Introduction to Evolutionary Computing
COMP 5660-001/6660-001/6666-V01 – Auburn University
Fall 2020 – Assignment Series 1
Evolutionary Algorithms for Solving NP-Complete Light Up Puzzle

Daniel Tauritz, Ph.D.

August 16, 2020

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 discipline appropriate technical reports. The problem that you will be solving, instances of the puzzle Light Up (also known as Akari), is a binary-determination logic puzzle belonging to the NP-Complete complexity class. Many important real-world problems are NP-Complete and can be reduced to other NP-Complete problems; therefore, being able to solve a particular NP-Complete problem technically means being able to solve all NP-Complete problems. 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.

Problem statement

Quoted from Wikipedia [`http://en.wikipedia.org/wiki/Light_Up_(puzzle)`]:
"Light Up is played on a rectangular grid of white and black cells. The player places light bulbs in white cells such that no two bulbs shine on each other, until the entire grid is lit up. A bulb sends rays of light horizontally and vertically, illuminating its entire row and column unless its light is blocked by a black cell. A black cell may have a number on it from 0 to 4, indicating how many bulbs must be placed adjacent to its four sides; for example, a cell with a 4 must have four bulbs around it, one on each side, and a cell with a 0 cannot have a bulb next to any of its sides. An unnumbered black cell may have any number of light bulbs adjacent to it, or none. Bulbs placed diagonally adjacent to a numbered cell do not contribute to the bulb count."

In this assignment you need to solve Light Up puzzles of arbitrary size, either specified in the prescribed data file format, or randomly generated.

Data File Format

Your programs need to be able to read in arbitrary size Light Up puzzles specified in the following format:

```
x
y
x1 y1 z1
```

```
x2 y2 z2
...
x<n> y<n> z<n>
```

where x is the number of columns, y is the number of rows, $x < i >$ and $y < i >$ are the coordinates of the black cells, and $z < i >$ specifies the number of bulbs which must be placed adjacent to its four sides with 5 indicating the absence of a number.

Example Problem Instance

The Akari tutorial puzzle at `http://www.nikoli.co.jp/en/puzzles/akari.html` can be encoded as follows:

```
7
7
1 1 1
1 2 5
2 4 5
2 6 4
3 2 5
3 7 5
4 3 5
4 5 2
5 1 1
5 6 1
6 2 1
6 4 5
7 6 5
7 7 5
```

Note from this example that the origin is located in the bottom left of the puzzle files, and is indexed at (1,1).

General implementation requirements

Your program needs to accept a problem instance file using the previously specified format and a configuration file as input. 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
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 (needed for stochastic algorithms to achieve statistically relevant results)
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 relative path+name of the problem instance file
2. The relative path+name of the solution file generated by the experiment
3. The random number generator seed used in the experiment (for experiment reproduction)
4. The algorithm parameters (enough detail to recreate the config file from the log)
5. An algorithm specific result log (specified in each assignment-specific section)

The solution file generated should have the exact following format: the Light Up puzzle being solved in the previously specified format followed by a line stating the quality of the solution by providing the number of white cells lit up and each subsequent line should list a single coordinate pair for a bulb, consisting from left to right of an integer indicating the column, a space, and an integer indicating the row.

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 of an experiment.

Example Problem Solution

The Akari sample puzzle solution at `http://www.nikoli.co.jp/en/puzzles/akari.html` is encoded as follows:

```
7
7
1 1 1
1 2 5
2 4 5
2 6 4
3 2 5
3 7 5
4 3 5
4 5 2
5 1 1
5 6 1
6 2 1
6 4 5
7 6 5
7 7 5
35
1 6
2 1
2 5
2 7
3 6
4 2
4 4
5 5
6 1
6 7
7 3
```

Version control requirements

For each assignment you will be given a new repository on [`https://classroom.github.com`] pre-populated with assignment-specific problem instances. **Please view your repository and the README.md**, it may answer questions you have after reading this section.

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` script that must 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 on the AU Tux machines and grade your output, all the TAs should have to do is execute `./run.sh problem1 config`. More information about the Tux machines can be found at [`https://www.eng.auburn.edu/admin/ens/helpdesk/off-campus/access-information.html`]. 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 before the deadline. 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 and please keep in mind that your submission will undergo code review.

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:

GREEN Required for all students in all sections.

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

RED 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 which generates uniform random placement for a uniform random number of bulbs between 1 and the number of white cells, to find the valid solution which maximizes the number of white cells which are lit up where a cell containing a bulb is also considered lit. Note that this means that you cannot use problem specific knowledge to shrink the search space, for instance by automatically placing bulbs on all four sides of a black cell containing a 4. Valid solutions are those where no two bulbs shine on each other and the number of adjacent bulbs to a black cell containing a number is obeyed. Invalid solutions have a fitness of zero. Invalid solutions do count towards the total number of fitness evaluations per run. Your configuration file should allow you to specify how many runs a single experiment consists of and how many fitness evaluations each run is alotted.

