Introduction to Evolutionary Computing
COMP 5970-002/6970-003/6976-V04 – Auburn University
Fall 2019 – Assignment Series 1
Evolutionary Algorithms for the Offline Cutting Stock Problem

Daniel R. Tauritz, Ph.D.

October 4, 2019

Synopsis

The goal of this assignment series is for you to become familiarized with (I) representing problems in mathematically precise terms, (II) implementing an Evolutionary Algorithm (EA) to solve a problem, (III) conducting scientific experiments involving EAs, (IV) statistically analyzing experimental results from stochastic algorithms, and (V) writing proper technical reports.

The Cutting Stock Problem [https://en.wikipedia.org/wiki/Cutting_stock_problem], also known as the Offline 2D Bin Packing Problem, is an extremely important real-world industrial problem belonging to the NP-hard complexity class. Real-world applications include aerospace (for instance, at Boeing's Sheet Metal Fabrication Center), shipbuilding, VLSI design, and manufacturing of shoes, clothing and furniture.

While highly efficient heuristics have been developed for several variations of the Cutting Stock Problem, particularly hard variations such as those involving irregular shapes have no known heuristics with low approximation ratios (the ratio between the generated solution quality and the optimum). Also, atypical variations sometimes require custom heuristics. One approach to address this is the use of a meta-heuristic such as an Evolutionary Algorithm to directly solve a given variation. Another approach is the use of a hyper-heuristic to automate the design of a custom heuristic. This assignment series takes the former approach.

These are individual assignments and plagiarism will not be tolerated. You must write your code from scratch in one of the approved programming languages. You are free to use libraries/toolboxes/etc, except search/optimization/EA specific ones.

General implementation requirements

The specific variation of the Cutting Stock Problem tackled in this assignment is as follows:

1. Assume a machine which can produce an arbitrary long sheet of material of a fixed width.

2. A problem instance is specified by a positive integer indicating the fixed width, a positive integer indicating the number of shapes to be cut, and an indexed sequence of shapes which need to be cut from the material stock.

3. Each shape is specified as a finite sequence of grid moves.

4. Each move is specified as a direction (U - up, D - down, L - left, R - right) and a positive integer indicating the number of grid cells in a straight line to be added to the shape, starting from the end of the last move.

5. The objective is to minimize the length of the material needed in order to accommodate all the shapes provided.
6. The solution consists of an indexed sequence of ordered triplets (x, y, r) containing coordinate pairs (x, y) and clockwise 90 degree rotation multipliers (r) valued 0, 1, 2, or 3, where (x, y) indicates the starting grid cell to cut the corresponding indexed shape, under the following constraints:
   (a) shapes cannot go off grid
   (b) shapes cannot overlap
   (c) the length of the material cut cannot exceed the sum of the greatest shape lengths, where greatest shape length is defined as the max of a shape's horizontal and vertical lengths

The problem instance file format consists of a line containing the value of the fixed *width* and the number of shapes, followed by one line per *shape* containing space delimited *moves*. Here is an example:

```
5 2
R1 D1
D1 L4 R1 U3 R3
```

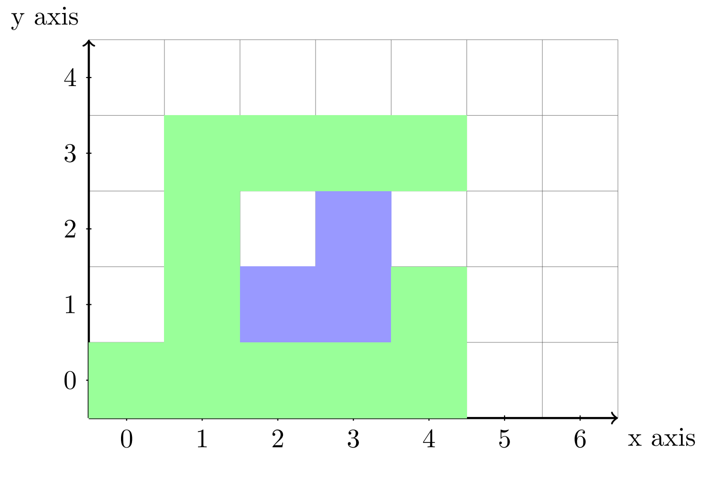
The solution file format consists of lines where each line corresponds to a shape and contains a coordinate pair of the form (x, y) where x is the length offset and y is the width offset, with the origin being bottom left, followed by a 0, 1, 2, or 3, indicating clockwise 90 degree rotation multipliers. Here are two solution file examples with accompanying grid visualization and fitness calculation for the previously given example problem instance:

