Repair Operator

The solution generated by solution modification operations during both 'Teaching phase' and 'Learning phase' may not be feasible, since multiple choice or multidimensional constraints may not be satisfied. We use a two phase procedure to regain solution feasibility. In phase 1, the multiple choice constraint is satisfied using a heuristic to maximize the objective function. Phase 2 obtains multidimensional feasibility by reducing the problem to a MCKP, using a surrogate constraint as our resource amount supplied by the knapsack.

Multiple Choice Feasibility

If more than one item is selected, or no item is selected from a group, we refer to our solution as infeasible by the multiple choice constraint of MMKP. In our procedure, feasibility is obtained by using a greedy heuristic to choose which item to keep from an infeasible group, while discarding all other item's. The greedy heuristic used can be described formally as:

$$x_{ij}=\begin{cases} 1, if \dfrac{v_{ij}}{\sum_{k\in M}(c_{ijk}/r_k)/|M|} = max\left\{ \dfrac{v_{ij}}{\sum_{k\in M}(c_{ijk}/r_k)/|M|} \;\; \forall x_{ij}=1 \mid j\in L_i \right\} \\ 0, \;\; otherwise \end{cases}$$

From the items, x, selected in group i, our objective is to find the one that maximizes the ratio of profit to resource use for all resource constraints of the knapsack. *[why did we choose to use this, instead of strictly profit],[we may also be able to say something here about how this performs for strong vs. weakly correlated problems].* If no items are selected in a group, one is selected at random. The procedure is as follows:

Fig _ : Multiple Choice Feasibility Operator

```
Input: Solution, S
Output: S, feasible by MC constraints
MCFeasOp(S):
FOR each group in S:
    if items selected > 1:
        selected_item = max{item profit to resource use ratio}
        de-select all items in group, except selected_item
    else if items selected = 0:
        select a random item in group
```

Multidimensional Feasibility

If any knapsack resource constraints are unsatisfied, the solution is infeasible by multidimensional constraints. Regaining feasibility for multidimensionality is based on swapping an item selected in a group for another, which does satisfy resource constraints. Since MMKP is strongly constrained, satisfying the limit of resources while not degrading objective function is a difficult procedure.*[go into more background of other procedures used]* For instance, Ren et al. [1] used a surrogate resource constraint, based on average resource-usage for all violated resource constraints, to simplify the decision criterion. Based on preliminary results demonstrating the superiority of this procedure compared to other's mentioned above, our procedure is similar.

We first define a surrogate resource constraint:

$$R^*_{ij} = \frac{\sum_{k \in M^v} c_{ijk}/r_k}{|M^v|}$$

M^v is a subset of M, where k is an element of M^v if $\sum\sum c_{ij}x_{ij} > r_k$, for all k of M. Thus M^v contains only those constraints violated. For this procedure, R^* is calculated for each item in each group, and the item which maximizes a change of R^* ,$R^*_{i,\text{selected item}} - R^*_{i,j}$, is swapped with the selected item of that class. Note this contrasts with Ren et al.[1] procedure which chooses a group at random, and iteratively selects item's with lower surrogate constraint from that group. Our procedure calculates the swap which will yield the largest difference, with the hope to regain feasibility faster.

Fig _ : Multidimensional Feasibility Operator

```
Input: Solution, S and maxI, max iterations
    to attempt making S feasible.
Output: S, feasible by MD constraints
MDFeasOp(S,maxI):
SET i to 0
WHILE S is infeasible and i<maxI:
    Find M^v, the set of violated resource constraints
    SET items_r to ∅
    FOR each group in S:
        SET selected_item to the item selected in group
        SET items_r to items_r ⋃ max{$R^*_{group,selected\_item} - R^*_{group,item}$}
    Swap item with max{items_r} for the current selected
        item in group
    SET i to i+1
ENDWHILE
```

Shown by the pseudocode above, this procedure continues until either a feasible solution is found or a maximum number of iterations is completed, stated by the caller as maxI. The maxI value we found worked well (adequate percent of solutions where made feasible in a reasonable amount of computational time) was M, the number of resource constraints. Also shown above M^v, the set of violated resource constraints, is calculated first every iteration, because the violated constraints may change as item's are swapped for a solution.

