Chapter 2

# BATCH PRESORTING PROBLEMS. I MODELS AND SOLUTION APPROACHES

This chapter is organized as follows: at first, we describe the problem and give a short classification. In Section 2.2 different formulations of the BPSP are presented. In Subsection 2.2.2 we consider an optimization version of $BPSP_1$. In Subsection 2.2.3 we formulate $BPSP_2$ and $BPSP_3$ as decision problems and additionally introduce optimization models. The complexity status of $BPSP_2$ is investigated in Section 2.3, and in Section 2.4 we show that there is a polynomial version of the BPSP. Also we consider a special subcase of a BPSP with $N^L = 2$ in offline and online situations and present corresponding algorithms in Section 2.5. Finally, in Section 2.6, some results derived for BPSPs with $N^L = 2$ are adapted to general BPSP.

## 2.1. Problem Description and Classification

We consider the problem of finding a finite sequence of objects of different types, that guarantees an optimal assignment of objects to given physical storage layers with a pre-sorting facility of limited capacity. This problem will be called the *Batch PreSorting Problem* (BPSP), because the objects have to be sorted within one batch before they are assigned to the layers. After sorting, the object with number $i$ will be assigned to layer $i$. For a more transparent presentation we speak of colors instead of types and thus consider all objects of type $k$ as having the same color $k$. We present three types of BPSP with different objective functions. The

objective function, $z$, of $\text{BPSP}_1$ minimizes the total number of layers not yet occupied by objects of a certain color $k$; such objects can be considered as occupying an empty layer (empty *w.r.t.* to $k$) at zero cost. Once each layer has an object of a given color, the cost does not change with further additions of that color. If the forgoing is true for all colors, $z$ gives the number of all objects to be distributed minus those already assigned to the layers. In $\text{BPSP}_2$ the objective is to minimize the maximum number of objects of the same color on the same layer. Finally, $\text{BPSP}_3$ aims to minimize the sum of the maximum number of objects of the same color over all layers.

We use the following example to illustrate the problem:

EXAMPLE 2.1 *Suppose, there are six objects of two different colors in the input sequence (see Fig. 2.1.1) and three layers.*

*Objects can be sorted within one batch, i.e., the objects 1, 2, 3 can be sorted, then they are assigned to the layers. After this the objects 4,5 and 6 can be sorted and assigned to the layers. Fig. 2.1.1 displays the content of the layers without pre-sorting. For this assignment the objective function value of* $BPSP_1$ *is* 2, *because the objects of the first batch occupy the layers at zero cost (layers were empty); the objects 4 and 5 occupy the layers 1 and 2, respectively, each with cost one, and object 6 occupies layer 3 at zero cost. The objective function value of* $BPSP_2$ *is* 2, *because the maximal number of objects of any color on all layers is* 2. *Finally, the objective function value of* $BPSP_3$ *is* 4, *because the maximal number of objects of the colors 1 and 2 over all layers is* 2 *for both colors. Clearly, this assignment is not optimal w.r.t. none of the three objective functions. The optimal objective function values for* $BPSP_1$, $BPSP_2$, *and* $BPSP_3$ *are* 0, 1, *and* 2, *respectively (see Fig. 2.1.2).*

## 2.2. Formulation of the Batch Presorting Problem

At first we introduce some notations used in this chapter:

- $N^O$ is the number of objects of different colors in a given sequence. These objects are indexed by $i$ or $j$ (for simplicity the positions of the objects are identified by their index values). $\mathcal{S}_k$ is the set of objects of color $k$, and $i \in \mathcal{S}_k$ means that the object at position $i$ in the sequence (also called "$i^{\text{th}}$ object" or "object $i$" for short) has color $k$;
- $N^K$ is the number of colors;
- $N^L$ is the number of layers;
- $N^S$ is the capacity of the pre-sorting facility.

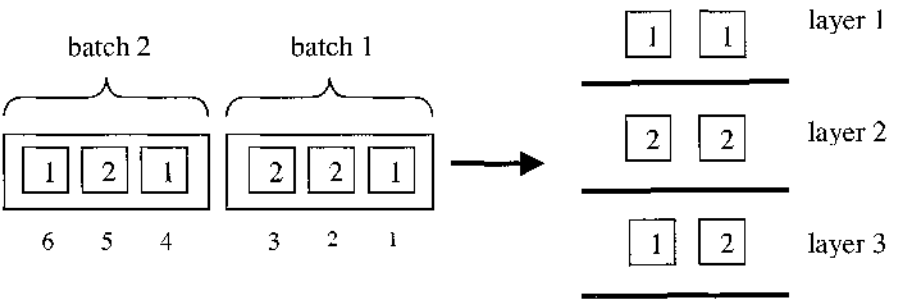


*Figure 2.1.1.* The input sequence and the content of the storage layers without pre-sorting. On the left part of the figure, the numbers 1, 2, ..., 6 refer to the objects while the numbers 1 and 2 in the squares denote the colors.

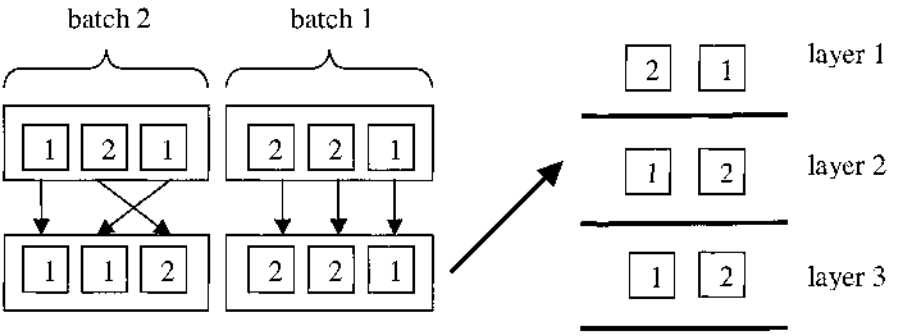


*Figure 2.1.2.* Optimal permutation and content of the storage layers after the assignment

### 2.2.1 Feasible Permutations

Before we talk about feasible permutations, recall the definition of permutation:

DEFINITION 2.2 *A permutation $\delta$ on a set of $N^O$ objects is a one-to-one mapping of set $\{1,\ldots,N^O\}$ onto itself, i.e., $\delta : \{1,\ldots,N^O\} \longrightarrow \{1,\ldots,N^O\}$. Thus $\delta(i) = j$ if the object originally positioned at $i$, is placed onto position $j$.*

In other words, if $\delta$ is a permutation, $\delta(i)$ denotes the position of object $i$ in the output sequence. In our case, only a subset of all possible permutations can be performed using the pre-sorting facility.[1]

[1]For the concrete technical functionality see Chapter 3.

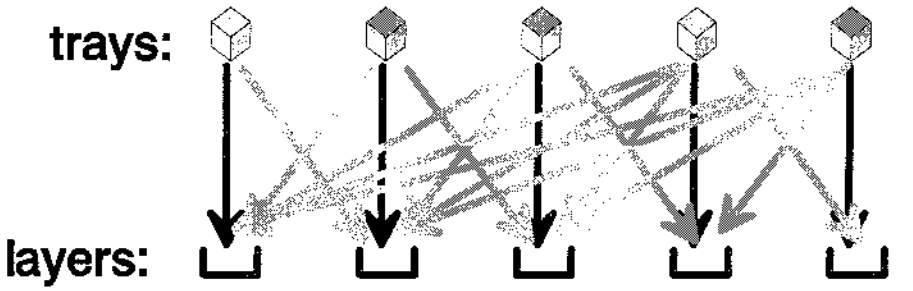


*Figure 2.2.3.* The set of all possible permutations for $N^S = 1$

THEOREM 2.3 *Let $N^S$ be the capacity of the pre-sorting facility. A permutation $\delta$ is realizable, if and only if for each object $i$, $\delta(i) \geq i - N^S$. If $N^O \leq N^S$ then there exist $N^O!$ realizable permutations, otherwise there will be $N^S!(N^S+1)^{N^O-N^S}$.*
***Proof.*** *(see, for instance, [39])* ■

Fig. 2.2.3 illustrates the result of Theorem 2.3. Notice, that if $N^O \leq N^S$ then there exist $N^O!$ realizable permutations and $N^S!(N^S+1)^{N^O-N^S}$ otherwise. In this work the terms *realizable* and *feasible* permutations are equivalent. Now we formally introduce the notion of a feasible permutation.

DEFINITION 2.4 *A permutation $\delta$ is feasible if for any $i = 1, ..., N^O$*

$$\delta(i) \geq i - N^S \tag{2.2.1}$$

*is fulfilled.*

### 2.2.2 Mathematical Formulation of $\mathbf{BPSP_1}$

As was defined above, only the permutations with $\delta(i) \geq i - N^S$ are feasible. Note that only the objects at the permuted positions $\delta(i) = j$ will be placed onto layer $l$, where $l \equiv j \bmod N^L$, *i.e.*, $l$ is a function of $j$. For example, if $N^O = 5$, $N^L = 2$, then objects with positions $j = 1, 3, 5$ will be placed onto layer $l = 1$, those with positions $j = 2, 4$ onto layer $l = 2$.

In addition we introduce the following notations:

$$\delta_{ij} = \begin{cases} 1, & \text{if } \delta(i) = j \\ 0, & \text{otherwise} \end{cases},$$

and

$$C'_{ij} = \begin{cases} 1, & \text{if layer } l,\ l \equiv j \bmod N^L, \text{ has already an object of the same color as object } i \\ 0, & \text{otherwise} \end{cases} . \tag{2.2.2}$$

The optimal permutation can be constructed from the solution of the following linear program [39]:

$$\min \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} C'_{ij}\delta_{ij} \quad , \tag{2.2.3}$$

$$\sum_{j=1}^{N^O} \delta_{ij} = 1, \quad 1 \le i \le N^O \quad , \tag{2.2.4}$$

$$\sum_{i=1}^{N^O} \delta_{ij} = 1, \quad 1 \le j \le N^O \quad , \tag{2.2.5}$$

$$\delta_{ij} = 0, \quad \forall i,j : j < i - N^S \quad , \tag{2.2.6}$$

$$\delta_{ij} \in \{0,1\}, \quad 1 \le i,j \le N^O \quad . \tag{2.2.7}$$

We can interpret the coefficient $C'_{ij}$ as the cost of placing object $i$ onto position $j$ (which uniquely identifies layer $l$). As (2.2.3) minimizes the total placing cost, it minimizes hence the total number of layers not yet occupied by objects of a certain color $k$. Such objects can populate an empty layer (empty *w.r.t.* to $k$) at zero cost. Infeasible permutations are excluded (depending on $N^S$), a priori by (2.2.6). Obviously, (2.2.6) corresponds to (2.2.1).

It is well known that this kind of integer program is totally unimodular (*cf.* [61]) and, thus, may be solved efficiently by some versions of the Simplex algorithm. Many special matching algorithms solve the problem in polynomial time (*cf.*, [72]). In practical applications (see Chapter 3), the performance very often depends on the number of attempts needed to output completely a set of orders (an order is a set of objects of different types). An attempt is considered successful if there exists at least one object of a given color on each layer (*i.e.*, belonging to the requested order). Therefore, for a given set of orders, the number of attempts needed for complete output is the maximum number of objects in these orders found on a single layer.

Consider, for instance, the following example: $N^O = 8$, $N^K = 2$ (*e.g.*, blue and yellow), $N^L = 2$, $C'_{ij} = 1$ for all $i,j$ (*i.e.*, one blue and one yellow object already exist on each layer). Let the first four objects be

blue and the others yellow. Suppose, $BPSP_1$ has two optimal solutions with objective function value 8:

1 Three blue objects are assigned to the first layer and one to the second; one yellow object to the first layer and three to the second.

2 Two objects of each color are assigned to both layers.

The numbers of attempts for complete output are $4 + 4 = 8$ in the first case and $3 + 3 = 6$ in the second (see Fig. 2.2.4). In terms of suffi-

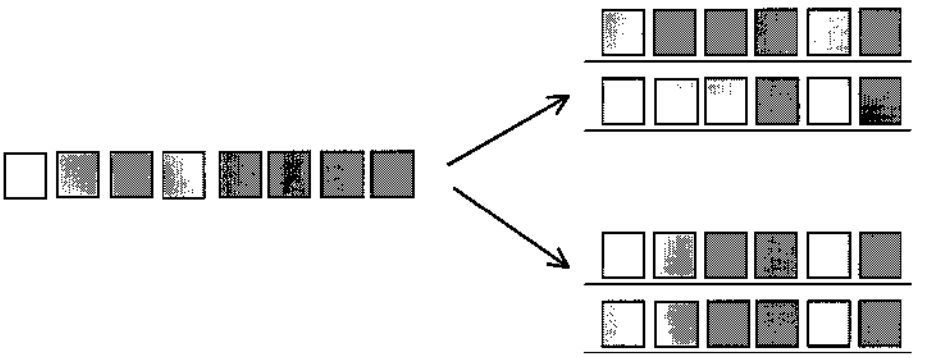

*Figure 2.2.4.* The example of two different assignments of objects to the storage layers.

ciency the second solution is preferable, because it needs fewer attempts for complete output. For practical applications we want to produce a solution with minimal number of attempts. Since $BPSP_1$ does not necessarily do so, we developed the following problem formulations.

Note that the formulation above does not contain the index $k$, because the information about the color of objects is hidden in the coefficients $C'_{ij}$. More precise, $C'_{ij} = C'_{ij}(k)$. Example 2.5 illustrates how the coefficients $C'_{ij}$ are constructed.

