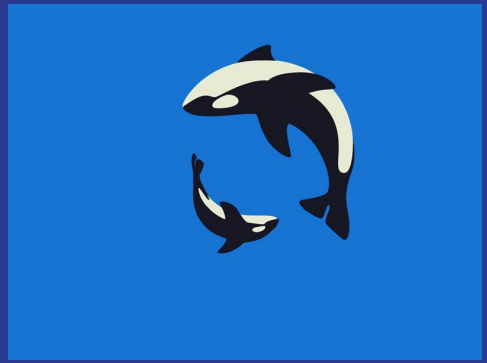
Estimadores DFSA

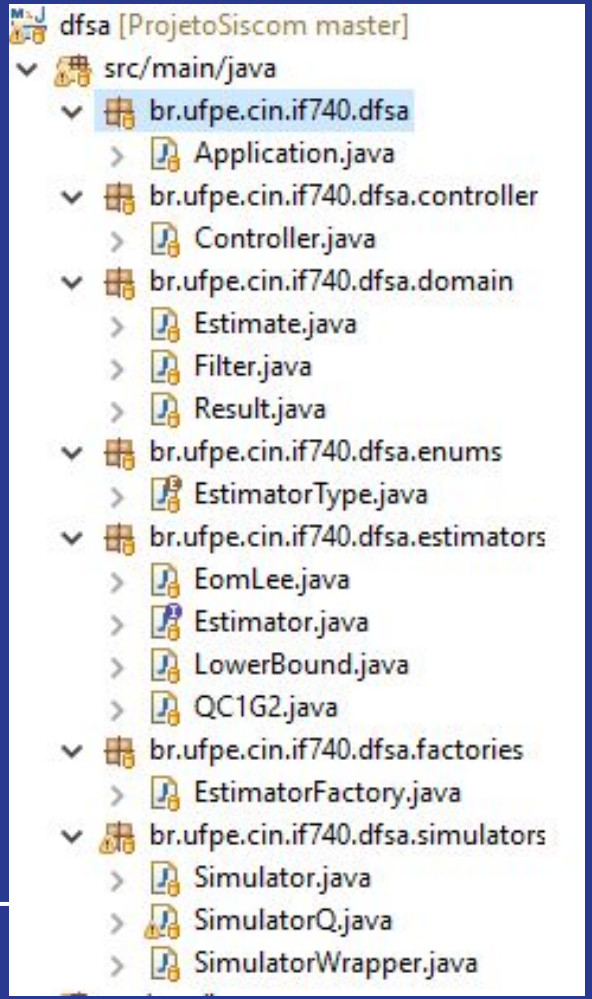
Adilson Angelo (aasj2)
Matheus Feliciano (mbdf)



Simulador

Implementação

- Linguagem
  - Java
- Bibliotecas
  - Spring Boot
- Gráficos
  - JavaScript
  - Chart JS



Detalhes de implementação

Main

```java
@SpringBootApplication
public class Application {
    public static void main( String[] args ) throws IOException, URISyntaxException {
        setOutputFile();

        SpringApplication.run(Application.class, args);
    }
```

Rest Controller

```java
@RestController
public class Controller {

    @RequestMapping("/")
    public String index() throws IOException {
        byte[] encoded = Files.readAllBytes(Paths.get("assets" + File.separator + "view"  + File.separator + "index.html"));
        return new String(encoded, "UTF-8");
    }

    @RequestMapping("/{filename}")
    public byte[] getFile(@PathVariable("filename") String fileName) throws IOException {
        String filePath = "assets" + File.separator + "view"  + File.separator + fileName;
        File file = new File(filePath);
        if(file.exists()) {
            return Files.readAllBytes(Paths.get(filePath));
        }
        return new byte[4];
    }

    @PostMapping(value = "/submit-filter")
    public @ResponseBody List<Result> submitFilter(@RequestBody Filter filter) {
        return SimulatorWrapper.getResults(filter);
    }

}
```

Interface Estimator

```java
public interface Estimator {

    int estimate (int success, int collision, int empty);

}
```

Lower Bound

```
public class LowerBound implements Estimator{

    public int estimate(int success, int collision, int empty) {
        return collision * 2;
    }

}
```

Eom-Lee

```java
public class EomLee implements Estimator{

    private double threshold;

    public EomLee() {
        this.setThreshold(0.001);
    }

    public int estimate(int success, int collision, int empty) throws ArithmeticException{
        double gama = EomLeeGamaIteration(success, collision, empty);

        return (int) Math.ceil(gama * collision);
    }

    private double EomLeeGamaIteration(int success, int collision, int empty) {
        double beta, currentGama, previousGama;
        currentGama = 2;
        previousGama = Double.MAX_VALUE;
        beta = Double.MAX_VALUE;

        while(Math.abs(previousGama - currentGama) >= this.threshold) {
            previousGama = currentGama;

            beta = EomLeeBeta(success, collision, empty, previousGama);
            currentGama = EomLeeGama(beta);
        }

        return currentGama;
    }

    private double EomLeeGama(double beta) {
        return (1 - Math.pow(Math.E, -(1/beta))) / (beta * (1 - (1 + 1/beta) * Math.pow(Math.E, -(1/beta))));
    }

    private double EomLeeBeta(int success, int collision, int empty, double previousGama) {
        int frameSize = success + collision + empty;

        return frameSize / (previousGama * collision + success);
    }
```

Simulator Wrapper

```java
public class SimulatorWrapper {
    public static List<Result> getResults(Filter filter) {
        List<Estimator> estimators = new LinkedList<Estimator>();

        if(filter.isLb())
            estimators.add(EstimatorFactory.getEstimator(EstimatorType.LOWER_BOUND));
        if(filter.isEl())
            estimators.add(EstimatorFactory.getEstimator(EstimatorType.EOM_LEE));
        if(filter.isQ())
            estimators.add(EstimatorFactory.getEstimator(EstimatorType.Q_C1_G2));

        long begining = new Date().getTime();

        List<Thread> threads = new LinkedList<Thread>();
        List<Simulator> sims = new LinkedList<Simulator>();

        for (Estimator e : estimators) {
            Simulator s;

            if(e.getType().equals(EstimatorType.Q_C1_G2)) {
                s = new SimulatorQ(filter.getNumTags(), filter.getStep(),
                        filter.getMaxTags(), filter.getIterations(), filter.getInitialFrameSize());
            } else {
                s = new Simulator(e, filter.getNumTags(), filter.getStep(),
                        filter.getMaxTags(), filter.getIterations(), filter.getInitialFrameSize());
            }

            sims.add(s);
            Thread t = new Thread(s);
            threads.add(t);
            t.run();
        }

        for(Thread t : threads) {
            try {
                t.join();
            } catch (InterruptedException e) {
                System.err.println(e);
            }
        }

        List<Result> results = new LinkedList<Result>();

        for(Simulator sim : sims) {
            results.add(sim.getResult());
        }

        long totalTime = new Date().getTime() - begining;
        long min = TimeUnit.MILLISECONDS.toMinutes(totalTime);

        System.out.println("> Tempo total do programa: " + min + " min " + (TimeUnit.MILLISECONDS.t
        return results;
    }
```

