1 Introduction

Image segmentation is a process that divides an image into many sets of pixels. In this assignment, you will segment images in parallel using Borůvka's MST algorithm.

Segmentation has many useful applications, one of which is posterization. So, while you're not quite writing your own implementation of Adobe Photoshop, this lab will get you 0.0001% of the way there!

2 Files

After downloading the assignment tarball from Autolab, extract the files by running:

```
tar -xvf segmentlab-handout.tgz
```

from a terminal window on a unix/linux system. Some of the files worth looking at are listed below. You should only modify the files denoted by *, as these will be the only ones that will be handed in terms of code

1. * `MkBoruvkaSegmenter.sml`
2. `Tests.sml`

Additionally, you should create a file called:

```
written.pdf
```

which contains the answers to the written parts of the assignment. Your written answers should be composed in a suitable word processor such as LaTeX, Word, OpenOffice, etc. *Handwritten (and scanned in) answers will NOT be accepted.* **Make sure your name is also in the `written.pdf` file.**

3 Submission

To submit your assignment to Autolab in Diderot, generate a handin archive `handin.tgz` with the command

```
make package
```

You can then submit your `handin.tgz` file on Diderot. You should submit the answers to the written questions in *written.pdf* to Gradescope at `https://www.gradescope.com/courses/225303`.

Note that your code submitted to Autolab may not be graded until after the due date. It is your responsibility to test your code thoroughly before submitting.

3.1 Important points

1. Before submission, always, *always* verify that the files you will submit are indeed the versions that you intended to submit. Be very careful to NOT submit the wrong lab's `written.pdf`, blank files, or old versions of the files. **We will not entertain "*Oh no, I submitted the wrong file!*" emails or Diderot posts after the due date**. We will adopt a *zero-tolerance* approach to this, and you will receive the grade of your most recent submission (which may be subject to late submission penalties!) *You have been warned*.

2. If the homework has a written component, the answers to each written question in *written.pdf* should be on one page and each answer should fit on one page. If you decide not to answer one of the questions, still have the empty page in the file. This will make everyone's life easier in Gradescope.

3. If the written part has a question that asks you to show that your code has a certain complexity bound, **you should include a copy of your code in your answer and annotate it as suitable with relevant complexity bounds using comments and then argue or compute the resulting bounds. Answers to such question without a copy of the code you submitted will not be accepted.**

4. The questions below ask you to organize your solutions in a number of modules written almost from scratch. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

5. You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

6. This assignment requires the use of library code spread over several files. The compilation of this is orchestrated by the compilation manager through several `.cm` files. Instructions on how to use CM can be found in the file `cm.pdf` under Resources on Diderot.

7. Style grading for this assignment will be evaluated on a 5 point scale: -2 through +2. Style points will be added to your final grade from the rest of your homework. **You should review the full style guide available on the Diderot under Resources.**

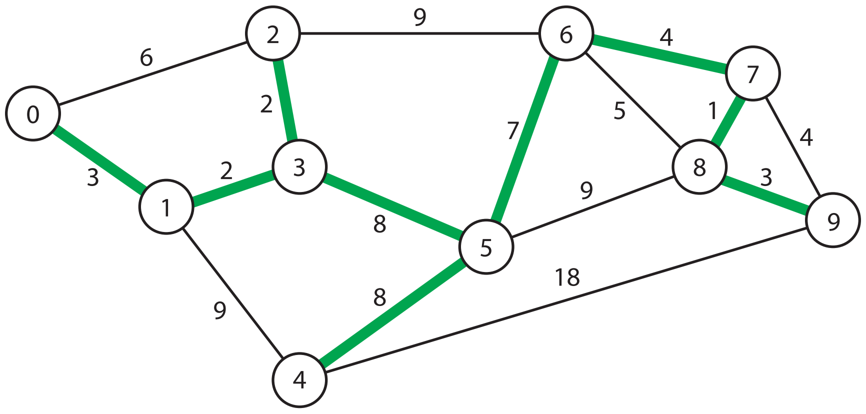
8. **You should submit your solutions on or before the deadline on the top of the first page. Failure to do so will cause substantial deductions of your grade. Late submissions of up to a day will scale your homework grade by 70% (so if you were to get 100% as your grade, you will now get 70%). Late submissions of up to 2 days will scale your homework grade by 50%. Beyond 2 days, you will receive 0 as your homework grade. This scaling will be handled outside Autolab.**

4 Minimum Spanning Trees

Recall that the minimum spanning tree (MST) of a connected undirected graph $G = (V, E)$ where each edge e has weight $w : E \to \mathbb{R}^+$ is the spanning tree T that minimizes

$$\sum_{e \in T} w(e)$$

For example, in the graph below, the MST shown in green has weight 38, which is minimal.

