Appendix A: Greedy start procedure

This appendix defines a greedy assignment procedure, which quickly determines a feasible start solution for SSA by assigning storage positions one after another in a myopic manner. Note that a feasible solution can also be determined via perfect matching (see Theorem ??). However, as maximum distances are not considered in this approach, our greedy heuristic aims at even better objective values. Nonetheless, feasibility is ensured. We assume that subsets of feasible storage positions S_i represent a partial order. This is true, whenever the assignment restrictions depend on size, weight, or another at least ordinal scaled parameter.

Our greedy heuristic is subdivided into three basic steps.

1. Sort the SKUs according to non-decreasing number of feasible storage positions (i.e., $|S_i|$). Ties are broken according to lower index.
2. Consider SKUs in the given order. All items of the current SKU are iteratively assigned to the storage position leading to the highest reduction of the objective value. If no reduction can be achieved, chose the open position leading to the highest decrease of critical measuring points, i.e., those measuring points determining the maximum distance of the respective SKU. When both criteria are unsuccessful, a random storage position is selected.
3. If SKUs are still unassigned go to step 2, otherwise end the procedure and return the storage assignment.

Data: problem data, procedure specific parameters υ, χ, and ϱ
Result: best assignment found Γ

```
1  Γ := assignment containing only already assigned items;
2  sort SKUs in non-ascending order of their weighted maximum distances;
3  foreach SKU i in given order do
4  |  for (j = 0; j < n_i; j++) do
5  |  |  itemAssigned := false;
6  |  |  if assignment of item of SKU i can reduce Z(Γ) then
7  |  |  |  realize assignment with highest saving;
8  |  |  |  itemAssigned := true;
9  |  |  |  break;
10 |  |  if itemAssigned == false then
11 |  |  |  if assignment of item of SKU i can reduce κ'_i then
12 |  |  |  |  realize assignment with highest saving;
13 |  |  |  |  itemAssigned := true;
14 |  |  |  |  break;
15 |  |  if itemAssigned == false then
16 |  |  |  assign item to randomly chosen storage position;
```

Algorithm 1: Greedy start procedure

Appendix B: Local search

This appendix specifies a straightforward local search procedure for SSA. It is based on swap moves (i.e., two items of different SKUs change their storage positions) and applies first-fit search. The three basic steps of our procedure are detailed in the following and summarized in Algorithm 2.

1. Starting with a feasible solution, first, the SKUs are sorted according to non-decreasing quantity of critical measuring points κ'_i. A measuring point is critical with regard to a specific SKU $i \in I$, if the distance towards the closest item of i equals the SKU's maximum distance. Note that a value of $\kappa'_i > 1$ indicates that a single swap move is not sufficient to improve the solution. Therefore, we consider the SKUs with lower κ'_i values first. As a tie-breaker we apply $\kappa''_i = w_i \cdot (\max_{\tau' \in D}\{\min\{d_{\tau',i}(\Gamma); \delta_{\tau',i}\}\} - LB_i)$, so that SKUs with higher potential savings are preferred. Note that there is an exception for SKUs with $\kappa'' = 0$. These SKUs are not considered in the iterative part of the procedure, but, nonetheless, their storage positions are potentially swapped (see Algorithm 2, line 4).

2. Next, each SKU is considered in the given SKU sequence. For each storage position of the currently selected SKU, all storage positions available as a possible counterpart of a swap and reachable in less than $\max_{\tau' \in D}\{\min\{d_{\tau',i}(\Gamma); \delta_{\tau',i}\}\}$ steps from at least one of the critical measuring points are tested. Whenever a swap leads to an improvement of the current solution Γ, the swap is executed and the procedure jumps back to step 1 (see Algorithm 2, lines 5 to 7).

   Based on the quantity of critical measuring points κ'_i, an improvement of the current solution is defined in two different ways:

   - $\kappa'_i = 1$: Having only one critical measuring point, it may be possible to improve the objective value by one single swap. In order to find this swap, we define an improvement as a decrease of the objective value.
   - $\kappa'_i > 1$: Having multiple critical measuring points, one single swap is not sufficient to improve the objective value. Therefore, we define an improvement of the current solution as a reduction of the number of critical measuring points without increasing the objective value.

3. The procedure terminates if no improvement has been realized when iterating through all SKUs during step 2 or after a previously defined timeout is reached (see Algorithm 2, lines 8 to 9).

Data: problem data, time limit Δ, bounds LB_i, start solution Γ
Result: best assignment found Γ

```
1 newIteration: sort SKUs according to non-decreasing κ' and apply non-increasing κ'' as
  tie-breaker;
2 foreach SKU i ∈ I in the given order do
3 |   foreach storage position s' ∈ {s ∈ S_i : x_is = 1} do
4 |   |   foreach potential swap position s'' ∈ {s ∈ S_i : x_is = 0} do
5 |   |   |   if swap of items assigned to s' and s'' improves Γ then
6 |   |   |   |   perform swap;
7 |   |   |   |   goto: newIteration;
8 |   if executionTime > Δ then
9 |   |   break;
```

Algorithm 2: Local search