LOMAP: LTL Optimal Multi-Agent Planner

**Overview**

**L**TL **O**ptimal **M**ulti-Agent **P**lanner (LOMAP) is a python package for automatic planning of optimal paths for multi-agent systems. LOMAP includes the implementations of the following algorithms:

* policy\_synthesis algorithm presented in: A. Ulusoy, M. Marrazzo, K. Oikonomopoulos, R. Hunter, and C. Belta, “Temporal Logic Control for an Autonomous Quadrotor in a Nondeterministic Environment,” *submitted*.
* multi\_agent\_optimal\_run and robust\_multi\_agent\_optimal\_run algorithms presented in: A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, “Optimality and robustness in multi-robot path planning with temporal logic constraints,” *submitted*.
* robust\_multi\_agent\_optimal\_run algorithm presented in: A. Ulusoy, S. L. Smith, and C. Belta, “Optimal Multi-Robot Path Planning with LTL Constraints: Guaranteeing Correctness Through Synchronization,” *Intl. Symp. on Distributed and Autonomous Robotic Systems*, Baltimore, MD, USA.
* incremental\_policy\_synthesis algorithm presented in: A. Ulusoy, T. Wongpiromsarn, and C. Belta, “Incremental Control Synthesis in Probabilistic Environments with Temporal Logic Constraints,” *IEEE Conf. on Decision and Control*, Maui, HI, USA, 2012.
* optimal\_run algorithm presented in: S. L. Smith, J. Tumova, C. Belta, and D. Rus, “Optimal Path Planning for Surveillance with Temporal Logic Constraints,” *International Journal of Robotics Research*, 30(14), pp. 1695-1708, 2011.

**How to Install**

The recommended way to install the LOMAP package is to type

pip install --allow-external pp --allow-unverified pp lomap

at the shell. Depending on your python installation you may need to execute the above command as the root, i.e. enter the following instead:

sudo pip install --allow-external pp --allow-unverified pp lomap

Once the installation finishes you will see the following message:

###

# LOMAP has been installed to PYTHON\_PKG\_DIR/lomap.

# To run the examples, copy the contents of PYTHON\_PKG\_DIR/lomap/examples

# to a writable directory.

###

where PYTHON\_PKG\_DIR is the Python package directory of your system. The examples folder includes three examples that demonstrate how to use this package. You will need to copy the contents of this directory to a writable directory to run them.

The source code of LOMAP is available at the [python package index](http://pypi.python.org/pypi/lomap). If you run into problems installing the package please make sure that your system meets the requirements given below.

Requirements

* NetworkX package (pip should take care of this as it processes dependencies, otherwise type pip install networkx).
* ParallelPython package (pip should take care of this as it processes dependencies, otherwise type pip install --allow-external pp --allow-unverified pp pp).
* matplotlib package (pip should take care of this as it processes dependencies, otherwise type pip install matplotlib).
* setuptools package.

**How to Use**

Implementations of the algorithms are self-explanatory and, in general, it should be enough to check their respective implementations under lomap/algorithms/. You can also check the examples that come with LOMAP as mentioned above. In the following, we will discuss a script that can be used to solve the case-study in our DARS 2012 paper to illustrate the basic usage of the library. You can download the script [here](https://calinbelta.com/wp-content/uploads/2024/01/dars_2012.docx). You’ll also need the model definitions of [[robot 1](https://sites.bu.edu/hyness/files/2014/05/robot_1.txt)](https://calinbelta.com/wp-content/uploads/2024/01/robot_1.docx) and [[robot 2](https://sites.bu.edu/hyness/files/2014/05/robot_2.txt)](https://calinbelta.com/wp-content/uploads/2024/01/robot_2.docx).

**Model Definitions**

If you check the contents of the robot\_1.txt file:

name robot-1

init {'i1':1}

;

i18 {'prop':{'gather21','gather2','gather'}}

i20 {'prop':{'gather22','gather2','gather'}}

i22 {'prop':{'upload2'}}

;