It is important to note Ren et al. [1] also added an improvement operator, complementary local search, after a feasible solution was found. In the interest of keeping our repair light-weight, we choose not to do so.

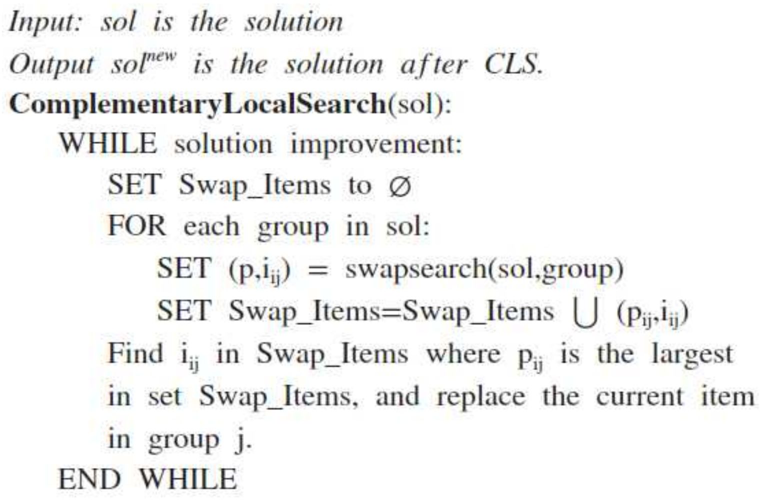
Neighborhood Search Procedure's

The complementary local search and reactive local search procedure used in this study both relate closely to those by Hifi et al. [2]. Complementary local search, in typical Hill Climbing fashion, iteratively improves the solution quality, while not allowing for any degradation. This is achieved using a swapping procedure which swaps an item in a group with another if improvement can be made on the objective function, while not violating feasibility. The Reactive local search is similar to Complementary local search, however allows for negative moves through penalizing solution features.

Complementary Local Search Procedure

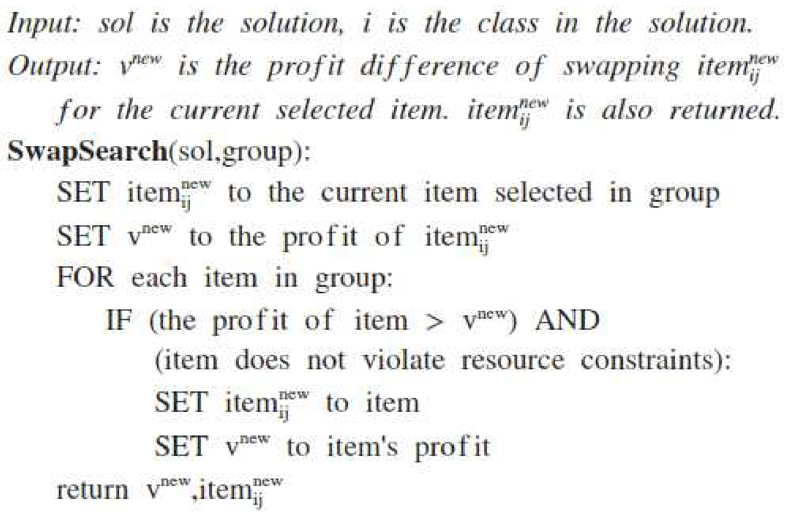
[we will have to talk about how much detail to go into here, as the procedure is published previously, for now I simply have a concise definition].

