Evolutionary Computation

Software Implementation of Metaheuristics:

Genetic Algorithms
Simulated Annealing
Particle Swarm Optimization

WLPZAC001 Zach Wolpe CSC5023Z

Knapsack Problem

Knapsack is a combinatorial search space problem defined as follows:

- Given a list of items each containing a '*value*' & a '*weight*'
- We can only take a certain amount of items, limited by the total *weight* of the items selected; thus we have a maximum *Capacity* that cannot be exceeded (the maximum what of fulling a knapsack).
- We wish to take a combination of items that *Maximizes* the *Total Value* of the selected items (items in the knapsack).

Whilst seemingly simplistic, the knapsack problem becomes very difficult as the number of items balloons: as the number of combinations grows exponentially as a function of the number of items.

The given Knapsack contains 150 items. Each item can either be

- Selected: 1
- Not Selected: 0

Resulting in:

$$2^{150} = 1.4272477e + 45$$

Permutations.

Consequently randomly search (trying all possibly combinations) is infeasible & would take millions of years to compute. We thus rely on intelligent search algorithms to attempt many possible combinations in a sophisticated way to find an elegant solution.

Though Knapsack is a toy problem, one can easily imagine how this algorithm could be applied to ANY combinatoric search space (& in fact is generalizable to any complex search domain).

Specifications

The given knapsack:

Combinatorial space:	2^150 = 1427247692705960000000000000000000000000000000
Maximum capacity:	822
Maximum iterations:	10'000
Best known optimum:	997

The same problem is solved by 3 algorithms: Genetic Algorithms (GA), Simulated Annealing (SA) & Particle Swarm Optimization (PSO). Full algorithm descriptions are available on the GitHub repository.

Genetic Algorithm (GA)

This GA implementation includes: one-point crossover (1PX); two-point crossover (2PX); roulette-wheel selection (RWS); tournament selection (TS); bit-flip mutation (BFM); exchange mutation (EXM); inverse mutation (IVM); insertion mutation (ISM); displacement mutation (DPM).

Algorithm sudo code (full description available on the git repository):

Algorithm 3.2.1: Pseudocode for the Genetic Algorithm.

```
Input: Population_size, Problem_size, P_crossover, P_mutation
Output: S_best
Population ← InitializePopulation(Population_size,
Problem_size);
EvaluatePopulation(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    Parents ← SelectParents(Population, Population_size);
    Children ← ∅;
    foreach Parent_1, Parent_2 ∈ Parents do
        Child_1, Child_2 ← Crossover(Parent_1, Parent_2, P_crossover);
        Children ← Mutate(Child_1, P_mutation);
        Children ← Mutate(Child_2, P_mutation);
    end
    EvaluatePopulation(Children);
    S_best ← GetBestSolution(Children);
    Population ← Replace(Population, Children);
end
return S_best;
```

Simulated Annealing (SA)

Algorithm sudo code (full description available on the git repository):

Algorithm 4.2.1: Pseudocode for Simulated Annealing.

```
Input: ProblemSize, iterations_max, temp_max
Output: S_best
S_current ← CreateInitialSolution(ProblemSize);
S_best ← S_current;
for i = 1 to iterations_max do
    S_i ← CreateNeighborSolution(S_current);
    temp_curr ← CalculateTemperature(i, temp_max);
    if Cost(S_i) ≤ Cost(S_current) then
        S_current ← S_i;
        if Cost(S_i) ≤ Cost(S_best) then
            S_best ← S_i;
        end
    else if Exp( (Cost(S_current) − Cost(S_i)) / temp_curr ) > Rand() then
        S_current ← S_i;
    end
end
return S_best;
```

Particle Swarm Optimization (PSO)

Algorithm sudo code (full description available on the git repository):

Algorithm 6.2.1: Pseudocode for PSO.

```
Input: ProblemSize, Population_size
Output: P_g_best
Population ← ∅;
P_g_best ← ∅;
for i = 1 to Population_size do
    P_velocity ← RandomVelocity();
    P_position ← RandomPosition(Population_size);
    P_cost ← Cost(P_position);
    P_p_best ← P_position;
    if P_cost ≤ P_g_best then
        P_g_best ← P_p_best;
    end
end
while ¬StopCondition() do
    foreach P ∈ Population do
        P_velocity ← UpdateVelocity(P_velocity, P_g_best, P_p_best);
        P_position ← UpdatePosition(P_position, P_velocity);
        P_cost ← Cost(P_position);
        if P_cost ≤ P_p_best then
            P_p_best ← P_position;
            if P_cost ≤ P_g_best then
                P_g_best ← P_p_best;
            end
        end
    end
end
return P_g_best;
```

Results

The full implementation & results are available at GitHub

Each algorithm was run for a varied spectrum of hyper parameters (defining changes in basic behaviour like the cooling rate of SA or the number of particles in a swarm in PSO). Performance of each specification was measured by '*squality*' or '*Solution Quality*' P which compares the best learnt solution for a given algorithm (maximum value of the knapsack) with best known solution of 997 - computed by random search on a super-computer. Squality gives the best performance as percentage of 997.

