The LPBasic module

About LPBasic.jl

LPBasic.jl is a julia module to facilitate JuMP framework's usage for an specific line planning optimization model called LPBasic. It was developed as part of a graduation project at PUC-Rio by Computer Science student André Mazal Krauss during 2019.

For more information visit the project's github page and read the report(in portuguese). You may also contact me directly: amk1710@gmail.com

LPBasic.jl Documentation

Dependencies

The following list includes all direct depencies for the full module's functionality. These dependencies might have their own dependencies, but Julia's Pkg should resolve them automatically.

Also note that OpenStreetMapPlot is not included in Julia's Registry. However, the package's website has straighforward install instructions. It's only use is to render transport networks with OpenStreetMap information(methods PlotInstance and PlotSolution), so it is possible to use the module without it if so desired.

- CSV (https://github.com/JuliaData/CSV.jl)
- Colors (https://github.com/JuliaGraphics/Colors.jl)
- JuMP (https://github.com/JuliaOpt/JuMP.jl)
- LightGraphs(https://github.com/JuliaGraphs/LightGraphs.jl)
- Luxor(https://github.com/JuliaGraphics/Luxor.jl)
- MathOptInterface(https://github.com/JuliaOpt/MathOptInterface.jl)
- MetaGraphs(https://github.com/JuliaGraphs/MetaGraphs.jl)
- OpenStreepMapX(https://github.com/pszufe/OpenStreetMapX.jl)
- OpenStreetMapXPlot(https://github.com/pszufe/OpenStreetMapXPlot.jl)
- Plots(https://github.com/JuliaPlots/Plots.jl)

Function Index

- `Main.LPBasic.Line`

- Main.LPBasic.ProblemInstance
- Main.LPBasic.BasicStats
- Main.LPBasic.CalculateCost
- Main.LPBasic.ConstructModel
- Main.LPBasic.GreatCircleDistance
- Main.LPBasic.IsEdgeOnLine
- Main.LPBasic.PlotInstance
- Main.LPBasic.PlotSolution
- Main.LPBasic.ReadGTFS
- Main.LPBasic.ReadNatureFeed
- Main.LPBasic.RunSuite
- Main.LPBasic.VerifyInstance
- Main.LPBasic.VerifySolution

Module

LPBasic — Module

```
LPBasic Module
```

Julia module implementing the LPBasic line planning problem.

This includes:

- defining structs for containing the relevant problema data
- implementing IO functionality to read relevant data from GTFS data into such structs
- a flexible functionality to construct JuMP models from such structs
- some visualization functionality

Data Structures

Main.LPBasic.Line — Type

```
Line
```

A basic data structure for model construction. Represents a directed walk through a transportation network

Constructor:

```
Line(id::Union{Int64, String}, walk::Array{Tuple{Int64, Int64}, 1}, frequency::
```

Fields:

- id::Union{Int64, String} - an Int64 or String to help distinguis this lines from other lines. Uniqueness is not enforced
- walk::Array{Tuple{Int64, Int64}, 1} - an array of edge indices, representing a directed walk through the transportation network
- frequency::Int64 - the base frequency for this line
- cost::Float64 - the cost per frequency unit of this line

`Main.LPBasic.ProblemInstance` — Type

```
ProblemInstance
```

Struct containing all data of a single LPBasic problem's instance.

Constructor:

```
ProblemInstance(network::MetaDiGraph, lines::Array{Line,1})
```

Fields:

- network::MetaDiGraph - representation the transportation network
- lines::Array{Line,1} - array containing all lines to be considered in the optimization

IO Functionality

`Main.LPBasic.ReadGTFS` — Function

```
ReadGTFS(directory_path, transportModeCode=3)::ProblemInstance
```

Read the data in an uncompressed GTFS feed into a ProblemInstance struct.

The transportModeCode parameter defines which mode of transportation should be considered, with the default value = 3 indicating buses. Another common value would be 1 for trains.

`Main.LPBasic.ReadNatureFeed` — Function

```
ReadNatureFeed(directory_path)::ProblemInstance
```

Read data from a feed as specified in this article

Model Construction

`Main.LPBasic.ConstructModel` — Function

```
function ConstructModel(instance::ProblemInstance; <keyword arguments>)::Model
```

Construct a JuMP model from the LPBasic problem instance and return it.

Arguments

- `useSlack::Bool=false`: whether to use slackened constraints. If used, slack_penalty must be defined
- `slack_penalty::Float64`: the penalty taken to the objective-value for each unit of slack taken
- `useBinaryVariant::Bool=false`: whether to use binary decision variables
- `minFrequencyMultiplier::Number=0.5`: the multiplier used to estimate an edge's minimum frequency requirement from the edge's base frequency
- `maxFrequencyMultiplier::Number=2.0`: the multiplier used to estimate an edge's maximum frequency requirement from the edge's base frequency
- `useNonIntegerDecisionVariables=false`: whether to use relaxed, non-integer decision variables

Visualization

`Main.LPBasic.PlotInstance` — Function

```
function PlotInstance(instance::ProblemInstance, osmpath::String)
```

Return a plot of the given instance using the given Open Street Map information.