### 2.2.3 Mathematical Formulation of $BPSP_2$ and $BPSP_3$

In this section we formulate $BPSP_2$ and $BPSP_3$ as decision problems. Most of the notations used in the previous section will be kept. Analogous to the notation $C'_{ij}$ from Section 2.2.2 we use the notation $C_{kl}$ - the number of objects of color $k$ already present on layer $l$. Additionally we define:

- an integer bound $B$;

- constants

$$S_{ik} = \begin{cases} 1, \text{ if } i \in \mathcal{S}_k \\ 0, \text{ otherwise} \end{cases}, \tag{2.2.8}$$

$$M_{jl} = \begin{cases} 1, \text{ if } j \equiv l \mod N^L \\ 0, \text{ otherwise.} \end{cases}. \tag{2.2.9}$$

This allows us to define

$$D_{ikjl} := S_{ik} M_{jl} \quad . \tag{2.2.10}$$

Now we can formulate the following decision problems:

D-BPSP$_2$: Is there a feasible permutation $\delta$ such that the maximal cost

$$\max_{k=1,\ldots,N^K,\ l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl} \delta_{ij} \right) \tag{2.2.11}$$

does not exceed $B$?

D-BPSP$_3$: Is there a feasible permutation $\delta$ such that the total cost

$$\sum_{k=1}^{N^K} \max_{l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl} \delta_{ij} \right) \tag{2.2.12}$$

does not exceed $B$?

**Remark:** The term $D_{ikjl}\delta_{ij}$ takes the value 1 if an additional object $i$ of color $k$ is placed onto layer $l$ by permutation $\delta$. As $C_{kl}$ denotes the number of objects of color $k$ already present on that layer, the cost $C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij}$ yields the number of objects after the permuted objects have all been placed in the layers. In other words, D-BPSP$_2$ is the problem of finding a permutation of objects such that the maximal number of objects of the same color on any layer is less than or equal to $B$ for all colors. Thus, for practical applications, the total cost term of D-BPSP$_2$ can be interpreted as a worst-case estimation of the performance and the total cost of D-BPSP$_3$, analogously, represents the average performance over all colors.

#### 2.2.3.1 An Optimization Version of BPSP$_2$

Since the objective is to minimize the maximal cost (2.2.11), we now formulate the decision problem as an optimization problem:

$$\min \max_{k=1,\ldots,N^K,\ l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl} \delta_{ij} \right) \quad .$$

There appears to be no easy way to solve the above problem efficiently. Therefore we transform it to an integer linear programming formulation. The *minimax* objective function is replaced by an equivalent linear formulation. For this aim new variables are introduced: $u_k$ – the maximal number of objects of color $k$ on any layer, *i.e.*,

$$u_k = \max_{l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \right) \tag{2.2.13}$$

and

$$y = \max_{k=1,\ldots,N^K} \{u_k\} \quad .$$

That allows us to define the transformed objective function

$$\min y \quad , \tag{2.2.14}$$

subject to additional constraints

$$u_k \leq y, \quad 1 \leq k \leq N^K \quad , \tag{2.2.15}$$

$$C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \leq u_k, \quad \begin{matrix} 1 \leq k \leq N^K \\ 1 \leq l \leq N^L \end{matrix} \quad , \tag{2.2.16}$$

$$\sum_{j=1}^{N^O} \delta_{ij} = 1, \quad 1 \leq i \leq N^O \quad , \tag{2.2.17}$$

$$\sum_{i=1}^{N^O} \delta_{ij} = 1, \quad 1 \leq j \leq N^O \quad , \tag{2.2.18}$$

$$\delta_{ij} = 0, \quad \forall i,j : j < i - N^S \quad , \tag{2.2.19}$$

$$\delta_{ij} \in \{0,1\}, \quad 1 \leq i,j \leq N^O \quad . \tag{2.2.20}$$

Let us make some remarks related to the above constraints:

Note that (2.2.14) and (2.2.15) imply the identity $y = \max_{k=1,\ldots,N^K} \{u_k\}$. The inequalities (2.2.16) express that the number of objects of color $k$ already on layer $l$ plus a number of objects of color $k$ assigned to this layer cannot be greater than the maximal number $u_k$ of objects of color $k$ on any layer. The assignment constraints (2.2.17)-(2.2.18) determine the permutations of the objects, *i.e.*, each object can take only one position in a new ordering and each new position can be filled only with one object. Depending on $N^S$, certain permutations can be excluded a priori by (2.2.19).

In some situations, when the difference between the number of objects of different orders is very large, it may not be advisable to minimize just the maximum number of objects of this set of orders found on a single layer. Instead, it is more efficient to minimize the total amount of output cycles. We treat this approach in the optimization problem $\mathrm{BPSP}_3$ introduced below.

#### 2.2.3.2 An Optimization Version of $\mathrm{BPSP}_3$

The objective function corresponding to (2.2.12) is:

$$\min \sum_{k=1}^{N^K} \max_{l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \right) \quad .$$

Using the new variables $u_k$ (2.2.13), we get:

$$\min \sum_{k=1}^{N^K} u_k \tag{2.2.21}$$

subject to (2.2.16)-(2.2.20).

Thus, it can be seen that $\mathrm{BPSP}_2$ and $\mathrm{BPSP}_3$ contain $\mathrm{BPSP}_1$ as a kernel. Unfortunately, the polyhedrons of $\mathrm{BPSP}_2$ and $\mathrm{BPSP}_3$ are not integral and, hence, the complexity issues of those problems are very important. The following section addresses the complexity of $\mathrm{BPSP}_2$.

EXAMPLE 2.5 *In this example we illustrate in detail the optimization models BPSP$_1$, BPSP$_2$, and BPSP$_3$ with possible optimal solutions using the following set of input data:*

*The number of objects, $N^O$, is 6, the number of colors, $N^K$, is 3. These objects are grouped together in the sets: $\mathcal{S}_1 = \{1\}$, $\mathcal{S}_2 = \{2,3,4\}$, $\mathcal{S}_3 = \{5,6\}$. The number of layers, $N^L$, is 3, and the capacity of the pre-sorting facility is $N^S = 2$. Coefficients reflecting the content of the layers have the following values: $C_{11} = 1$, $C_{21} = 1$, $C_{22} = 2$, $C_{33} = 1$, all other $C_{kl} = 0$. Fig. 2.2.5 shows the input sequence and the content of the layers before the assignment.*

*BPSP$_1$: For this model the construction of coefficients $C_{kl}$ is simplified, so we do not need to calculate the number of objects of color $k$ on layer $l$, but only need to indicate whether layer $l$ has an object of color $k$ (see 2.2.2). Regarding this definition, only the coefficients $C'_{11}, C'_{21}, C'_{22}, C'_{31}, C'_{32}, C'_{41}, C'_{42}, C'_{53}$, and $C'_{63}$ have value 1.*

$$f_1 = \min(\delta_{11} + \delta_{21} + \delta_{22} + \delta_{31} + \delta_{32} + \delta_{41} + \delta_{42} + \delta_{53} + \delta_{63}) \tag{2.2.22}$$


*Figure 2.2.5.* The input sequence and the content of the layers before the assignment

*subject to*

$$\sum_{j=1}^{6} \delta_{ij} = 1, \quad 1 \le i \le 6 \quad , \tag{2.2.22}$$

$$\sum_{i=1}^{6} \delta_{ij} = 1, \quad 1 \le j \le 6 \quad ,$$

$$\delta_{41}, \delta_{51}, \delta_{52}, \delta_{61}, \delta_{62}, \delta_{63} = 0 \quad , \tag{2.2.24}$$

$$\delta_{ij} \in \{0, 1\}, \quad 1 \le i, j \le 6 \quad . \tag{2.2.25}$$

*Now consider $BPSP_2$:*

$$f_2 = \min y \tag{2.2.26}$$

*subject to*

$$u_1, u_2, u_3 \le y \quad , \tag{2.2.27}$$

$$\begin{aligned} C_{11} + (\delta_{11} + \delta_{14}) &\le u_1 \quad , \\ C_{12} + (\delta_{12} + \delta_{15}) &\le u_1 \quad , \\ C_{13} + (\delta_{13} + \delta_{16}) &\le u_1 \quad , \end{aligned} \tag{2.2.27}$$

$$\begin{aligned} C_{21} + (\delta_{21} + \delta_{24} + \delta_{31} + \delta_{34} + \delta_{41} + \delta_{44}) &\le u_2 \quad , \\ C_{22} + (\delta_{22} + \delta_{25} - \delta_{32} + \delta_{35} + \delta_{42} + \delta_{45}) &\le u_2 \quad , \\ C_{23} + (\delta_{23} + \delta_{26} + \delta_{33} + \delta_{36} + \delta_{43} + \delta_{46}) &\le u_2 \quad , \end{aligned} \tag{2.2.28}$$

$$\begin{aligned} C_{31} + (\delta_{51} + \delta_{54} + \delta_{61} + \delta_{64}) &\le u_3 \quad , \\ C_{32} + (\delta_{52} - \delta_{55} + \delta_{62} + \delta_{65}) &\le u_3 \quad , \\ C_{33} + (\delta_{53} + \delta_{56} + \delta_{63} + \delta_{66}) &\le u_3 \quad , \end{aligned} \tag{2.2.29}$$

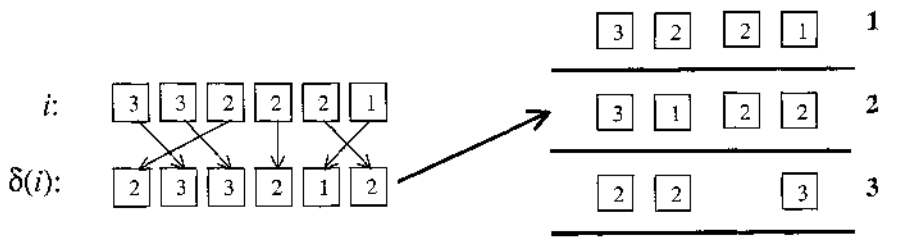


*Figure 2.2.6.* The input sequence and the content of the layers after the assignment

*and (2.2.23)-(2.2.25).*
*BPSP*$_3$*:*

$$f_3 = \min(u_1 + u_2 - u_3)$$

*subject to (2.2.23)-(2.2.25) and (2.2.28)-(2.2.30).*
*For all these models, the permutation* $\delta = (2, 1, 3, 6, 4, 5)$ *is optimal. The objective functions* $f_1$*,* $f_2$ *and* $f_3$ *have the values* 1*,* 2 *and* 4*, respectively. Fig. 2.2.6 illustrates this example.*

## 2.3. Complexity Results

THEOREM 2.6 *Problem D-BPSP*$_2$ *is NP-complete.*

**Proof.** It is easy to see that BPSP$_2 \in$ NP, since a nondeterministic algorithm needs only to guess a permutation of the variables and to check in polynomial time whether that permutation satisfies all the given constraints. We proceed by showing that the *3-SAT (3-Satisfiability)* problem can be polynomially reduced to BPSP$_2$. Concerning the complexity issues of the 3-SAT problem we refer the reader to [32]. Below we give the definition of the Satisfiability problem [64].

DEFINITION 2.7 *Let* $X = \{x_1, x_2, ..., x_n\}$ *be a set of* $n$ *Boolean variables. A literal* $y_i$ *is either a variable* $x_i$ *or its negation* $\bar{x}_i$*. A clause* $F_j$ *is a disjunction of literals. Let formula* $F = F_1 \wedge F_2 \wedge ... \wedge F_m$ *be a conjunction of* $m$ *clauses. The formula* $F$ *is satisfiable if and only if there is a truth assignment* $t : X \to \{0, 1\}$*, which simultaneously satisfies all clauses* $F_j$ *in* $F$*. The Satisfiability problem is the problem to decide for a given instance* $(X, F)$ *whether there is a truth assignment for* $X$ *that satisfies* $F$*. The 3-SAT problem is a restriction of the Satisfiability problem where each clause contains exactly 3 literals. The 3-SAT problem is still NP-complete.*

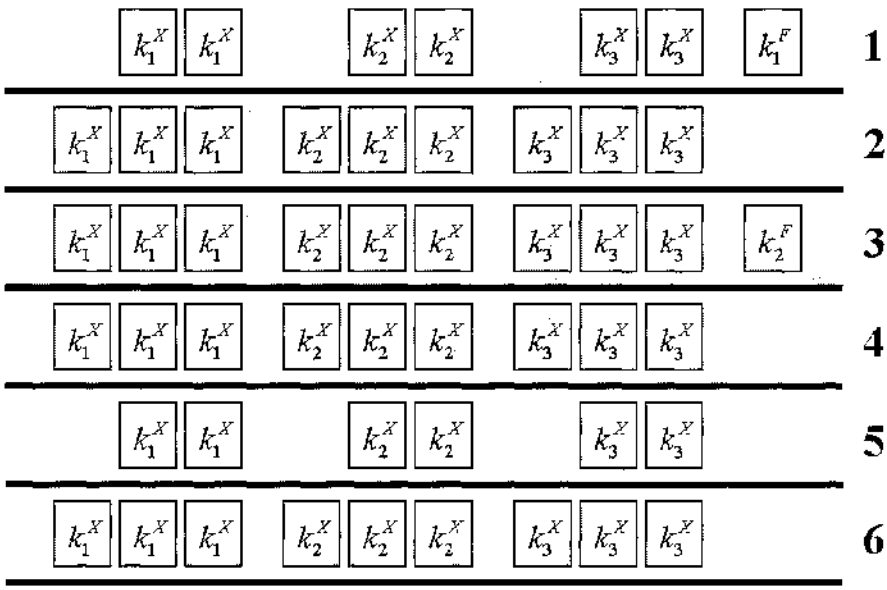


*Figure 2.3.7.* The content of the storage before distributing the objects from the sequence *seq*

For an arbitrary instance $(X, F)$ of the *3-SAT* problem we define an instance of the sequencing problem $\text{BPSP}_2$, such that there exists a feasible permutation of the objects $\delta$ with

$$\max_{k=1,\ldots,N^K,\ l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \right) \leq B$$

if and only if there is a truth assignment $t : X \to \{0, 1\}$ satisfying $F$.

