IGRAPH Usage

David Pattison
david.pattison@strath.ac.uk

December 22, 2014

1 Overview

This document describes the IGRAPH goal recogniser and associated project dependencies. See [4, 5, 6] for more on how the underlying modelling works. Note that [6] is now not completely representative of how the plan threader works.

2 Introduction

IGRAPH is a heuristic-based goal recogniser, which allows goal recognition to be performed on PDDL level 1 input domain and problem files. Unlike other recognisers, it does not rely on a plan-library for recognition.

IGRAPH — Intermediate Goal Recognition with A Planning Heuristic (previously known as AUTOGRAPH [5], a term which appears frequently in the code), accepts input from a PDDL 2.1 [3], which should allow rapid deployment of the system and development/evaluation of associated domains.

The system uses heuristics as a means of determining which goals an agent is pursuing. This is done by breaking the set of all possible goals (including conjunctions) into multiple sets of mutually-exclusive facts, and determining which of these facts has had the most "work" put into its achievement by the observed plan.

Note that for this work, the "agent" is implicitly that which is executing the plan. That is, in a logistics problem, truck drivers are not individual agents, but rather the person/planner which generated the overall plan is. This is merely a formality and can be overlooked for most problems, but as there is much work in multi-agent recognition, it is best to have this clear from the offset. Furthermore, IGRAPH was initially intended to support multiple agents, but while this was ultimately removed, some evidence of it remains in the code (such as the use of `recogniser.learning.agent.NullAgent`) in certain places.

3 Environment

IGRAPH has largely been written under Linux and makes use of Linux-style paths for external script calls, but there is no reason it should not work on Window/Mac (see below for further details).

The recogniser has been written for comfort, and not for speed. That is, while some parts have been optimised to run quickly, others have definitely not! Therefore, memory requirements for the JVM can vary greatly, however, as a general rule 2GB should be enough for large problems (`-Xmx=2048M`).

- IGRAPH is written in Java, and should run on Java 5+.
- Strictly, Python 2 is required to run Helmert's SAS$^+$ translator, but Python 3 should also work. If you are having issues, the file you probably want to look at is under ⟨JavaSAS/IGRAPH⟩/lama/translate/translate.py.
- Perl is used to launch the SAS$^+$ translator from within Java. This is a single line of code and could easily be changed to your environment of choice.
- IGRAPH is domain independent, but the external scripts it uses are largely coded for Linux. However, I have been successful in running the system on Windows using Cygwin. A copy of the SAS$^+$ translator as a Windows 7 64-bit compiled executable. IGRAPH should automatically detect which operating system is in use and call the appropriate translator script — but is more likely to fail because of hard-coded paths to Python. See JavaSAS/src/sas/parser/SASTranslator.java for the relevant code, and change the path/script accordingly for your system.

4 Dependencies

- **JavaFF2.1** — An almost complete rewrite of the JavaFF planner [1]. Used as a baseline for IGRAPH development as many concepts are shared between planning and recognition.
- **JavaSAS** — An implementation of the SAS$^+$ formalism in Java. Uses Helmert's PDDL to SAS$^+$ translator [2] the actual translation, then parses in the resulting output to a more useful Java form. Used as a source of heuristics by IGRAPH.
- **NewPlanThreader** — A library for constructing graph-based representations of an observed plan. Used by IGRAPH as a source of "work" functions.

4.1 Third-party dependencies

- **JGraphT** — Used by IGRAPH, JavaFF2.1 and NewPlanThreader. Should be in each project's "lib" directory.
- **PDDL to SAS$^+$ Translator** — There are 2 copies of Helmert's SAS$^+$ translator. JavaSAS holds an unmodified copy of the translator [1] which is used by this project alone. IGRAPH holds a separate copy of the 2011 translator, which has been modified to prevent optimisation of the output files based on the true goal. **DO NOT MODIFY, DELETE**

[1] I *think* it is the 2008 version bundled with the original version of Lama [7]. AUTOGRAPH used the 2004 version, and since then the 2011 version has been released.

OR OVERWRITE THIS COPY! It will result in IGRAPH having a far more tightly optimised problem to solve, which will give you false positives in your results.

5 Usage

IGRAPH can be used in two manners — as a way of evaluating performance on a given domain and problem file, or as a goal recogniser embedded within your application of choice. Section 5.1 describes the former, while Section 5.2 briefly describes the latter.