Substantially slower than the PlotInstance method not using OSM information, but the result is better looking.

```
PlotInstance(instance::ProblemInstance; out_path::String = "./", should_preview
```

Return a simple Luxor drawing of the given instance

Substantially faster than the PlotInstance method using Open Street Map information, but draws only simple lines on a blank background.

`Main.LPBasic.PlotSolution` — Function

```
PlotSolution(instance::ProblemInstance, solution::Array{Bool, 1}, osmpath::Str
```

Identical to PlotInstance(instance::ProblemInstance, osmpath::String), but paint unused lines differently

Substantially slower than the PlotSolution method not using OSM information, but the result is better looking.

```
PlotSolution(instance::ProblemInstance, solution::Array{Bool, 1};  <keyword arg
```

Return a simple Luxor drawing of the given instance, painting lines used in solution differently

Keywork arguments:

- [out_path]: optional path determining where to save the image. If this isn't set, a time-stamped name will be used
- should_preview=true: should the image be previewed in the default image viewer?

Miscellaneous

`Main.LPBasic.IsEdgeOnLine` — Function

```
IsEdgeOnLine(instance::ProblemInstance, edge::LightGraphs.SimpleGraphs.SimpleE
```

Check if the a given *edge* is contained in a the line indexed by *lineIndex* in the given LPBasic problem *instance*.

Returns:

- true/false
- 1/0, if numerical_returns == true

Implementation:

- Has O(1) time complexity, by using an internal dictionary pairing each edge to its respective lines

---

`Main.LPBasic.CalculateCost` — Function

```
CalculateCost(instance::ProblemInstance, solution::Array{Bool,1})::Float64
```

Calculate total line cost for given solution on given instance.

A solution for this method is an array of booleans indicating accepting or rejecting a line with its basic frequency

```
CalculateCost(instance::ProblemInstance, solution::Array{Bool,1})::Float64
```

Calculate total line cost for given solution on given instance.

A solution for this method is an array of integers representing each line's taken frequency

---

`Main.LPBasic.RunSuite` — Function

```
RunSuite(gtfs_path::String, outpath::String = "./")
```

Run through the optimization pipeline reading data from gtfs_path and outputting to outpath

Can be used as a sort of example script for the module's usage.

`Main.LPBasic.BasicStats` — Function

```
BasicStats(instance::ProblemInstance, solution::Array{Bool,1})
```

Boolean/binary variant of the BasicStats function.

In this solution, each line has 0 frequency or its base-frequency

```
BasicStats(instance::ProblemInstance, solution::Array{T,1} where {T<:Integer})
```

Populate and return a dictionary with basic information about the instance and solution.

The returned dictinary has the following key/value pairings:

- QntLines: the quantity of lines considered in the instance
- QntStops: the quantity of stops/nodes in the transportation network/graph
- QntEdges: the quantity of conections/edges in the transportation network/graph
- TakenLines: the quantity of lines used in the solution(taken frequency > 0)
- AverageFrequency: the average frequency of service in all conections/edges
- StandardDeviation: the standard deviation for the average above
- Variance: the variance for the average above
- Median: the median value for the average above
- LowestFrequency: the lowest frequency of service among all conections/edges
- HighestFrequency: the highest frequency of service among all conections/edges
- LineCost: the sum of costs for taking all lines in the solution with the given frequency

`Main.LPBasic.VerifyInstance` — Function

```
VerifyInstance(network::MetaDiGraph, lines::Array{Line,1})
VerifyInstance(instance::ProblemInstance)
```

Verify if a given problem instance is valid per the LPBasic requirements. Raise error if there are inconsistensies, otherwise return true.

This includes checking if:

- all lines are unique(unique id, unique walk, unique frequency)
- each edge has defined its expected properties
- there are no invalid values for frequencies and weights

to-do:

- is the network/graph is connected?(is there is a path between every pair of vertices?)
- redundant information matches one another? (:line_coverage)

return value:

- true - indicating that the instance is valid, or

- *raise error* - indicating that the instance is not valid and the reason why

```
VerifyInstance(instance::ProblemInstance)
```

Do the same as VerifyInstance(instance.network, instance.lines)

`Main.LPBasic.VerifySolution` — Function

```
VerifySolution(instance::ProblemInstance, solution::Array{Bool,1})
```

Verify if a given solution is feasible per the LPBasic requirements.

Also check for construction errors in the instance and solution itself Return a tuple (true,"Valid") or (false, <error_string>)

`Main.LPBasic.GreatCircleDistance` — Function

```
GreatCircleDistance(latitude1::Number, longitude1::Number, latitude2::Number,
```

Calculate the great circle distance between points (latitude1, longitude1) and (latitude2, longitude2) on the earth's surface.

Implementation

Uses the Haversine formula(https://en.wikipedia.org/wiki/Haversine_formula) supposing the Earth is perfectly round

Authoring information

Author: André Mazal Krauss, Computer Sciences student at Pontifícia Universidade Católica do Rio Janeiro(PUC-Rio)

Project Supervision: Marcus Poggi