Given the limited computational power, an squality >= 0.8 would be considered successful, however all algorithms exceeded this benchmark, with GA & PSO maxing out at squality > 1 (meaning better than the best known solution).

Summary reports of highest performing configurations of each algorithm are below:

```
    Evaluation | 2020-04-22 13:37
    Configuration:  ga_default_25.json

                    GA | #10000 | TS | 2PX (0.003) | EXM (0.7)

==================================================================================

         weight value  squality                                          genotype
0       801   670  0.672016  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, ...
1       801   670  0.672016  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, ...
2       801   670  0.672016  [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, ...
3       804   714  0.716148  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4       804   714  0.716148  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...     ...   ...       ...                                                ...
9995    820  1046  1.049147  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...
9996    820  1046  1.049147  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...
9997    820  1046  1.049147  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...
9998    820  1046  1.049147  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...
9999    820  1046  1.049147  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, ...

[10000 rows x 4 columns]

----------------------------------------------------------------------------------

    [Statistics]

    Runtime             890848.5028 ms

    Convergence

          weight value  squality
2500     799   972  0.974925
5000     822   NaN       NaN
7500     819  1041  1.044132
10000    820  1046  1.049147

    Plateau

    Longest sequence, 30-9999 with improvement less average 3%

    Best Run

            index:      9395
            weight:     820
            value:      1046
            squality:   1.0491474423269809
            genotype:   0000000101...

==================================================================================
```

```
    Evaluation | 2020-04-24 17:11
    Configuration:  pso_default_15.json

                    pso | #1000 | inertia=0.85 | n particles=100 | c1=0.6 | c2=0.5

==================================================================================

        weight value  squality                                          genotype
0      816   625  0.639713  [0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
1      817   429  0.439099  [0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
2      808   686  0.702149  [0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, ...
3      787   467  0.477994  [0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, ...
4      799   421  0.430911  [0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, ...
..     ...   ...       ...                                                ...
995    818  1067  1.092119  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, ...
996    799   554  0.567042  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, ...
997    822  1063  1.088025  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, ...
998    111   163  0.166837  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, ...
999    798   559  0.572160  [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, ...

[1000 rows x 4 columns]

----------------------------------------------------------------------------------

    [Statistics]

    Runtime             1953839.2534 ms

    Convergence

        weight value  squality
250     819   985  1.008188
500     813   507  0.518936
750     798   608  0.622313
1000    798   559  0.572160

    Plateau

    Longest sequence, 566-995 with improvement less average 3%

    Best Run

            index:      995
            weight:     818
            value:      1067
            squality:   1.0921187308085978
            genotype:   0000000101...

==================================================================================
```

```
    Evaluation | 2020-04-23 14:00
    Configuration:  sa_default_01.json

                    sa | #39609 | Initial Temp:10000 | Cooling Rate:0.5

==================================================================================

         weight value  squality                                          genotype
0        802   555  0.568066  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ...
1        818   536  0.548618  [0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, ...
2        791   543  0.555783  [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, ...
3        820   483  0.494371  [0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, ...
4        809   519  0.531218  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...      ...   ...       ...                                                ...
39604    804   545  0.557830  [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, ...
39605    806   491  0.502559  [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, ...
39606    811   522  0.534289  [0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, ...
39607    803   559  0.572160  [1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
39608    790   552  0.564995  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...

[39609 rows x 4 columns]

----------------------------------------------------------------------------------

    [Statistics]

    Runtime             914755.7757 ms

    Convergence

         weight value  squality
9902     798   531  0.543501
19804    795   452  0.462641
29706    807   490  0.501535
39609    790   552  0.564995

    Plateau

    Longest sequence, 8857-39608 with improvement less average 3%

    Best Run

            index:      8857
            weight:     810
            value:      826
            squality:   0.8454452405322416
            genotype:   0000001001...

==================================================================================
```

These truly remarkable algorithms are capable of effective search in inconceivably sparse high dimensional spaces. Note, whilst these applications are applied to discrete problem the same techniques are readily applied to continuous search spaces with minor re-specifications (simple genotype encoding, phenotype encoding & fitness evaluation updates). Thus applications may be extended to any domain that can be mapped to a theoretical mathematical function.

Fin.