# Local search

Per la local search abbiamo deciso di non utilizzare l’algoritmo greedy per trovare la prima soluzione, ma di scegliere la prima soluzione in maniera più semplice. Cioè ordinano tutti gli oggetti nelle relative classi in base alla somma dei pesi. In seguito, cerchiamo gli oggetti con lo stesso indice che soddisfano il requisito di stare nello zaino.

In seguito, ordiniamo gli oggetti all’interno delle classi in base al valore, come erano all’inizio. E cerchiamo la corrispondenza degli oggetti.

Pseudo codice:

// A pre-algorithm sorting.

// It sorts every item in every class by weights

classes.stream()

.map(MyClass::getItems)

.forEach(items

-> items.sort((o1, o2)->   
 o2.getWeights().stream().reduce(0, Integer::sum)

- o1.getWeights().stream().reduce(0, Integer::sum)));

// search for first valid occurrence based on weight

for (int k = 0; k < biggestClass; k++) {

if (verifySolution(chosenItem))

break;

for (int j = 0; j < chosenItem.length; j++) {

chosenItem[j]++;

}

}

// sort item by values

this.classes.stream()

.map(MyClass::getItems)

.forEach(items -> items.sort((o1, o2) -> o2.getValue() - o1.getValue()));

// search the items position in the new sorted classes

for (int j = 0; j < chosenItem.length; j++) {

Item ourItem = orderedByWeight.get(j).getItems().get(chosenItem[j]);

for (int i = this.classes.get(j).getItems().size() - 1; i >= 0; i--) {

if (this.classes.get(j).getItems().get(i).equals(ourItem)) {

newChosenItems[j] = i;

}

}

}

chosenItem = newChosenItems.clone();

Poi per cercare una soluzione migliore utilizziamo il local search utilizzando il un sistema che abbiamo pensato basandoti sul 2-opt, utilizzato per risolvere il TPS. Noi invece di invertire due indici, ne aumentiamo uno, e ne diminuiamo un altro.   
do{  
 twoOpt();  
}  
while (!Arrays.*equals*(*previousChosenItem*, chosenItem));

Per quel che riguarda il vicinato utilizziamo la logica Steepest descent; quindi, esploriamo tutto il vicinato in qualunque caso.

La nostra condizione di stop è la seguente: !Arrays.*equals*(*previousChosenItem*, chosenItem)   
Quindi il nostro algoritmo utilizza un approccio iterativo, in cui memorizza il risultato prima della chiamata al metodo twoOpt() e dopo, se questi sono uguali vuol dire che non è stato trovato nessun vicino migliore, e solo in quel caso si ferma.