CS4205 - Evolutionary Algorithms - 2020/2021

Assignment 1 – Solving MAXCUT with GA, EDAs, and GOMEAs

Background

The Weighted Maximum Cut (MAXCUT) problem is a well-known NP-complete graph problem. The objective of this problem is coloring each vertex in the graph either blue or red, and to maximize the total weight of edges between vertices of different colors. In a sense, each edge going from a red to a blue vertex, or vice versa, is cut. This is shown in Figure 1, where the total objective value is equal to 13.

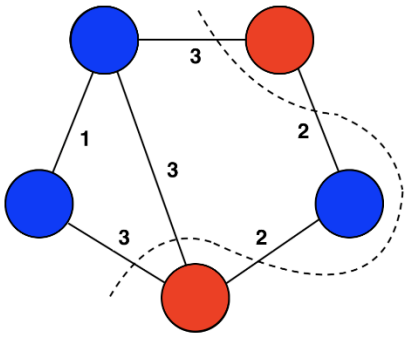


Figure 1. *An example instance of the MAXCUT problem.*

Formally, the MAXCUT problem can be defined as follows: Given an undirected graph $G = (V, E)$, where each edge $e \in E$ has an assigned weight $w(e)$, we are interested in finding a partitioning of V into two subsets, $S \subseteq V$ and $\bar{S} = V \setminus S$, such that the total weight of edges between different partitions is maximized.

Many different graph structures are possible, e.g., sparse, dense, fully connected, planar, grid, or torus graphs. The structure of the graph can greatly influence how difficult it is to optimize, and the performance of algorithms is expected to vary between graphs with different structures. It is therefore interesting to explore the performance of the considered algorithms on different graph structures.

Problem instances

Two types of MAXCUT instances are made available that you should consider in this assignment:

- Fully connected graphs with randomly sampled weights
- Square grid-graphs with randomly sampled weights

All edges have edge weights randomly generated following a Beta distribution with parameters $\alpha = 100$ and $\beta = 1$. To obtain the weights, linear scaling is used to the range of [1;5].

EAs

There are three EAs made available to choose from (one for each student in the group; decide amongst yourselves who studies which algorithm). For each algorithm, two types of variation should be considered. One variation operator is already implemented, the other you need to implement yourself. The EAs to be considered are:

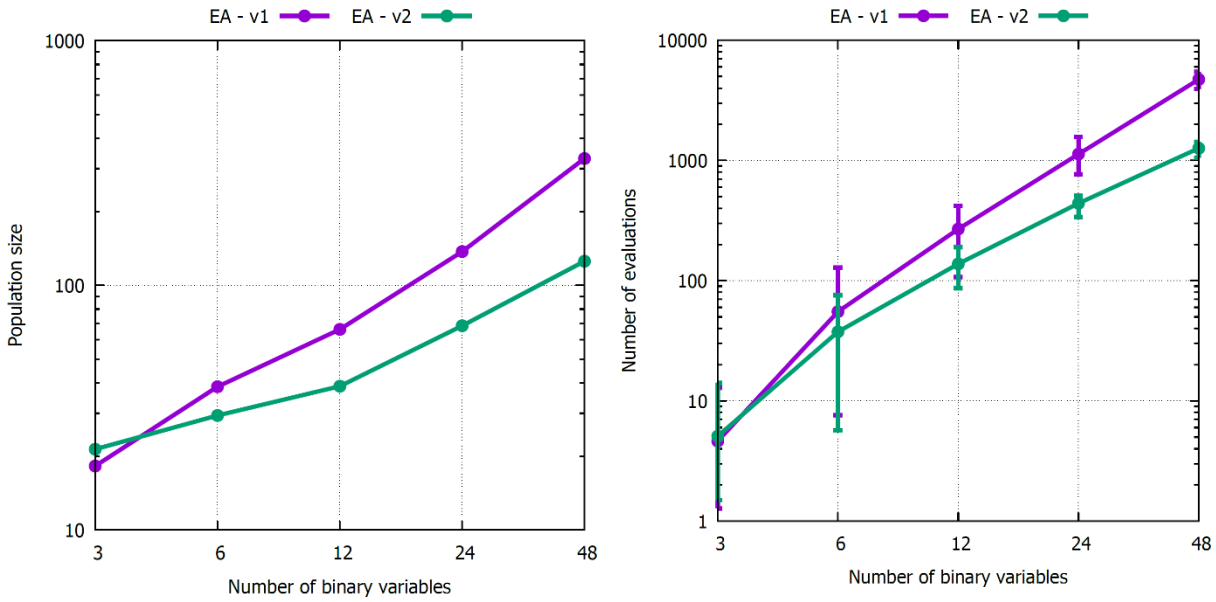
- simple Genetic Algorithm (sGA)
    a. with uniform crossover
    b. with one-point crossover
- Estimation-of-Distribution Algorithm (EDA)
    a. with the univariate factorization
    b. with the marginal product factorization
- Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA)
    a. with the univariate factorization
    b. with the linkage tree

