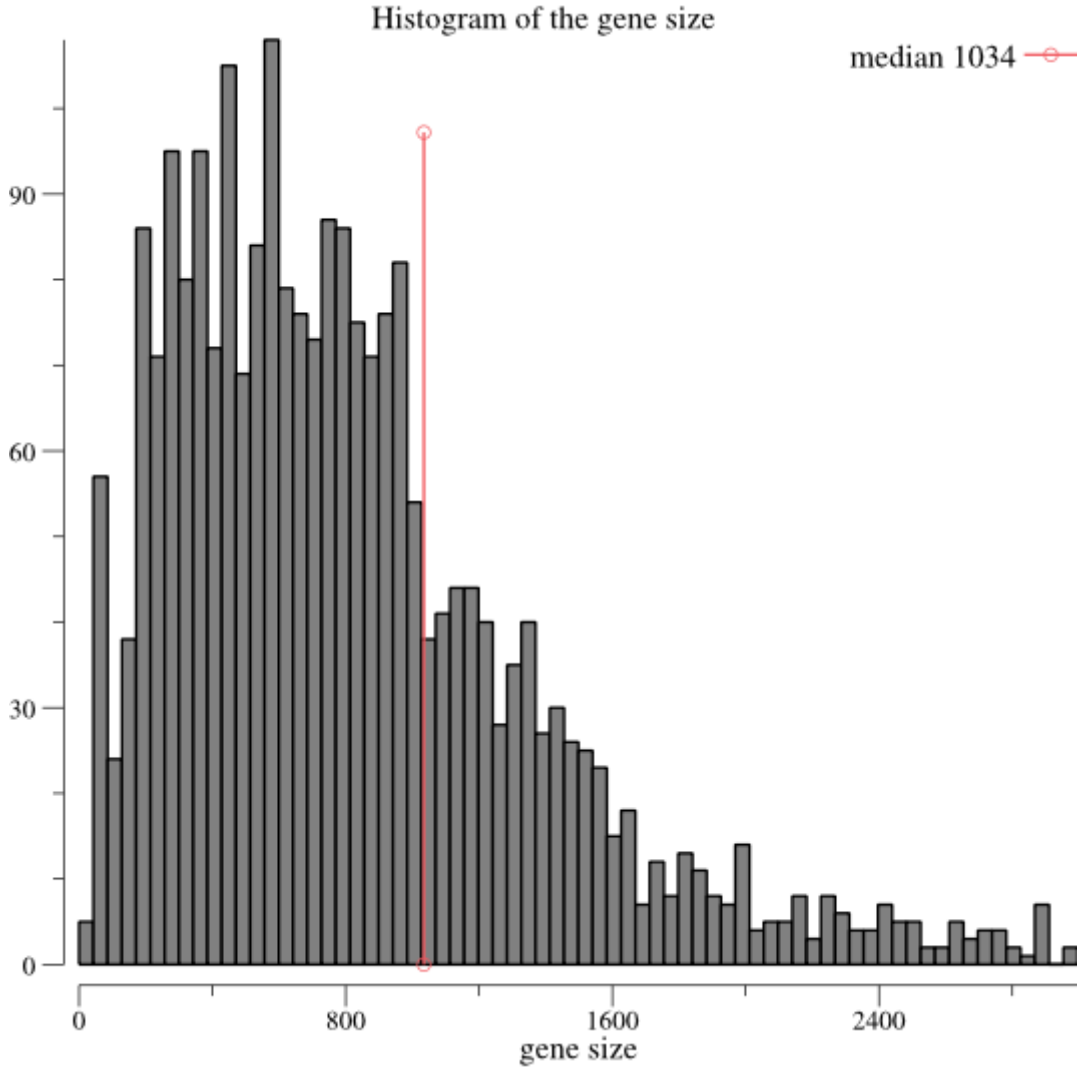
```go
// ExtractCDS project main.go
package main

import (
    "bufio"
    "flag"
    "fmt"
    "github.com/gonum/plot"
    "github.com/gonum/plot/plotter"
    "github.com/gonum/plot/plotutil"
    "os"
    "path/filepath"
    "runtime"
    "strconv"
    "strings"
    "time"
)
const MaxGeneLine int = 400
const MaxGeneLength int = 3000

func MaxParallelism() int {
    maxProcs := runtime.GOMAXPROCS(0)
    numCPU := runtime.NumCPU()
    if maxProcs < numCPU {
        return maxProcs
    }
    return numCPU
}

//compute duration
func elapsedTime(s string, startTime time.Time) time.Duration {

    return time.Since(startTime)
}

//Extract a token gene with the start/stop value
func TokenGene(data []byte, atEOF bool) (advance int, token []byte, err
error) {
    advance, token, err = bufio.ScanLines(data, atEOF)
    if err == nil && token != nil {

        f := func(c rune) bool {
            return c == '.' || c == ' '
        }
        // Separate into fields with func.
