Problem 1: Gate Sizing using Geometric Programming

1. Introduction

Gate sizing plays an important role in circuit design exploration. Sizing gates provides a way to analyze the tradeoff between circuit delay, area, and power. This tradeoff arises from the fact that larger gates can drive a load more quickly than a smaller gate, thus reducing circuit delay by increasing area and power.

For gate sizing, each gate i in a circuit is assigned a scale factor x_i for the gate sizing problem. The delay, area, and power are all functions of x. The particular optimization problem being solved in this problem is formulated as follows:

$$\begin{aligned} &\text{minimize} && \text{Delay} \\ &\text{subject to} && \text{Power} \leq \text{Power}_{\max} \\ & && \text{Area} \leq \text{Area}_{\max} \\ & && x_i \geq 1 \quad \forall i \end{aligned}$$

where Delay is the maximum delay through a path in the circuit, Power is the total power dissipated in all gates, $\text{Power}_{\max}$ is the maximum allowed power, Area is the area of all of the gates, and $\text{Area}_{\max}$ is the maximum allowed total area. Area and the Power model used for this problem depend linearly on x while the RC model used to calculate Delay is a non-linear function of x. This formulation is a geometric program (GP) which implies convexity and tractability. In order for a formulation to be a GP the objective and inequality constraints must be *posynomials*.

2. Problem Statement

2.1 Description

In this problem, early-stage analysis of combinational logic circuits is performed using gate sizing. Your task is to complete the function `populateFormulation` which is a member of the `gateSizer` class in `gateSizer.cpp` and `gateSizer.h`. This function prepares the data structures before the solver is called. The recursive computation of the maximum delay from a primary input to the output of gate i, called T_i in the reference paper, is also to be implemented as the formulation is being prepared. Once the `populateFormulation` function is correctly implemented, the program will calculate the delay, power, and area for a variety of limits, as shown in the reference paper.

2.2 Key Information/Assumptions

The `gateSizing` program takes the test name as the only input as in

```
>> ./gateSizing ../tests/boyd
```

The program writes a GP formulation to a file with suffix .eo, calls Mosek, and reads in the solution from Mosek. The .eo file is the input for the solver Mosek, in particular, the `mskexpopt` solver. Details on the file format can be found in reference [2]. The `mosek` member of `gateSizer` called `formulation`, stores each of the three components in the .eo file: the coefficient of each monomial term, the association of each monomial term (the constraint it belongs to), and the variables and the corresponding powers of the variables in each monomial. The problem can be solved by executing the command

```
>> mskexpopt problem.eo
```

which, in addition to displaying summary information on the standard output, will generate a .sol file containing the objective and variable values. (variable values in the .sol file are the natural log of the desired values, e.g. $\ln(x_i)$) The interaction with Mosek is implemented in `mosek.h` and `mosek.cpp`.

It is important to note that only inequality constraints of the form $posynomial <= 1$ are supported by `mskexpopt`, i.e. equality constraints cannot be used. Although the formulation in the reference paper uses equality constraints, it can be reformulated so that only inequality constraints are used. The max operator also cannot be used directly. An example of how to replace the max operator with equivalent constraints is presented in the comments above `populateFormulation` in `gateSizer.cpp`.

At the time the `populateFormulation` function is called, the posynomials for gate delay (`m_delay` members of class `cell`) are already constructed. A variety of helper functions in class `gateSizer` and operations defined for `posynomial` and `monomial`, can be used to help accomplish this task.

2.3 Input/Output Specification

2.3.1 Input Specification

The input test cases are located in the tests directory. Each test case is specified in .bdnet file format and can be parsed by the provided code. A companion cell library is hard-coded in `cellLibrary.h` and `cellLibrary.cpp`. The cell library specifies all the parameters associated with the different types of cells. (The parameters are called a, e, f, α, β, and γ in the reference paper)

2.3.2 Output Specification

On successful completion, an excerpt of the program output when the command:

