0-1 Knapsack Problem（0-1背包问题）

Suppose that there are objects of number n, which has weight w_i and profit p_i. We should take some of them with weight no more than c and make the sum of profit (of objects we take) as great as possible.

We will discuss some variants of (0-1) knapsack problem: the set-union knapsack problem (SUKP), the discounted {0-1} knapsack problem (D{0-1}KP), and the bounded knapsack problem (BKP).

Set-Union Knapsack Problem (SUKP)

- Let U={1, ... ,m} be the set of objects. Object i is with weight w_i.
- Let S={S_1, ... ,S_n}. Each S_i is a subset of U, with profit p_i.
- Choices we can choose is subsets of S. When we take S_{a1}, ... ,S_{ak}, the profit is $p_{a1}+...+p_{ak}$, and the weight is sum of weight of objects in '$S_{a1} \cup ... \cup S_{ak}$'.
- If m=n and S_i={i}, it's a 0-1 knapsack problem.

Discounted {0-1} Knapsack Problem (D{0,1}KP)

- Let $U=\{U_{11}, U_{12}, U_{21}, U_{22}, \ldots, U_{n1}, U_{n2}\}$, in which U_{i1} and U_{i2} are in a group S_i, for $i=1, \ldots, n$. Each U_{ij} is with weight of w_{ij} and profit p_{ij}.
- If we take both U_{i1} and U_{i2}, the profit is still $p_{i1}+p_{i2}$, but the weight is $w_i<w_{i1}+w_{i2}$, which reflect the word 'discouted'.

Bounded Knapsack Problem (BKP)

- Let U={1, ... ,n} be the set of objects. Object i is with weight w_i and profit p_i.
- What is different from {0-1}KP is that the number of an object can be more than one, written b_i. That is to say that we can take i-th object of number b_i at most.
- If b_i=1 (for all i=1, ... n), it's a 0-1 knapsack problem.

Genetic Algorithm（GA，遗传算法）

Genetic algorithm can be used in some optimization problems（优化问题）. When we want to find the optimal solution（最优解） satisfying some limitations, We can follow these steps.

- Preparations: Generate solutions randomly (or in other ways) of number NP. Let these solutions are of generation 0. Let t=0. (t is the generation.)

Genetic Algorithm（GA，遗传算法）

- Step 1: Act the crossover operator（交叉算子） on some items (written x_1, ... , x_a) of generation t, and get item y.
- Step 2: Act the mutation operator（变异算子） on y.
- Step 3: Because the new y may be infeasible, we adjust it to make it feasible. This step usually use a simple algorithm, such as greedy algorithm.

Genetic Algorithm（GA，遗传算法）

- Step 4: Judge whether y is fit enough (for example, whether fitness of y is greater than that of a given item in generation t), and put a proper one in the t+1 generation. t=t+1.
- Repeat steps above so that there is NP items in the generation.
- Make more generation until the fitness of items don't increase. Then we get the optimal solution.

Genetic Algorithm（GA，遗传算法）

The keys of Genetic algorithm are the crossover operator and the mutation operator.

Residue Classes of Module n（模n剩余类）

- Z_n={[0], [1], ... ,[n-1]}, and [n]=[0], [n+1]=[1], ... , [k+n]=[k], for all k∈Z.
- [a]+[b]=[a+b], [a][b]=[ab], and we can proved that Z_n is a group, even a ring.
- example: n=10,

[6]+[7]=[13]=[3], [5]-[9]=[-4]=[6],

[8][9] (=[-2][19]) =[72] (=[-38]) =[2]

Direct Product（直和）

- If $G_1, \ldots, G_n$ are groups, we can define group $G_1 \times \ldots \times G_n = \{(g_1, \ldots, g_n) \mid g_i \in G_i, i=0,1, \ldots, n\}$, in which $(g_1, \ldots, g_n)+(h_1, \ldots, h_n)=(g_1+h_1, \ldots, g_n+h_n)$.
- Moreover, if $G_1, \ldots, G_n$ are rings, then $(g_1, \ldots, g_n)(h_1, \ldots, h_n)=(g_1h_1, \ldots, g_nh_n)$.

Genetic Algorithm in SUKP

- We use a vector of length of n (equals to the size of S), whose components are all 0 or 1. We can regard a vector as an element in $Z_2 \times ... \times Z_2$, and use operators in group or ring on it.
- If the i-th components is 1, it means we take S_i, otherwise it means we don't take S_i.
- Preparations: Generate solutions of number NP randomly. t=0.

Group Theory-based Optimization Algorithm (GTOA) in SUKP

- Crossover Operator: $C(X_1,X_2,X_3)=X_1+F(X_2-X_3)$, in which X_1, X_2 and X_3 are input vectors, and F is a random vector with compents of 1, 0 or -1.
- Mutation Operator: SMO(X). Give a probability p (in (0,1)). For each component in X (written x_i), generate a random number r in (0,1). If $r<p$, make $x_i=1-x_i$, else x_i doesn't change.
- Adjustment: S-GROA (greedy algorithm)

Group Theory-based Optimization Algorithm (GTOA) in SUKP

- Judgement: Suppose that X is in the previous generation and Y is a new item. If the total profit of Y is greater than that of X, put Y into the next generation, otherwise put X into the next generation.

Genetic Algorithm in D{0,1}KP

- We use a vector of length n, similar to that in SUKP, but for which group we have 4 choices, so the vector is in $Z_4 \times ... \times Z_4$ instead of $Z_2 \times ... \times Z_2$.
- The 0,1,2,3 value of the i-th components in the vector separately means taking none (in the i-th group), taking the first one, taking the second one, and taking both of them.
- Preparations: Similar to that in SUKP.

GTOA in D{0,1}KP

- Crossover Operator: $C(X_1,X_2,X_3)=X_1+F(X_2-X_3)$. Or $C(X_1, X_2, X_3, X_4)=X_1+X_2(X_3-X_4)$.
- Mutation Operator: IRMO(X). Give a probability p (in (0,1)). For each component in X (written x_i), generate a random number r in (0,1). If $r<p$, half-probability make $[x_i]=[-x_i]$, another half-probability change $[x_i]$ to a random value (not equals to $[x_i]$). If $r\geq p$, x_i doesn't change.
- Adjustment: D-GROA (greedy algorithm)
- Judgement: Similar to that of SUKP

Genetic Algorithm in BKP

- Suppose that the number of the i-th object is b_i.
- We use a vector of length n in $Z_{b1+1} \times \ldots \times Z_{bn+1}$.
- The i-th component is the number of the i-th object we take.
- Preparations: Similar to that in SUKP.

GTOA in BKP

- Crossover Operator: $C(X_1,X_2,X_3)=X_1+F(X_2-X_3)$.
- Mutation Operator: IRMO(X).
- Adjustment: B-GROA (greedy algorithm)
- Judgement: Similar to that of SUKP