        fields := strings.FieldsFunc(string(token), f)
        if len(fields) > 0 && (fields[0] == "gene") {
            return
        } else {
            return advance, nil, nil
        } //we return empty toke, but take care of "advance"
    }
    return
}

```

```
//scan the gbk file and compute an histogram about gene distribution
func processFile(s string, shortTreament bool) plotter.XYs {
    file, _ := os.Open(s)

    reader := bufio.NewReader(file)
    scanner := bufio.NewScanner(reader)

    scanner.Split(TokenGene)
    // Validate the input
    lineNumber := 0
    //shortTreament := true
    i_start := 0
    i_stop := 0
    local_distrib := make(plotter.XYs, MaxGeneLength) //MaxGeneLength
//taille max
    for scanner.Scan() {
        //fmt.Printf("Text: %s\n", scanner.Text())
        lineNumber++
        f := func(c rune) bool {
            return c == '.' || c == ' ' || c == '(' || c == ')'
        }
        // Separate into fields with func.
        fields := strings.FieldsFunc(string(scanner.Text()), f)
        //fmt.Println(fields)
        if len(fields) > 2 {
            if fields[1] == "complement" {
                i_stop, _ = strconv.Atoi(fields[2])
                i_start, _ = strconv.Atoi(fields[3])
            } else {
                i_stop, _ = strconv.Atoi(fields[2])
                i_start, _ = strconv.Atoi(fields[1])
            }

            delta_i := i_stop - i_start
            //check order
            if delta_i < 0 {
                delta_i = -delta_i
            }
            //limit gene size to 3000
            if delta_i < MaxGeneLength {
                //fmt.Println(delta_i)
                local_distrib[delta_i].X = float64(delta_i)
                local_distrib[delta_i].Y++
            }
        }
        //fmt.Printf("%d %d\n", i_start, i_stop)
        if shortTreament && (lineNumber > MaxGeneLine ) {
            break
        }
    }
    return local_distrib

}

//draw the histogram and the median
```

```
111 func drawHistogramGene(local_v plotter.XYs, filePathName string) {
112
113     // Make a plot and set its title.
114     p, err := plot.New()
115     if err != nil {
116         panic(err)
117     }
118
119     p.Title.Text = "Histogram of the gene size"
120
121     h, err := plotter.NewHistogram(local_v, 70)
122
123     mediandist := make([]float64, len(local_v))
124     var j int
125     for _, val := range local_v {
126         if val.X > 0 {
127             mediandist[j] = val.X
128             j++
129         }
130     }
131     p.Legend.Top = true
132
133     medMinMax, _, _ := plotutil.MedianAndMinMax(mediandist[:j])
134     _, _, _, ymax := h.DataRange()
135     p.Add(h)
136     plotutil.AddLinePoints(p, "median "+strconv.Itoa(int(medMinMax)),
137 plotter.XYs{{medMinMax, 0.0}, {medMinMax, (0.9 * ymax)}})
138
139     // The normal distribution function
140
141
142     p.X.Label.Text = "gene size"
143
144     _, filename := filepath.Split(filePathName)
145     var extension = filepath.Ext(filename)
146     var name = filePathName[0 : len(filePathName)-len(extension)]
147     //plot in a png file
148     if err := p.Save(400, 400, name+"_gene_hist.png"); err != nil {
149         panic(err)
150     }
151 }
152
153 func main() {
154     var defaultfile string
155     fmt.Printf("MaxThread %d NumCPU %d \n", runtime.GOMAXPROCS(0),
156 runtime.NumCPU())
157     startTime := time.Now()
158
159     defaultfile = `E:\golang-test\NC_002662.gbk`
160     var imputFile = flag.String("filename", defaultfile, "the filepath
161 to process - gbk format")
162     var shortTreatment = flag.Bool("shortprocess", false, "shortprocess
163 (boolean) will treat only the first "+strconv.Itoa(MaxGeneLine)+" gene
164 values")
165
```

```
166     flag.Parse()
167
168     _, err := os.Stat(*imputFile)
169     if err != nil {
170         fmt.Fprintf(os.Stderr, "the file %s doesn't exist! \n",
171 *imputFile)
172         os.Exit(1)
173     }
174     local_plot_values := processFile(*imputFile, *shortTreatment)
175     drawHistogramGene(local_plot_values, *imputFile)
176     durationTime := elapsedTime("prog", startTime)
177     fmt.Printf("time elapsed %s \n", durationTime)
178 }
179
```



180