SATFC MANUAL

[Auction]omics

ALEXANDRE FRÉCHETTE*, NEIL NEWMAN*, KEVIN LEYTON-BROWN*

March 29, 2016

CONTENTS

* *Department of Computer Science, University of British Columbia, British Columbia, Canada.*

1 INTRODUCTION

ABSTRACT SATFC (*SAT-based Feasibility Checker*) solves radio-spectrum repacking feasibility problems arising in the FCC's upcoming reverse auction. It combines a formulation of feasibility checking based on propositional satisfiability, heuristics, a complete SAT solver, and a local search SAT solver both tuned for the types of instances observed in auction simulations.

AUTHORS & COLLABORATORS SATFC is the product of the ideas and hard work of Auctionomics, notably Alexandre Fréchette, Neil Newman, Paul Cernek, Emily Chenn, Guillaume Saulnier-Comte, Nick Arnosti, and Kevin Leyton-Brown.

CONTACT Questions, bug reports and feature suggestions should be directed to Neil Newman – newmanne@cs.ubc.ca.

1.1 Licenses

SATFC is released under the GNU General Public License (GPL) - `http://www.gnu.org/copyleft/gpl.html`.

1.2 System Requirements

SATFC is primarily intended to run on Unix-like platforms. It requires Java 8 to run (all testing has been done with Sun). It also needs our modified version of the SAT solvers CLASP and SATenstein (built on top of UBCSAT) compiled for JNA library usage, which in turn necessitates GCC v4.8.1 or higher as well as the standard Unix C libraries and libtbb. See Section 3 to learn how to compile CLASP and SATENSTEIN . The default portfolio of solvers that SATFC ships with runs 8 solvers in parallel, so you should allocate one SATFC program per 8 cores available. If the default portfolio is overridden (see section 5), then this recommendation would change according to the new portfolio.

1.3 Version

This manual is for SATFC v1.9

2 DESCRIPTION

2.1 Problem

At the core of the reverse auction part of the radio-spectrum incentive auction lies the problem of (re)assigning television stations to a smaller range of broadcast channels than what they currently occupy subject to various interference constraints (often referred to as the Station Packing Problem or the Feasibility Checking Problem). These constraints are pairwise between stations, hence the whole problem can be cast to an (extended) Graph Coloring Problem, where stations are nodes of the graph, edges correspond to interference constraints and colors to available broadcast channels. Unfortunately, the latter problem is known to be NP-complete, *i.e.* computationally challenging. Furthermore, in the latest reverse auction designs, feasibility checking must be executed very frequently, sometimes upwards of a thousand times per auction round. This problem is thus the fundamental computational bottleneck of the auction. Fortunately, the distribution of feasibility checking problems encountered in a typical auction is very specific, and that is what SATFC leverages.

2.2 SATFC

To take advantage of the specific distribution of encountered feasibility checking problems, SATFC first translates feasibility checking into its propositional satisfiability (SAT) version. This allows us to leverage the body of research on high performance SAT solvers developed over the last years. Through extensive empirical evaluations, we have identified the SAT solver CLASP as the best solver when tuned on our specific instance set using SMAC, a sequential, model-based algorithm configurator. In addition to using a highly configured version of clasp, SATFC solves easy instances using a heuristic pre-solving algorithm based on previous partial satisfiable assignments. Recent versions of SATFC also include a configured version of SATENSTEIN , a local search based SAT solver. Finally, some engineering was required to make the whole pipeline as efficient as possible.

2.3 Efficient Usage

SATFC was developed to work particularly well on the auction designs that are being studied by the FCC. For example, in these designs, one expects to encounter many problems to be solved sequentially, a good fraction of which are simple. SATFC thus extensively leverages any partial feasible assignments to get rid of easy instances quickly, and has a main solver tuned for a specific (empirically defined) distribution of harder instances.

3 USAGE

3.1 Setting Up SATFC

Compiled, ready-to-go releases of SATFC are available from the SATFC repository. Packaged with SATFC is a source jar, necessary libraries as well as a copy of CLASP and SATENSTEIN .

3.1.1 *Compiling* CLASP

CLASP will need to be compiled for your machine. To compile the CLASP packaged with a SATFC release, do the following:

```
> cd <SATFC release directory>
> cd clasp/
> bash compile.sh
```

This will put a compiled CLASP library, `libjnaclasp.so`, in the clasp/jna folder. Note that this version of CLASP requires a recent version of libtbb (`https://www.threadingbuildingblocks.org/`) that may not be part of your distribution by default - keep an eye on the output of the compile script and check your package manager for the library. If the compile script fails, it may require some command line arguments depending on where you have libtbb on your machine. These two arguments are absolute paths to your TBB_INCLUDE directory and TBB_LIB directory, respectively. You may also have to update your LD_LIBRARY_PATH environment variable to include the TBB_LIB folder.

3.1.2 *Compiling* SATENSTEIN

SATENSTEIN will need to be compiled for your machine. To compile the SATENSTEIN packaged with a SATFC release, you will need to install CMake and do the following:

```
> cd <SATFC release directory>
> cd satenstein/
> bash compile.sh
```

This will put a compiled SATENSTEIN library, `libjnasatenstein.so`, in the satenstein/jna folder.

3.1.3 *Data*

The broadcast interference constraint data (as specified by the FCC) consists of two files, one specifying station domains, and one specifying pairwise interference between stations. As one of its required arguments, SATFC expects a path to a folder containing both of these files, named `Domain.csv` and `Interference_Paired.csv`, respectively.

