An Evolution Program

Objective

Solve the following optimization of the 0/1 Knapsack Problem.

Given

- Weights, W_i
- Profits, P_i
- Capacity, C,

Find a binary vector

$$\vec{x} = \langle x_1, x_2, \dots, x_n \rangle, x_i \in \{0, 1\},$$

which maximizes

$$P(x) = \sum_i^n (x_i \cdot P_i),$$

subject to

$$\sum_i^n (x_i \cdot W_i) \leq C.$$

Test Data

The difficulty of the knapsack problem can be contributed to by the level of correlation of the Profits P_i and Weights W_i.

This project uses a weakly correlated test data set to provide some challenge while still remaining solvable using penalty methods.

For each $i = 1..n$

$$W_i = rand.float(1, v),$$

$$P_i = \max(0, W_i + rand.float(-r, r))$$

Test Data Parameters

Defaults shown.

- Number of items to choose from, $n = 20$
- Upper bound for weight, $v = 10$

- Profit correlation factor, $r = 5$
- Capacity, $C = 40$

Example Menu

```
=================================
        Problems To Solve
=================================
    1 - Binary Knapsack
=================================

=================================
Which problem? [1]:
=================================
Enter Number of Items [20]:
=================================
Enter Maximum Weight of One Item [10.0]:
=================================
Enter Profit Correlation Factor [5.0]:
=================================
Enter Knapsack Capacity [40.0]:

=================================
        Penalty Methods
=================================
    1 - Logarithmic
    2 - Absolute
    3 - Dynamic
=================================

=================================
Which Penalty Method? [1]:
=================================
Enter Rho for Logarithmic Penalty [1.3]:

=================================
        Selection Mechanisms
=================================
    1 - Proportional
    2 - Truncation
    3 - Tournament (Deterministic)
    4 - Tournament (Stochastic)
    5 - Linear Ranking
=================================
```

```
==================================
Which mechanism? [1]: 4
==================================
Enter Probability of Fittest Winner [0.9]:

==================================
        Crossover Methods
==================================
    1 - Single Point
    2 - P-Uniform
    3 - Majority Voting
==================================

==================================
Which Crossover Method? [1]:
==================================
Enter Probability of Crossover [0.65]:
==================================
Population Size [30]:
==================================
Probability of Mutation [0.05]:
==================================
Max Generations [50]:
==================================
Maximize [Y/n]:
==================================
Single run? [y/N]:
==================================
Collect stats? [Y/n]: Y
==================================
Number of Runs [30]:
```

Example Profits | Weights

```
Profits         Weights
[[ 1.41120996  4.66918325]
 [ 0.98976516  1.49829436]
 [ 9.7947617   8.0968139 ]
 [ 8.15973775  3.58574666]
 [ 8.38648779  5.05315528]
 [ 3.8183104   3.73521075]
 [ 7.0378312   5.73759572]
 [ 9.96465678  6.61430992]
 [ 7.46429557  7.99097912]
 [ 9.79907955  7.17617481]
```

```
 [11.19767219  9.82844977]
 [ 5.03030377  6.40734483]
 [ 3.47146217  8.32571668]
 [ 7.79765528  7.37780637]
 [ 0.          1.24781212]
 [ 5.58830843  9.13840496]
 [ 5.53806313  5.04914366]
 [ 0.          2.07032189]
 [ 7.50751635  8.51770158]
 [ 0.2206161   2.82023404]]
```

Example Default Options

```
Options: {
  "Crossover_Method": "<class 'binary_knapsack.crossover_method.single_point.SinglePoint'>",
  "Penalty_Method": "<class 'binary_knapsack.penalty_method.logarithmic.LogarithmicPenalty'>
  "Select_Mechanism": "<class 'binary_knapsack.selection_mechanism.proportional.Proportional
  "crossover_parameters": {
    "p_c": 0.65
  },
  "maximize": true,
  "p_m": 0.05,
  "penalty_parameters": {
    "rho": 1.3
  },
  "pop_size": 30,
  "problem_instance": "Profits\t\tWeights\n[[ 1.41120996  4.66918325]\n [ 0.98976516  1.4982
  "problem_parameters": {
    "capacity": 40.0,
    "dims": 20,
    "max_item_weight": 10.0,
    "profit_correlation_factor": 5.0
  },
  "selection_parameters": {},
  "t_max": 50
}
```