Simulator

```
public class Simulator implements Runnable{

    private int numTags;
    private final int step;
    private final int maxTags;
    private final int iterations;
    private final int initialFrameSize;
    private Estimator estimator;
    private List<Estimate> results;

    public Simulator(Estimator estimator, int numTags, int step, int maxTags, int iterations, int initialFrameSize) {
```

Simulator

```
public void run() {
    System.out.println("\nInitializing " + this.estimator.getClass().getSimpleName() + "\n")

    List<Estimate> estimates = new LinkedList<Estimate>();
    for(; numTags <= maxTags; numTags += step) {
        int totalEmpty = 0;
        int totalSuccess = 0;
        int totalCollision = 0;
        long totalTime = 0;

        for(int i = 0; i < iterations; i++) {
            long begining = new Date().getTime();

            int success, collision, empty;
            int frameSize = initialFrameSize;
            int tagsRemaining = numTags;

            while(tagsRemaining > 0) {

                success = 0; collision = 0; empty = 0;

                int[] frame = new int[frameSize];

                for(int j = 0; j < tagsRemaining; j++)
                    frame[randomInt(0, frameSize-1)]++;

                for(int j = 0; j < frame.length; j++)
                    if(frame[j] == 1)
                        success++;
                    else if(frame[j] > 1)
                        collision++;
                    else if(frame[j] < 1)
                        empty++;

                tagsRemaining -= success;

                frameSize = this.estimator.estimate(success, collision, empty);

                totalEmpty += empty;
                totalSuccess += success;
                totalCollision += collision;
            }
            long end = new Date().getTime();

            totalTime += end - begining;
        }

        double avgSuccess = (double) totalSuccess/iterations;
        double avgEmpty = (double) totalEmpty/iterations;
        double avgCollision = (double) totalCollision/iterations;
        double avgTime = (double) totalTime/iterations;

        System.out.println("-> " + avgSuccess + " " + avgEmpty + " " + avgCollision + " " +

        estimates.add(new Estimate(avgSuccess, avgCollision, avgEmpty, avgTime));
    }
    this.setResults(estimates);
}
```

SimulatorQ

```
public SimulatorQ(int numTags, int step, int maxTags, int iterations, int initialFrameSize) {
    super(EstimatorFactory.getEstimator(EstimatorType.Q_C1_G2),
            numTags, step, maxTags, iterations, initialFrameSize);

    this.qfp = 4.0;
    this.q = (int) Math.round(this.qfp);
    this.c = 0.5;
}
```

SimulatorQ

```java
public void run() {
    System.out.println("\nInitializing SimulatorQ\n");

    List<Estimate> estimates = new LinkedList<Estimate>();

    for(int numTags = getNumTags(); numTags <= getMaxTags(); numTags += getStep()) {

        int totalEmpty = 0;
        int totalSuccess = 0;
        int totalCollision = 0;
        long totalTime = 0;

        for(int i = 0; i < getIterations(); i++) {
            long begining = new Date().getTime();

            int tagsRemaining = numTags;
            tagsSN = new LinkedList<Integer>();

            query(tagsRemaining);
            int check = checkQuery();

            while(tagsRemaining > 0) {
                switch (check) {
                case 0:
                    queryAdj(false, tagsRemaining);

                    totalEmpty++;
                    break;
                case 1:
                    tagsSN.remove(new Integer(0));
                    tagsRemaining--;
                    queryRep(tagsRemaining);

                    totalSuccess++;
                    break;
                default:
                    queryAdj(true, tagsRemaining);

                    totalCollision++;
                    break;
                }

                check = checkQuery();
            }
            long end = new Date().getTime();
            totalTime += end - begining;
        }

        double avgSuccess = (double) totalSuccess/this.getIterations();
        double avgEmpty = (double) totalEmpty/this.getIterations();
        double avgCollision = (double) totalCollision/this.getIterations();
        double avgTime = (double) totalTime/this.getIterations();

        System.out.println(this.getEstimator().getClass().getSimpleName() + " -> " + avgSuccess

        estimates.add(new Estimate(avgSuccess, avgCollision, avgEmpty, avgTime));
    }
    this.setResult(new Result(this.getEstimator().getClass().getSimpleName(), estimates));
}
```

SimulatorQ

```
private int checkQuery() {
    int zeros = 0;
    for(Integer i : tagsSN) {
        if(i == 0) zeros++;
        if(zeros > 1) break;
    }
    return zeros;
}

private void query(int numTags) {
    this.tagsSN.clear();
    for(int i = 0; i < numTags; i++) {
        tagsSN.add(randomInt(0, (int)Math.pow(2, this.q) - 1));
    }
}

private void queryAdj(boolean collision, int numTags){
    if(collision) {
        qfp = Math.min(15, qfp + c);
        q = (int) Math.round(qfp);
    } else {
        qfp = Math.max(0, qfp - c);
        q = (int) Math.round(qfp);
    }

    query(numTags);
}

private void queryRep(int numTags) {
    for(int i = 0; i < numTags; i++) {
        tagsSN.set(i, tagsSN.get(i) - 1);
    }
}
```

Métricas

```java
public class Estimate {

    private double total;
    private double success;
    private double collision;
    private double empty;
    private double time;
    private double efficiency;

    public Estimate(double success, double collision, double empty, double time) {
        this.total = success + collision + empty;
        this.setSuccess(success);
        this.setCollision(collision);
        this.setEmpty(empty);
        this.setTime(time);
        this.setEfficiency(100 * (success/total));
    }
```

Resultados

```java
public class Result {
    private String estimator;
    private List<Estimate> estimates;

    public Result(String estimator, List<Estimate> estimates) {
        this.estimator = estimator;
        this.estimates = estimates;
    }

    public List<Estimate> getEstimates() {
        return estimates;
    }

    public void setEstimates(List<Estimate> estimates) {
        this.estimates = estimates;
    }

    public String getEstimator() {
        return estimator;
    }

    public void setEstimator(String estimator) {
        this.estimator = estimator;
    }
}
```

Gráfico

```
$.ajax({
    url: "/submit-filter",
    contentType: 'application/json; charset=utf-8',
    data: JSON.stringify({
        numTags: numTags,
        maxTags: maxTags,
        step: step,
        iterations: iterations,
        initialFrameSize: initialFrameSize,
        lb: lb,
        el: el,
        q: q
    }),
    method: "POST",
    async: false,
    beforeSend: function () {
        $("#loading").show();
    },
    success: function (data) {
        console.log(data);
        generateCharts(data);
        $("#loading").fadeOut(1000);
        $("#btn-submit").removeAttr("disabled");
    },
    error: function (err) {
        console.error(err);
        $("#loading").fadeOut(1000);
        $("#btn-submit").removeAttr("disabled");
    }
});
```

