CS5401 FS2018 - Solving NP-Complete Light Up Puzzle

Daniel Tauritz, Ph.D.

September 3, 2018

Synopsis

The goal of this assignment set 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, random 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 proscribed 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.

General implementation requirements

Your program needs to by default take as input a configuration file *default.cfg* in which case it should run without user interaction and you must provide a command line override (this may be handy for testing on different configuration files) which allows the user to specify a different configuration file. The configuration file should at minimum:

1. specify either to read a datafile containing a Light Up puzzle in the previously specified format, or to randomly generate a solvable Light Up puzzle where the intent is to have a non-zero probability of generating any solvable board
2. in case of reading in a datafile, specify the relative file path+name of the datafile to read
3. in case of randomly generating a Light Up puzzle, specify the number of rows and columns and any additional parameters useful for controlling puzzle generation
4. either an indicator specifying whether the random number generator should be initialized based on time in microseconds or the seed for the random number generator (to allow your results to be reproduced)
5. all black-box search algorithm parameters
6. the number of runs a single experiment consists of (needed for stochastic algorithms to achieve statistically relevant results)
7. the relative file path+name of the log file

8. the relative file path+name of the solution file

The log file should at minimum include the name of the Light Up puzzle data file or indicate that each run a different puzzle was randomly generated and provide the number of rows and columns and any additional parameters used for controlling puzzle generation, the random number generator seed, the algorithm parameters, and an algorithm specific result log (specified in the 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.