Assignment

As for each assignment, there is an *individual-gradable part* and a *group-gradable part*. Generic items for these parts are described in the document that details the general setup of the practical assignments (such as which sections to write in the paper). Specific items for this practical assignment are provided below.

Individual-gradable part

1. Implement the missing variation operator. In the paper, state how you did this, including pseudo-code.
2. Research the (in)capability of your EA to solve MAXCUT in case of the two different types of problem instance. Specifically, do this by empirically analyzing the *scalability* of your EA. Scalability is closely related to the notion of computational complexity that is central to computer science, as it enables observing what happens to the required resources (i.e., memory and runtime) to solve a problem as the problem size increases.
    a. For a given value of ℓ (problem length; here: the number of nodes in the MAXCUT graph), you need to see how large the population size n needs to be to solve the problem reliably. Take *solving reliably* to mean finding the optimal solution in 10 out of 10 independent runs of the EA. To find n, come up with a *population-sizing approach* and code this (e.g., in (Python) scripts). Make your approach as *efficient as possible* in terms of computational complexity. For this, you may assume that the larger you set the population size, the larger the number of evaluations that the EA will require. Provide pseudo-code, explain why your approach is valid, and explain its computational complexity. More efficient approaches will get higher marks.
    b. As you want to see what happens as the problem size increases, do population-sizing experiments for increasing values of ℓ. To get a good idea of what happens, do so exponentially (go as large as you can, given your computing resources). Because EAs are stochastic, so will your population-sizing approach be. Therefore, repeat your population-sizing approach at least 10 times for each value of ℓ and take the average. Ultimately, you should then be able to plot the so-found average minimally-required population size as well as, after re-running another 10 experiments, the associated average number of required evaluations and variation thereof (such as the interdecile range or the variance) as a function of the problem size. Do this for both types of variation. You should get two graphs that look something like this (*note*: these graphs are for an unspecified EA and for an unspecified problem, so what is in your graph and on your axes will be different):



c. Include these graphs for both types of problem instance in your paper and explain what you see. Include well-motivated (potentially supported by additional experiments, of your design) answers to:
   I. Why are the results different for the two types of variation?
   II. Are the differences more profound for one or the other type of problem instance? If yes, why? If no, why not?
   III. Are the differences statistically significant? *Important*: explain how you test this and why this is a valid approach.

3. Implement a simple bit-flip mutation operator that is applied to every offspring at the end of a generation, just before the offspring are re-integrated with the population. Remember to re-evaluate each offspring if any changes were actually applied to it. The bit-flip mutation operator should simply flip each bit independently with a pre-specified mutation probability p_m. In other words, if $p_m = 0$, the algorithm should behave exactly the same as before. Now, repeat the experiment of finding the optimal population size, but also find the optimal mutation probability. Again, provide pseudo-code in the paper to show how you did this. Note: your EA may no longer converge and/or terminate efficiently, especially for large values of p_m. You may therefore need to propose a different termination criterion in order to be able to determine the optimal population size.

Group-gradable part

1. For each EA, choose the best variant and compare the final EA variants by plotting them in scalability graphs that depict all 3 EAs, for the two types of problem instances separately. Moreover, include well-motivated (potentially supported by additional experiments, of your design) answers to:
   a. Why are the results different for the various EAs?
   b. Are the differences more profound for one or the other type of problem instance? If yes, why? If no, why not?
   c. Are the differences statistically significant? *Important*: explain how you test this and why this is a valid approach.