<td>28</td>
</tr>
<tr>
<td>$T_0 = 10$</td>
<td>100</td>
<td>18</td>
</tr>
<tr>
<td>$T_0 = 5$</td>
<td>83</td>
<td>13</td>
</tr>
<tr>
<td>$T_0 = 1$</td>
<td>17</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: Variations of the starting temperature $T_0$ on table 2.

Likewise, table 4 and 5 contains the same experiment, except for different numbers of repetitions $N_p$ and cooling factors $\alpha$ respectively. The tables show SA is able to return a global minimum, with high probability, if the starting temperature $T_0$ is chosen highly enough, and the cooling happens slowly enough ($N_p$ and $\alpha$ large enough).

<table>
<thead>
<tr>
<th>Variation on $N_p$</th>
<th># returning a global minimum out of 100 runs</th>
<th>duration of 1 run (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no variations ($N_p = 1000$)</td>
<td>100</td>
<td>41</td>
</tr>
<tr>
<td>$N_p = 500$</td>
<td>100</td>
<td>21</td>
</tr>
<tr>
<td>$N_p = 250$</td>
<td>98</td>
<td>11</td>
</tr>
<tr>
<td>$N_p = 50$</td>
<td>78</td>
<td>2</td>
</tr>
<tr>
<td>$N_p = 10$</td>
<td>18</td>
<td>$&lt; 1$</td>
</tr>
</tbody>
</table>

Table 4: Variations of the number of repetitions $N_p$ on table 2.
Variation on $\alpha$ & \# returning a global minimum out of 100 runs & duration of 1 run (seconds) \\
no variations ($\alpha = 0.95$) & 100 & 41 \\
$\alpha = 0.8$ & 99 & 10 \\
$\alpha = 0.5$ & 77 & 4 \\

| Table 5: Variations of the cooling factor $\alpha$ on table 2. |

8.4 Variations in the Initial State

From the acceptance probability

$$\alpha_{xy} = \min \left( 1, \exp \left[ \frac{g(x) - g(y)}{T} \right] \right),$$

one can see that if $T$ is big, relatively to $|g(x) - g(y)|$, the probability to accept an ascends, can still be of significant size. For the minimizing state $A_{\text{min}}$ from 8.1.3, the value for the objective function is $g(A_{\text{min}}) = 0$. The maximum value is obtained with the zero matrix $g(0_{I \times J}) = 960$. We can calculate the acceptance probability, for the extreme case where we propose the worse possible state (zero matrix), when the current state is the global minimum point $A_{\text{min}}$. This acceptance probability gives a lower bound for all the acceptance probabilities, for fixed $T$. For $T = 500$, $x, y \in S$, we have

$$1 \geq \alpha_{xy} \geq \alpha_{A_{\text{min}}, 0_{I \times J}} = \exp \left[ \frac{g(A_{\text{min}}) - g(0_{I \times J})}{T} \right] \approx 0.15.$$ 

For the Spray Neighborhood structure, proposals where all $I$ rows are altered are not possible (only 1 row is altered for each proposal). Therefore the lower bound for $\alpha_{xy}$ is even greater in our implementation.

Since the starting temperature was set to $T_0 = 500$, during the first $N_p = 1000$ repetitions, each proposal state has a significant probability to be accepted. For one run with the parameters as in table 2, about 95% of all proposal were accepted during the first $N_p$ repetitions. This high proportion of accepted states in the early phase, can also be observed from the graph in fig. 11, where the value of $g$ is highly oscillating.

We expect, that the initial state $X(0)$ can therefore have practically no influence on the final state $X(t_{\text{final}})$.

To show this, we did 1000 runs, where the number of repetitions was set to $N_p = 50$ (the other parameters as in table 2).

500 of these runs had $A_{\text{min}}$ as the initial state. The other 500 had $0_{I \times J}$ (zero matrix) as the initial state.

Table 6 shows the amount of final states equal to a global minima, for each of the 500 runs. The small difference backs the theory.
Table 6: Variations with the initial state, where $N_p = 50$ and the other parameters as table 2. For both initial state, 500 SA runs are done. The number of final states which are a global minimum points are counted.