Interface

Estimadores DFSA

Alunos:

- Adilson Angelo (aasj2)
- Matheus Feliciano (mbdf)

Initial tag number: 100

Number of iterations: 2000

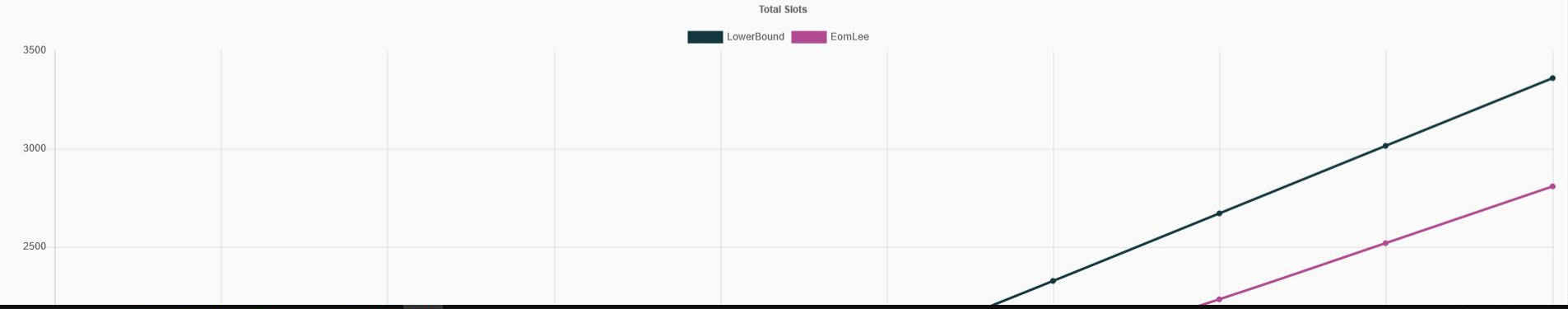
Step: 100

Initial frame size: 64

Maximum tag number: 1000

- [x] Lower Bound
- [x] Eom-Lee
