Errata Brucker/Knust: Complex Scheduling, Springer, 2006

- p.25, 1: Given \( n \) positive integers ...
- p.26, 1: \( C_i + p_j \leq C_j \)
- p.30, Fig. 2.2, Step 7: FOR ALL \( j \in S \)
- p.37, Fig. 2.9, Step 2: FOR ALL \((i,j) \in V \times V\) with \( i \neq j \)
- p.39, 3: maximizes
- p.43, 10: \( \frac{n^m}{m} \)
- p.52, (2.40): \( n \sum_{j=1}^{m} r_{jk} \min\{t+p_j, T\} \leq R_k \) \( (k = 1, \ldots, r; t = 0, \ldots, T) \)
- p.53, -9: a typical value of \( m \) is no more than 50
- p.54, -5: If \( a_3 = 2 \), then (2.47) implies \( a_1 = 0 \)
- p.55, (2.50): \( \sum_{i=1}^{m} w_i a_i \leq W \ldots \)
- p.59, Figure 2.12(d): \( c'_{ij} \) instead of \( c_{ij} \)
- p.62, Theorem 2.7: The proof should read “Define \( x^1 := x - x^0 \). If \( x_{ij}^1 = x_{ij} - x_{ij}^0 > 0 \), then \( 0 \leq x_{ij}^0 < x_{ij} \leq u_{ij} \). Thus, the arc \((i,j)\) belongs to \( G(x^0) \) and we consider \( x_{ij}^1 \) as a flow on this arc in \( G(x^0) \). On the other hand, if \( x_{ij}^1 = x_{ij} - x_{ij}^0 < 0 \), then \( 0 \leq x_{ij} < x_{ij}^0 \). Thus, the arc \((j,i)\) belongs to \( G(x^0) \) and we consider \(-x_{ij}^1\) as a flow on this arc in \( G(x^0) \). ...”
- p.65, 12: \( u[X, \overline{X}] = \ldots \)
- p.65, -10: (for node \( j = p \))
- p.71, 18: the algorithm performs \( 2 \cdot 2^n \) augmentations
- p.78, Fig.2.23 should read:

```
Algorithm Branch-and-Bound Knapsack
1. \( L := 0; k := 0; \)
2. \( \text{FOR } j := k + 1 \text{ TO } n \text{ DO} \)
3. \( x_j := \left[ \frac{(b - \sum_{i=1}^{j-1} a_i x_i)}{a_j} \right]; \)
4. IF \( \sum_{i=1}^{n} c_i x_i > L \) THEN
5. \( L := \sum_{i=1}^{n} c_i x_i; \ x^* := x \)
6. ENDIF;
7. \( l := n - 1; \)
8. IF no \( 1 \leq k \leq l \) with \( x_k > 0 \) exists THEN STOP;
9. Determine the largest \( 1 \leq k \leq l \) with \( x_k > 0 \);
10. \( x_k := x_k - 1; \)
11. IF \( \sum_{i=1}^{k} c_i x_i + \frac{c_{k+1}}{a_{k+1}} (b - \sum_{i=1}^{k} a_i x_i) < L + 1 \) THEN
12. \( l := k - 1; \)
13. GOTO 8;
14. ENDIF;
15. ELSE GOTO 2;
```
• p.81, 19: where initially \( c(j, k) := -\infty \) for \( j = 0, \ldots, n \) and \( k < 0 \)

• p.82, 3: \( F(S, j) = \min_{i \in \Sigma \setminus \{j\}} \{F(S \setminus \{j\}, i) + c_{ij}\} \)

• p.102, (3.16): \( P(\Omega_{i, k} \setminus \{i\}) \)

• p.102, -12: The definitions of \( \Delta_{\lambda k} \) and \( H_{\lambda k} \) have to be

\[
\Delta_{\lambda k} := \max_{\nu \in \Omega_{\lambda k}} \{r_{\nu} + P_{\nu, k}\}, \quad \text{and} \quad H_{\lambda k} := \max_{\nu \in \Omega_{\lambda k}} \{r_{\nu} + P_{\nu, k}\}.
\]

• p.103, -3: The recursion should read \( P_{n+1, k} = \Delta_{n+1, k} = H_{1 k} := 0 \),

\[
P_{\lambda, k} = \sum_{\mu \geq \lambda} p_{\mu} = \begin{cases} P_{\lambda+1, k} + p_{\lambda}, & \text{if } d_{\lambda} \leq d_{k} \\
\lambda+1, & \text{otherwise}
\end{cases}
\]

for \( \lambda = n, \ldots, 1 \), and

\[
\Delta_{\lambda, k} := \max_{\nu \geq \lambda} \{r_{\nu} + P_{\nu, k}\} = \begin{cases} \max\{\Delta_{\lambda+1, k}, r_{\lambda} + P_{\lambda, k}\}, & \text{if } d_{\lambda} \leq d_{k} \\
\Delta_{\lambda+1, k}, & \text{otherwise}
\end{cases}
\]

for \( \lambda = n, \ldots, 1 \), and

\[
H_{\lambda, k} := \max_{\nu \geq \lambda} \{r_{\nu} + P_{\nu, k}\} = \begin{cases} \max\{H_{\lambda-1, k}, r_{\lambda-1} + P_{\lambda-1, k}\}, & \text{if } d_{\lambda-1} \leq d_{k} \\
H_{\lambda-1, k}, & \text{otherwise}
\end{cases}
\]

for \( \lambda = 2, \ldots, n \)