<table>
<thead>
<tr>
<th>Used Initial State</th>
<th># of final states which are a global minimum, out of 500 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0_{I \times J}$ (zero matrix)</td>
<td>392</td>
</tr>
<tr>
<td>$A_{\min}$ (global minimum)</td>
<td>388</td>
</tr>
</tbody>
</table>

Table 7: Variations in $T_{\text{stop}}$ and the number of runs which found the optimal solution (out of 100).

<table>
<thead>
<tr>
<th>Variation on $T_{\text{stop}}$</th>
<th># returning a global minimum out of 100 runs</th>
<th>duration of 1 run (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no variations ($T_{\text{stop}} = 0.5$)</td>
<td>100</td>
<td>41</td>
</tr>
<tr>
<td>$T_{\text{stop}} = 5$</td>
<td>100</td>
<td>28</td>
</tr>
<tr>
<td>$T_{\text{stop}} = 10$</td>
<td>91</td>
<td>21</td>
</tr>
<tr>
<td>$T_{\text{stop}} = 15$</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>$T_{\text{stop}} = 20$</td>
<td>1</td>
<td>19</td>
</tr>
</tbody>
</table>

8.5 Variations in the Stopping Criteria

Table 7 shows the effect of using a higher stopping temperature $T_{\text{stop}}$, again by performing 100 runs. Raising $T_0$ to the level that ascends are still frequently accepted, will result in poor results, regardless the other parameters and used neighborhood structure.

8.6 Improve the result for $N_p = 10$

For $N_p = 10$, 18 out of 100 runs returned a global minimum. By using different neighborhood structures, we can possibly increase the number of successful runs (returning a global minimum). We do this by trying different values of $k$ (the number of tasks in the suggestions set), and different weights for the suggestion set $\mathcal{J}$, used in the Spray neighborhood structure. Table 8 shows the weights have great impact on the results. Interesting observations are the poor performance of the CAN-DO weights which we cannot explain. Also, the CAN-FINISH weights by itself gives bad results, possibly due a high occurrence of zero weight vectors, happening when the staff member cannot finish a single tasks. However, CAN-FINISH
in combination with the IS-USEFUL weight, seems to be more effective than IS-USEFUL weights alone.
The weights combination, with $k = 1$ or $k = 5$:

$$\text{IS-USEFUL} + 100 \times \text{CAN-FINISH},$$

shows a significant improvement over the CAN-DO weights, as the amount of successful runs is about 5 times higher.
For staff member $i$, this combination prioritize (100 times more weight) tasks which he can finish (if the staff member isn’t already assigned to them). If these ‘can-finish’ tasks are not available, tasks where he is still useful are selected for the suggestion set $J$ (if he isn’t already assigned to them).

9 Global recommendations for setting the SA parameters

From the experiments in test case 1, we have set up some global recommendations to choose the parameters.

9.1 Initial State

The $I \times J$ zero matrix can be chosen as the initial state. Experiments shown the initial state has no influence on the result.

9.2 $N_p$, $\alpha$, $T_0$ and $T_{\text{stop}}$

$\alpha$ can be chosen $0.8 < \alpha < 1$, and $N_p \geq 10$. Bigger values will improve the results, but increases computations time as well. There is a balance between calculation time and performance, it is up to the user to make a suitable choice for his application.
One should choose $T_{\text{stop}}$ small enough to ensure the algorithm has entered a stable phase, where ascends are unlikely to happen. By keeping track of the current objective function value, one can observe for which temperature the algorithm has stabilized to the level that (almost) no ascends occur anymore. $T_0$ should be chosen of such amount, that most ($> 90\%$) proposals are accepted in the early phase of the annealing process.