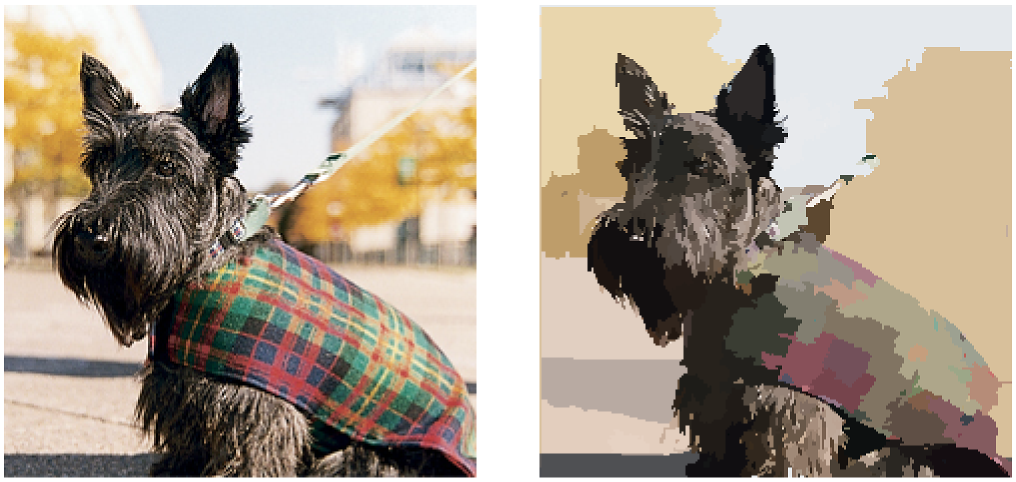


You should be familiar with Kruskal's and Prim's algorithms for finding minimum spanning trees. However, both algorithms are sequential. For this problem, you will use the parallel MST algorithm, Borůvka's algorithm, as presented in lecture.

5 Image Segmentation

Image segmentation is defined as the process of "partitioning a digital image into multiple sets of pixels". The goal of image segmentation is to reduce an image into a simpler form, often that is easier to analyze than the original image.

If we segment an image into disjoint, connected sets of pixels, then color each set the same color, we see a posterization effect. An example of this process is shown below.



We can use MSTs to segment images in this way. Specifically, we will

(a) construct a grid-graph where vertices are pixels and edges connect adjacent pixels,

(b) weight each edge with the difference in color of its endpoints (similar colors result in low weight while different colors result in high weight),

(c) compute the minimum spanning tree of the graph while allowing certain "high-weight" edges to be deleted (more on this soon) such that the graph becomes disconnected.

In fact, this process computes a *minimum spanning forest* (MSF) of a now *disconnected* graph, where each MST within the resulting MSF is considered to be one "segment" of the original graph.

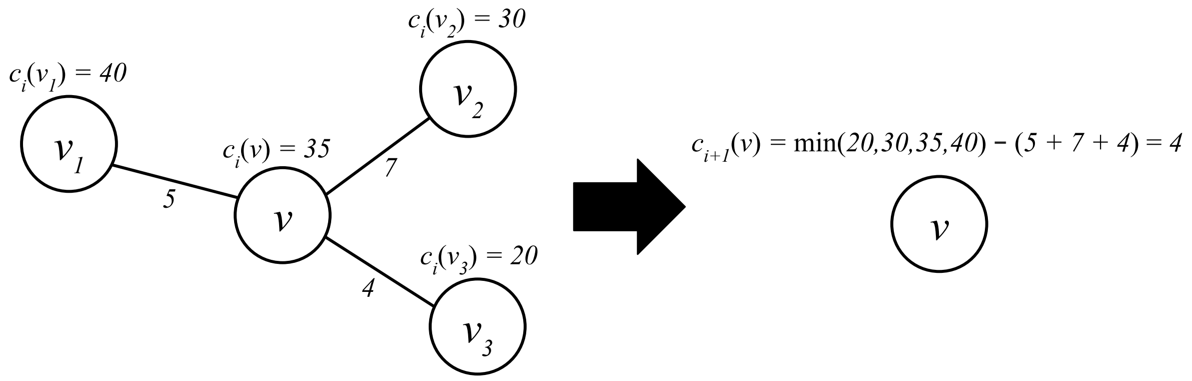
5.1 Algorithm

As mentioned above, we represent the image as a weighted, undirected graph. Our image segmentation algorithm is an extension of Borůvka's algorithm. We begin by assigning some number of "credits" to each vertex. These credits are a sort of currency which vertices may spend in order to contract with other vertices. As we will see, the amount of credit correlates with the size of the segments in the output.