Solution Example 1

Solution File

```
3,2,1
4,1,0
```

Grid Visualization



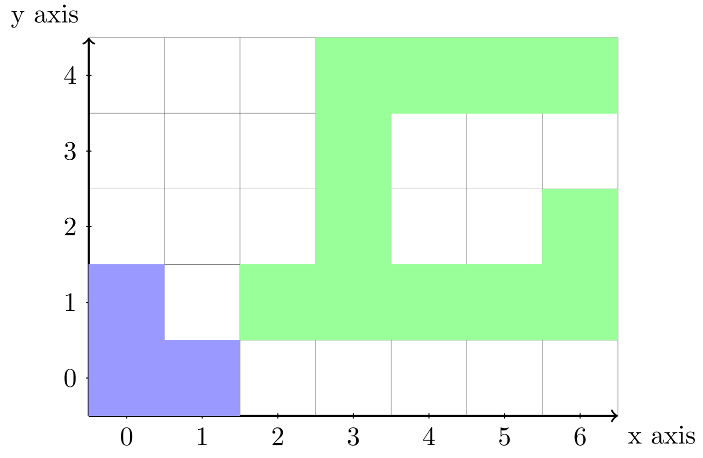
Objective Value Calculation: length used = 5
Example Fitness Calculation: max length - length used = (max (5,4) + max(2,2)) - 5 = (5 + 2) - 5 = 7 - 5 = 2

Solution Example 2

Solution File

```
1,0,2
6,2,0
```

Grid Visualization



Objective Value Calculation: length used = 7
Different Example Fitness Calculation: - length used = -7

At minimum, you will be expected to have the following parameters configurable:

1. An indicator specifying whether the random number generator should be initialized based on time in microseconds or on a specified seed; in case of the latter, the config file should also define the seed for the random number generator (to allow your results to be reproduced)
2. Search algorithm and all black-box search algorithm parameters (e.g., Random Search, EA)
3. The number of runs a single experiment consists of
4. The maximum number of fitness evaluations each run is allotted
5. The relative file path+name of the log file
6. The relative file path+name of the solution file

Your config file should be human-readable with documentation on acceptable configuration values. This documentation may be placed in the README or in comments within the config file itself.

The log file should at minimum include:

1. The path of the problem instance files
2. The random number generator seed
3. The algorithm parameters (enough detail to recreate the config file from the log)
4. An algorithm specific result log (specified in the assignment specific section)

The fitness of a solution must be proportional to the quality of the solution it encodes. Note that fitness per definition correlates higher values with better solutions. In case of a formulation as a minimization problem, you would need to transform the objective value to obtain the fitness value of the corresponding maximization problem.

Solution files should consist of the best solution found in any run.

Version control requirements

For each assignment you will be given a new repository on [`https://classroom.github.com`] **Please view your repository and the README.md** It may clear things up after reading this.

Included in your repository is a script named "finalize.sh", which you will use to indicate which version of your code is the one to be graded. When you are ready to submit your final version, run the command `./finalize.sh` from your local Git directory, then commit and push your code. This will create a text file, "readyToSubmit.txt", that is pre-populated with a known text message for grading purposes. You may commit and push as much as you want, but your submission will be confirmed as "final" if "readyToSubmit.txt" exists and is populated with the text generated by "finalize.sh" at 10:00pm on the due date. If you do not plan to submit before the deadline, then you should NOT run the "finalize.sh" script until your final submission is ready. If you accidentally run "finalize.sh" before you are ready to submit, do not commit or push your repo and delete "readyToSubmit.txt". Once your final submission is ready, run "finalize.sh", commit and push your code, and do not make any further changes to it.