9.3 Weights and $k$

The combination

$$\text{IS-USEFUL} + 100 \times \text{CAN-FINISH},$$

was shown to significantly improve results for test case 1.
Weights: | # returning a global minimum out of 100 runs | duration (seconds) |
---|---|---|
no variations (CAN-DO weights, $k = 1$) | 18 | < 1 |
CAN-DO, $k = 5$ | 25 | < 1 |
CAN-DO, $k = 1$ | 0 | < 1 |
CAN-DO, $k = 5$ | 1 | < 1 |
IS-USEFUL, $k = 1$ | 77 | < 1 |
IS-USEFUL, $k = 5$ | 78 | < 1 |
IS-USEFUL, $k = 1$ | 79 | < 1 |
IS-USEFUL, $k = 5$ | 78 | < 1 |
CAN-FINISH, $k = 1$ | 5 | < 1 |
CAN-FINISH, $k = 5$ | 5 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 1$ | 39 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 5$ | 44 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 1$ | 39 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 5$ | 41 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 1$ | 84 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 5$ | 80 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 1$ | 88 | < 1 |
IS-USEFUL + 100 $\times$ CAN-FINISH, $k = 5$ | 80 | < 1 |

Table 8: Variations in the weights and $k$, where the number of repetitions was set to $N_p = 10$, and the remaining parameters as in table 2.
We recommend to try different weights and $k$ values, and linear combinations similar to the above. More experiments are needed to show if combinations exist which perform well on assignment problems of any size and design. If one is unsure about which weights to use, one could use the following approach: define $z$ weights, and for each proposal randomly decide which of the $z$ weights to use. By keeping track of the proportion of accepted states, one gets an indication for the effectiveness.

10 Test Case 2 — Computer (SA) versus Humans

Next, we compare how SA performs against a test panel of humans. For this experiment, we constructed a planning where a global minimum / solution is not easy to see.

To help our human test panel, a graphical user interface (GUI) was written in the form of a HTML5 web page.

The web page (fig. 12) showed tasks and staff members. The staff members can be drag and dropped to the tasks. Besides the drag and drop feature, it has the following features:

- Color indication for the tasks. Tasks where all the required skills are meet are marked green, the remaining red. Figure 12 shows task 5 is

![Screen-shot of the interactive HTML5 page to aid our human test team with creating a planning for test case 2.](image)
marked green, meaning enough skills have been assigned.

- Color indication for the staff members. Unassigned staff members are indicated by a green dot. Staff members which are assigned to tasks, but still have time available for other tasks are indicated by an orange dot. Staff members with no time available for other tasks are indicated by a red dot.

- If the user assigns a staff member to task \( j \), which causes overlap with \( k \), the web page automatically de-assigns the staff member from \( k \). This automatic de-assignment helps the user to ensure no plannings with overlap occur.

- Each skill was marked by a unique color, making it easy for the human eye to grasp which staff members are useful for which task.

- Once submitted, the web page saves the planning, the elapsed time in seconds, and the cost.

A test panel was told to use as much time as needed, and at least use 10 minutes to get familiar with the GUI and the planning problem.

### 10.1 Problem Formulation

The constructed problem contains \( J = 12 \) tasks, and \( I = 11 \) staff members, with \( N = 4 \) skills.

#### 10.1.1 Required Skill matrix and Available Skill matrix

The work matrix \( W = [\vec{w}_1, \ldots, \vec{w}_{12}] \) is defined by the following 12 tasks:

\[
\vec{w}_1 = (1, 10, (1, 2, 1, 1)') \\
\vec{w}_2 = (11, 22, (2, 1, 1, 2)') \\
\vec{w}_3 = (23, 30, (3, 3, 2, 3)') \\
\vec{w}_4 = (1, 6, (4, 2, 3, 0)') \\
\vec{w}_5 = (7, 12, (1, 0, 2, 1)') \\
\vec{w}_6 = (13, 20, (3, 4, 0, 3)') \\
\vec{w}_7 = (21, 28, (1, 2, 4, 2)') \\
\vec{w}_8 = (29, 32, (2, 1, 3, 1)') \\
\vec{w}_9 = (1, 9, (2, 2, 2, 2)') \\
\vec{w}_{10} = (11, 19, (1, 1, 2, 1)') \\
\vec{w}_{11} = (21, 25, (3, 1, 1, 2)') \\
\vec{w}_{12} = (27, 31, (1, 2, 0, 1)').
\]
The overlap matrix $P$ is given in the appendix.
The available skill matrix $S$ is defined by the following 11 staff members:

$$
S = \begin{pmatrix}
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1
\end{pmatrix}.
$$

One can observe from the available skill matrix, that staff member 1 and 2 can be assigned to task 5, to meet the task’s required skill (as is shown in fig. 12).