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) with <evals> indicating 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.

Because random search is expected to perform very poorly for larger, more complex puzzles, you should specify a user parameter in the configuration file which controls whether the black cell number constraint is enforced or not (i.e., required for a solution to be counted as valid) and use that parameter to configure your experiments to not enforce that particular constraint. This should allow random search to find some valid solutions. Also note that your program still needs to be able to enforce the black cell number constraint if the user specifies that.

The deliverables of this assignment are:

GREEN 1 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.).

GREEN 2 for each problem instance, a configuration file configured for 10,000 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).

GREEN 3 a document headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 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 August 30, 2020.

Grading

The point distribution per deliverable category is as follows:

Algorithmic	45%
Configuration files and parsing	20%
Logging and output/solution files	15%
Good programming practices including code reliability and commenting	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, which evolves placements of bulbs on white cells, but no more than one bulb in a white cell, to find the valid solution which maximizes the number of white cells which are lit up where a cell containing a bulb is also considered lit. Valid solutions are those where no two bulbs shine on each other and the number of adjacent bulbs to a black cell containing a number is obeyed. Invalid solutions have a fitness of zero. Invalid solutions do count towards the total number of fitness evaluations per run. Your configuration file should allow you to specify how many fitness evaluations each run is alotted.

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

Initialization Uniform Random, Validity Forced plus Uniform Random (Validity Forced shrinks the search space by automatically placing bulbs on the indicated number of sides of a black cell when there is only one unique way to do this.)

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 fitness for n evals

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 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) with <evals> indicating 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.

Because a non-constraint solving EA such as the one in Assignment 1b is expected to perform poorly for larger, more complex puzzles, you must for this assignment specify a user parameter in the configuration file which controls whether the black cell number constraint is enforced or not (i.e., required for a solution to be counted as valid) and use that parameter to configure your experiments to not enforce that particular constraint. This should allow your EA to find valid solutions. Note that for this assignment this is required, not optional. Also note that your program still needs to be able to enforce the black cell number constraint if the user specifies that.

The deliverables of this assignment are:

GREEN 1 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.)

GREEN 2 for each of the 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 (these should go in the repo's *logs* and *solutions* directories)

GREEN 3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 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).

YELLOW 1 Up to 10% (bonus points for COMP 5660 students) for implementing Stochastic Uniform Sampling parent selection method, explaining how your implementation enforces uniform sampling, and comparing Stochastic Uniform Sampling parent selection against another parent selection method of your choice using appropriate statistical analysis and a brief explanation of your findings.

RED 1 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 can earn up to 15% for this investigation.

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 13, 2020.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	GREEN	YELLOW	RED
Algorithmic	45%	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	15%	25%	25%

Assignment 1c: Constraint-Satisfaction EA

Implement a constraint-satisfaction EA with your choice of representation and associated variation operators, which evolves placements of bulbs on white cells, but no more than one bulb in a white cell, to find the valid solution which maximizes the number of white cells which are lit up where a cell containing a bulb is also considered lit. Valid solutions are those where no two bulbs shine on each other and the number of adjacent bulbs to a black cell containing a number is obeyed. Invalid solutions have their fitness penalized by applying a penalty function where the size of the penalty corresponds to the extent that a solution violates validity (i.e., the more bulbs that shine on each other, the more they are penalized; the same for the amount of black cell number violations). Invalid solutions count towards the total number of fitness evaluations per run. Your configuration file should allow you to specify how many fitness evaluations each run is allotted.

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

Fitness Function Original Problem Statement Fitness Function, Constraint Satisfaction Problem Statement Fitness Function

Initialization Uniform Random, Validity Forced plus Uniform Random (Validity Forced shrinks the search space by automatically placing bulbs on the indicated number of sides of a black cell when there is only one unique way to do this.)

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

Survival Strategy Plus, Comma (AKA a generational EA)

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

Termination Number of evals, no change in fitness for n evals

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
- penalty function coefficient

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) with <evals> indicating 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.

Because a non-constraint solving EA such as the one in Assignment 1b is expected to perform poorly for larger, more complex puzzles, you must for this assignment specify a user parameter in the configuration file which controls whether the black cell number constraint is enforced or not (i.e., required for a solution to be counted as valid). The deliverables of this assignment are:

GREEN 1 your source code, with the added EA configuration options, configured to compile and run with '*./run.sh <problem1-filepath> <configuration-filepath>*' (including any necessary support files such as makefiles, project files, etc.)

GREEN 2 for each of the provided problem instances, a configuration file configured for 10,000 fitness evaluations, a timer initialized random seed, 30 experimental runs, black cell number constraint enforced, and the best EA configuration you can find, along with the corresponding log and solution files (these should go in the repo's *logs* and *solutions* directories),

GREEN 3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 Assignment 1c", where x and y need to reflect the section you are enrolled in', containing:

- Descriptions of experiments, including graphs (box plots preferred), result tables, statistical analysis, and justification of EA configuration, to investigate the following:
  1. How much improvement does a constraint-satisfaction EA employing a penalty function obtain versus a plain-vanilla EA? Note that vanilla fitness may not be directly compared with penalized fitness.
  2. How significant is the impact of Validity Forced plus Uniform Random initialization versus plain Uniform Random initialization for both a plain-vanilla EA and a constraint-satisfaction EA? Explain your findings!
  3. How is the penalty function coefficient correlated with the solution quality of your constraint-satisfaction EA? Explain your findings!
- 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 in the appropriate repo directories.