i1 i2 {'weight':1, 'control':'x'}

i2 i3 {'weight':1, 'control':'x'}

i2 i5 {'weight':4, 'control':'x'}

i3 i4 {'weight':1, 'control':'x'}

i3 i14 {'weight':4, 'control':'x'}

i4 i1 {'weight':1, 'control':'x'}

i5 i6 {'weight':1, 'control':'x'}

i6 i7 {'weight':1, 'control':'x'}

i7 i8 {'weight':1, 'control':'x'}

i7 i21 {'weight':2, 'control':'x'}

i8 i5 {'weight':1, 'control':'x'}

i8 i17 {'weight':2, 'control':'x'}

i9 i8 {'weight':4, 'control':'x'}

i9 i10 {'weight':1, 'control':'x'}

i10 i11 {'weight':1, 'control':'x'}

i11 i12 {'weight':1, 'control':'x'}

i12 i9 {'weight':1, 'control':'x'}

i12 i15 {'weight':4, 'control':'x'}

i13 i14 {'weight':1, 'control':'x'}

i13 i4 {'weight':4, 'control':'x'}

i14 i15 {'weight':1, 'control':'x'}

i14 i19 {'weight':2, 'control':'x'}

i15 i16 {'weight':1, 'control':'x'}

i16 i13 {'weight':1, 'control':'x'}

i17 i3 {'weight':2, 'control':'x'}

i17 i18 {'weight':1, 'control':'x'}

i18 i17 {'weight':1, 'control':'x'}

i19 i9 {'weight':2, 'control':'x'}

i19 i20 {'weight':1, 'control':'x'}

i20 i19 {'weight':1, 'control':'x'}

i21 i10 {'weight':2, 'control':'x'}

i21 i22 {'weight':1, 'control':'x'}

i22 i21 {'weight':1, 'control':'x'}

the file consists of three sections separated by semicolons. The first section gives the name of the model and the initial state. The second section gives the set of propositions that are observed at each state. Finally, the last section defines the edges of the model, along with weight and control information. The contents of the robot\_2.txt file are similar.

**Calling LOMAP Methods**

Next, we will briefly discuss the contents of dars\_2012.py file to show how one can use the model definitions and the LOMAP package to actually solve a planning problem

#!/usr/bin/env python

import lomap

import sys

import logging

# Classes derived from namedtuple

from collections import namedtuple

Rho = namedtuple('Rho', ['lower', 'upper'])

def main():

 with lomap.Timer('File I/O'):

 r1 = lomap.Ts()

 r2 = lomap.Ts()

 # Case-Study

 r1.read\_from\_file('robot\_1.txt')

 r2.read\_from\_file('robot\_2.txt')

 formula = '[](gather1 -> X(!gather1 U upload1)) && [](gather2 -> X(!gather2 U upload2)) && []((gather11->gather22)&&(gather12->gather21)&&(gather21->gather12)&&(gather22->gather11)) && []<> gather'

 opt\_prop = set(['gather'])

 with lomap.Timer('DARS 2012 Case-Study'):

 ts\_tuple = (r1, r2)

 # deviation values of agents

 rhos = [Rho(lower=0.95, upper=1.05), Rho(lower=0.95, upper=1.05)]

 prefix\_length, prefixes, suffix\_cycle\_cost, suffix\_cycles = lomap.robust\_multi\_agent\_optimal\_run(ts\_tuple, rhos, formula, opt\_prop)

 print "Cost: %d" % suffix\_cycle\_cost

 print "Prefix length: %d" % prefix\_length

def config\_debug():

 # create root logger

 logger = logging.getLogger()

 logger.setLevel(logging.INFO)

 # create file handler

 fh = logging.FileHandler('lomap.log', mode='w')

 fh.setLevel(logging.DEBUG)

 # create console handler

 ch = logging.StreamHandler()

 ch.setLevel(logging.INFO)

 # create formatter and add it to the handlers

 formatter = logging.Formatter('%(levelname)s %(name)s - %(message)s')

 fh.setFormatter(formatter)

 ch.setFormatter(formatter)

 # add the handlers to the logger

 logger.addHandler(fh)

 logger.addHandler(ch)

 logger.debug('Logger initialization complete.')

if \_\_name\_\_ == '\_\_main\_\_':

 config\_debug()

 main()

We will focus on the function main(), as all config\_debug() does is to configure the logging library. As you may notice, the main() function consists of two parts. The first part creates two instances of the *transition system* object by calling lomap.Ts(), reads their definitions from their respective files, defines the formula and the optimizing proposition. You can find more information on the object types that LOMAP defines by checking the contents of the lomap/classes folder. Then, in the second part, we solve the problem by calling the robust\_multi\_agent\_optimal\_run method of the LOMAP package. Again, for more information on the algorithms that LOMAP provides, please check the lomap/algorithms folder. Then, having defined the models and written the python script to solve the problem, we execute the script by typing python dars\_2012.py, or by calling it directly if it’s executable already, to obtain an optimal run that satisfies the mission. Lomap creates a log file called lomap.log that contains the console output in case you want to examine it later.

**License & Copying**

LTL Optimal Multi-Agent Planner (LOMAP)

Copyright (C) 2012-2015, Alphan Ulusoy (alphan@bu.edu)

This program is free software; you can redistribute it and/or

modify it under the terms of the GNU General Public License

as published by the Free Software Foundation; either version 2

of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA

02110-1301, USA.

A copy of the GNU General Public License is included in the source archive available at the [python package index](http://pypi.python.org/pypi/lomap), in a file called ‘COPYING’.

LOMAP uses two third party programs: LTL2BA and scheck. You can find their sources in this distribution, in a directory called ‘third\_party\_sources’. For your convenience, binaries of these programs are already included in this distribution, in a folder called ‘binaries’. LOMAP also includes some code that is adapted and/or taken from NetworkX Python package v1.6, available at [http://networkx.github.io](http://networkx.github.io/). See the file named ‘README’ in the source archive available at the [python package index](http://pypi.python.org/pypi/lomap) for copyright notices and licenses of LTL2BA, scheck, and NetworkX.