- [ ] Q

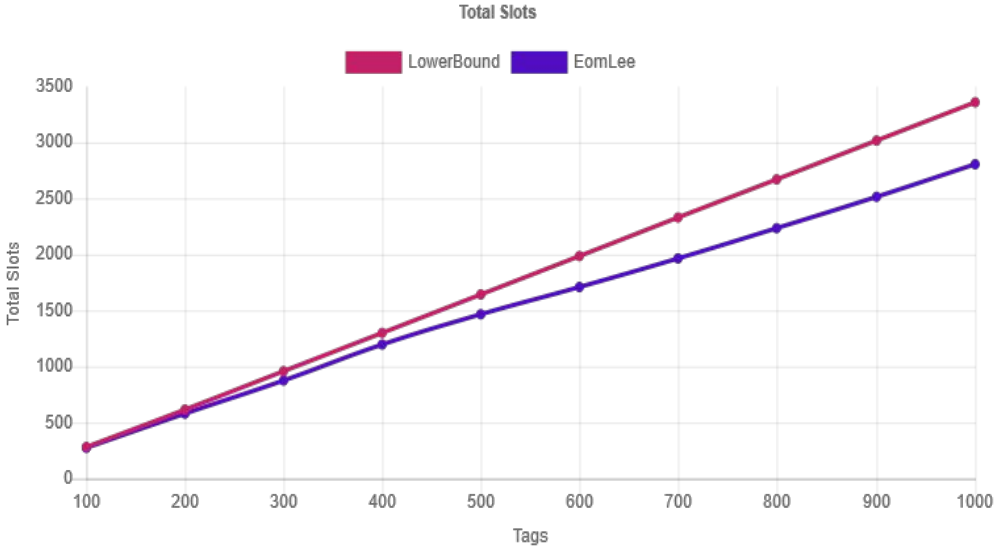
RUN

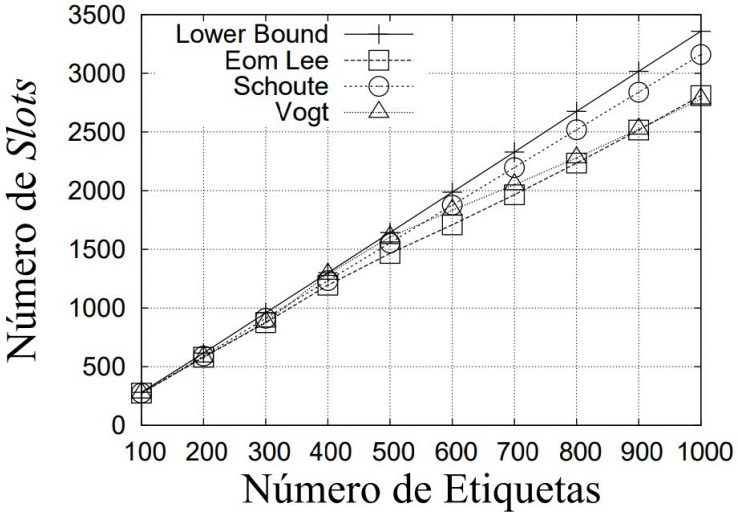


Gráficos

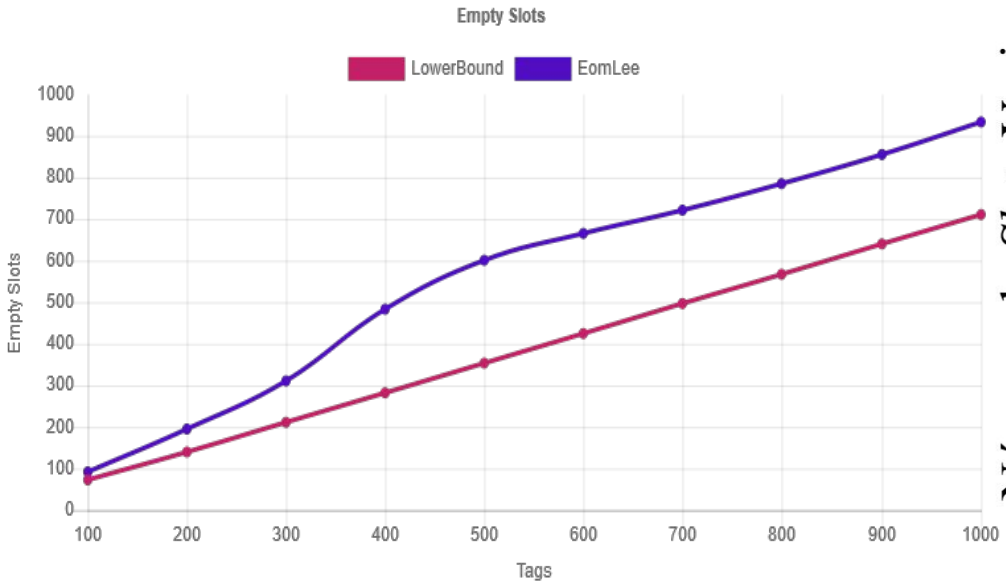
Eom-Lee
&
Lower Bound

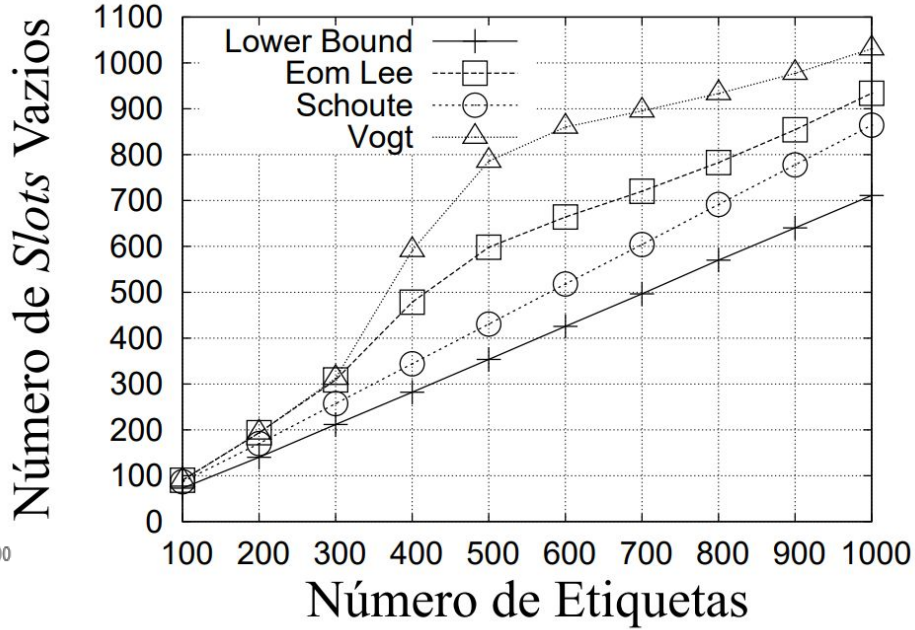
Total de slots



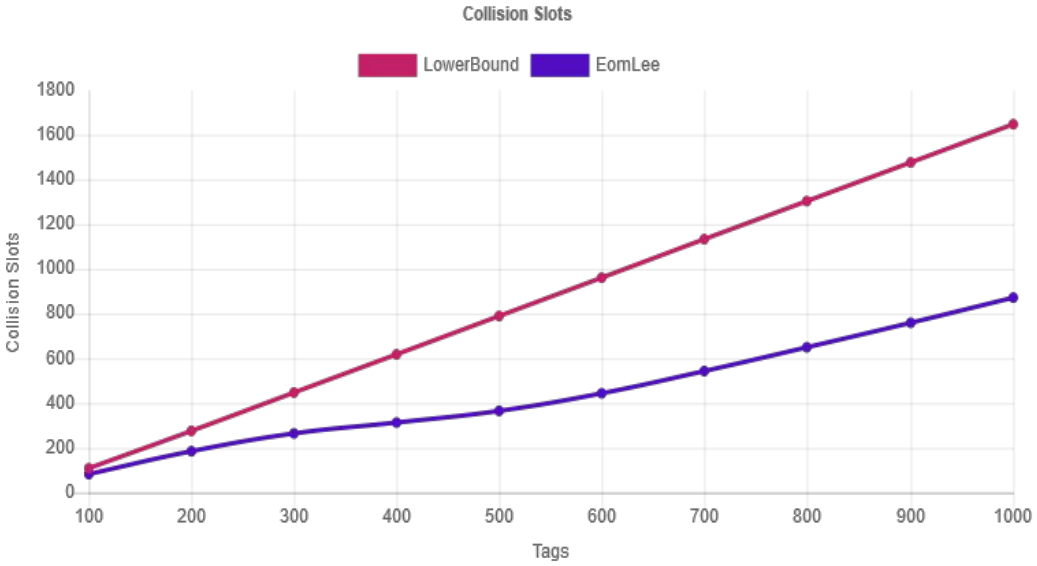