YELLOW 1 Up to 15% (bonus for COMP 5660 students) for implementing self-adaptivity of mutation rate with a corresponding investigation of performance. You need to compare your self-adaptive EA with the base approach (with or without constraint satisfaction so long as you are consistent), showing the results in tables and figures, as well as providing accompanying statistical analysis.

RED 1 Up to 15% bonus points can be earned by implementing a repair function in addition to the specified approach for generating valid solutions. You need to compare your repair function with the base constraint-satisfaction 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 constraint-satisfaction 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.
This bonus investigation needs to be documented in a separate section of the required document marked as "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.

RED 2 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.

RED 3 Up to 10% bonus points for implementing an adaptive mutation rate (using a heuristic of your choice) and investigating performance differences compared to a self-adaptive mutation rate (with or without constraint satisfaction so long as you are consistent). Report on your implementation and investigation, employing the appropriate statistical tests, and explain your findings.

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 27, 2020.

Grading
The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	GREEN	YELLOW	RED
Algorithmic	50%	35%	35%
Configuration files and parsing	5%	5%	5%
Logging and output/solution files	10%	0%	0%
Good programming practices & robustness	10%	10%	10%
Writing	10%	25%	25%
Statistical analysis	15%	25%	25%

Assignment 1d: Multi-Objective EA

The problem statement as employed so far, specifies as objective to maximize the number of cells lit up, while meeting certain hard constraints, and without regard for the number of bulbs placed. However, more realistically the problem probably would be stated to maximize the number of cells lit up while minimizing the number of constraint violations. This requires a tradeoff between these three objectives (number of lit cells, number of black cell constraint violations, and the number of bulbs placed in lit cells). Therefore, in this assignment you need to implement a Pareto-front-based multi-objective EA (MOEA), such as simplified NSGA-II without crowding, to find within a configurable number of fitness evaluations the Pareto optimal front for the trade-off between these objectives with your choice of representation and associated variation operators, which evolves placements of bulbs on white cells, but no more than one bulb in a white cell. Your configuration file should allow you to specify how many fitness evaluations each run is alotted.

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

Initialization Uniform Random, Validity Forced plus Uniform Random (Validity Forced shrinks the search space by automatically placing bulbs on the indicated number of sides of a black cell when there is only one unique way to do this.)

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 evals

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 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><average second objective subfitness><best second objective fitness><average third objective subfitness><best third 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 subfitness of the best individual in the population at that number of evals (so local best, not global best!), the first objective is to maximize the number of cells lit up, the second objective is to minimize the number of bulbs shining on each other, and the third objective is to minimize the number of black cell adjacency constraint violations.

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: each solution in the Pareto front should have its own tab delimited row in the form:
<first subfitness><tab><second subfitness><tab><third subfitness><tab><number of solutions in the Pareto front>
(not including the < and > symbols) followed by one row each per bulb, consisting from left to right of an integer indicating the column, a space, and an integer indicating the row.
The deliverables of this assignment are:

GREEN 1 your source code, with the added EA configuration options, configured to compile and run with '*./run.sh <problem1-filepath> <configuration-filepath>*' (including any necessary support files such as makefiles, project files, etc.)

GREEN 2 for each of the 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 (these should go in the repo's *logs* and *solutions* directories),

GREEN 3 a document in PDF format headed by your name, AU E-mail address, and the string "COMP x66y Fall 2020 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 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 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 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 in the appropriate repo directories.

YELLOW 1 Up to 10% (bonus for COMP 5660 students) can be earned by adding an option for fitness sharing and investigating and reporting on how the addition of fitness sharing affects the performance compared to your base MOEA.

YELLOW 2 Up to 10% (bonus for COMP 5660 students) can be earned by adding an option for crowding and investigating and reporting on how the addition of crowding affects the performance compared to your base MOEA.

RED 1 Up to 10% bonus points can be earned by in addition to the main 1d assignment, adding as fourth objective minimizing the number of bulbs placed. You need to investigate and report on how your MOEA's behavior and performance are impacted by having four conflicting objectives rather than just three.

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 11, 2020.

Grading

The point distribution per deliverable category is as follows:

Assessment Rubric \ Deliverable Category	GREEN	YELLOW	RED
Algorithmic	40%	35%	35%
Configuration files and parsing	5%	5%	5%
Logging and output/solution files	5%	0%	0%
Good programming practices & robustness	10%	10%	10%
Writing	25%	25%	25%
Statistical analysis	15%	25%	25%