### 10.1.2 Cost function

The costs are calculated with $p = (1, 1, 1, 1)'$, and task-time vector $t = (1, 1, \ldots, 1)$, with the cost function from section 4. The task-time vector was chosen for its simplicity, and doesn’t contains the actual task durations.

### 10.2 Results

Table 10 contains the cost and the used calculation time, used for both the SA algorithm and each of the persons of the test panel. For the SA-algorithm, the parameters from table 9 were used.
SA found the lowest possible cost in all 100 runs (the assignment matrix for one of these runs with 0 cost can be found in the appendix). Not one of the persons in the test panel managed to get a cost of 0.

<table>
<thead>
<tr>
<th>Human / SA</th>
<th># spend (seconds)</th>
<th>Cost (lower is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Person 1</td>
<td>706</td>
<td>3</td>
</tr>
<tr>
<td>Test Person 2</td>
<td>778</td>
<td>5</td>
</tr>
<tr>
<td>Test Person 3</td>
<td>836</td>
<td>3</td>
</tr>
<tr>
<td>Test Person 4</td>
<td>939</td>
<td>9</td>
</tr>
<tr>
<td>Test Person 5</td>
<td>1499</td>
<td>12</td>
</tr>
<tr>
<td>Test Person 6</td>
<td>891</td>
<td>8</td>
</tr>
<tr>
<td>Test Person 7</td>
<td>942</td>
<td>3</td>
</tr>
<tr>
<td>SA</td>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 10: Humans versus Computer (SA) results. With the used parameters, SA found a global minimum in 100 out of 100 run. One of these runs took about 22 seconds.

11 Test Case 3 — The Railway case

For a Dutch railway construction company, an adapted version of the SA algorithm with neighborhood structure 7.1.2 was applied for the scheduling of the construction staff. The railway construction company performs construction projects, and maintenance activities. The work and construction staff are very versatile, a typical example of a multi skilled environment.

For the test case, all tasks in the planning took place in the past. Therefore the output can not be used as the actual planning, but can be compared to the realized planning. 73 staff members are scheduled on 126 tasks that took place in a time period of 3 days, on different locations in the Netherlands.

The total amount of work requested 1400 man-hours.

11.1 Current Situation

The company has 3 planners which manually schedule the available construction staff among the projects. Planning software is used, but does not assist the planners by creating suggestions.

If for a project insufficient construction staff is available, external staff is hired. In general, external staff requires more time for the same work, and requires supervision of the internal staff. For these reasons, project managers
prefer internal staff over external staff. If external staff is assigned to their projects, they wish no more than 4 out of 5 of the staff is external, since one internal staff member can supervise at most 4 externals a time.
The external staff is paid a fixed amount for each hour they are hired. The amount of hired external staff is therefore a big component of the costs for a project. Evidently, efficient assignment of the internal staff reduces the amount of hired external staff. With the help of the SA algorithm we hope the amount of external hired staff hours can be reduced.
Problems the planners are currently facing are:

- Spread the hired external staff evenly among all works, to get a fair internal/external staff proportion that the project managers find workable.
- Efficiently plan the internal staff along their requested vacation.

### 11.1.1 Constraints

The planners try to assign the internal staff as good as possible, and have to deal with the following constraints:

1. **(union rules)** Union rules and work regulations demand that each staff member works in shifts of maximal $S = 9.5$ hours, after which he enjoys his mandatory rest period of minimum $R = 12$ hours.

2. **(vacation requests)** Deal with the vacation requests of each staff member. Staff members can only be assigned during their available working days.

3. **(internal/external balance)** Spread the external hired staff equally among all works, preventing some tasks are only employed by external staff. The request of the projects managers is that on each 5 staff members, at least 1 is internal staff. This amount is rounded to above, so for 7 assigned staff members, at least 2 should be internal, and 5 can be external. If for a task $j$, $\mu_j$ internal and $\xi_j$ external staff members are assigned to meet all the required skills, the number of external staff members that violates the internal/external balance constraint is defined as

$$\gamma_j = \max \left( 0, \left\lceil \frac{\mu_j + \xi_j}{5} \right\rceil - \mu_j \right).$$

4. **(travel time)** Take the travel time between the various projects into consideration.
11.2 Railway Test Case Planning Setup and Modifications

$I = 73$ internal construction staff members are assigned to $J = 126$ tasks, yielding an optimization problem over a space containing $2^{9198} \approx 10^{2768}$ elements.