We choose $B = 3$ and $N^S = 1$ (*i.e.*, an object can move forward at most by one position). The number of layers is

$$N^L := 2(m+1) \quad , \tag{2.3.1}$$

the number of objects is

$$N^O := 2nN^L \quad , \tag{2.3.2}$$

and the number of sets $\mathcal{S}_k$ is

$$N^K := 2n - m + 2mn \quad . \tag{2.3.3}$$

Specifically, we have

- $2n$ colors $k_i^X$, $k_i^L$, one for each variable $x_i$,
- $m$ colors $k_j^F$, one for each clause $F_j$,
- the first group of $mn$ auxiliary colors $k_{ij}^{XF}$, and
- the second group of $mn$ auxiliary colors $k_{ij}^{\bar{X}F}$.

The sequence $seq$ consists of $2n$ subsequent parts $seq_i$ of $N^L$ objects each, *i.e.*, $seq := (seq_1, seq_2, ..., seq_n, seq_{n+1}, seq_{n+2}, ..., seq_{2n})$. Each $seq_i$ has $2(m+1)$ objects, *i.e.*, $seq_i := (q_{i,1}, q_{i,2}, ..., q_{i,2m-2})$. Now we define the sets $\mathcal{S}_k$, *i.e.*, the color of each object contained in the sequence $seq_i$ :

the first and last objects of $seq_{2i-1}$ and $seq_{2i}$ always have the colors of the variable $x_i$, *i.e.*,

$$q_{2i-1,1}, q_{2i,1} \in \mathcal{S}_{k_i^X} \tag{2.3.4}$$

$$q_{2i-1,N^L}, q_{2i,N^L} \in \mathcal{S}_{k_i^L}. \tag{2.3.5}$$

The colors of the $2m$ objects in between are defined depending on the occurrence of variable $x_i$ in the clauses $F_j$ as follows:

$$x_i \in F_j \Longrightarrow \begin{array}{r} q_{2i,2j} \in \mathcal{S}_{k_j^F} \\ q_{2i,2j+1} \in \mathcal{S}_{k_{ij}^{XF}} \\ q_{2i-1,2j}, q_{2i-1,2j+1} \in \mathcal{S}_{k_{ij}^{XF}} \end{array} \tag{2.3.6}$$

$$\bar{x}_i \in F_j \Longrightarrow \begin{array}{r} q_{2i-1,2j} \in \mathcal{S}_{k_j^F} \\ q_{2i-1,2j+1} \in \mathcal{S}_{k_{ij}^{XF}} \\ q_{2i,2j}, q_{2i,2j+1} \in \mathcal{S}_{k_{ij}^{\bar{X}F}} \end{array} \tag{2.3.7}$$

$$x_i,\ \bar{x}_i \notin F_j \Longrightarrow \begin{array}{r} q_{2i-1,2j}, q_{2i-1,2j+1} \in \mathcal{S}_{k_{ij}^{XF}} \\ q_{2i,2j}, q_{2i,2j+1} \in \mathcal{S}_{k_{ij}^{\bar{X}F}} \end{array} \tag{2.3.8}$$

The values of the cost function $C_{kl}$ (which reflects the content of the storage) are defined for any $\{k, l\}$ as:

$$C_{k_i^X,l} = \begin{cases} 2, \text{ if } l \in \{1, N^L - 1\} \\ 3, \text{ else} \end{cases}, \ \forall i \in \{1, ..., n\} \tag{2.3.9}$$

$$C_{k_j^F,l} = \begin{cases} 1, \text{ if } l = 2j - 1 \\ 0, \text{ else} \end{cases}, \ \forall j \in \{1, ..., m\} \tag{2.3.10}$$

$$C_{k_{ij}^{XF},l}, C_{k_{ij}^{\bar{X}F},l}, C_{k_i^L,l} = 0, \forall i, j, l. \tag{2.3.11}$$

Fig. 2.3.7 illustrates these coefficients for an example with $n = 3$ and $m = 2$. In the remaining part of the proof we show that a feasible permutation of the objects in *seq* with

$$\max_{k=1,\ldots,N^K,\ l=1,\ldots,N^L} \left( C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \right) \leq 3$$

exists if and only if $F$ is satisfiable, *i.e.*, there is a truth assignment $t : X \to \{0, 1\}$ that satisfies each clause in $F$.

First, we assume that $F$ is satisfiable. We show that there exists a feasible permutation of objects with no more than 3 objects of the same color on any layer. Recall that:

- for each color $k_i^X$ there are already exactly three objects on each of the layers $2, ..., N^L-2, N^L$, two objects on each of the layers $1, N^L-1$, one object in $seq_{2i-1}$ and one object in $seq_{2i}$;
- for each color $k_j^F$ there is one object on layer $2j - 1$ and there are exactly three objects in *seq*;
- for each color $k_{ij}^{XF}$, $k_{ij}^{\bar{X}F}$ there exist at most two objects in *seq*;
- for each color $k_i^L$ only two objects exist in $seq_i$.

Now consider the first object of subsequence $seq_{2i-1}$. This object has color $k_i^X$. It has to be sent to layer $N^L - 1$ or has to remain on the first layer, because all other layers already contain three objects of this color. The layers 1 and $N^L - 1$ already contain two objects of color $k_i^X$ and can accommodate only one additional object each. Therefore, the second object of color $k_i^X$ – the first object of $seq_{2i}$ – has to be sent to the layer not used by the first object. This will be done by moving the objects to the nearest layer (*i.e.*, move of minimal distance), such that for any $j \in seq_i$ the following holds: $\delta(j) \in seq_i$. That means we can discuss each subsequence independently. We will use this alternation to map the truth assignments into the permutations set $\delta$:

$$t(x_i) = 1 \Longrightarrow \begin{cases} \delta(q_{2i-1,1}) = q_{2i-1,N^L-1} \\ \delta(q_{2i-1,j}) = q_{2i-1,j-1} \quad , \quad 1 < j < N^L \\ \delta(q_{2i-1,N^L}) = q_{2i-1,N^L} \\ \delta(q_{2i,j}) = q_{2i,j} \quad , \quad 1 \leq j \leq N^L \end{cases} \tag{2.3.12}$$

$$t(x_i) = 0 \Longrightarrow \begin{cases} \delta(q_{2i,1}) = q_{2i,N^L-1} \\ \delta(q_{2i,j}) = q_{2i,j-1} \quad , \quad 1 < j < N^L \\ \delta(q_{2i,N^L}) = q_{2i,N^L} \\ \delta(q_{2i-1,j}) = q_{2i-1,j} \quad , \quad 1 \leq j \leq N^L \end{cases} \tag{2.3.13}$$

In this way, if $t$ assigns 1 to $x_i$, then from (2.3.12) the first object of $seq_{2i-1}$ moves to layer $N^L - 1$, and all other objects in $seq_{2i-1}$ move one layer up (except for the last object which remains on the last layer). All objects of $seq_{2i}$ keep their positions. Otherwise, if $t$ assigns 0 to $x_i$, then it follows from (2.3.13) that all objects from $seq_{2i-1}$ keep their positions and the first object of $seq_{2i}$ moves to the layer $N^L - 1$. All other objects in $seq_{2i}$ move one layer up (except for the last object which remains on the last layer).

Assume there is a layer $l$ with four or more objects of the same color. Let us first discuss the color. It cannot be one of $k_{ij}^{XF}, k_{ij}^{\bar{X}F}, k_i^L$ since there are at most two of those and $C_{kl}$ is zero for them. Without loss of generality we assume that there are four objects of color $k_j^F$ on layer $l$. Since $|\mathcal{S}_{k_l^F}| = 3$, $C_{k_j^F,l}$ must be 1 and because of (2.3.10) it follows that $l = 2j - 1$. Now consider one of the three objects. From (2.3.6) and (2.3.7) it is known that $\mathcal{S}_{k_l^F} \subset \{q_{2i,2j}, q_{2i-1,2j}\}$. We distinguish two cases:

**Case 1:**

$$q_{2i,2j} \in \mathcal{S}_{k_l^F} \overset{(2.3.6)}{\Longrightarrow} x_i \in F_j \wedge \delta(q_{2i,2j}) = q_{2i,2j-1} \tag{2.3.13}$$

$$\text{since } l = 2j-1 \overset{(2.3.13)}{\Longrightarrow} t(x_i) = 0 \tag{2.3.14}$$

**Case 2:**

$$q_{2i-1,2j} \in \mathcal{S}_{k_l^F} \overset{(2.3.6)}{\Longrightarrow} \bar{x}_i \in F_j \wedge \delta(q_{2i-1,2j}) = q_{2i-1,2j-1} \tag{2.3.15}$$

$$\text{since } l = 2j-1 \overset{(2.3.12)}{\Longrightarrow} t(\bar{x}_i) = 0 \tag{2.3.16}$$

Therefore, in both cases the truth value of the literal of $x_i$ in $F_j$ is false, so $F_j$ contains one false literal. The same reasoning holds for the other two objects. Thus, $F_j$ contains three false literals and is therefore false itself. This is a contradiction to the assumption that $t$ satisfies $(X, F)$.

For the missing direction of the proof, we assume that $(X, F)$ is unsatisfiable. Our goal is to prove that for any permutation $\delta$ there exists at least one color for which four objects are located on the same layer. Barring trivial cases, that will be color $k_j^F$.

Consider the permutations where $\delta(q_{2i,1}) \equiv l \bmod N^L$ with $2 \leq l < N^L - 1$ or $l = N^L$, *i.e.*, when the first object of subsequence $2i$ is not moved to the first or last-but-one layer. Because of (2.3.9) $C_{k_i^X,l} = 3$, layer $l$ contains four objects of color $k_i^X$. Therefore we only need to consider the remaining cases, *i.e.*, when the first object moved to the first or to the next to last layer.

With the same reasoning we can assume that $\delta(q_{2i-1,1}) \equiv \pm 1 \bmod N^L$. So we only deal with cases where

$$\delta(q_{2i,1}) \equiv 1 \bmod N^L \text{ and } \delta(q_{2i-1,1}) \equiv -1 \bmod N^L \tag{2.3.18}$$

or

$$\delta(q_{2i,1}) \equiv -1 \bmod N^L \text{ and } \delta(q_{2i-1,1}) \equiv 1 \bmod N^L \quad . \tag{2.3.19}$$

Let us define a truth assignment $t_\delta$ by

$$t(x_i) = \begin{cases} 1 & \text{if } \delta(q_{2i,1}) \equiv 1 \bmod N^L \\ 0 & \text{if } \delta(q_{2i,1}) \equiv -1 \bmod N^L \end{cases} \quad . \tag{2.3.20}$$

Furthermore, (2.2.1) tells us that $\delta(j) \geq j-1$ as $N^S = 1$ by construction. Since $(X, F)$ is unsatisfiable by assumption, there must be some clause $F_j$ which is not satisfied by $t_\delta$.

**Case 1**: $F_j$ contains a non-negated literal $x_i$.

Then $t_\delta(x_i) = 0$ and from (2.3.20) we know, that $\delta(q_{2i,1}) \equiv -1 \bmod N^L$ and from (2.3.19) that $\delta(q_{2i-1,1}) \equiv 1 \bmod N^L$. Because $\delta(q_{2i,1}) \geq q_{2i,1} - 1 \equiv 0 \bmod N^L$, so $\delta(q_{2i,1}) \geq q_{2i,1} + N^L - 2$ since $\delta(k) > k - 2$, *i.e.*, the tray $q_{2i,1}$ stays in the additional storage at least until $N^L - 2$ objects passed. Therefore, $\delta(q_{2i,k}) = q_{2i,k} - 1$ for $2 \leq k \leq N^L - 1$, especially for $k = 2j$. From (2.3.6) we know that $q_{2i,2j} \in \mathcal{S}_{k_j^F}$, hence there is one additional object of color $k_j^F$ in layer $2j-1$.

**Case 2**: $F_j$ contains a negated literal $\bar{x}_i$.

Then $t_\delta(x_i) = 1$ and from (2.3.20) we know that $\delta(q_{2i,1}) \equiv 1 \bmod N^L$ and from (2.3.18) that $\delta(q_{2i-1,1}) \equiv -1 \bmod N^L$. As in the first case, we conclude that $\delta(q_{2i-1,1}) \geq q_{2i-1,1} + N^L - 2$ and, therefore, it follows that $\delta(q_{2i-1,2j}) = q_{2i-1,2j} - 1 \equiv 2j - 1 \bmod N^L$. Since (2.3.7) $q_{2i-1,2j} \in \mathcal{S}_{k_j^F}$ there is an additional object of color $k_j^F$ on layer $2j-1$.

Hence, for each of the three literals of $F_j$ there is one object of color $k_j^F$ on layer $2j-1$ and $C_{2j-1,k_j^F} = 1$ according to (2.3.10). Therefore, layer $2j-1$ contains four objects of color $k_j^F$ and the proof is complete.

To see that this transformation can be performed in polynomial time, it suffices to observe that the number of layers, the number of colors and the number of objects in *seq* are bounded by a polynomial in $2(m+1)$, $4n(m+1)$ and $2n+m+2mn$, respectively. Hence the size of the $\text{BPSP}_2$ instance is bounded above by a polynomial function of the size of the 3-SAT instance (q.e.d). ■

We illustrate Theorem 2.6 by the following example.