In both cases, this section assumes you are using something like Eclipse for setting up paths. For IGRAPH and all its dependencies, make sure that everything under the respective "lib" directory is added to the Java classpath. Note that there may be some classes in the JAR which contain errors — these are old or deprecated code which can be ignored/removed from the PATH/deleted.

5.1 Using the IGRAPH Test Harness

If you wish to evaluate the performance of IGRAPH on your domain and problem files, the easiest way is to use `recogniser.IGRAPHTestHarness.java`, which will allow you to see how accurate hypotheses are against a known-goal.

Basic IGRAPH usage is as follows:

```
java recogniser.IGRAPHTestHarness
   ⟨domain file⟩
   ⟨problem file⟩
   ⟨solution file⟩
   ⟨output path prefix⟩
```

All output is to stdout, so it may be easier to `grep` this for what you are looking for (that is, there is a LOT of text output when executing!).

IGRAPH and the test harness have a multitude of switches for performing tests with. See (or execute) IGRAPH/src/IGRAPHTestHarness.java for a list of these, which as of December 22, 2014are as follows.

- `-heuristic {Max,FF,CG,CEA,JavaFF,Random}` — The heuristic to use during recognition. Defaults to CEA.
- `-likelihood {ML,MLThreaded,SA}` — The *work* function to use during recognition. Defaults to `SA`.
- `-filter {Greedy,Stability}` — The technique used for filtering goal hypotheses. Only Greedy is currently supported/working.
- `-minProbability [0:1]` – Unused.
- `-minStability [0:1]` – Unused.
- `-bayesLambda (0:1)` – The value for λ during Bayesian updates. Can be ultimately ignored as it should make no difference to output. Defaults to 0.8.
- `-goalSpace {Map,BDD}` — Ignored. Only `Map` is currently supported.

- `-verifyGoalSpace` {`0,1`} — If true (1), the total probability of each sub-goal-space is verified to sum to 1 after each Bayesian update. Defaults to 0.

- `-translate` {`0,1`} — Whether to do SAS$^+$ translation at runtime or not. If 0 it is assumed that the relevant SAS$^+$ files already exist in the relevant location (that is, they contain the correct information for the domain and problem file currently being evaluated). If 1, the translator is called before recognition begins. Note that this will cause a `fork()` by Java on the underlying operating system, which means that all of the memory currently allocated to IGRAPH/Java will be *duplicated* just to call the Python translator (which has a trivial memory usage). Therefore, it is better to perform translation prior to calling, lest you want half your available memory to be consumed on a minor script execution. Defaults to 1.

- `-initial` {`Uniform,CV`} — The type of initial probability distribution across the goal-space(s). `Uniform` results in a uniform distribution across the n goals in the goal-space, while `CV` assigns the *causality-value* as described in [4]. Defaults to `CV`.

- `-partial` {`0,1`} — Ignored. Whether to treat observation as potentially partially-observable.

- `-bounded` {`0,1`} — Whether to generate *bounded hypotheses* after each observation. This can slow runtimes considerably on some problems, even though it is multi-threaded. Defaults to 1.

- `-stabilityThreshold [0:1]` — The minimum stability value of a goal for inclusion in the final hypothesis. Defaults to 1.

- `-multithreaded` {`0,1`} — Whether to use multi-threading during observation. Massively speeds up the recognition process when enabled. Defaults to 1.

- `-visual` {`0,1`} — Whether to show a visual representation of the goal-spaces as observation continues. Very useful for debugging. Defaults to 0.

As a concrete example, this is (largely) the command generated by Eclipse when running a test using the test harness, and all dependencies extracted to respective projects in the same root director. Here, `depots` problem 6 is evaluated using a solution file with the causal graph heuristic [2] and *single-action* work function. The goal-space is initialised with the *causality-value* probability distribution.

```
java -classpath ../IGRAPH/bin;
   ../JavaFF2/bin;
   ../JavaFF2/lib/jgrapht-0.8.2/jgrapht-jdk1.6.jar;
   ../JavaSAS/bin;
   ../NewPlanThreader/bin;
   ../IGRAPH/lib/jgrapht-jdk1.6.jar
recogniser.IGRAPHTestHarness
```

```
   ../Domains/ipc3/depots_ipc3/domain.pddl
   ../Domains/ipc3/depots_ipc3/pfile06
   ../Domains/ipc3/depots_ipc3/solutions/pfile6.soln
   /tmp/igraph_out
-heuristic CG -work SA -initial CV
```

5.2 Using IGRAPH in an Application

If you wish to use IGRAPH in your application, you probably will want to avoid the text harness, which assumes you have a file on-disk or a known-goal you want to evaluate against. In truth, you will probably have to refer to `IGRAPHTestHarness.java` for help in figuring out how to integrate the recogniser at all, as this process is not very obvious.

