LifT_Solver Manual

July 3, 2020

1 Cloning the repository

1. Type `git clone https://github.com/AndreaHor/LifT_Solver.git` for cloning the repository.
2. Type `git submodule update --remote --recursive --init` for getting the submodules.

2 Setting up Gurobi

1. Download Gurobi from `www.gurobi.com` and get the license.
2. Go to `LifT_Solver/solverILP/cmake/modules/FindGUROBI.cmake` and make sure that your Gurobi version matches the version in this file. If not, modify the respective parts of the file to match your Gurobi version.
3. Set up your system variables to contain paths to Gurobi as described below.

Linux Open `.bash_profile` in your home directory and add the following lines containing location of your files:

```
export GUROBI_HOME="/your/path/to/gurobi/gurobi902/linux64"
export PATH="$PATH:$GUROBI_HOME/bin"
export LD_LIBRARY_PATH="$GUROBI_HOME/lib"
export GRB_LICENSE_FILE="/your/path/to/gurobi/license/gurobi.lic"
```

You need to rebbot / logout and login after you do the changes.

Possible problems If you are not able to compile LifT_Solver because of some Gurobi errors although you followed the above lines, it may be due to incompatible gcc version. You can try the following:
Open terminal, go to `gurobi902/linux64/src/build` and run `make` here. You obtain a new file `libgurobi_c++.a`. Go to `gurobi902/linux64/lib` and replace the old file `libgurobi_c++.a` with the one that you generated. **Note:** You should back up the original `libgurobi_c++.a` before replacing it!

3 Program Use

After successful compilation, you should obtain executable file `run-disjoint-paths`. You can run the solver by calling, `./run-disjoint-paths -c params_sequence.ini`. Instead of `params_sequence.ini`, you can use the path to your parameter file. You find an example parameter file in directory `examples`.

3.1 Parameter File

Due parameters

- [SOLVER]
  A keyword that must be placed in the file before the other parameters.
- `INPUT_GRAPH = /home/LifT_Solver/examples/ex1/problemDesc`
  Path to the file with input graph.

Optional parameters

- `INPUT_FRAMES=/home/LifT_Solver/examples/ex1/problemDesc_frames`
  Input file with time frame information. If this parameters is not provided, its value is set to the value of INPUT_GRAPH with suffix "_frames".
- `OUTPUT_PATH = /home/LifT_Solver/examples/ex1/`
  Path to a directory where the output files will be stored. If this parameter is not provided, it is set to the path where the input graph is stored. Do not forget the slash at the end of the string.
- `OUTPUT_PREFIX = output`
  Multiple output files will be produced. This is their common prefix. If not set, the default value `output` will be used.
- `DEBUG_OUTPUT_FILES = 0`
  Expects 0/1 values. If set to 1, the solver outputs preliminary feasible solutions obtained during its run. Default value is 0.
- `SPARSIFY = 1`
  Expects value 0/1. If set to 0, no sparsification is done, so parameters related to sparsificatioin are not needed. If set to 1, sparsification is done. Default and recommended value is 1.
- `MAX_TIMEGAP = 900`
  Expects a positive integer. Maximum number of time frames to be used for the input. Only frames 1,2,...,MAX_TIMEGAP are used for the computation. If you want to use all time frames, either leave this parameter out or set it to zero or to a sufficiently large value.
- `INPUT_COST = 0.0`
  Expects a real value. Cost of starting a track. Default value is 0.
- `OUTPUT_COST =0.0`
  Expects a real value. Cost of terminating a track. Default value is 0.

- `SMALL_INTERVALS = 40`
  Expects a non-negative integer stating the amount of time frames used for creating sub-tasks (step one in the two-step procedure). If set to 0, solver operates on the whole graphs as they are given (after their sparsification). If set to a higher value, graphs are divided into smaller intervals of the given length. Obtained solution on intervals is used for creating tracklets. These tracklets become nodes of newly created graphs. The new graphs are used for further optimization. Default value is 40.

- `GUROBI_REL_GAP = 0`
  Expects a non-negative real value. Setting Gurobi relative gap. Influences solutions where the problem graph nodes correspond one to one to the input graph nodes.

- `GUROBI_REL_GAP_TRACKLET = 0`
  Expects a non-negative real value. Setting Gurobi relative gap. Influences solution where the problem graph nodes correspond to tracklets.

Base graph sparsificatioin parameters

- `MAX_TIMEGAP_BASE = 50`
  Expects a positive integer. Maximal time gap for the base edges (amount of time frames).

- `KNN_K = 3`
  Expects a positive integer. For every vertex and every time gap, up to `KNN_K` nearest neighbors can be connected by a base edge with that vertex.

- `KNN_GAP = 6`
  Expects a non-negative integer. Up to `KNN_GAP`, all found nearest neighbors are connected with a processed vertex. Beyond `KNN_GAP`, edges to nearest neighbors have to additionally satisfy other constraints in order to be used.

- `BASE_THRESHOLD = 0.0`
  Expects a real value. Upper threshold for cost of base edges with a bigger time gap than `KNN_GAP`. Edges longer than `KNN_GAP` have to have cost lower than this threshold in order to be used. Default value is 0.