EXAMPLE 2.8 *Suppose there is the following instance of the 3-SAT problem:* $F = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$, *i.e.*, $n = 3$ *and* $m = 2$. *Now define*

*all data needed for constructing the sequence seq. By (2.3.1)-(2.3.3) we obtain* $N^L = 6$, $N^O = 36$, *and the number of sets* $\mathcal{S}_k$ *is* $N^K = 20$, *i.e.,*

- $k_i^X$, $k_i^L$, $i = 1, 2, 3$;
- $k_j^F$, $j = 1, 2$;
- $k_{ij}^{XF}$, $k_{ij}^{\bar{X}F}$, $i = 1, 2, 3$, $j = 1, 2$.

*Fig. 2.3.7 shows the content of the storage system w.r.t. formulas (2.3.9) and (2.3.10) before the objects from seq are distributed. Fig. 2.3.8 shows the sequence* $seq = (seq_1, seq_2, ..., seq_6)$. *In addition, this picture illustrates the assignments, e.g., for* $x_1$, *described by formulas (2.3.12) and (2.3.13) and the subsequent assignment to the layers. Similar transformations apply to the sequences* $seq_3$, $seq_4$, $seq_5$, $seq_6$ *(the first two correspond to* $x_2$, *the others to* $x_3$*).*

## 2.4. Polynomial Subcases

The decision problem D-BPSP$_2$ is shown to be NP-complete (see Section 2.3). What makes this problem difficult? In this section we consider the problem with some additional assumptions. The first assumption is that the numbers of elements in sets $\mathcal{S}_k$, *i.e.*, values $|\mathcal{S}_k|$, are known in advance. The second one is that the capacity of the pre-sorting facility is large enough, *i.e.*, given $N^S \geq N^L - 1$, any permutation of objects can be realized. This assumption is only needed for an alternative model formulation provided in Subsection 2.4.2.

### 2.4.1 Reformulation of BPSP$_2$ and BPSP$_3$

In Section 2.2.3.2 we suggested model formulations with continuous variables $u_k$. We can decrease the number of variables if the values of $|\mathcal{S}_k|$, the number of objects of color $k$ in the input sequence, are known in advance. Clearly, the optimal permutation of the objects is one that allows a uniform distribution of objects over all layers. Of course, we need to add to the value $|\mathcal{S}_k|$ all objects of color $k$ that are already in the layers $\sum_{l=1}^{N^L} C_{kl}$. It can happen that $\max\limits_{l=1,...,N^L} C_{kl} > \left\lceil \frac{|\mathcal{S}_k| - \sum_{l=1}^{N^L} C_{kl}}{N^L} \right\rceil$. Therefore, we have to calculate values $\mathcal{S}_k^* = \max\left( \left\lceil \frac{\mathcal{S}_k| + \sum_{l-1}^{N^L} C_{kl}}{N^L} \right\rceil, \max\limits_{l=1,...,N^L} C_{kl} \right)$. Thus, the corresponding objective function value for BPSP$_3$ is $\sum_{k=1}^{N^K} u_k = \sum_{k=1}^{N^K} \mathcal{S}_k^*$ and the equality $\max\limits_{k=1,...,N^K} u_k = \max\limits_{k=1,...,N^K} \mathcal{S}_k^*$ holds for BPSP$_2$. Then we can modify the optimization models from Section 2.2.3.2 and obtain the transformed model:

D-BPSP$^{/}$:

$$\sum_{j=1}^{N^O} \delta_{ij} = 1, \quad 1 \le i \le N^O \quad , \tag{2.4.1}$$

$$\sum_{i=1}^{N^O} \delta_{ij} = 1, \quad 1 \le j \le N^O \quad , \tag{2.4.2}$$

$$C_{kl} + \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \le \mathcal{S}_k^*, \quad \begin{matrix} 1 \le k \le N^K \\ 1 \le l \le N^L \end{matrix} \quad , \tag{2.4.3}$$

$$\delta_{ij} = 0, \quad \forall i,j : j < i - N^S \quad , \tag{2.4.4}$$

$$\delta_{ij} \in \{0,1\}, \quad 1 \le i,j \le N^O \quad . \tag{2.4.5}$$

This is not yet an optimization problem since only a feasible point of (2.4.1)-(2.4.5) need to be found. If the capacity of the pre-sorting facility is not large enough, the feasible set can be empty.

EXAMPLE 2.9 *Here we demonstrate the calculation of $\mathcal{S}_k^*$ values on the data from Example 2.5:*

$$\mathcal{S}_1^* = \max\left(\left\lceil \frac{S_1|+C_{11}+C_{12}+C_{13}}{N^L} \right\rceil, \max_{j=1,2,3} C_{1j}\right) = \max(\lceil \tfrac{1+1+0-0}{3} \rceil, 1) = 1;$$

$$\mathcal{S}_2^* = \max\left(\left\lceil \frac{|S_2|+C_{21}+C_{22}+C_{23}}{N^L} \right\rceil, \max_{j=1,2,3} C_{2j}\right) = \max(\lceil \tfrac{3+1+2+0}{3} \rceil, 2) = 2;$$

$$\mathcal{S}_3^* = \max\left(\left\lceil \frac{|S_3|+C_{31}+C_{32}+C_{33}}{N^L} \right\rceil, \max_{j=1,2,3} C_{3j}\right) = \max(\lceil \tfrac{2+0-0+1}{3} \rceil, 1) = 1.$$

### 2.4.2 An Alternative Model Formulation of D-BPSP$^{/}$

In the following we show that D-BPSP$^{/}$ without (2.4.4) is polynomially solvable. At first, we present another mathematical model formulation named $D - \overline{BPSP^{/}}$ with the following integer variables:

$$x_{kl} \text{ is the number of objects of color } k \text{ on layer } l \quad .$$

Notice that these variables can be derived easily from the $\delta_{ij}$ variables:

$$x_{kl} = \sum_{i=1}^{N^O} \sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \quad . \tag{2.4.6}$$

Suppose $N^O = pN^L$, where $p \in \mathbb{Z}$. Then the decision problem D-$\overline{\text{BPSP}^/}$ concerns whether an assignment exists for $x_{kl}$ satisfying the constraints:

$$\sum_{l=1}^{N^L} x_{kl} = |\mathcal{S}_k|, \quad 1 \le k \le N^K \quad , \tag{2.4.7}$$

$$\sum_{k=1}^{N^K} x_{kl} = \frac{N^O}{N^L}, \quad 1 \le l \le N^L \quad , \tag{2.4.8}$$

$$0 \le x_{kl} \le \mathcal{S}_k^* - C_{kl}, \quad 1 \le k \le N^K, \quad 1 \le l \le N^L \quad . \tag{2.4.9}$$

After that we show the correspondence between D-BPSP$^/$ and D-$\overline{\text{BPSP}^/}$ by demonstration that a solution of the first problem implies a solution of the second one and vice versa.

THEOREM 2.10 *Problems D-BPSP$^/$ (without 2.4.4) and D-$\overline{\text{BPSP}^/}$ are equivalent.*

***Proof.*** *We show that if there is a solution of D-BPSP$^/$ $\delta_{ij} \in \{0,1\}$, $1 \le i,j \le N^O$ that satisfies (2.4.1)-(2.4.3), then the solution $x_{kl}$ of D-$\overline{\text{BPSP}^/}$ satisfies (2.4.7)-(2.4.9), and vice versa.*

*1:* $\sum_{l=1}^{N^L} x_{kl} \overset{(2.4.6)}{=} \sum_{l=1}^{N^L}\sum_{i=1}^{N^O}\sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \overset{(2.2.10)}{=} \sum_{l=1}^{N^L}\sum_{i=1}^{N^O}\sum_{j=1}^{N^O} S_{ik}M_{jl}\delta_{ij}$

$= \sum_{i=1}^{N^O} S_{ik} \sum_{j=1}^{N^O} \delta_{ij} \sum_{l=1}^{N^L} M_{jl} \overset{(2.2.9)}{=} \sum_{i=1}^{N^O} S_{ik} \underbrace{\sum_{j=1}^{N^O} \delta_{ij}}_{=1 \text{ by } (2.4.1)}$

$= \sum_{i=1}^{N^O} S_{ik} \overset{(2.2.8)}{=} |\mathcal{S}_k|, \quad 1 \le k \le N^K \Longleftrightarrow \sum_{j=1}^{N^O} \delta_{ij} = 1, \quad 1 \le i \le N^O;$

*2:* $\sum_{k=1}^{N^K} x_{kl} \overset{(2.4.6)}{=} \sum_{k=1}^{N^K}\sum_{i=1}^{N^O}\sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \overset{(2.2.10)}{=} \sum_{k=1}^{N^K}\sum_{i=1}^{N^O}\sum_{j=1}^{N^O} S_{ik}M_{jl}\delta_{ij}$

$= \sum_{j=1}^{N^O} M_{jl} \sum_{i=1}^{N^O} \delta_{ij} \sum_{k=1}^{N^K} S_{ik} \overset{(2.2.8)}{=} \sum_{j=1}^{N^O} M_{jl} \underbrace{\sum_{i=1}^{N^O} \delta_{ij}}_{=1 \text{ by } (2.4.2)}$

$= \sum_{j=1}^{N^O} M_{jl} \overset{(2.2.9)}{=} \frac{N^O}{N^L}, \quad 1 \le l \le N^L \Longleftrightarrow \sum_{i=1}^{N^O} \delta_{ij} = 1, \quad 1 \le j \le N^O;$

*3: The constraints (2.4.3) and (2.4.9) are the same. Indeed,*

$$x_{kl} \le \mathcal{S}_k^* - C_{kl} \Longleftrightarrow x_{kl} + C_{kl} \le \mathcal{S}_k^* \overset{(2.4.6)}{\Longleftrightarrow} C_{kl} + \sum_{i=1}^{N^O}\sum_{j=1}^{N^O} D_{ikjl}\delta_{ij} \le \mathcal{S}_k^*,$$

$\forall k = \{1, ..., N^K\}, \quad \forall l = \{1, ..., N^L\}$ *(q.e.d).* ■

EXAMPLE 2.11 *Here we illustrate how to convert a solution of D-$\overline{BPSP^{/}}$ to a solution of D-BPSP$^{/}$. Consider the input sequence and sets*

$$\mathcal{S}_k = \{i_{c_k} | c_k = 1, \ldots, |\mathcal{S}_k|\}$$

*from Example 2.5. Suppose a solution of D-$\overline{BPSP^{/}}$ to be:*

$$\begin{array}{lll} x_{11} = 0, & x_{12} = 1, & x_{13} = 0, \\ x_{21} = 1, & x_{22} = 0, & x_{23} = 2, \\ x_{31} = 0, & x_{32} = 1, & x_{33} = 1. \end{array}$$

*We construct the solution $\delta_{ij}$ of D-BPSP$^{/}$ as follows:*
*For all $k = 1, ..., N^K, \quad l = 1, ..., N^L$ set initially $c_k := 1$ and $i_l := 0$. Then repeat the following steps while $x_{kl} \geq 1$:*

- *let $i^* := i_{c_k}, \quad c_k := c_k + 1$;*
- *let $j^* := l + i_l N^L, \quad i_l := i_l + 1$;*
- *let $\delta_{i^* j^*} := 1$ and $x_{kl} := x_{kl} - 1$.*

*Using this procedure for $N^K = 3$ and $N^L = 3$ we obtain the solution: $\delta = (2, 1, 3, 6, 5, 4)$. Notice that this permutation yields the same assignment to the layers as in Example 2.5. But it is not feasible there, because the capacity of the pre-sorting facility is not large enough.*

Let us now show that the problem D-$\overline{\text{BPSP}^{/}}$ is polynomially solvable. We use the following results from [61].

THEOREM 2.12 *([61], Part III.1 "Totally unimodular matrices") Let $A$ be a $(0, 1, -1)$ matrix with no more than two nonzero elements in each column. Then $A$ is totally unimodular iff the rows of $A$ can be partitioned into two subsets $A^1$ and $A^2$ such that if a column contains two nonzero elements, the following statements are true:*

1 *If both nonzero elements have the same sign, then one is in a row contained in $A^1$ and the other is in a row contained in $A^2$.*

2 *If the two nonzero elements have opposite sign, then both are in rows contained in the same subset.*

THEOREM 2.13 *([61], Part III.1 "Totally unimodular matrices") If $A$ is totally unimodular, if $b, b', d$ and $d'$ are integral, and if $P(b, b', d, d') =$*

$\{x \in R^n : b' \le Ax \le b, \quad d' \le x \le d\}$ *is not empty, then* $P(b, b', d, d')$ *is an integral polyhedron.*

THEOREM 2.14 *The problem* $\overline{D\text{-}BPSP'}$ *is polynomially solvable.*

***Proof.*** *We re-write (2.4.7)-(2.4.9) as:*

$$Ax = S^1_{N^K+N^L} \quad , \tag{2.4.10}$$

$$0 \le x \le S^2_{N^K N^L} \quad , \tag{2.4.11}$$

*where*

$$A = \begin{pmatrix} A^1 \\ A^2 \end{pmatrix} \quad , \tag{2.4.12}$$

$$A^1 = \begin{pmatrix} 1 & 1 & \dots & 1 & 0 & 0 & & 0 & & 0 & 0 & & 0 \\ 0 & 0 & & 0 & 1 & 1 & \dots & 1 & & 0 & 0 & & 0 \\ 0 & 0 & & 0 & 0 & 0 & & 0 & \dots & 0 & 0 & & 0 \\ 0 & 0 & & 0 & 0 & 0 & & 0 & & 1 & 1 & \dots & 1 \end{pmatrix} \quad ,$$

$$A^2 = \begin{pmatrix} 1 & 0 & & 0 & 1 & 0 & & 0 & \dots & 1 & 0 & & 0 \\ 0 & 1 & & 0 & 0 & 1 & & 0 & \dots & 0 & 1 & & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & & 1 & 0 & 0 & & 1 & \dots & 0 & 0 & & 1 \end{pmatrix} \quad ,$$

*and*

$$S^1_{N^K+N^L} = (|\mathcal{S}_1|, |\mathcal{S}_2|, ..., |\mathcal{S}_k|, \frac{N^O}{N^L}, \frac{N^O}{N^L}, ..., \frac{N^O}{N^L})^T,$$

$$S^2_{N^K N^L} = (\mathcal{S}^*_1 - C_{11}, ..., \mathcal{S}^*_1 - C_{1,N^L}, ..., \mathcal{S}^*_{N^K} - C_{N^K,1}, ..., \mathcal{S}^*_{N^K} - C_{N^K,N^L})^T,$$

*and*

$$x = (x_{11}, x_{12}, ..., x_{1,N^L}, x_{21}, x_{22}, ..., x_{2,N^L}, ..., x_{N^K,1}, x_{N^K,2}, ..., x_{N^K,N^L})^T.$$

*At first, notice that the matrix A is totally unimodular:*

*By Theorem 2.12, the matrix A is partitioned into two subsets (2.4.12), and each subset contains exactly one nonzero element of the same sign in each column.*

*Secondly, the polyhedron of (2.4.10)-(2.4.11) is integral by Theorem 2.13:* $b' = b = S^1_{N^K+N^L}$, $d' = 0$, $d = S^2_{N^K N^L}$ *are integral, and obviously, the system of linear inequalities (2.4.10)-(2.4.11) always has a solution. And again from [61] it is well known that such problems are polynomially solvable (q.e.d).* ■

## 2.5. The Case of Two Layers

In this section we analyze the BPSP with $N^L = 2$. We show that in this case the BPSP is polynomially solvable, and construct corresponding polynomial algorithms.