$N = 92$ different skills are used to describe each staff member. The available skill matrix $S$, the required skill matrix $W$, and time vector $t$ are directly obtained by rearranging the existing staff and project information. Due to confidentiality reasons, this document doesn’t contain the matrices $S$ and $W$.

We assume all the external staff members possess all the $N = 92$ skills, hence the available skill matrix of an external staff member, $s_{\text{external}}$, is set to

$s_{\text{external}} = (1, \ldots, 1)$.

If after assignment of the internal staff, tasks still have required skills which are not met, we assume enough external staff can be hired. For example, if for a task $(3, 5, 2)$ is the required skill, with no internal staff assigned, the company can always hire 5 externals, such that all the required skills are met (in this example the internal/external balance constraint is violated).

For this reason, we do not actually schedule the external staff. We assign the internal staff, and calculate the necessary external hired staff hours, needed to meet each task’s required skill.

The total amount of external hired hours is a good indicator for a good planning.

11.3 Soft and Hard constraints

The union rules, vacation requests, and travel time all are hard constraints — they have to be satisfied anyhow. The internal/external balance is a soft constraint, highly preferable to take into consideration, but not crucial (cases exists where it was violated). For the implementations of the hard constraints we choose to never propose states violating them, by adapting the neighborhood structure. The soft constraint, handling the internal/external balance, is implemented by inquiring an additional penalty in the cost function.

11.3.1 Travel-Overlap Matrix $T$

Each task is associated with a location. By introducing travel-overlap, travel time between various tasks can easily be implemented.

Say $w_1$ and $w_2$ are two tasks, where $w_1$ starts before $w_2$. Two tasks $w_1, w_2$ travel-overlap if: they overlap, or the staff employee can not arrive on time at the location of $w_2$, after finishing $w_1$ (considering the travel-time).
Note that for two tasks with the same location, travel-overlap is the same as overlap.

Figure 14 shows an example of two tasks which travel-overlap, although they do not overlap in our conventional definition.

Figure 14: Tasks $w_1$ and $w_2$, travel-overlap, but do not overlap. The dashed arrow represents the travel time. A staff member that finished task $w_1$ and travelled to $w_2$, would arrive late.

The definition of travel-overlap requires knowledge of the travel-time between each pair of tasks. We simplified the travel-time to 1.5 hours, if the tasks are not executed on the same location, and 0 otherwise.

**Definition.** Define the $J \times J$ travel-overlap matrix $T = [t_{j,k}]$ as

$$t_{j,k} = \begin{cases} 
J & \text{if } j = k, \\
1 & \text{if task } j \text{ and } k \text{ overlap or travel-overlap, for } j \neq k \\
0 & \text{else.}
\end{cases}$$

By replacing the overlap matrix $P$ by the travel-overlap matrix $T$, one takes all the travel times between tasks into consideration. For other applications one could similarly use this method, like cleaning time or preparation time between tasks.

### 11.3.2 Deal with the internal/external balance

For the cost function, the conventional costs of section 4 are used, with two additional terms, an 'external hire' component (1), and an 'internal/external spread' component (2). The used penalty vector for the conventional costs was set to $p = (1, 1, \ldots, 1)$.

The first component, is the sum of external hire hours, and is defined as

$$C_{\text{external hire}}(A) = \text{Col-max} \left[ \max(W - SA, 0_{J \times N}) \right] t.$$ 

Some justification: the $j^{\text{th}}$ column of $W - SA$ contains positive components, for required skill which are not met. If $m$ is the maximum for column/task $j$, then $m$ external staff members with the right skills would make sure task $j$ can be performed well. Hence the Col-max, which returns a vector of length $J$, is taken over $\max(W - SA, 0_{J \times N})$ to calculated the necessary amount of external staff for each task. Multiplied with the task time-vector $t$ results in the amount of external hired hours.
The second component counts the total amount of external staff members which violates the internal/external balance. If vector $\gamma = (\gamma_1, \ldots, \gamma_J)$ defined the number of external hired staff members that violates the internal/external balance constraints for each task, we define $C_{\text{internal/external}}$ as

$$C_{\text{internal/external}}(A) = \|\gamma\|_1.$$ 

With the two additional components defined, and scalars $F_{\text{spread}}, F_{\text{hire}} \in \mathbb{R}$, the railway cost function is defined as

$$C_{\text{Railway}}(A) = C(A) + F_{\text{hire}} C_{\text{external hire}}(A) + F_{\text{spread}} C_{\text{internal/external}}(A).$$