By each assignment deadline, the code currently pushed to Master will be pulled for grading. You are required to include a `run.sh` that needs to be configured to compile and run with the command `./run.sh` using a problem file provided by the TAs and a configuration provided by you, passed in that order through the command line. Specifically, in order to run your submission and grade your output, all the TAs should have to do is execute `./run.sh problem1 config`. The TAs should not have to worry about external dependencies. Any files created by your assignment must be created in the present working directory or subfolders in the present working directory.

Submissions, penalties, documents, and bonuses

You may commit and push to your repository at anytime. At submission time, your latest, pushed, commit to the master branch will be graded (if there is one). In order to ensure that the correct version of your code will be used for grading, after pushing your code, visit [`https://github.com`] and verify that your files are present. If for any reason you submit late, then **please notify the TAs when you have submitted.** If you do submit late, then your first late submission will be graded.

The penalty for late submission is a 5% deduction for the first 24 hour period and a 10% deduction for every additional 24 hour period. So 1 hour late and 23 hours late both result in a 5% deduction. 25 hours late results in a 15% deduction, etc. Not following submission guidelines can be penalized for up to 5%, which may be in addition to regular deduction due to not following the assignment guidelines.

Your code needs to compile/execute as submitted without syntax errors and without runtime errors. Grading will be based on what can be verified to work correctly as well as on the quality of your source code. You must follow the coding requirements as stated in the syllabus.

Documents are required to be in PDF format; you are encouraged to employ LaTeX for typesetting.

All four assignments in this series are weighted equally.

Deliverable Categories

There are three deliverable categories, namely:

a Required for all students in all sections.

b Required for students in the 6000-level sections, bonus for the students in the 5000-level section.

c Bonus for all students in all sections.

Note that the max grade for the average of all assignments in Assignment Series 1, including bonus points, is capped at 100%.

Assignment 1a: Random Search

Implement a random search algorithm to optimize the cut pattern for each of the problem instances provided.

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run *i*" where *i* indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><fitness> (not including the < and > symbols) where <evals> indicates the number of evals executed so far and <fitness> is the value of the fitness function at that number of evals. The first row has 1 as value for <evals>. Rows are only added if they improve on the best fitness value found so far that run. The solution file should consist of the best solution found during any run and be in the specified format. Log files and solution files should be placed in their respective directories, *logs* and *solutions*.

The deliverables of this assignment are:

a1 your source code, configured to compile and run with '*./run.sh <problem1-filepath> <configuration-filepath>*' (including any necessary support files such as makefiles, project files, etc.).

a2 for each problem instance, a configuration file configured for 1000 fitness evaluations, timer initialized random seed, and 30 runs, along with the corresponding log file and the corresponding solution file (these should go in the repo's *logs* and *solutions* directories).

a3 a document headed by your name, AU E-mail address, and the string "COMP x97y Fall 2019 Assignment 1a", where x and y need to reflect the section you are enrolled in, containing for each provided problem instance, the evals versus fitness plot corresponding to your log file which produced the best solution.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitHub, by *pushing* your latest commit to the *master* branch. The due date for this assignment is 10:00 PM on Sunday September 1, 2019.

Grading
The point distribution is as follows:

Algorithmic	50%
Configuration files and parsing	15%
Logging and output/solution files	15%
Good programming practices	10%
Document containing evals versus fitness plots	10%

Assignment 1b: Evolutionary Algorithm Search

Implement a $(\mu + \lambda)$-EA with your choice of representation and associated variation operators, including at minimum one unary (i.e., mutation) and one multi-ary variation operator (i.e., recombination), to find the valid solution which optimizes the cut pattern for each of the problem instances provided. To avoid invalid solutions from entering the population, implement the following base approach:

- Initialize the start population with random valid solutions (randomly generated each run).
- When recombining, check each gene selected for inheritance and if it causes invalidity, replace with a randomly generated allele which does not.
- When mutating a gene, check the new mutated allele and if it causes invalidity, generate a new mutated allele; repeat until the new mutated allele does not cause invalidity.