### 2.5.1 Offline Situations

In offline situations we know the arrival sequence of the $N^O$ objects in advance. Consequently, the number $|\mathcal{S}_k|$ of objects of type $k$ is also known. It is obvious that the best solution is to assign half of objects of each type to layer 1 and the other half to layer 2. If $|\mathcal{S}_k|$ is even, then there are exactly $\frac{|\mathcal{S}_k|}{2}$ objects of the type $k$ on each layer, otherwise there can be no more than $\left\lceil \frac{|\mathcal{S}_k|}{2} \right\rceil$ objects. Thus, the optimal objective function value is

$$f(u) = \sum_{k=1}^{N^K} u_k := \sum_{k=1}^{N^K} \left\lceil \frac{|\mathcal{S}_k|}{2} \right\rceil \quad , \tag{2.5.1}$$

where $u_k$ represents the maximal number of objects of type $k$ over all layers (we use the objective function for $\text{BPSP}_3$). Without loss of generality we assume that $N^O$ is even. The assignment of the objects can be realized as follows. We group each pair of two consecutive objects so that the whole sequence is divided into $\frac{N^O}{2}$ pairs. Let $\mathcal{L} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{\frac{N^O}{2}}\}$ be a list of these pairs. Obviously, in each pair only two types are presented, $k_1$ and $k_2$. Assume that in each pair $\mathcal{P}_i$ in $\mathcal{L}$ the first object with number $(2i-1)$ has type $k_1$ and the second object with number $2i$ belongs to type $k_2$. For each type $k$ we define two functions:

- $p[k]$, that provides us with information about the objects assigned so far. This function takes values depending on the relation between the numbers of objects of type $k$ on the first and second layer ($M_{L_1}$ and $M_{L_2}$, respectively), *i.e.*,

$$p[k] = \begin{cases} -1, & M_{L_2}(k) = M_{L_1}(k) + 1 \\ 0, & M_{L_1}(k) = M_{L_2}(k) \\ 1, & M_{L_1}(k) = M_{L_2}(k) + 1 \end{cases} ;$$

- $d[k]$, that gives the number of objects of type $k$ which are already assigned to the layers.

Algorithm $D_1$ generates an assignment with the optimal objective function value for (2.5.1). The decision variables are defined in the following way:

$$\delta_{ij} = \begin{cases} 1, \text{ if object number } i \text{ is assigned to position number } j \\ 0, \text{ otherwise} \end{cases} .$$

The algorithm starts with the first pair of $\mathcal{L}$ and $\delta_{ij} := 0$ for all $i$ and $j$. The objects from this first pair have types $k_1$ and $k_2$, respectively.

Let the object of type $k_1$ go to the first layer, and the object of type $k_2$ to the second. We save this assignment in $p[k_1]$ and $p[k_2]$ and delete this pair from $\mathcal{L}$. Then we choose the next pair according to the rule described in Step 4. The next time an object of type $k_1$ appears in a pair, we assign it to layer 2, and the other object of this pair to layer 1. We proceed as long as there are elements left over in $\mathcal{L}$.

Algorithm $D_1$

Steps of the algorithm:

1 Generate the list of pairs of objects $\mathcal{L} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{\frac{N!}{2}}\}$.

2 Calculate $p[k]$ for every $k$;

If all $p[k] = 0$ then goto Step 3, else goto Step 4.

3 Take the first element $\mathcal{P}_i$ from $\mathcal{L}$;

Determine the types $k_1$ and $k_2$ of objects $2i-1$, $2i$, respectively;

Goto Step 6.

4 Take the first element $\mathcal{P}_i$ from $\mathcal{L}$ for which the following condition holds: $p[k_1] \neq 0$ or $p[k_2] \neq 0$ or both, where $k_1$ and $k_2$ are the types of the objects $2i-1$ and $2i$, respectively.

5 If $p[k_1] = \begin{cases} +1, & \text{then goto Step 6,} \\ -1, & \text{then goto Step 7,} \\ 0, & \text{then goto Step 8.} \end{cases}$

6 Assign $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, *i.e.*, change the order of objects;

Goto Step 9.

7 Assign: $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, *i.e.*, keep the order of objects;

Goto Step 9.

8 If $p[k_2] = \begin{cases} +1, & \text{then goto Step 7,} \\ -1, & \text{then goto Step 6.} \end{cases}$

9 Change $p[k_1]$ and $p[k_2]$ correspondingly to the assignment on Step 6 or Step 7;

Delete $\mathcal{P}_i$ from $\mathcal{L}$, goto Step 2.

Note that the above algorithm does not consider the cases $p[k_1] = p[k_2] = 1$ and $p[k_1] = p[k_2] = -1$. As the following lemma shows, these cases do not occur.

LEMMA 2.15 *After completing each step of algorithm $D_1$ the following holds: for any $n$ such that $\mathcal{P}_n \in \mathcal{L}$ and for any type $k$ there exist at most two $p[k] \neq 0$. If $\exists k_1, k_2 : p[k_1] \neq 0$, $p[k_2] \neq 0$, then either $p[k_1] = 1$ and $p[k_2] = -1$, or $p[k_1] = -1$ and $p[k_2] = 1$, and, additionally, for all $k \neq k_1$, $k \neq k_2 : p[k] = 0$.*

*Proof:*

**Proof.** We prove by induction over $n = 1, ..., \frac{N^O}{2}$.

$n = 1$. This means that we start with the objects with number 1 and 2 and $p[k] = 0$, $k = 1, ..., N^K$.

Consider the following cases:

1 Suppose the objects have different types. Let $1 \in \mathcal{S}_{k_1}$, $2 \in \mathcal{S}_{k_2}$ (the case $2 \in \mathcal{S}_{k_1}$, $1 \in \mathcal{S}_{k_2}$ is analogous). The first object goes to the first layer, *i.e.*, we assign $p[k_1] := 1$ in Step 8, the second one goes to the second layer, *i.e.*, we assign $p[k_2] := -1$ in Step 8. All other values $p[k]$ remain unchanged, *i.e.*, $p[k] = 0$, $k \neq k_1, k_2$.

2 Suppose both objects have the same type. Let $1, 2 \in \mathcal{S}_{k_1}$ (the case $1, 2 \in \mathcal{S}_{k_2}$ is analogous). After Step 8: $p[k_1] = 0 + 1 - 1 = 0$. All other values $p[k]$ are left unchanged, *i.e.*, $p[k] = 0$, $k \neq k_1$. So, we have $p[k] = 0$ for any $k$.

Thus, for $n = 1$ the statement is true. Now let us consider the induction step. Suppose the assumption is true for $n - 1$. Let us evaluate $p[k]$ for the case $n$:

1 The objects have different types $k_1$ and $k_2$. Depending on step $n - 1$ there are the following six possibilities:

- $p[k_1] = 1$; $p[k] = 0$, $k \neq k_1$.
  After Step 8: $p[k_1] = 1 - 1 = 0$; $p[k_2] = 0 + 1 = 1$; $p[k] = 0$, $k \neq k_2$;
- $p[k_1] = -1$; $p[k] = 0$, $k \neq k_1$.
  After Step 8: $p[k_1] = -1 - 1 = 0$; $p[k_2] = 0 - 1 = -1$; $p[k] = 0$, $k \neq k_2$;
- $p[k_1] = 1$, $p[k_2] = -1$; $p[k] = 0$, $k \neq k_1, k_2$.
  After Step 8: $p[k_1] = 1 - 1 = 0$; $p[k_2] = -1 + 1 = 0$; So, $\forall k : p[k] = 0$;
- $p[k_1] = -1$, $p[k_2] = 1$; $p[k] = 0$, $k \neq k_1, k_2$.
  After Step 8: $p[k_1] = -1 + 1 = 0$; $p[k_2] = 1 - 1 = 0$; $p[k] = 0$; So, $\forall k : p[k] = 0$;
- $p[k_1] = 1$, $\exists k_3 : p[k_3] = -1$; $p[k_2] = 0$; $p[k] = 0$, $k \neq k_1$, $k \neq k_3$.

After Step 8: $p[k_1] = 1-1 = 0$; $p[k_2] = 0+1 = 1$; $p[k_3] = -1+0 = -1$; $p[k] = 0$, $k \neq k_2$, $k \neq k_3$;

- $p[k_1] = -1$, $\exists k_3 : p[k_3] = 1$; $p[k_2] = 0$; $p[k] = 0$, $k \neq k_1$, $k \neq k_3$.
  After Step 8: $p[k_1] = -1+1 = 0$; $p[k_2] = 0-1 = -1$; $p[k_3] = 1+0 = 1$; $p[k] = 0$, $k \neq k_2$, $k \neq k_3$.

These variants are symmetric with respect to $k_1$ and $k_2$ in the sense that we can replace $k_1$ by $k_2$ and vice versa. So, one can see that in any case we have no more then two values $k$ with $p[k] \neq 0$. If we have exactly two such $k$, $p[k_1]$ and $p[k_2]$ have different signs.

2 Both objects have the same type $k_1$. Depending on the previous information during step $n$ there are four possibilities:

- $p[k_1] = 1$; $p[k] = 0$, $k \neq k_1$.
  After Step 8: $p[k_1] = 1-1+1 = 1$; $p[k] = 0$, $k \neq k_1$;
- $p[k_1] = -1$; $p[k] = 0$, $k \neq k_1$.
  After Step 8: $p[k_1] = -1+1-1 = -1$; $p[k] = 0$, $k \neq k_1$;
- $p[k_1] = 1$, $p[k_2] = -1$; $p[k] = 0$, $k \neq k_1$, $k \neq k_2$.
  After Step 8: $p[k_1] = 1-1-1 = 1$; $p[k_2] = -1-0 = -1$; $p[k] = 0$, $k \neq k_1$, $k \neq k_2$;
- $p[k_1] = -1$, $p[k_2] = 1$; $p[k] = 0$, $k \neq k_1$, $k \neq k_2$.
  After Step 8: $p[k_1] = -1+1-1 = -1$; $p[k_2] = 1+0 = 1$; $p[k] = 0$, $k \neq k_1$, $k \neq k_2$.

Again, these variants are symmetric with respect to $k_1$ and $k_2$. Thus, in any case we have no more then two $k$: $p[k] \neq 0$. In case we have exactly two $k$, $p[k_1]$ and $p[k_2]$ have different signs (q.e.d).

■

The following theorem ensures that the algorithm $D_1$ finds an optimal solution in polynomial time.

THEOREM 2.16 *Algorithm $D_1$ constructs an optimal solution in polynomial time.*

***Proof.*** *The optimality of the generated assignment follows directly from algorithm $D_1$ and the lemma above, because we always assign the objects in such a way that one half of the objects of type $k$ is assigned to section 1 and the second half to section 2. This is possible as long as we have the pre-sorting facility of capacity one. Let us see how many operations are necessary to construct the optimal solution. All steps of algorithm $D_1$ are repeated at most $\frac{N}{2}$ times, because each time we delete one element*

*from $\mathcal{L}$. Initially, $\mathcal{L}$ contains $\frac{N}{2}$ elements. In order to check if $i \in \mathcal{S}_k$ we need no more than $N^K$ operations. Checking if all $p[k] = 0$ requires also $N^K$ operations. And finally, Step 3 can be repeated at most $\frac{N}{2} - 3$ times, because $i$ has a value at least two before the step. So, in total there are $O(N^O, N^K) = \frac{N^O}{2}(N^K + N^K + \frac{N^O}{2} - 3) = \frac{(N^O)^2}{4} + N^O N^K - \frac{3N^O}{2}$. As one can see it is polynomial in the number of objects $N^O$ as well as in the number of types $N^K$ (q.e.d).* ■

### 2.5.2 Online Situations

In the following we present an online algorithm that is linear in $N^O$ and $N^K$. It is, therefore, much faster than the offline algorithm $D_1$. The sequence of objects appears in pairs: $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{N^O/2}$. When we assign layers for the objects from $\mathcal{P}_i$, the online algorithm does not have any knowledge of $\mathcal{P}_j$, $j > i$. Only after $\mathcal{P}_i$ has been served, the next pair $\mathcal{P}_{i+1}$ becomes known. In total there are $N^O$ objects. Without loss of generality, $N^O$ is assumed to be even. We group two consecutively objects into $\frac{N^O}{2}$ pairs. Obviously, in each pair of objects there are no more than two types $k_1$ and $k_2$. Let $C_{kl}$ specify the number of objects of type $k$ already assigned to the layer $l$.