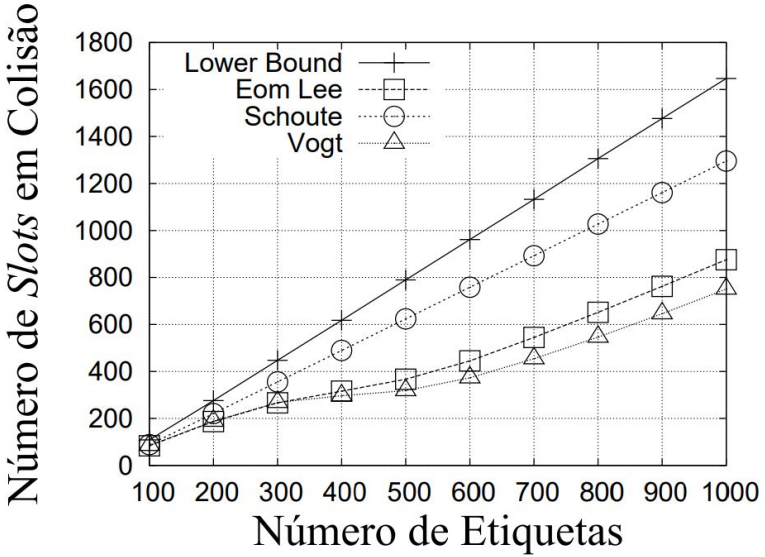
Slots vazios



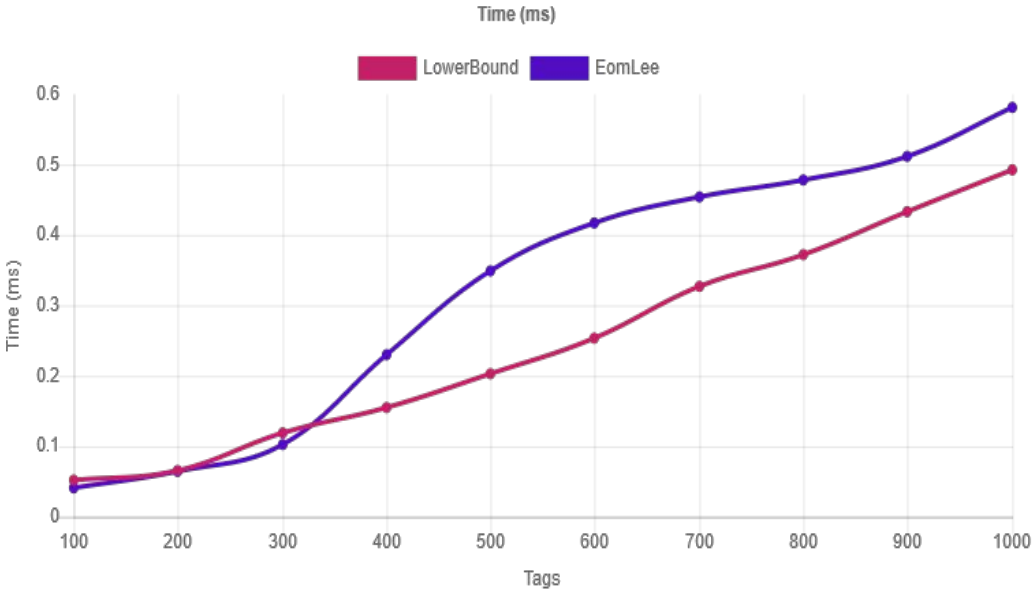


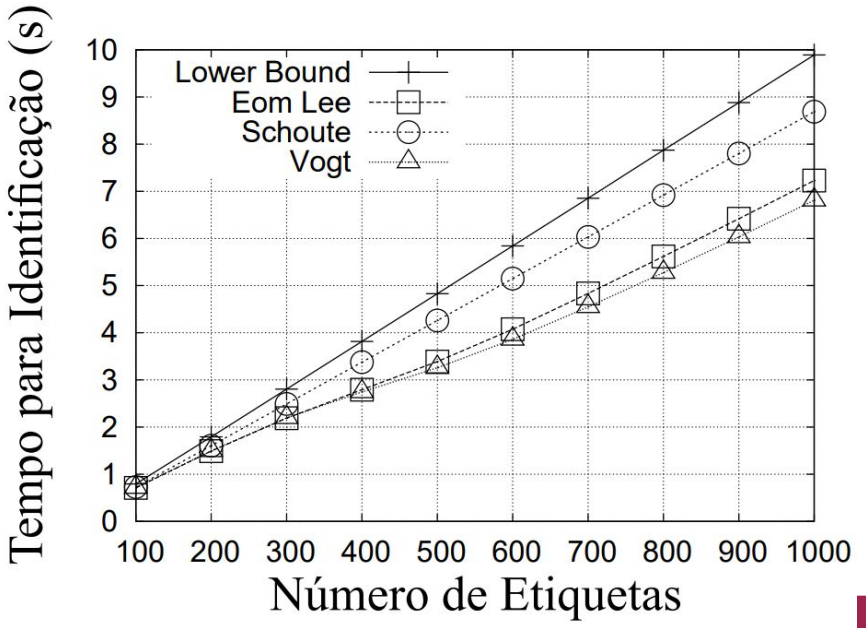
Número de colisões



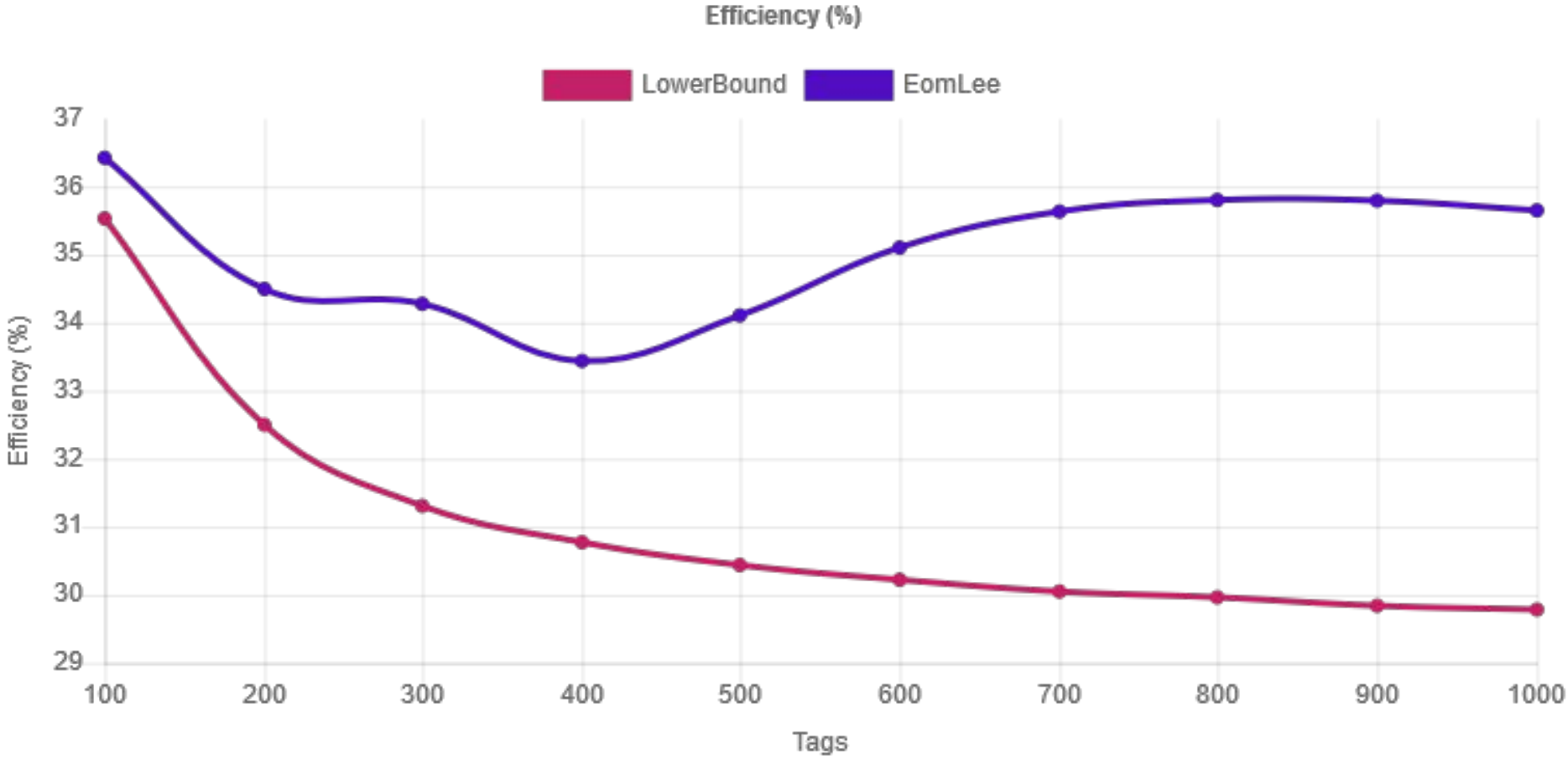


Tempo

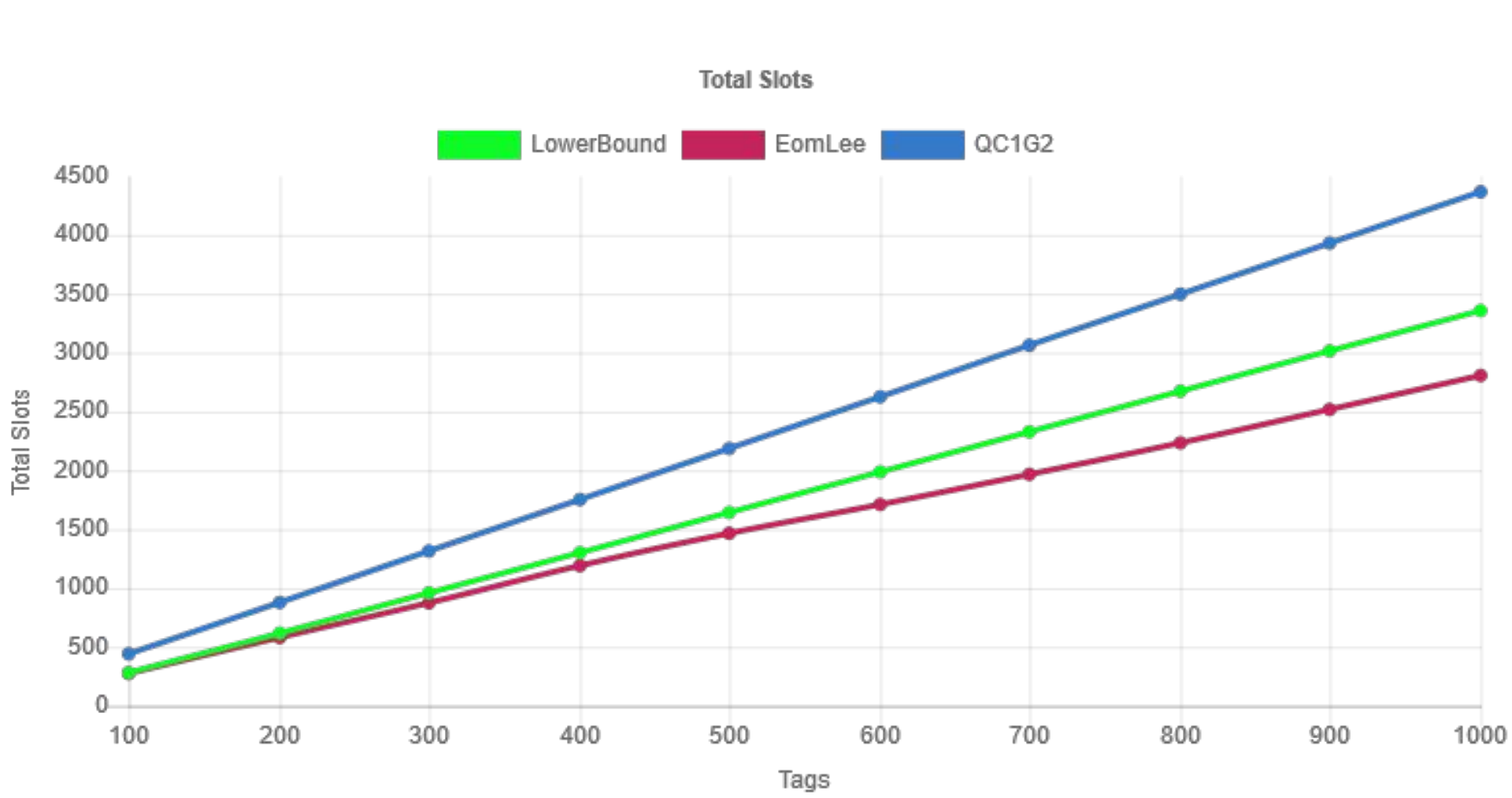

Time (ms)
LowerBound