The scalars $F_{\text{spread}}$ and $F_{\text{hire}}$ puts different weight on each component of the cost function. If $F_{\text{spread}}$ is chosen too low, the soft constraint internal/external balance, would not be taken enough into consideration. After several test runs, the following values returned good results and are used:

$$F_{\text{hire}} = 15, \quad F_{\text{spread}} = 150.$$ 

Ultimately, the interest of the company is reducing the total amount external hired hours, the $C_{\text{external hire}}$ part of the cost function. We don’t directly minimize on the external hire hours, since its behavior is too coarse. By minimizing on the finer $C_{\text{Railway}}(A)$, Simulated Annealing performed better, most likely due to that the $C(A)$ component helps searching in the right direction between states, where the more coarse $C_{\text{external hire}}$ would remain constant.

### 11.3.3 The neighborhood structure modifications

For the neighborhood structure, we make 3 modifications, for each of the 3 hard constraints.

**Creating Shifts, with $k = \infty$**  With $k = \infty$, a suggestion set $\tilde{J}$ is constructed like described in . However, we want to have that for each task $j \in \tilde{J}$:

$$\max_{j \in \tilde{J}} b_j - \min_{j \in \tilde{J}} a_j \leq S,$$

which would construct a shift for staff member $i$. Here $S$ denotes the shift duration in hours. Recall that $a_j$ and $b_j$ denotes the start and end time of task $j$ respectively, which unit is set to hours for the railway case.

The additional criterion is to only add tasks to $\tilde{J}$ which would not make the shift longer than $S$ hours. Applying this criterion gives the following scheme:
Suggestion set $\mathcal{J}$ for the Railway Spray Neighborhood structure

**Construction of suggestion set $\mathcal{J}$ for shift length $S \in \mathbb{R}$, given $A_x$ and $i$.**

For $\mathcal{S}$, we construct a set of non-overlapping tasks $\mathcal{J}$, by

1. Set $\mathcal{J} = \emptyset$.
2. Define a weight vector $\eta = (\eta_1, \ldots, \eta_J)$.
3. Draw $j \sim \eta_j$.
4. Update weight vector $\eta$, by setting
   
   $\eta_j = 0$ for all $j \in \mathcal{N}_3(T)$,
   
   and $\eta_j = 0$ for all $j$ for which
   
   $\max_{k \in \mathcal{J} \cup \{j\}} b_k - \min_{k \in \mathcal{J} \cup \{j\}} a_k > S$.

   Here, for travel-overlap matrix $T$,
   
   $\mathcal{N}_3(T) = \{ j : T[j, l] = 1, \text{ for some } l \in \mathcal{J} \}$.

5. Return $\mathcal{J}$ if:
   
   $\|\eta\| = 0$.

   Else, repeat from step 3.

To implement the mandatory rest period, a rest-overlap matrix is firstly calculated, similar to the travel-overlap. Tasks $w_1$ and $w_2$ rest-overlap if one task falls into the rest period of the other. For the railway case, this rest period is set to 12 hours.

**Definition.** Define the $J \times J$ rest overlap matrix $R = [r_{j,k}]$ as

\[
r_{j,k} = \begin{cases} 
J & \text{if } j = k, \\
1 & \text{if task } j \text{ and } k \text{ overlap or rest-overlap, for } j \neq k \\
0 & \text{else}.
\end{cases}
\]

Now $\mathcal{N}_3$ in 7.1.2 (the Spray neighborhood structure, for some suggestions set $\mathcal{J}$) is calculated with the rest-overlap matrix $R$ instead of $P$. De-assignment now assures the mandatory rest is always taken into account.