• p.106, (3.26): \( P(\Omega_{j, i} \setminus \{i\}) \)

• p.108, 2: The recursion should read:

Starting with \( P'_{j, 0} := 0 \), the values

\[
P'_{j, \lambda} = \sum_{\mu \leq \lambda} p_{\mu} = \begin{cases} P'_{j, \lambda-1} + p_{\lambda}, & \text{if } r_{\lambda} + p_{\lambda} \geq r_{j} + p_{j} \\
P'_{j, \lambda-1}, & \text{otherwise}
\end{cases}
\]

for \( \lambda = 1, \ldots, n \) can be computed in constant time from the previous values \( P'_{j, \lambda-1} \). Furthermore, with

\[
P_{j, \lambda} = \begin{cases} -\infty, & \lambda < j \\
P'_{j, \lambda}, & \lambda \geq j
\end{cases}
\]

and the initial values \( \Delta_{j, 0} := -\infty \) we calculate the values \( \Delta_{j, \lambda} \).

• p.109: The \( O(n^2) \)-description of the input-or-output test is wrong. It can be repaired, but then an \( O(n^3) \)-algorithm is obtained.

• p.114, 1: For example, if the hypothetical constraint \( S_i > t_i \) with \( t_i \in [r_i, d_i - p_i] \) leads to a contradiction, the time window of activity \( i \) can be reduced to \( [r_i, t_i + p_i] \).

• p.121, 5: \( \frac{UB - LB}{LB} \)

• p.132, 8: upper bound (instead of “lower bound”)

• p.144, -16: \( \left[ \max_{k=1}^{r} \{t^{m_{k-1}}\}, T \right] \)

Section 3.4.2: The algorithms are described in such a way that \( R_k(t) \) denotes the capacity of resource \( k \) in the interval \( [t, t + 1] \).

• p.148, Fig. 3.38: It would be more precise to replace steps 2 and 8 by:

2. Let \( E_k \) be the set of all activities \( i \) without predecessor
and \( r_{ik} \leq R_k(\tau) \) for \( k = 1, \ldots, r \) and all \( \tau \in [0, \ldots, p_i - 1] \);

8. Add \( j \) to \( A_{\lambda} \) and update the set \( E_\lambda \) by eliminating
and all activities \( i \) with \( r_{ik} > R_k(\tau) \) for some
resource \( k \) and a \( \tau \in \{t_\lambda, t_\lambda + p_i - 1\} \);
• p.149: Replace all $i$ by $j$ in the proof of Theorem 3.5.

• p.152, -1: We have at most $n(n - 1)/2$ feasible left shift operators and at most $n(n - 1)/2$ feasible right shift operators for a permutation.

• p.159: In the procedure in Fig.3.49 we have to distinguish between the set of eligible activities $E$ and the set of branching candidates $B$: 

```
Procedure B&B ($\lambda, F_\lambda, E_\lambda, S, s_{\lambda-1}$)
1. $B_\lambda := E_\lambda$;
2. WHILE $B_\lambda \neq \emptyset$ DO
3. Choose an activity $j \in B_\lambda$; $B_\lambda := B_\lambda \setminus \{j\}$;
4. Calculate the smallest time $t \geq s_{\lambda-1}$ such that $j$
   can be scheduled in the interval $[t, t + p_j]$ without
   violating resource or precedence constraints;
5. Schedule $j$ in the interval $[t, t + p_j]$ and set $S_j := t$;
6. $F_\lambda := F_\lambda \cup \{j\}$; $E_\lambda := E_\lambda \setminus \{j\}$; $s_\lambda := t$;
7. IF $\lambda = n + 1$ THEN
8. $UB := \min \{UB, S_{n+1}\}$; unschedule $n + 1$;
9. RETURN
10. ELSE
11. Let $E_{\lambda+1} := E_\lambda$ and add to $E_{\lambda+1}$ all successors
    $i \not\in E_\lambda$ of $j$ for which all predecessors are in $F_\lambda$;
12. Calculate a lower bound $LB(S)$ for all extensions
    of the current partial schedule $S$;
13. IF $LB(S) < UB$ THEN
14. $B&B (\lambda + 1, F_\lambda, E_{\lambda+1}, S, s_\lambda)$;
15. Unschedule $j$ and set $F_\lambda := F_\lambda \setminus \{j\}$; $E_\lambda := E_\lambda \cup \{j\}$;
16. ENDIF
17. ENDWHILE
```

• p.160, 15: The swapping rule, which can be applied for regular objective functions, should
more precisely be formulated as:

**Swapping rule:** If two activities $j, i$ are started at the same time, they can be swapped in
the sequence without resulting in a worse schedule. Thus, if we have considered all extensions
of $\ldots, i, j$ we do not have to consider the extensions of $\ldots, j, i$. Hence, in this situation it is
sufficient to consider only extensions $\ldots, i, j$ with $i < j$.

• p.164, Fig. 3.51: In the left box for $\lambda = 2$ it has to be $A_2 = \{1\}$. In the box above schedule
$S_4$ activities 2 and 3 should start at the same time.

• p.182, 13: $(b_k - y)^+ = \min \{\alpha_k \mid \alpha_k \geq 0, y + \alpha_k \geq b_k\}$ for $k = 1, \ldots, l$ and $(y - b_{k-1})^+ = \min \{\alpha_k \mid \alpha_k \geq 0, -y + \alpha_k \geq -b_{k-1}\}$ for $k = l + 1, \ldots, m + 1$