Representation

- Chromosomes (input vector $\vec{x}$) made up of bit-string values, e.g. $\langle 10010101101101100110 \rangle$.
- The i^{th} item is selected if and only if $x_i = 1$.

Evolution Program Parameters

Defaults shown.

- Population size, $N = 300$
- Mutation probability, $p_m = 0.05$
- Maximum generations, $t_{max} = 50$

Penalty Methods

Logarithmic Penalty

I chose this static penalty method because of its ease of use and proven performance.

$$Penalty(x) = log_2(1 + \rho \cdot (\sum_i^n x_i \cdot W_i - C))$$

Parameters for Dynamic Penalty

- Scalar of degree of violation, $\rho = 1.3$

Results

```
Stats over 30 runs:
Best Overall: (Score: 56.96086516860768 :: [0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0] => cost
Mean Best Fitness: 53.06670943662913
Standard Deviation of Best Fitness: 1.771948255995935
Mean Generation Best Was Acheived: 31.5
Standard Deviation of Generations: 13.037765657248688
```

Absolute Penalty

Overage of any constraint leads to a zero fitness.

I chose this static penalty method because of its ease of use and to prove penalization works.

Results I could not get results due to the following error:

```
  File ".\src\binary_knapsack\selection_mechanism\proportional.py", line 49, in _generate_pr
    assert self.sum_of_fitnesses > 0
           ^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
```

The absolute penalty method was not able to acheive feasible results.

Dynamic Penalty

As generations go by, the penalty for the same violation goes up.

$$Penalty(x) = (C * t)^{alpha} * \sum_{j}^{m} f_j(x)^{beta}$$

I chose this dynamic penalty method to test the claims that this method leads to premature convergence.

Parameters for Dynamic Penalty

- Scalar of generation number, $C = 0.5$
- Exponent of temporal harshening, $alpha = 2.0$
- Exponent of degree of violation, $beta = 2.0$

Results

```
Stats over 30 runs:
Best Overall: (Score: 55.5352304177474 :: [0 1 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0] => costi
Mean Best Fitness: 52.74168473724467
Standard Deviation of Best Fitness: 1.3731685279839918
Mean Generation Best Was Acheived: 29.933333333333334
Standard Deviation of Generations: 13.01520478346597
```

This did not perform as well as logarithmic, and also did not acheive feasible results sometimes.

Selection Mechanisms

Proportional Selection

An individual's probability of advancement equal to fitness score over sum of all fitness scores.

$$P(x_i) = f(x_i) / \sum_{j}^{N} f(x_j)$$

Results

```
Stats over 30 runs:
Best Overall: (Score: 56.96086516860768 :: [0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0] => cost
Mean Best Fitness: 53.09012691182008
```

```
Standard Deviation of Best Fitness: 1.4157700470325698
Mean Generation Best Was Acheived: 30.3
Standard Deviation of Generations: 12.607273033187365
```

Truncation Selection

The top $\tau\%$ are selected for advancement. The next generation is sampled uniformly random with replacement from the top $\tau\%$.

Parameters for Truncation Selection

- Top tau, $\tau = 0.4$

Results

```
Stats over 30 runs:
Best Overall: (Score: 56.96086516860768 :: [0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0] => cost
Mean Best Fitness: 51.930860569359396
Standard Deviation of Best Fitness: 5.591743945371317
Mean Generation Best Was Acheived: 26.866666666666667
Standard Deviation of Generations: 16.390512160664443
```

Deterministic Tournament Selection

Pick two individuals from the current population uniformly randomly with replacement.
Advance the chromosome $\vec{x}$ with the better fitness score.
Repeat N times.