Shifts and rest periods are now taken care of. Vacation requests are handled by adaption the weights.
11.3.4 Weights and vacation requests

Let \( \eta \) be some weight vector for staff member \( i \). To respect all the individual vacation request for \( i \), vector \( \eta^v \) is set to 0 for each component/task that falls into a vacation period (vacation task):

\[
\eta^v_j = \begin{cases} 
\eta_j & \text{if task } j \text{ does not fall in a vacation period.} \\
0 & \text{if task } j \text{ falls in a vacation period.} 
\end{cases}
\]

11.3.5 Weights

After several runs and experiments similar as in test case 1, the following combination of weight vectors returned good results:

\[
\begin{align*}
\mathfrak{A} &= \text{CAN-DO}^v \\
\mathfrak{B} &= \text{CAN-DO}^v + 200 \text{IS-USEFUL}^v \\
\mathfrak{C} &= \text{CAN-DO}^v + 20 \text{IS-USEFUL}^v + 200 \text{CAN-FINISH}^v \\
\mathfrak{D} &= \text{diag}(\text{SCARCITY}) \mathfrak{A} \\
\mathfrak{E} &= \text{diag}(\text{SCARCITY}) \mathfrak{B} \\
\mathfrak{F} &= \text{diag}(\text{SCARCITY}) \mathfrak{C}
\end{align*}
\]

Like in test case 1, each weight reflect principles a planner might consider. \( \mathfrak{C} \) prioritizes tasks which the staff member can finish, but if these are not available tasks where he is still useful have the priority to be assigned to in his new shift. If he is not useful anywhere, he is put on a task he is able to do, by means of possessing relevant skills. For each proposal, one of the 6 above weight vectors was picked uniformly.

11.4 Railway Case SA parameters

The railway case specific parameters are

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S ) (shift lengths in hours)</td>
<td>9.5</td>
</tr>
<tr>
<td>( R ) (mandatory rest in hours)</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 11: Railway case specific parameters

All SA specific parameters are set as in table 12.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Chosen value / method</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>250</td>
</tr>
<tr>
<td>$N_p$</td>
<td>500</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>.95</td>
</tr>
<tr>
<td>$k$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Weights</td>
<td>Randomly picked from {A, B, C, D, E, F}</td>
</tr>
<tr>
<td>$T_{\text{stop}}$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 12: SA parameters used to plan the Railway case.

11.5 Quality of output planning

With the parameters as in the previous section 11.4, a planning was created and handed over the construction company. The planning was received very well. In general, a reduction of the hired external staff members was achieved.

To test SA for the railway company, lots of assumptions had to be made to construct the available skill matrix $S$, required skill matrix $W$, and travel time. The hired external hours from the output of the SA-algorithm are for this reason not directly comparable to the actual realized hired hours, though it was a reduction of about 50%.

However, the stability of the algorithm and the impact of a reduction of internal staff can be quantified by experiments.

11.6 200 Re-Runs

To tell something about the stability of the algorithm, we performed 200 test runs, with identical parameters as stated in table 12.

In 100 of these runs, we used all the available staff. In the other 100 runs, we randomly removed 10 internal staff members. For both runs, box-plots are made showing the amount of hours external staff was hired.

11.6.1 Results with all internal staff available

In fig. 15 we find a box-plot of the amount of hours, external staff is hired for each of the 100 runs, giving an indication of the stability of SA. All the work in total demands 1400 man-hours.
If a staff member does not requests any vacation during the 3 days the projects took place, he is obliged to take his mandatory rest between shifts and therefore cannot work 24 hours a day. If for staff member $i$, $\rho_i$ denotes the amount of hours he is available, and $\tau_i$ the amount of hours he is assigned to tasks, then the proportion:

$$\pi_i = \frac{\tau_i}{\rho_i}$$

tells something about the efficiency of assignment.

The maximum of 1 can not always be obtained because of the work regulations. If a staff member requests to be available only from 00:00 till 08:00, he can be assigned 100% of his available time, since the rest period falls outside his available time frame. However, if he is available from 00:00 to 23:59 on each day, the rest period falls inside his available time and he cannot be assigned 100% during his available time frame.