You need to implement support for all of the following EA configurations, where operators with multiple options are comma separated:

Initialization Uniform Random

Parent Selection Fitness Proportional Selection, k-Tournament Selection with replacement

Survival Selection Truncation, k-Tournament Selection without replacement

Termination Number of evals, no change in average population fitness for n generations, no change in best fitness in population for n generations

Your configuration file should allow you to select which of these configurations to use. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ
- λ
- tournament size for parent selection
- tournament size for survival selection
- mutation rate
- number of evals till termination
- n for termination convergence criterion

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><average fitness><tab><best fitness> (not including the < and > symbols) where <evals> indicates the number of evals executed so far, <average fitness> is the average population fitness at that number of evals, and <best fitness> is the fitness of the best individual in the population at that number of evals (so local best, not global best!). The first row has $< \mu >$ as value for <evals>. Rows are added after each generation, so after each λ evaluations. The solution file should consist of the best solution found in any run.

The deliverables of this assignment are:

a1 your source code (including any necessary support files such as makefiles, project files, etc.)

a2 for each of the three provided problem instances, a configuration file configured for 10,000 fitness evaluations, a timer initialized random seed, 30 experimental runs, and the best EA configuration you can find, along with the corresponding log and solution files,

a3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x97y Fall 2019 Assignment 1b", where x and y need to reflect the section you are enrolled in, containing:

- for each problem instance, a plot which simultaneously shows evals versus average local fitness and evals versus local best fitness, averaged over all runs; box plots are preferred,
- your statistical analysis for both experiments comparing the average final best random search result (note that in Assignment 1a you plotted the best run results as opposed to the average results) versus the average final best result you obtained in Assignment 1b (report said averages along with standard deviation, describe the test statistic you employed, and the appropriate analysis results for said test statistic).

b1 This deliverable is concerned with creating significantly different multi-ary and/or unary variation operators, and comparing the different combinations of variation operators, showing the results in tables and figures, as well as providing accompanying statistical analysis. Of particular interest is under which circumstances a particular combination of variation operators outperforms another combination, while underperforming on another. Statistical analysis followed by a brief discussion is needed. You also need to explain the design of your variation operators. Note that the provided three datasets may be insufficient to show the differences; therefore, we will also provide you with a problem instance generator to test many different kinds of problem instances.
This investigation needs to be documented in a separate section of the required document marked as "Investigation of Variation Operators". You also need to indicate in your source files any code which pertains to this investigation and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this investigation, and which doesn't. Students enrolled in COMP 5970 will earn up to 15% bonus for the multi-ary part of this investigation, up to 15% for the unary part of this investigation, or up to 25% for both. Students enrolled in COMP 6970/6976 must do both and this will count for 25% regular points (not bonus) beyond the A deliverables.

c1 Up to 15% bonus points can be earned by implementing a true repair function in addition to the specified approach for generating valid solutions. You need to compare your repair function with the base approach, showing the results in tables and figures, as well as providing accompanying statistical analysis. Of particular interest is under which circumstances your repair function outperforms the base approach, while underperforming on another. Statistical analysis followed by a brief discussion is needed. Either way, you need to explain the design of your repair function. Note that the provided three datasets may be insufficient to show the differences; therefore, we will also provide you with a problem instance generator to test many different kinds of problem instances.
This bonus investigation needs to be documented in a separate section of the required document marked as "True Repair Function". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitHub, by *pushing* your latest commit to the *master* branch. The due date for this assignment is 10:00 PM on Sunday September 15, 2019.