Lifted graph sparsification parameters

- `MAX_TIMEGAP_LIFTED = 60`
  Expects a non-negative integer. Maximal time gap for the lifted edges.

- `DENSE_TIMEGAP_LIFTED =4`
  Expects a non-negative integer. Every lifted edge with time gap lower or equal to this value will be added if it satisfies the constraints on its cost value given by `NEGATIVE_THRESHOLD_LIFTED` and `POSITIVE_THRESHOLD_LIFTED`.

- `NEGATIVE_THRESHOLD_LIFTED = -0.5`
  Expects a non-positive real value. Lifted edges with negative cost will only be added if their cost value is lower or equal to this number.

- `POSITIVE_THRESHOLD_LIFTED = 0.5`
  Expects a non-negative real value. Lifted edges with positive cost will only be added if their cost value is higher or equal to this number.

- `LONGER_LIFTED_INTERVAL = 4`
  Expects a positive integer. Influences lifted edges with time gap between `DENSE_TIMEGAP_LIFTED` and `MAX_TIMEGAP_LIFTED`. If set to value n, only every n-th time gap will be used for adding lifted edges.

- `TRACKLET_SIZE=20`
  Expects a non-negative integer. Maximal length of tracklets used in the second step of our two-step procedure. After finishing the first step, resulting tracks are further cut in time frames given by multiples of `TRACKLET_SIZE`. If this parameter is set to 0, no splitting after the first step is done and the whole tracks resulting from the first step are used as tracklets. Default value is equal to `SMALL_INTERVALS`.

- `ALL_BASE_TRACKLET=1`
  Expects 0/1. If set to 1, all base edges from the input graph (before sparsification) can be used for connecting two tracklets. If set to zero, only those tracklets can be connected with a new base edge, if the cumulative cost between them is lower or equal to `BASE_THRESHOLD`.

- `DENSE_TRACKLETS = 1`
  Expects 0/1. It has influence only if `SMALL_INTERVALS=0`. That is, when problem is solved directly on the whole input graphs without dividing into intervals. If set to zero, no second step is performed. If set to one, resulting tracks are cut into tracklets and the second step of the two-step procedure is run on the obtained tracklet graphs.

- `OPTIMIZE_PATHS = 0`
  Expects 0/1. Influences creating tracklets from tracks. In order to understand this parameter, first see the "Note to sparsification" below this parameter list. In the second step, all edges of the input graphs up to certain time gap are used for evaluating cost and enable connections between tracklets. Therefore, it may be beneficial to split some tracks with respect to this more accurate evaluation. If this parameter is one, optimal splitting is searched for every initial track. If set to zero, only simple search for advantageous break points is done.

- `COMPLETE_GAP_IN_TRACKLET= 1`
  Expects 0/1. Influences how long edges are used for creating tracklet graph and for evaluating cost between tracklets. If set to one, all base and lifted edges up to `MAX_TIMEGAP_COMPLETE` are used. If set to zero, `MAX_TIMEGAP_BASE` and `MAX_TIMEGAP_LIFTED` are used for selecting the base and the lifted edges.

- `MAX_TIMEGAP_COMPLETE = 60`
  Expects a positive integer. Influences the second step of the two-step procedure. If `COMPLETE_GAP_IN_TRACKLET` is set to one, this value is used for selection of base and lifted edges in creating the tracklet graph. Its default value is the maximum from `MAX_TIMEGAP_BASE` and `MAX_TIMEGAP_LIFTED`.

- `INTERVAL_FILE=/home/LifT_Solver/examples/ex1/output-all-paths-INTERVALS.txt`
  Can be used for loading the results of the first step and directly running the second step. For this scenario, parameter `TASK` must be set to `T`. Expects path to a file where results of the first step are stored. These files have suffix `"-all-paths-INTERVALS.txt"`.

- `TASK=C`
  Expects either character `C` for performing the complete task. That is both steps of the two-step procedure or `T` for only performing the second step. In this case, `INTERVAL_FILE` must be set to a valid path to a file.

Note to sparsification The above mentioned sparsification parameters influence only the first step of our two-step procedure. In the second step, aggregated costs between two tracklets are obtained by summing up all relevant lifted edges up to `MAX_TIMEGAP_LIFTED` plus cost of the base edge connecting the last vertex of the first tracklet with the first vertex of the second tracklet.

3.2 Graph File

- First line contains one number: number of vertices (without counting s and t).
- Second line must be empty!
- Further lines contain definitions of edges.

Note that two special vertices s and t are added automatically. Vertices are numbered from zero. If the given number of vertices is n, then vertex s has index n and vertex t has index $n + 1$. Edges from s to all vertices are automatically added to G. Edges from all vertices to t are automatically added to G too. The file has to contain just edges between vertices $0, \ldots, n - 1$.

Defining edges Each edge is defined on a separate line. Each line must contain three comma separated values: Index of the first vertex, index of the second vertex, edge cost.

3.3 File with Time Frames

This file must provide list of vertices that belong to each time frame. Every vertex must be on a separate line. Every line must contain two comma separated values. The first value is the index of a graph vertex (numbered from zero), the second is the index of its time frame (numbered from one). The vertices must be stated in ascending order. That is, first line contains vertex 0, second line contains vertex 1, etc.