Figure 16 contains a box-plot of the $\pi_i$’s for each staff member, of the planning associated with the minimum amount of external hired hours (219 hours), where fig.17 contains a box-plot of the $\pi_i$’s of the planning associated with the maximum amount of external hired hours (264 hours). Since

$$\frac{S}{S + R} = \frac{9.5}{9.5 + 12} \approx 0.44,$$
one expects staff members which are available on full days, are assigned 44% of their available time. 53 out of the 73 staff members are willing to work on full days. The remaining 20 requested only to work during day time or at night.

Outliers above are due to assignments where the rest period completely falls outside his available time. Both plots show a high efficiency was obtained. Individual checkups shows that the staff members are indeed assigned to clusters of tasks of about 9.5 hours, and that the minimum rest period of 12 hours was frequently found to be no more than 13 hours.

Figure 16: Box-plot of the efficiency of assigning for the minimum planning.
Figure 17: Box-plot of the efficiency of assigning for the maximum planning.

Planning which returned the maximum amount of hired externals

11.7 Impact on working with 10 less staff members

With 10 less available staff members, we performed 100 re-runs with identical parameters. For each of the 100 runs, 10 staff members are randomly picked and removed from the available internal staff members.

In fig. 18 we find a box-plot of the amount of hours, external staff is hired for each of the 100 runs. As one expects, the box-plot shows more hours of hired external staff than in the box-plot of fig. 15.
Figure 18: Box-plot of the amount of necessary hired external hours, where 10 internal staff members were randomly removed in each run.

With $\pi_i$ the proportion of time staff member $i$ is working from his available time, fig. 19 contains a box-plot of the $\pi_i$'s for each staff member, of the planning associated with the minimum amount of external hired hours with 10 less staff members (294 hours).

Figure 20 contains a box-plot of the $\pi_i$ for each staff member, of the planning associated with the maximum amount of external hired hours with 10 less staff members (390 hours).
Figure 19: Box-plot of the efficiency of assigning for the minimum planning. 10 internal staff members were removed.
The box-plots and individual checkups again show a high efficiency of assignment.

11.8 Implementation in Planning Software

The construction company was very pleased with the introduction of a tool which can improve the quality of the planning. On the moment of writing, they are exploring the possibilities to implement the algorithm into their planning process.

12 Conclusions & Recommendations for Future Work

Simulated Annealing with our used (mixed) spray neighborhood structure, has proven to be a powerful tool to improve the quality of planning, in toy examples as well as the case where it successfully scheduled the construction staff on multiple projects. Where humans have difficulties to comprehend the complexity of planning problems of significant size, Simulated Annealing returns good results within acceptable calculation time.
With the currently implemented cooling schedule, the user is required to set several parameters ($T_0$, $T_{stop}$, $N_p$, $\alpha$), which when wrongly chosen leads to poor results after excessive calculation time.

Ideally one would use a cooling schedule with less parameters involved in a more accessible manner. Besides simplifying the usage, different cooling schedules can have impact on the performance as well.

A related planning problem untouched is which of the $I$ staff members are best to hire/use, and which shall remain idle. Related situations are companies that frequently hire external staff from employment agencies, and have to make a choice in which staff members to hire from the available pool, and how to assign them to the tasks. But the problem also rises if the demand of work is of such small amount, that not all staff members are necessary to assign.

Within our model this translates to: Given a set of staff members $\{1, 2, \ldots, I\}$ with associated skill vectors, finding a subset of staff members, $Z \subset \{1, 2, \ldots, I\}$, with corresponding skill vectors, that minimizes planning costs. Realistic adoptions have to be made in the cost function. A new neighborhood structure have to be composed that smartly picks from the available staff during the annealing process.
APPENDIX

A Matrices for Test Case 1

Overlap matrix $P$

$$
\begin{pmatrix}
J & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & J & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & J & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & J & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & J & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & J & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & J & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & J & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & J \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & J
\end{pmatrix}
$$

Task-time vector $t$

$$(30, 30, 20, 20, 15, 15, 15, 15, 12, 12, 12, 12, 12, 12, 12)'$$

B Matrices for Test Case 2

Overlap matrix $P$

$$
\begin{pmatrix}
J & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & J & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & J & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & J & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & J & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & J & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & J & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & J
\end{pmatrix}
$$
An assignment matrix $A$ with cost 0, found with SA with the parameters as in table 9:

$$A = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 
\end{pmatrix}.$$  

References


