// 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

**func** drawHistogramGene(local\_v plotter.XYs, filePathName string) {

    // Make a plot and set its title.

    p, err := plot.New()

**if** err != nil {

        panic(err)

    }

    p.Title.Text = "Histogram of the gene size"

    h, err := plotter.NewHistogram(local\_v, 70)

    mediandist := make([]float64, len(local\_v))

**var** j int

**for** \_, val := **range** local\_v {

**if** val.X > 0 {

            mediandist[j] = val.X

            j++

        }

    }

    p.Legend.Top = true

    medMinMax, \_, \_ := plotutil.MedianAndMinMax(mediandist[:j])

    \_, \_, \_, ymax := h.DataRange()

    p.Add(h)

    plotutil.AddLinePoints(p, "median "+strconv.Itoa(int(medMinMax)), plotter.XYs{{medMinMax, 0.0}, {medMinMax, (0.9 \* ymax)}})

    // The normal distribution function

    p.X.Label.Text = "gene size"

    \_, filename := filepath.Split(filePathName)

**var** extension = filepath.Ext(filename)

**var** name = filePathName[0 : len(filePathName)-len(extension)]

    //plot in a png file

**if** err := p.Save(400, 400, name+"\_gene\_hist.png"); err != nil {

        panic(err)

    }

}

**func** main() {

**var** defaultfile string

    fmt.Printf("MaxThread %d NumCPU %d \n", runtime.GOMAXPROCS(0), runtime.NumCPU())

    startTime := time.Now()

    defaultfile = `E:\golang-test\NC\_002662.gbk`

**var** imputFile = flag.String("filename", defaultfile, "the filepath to process - gbk format")

**var** shortTreatment = flag.Bool("shortprocess", false, "shortprocess (boolean) will treat only the first "+strconv.Itoa(MaxGeneLine)+" gene values")

    flag.Parse()

    \_, err := os.Stat(\*imputFile)

**if** err != nil {

        fmt.Fprintf(os.Stderr, "the file %s doesn't exist! \n", \*imputFile)

        os.Exit(1)

    }

    local\_plot\_values := processFile(\*imputFile, \*shortTreatment)

    drawHistogramGene(local\_plot\_values, \*imputFile)

    durationTime := elapsedTime("prog", startTime)

    fmt.Printf("time elapsed %s \n", durationTime)

}