When we contract a star, we have to "spend credits." Specifically, on round i, suppose we're contracting a star X consisting of vertices V_X, edges E_X, and center $v \in V_X$. Let $c_i(u) : u \in V_X$ be the credits of these vertices on this round. We update the credit of the center according to the formula

$$c_{i+1}(v) = \min_{u \in V_X} c_i(u) - \sum_{e \in E_X} w(e).$$

An example is shown below. Note that we don't need to update the credits of the satellites, because they have been contracted and are no longer present in the graph.



There is one caveat: when a vertex runs out of credit, it is no longer allowed to contract. So, at the end of each round of Borůvka's, we try to remove edges which might cause credits to go negative. Specifically, if E_i' is the set of edges remaining at the end of some round i, then the edge set given as input to round $i+1$ is

$$\left\{(u,v) \in E_i' \;\middle|\; \min(c_{i+1}(u), c_{i+1}(v)) \geq w(u,v)\right\}.$$

Notice that in this formula, we use the values c_{i+1} rather than c_i, because E_i' contains only the edges which remain between star centers.

(If even after this process, or if in the initial input, there are nodes that will contract to negative credits, they can be contracted that round.)

It turns out that if we assign an initial credit of ∞ to each vertex, then this algorithm is actually identical to traditional Borůvka's, and can still be used to compute MSTs! We will take advantage of this fact to autograde your code.

6 Implementation and Logistics

Vertices will be labeled $0 \leq v < n$. The input graph is undirected, simple (no self-loops, at most one undirected edge between any two vertices), connected, and has no negative weighted edges. We will represent both the input and output as "edge sets," implemented simply as unordered sequences. Each edge is a triple (u, v, w), indicating an undirected edge between u and v with weight w. For every (u, v, w) in the input, you should assume the input also contains (v, u, w).

6.1 Borůvka Segmenter

Task 6.1 (90 pts). In `MkBoruvkaSegmenter.sml`, implement the function

```
val segment : int → (edge Seq.t * int) → edge Seq.t
```

where (`segment` c (E, n)) implements the algorithm described in the previous section. The inputs are c: the initial credit for each vertex; E: the edge sequence; and n: the number of vertices (recall that vertices are labeled $0 \leq v < n$ within E). For full credit, your algorithm should require $O(|E| \log^2 n)$ work and $O(\log^3 n)$ span in expectation.

Your function should output the sequence of edges chosen for contraction. In the case of $c = \infty$, this should be the minimum spanning tree of the input graph. Unlike the input, you should only include edges pointing in one direction (for each undirected edge in the output, you should either include (u, v, w) or (v, u, w), but not both).

You should use sequences as the data structure for this algorithm. Solutions that use tables will receive *no credit*.

We will allocate 65 points for correctly implementing Borůvka's algorithm, and 25 points for correctly segmenting images. (I.e., if you ignore the argument c and simply compute the MST, then you will be awarded at most 65 points).

6.2 Pseudorandomness

A *pseudorandom number generator* is an algorithm which computes sequences of numbers which appear random. These algorithms are in fact deterministic, and are often "seeded" by some user-provided number.

One of the arguments given to the `MkBoruvkaSegmenter` functor is the structure `Rand`, ascribing to `RANDOM210`. This structure implements a *deterministic parallel random number generator* (DPRNG) with splittable seeds. You will need to use `Rand` to implement Borůvka's algorithm. Specifically, you might find the following function helpful. Given n and an input seed r, this function produces n random booleans along with a fresh seed r'. Its work and span are $O(n)$ and $O(1)$, respectively.

```
fun flipCoins n r =
  let val (r', chooseSeed) = Rand.splitTab (r, n)
  in (r', Seq.tabulate (Rand.bool o chooseSeed) n)
  end
```

Note that due to the deterministic nature of `Rand`, if you call this function twice with the same r, you will get the same sequence of booleans as output both times. This is why the function returns a fresh seed, r', to be used in a subsequent call.

You will need to construct an initial seed for your algorithm with `Rand.fromInt`. You may pass any integer you would like to this function. Changing this number will only result in small aesthetic changes in the output of your segmenter.