`recogniser.IGRAPH.java` is probably the most obvious object to use, but this is really just a wrapper for `recogniser.BayesianGoalRecogniser.java` which does all of the work. This latter class contains all the true functionality, and would allow you to use an already-grounded problem. While there are a multitude of methods in this class, only a couple are of relevance for simple recognition to occur.

When an action is observed, calling
`BayesianGoalRecogniser.actionObserved(javaff.data.Action)`
will tell IGRAPH to process the action and update the goal-space appropriately. Next, to get a hypothesis either call `BayesianGoalRecogniser.getImmediateHypothesis()` to get an *intermediate* hypothesis[2], or `BayesianGoalRecogniser.getFinalHypothesis()` if you know that observation has finished and want a *final* hypothesis.

5.3 Terminating IGRAPH

As IGRAPH is multi-threaded, it must be told when to close the corresponding thread pool(s). This can be done through `IGRAPH.terminate()` or `BayesianGoalRecogniser.terminate()`. If you are using the test harness, it will do this automatically for you.

6 Useful Tools

IGRAPH comes bundled with several tool/test classes used during development which may be of interest to your research. These are briefly described as follows, and can all be found under the `recogniser.test` package.

- `AbandonmentTestHarness` — A testbed for evaluating IGRAPH when an agent abandons their goal before achievement, and switches to another. See the code for more details on what is required/output by this file.

- `ContinuousPlanner` — A wrapper for JavaFF 2.1 which generates a complete plan for a given goal then cuts the last X% (determined by the user). The state resulting from the execution of this initial part of the plan is

[2]One day this method will be renamed to `getIntermediateHypothesis()`, but today is not that day.

then used as the initial state for another plan whose goal is either a random goal in the goal-space or a user-defined goal. Again, see code for usage. This was created as a means of generating solutions to be fed into `AbandonmentTestHarness`.

- `GraphGenerator` — Generates PDDL domains based on common graph structures (grid, random, wheel etc.). The *city* structure attempts to mimic a state-space with local neighbourhoods. Note that using this class will require some hard-coding of paths/uncommenting of code.

7 Notes

Read the section on 3rd party dependencies!

8 Citing IGRAPH

The most relevant conference publication is from ICAPS 2011, while the thesis is also naturally a useful reference.

-
  ```
  @INPROCEEDINGS{Pattison2011_002,
  author = "David Pattison and Derek Long",
  title = "Accurately determining intermediate and terminal plan
  states using Bayesian goal recognition",
  booktitle = "Proceedings of the First Workshop on Goal, Activity
  and Plan Recognition (GAPRec)",
  year = "2011",
  editor = "David Pattison and Derek Long and Christopher W. Geib",
  pages = "32 -- 37",
  month = "June"
  }
  ```

-
  ```
  @PhdThesis{PattisonPhD,
  author = David Pattison,
  title = A New Heuristic-Based Model of Goal Recognition Without
  Libraries,
  school = University of Strathclyde,
  year = 2015
  }
  ```

9 Useful Links

- PAIR — http://www.planrec.org — Homepage of the PAIR workshop. Contains some useful links to other plan/goal recognisers and datasets.

- FastDownward — http://www.fast-downward.org — The homepage of the Fast Downward planning system, from which the SAS$^+$ translator scripts are taken.

References

[1] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. Teaching forward-chaining planning with JavaFF. In *Colloquium on AI Education, 23rd AAAI Conference on Artificial Intelligence*, 2008.

[2] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 161–170, 2004.

[3] Derek Long and Maria Fox. The third international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[4] David Pattison. *A New Heuristic-Based Model of Goal Recognition Without Libraries*. PhD thesis, University of Strathclyde, 2015.

[5] David Pattison and Derek Long. Domain independent goal recognition. In *STAIRS 2010: Proceedings of the Fifth Starting AI Researchers' Symposium*, volume 222, pages 238 – 250. IOS Press, August 2010.

[6] David Pattison and Derek Long. Extracting plans from plans. In S. Fratini, A. Gerevini, D. Long, and A. Saetti, editors, *Proceedings of the 28th Workshop of the UK Special Interest Group on Planning and Scheduling, PLANSIG'10*, pages 149 – 156, December 2010.

[7] Matthias Westphal and Silvia Richter. The LAMA planner. using landmark counting in heuristic search, 2008. Short paper for IPC 2008.