EomLee
Time (ms)
0
0.1
0.2
0.3
0.4
0.5
0.6
100
200
300
400
500
600
700
800
900
1000
Tags

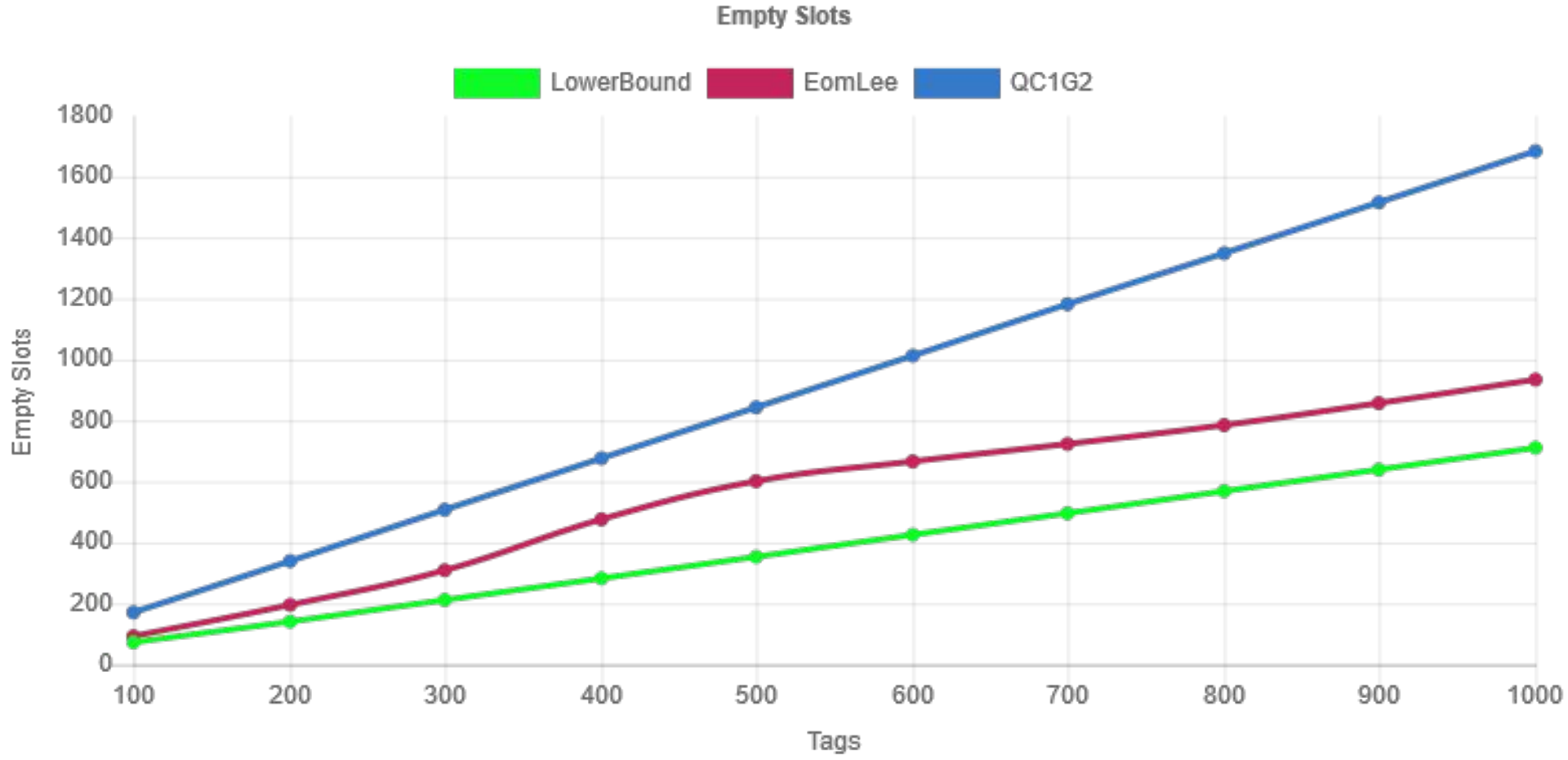

Tempo para Identificação (s)
Lower Bound
Eom Lee
Schoute
Vogt
0
1
2
3
4
5
6
7
8
9
10
100
200
300
400
500
600
700
800
900
1000
Número de Etiquetas