6.3 Recommendations

The challenge of this lab is converting high-level descriptions of algorithms into SML. Before attempting to implement anything, you will need to carefully read and understand both (a) the Borůvka pseudocode given in the textbook, and (b) the English description of the segmenting algorithm described in section 5.1. When you begin writing code, we highly recommend working in the following order:

1. Ignore the credits, and simply implement Borůvka's algorithm. Completing this step will earn you up to 65 points.
2. Modify your implementation of Borůvka's algorithm to additionally manipulate credits according to the description given in section 5.1.

In your implementation, since the graph is enumerated, you will be using sequences rather than tables and sets. You might find the function `Seq.inject` particularly useful.

7 Testing (And Segmenting!)

There are three ways to test your code.

1. Run your segmenter at the REPL. For example, the following segments the image `images/dog.png` with an initial credit of 1000, writing the resulting image at `images/my-dog-1000.png`.

   ```
   - CM.make "segmenting.cm";
   ...
   - SegTester.makeSegsFile ("images/dog.png", "images/my-dog-1000.png", 1000);
   ```

   We have provided a few sample images in `images/`, and example segmentations of these in `images/out/` for reference. **Note that running the segmenter should work fine on** `unix.qatar.cmu.edu`. We can't guarantee anything for running it locally on your own machine. So you may have to use the `unix.qatar.cmu.edu`.[1,2]

2. In `Sandbox.sml`, write whatever testing code you'd like. You can then access the sandbox at the REPL:

   ```
   - CM.make "sandbox.cm";
   ...
   - open Sandbox;
   ```

3. In `Tests.sml`, add test cases according to the instructions given. Then run the autograder:

[1] Your Python environment needs to have the Image library and PIL installed. A useful resource is `https://pillow.readthedocs.io/en/stable/`

[2] Also, when you copy your working directories to `unix.qatar.cmu.edu`, please run `chmod +x *.py` for all the Python files in `segmenting` directory.

```
- CM.make "autograder.cm";
...
- Autograder.run ();
```

To autograde your code, we will use an initial credit of $c = \infty$ and verify that you compute a minimum spanning tree. We will hand-grade segmentation qualitatively.

8 Written

Tuning Parameters. Let's say you are given a graph in which every vertex has the same degree d. You want to use star partitioning to contract the graph, and you can bias your coin for deciding which vertices are centers and which are potential satellites. We'd like to try to maximize the number of vertices that will be removed in expectation.

Task 8.1 (5 pts). First, let x be the probability that a node is marked as a satellite. Give an expression for the probability that this node gets contracted.

Task 8.2 (5 pts). What value of x maximizes the expected number of contractions?

Tree Contraction with Edge Partitioning. Recall that edge partitioning is a graph contraction strategy that partitions the graph into disjoint connected components consisting of at most 2 vertices each (also called a vertex matching). For arbitrary graphs, we have seen that edge partitioning is not a sufficient strategy because it cannot always remove at least a constant fraction of the number of edges in expectation. However, edge partitioning *is* viable in certain situations – for example, with certain kinds of trees.

Define a 3-tree to be an undirected, acyclic, connected graph where **every vertex has degree at most 3**. We want to prove that we can contract a 3-tree using edge partitioning in expected $O(n)$ work and $O(\log^2 n)$ span.

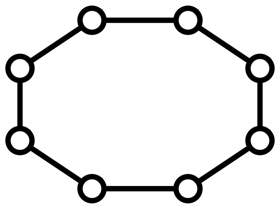
Task 8.3 (10 pts). Prove that, in a 3-tree, the number of vertices with degree strictly less than 3 is at least $\frac{n}{k}$ for some constant $k > 1$. Provide a specific value of k.

Task 8.4 (20 pts). Describe a randomized strategy for selecting edges to contract in a 3-tree. Show that in expectation, one round of contraction will remove at least a constant fraction of the number of edges. For each round, your strategy must be implementable in $O(n)$ work and $O(\log n)$ span; you may assume the vertices are labeled as integers for cost analysis.

(Hint: Your solution will need to use k from the previous question. Make sure the graph satisfies the 3-tree invariant throughout the contraction process.)

Task 8.5 (5 pts). Using the results of the previous part, justify the entire contraction process has expected $O(n)$ work and $O(\log^2 n)$ span.

Basic Graph Contraction Consider a cycle graph with n nodes like the one shown below, and answer the short questions below. Please provide *very short* justifications for your answers. Answers without justification will not get any credits.



Task 8.6 (3 pts). Suppose each edge flips a fair coin that comes up either heads or tails. We select an edge for contraction if it flips heads and the adjacent edges flip tails. What is the expected number of edges contracted after one such round?

Task 8.7 (3 pts). Suppose each vertex randomly selects one of its incident edges for possible contraction and an edge (u, v) is contracted if both vertices u and v choose that edge. What is the expected number of edges contracted after one such round?