Algorithm $D_2$

For $i := 1$ to $\frac{N^O}{2}$ do:

1 Determine types $k_1$ and $k_2$ of the objects $2i - 1$, $2i$ respectively.

2 If $(k_1 = k_2)$ or $(C_{k_1 1} = C_{k_1 2})$ and $(C_{k_2 1} = C_{k_2 2})$, then goto Step 4, else goto Step 3.

3 If $(C_{k_1 1} > C_{k_1 2})$ or $(C_{k_1 1} = C_{k_1 2}$ and $C_{k_2 2} > C_{k_2 1})$, then goto Step 5, else goto Step 4.

4 Assign $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, *i.e.*, keep the order of objects unchanged;

Goto Step 6.

5 Assign $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, *i.e.*, change the order of objects;

Goto Step 6.

6 Change $C_{k_1 1}$, $C_{k_1 2}$, $C_{k_2 1}$ and $C_{k_2 2}$ with respect to the assignment.

End For.

THEOREM 2.17 *The algorithm $D_2$ finds the best online solution in polynomial time.*

***Proof.*** *We prove the theorem by induction over* $n = 1, ..., \frac{N}{2}$.

$n = 1$, *i.e,* $C_k^0 = 0$, $k = 1, ..., N^K$.

$$f^1(u) = \sum_{k=1}^{N^K} u_k^1 = \sum_{k=1}^{N^K} \max\{C_{k1}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i1}; C_{k2}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i2}\}$$

*Consider the following cases:*

*1. Both objects have the same type:* $\{1, 2\} \subseteq \mathcal{S}_{k_1}$.

$$f^1(u) = \sum_{k=1}^{N^K} u_k^1 =$$

$$= \sum_{k=1}^{N^K} \max\{C_{k1}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i1}; C_{k2}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i2}\}$$

$$= \max\{C_{k_1 1}^0 + \sum_{i \in \mathcal{S}_{k_1}} \delta_{i1}; C_{k_1 2}^0 + \sum_{i \in \mathcal{S}_{k_1}} \delta_{i2}\}$$

$$+ \max\{C_{k_2 1}^0 + \sum_{i \in \mathcal{S}_{k_2}} \delta_{i1}; C_{k_2 2}^0 + \sum_{i \in \mathcal{S}_{k_2}} \delta_{i2}\}$$

$$= \max\{C_{k_1 1}^0 + \delta_{11} + \delta_{21}; C_{k_1 2}^0 + \delta_{12} + \delta_{22}\} + \max\{C_{k_2 1}^0; C_{k_2 2}^0\}$$

$$= \max\{\delta_{11} + \delta_{21}; \delta_{12} + \delta_{22}\} = 1,$$

*Therefore, the objective function value is 1, independent of how we permute the objects. Only one and exactly one element of the sums* $\delta_{11} + \delta_{21}$ *and* $\delta_{12} + \delta_{22}$ *has to be equal to* 1. *Consequently,* $\delta_{11} + \delta_{12} = \delta_{21} + \delta_{22} = 1$.

*2. The case* $\{1, 2\} \subseteq \mathcal{S}_{k_2}$ *is analogous to Case 1.*

*3. The objects have different types:* $\{1\} \subseteq \mathcal{S}_{k_1}$, $\{2\} \subseteq \mathcal{S}_{k_2}$ $(k_1 \neq k_2)$.

$$f^1(u) = \sum_{k=1}^{N^K} u_k^1$$

$$= \sum_{k=1}^{N^K} \max\{C_{k1}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i1}; C_{k2}^0 + \sum_{i \in \mathcal{S}_k} \delta_{i2}\}$$

$$= \max\{C_{k_1 1}^0 + \sum_{i \in \mathcal{S}_{k_1}} \delta_{i1}; C_{k_1 2}^0 + \sum_{i \in \mathcal{S}_{k_1}} \delta_{i2}\}$$

$$+ \max\{C_{k_2 1}^0 + \sum_{i \in \mathcal{S}_{k_2}} \delta_{i1}; C_{k_2 2}^0 + \sum_{i \in \mathcal{S}_{k_2}} \delta_{i2}\}$$

$$= \max\{C_{k_1 1}^0 + \delta_{11}; C_{k_1 2}^0 + \delta_{12}\} + \max\{C_{k_2 1}^0 + \delta_{21}; C_{k_2 2}^0 + \delta_{22}\}$$

$$= \max\{\delta_{11}; \delta_{12}\} + \max\{\delta_{21}; \delta_{22}\} = 1 + 1 = 2.$$

*Therefore, the objective function value is 2, independent of how we permute the objects. Only one and exactly one element of the pairs* $\{\delta_{11}; \delta_{12}\}$ *and* $\{\delta_{21}; \delta_{22}\}$ *has to be equal* 1. *Consequently,*

$$\max\{\delta_{11}; \delta_{12}\} = \max\{\delta_{21}; \delta_{22}\} = 1.$$

*4. The case* $\{2\} \subseteq \mathcal{S}_{k_1}$, $\{1\} \subseteq \mathcal{S}_{k_2}$ *is symmetric to Case 3.*

*Thus, for $n = 1$ we get the minimal objective function value. Now consider the induction step. Suppose, the assumption is true for $n-1$. Let us now analyze the case $n$. Again, we have to consider several cases:*

*1. Both objects are of the same type: $\{2n-1,\ 2n\} \subseteq \mathcal{S}_{k_1}$.*

$$
\begin{aligned}
f^n(u) &= \sum_{k=1}^{N^K} u_k^n \\
&= \sum_{k=1}^{N^K} \max\{C_{k1}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i1}; C_{k2}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i2}\} \\
&= \sum_{k=1,k\neq k_1,k_2}^{N^K} \max\{C_{k1}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i1}; C_{k2}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i2}\} \\
&+ \max\{C_{k_11}^{n-1} + \sum_{i\in\mathcal{S}_{k_1}} \delta_{i,2n-1}; C_{k_12}^{n-1} + \sum_{i\in\mathcal{S}_{k_1}} \delta_{i,2n}\} \\
&+ \max\{C_{k_21}^{n-1} + \sum_{i\in\mathcal{S}_{k_2}} \delta_{i,2n-1}; C_{k_22}^{n-1} + \sum_{i\in\mathcal{S}_{k_2}} \delta_{i,2n}\} \\
&= \max\{C_{k_11}^{n-1} + \delta_{2n-1,2n-1} + \delta_{2n,2n-1}; C_{k_12}^{n-1} + \delta_{2n-1,2n} + \delta_{2n,2n}\} \\
&+ \max\{C_{k_21}^{n-1}; C_{k_22}^{n-1}\} + \sum_{k=1,k\neq k_1,k_2}^{N^K} u_k^{n-1} \\
&= u_{k_1}^{n-1} + 1 + u_{k_2}^{n-1} + \sum_{k=1,k\neq k_1,k_2}^{N^K} u_k^{n-1} \\
&= f^{n-1}(u) + 1.
\end{aligned}
$$

*The objective function value increases by 1, independent of the permutation of the objects. Cf. Case 1 in the initial step of the induction.*

*2. The case $\{2n-1, 2n\} \subseteq \mathcal{S}_{k_2}$ is analogous to Case 1.*

*3. The objects have different types: $\{2n-1\} \subseteq \mathcal{S}_{k_1}$, $\{2n\} \subseteq \mathcal{S}_{k_2}$.*

$$
\begin{aligned}
f^n(u) &= \sum_{k=1}^{N^K} u_k^n \\
&= \sum_{k=1}^{N^K} \max\{C_{k1}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i1}; C_{k2}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i2}\} \\
&= \sum_{k=1|k\neq k_1,k_2}^{N^K} \max\{C_{k1}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i1}; C_{k2}^{n-1} + \sum_{i\in\mathcal{S}_k} \delta_{i2}\} \\
&+ \max\{C_{k_11}^{n-1} + \sum_{i\in\mathcal{S}_{k_1}} \delta_{i,2n-1}; C_{k_12}^{n-1} + \sum_{i\in\mathcal{S}_{k_1}} \delta_{i,2n}\} \\
&+ \max\{C_{k_21}^{n-1} + \sum_{i\in\mathcal{S}_{k_2}} \delta_{i,2n-1}; C_{k_22}^{n-1} + \sum_{i\in\mathcal{S}_{k_2}} \delta_{i,2n}\} \\
&= \max\{C_{k_11}^{n-1} + \delta_{2n-1,2n-1}; C_{k_12}^{n-1} + \delta_{2n-1,2n}\} \\
&+ \max\{C_{k_21}^{n-1} + \delta_{2n,2n-1}; C_{k_22}^{n-1} + \delta_{2n,2n}\} + \sum_{k=1|k\neq k_1,k_2}^{N^K} u_k^{n-1}
\end{aligned}
$$

*In this case the objective function value depends on the permutation of the objects. Consider all possible combinations and calculate the objective function value after changing the order of the objects, or if the order of objects was kept:*

| | | *Change* | *Keep* |
|---|---|---|---|
| $C_{k_1 1}^{n-1} > C_{k_2 1}^{n-1}$ | $C_{k_2 1}^{n-1} > C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+1$ | $f^{n-1}(u)-1$ |
| | $C_{k_2 1}^{n-1} = C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+1$ | $f^{n-1}(u)+2$ |
| | $C_{k_2 1}^{n-1} < C_{k_2 2}^{n-1}$ | $f^{n-1}(u)$ | $f^{n-1}(u)+2$ |
| $C_{k_1 1}^{n-1} = C_{k_1 2}^{n-1}$ | $C_{k_2 1}^{n-1} > C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+2$ | $f^{n-1}(u)+1$ |
| | $C_{k_2 1}^{n-1} = C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+2$ | $f^{n-1}(u)+2$ |
| | $C_{k_2 1}^{n-1} < C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+1$ | $f^{n-1}(u)+2$ |
| $C_{k_1 1}^{n-1} < C_{k_1 2}^{n-1}$ | $C_{k_2 1}^{n-1} > C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+2$ | $f^{n-1}(u)$ |
| | $C_{k_2 1}^{n-1} = C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+2$ | $f^{n-1}(u)+1$ |
| | $C_{k_2 1}^{n-1} < C_{k_2 2}^{n-1}$ | $f^{n-1}(u)+1$ | $f^{n-1}(u)+1$ |

*Keeping the order implies* $\delta_{2n-1,2n-1} := \delta_{2n,2n} := 1$,
*Changing the order implies* $\delta_{2n-1,2n} := \delta_{2n,2n-1} := 1$;

*Note that algorithm* $D_2$ *assigns the values to the decision variables in such a way that the objective function is minimized, i.e., the value for* $f^n(u)$ *is minimal as long as* $f^{n-1}(u)$ *is minimal (assumption of the induction step).*

*4. The case* $\{2n\} \subseteq \mathcal{S}_{k_1}$, $\{2n-1\} \subseteq \mathcal{S}_{k_2}$ *is symmetric to Case 3. Thus, instead of keeping we need to change the order of the objects. This is what algorithm* $D_2$ *does.*

*Let us now see how many operations are necessary to construct the optimal solution. For checking if* $i \in \mathcal{S}_k$ *we need no more than* $N^K$ *operations, since in the worst case the set* $\mathcal{S}_k$ *can contain all input objects. Steps 1 to 6 are repeated* $\frac{N^O}{2}$ *times. Thus, in total we have* $O(N^O) = \frac{N^O}{2} N^K$. *As one can see, it is polynomial with respect to the number of objects* $N^O$ *as well as to the number of types* $N^K$ *(q.e.d).* ■

## 2.5.3 Algorithms for Online Situations with Lookahead

### 2.5.3.1 Definition of a Lookahead

Very often the knowledge of future requests would help to improve the objective function value and to construct more efficient algorithms. Such situations will be referred as situations with lookahead. In recent years, a lot of problems with lookahead have been studied (paging problems, bin packing problems, graph problems and so on). It was expected that knowing a lookahead will improve the solution for the BPSP as well. The reason is the following: suppose we have the input sequence described in Example 2.23. Obviously, in this case, if we knew the third pair while

serving the second - we will obtain a better solution. So, in this section we describe algorithms with lookahead for the BPSP with $N^L = 2$ (later we make a brief comparative analysis of algorithms with lookahead and without).

Formal definitions of lookahead can be found in [3], [38]. In this book we use the simplified definitions given in [29]:

DEFINITION 2.18 *Weak lookahead with respect to the number of requests (WLNR) The online algorithm sees the present request at time $t$ and the next $\lambda - 1$ succeeding requests. Request $t + \lambda$ is not seen by the algorithm at time $t$. However, once the request $\delta(t)$ is processed, a new request, i.e., $\delta(t + \lambda)$, becomes known.*

DEFINITION 2.19 *Strong lookahead with respect to the number of requests (SLNR) The online algorithm sees $\lambda$ present requests at time $t$, and before all these are processed no new request becomes known.*

In our case, we consider a lookahead containing not simple requests (objects), but pairs of objects, *i.e.*, we say that the size of the lookahead is $\lambda = n$ if the algorithm sees the present pair at time $t$ and sees and considers $n$ succeeding pairs.