Grading

The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	A	B	C
Algorithmic	50%	35%	35%
Configuration files and parsing	10%	5%	5%
Logging and output/solution files	10%	0%	0%
Good programming practices & robustness	10%	10%	10%
Writing	10%	25%	25%
Statistical analysis	10%	25%	25%

Assignment 1c: Constraint Satisfaction and Self-Adaptation

Implement a self-adaptive EA with penalty function and your choice of representation and associated variation operators, including at minimum one unary (i.e., mutation) and one multi-ary variation operator (i.e., recombination), to find the valid solution which optimizes the cut pattern for each of the problem instances provided. Make the constraint satisfaction approach user-configurable, including the simple approach from the A deliverable from Assignment 1b, and a penalty function approach. Also, select at least one appropriate EA strategy parameter not related to the penalty function, to add a self-adaptive option for, with a flag in your configuration file to enable or disable.
You need to implement support for at least all of the following EA configurations, where operators with multiple options are comma separated:

Initialization Uniform Random

Parent Selection Uniform Random, Fitness Proportional Selection, k-Tournament Selection with replacement

Survival Strategy Plus, Comma

Survival Selection Uniform Random, Truncation, Fitness Proportional Selection, k-Tournament Selection without replacement

Termination Number of evals, no change in average population fitness for n generations, no change in best fitness in population for n generations

Your configuration file should allow you to select which of these configurations to use. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ
- λ
- penalty coefficient
- tournament size for parent selection
- tournament size for survival selection
- mutation rate
- number of evals till termination
- n for termination convergence criterion

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><self-adaptive value><tab><average fitness><tab><best fitness> (not including the < and > symbols) where <evals> indicate the number of evals executed so far, <self-adaptive value> is the value of the self-adaptive strategy parameter, <average fitness> is the average population fitness at that number of evals, and <best fitness> is the fitness of the best individual in the population at that number of evals (so local best, not global best!). The first row has $< \mu >$ as value for <evals>. Rows are added after each generation, so after each λ evaluations. The solution file should consist of the best solution found in any run.

The deliverables of this assignment are:

a1 your source code (including any necessary support files such as makefiles, project files, etc.)

a2 for each of the three provided problem instances, a configuration file configured for 10,000 fitness evaluations, a timer initialized random seed, 30 experimental runs, and the best EA configuration you can find, along with the corresponding log and solution files,

a3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x97y Fall 2019 Assignment 1c", where x and y need to reflect the section you are enrolled in', containing:

- A detailed description of your self-adaptive EA strategy parameter.
- Descriptions of experiments, including graphs (box plots preferred), result tables, statistical analysis, and justification of EA configuration, to investigate the following:
    1. For each of the three datasets provided, does the self-adaptivity significantly increase performance? Explain why self-adaptivity either provides the same benefit or a differing benefit between the three datasets provided.
    2. For each of the three datasets provided, to what extent does the penalty coefficient impact performance? Also, to what extent is there a correlation between a specific dataset and the choice of penalty coefficient?
- The configuration files needed to recreate the results reported in your document; clearly mark them as to which configuration file goes with which reported experiment and document in the README.md file exactly how to use the configuration files (in addition, self-documenting configuration files are strongly encouraged).
- Corresponding log files (training and test) and solution files.

b1 Investigate and report on implementing self-adaptivity for a second appropriate EA strategy parameter, also not related to the penalty function. Examine both the separate impacts of the two self-adaptive parameters and the combined impact. This investigation needs to be documented in a separate section of the required document entitled "Comparison of self-adaptive strategy parameters". Students enrolled in COMP 5970 can earn up to 15% bonus for this investigation. For students enrolled in COMP 6970/6976 this will count for 15% regular points (not bonus) beyond the A deliverables.

c1 Okay, so you know you want to do it: play with making the penalty coefficient self-adaptive. Here's your chance to do so and get credit for it too! Up to 10% bonus points can be earned by investigating and reporting on making the penalty coefficient self-adaptive.
This bonus investigation needs to be documented in a separate section of the required document marked as "Self-adaptive Penalty Coefficient". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

c2 Up to 10% bonus points can be earned by investigating self-adaptivity of a significantly different multi-ary or unary variation operator versus the base requirement, and comparing the different self-adaptive variation operator combinations, showing the results in tables and figures, as well as providing accompanying statistical analysis. Of particular interest is under which circumstances a particular combination of (self-adaptive) variation operators outperforms another combination, while underperforming on another. Statistical analysis followed by a brief discussion is needed. Either way, you need to explain the design of your self-adaptive variation operators. Note that the provided three datasets may be insufficient to show the differences; therefore, a problem instance generator shall be provided to test many different kinds of problem instances.
This bonus investigation needs to be documented in a separate section of the required document marked as "Comparison of Self-Adaptive Variation Operators". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

c3 If you did not previously try Assignment 1b, Deliverable c1 (True Repair Function), then you may still do so, for up to 15% bonus points.