Task 8.8 (3 pts). Suppose each edge randomly picks a unique random number and an edge is contracted if its random number is the smallest among its and its neighbor edges' numbers. What is the expected number of edges contracted after one such round?

Maximal Independent Sets Two vertices in a graph are *independent* if there is no edge between them. A set $I \subseteq V$ is an *independent set* of vertices if and only if no vertices in I share an edge.

The *Maximal Independent Set* (MIS) problem is the following: given an undirected graph $G = (V, E)$, find an independent set $I \subseteq V$ such that for all $v \in (V \setminus I)$, $I \cup \{v\}$ is not an independent set. Such a set I is maximal in the sense that we can't add any other vertex and keep it independent, but it easily may not be a maximum—i.e. largest—independent set in the graph.[3]

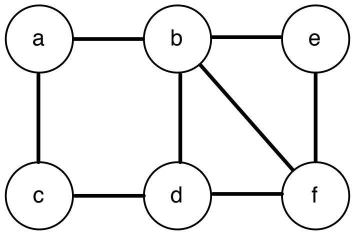


Figure 1: MIS example

For example, consider the graph in Figure 1. The set $\{a, d\}$ is an independent set, but not maximal because $\{a, d, e\}$ is also an independent set. On the other hand, the set $\{a, f\}$ is a maximal independent set because there's no vertex that we can add without losing independence, but it is not a maximum independent set because $\{a, d, e\}$ is independent and larger.

Note that in MIS, we are *not* interested in computing the overall-largest independent set: while maximum independent sets are maximal independent sets, maximal independent set are not necessarily maximum independent sets!

A parallel algorithm for MIS problem takes a graph G as input and proceeds in iterations. In each iteration

1. All vertices pick a unique random number
2. If a vertex has the highest number among all its neighbors, then it joins the MIS and deletes itself and all of its neighbors (and their edges) from the graph

So an iteration of MIS contracts the graph into a smaller graph. (Note that in a given iteration, a vertex with no edges will surely join the MIS and delete itself.) The algorithm terminates when there are no more vertices and edges in the graph.

Task 8.9 (8 pts). Consider a single step of the MIS algorithm. Calculate the probability p_v that a vertex v is selected as part of the independent set during that step. p_v should be written in terms of $\deg(v)$. Answers without supporting reasoning will not receive full credit.

Task 8.10 (4 pts).

Suppose that the maximum degree in G is d, a constant positive integer, and G has n vertices and m edges. What is the expected number of the vertices in the graph after the first round? Answers without supporting reasoning will not receive full credit.

[3]Note that we say "a" not "the" deliberately—there may well be many non-equal independent sets of maximum size.

Putting the MIS Solution to Work – Schedulling Final Exams So you think understanding the MIS problem was hard. Some people have much harder problems to deal with. Every semester Dean Selma has to schedule final exams for all the students. Obviously a student can take only one exam at a time, so Dean Selma has to work extra hard to make sure exam scheduling conflicts do not happen, but does not want to spread the exams over a very long period of time – everyone wants to go home early.[4]

Formally, you are given a set of students S. Each student $s \in S$ is taking some set of exams for courses X_s in a semester. So, for example, 15–210 is in X_s for all students s registered for *15-210*. Your goal is to create a set of time slots T whose elements are sets of exams X_t such that $\forall X_t \in T$, $\forall s \in S$, $|X_t \cap X_s| \leq 1$.

The *exam scheduling* problem is to find a T such that $|T|$ is as small as possible. It turns out the optimal exam scheduling problem is also known to be NP-hard, which means that it is extremely unlikely that it has an algorithm with a polynomial amount of work in the number of exams to be scheduled. But having worked on MIS, you may already have some ideas to find an *approximate solution* to this problem. Your task in this question is to devise an approximate algorithm for helping Dean Selma schedule the final exams.

Task 8.11 (5 pts). State qualitatively with a *short*[5] sentence, what the constraint above, $\forall X_t \in T, \forall s \in S, |X_t \cap X_s| \leq 1$ means.

Task 8.12 (10 pts). Describe a reduction from the exam scheduling problem to the MIS problem. While this reduction doesn't need to be overly formal, it must be precise enough to be unambiguous. What I would like to see in your reduction is

(i) how the information about students and exams get translated to a graph representation, and

(ii) how then the scheduling problem is formulated in terms of an MIS problem.

Task 8.13 (5 pts). How do you use your solution to MIS in the previous question, to actually schedule exams?

[4]Well that was the thinking before the COVID-19 struck.

[5]I really mean short – no stories!