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
```

GitLab requirements

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

At the end of the assignment deadline, the code currently pushed to Master will be pulled for grading if it is marked as ready. You are required to include a `run.sh` that needs to be configured to compile and run with the command `./run.sh` using a configuration provided by you. Specifically, in order to run your submission and grade your output, all the TAs should have to do is execute `./run.sh` to run it with the default configuration file, or `./run.sh config.cfg` to run it with an alternate configuration file called `config.cfg`. 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.

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 11:59pm 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.

Resubmissions, 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://git-classes.mst.edu`] and verify that your files are present. If for any reason you submit late, then **please notify the TA 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.

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

Some assignments may offer bonus points for extra work, but note that the max grade for the average of all assignments is capped at 100%.

Assignment 1a: Random Search

Implement a random search which generates uniform random 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. 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 may for this random search 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 random search to find some valid solutions. Note that this is a purely optional addition. 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:

1. your source code (including any necessary support files such as makefiles, project files, etc.)
2. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, 10,000 fitness evaluations, timer initialized random seed, and 30 runs, along with the corresponding log file and the corresponding solution file
3. a configuration file configured for reading in the puzzle file provided on the course website, 10,000 fitness evaluations, timer initialized random seed, and 30 runs, along with the corresponding log file and the corresponding solution file
4. a document headed by your name, S&T E-mail address, and the string "CS5401 FS2018 Assignment 1a", containing two evals versus fitness plots, one corresponding to the run in your log file for the randomly generated puzzles experiment which produced the best solution, the other for the run in your log file for the provided puzzle experiment which produced the best solution

Include a readme file to explain how to compile your code. The due date for this assignment is 11:59 PM on Sunday September 2, 2018.

Grading
The maximum number of points you can get is 50. The point distribution is as follows:

Algorithmic	25
Configuration files and parsing	10
Logging and output/solution files	5
Good programming practices including code reliability and commenting	5
Document containing evals versus fitness plots	5

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:

1. your source code (including any necessary support files such as makefiles, project files, etc.)
2. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, and the best EA configuration you can find where initialization is plain Uniform Random, along with the corresponding log file and the corresponding solution file
3. a configuration file configured for reading in the puzzle file provided on the course website, at least 10,000 fitness evaluations, timer initialized random seed, and 30 runs, along with the corresponding log file and the corresponding solution file
4. a document headed by your name, S&T E-mail address, and the string "CS5401 FS2018 Assignment 1b", containing:

- two plots which simultaneously show evals versus average local fitness and evals versus local best fitness, one corresponding to your log file for the randomly generated puzzles experiment and averaged over all runs, the other corresponding to your log file for the provided puzzle experiment averaged over all runs
- your statistical analysis for both experiments comparing the average final local 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 local 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).

Include a readme file to explain how to compile your code. The due date for this assignment is 11:59 PM on Sunday September 16, 2018.

Grading
The maximum number of points you can get is 100. The point distribution is as follows:

Algorithmic	50
Configuration files and parsing	10
Logging and output/solution files	10
Good programming practices including code reliability and commenting	10
Document containing evals versus fitness plots	10
Statistical analysis	10

Bonus
Up to 5% bonus points can be earned (though note that the max grade for the average of all assignments is capped at 100%) by performing a comparison of the regular deliverable experiments and those experiments re-run with Validity Forced plus Uniform Random initialization. Provide all the same type deliverables as for the regular deliverables, including plots and statistical analysis. Make sure that it is perfectly clear for all files submitted which are addressing the main assignment and which the bonus!

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 alotted.

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

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:

1. your source code (including any necessary support files such as makefiles, project files, etc.)
2. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing the penalty fitness function, along with the corresponding log file and the corresponding solution file
3. a configuration file configured for reading in the puzzle file provided on the course website, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing the penalty fitness function, along with the corresponding log file and the corresponding solution file
4. a document headed by your name, S&T E-mail address, and the string "CS5401 FS2018 Assignment 1c", containing a write-up containing appropriate plots, statistical analysis, and actual words explaining said plots and statistical analysis, to answer the following questions both for the randomly generated puzzles and for the provided puzzle:

- How much improvement does a constraint-satisfaction EA employing a penalty function obtain versus a plain-vanilla EA?
- 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!
- How is the penalty function coefficient correlated with the solution quality of your constraint-satisfaction EA? Explain your findings!

Include a readme file to explain how to compile your code. The due date for this assignment is 11:59 PM on Sunday October 7, 2018.

Grading
The maximum number of points you can get is 100. The point distribution is as follows:

Algorithmic	50
Configuration files and parsing	5
Logging and output/solution files	5
Good programming practices including code reliability and commenting	10
Document containing evals versus fitness plots	15
Statistical analysis	15

Bonus
Up to 20% bonus points can be earned (though note that the max grade for the average of all assignments is capped at 100%) by implementing an additional constraint satisfaction technique and expanding your document with wording including relevant plots and statistical analysis comparing the additional technique to the penalty function you implemented for the main assignment. The amount of bonus is dependent on both the quality of your work as well as the sophistication of the technique (e.g., simply killing off all invalid solutions will only earn you a few bonus points, while implementing for instance a repair function or closed under operation operators may earn up to the full 20%). If you do the bonus, then instead of the above listed deliverables you will need to submit the following deliverables:

1. your source code (including any necessary support files such as makefiles, project files, etc.) including the ability to select the constraint satisfaction technique in the configuration file

2. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing the penalty fitness function, along with the corresponding log file and the corresponding solution file

3. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing your additional constraint satisfaction technique, along with the corresponding log file and the corresponding solution file

4. a configuration file configured for reading in the puzzle file provided on the course website, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing the penalty fitness function, along with the corresponding log file and the corresponding solution file

5. a configuration file configured for reading in the puzzle file provided on the course website, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, black cell number constraint enforced, and the best EA configuration you can find employing your additional constraint satisfaction technique, along with the corresponding log file and the corresponding solution file

6. a document headed by your name, S&T E-mail address, and the string "CS5401 FS2018 Assignment 1c with bonus", containing a write-up containing appropriate plots, statistical analysis, and actual words explaining said plots and statistical analysis, to answer the following questions both for the randomly generated puzzles and for the provided puzzle:

   - How much improvement does a constraint-satisfaction EA employing a penalty function obtain vs. an EA employing your additional constraint satisfaction technique vs. a plain-vanilla EA?
   - How significant is the impact of Validity Forced plus Uniform Random initialization versus plain Uniform Random initialization for a plain-vanilla EA, a constraint-satisfaction EA employing a penalty function, and a constraint satisdaction technique employing your additional technique? Explain your findings!
   - How is the penalty function coefficient correlated with the solution quality of your constraint-satisfaction EA employing a penalty function? Explain your findings!

Make sure that it is perfectly clear for all files submitted which are addressing the main assignment and which the bonus!

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. Therefore, in this assignment you need to implement a Pareto-front-based multi-objective EA (MOEA), such as NSGA, NSGA-II, SPEA, SNDL-MOEA, or ϵ-MOEA, 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:

1. your source code (including any necessary support files such as makefiles, project files, etc.)
2. a configuration file configured for randomly generating for each individual run puzzles of 10 columns by 12 rows, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, and the best EA configuration you can find, along with the corresponding log file and the corresponding solution file
3. a configuration file configured for reading in the puzzle file provided on the course website, at least 10,000 fitness evaluations, timer initialized random seed, 30 runs, and the best EA configuration you can find, along with the corresponding log file and the corresponding solution file
4. a document headed by your name, S&T E-mail address, and the string "CS5401 FS2018 Assignment 1d", containing a write-up explaining exactly how your MOEA works (i.e., sufficient to allow your work to be reproduced), containing appropriate plots, tables, statistical analysis, and actual words explaining said plots, tables, and statistical analysis, to answer the following questions both for the randomly generated puzzles and for the provided puzzle:
   - How significant is the impact of Validity Forced plus Uniform Random initialization versus plain Uniform Random initialization for your MOEA? Explain your findings.
   - Which combination of parent selection, survival strategy, and survival selection options works best? Explain your findings and include results for at minimum two parent selection options, both survival strategies, and two survival selection options (but not uniform random for both parent and survival selection).

Include a readme file to explain how to compile your code. The due date for this assignment is 11:59 PM on Sunday October 21, 2018.

Grading
The maximum number of points you can get is 100. The point distribution is as follows:

Algorithmic	40
Configuration files and parsing	5
Logging and output/solution files	5
Good programming practices including code reliability and commenting	10
Document containing evals versus fitness plots	25
Statistical analysis	15

Bonus
There are two bonuses available for this assignment (though note that the max grade for the average of all assignments is capped at 100%), namely:

Bonus 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 and providing corresponding deliverables mirroring the main 1d assignment, except that the question you answer is how increasing the number of objectives impacts the number of levels of non-domination and how that in its turn affects the performance of your MOEA.

Bonus 2: Up to 10% bonus points can be earned by in addition to the main 1d assignment, adding an option for either fitness sharing or crowding and providing corresponding deliverables mirroring the main 1d assignment, except that the question you answer is how the addition of fitness sharing or crowding affects the performance of your MOEA. An additional 5% bonus points can be earned by adding options for both fitness sharing and crowding and providing corresponding deliverables mirroring the main 1d assignment, except that the question you answer is a three-way comparison between the main 1d MOEA, that MOEA with fitness sharing, and that MOEA with crowding.

Make sure that it is perfectly clear for all files submitted which are addressing the main assignment and which the bonuses!