#### 2.5.3.2 An Algorithm with a Weak Lookahead, size $\lambda = 1$

The idea of this algorithm is quite simple: the sequence of objects is represented in pairs as was described above (for algorithm $D_2$). At the time we have to serve the current pair $\mathcal{P}_i$, also objects from pair $\mathcal{P}_{i-1}$ are known. It means there are four objects (two in each pair) and there are four possibilities to permute the objects: to keep the order of objects in both pairs, to change the order in both pairs and to keep the order of objects in one pair, changing the order in the second one. Then, depending on the values of the evaluated objective function $z = \sum_{k=1}^{N^K} \sum_{l=1}^{N^L} C_{kl}$, we realize the most efficient ($z$ is minimal) permutation on objects from the current pair $\mathcal{P}_i$. Next we consider the following pair of objects, $\mathcal{P}_{i+1}$, and produce the same calculations. For the last pair, $\mathcal{P}_{\frac{N^O}{2}}$, we take the decision based on the previous assignment.

Algorithm $WL_1$

Initialize $i := 1$.

1 Determine the types $k_1$ and $k_2$ of the objects $2i - 1$, $2i$, respectively.

2 If $i < \frac{N^O}{2}$, then goto Step 3, else goto Step 5.

3 Determine the types $l_1$ and $l_2$ of the objects $2i+1$, $2i+2$, respectively;

Goto Step 4.

4 If $k_1 \neq l_1, l_2$ and $k_2 \neq l_1, l_2$, then goto Step 5, else goto Step 6.

5 For the current $i$, $k_1$ and $k_2$ apply Steps 2-6 of algorithm $D_2$;
If $i = \frac{N^O}{2}$, then STOP, otherwise Goto 15.

6 Assign temporarily $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, $\delta_{2i+1,2i+1} := \delta_{2i+2,2i+2} := 1$, *i.e.*, keep the order of both: the first pair of objects and the second pair of objects.

7 Calculate the value of $S_K^K = \sum_k \sum_l C_{kl}$.

8 Assign temporarily $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, $\delta_{2i+1,2i+2} := \delta_{2i+2,2i+1} := 1$, *i.e.*, keep the order of objects for the first pair and change the order of objects for the second pair.

9 Calculate the value of $S_C^K = \sum_k \sum_l C_{kl}$.

10 Assign temporarily $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, $\delta_{2i-1,2i+1} := \delta_{2i+2,2i+2} := 1$, *i.e.*, keep the order of objects for the second pair and change the order of objects for the first pair.

11 Calculate the value of $S_K^C = \sum_k \sum_l C_{kl}$.

12 Assign temporarily $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, $\delta_{2i+1,2i+2} := \delta_{2i+2,2i-1} := 1$, *i.e.*, to change the order of both: the first pair of objects and the second pair of objects.

13 Calculate the value $S_C^C = \sum_k \sum_l C_{kl}$.

14 Compare the values of $S_K^K$, $S_C^K$, $S_K^C$, $S_C^C$. If $S_K^K$ or $S_C^K$ has the minimal value, then finally assign $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, otherwise assign $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$.

15 $i := i + 1$;
Goto Step 1.

#### 2.5.3.3 Algorithms with a Weak Lookahead, size $\lambda \geq 2$

If we have a lookahead of size $\lambda \geq 2$ there are two different strategies to handle the rest of the input sequence with number of pairs in lookahead less or equal than $\lambda$. The first possibility: when we consider pair $\mathcal{P}_\lambda$, such that $\lambda = \frac{N^O}{2} - i$ to run the offline algorithm $D_1$ for the rest of the sequence (algorithm $\mathrm{WL}_\lambda^1$); the second possibility: to serve each

individual pair separately irrespective of how many pairs are left in the lookahead (algorithm $\mathrm{WL}^2_\lambda$).

Algorithm $\mathrm{WL}^1_\lambda$

1 For $i := 1$ to $\frac{N^O}{2}$ do:

2 If $\lambda = \frac{N^O}{2} - i$ then apply the offline algorithm $D_1$ to the input sequence $\mathcal{P}_i, ..., \mathcal{P}_{i+\lambda}$ and *STOP*, else goto the next step.

3 Call algorithm $D_1$ with $\mathcal{P}_{i-1}, ..., \mathcal{P}_{i+\lambda}$ as input sequence.

4 Assign temporarily $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, *i.e.*, keep the order of objects in $\mathcal{P}_i$ unchanged.

5 Calculate the value of $S_K = \sum_k \sum_l C_{kl}$.

6 Assign temporarily $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, *i.e.*, change the order of objects in $\mathcal{P}_i$.

7 Calculate the value of $S_C = \sum_k \sum_l C_{kl}$.

8 If $S_K < S_C$ then assign constantly $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, else assign constantly $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$.

End For.

Algorithm $\mathrm{WL}^2_\lambda$

For $i := 1$ to $\frac{N^O}{2}$ do:

1 If $\lambda < (\frac{N^O}{2} - i)$, then goto Step 2, else goto Step 3.

2 Apply algorithm $D_1$ to the input sequence $\mathcal{P}_{i+1}, ..., \mathcal{P}_{i+\lambda}$ and goto Step 6.

3 If $\frac{N^O}{2} - i = 0$, then goto Step 4, else goto Step 5.

4 Apply algorithm $D_2$ to the last pair $\mathcal{P}_i$ and *STOP*.

5 Apply algorithm $D_1$ to the input sequence $\mathcal{P}_{i+1}, ..., \mathcal{P}_{\frac{N^O}{2}}$ and goto Step 6.

6 Assign temporarily $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, *i.e.*, keep the order of objects in $\mathcal{P}_i$ unchanged.

7 Calculate the value of $S_K = \sum_k \sum_l C_{kl}$.

8 Assign temporary $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$, *i.e.*, change the order of objects in $\mathcal{P}_i$.

9 Calculate the value of $S_C = \sum_k \sum_l C_{kl}$.

10 If $S_K < S_C$ then finally assign $\delta_{2i-1,2i-1} := \delta_{2i,2i} := 1$, else assign $\delta_{2i-1,2i} := \delta_{2i,2i-1} := 1$.

End For.

#### 2.5.3.4 An Algorithm with a Strong Lookahead

In the following we present an online algorithm with strong lookahead of size $\lambda$. It means that at the beginning pairs $\mathcal{P}_1, ..., \mathcal{P}_\lambda$ are known, after serving them pairs $\mathcal{P}_{\lambda+1}, ..., \mathcal{P}_{2\lambda}$ become known, and so on. Therefore, we divide the input sequence into $q := \lceil \frac{N^O}{\lambda} \rceil$ groups and operate with each group individually.

Algorithm $\mathrm{SL}_\lambda$

For $i := 1$ to $q$ do:
Apply the offline algorithm $D_1$ to the input sequence $\mathcal{P}_{(i-1)\lambda+1}, ..., \mathcal{P}_{i\lambda}$.
End For.

### 2.5.4 Competitive Analysis

The efficiency of online algorithms can be evaluated using competitive analysis, where online algorithms are compared with an optimal offline algorithm. First we collect some definitions and terms used throughout this chapter. One basic concept used for estimating the performance of online algorithms is the *competitive ratio*. The competitive ratio of an online algorithm can be defined with respect to an optimal offline algorithm, which knows an input sequence in advance and produces an optimal solution with minimal cost. Let $A$ be an online algorithm, $\delta$ is an input sequence of requests and $f_A(\delta)$ is the cost achieved by $A$ on the input sequence $\delta$. Denote by $f_{OPT}(\delta)$ the cost achieved by an optimal offline algorithm $OPT$ on the same input. Now we can give the definition of the competitive ratio of the online algorithm $A$.

DEFINITION 2.20 *([11])An online algorithm $A$ is $c-$competitive if there is a constant $\alpha$ such that for all finite input sequences $\delta$,*

$$f_A(\delta) \leq c \cdot f_{OPT}(\delta) + \alpha.$$

*When for the additive constant $\alpha \leq 0$ holds, we may say for emphasis that $A$ is strictly $c-$competitive.*

In the following the competitiveness of the algorithms from Section 2.5 is analyzed. We will show that algorithm $D_2$ is strictly $\frac{3}{2}-$ competitive. To do this we have to prove the following two lemmas.

LEMMA 2.21 *Any online algorithm which solves the given sequencing problem is strictly $N^L-$ competitive, where $N^L$ is the number of layers.*
***Proof.*** *The objective function has the largest value if all objects of the same type are assigned to the same layer. Thus, the maximum number, $u_k$, of objects of type $k$ in any layer is $|\mathcal{S}_k|$. Evaluate the objective function value: $f(u) = \sum_{k=1}^{N^K} u_k = \sum_{k=1}^{N^K} |\mathcal{S}_k|$. Clearly, the corresponding optimal offline objective function value is $f_{D_1}(u) = \sum_{k=1}^{N^K} u_k = \sum_{k=1}^{N^K} \left\lceil \frac{|\mathcal{S}_k|}{N^L} \right\rceil$. Suppose, that for all $k$ : $|\mathcal{S}_k| \equiv 0 \ mod\ N^L$, then $\left\lceil \frac{|\mathcal{S}_k|}{N^L} \right\rceil = \frac{|\mathcal{S}_k|}{N^L}$. If there exists $k$ such that $|\mathcal{S}_k| \equiv \{1, \ldots, N^L - 1\} \ mod\ N^L$, then $\left\lceil \frac{|\mathcal{S}_k|}{N^L} \right\rceil > \frac{|\mathcal{S}_k|}{N^L}$ and*

$$\frac{f(u)}{f_{D_1}(u)} = \frac{\sum_{k=1}^{N^K} |\mathcal{S}_k|}{\sum_{k=1}^{N^K} \left\lceil \frac{|\mathcal{S}_k|}{N^L} \right\rceil} \leq \frac{\sum_{k=1}^{N^K} |\mathcal{S}_k|}{\sum_{k=1}^{N^K} \frac{|\mathcal{S}_k|}{N^L}} \leq \frac{\sum_{k=1}^{N^K} |\mathcal{S}_k|}{\frac{1}{N^L} \sum_{k=1}^{N^K} |\mathcal{S}_k|} \leq N^L \quad .$$

*This implies that the competitive ratio is $N^L$ (q.e.d).* ■

Consider a situation with only two layers, *i.e.*, $N^L = 2$. Then the following lemma holds.

LEMMA 2.22 *If an algorithm $A$ is a $c-$competitive online algorithm, then $\frac{3}{2} \leq c \leq 2$.*
***Proof.*** *From Lemma 2.21 with $N^L = 2$ we conclude that the upper bound for $c$ is 2. We show that $c$ cannot be less than $\frac{3}{2}$. For any positive integer number $G$ we define $K := 4G$, $N := 8G$ and $|\mathcal{S}_k| = 2$ for all $k = 1, \ldots, N^K$, i.e., we only have two objects of each type. Consider the following input sequence $S = S_1 S_2 \ldots S_G$, where $S_g$ $(g = 1, \ldots, G)$ consists of eight elements. $S_g = \{8g-7, \ldots, 8g-4, 8g-3, \ldots, 8g\}$. Let $8g-7 \in \mathcal{S}_{4g-3}$; $8g-6 \in \mathcal{S}_{4g-2}$; $8g-5 \in \mathcal{S}_{4g-1}$; $8g-4 \in \mathcal{S}_{4g}$. Define the types for the next four objects $\{8g-3, \ldots, 8g\}$ depending on how algorithm $A$ operates. Any online algorithm (with the same number of known objects) gets the input sequence of objects in pairs. The algorithm has to decide whether to keep the order of objects or to change it. If the algorithm keeps the order, the first object goes to the first layer, the second one to the second. Otherwise, the first object is sent to the second*

*layer, and the second one to the first layer. Hence, after serving the first two pairs* $\{8g-7, 8g-6\}$ *and* $\{8g-5, 8g-4\}$ *we can get four cases (see Table 2.1) and depending on the decision we can construct the next two pairs of objects:* $\{8g-3, 8g-2\}$ *and* $\{8g-1, 8g\}$ *for* $S_g$ *(see Table 2.2):*

*Table 2.1: Results sequences after running the algorithm A*

| | | | |
|---|---|---|---|
| *1* | $8g-7 \in \mathcal{S}_{4g-3}; 8g-6 \in \mathcal{S}_{4g-2};$ | $8g-5 \in \mathcal{S}_{4g-1}; 8g-4 \in \mathcal{S}_{4g};$ |
| *2* | $8g-7 \in \mathcal{S}_{4g-3}; 8g-6 \in \mathcal{S}_{4g-2};$ | $8g-5 \in \mathcal{S}_{4g}; 8g-4 \in \mathcal{S}_{4g-1};$ |
| *3* | $8g-7 \in \mathcal{S}_{4g-2}; 8g-6 \in \mathcal{S}_{4g-3};$ | $8g-5 \in \mathcal{S}_{4g-1}; 8g-4 \in \mathcal{S}_{4g};$ |
| *4* | $8g-7 \in \mathcal{S}_{4g-2}; 8g-6 \in \mathcal{S}_{4g-3};$ | $8g-5 \in \mathcal{S}_{4g}; 8g-4 \in \mathcal{S}_{4g-1};$ |

*Then an algorithm has to serve the next 2 pairs:* $\{8g-3, 8g-2\}, \{8g-1, 8g\}$.