c4 Up to 10% bonus points can be earned by empirically investigating when for this Cutting Stock Problem it is better to employ a true repair function and when it is better to employ a penalty function, showing the results in tables and figures, as well as providing accompanying statistical analysis. Note that the provided three datasets may be insufficient to show the differences; therefore, a problem instance generator shall be provided to test many different kinds of problem instances.
This bonus investigation needs to be documented in a separate section of the required document marked as "Comparison of Penalty & Repair Functions". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitHub, by *pushing* your latest commit to the *master* branch. The due date for this assignment is 10:00 PM on Sunday October 6, 2019.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	A	B	C
Algorithmic	50%	35%	35%
Configuration files and parsing	10%	5%	5%
Logging and output/solution files	10%	0%	0%
Good programming practices & robustness	10%	10%	10%
Writing	10%	25%	25%
Statistical analysis	10%	25%	25%

Assignment 1d: Multi-Objective EA

Often when solving real-world problems, there are several conflicting objectives and the purpose of optimization is to provide a Pareto optimal trade-off surface to the decision maker. For the problem of cutting stock, there can, for instance, be a trade-off between minimizing the length of the material needed, and minimizing stock width. This is because in many industries, stock contains imperfections (often in the form of runs) that need to be cut off, which is facilitated by smaller widths.

In this assignment you need to implement a Pareto-front-based multi-objective EA (MOEA) with your choice of representation and associated variation operators, including at minimum one unary (i.e., mutation) and one multi-ary variation operator (i.e., recombination), to find within a configurable number of fitness evaluations, the Pareto optimal front representing the searched for trade-off surface, for the objectives of (1) minimizing stock length (the same objective as in 1a and 1b), and (2) minimizing stock width. Note that you need to employ reasonable MOEA operator and parameter choices in your experiments. Note that this is not a constraint satisfation EA such as in 1c.
You need at minimum to implement support for all of the following EA configurations, where operators with multiple options are comma separated:

Initialization/Distribution Uniform Random

Initialization/Seeding With or without seeding (i.e., with seeding means that a user-defined set of solutions in the initial population are seeded with the rest of the initial population created per the Initialization/Distribution option)

Parent Selection Uniform Random, Fitness Proportional Selection, k-Tournament Selection with replacement (where in the latter two, fitness is determined by level of non-domination)

Survival Strategy Plus, Comma

Survival Selection Uniform Random, Truncation, Fitness Proportional Selection, k-Tournament Selection without replacement (where in the latter three, fitness is determined by level of non-domination)

Termination Number of evals, no change in top non-dominated level of population for n generations

Your configuration file should allow you to select which of these configurations to use. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ
- λ
- tournament size for parent selection
- tournament size for survival selection
- Number of evals till termination
- n for termination convergence criterion

The result log should be headed by the label "Result Log" and consist of two sections, the first containing the requirements specified above in the general implementation requirements section, and the second containing empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form:
<evals><tab><average first objective subfitness><tab><best first objective subfitness>
<tab><average second objective subfitness><tab><best second objective fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far, <average subfitness> is the average population subfitness at that number of evals, <best subfitness> is the best population subfitness at that number of evals (so local best, not global best!), the first objective is to minimize stock length, and the second objective is to minimize stock width (don't forget to suitably transform these minimization objectives into subfitness functions which are being maximized).