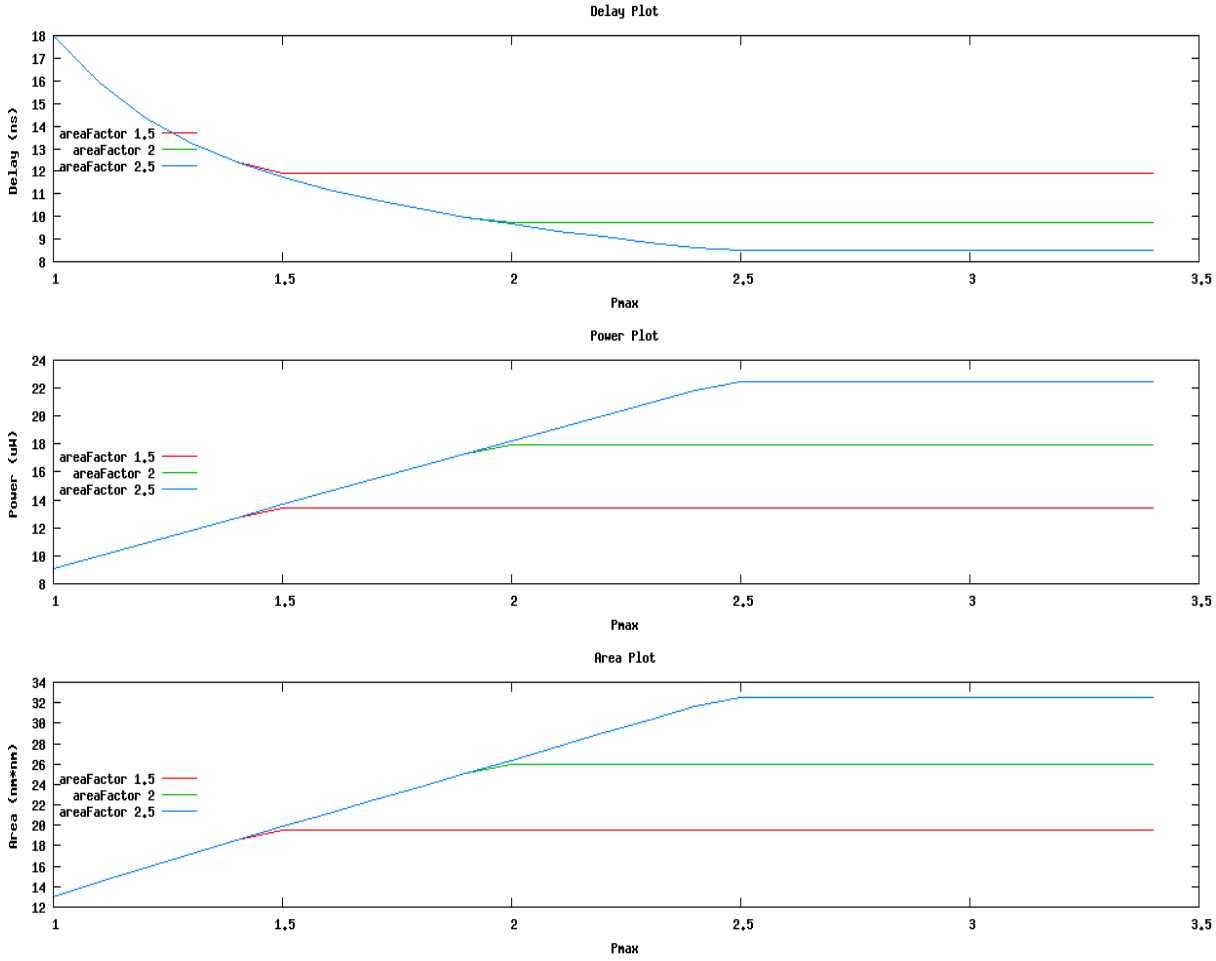
```
>> ./gateSizing tests/boyd | grep Delay
```

is entered on the command line is shown below:

```
areaFactor: 1.5   powerFactor: 1    Delay: 18ns Power: 9.1uW      Area:
13nm^2
```

```
areaFactor: 1.5   powerFactor: 1.1  Delay: 15.9121ns  Power: 10.01uW    Area:
14.4303nm^2
areaFactor: 1.5   powerFactor: 1.2  Delay: 14.3628ns  Power: 10.92uW    Area:
15.8757nm^2
areaFactor: 1.5   powerFactor: 1.3  Delay: 13.2588ns  Power: 11.83uW    Area:
17.2442nm^2
...
```

A result file will also be generated for each `areaFactor` in the tests directory, for example `boydaf1.5.res`. The `tests/boyd.png` tradeoff plot generated will look as follows:



3. Grading

Your program will be tested on 8 testbenches. For each testcase, you will be awarded one point if the output of your program matches the expected results. Each individual test will be allowed to run for 120s. If your program fails to produce a complete result within 120s, you will not get any points.

4. References

[1] S. Boyd, S.-J. Kim, L. Vandenberghe, A. Hassibi, "A Tutorial on Geometric Programming", Optimization and Engineering, 8(1), 2007, pp. 67-127

[2] Mosek exponential optimization: http://docs.mosek.com/6.0/capi/node008.html#253223784 (Sections 6.2.1-6.2.5) Mosek is the solver used to solve the GP. This explains the I/O file formats used by Mosek for exponential optimization (the transformation $x = \ln(y)$ [where x refers to the variables in the Mosek example, not the scale factors in the GP sizing problem] converts the exponential optimization formulation into a standard form GP in y). This information may be of value in debugging.