Results

```
Stats over 30 runs:
Best Overall: (Score: 55.5352304177474 :: [0 1 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0] => costi
Mean Best Fitness: 49.19210281339055
Standard Deviation of Best Fitness: 8.154572784009153
Mean Generation Best Was Acheived: 24.666666666666668
Standard Deviation of Generations: 17.793881595150122
```

Stochastic Tournament Selection

Same as deterministic tournament selection, except the worse chromosome $\vec{x}$ advances with a typically small random chance.

Parameters for Stochastic Tournament Selection

- Probability of the higher performer winning, $prob = 0.9$

Results

```
Stats over 30 runs:
Best Overall: (Score: 56.96086516860768 :: [0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0] => cost
Mean Best Fitness: 46.870285204235365
Standard Deviation of Best Fitness: 5.924614709499177
Mean Generation Best Was Acheived: 13.533333333333333
Standard Deviation of Generations: 16.036901889773546
```

Linear Ranking Selection

Each chromosome $\vec{x}$ is ranked from best to worst based on fitness score.
An individual's probability of advancement is proportional to its rank.

Parameters for Linear Ranking Selection

- Expected # of copies of best $\vec{x}$, $max = 1.2$

Results

```
Stats over 30 runs:
Best Overall: (Score: 54.89214401149931 :: [0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0] => cost
Mean Best Fitness: 47.97313785058804
Standard Deviation of Best Fitness: 2.9990674396662595
Mean Generation Best Was Acheived: 22.133333333333333
Standard Deviation of Generations: 15.622064168633058
```

Crossover Methods

Single-Point Crossover

From two parents, randomly choose a cut-point and swap sides of each parent to create two children.

Parameters for Single-Point Crossover

- Crossover probability, $p_c = 0.65$

Results

```
Stats over 30 runs:
Best Overall: (Score: 55.5352304177474 :: [0 1 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0] => costi
Mean Best Fitness: 52.81255744849033
Standard Deviation of Best Fitness: 1.328633982424607
Mean Generation Best Was Acheived: 30.333333333333332
Standard Deviation of Generations: 14.25794125702897
```

P-Uniform Crossover

From two parents and for each locus, randomly choose which parent to use at that locus. Choose the better parent with probability p. This process produces two children per set of parents.

Parameters for P-Uniform Crossover

- Crossover probability, $p_c = 0.65$
- Probability that the higher-performing parent donates their allele, $p = 0.5$

Results

```
Stats over 30 runs:
Best Overall: (Score: 56.96086516860768 :: [0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0] => cost
Mean Best Fitness: 53.27474215638246
Standard Deviation of Best Fitness: 1.5440293311694724
Mean Generation Best Was Acheived: 30.466666666666665
Standard Deviation of Generations: 11.315868896770096
```

Majority Voting Crossover

From multiple parents and for each locus, deterministicly choose the most common allele for that locus.

Parameters for Majority Voting Crossover

- Crossover probability, $p_c = 0.65$
- The number of parents to produce one child, $parents = 4$

Results

```
Stats over 30 runs:
Best Overall: (Score: 55.30528934249181 :: [0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0] => cost
Mean Best Fitness: 50.370564380480324
Standard Deviation of Best Fitness: 2.406028580696979
```

```
Mean Generation Best Was Acheived: 31.833333333333332
Standard Deviation of Generations: 14.758236871508586
```

Mutation Methods

Gene-wise Mutation

Each bit (x_i) of chromosome $\vec{x}$ has equal but independent chance (p_m) to mutate a small amount.

Termination Conditions

Simulate until one of the following has been met:

- Simulation reaches t_{max} generations.
- Population fully converges.

Local Installation

From the project root, run `pip install -e src/`.

See repository for full code.

Certification

I certify that this report is my own, independent work and that it does not plagiarize, in part or in full, any other work.

- Austin Hester