The first row has $< \mu >$ as value for <evals>. Rows are added after each generation, so after each λ evaluations. The solution file should consist of the best Pareto front found in any run, where we count Pareto front $P1$ as better than Pareto front $P2$ if the proportion of solutions in $P1$ which dominate at least one solution in $P2$ is larger than the proportion of solutions in $P2$ which dominate at least one solution in $P1$. The solution file should be formatted as follows:
the first line should list the number of solutions in the Pareto front, followed by blank-line-separated solution descriptions. Individual solutions are formatted identically to previous assignments.
The deliverables of this assignment are:

a1 your source code (including any necessary support files such as makefiles, project files, etc.),

a2 for each of the three provided problem instances, a configuration file configured for 10,000 fitness evaluations, a timer initialized random seed, 30 experimental runs, and the best MOEA configuration you can find, along with the corresponding log and solution files,

a3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x97y Fall 2019 Assignment 1d", where x and y need to reflect the section you are enrolled in', containing:

- A detailed description of your MOEA.

- Detailed descriptions of experiments, including graphs (box plots preferred), result tables, statistical analysis, and justification of EA configuration, to investigate for each of the three datasets provided, and for at minimum three diverse operator and strategy parameter configurations (diverse both in the sense that the configurations are significantly different, and that those differences cause diverse levels of performance), the trade-off between the two specified objectives, both in terms of the best Pareto front obtained employing the definition of best given above in the result log specification, and the diversity of the obtained Pareto fronts (i.e., are the generated solutions evenly distributed, or clustered) employing your choice of a reasonable diversity measure. Also, for each of the three datasets provided, to what extent is there a correlation between a specific dataset and the choice of strategy parameters?
- The configuration files needed to recreate the results reported in your document; clearly mark them as to which configuration file goes with which reported experiment and document in the README.md file exactly how to use the configuration files (in addition, self-documenting configuration files are strongly encouraged).
- Corresponding log files and solution files.

b1 Add options for both fitness sharing and crowding, and investigating and reporting on how their addition affects the performance of your MOEA, including a three-way comparison between the main MOEA, that MOEA with fitness sharing, and that MOEA with crowding. This investigation needs to be documented in a separate section of the required document entitled "Fitness Sharing and Crowding". Students enrolled in COMP 5970 can earn up to 15% bonus for this investigation. For students enrolled in COMP 6970/6976 this will count for 15% regular points (not bonus) beyond the A deliverables.

c1 Up to 10% bonus points can be earned by adding a third objective which conflicts with the two objectives of the main assignment; you do not need to map it to a real-world scenario. You do need to investigate and report on how your MOEA's behavior and performance are impacted by having three conflicting objectives rather than just two. This bonus investigation needs to be documented in a separate section of the required document marked as "Triple-Objective Investigation". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not.

c2 Up to 10% bonus points can be earned by adding an option to your configuration to replace the width minimization objective with a cut minimization objective, detailing your cut minimization approach, and investigating and reporting on the impact on your MOEA's behavior and performance by comparing the impact on the length objective and the behavior of your EA between the width objective and the cut objective. Design an appropriate comparison which provides statistical analysis. Minimize the number of straight-line cuts across the stock required to cut out the set of shapes. Any sequence of shape-edges which form a straight line that does not pass through the middle of another shape constitutes one straight-line cut. Edges can be connected by a straight-line cut across empty space. There are several real-world applications to this, including but not limited to glass cutting, where minimizing the amount of stock cutting is important in order to minimize breakage/defects in the resulting shapes. This bonus investigation needs to be documented in a separate section of the required document marked as "Cut Minimization". You also need to indicate in your source files any code which pertains to this bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TAs to see with a glance what part of your submission pertains to this bonus, and which does not. The cut minimization objective for this deliverable can not also be used as the third objective for deliverable c1.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitHub, by *pushing* your latest commit to the *master* branch. The due date for this assignment is 10:00 PM on Sunday October 20, 2019.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	A	B	C
Algorithmic	50%	35%	35%
Configuration files and parsing	10%	5%	5%
Logging and output/solution files	10%	0%	0%
Good programming practices & robustness	10%	10%	10%
Writing	10%	25%	25%
Statistical analysis	10%	25%	25%