*Table 2.2: Recommended sequences for the next objects:* $\{8g-3, \ldots, 8g\}$

| | | |
|---|---|---|
| *1* | $8g-3 \in \mathcal{S}_{4g-3}; 8g-2 \in \mathcal{S}_{4g-1};$ | $8g-1 \in \mathcal{S}_{4g-2}; 8g \in \mathcal{S}_{4g};$ |
| *2* | $8g-3 \in \mathcal{S}_{4g-3}; 8g-2 \in \mathcal{S}_{4g};$ | $8g-1 \in \mathcal{S}_{4g-2}; 8g \in \mathcal{S}_{4g-1};$ |
| *3* | $8g-3 \in \mathcal{S}_{4g-2}; 8g-2 \in \mathcal{S}_{4g-1};$ | $8g-1 \in \mathcal{S}_{4g-3}; 8g \in \mathcal{S}_{4g};$ |
| *4* | $8g-3 \in \mathcal{S}_{4g-2}; 8g-2 \in \mathcal{S}_{4g};$ | $8g-1 \in \mathcal{S}_{4g-3}; 8g \in \mathcal{S}_{4g-1};$ |

*Let us determine the objective function value after serving these pairs. Regardless of the algorithm's decision, the maximum number,* $u_k$*, of objects in any layer has the following values. Consider case 1:*

*If an algorithm changes the order of* $\{4g-3, 4g-1\}$*, then* $u_{4g-3} = 1$, $u_{4g-1} = 2$*; otherwise* $u_{4g-3} = 2$, $u_{4g-1} = 1$*. If an algorithm changes the order of* $\{4g-2, 4g\}$*, then* $u_{4g-2} = 1$, $u_{4g} = 2$*; otherwise* $u_{4g-3} = 2$, $u_{4g-1} = 1$*. Thus, for* $S_g$ *we have* $f_{A_{S_g}}(u) = u_{4g-3} - u_{4g-1} + u_{4g-2} + u_{4g} = 3 + 3 = 6$*. We get the same result for the remaining cases. Since objects from different* $S_g$ *have different types, we obtain*

$$f_A(u) = \sum_{g=1}^{G} f_{A_{S_g}}(u) = 6G = 6\frac{N^K}{4} = \frac{3}{2}N^K \quad .$$

*At the same time the optimal objective function value is*

$$f(u) = \sum_{k=1}^{N^K} u_k = \sum_{k=1}^{N^K} \frac{N_k}{2} = \sum_{k=1}^{N^K} 1 = N^K \quad .$$

*This means* $c = \frac{3}{2}$ *(q.e.d).* ■

Let us illustrate Lemma 2.22 by the following example.

EXAMPLE 2.23

If $G = 1 \Longrightarrow N^K = 4$, $N^O = 8$; $|S_k| = 2$ for $k = 1, ..., 4$, we have to construct the input sequence $S = S_1$ where $S_1$ consists of eight objects. The first four objects have types $1, 2, 3, 4$, respectively. After executing the algorithm we get one of the following cases:

- the algorithm kept the order of objects for both pairs. Therefore, we get the output sequence: $1, 2, 3, 4$. In this case the next four objects (from 5 to 8) must have types $1, 3, 2, 4$, respectively;
- the algorithm changed the order of the first pair and kept the order of the second. We get: $2, 1, 3, 4$. In this case the next four objects (from 5 to 8) must have types $2, 3, 1, 4$;
- the algorithm kept the order of the first pair and changed the order of the second. We get: $1, 2, 4, 3$. In this case the next four objects (from 5 to 8) must have types $1, 4, 2, 3$;
- the algorithm changed the order of both pairs. We have: $3, 1, 4, 2$. In this case the next four objects (from 5 to 8) must have types $3, 4, 1, 3$.

For the first case, independently of the actions of the algorithm on the objects $5, 6, 7, 8$, after these actions the following statements hold:

Both objects of the same type, either 1 or 3 are on the same layer;

Both objects of the same type, either 2 or 4 are on the same layer.

An equivalent argumentation holds for all other cases. Therefore, the best value of the objective function for the *online* case is

$$f_A(u) = \sum_{k=1}^{4} u_k = 1 + 2 + 1 + 2 = 6 \quad . \tag{2.5.2}$$

At the same time we get the optimal value of the objective function for the *offline* case:

$$f(u) = \sum_{k=1}^{4} u_k = \sum_{k=1}^{4} \frac{|S_k|}{2} = 4 \quad . \tag{2.5.3}$$

Using the Lemmas 4 and 5 we can now prove the following theorem.

THEOREM 2.24 *The algorithm $D_2$ is strictly $\frac{3}{2}$-competitive.*

***Proof.*** *The algorithm $D_2$ operates as follows. At first, the sequence of all incoming objects is grouped into a list of pairs of objects. These pairs*

*are numbered from* 1 *through* $\frac{N^O}{2}$. *From each pair of objects with types* $k_1$ *and* $k_2$, *the object of type* $k_1$ *is assigned to section* $L_1$ *and the object of type* $k_2$ *is assigned to section* $L_2$. *Thus, the next time an object of type* $k_1$ *appears in a pair, it is assigned to section* $L_2$ *and the other object of the current pair to section* $L_1$. *A problem could only occur if both objects of the current pair should be assigned to the same section, but this is not possible. This case was already considered in the proof of Lemma 2.22. Therefore the competitive ratio of algorithm* $D_2$ *is* $\frac{3}{2}$ *(q.e.d).* ■

### 2.5.5 Comparison of $D_2$ and Online Algorithms with Lookahead

Here we show that additional knowledge of input data improves the objective function value. Consider the worst case for the algorithm $D_2$, *i.e.*, the following input sequence $(1, 2, 3, 4, 1, 3, 2, 4)$ (the construction rule was described in Lemma 2.22 and Example 2.23). In Table 2.3, the objective function values, $f(i)$, for offline and online algorithms are presented, as well as the resulting permutations $\delta(i)$.

Table 2.3: Comparison of offline and online algorithms

| alg. | $\lambda$ | $f(i)$ | $\delta(i)$ |
|---|---|---|---|
| $D_1$ | 0 | 4 | $(1,2,4,3,6,5,7,8)$ |
| $D_2$ | 0 | 6 | $(1,2,3,4,6,5,7,8)$ |
| $WL_1$ | 1 | 4 | $(1,2,4,3,6,5,7,8)$ |
| $WL_\lambda^1$ | 1 | 6 | $(2,1,3,4,5,6,7,8)$ |
| | 2 | 4 | $(1,2,4,3,6,5,7,8)$ |
| | 3 | 4 | $(1,2,4,3,6,5,7,8)$ |
| $WL_\lambda^2$ | 1 | 4 | $(2,1,3,4,5,6,8,7)$ |
| | 2 | 6 | $(2,1,4,3,6,5,8,7)$ |
| | 3 | 6 | $(2,1,4,3,6,5,8,7)$ |
| $SL_\lambda$ | 1 | 6 | $(1,2,3,4,5,6,7,8)$ |
| | 2 | 6 | $(1,2,3,4,5,6,7,8)$ |
| | 3 | 4 | $(1,2,4,3,6,5,7,8)$ |

Thus, with only one pair of objects in the lookahead algorithms $WL_1$, $WL_\lambda^2$ produce an optimal permutation. Algorithms $WL_\lambda^1$ and $SL_\lambda$ require the knowledge of two and three pairs, respectively, to get an optimal permutation.

## 2.6. Extensions

In this section we consider a specific case of the sequencing problem, in which the number of layers is a power of 2, *i.e.*, $2^P$. Accordingly, without loss of generality, the number of known objects, $N^O$, is divided by $2^P$.

We show that for this problem there exists a polynomial algorithm if the capacity $N^S$ of the pre-sorting facility is sufficiently large.

In this case we can use the algorithm $D_2$ consequently $2^{P-1}$ times. In the following, we refer to this modification as algorithm $D_2^P$.

Algorithm $D_2^P$

Apply the algorithm $D_2$ to

- the whole sequence of $N^O$ objects.

  Output: two subsequences of $\frac{N^O}{2}$ objects;

- each of two subsequences of $\frac{N^O}{2}$ objects.

  Output: four subsequences of $\frac{N^O}{4}$ objects;

  ... ... ...

- each of $2^{P-1}$ subsequences of $\frac{N^O}{2^{P-1}}$ objects.

  Output: $2^P$ subsequences of $\frac{N^O}{2^P}$ objects.

THEOREM 2.25 *The $D_2^P$ algorithm described above is strictly $(\frac{3}{2})^P$-competitive.*

***Proof.*** *Follows directly from Theorem 2.24 and the description of $D_2$ (q.e.d).* ■

THEOREM 2.26 *The algorithm $D_2^P$ constructs a solution in polynomial time.*

**Proof.** By Theorem 2.17 the complexity of $D_2$ is $O(N^O, N^K) = \frac{N^O}{2}N^K$. Now we calculate the complexity of $D_2^P$. The first step of $D_2^P$ costs $\frac{N^O}{2}N^K$ operations, the second - $2\frac{N^O}{2^2}N^K$. Then we run $D_2$ four times for the $\frac{N^O}{2^2}$ objects, and thus the number of operations is $4\frac{N^O}{2^3}N^K$ and so on. In total the complexity is:

$$\begin{aligned} O(N^O, N^K) &= \frac{N^O N^K}{2} + 2\frac{N^O N^K}{2^2} + ... + 2^{P-1}\frac{N^O N^K}{2^P} \\ &= \sum_{p=1}^{P} 2^{p-1}\frac{N^O N^K}{2^p} = \frac{N^O N^K}{2}P \quad (q.e.d) \quad . \end{aligned}$$

■

As soon as a polynomial algorithm for three layers will be constructed, it is possible to develop an algorithm similar to $D_2^P$ for the problems with number of layers $3^P$, and $2^{P_1}3^{P_2}$.

## 2.7. Summary und Future Research

In this chapter we developed mathematical formulations and integer programming models for the Batch PreSorting Problems (BPSP). The main result is the proof of NP-completeness. So far it is still open whether $BPSP_3$ is polynomial or NP-complete. However, we expect that $BPSP_3$ is NP-complete as well. As the complexity proof for $BPSP_2$ involved the SAT problem, it is useful to adapt some existing efficient algorithms that have been developed to solve the SAT problem to the $BPSP_2$ problem.

In addition, we have investigated what makes this problem difficult. We constructed an alternative formulation and showed that it has an integer polyhedron and is equivalent to a restricted variant of $BPSP_2$ and $BPSP_3$; thus, this restricted version is also polynomially solvable. In addition we considered a polynomial subcase of the original formulation - the case of two layers. For this special case, we constructed one exact offline algorithm and several online algorithms with and without lookahead. We compared them using competitive analysis. One of those algorithm was adapted for a more general case of $2^P$ layers.

For further research, we suggest to construct a similiar polynomial algorithm for the case of three layers, and then to modify it for the problem with $2^{P_1}3^{P_2}$ layers in online case. For a better estimation of the efficiency of such algorithms it would be helpful to get a non-trivial (different from a constant) lower bound for a general BPSP. Another possible research direction is to construct a transformation from the problem with two layers/many colors to the problem with many layers/two colors.

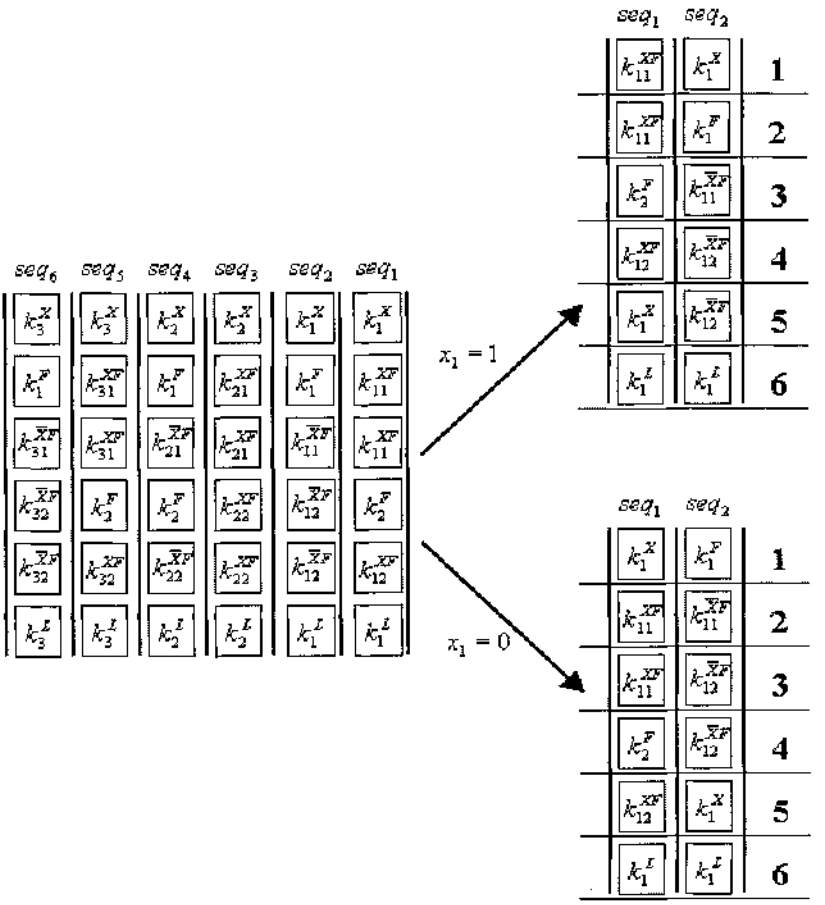


*Figure 2.3.8.* Changes in the sequence after assigning a value to $x_1$

http://www.springer.com/978-1-4020-2971-4

Online Storage Systems and Transportation Problems with Applications
Optimization Models and Mathematical Solutions
Kallrath, J.
2005, XIV, 222 p., Hardcover
ISBN: 978-1-4020-2971-4