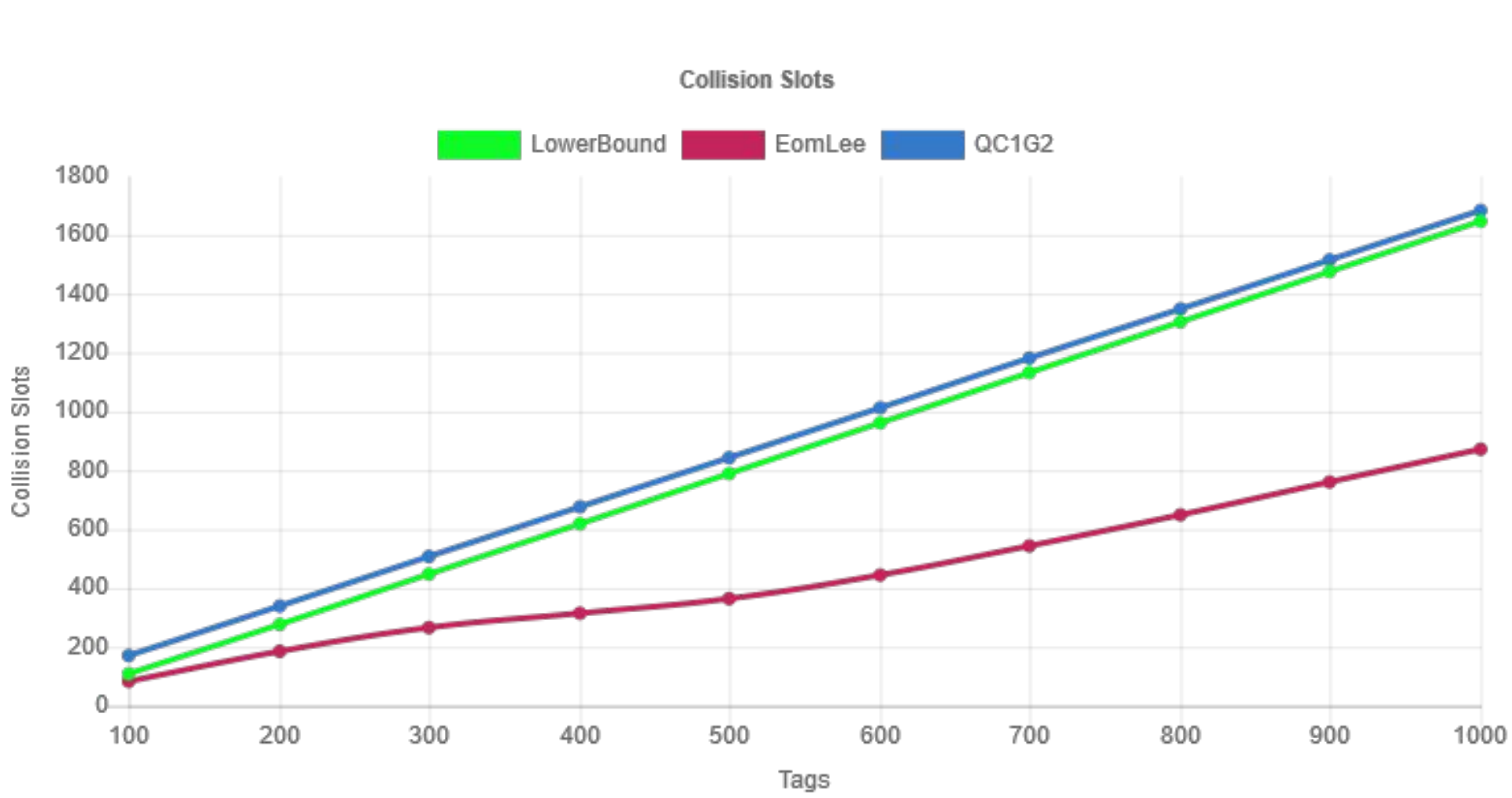
Eficiência

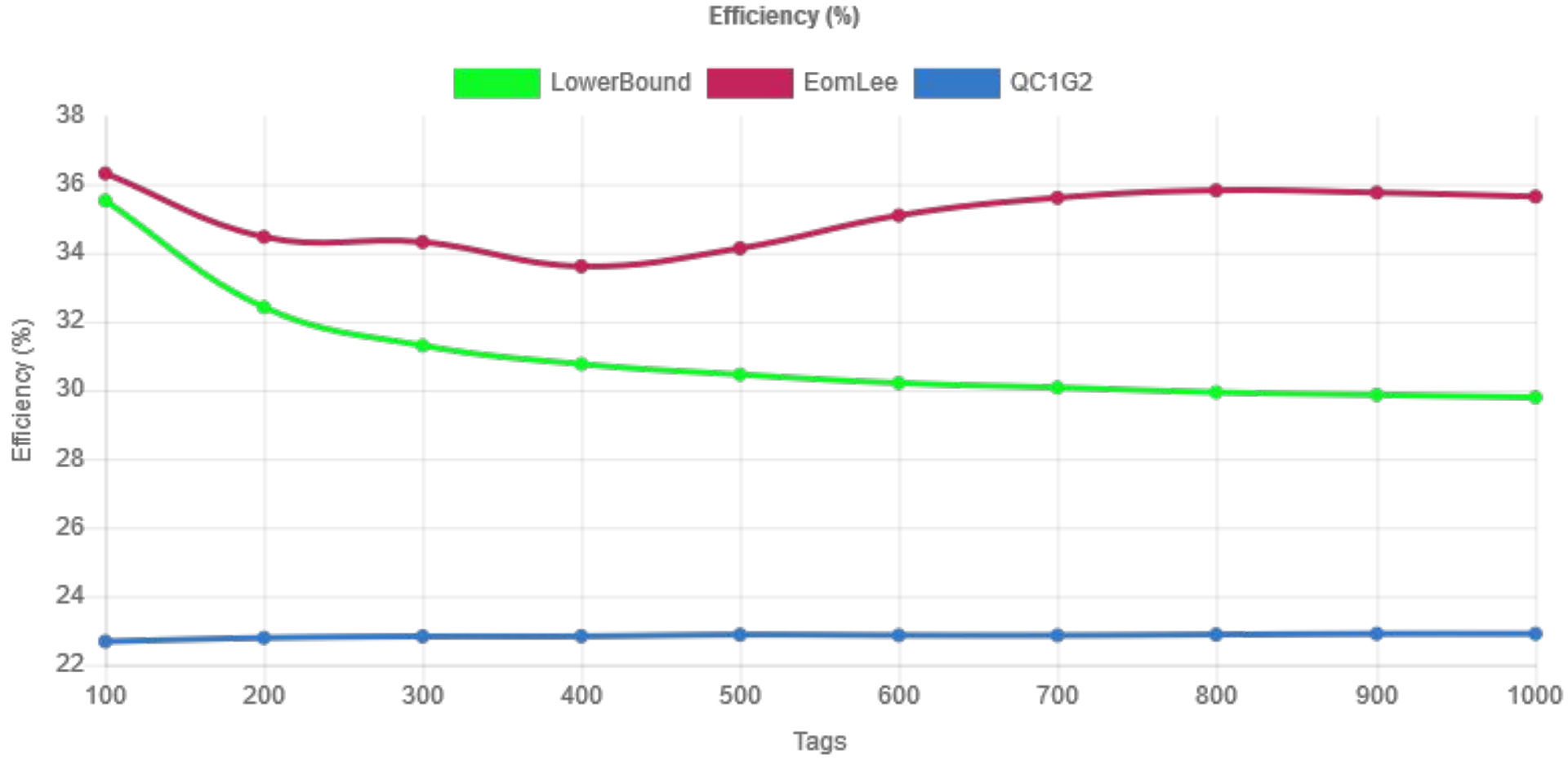

Efficiency (%)
LowerBound
EomLee
37
36
35
34
33
32
31
30
29
Efficiency (%)
100
200
300
400
500
600
700
800
900
1000
Tags

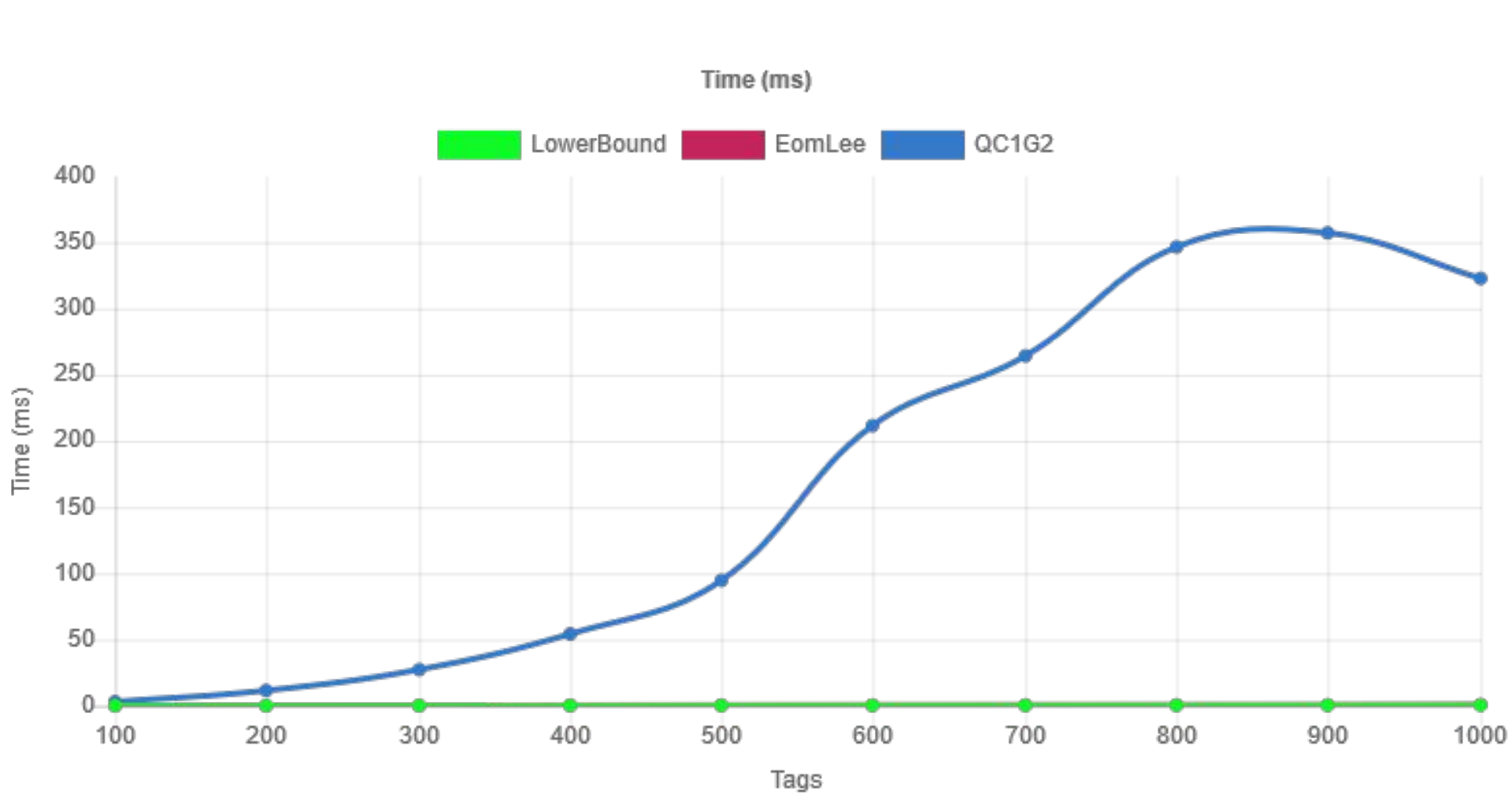
Algoritmo Q


Total Slots
LowerBound
EomLee
QC1G2
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Total Slots
100
200
300
400
500
600
700
800
900
1000
Tags


Empty Slots
LowerBound
EomLee
QC1G2
1800
1600
1400
1200
1000
800
600
400
200
0
Empty Slots
100
200
300
400
500
600
700
800
900
1000
Tags


Collision Slots
LowerBound
EomLee
QC1G2
1800
1600
1400
1200
1000
800
600
400
200
0
Collision Slots
100
200
300
400
500
600
700
800
900
1000
Tags


Efficiency (%)
LowerBound
EomLee
QC1G2
38
36
34
32
30
28
26
24
22
Efficiency (%)
100
200
300
400
500
600
700
800
900
1000
Tags


Time (ms)
LowerBound
EomLee
QC1G2
400
350
300
250
200
150
100
50
0
Time (ms)
100
200
300
400
500
600
700
800
900
1000
Tags

Obrigado