This procedure uses iterative improvement of the initial feasible solution through two phases. First, a swapping strategy which determines one item in each group which will raise objective function value without violating feasibility. Next, from all groups we find the 'best swap', i.e. the one which raises our objective function the most, and replace the old item of that group with new. This procedure is enumerated below:



```
Input: sol is the solution
Output $sol^{new}$ is the solution after CLS.
ComplementaryLocalSearch(sol):
    WHILE solution improvement:
        SET Swap_Items to $\varnothing$
        FOR each group in sol:
            SET $(p,i_{ij})$ = swapsearch(sol,group)
            SET Swap_Items=Swap_Items $\bigcup$ $(p_{ij},i_{ij})$
        Find $i_{ij}$ in Swap_Items where $p_{ij}$ is the largest
        in set Swap_Items, and replace the current item
        in group j.
    END WHILE
```

where,

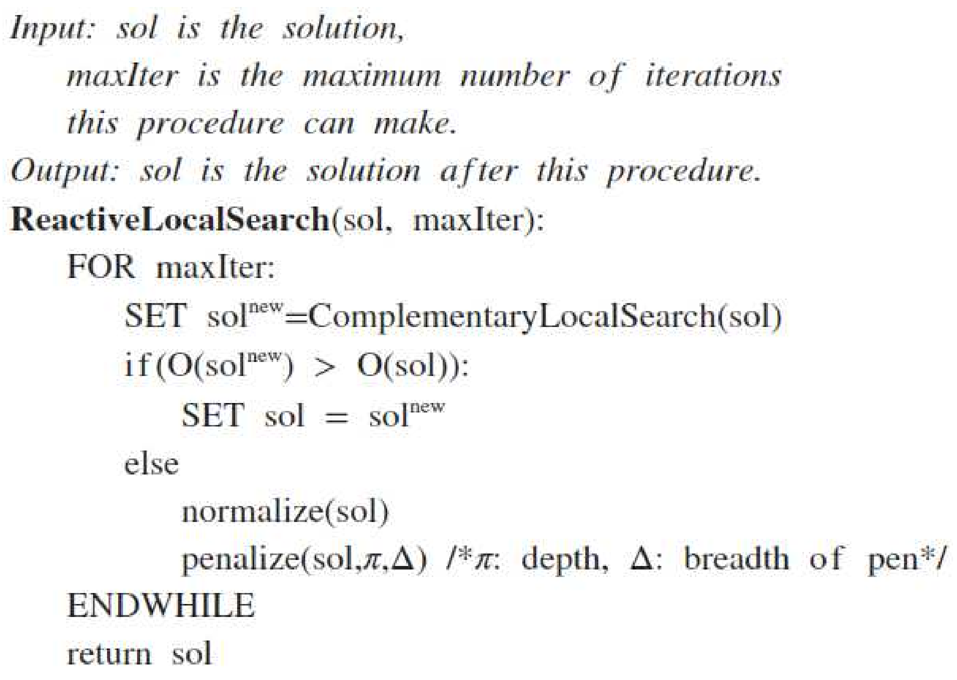


```
Input: sol is the solution, i is the class in the solution.
Output: $v^{new}$ is the profit difference of swapping $item_{ij}^{new}$
    for the current selected item. $item_{ij}^{new}$ is also returned.
SwapSearch(sol,group):
    SET $item_{ij}^{new}$ to the current item selected in group
    SET $v^{new}$ to the profit of $item_{ij}^{new}$
    FOR each item in group:
        IF (the profit of item > $v^{new}$) AND
            (item does not violate resource constraints):
            SET $item_{ij}^{new}$ to item
            SET $v^{new}$ to item's profit
    return $v^{new}$,$item_{ij}^{new}$
```

Reactive Local Search

Procedure

[we will have to talk about how much detail to go into here, as the procedure is published previously, for now I simply have a concise definition].



```
Input: sol is the solution,
    maxIter is the maximum number of iterations
    this procedure can make.
Output: sol is the solution after this procedure.
ReactiveLocalSearch(sol, maxIter):
    FOR maxIter:
        SET $sol^{new}$=ComplementaryLocalSearch(sol)
        if(O($sol^{new}$) > O(sol)):
            SET sol = $sol^{new}$
        else
            normalize(sol)
            penalize(sol,$\pi$,$\Delta$) /*$\pi$: depth, $\Delta$: breadth of pen*/
    ENDWHILE
    return sol
```

Resources

[1] Z. Ren and Z. Feng (2010). An ant colony optimization approach to the multiple-choice multidimensional knapsack problem. *ACM*

[2] M Hifi and A. Sibihi (2004). Heuristic algorithms for the multiple-choice multidimensional knapsack problem. JORS. 0:1-10