DOMAINS The domains file consists of a CSV with no header where each row encodes a station's domain (the list of channels on which a station can broadcast). The first entry of a row is always the `DOMAIN` keyword, the second entry is the station ID, and all subsequent entries are domain channels. Note that SATFC uses this file to define the set of available stations (so it should contain all the station IDs that will be used in defining problems).

INTERFERENCES The interferences file consists of a CSV with no header where each row is a representation of many pairwise interference constraints between a single subject station on a single subject channels with possibly many target stations. Specifically, the entries of a row are, in order, a key indicating the type of constraint, a subject and target channel on which the constraint applies, the subject station of the constraint, and then a list of target stations to which the constraint applies. Here it is in more compact format:

```
<key>,<subject channel>,<target channel>,<subject station>,<target station 1>,...
```

There are five possible constraint keys: `CO`, `ADJ+1`, `ADJ-1`, `ADJ+2`, and `ADJ-2`. The former indicates a *co-channel constraint*, stating that the subject station and any target station cannot be on the same target/-subject channel. The second describes an *adjacent plus one constraint* which implies that the subject station cannot be on its subject channel c together with any target station on the target channel $c + 1$. The third one is an *adjacent minus one constraint*, which is just an adjacent plus one constraint with subject station and channels interchanged with target station and channel, respectively. The fourth describes an *adjacent plus two constraint* which implies that the subject station cannot be on its subject channel c together with any target station on the target channel $c + 2$. The third one is an *adjacent minus two constraint*, which is just an adjacent plus two constraint with subject station and channels interchanged with target station and channel, respectively.

Here are two examples of interference constraints:

```
CO,4,4,1,2,3
```

This constraint means that neither stations 1 and 2 nor stations 1 and 3 can be jointly assigned to channel 4

```
ADJ+1,8,9,1,2,3
```

This constraint implies that neither stations 1 and 2 nor stations 1 and 3 can be on channels 8 and 9 respectively.

```
ADJ-1,8,7,1,2,3
```

This constraint implies that neither stations 1 and 2 nor stations 1 and 3 can be on channels 8 and 7 respectively.

3.2 Understanding Output

When SATFC solves a problem, it returns a run result, runtime, and a witness assignment. The run result can be one of `SAT`isfiable, `UNSAT`isfiable or `TIMEOUT` (or the rarer `CRASHED`) and corresponds to the repackability of the given problem. The runtime is the wall time, in seconds, that SATFC has taken to solve the given problem. Finally, the witness assignment is present only if the given problem is repackable (*i.e.* `SAT`) and consists of a new valid channel assignment for the given stations.

3.3 Using SATFC As a Library

The most efficient way of using SATFC is as a Java library. This source code is packaged with SATFC 's release. The code is well documented; the simplest entry point is the `SATFCFacade` object, and its corresponding builder `SATFCFacadeBuilder`. The `SATFCFacade` is meant to be reused and should not be recreated for each problem (this will incur a performance penalty). Remember to call `close` on the `SATFCFacade` when you are finished using it. As a simple example, the problem of packing stations 1 and 2 into channels 14,15,16,17,18,19,20 and station 3 into channels 14,15,16 would be specified by the following:

```
final SATFCFacadeBuilder satfcFacadeBuilder = new SATFCFacadeBuilder();
// set any options on the builder here, such as specifying the path to the clasp library
try (SATFCFacade facade = satfcFacadeBuilder.build()) { // SATFCFacade implements AutoCloseable
    final Map<Integer, Set<Integer>> domains = new HashMap<>();
    domains.put(1, new HashSet<>(Arrays.asList(14,15,16,17,18,19,20)));
    domains.put(2, new HashSet<>(Arrays.asList(14,15,16,17,18,19,20)));
    domains.put(3, new HashSet<>(Arrays.asList(14,15,16)));
    final Map<Integer, Integer> previousAssignment = new HashMap<>();
    // Fill in previous assignment here, or leave blank for no previous assignment
    double cutoff = 60; // solve for 60 seconds
    long seed = 1; // random seed
    String stationConfigurationFolder = "interference-data/fcc-interference-data/";
    SATFCResult result = facade.solve(
            domains,
            previousAssignment,
            cutoff,
            seed,
            stationConfigurationFolder
    );
    System.out.println(result.getResult());
    System.out.println(result.getWitnessAssignment());
}
```

The reader may find it helpful to consult Section 3.1.3 to understand the components at play.

3.4 Using SATFC from the Command Line

> **Warning**
> Every time SATFC is launched from the command line, it will have to load the Java Virtual Machine, necessary libraries as well as any constraint data corresponding to the specified problem. This is a significant overhead compared to the time required to solve easy instances. If it is necessary to solve large numbers of instances, many of which are easy, we suggest using SATFC as a Java library.

To use SATFC from the command line to solve feasibility checking problems, go in the SATFC directory and execute the following:

```
> ./bin/SATFC -DATA-FOLDERNAME <data folder> -DOMAINS <station domains>
```

where

- `data folder` points to a folder containing the *broadcasting interference constraints data* (`Domain.csv` and `Interference_Paired.csv` files) - see Section 3.1.3 for further information, and

- `station domains` is a string listing a set of stations to be packed, and for each station a set of eligible channels. This *domains string* consists of *single station domains strings* joined by ';', where each single domain string consists of a station numerical ID, a ':', and a list of integer channels joined by ','.

  For example, the problem of packing stations 1 and 2 into channels 14,15,16,17,18,19,20 and station 3 into channels 14,15,16 would be specified by the following string:

  `1:14,15,16,17,18,19,20;2:14,15,16,17,18,19,20;3:14,15,16`

It is recommended to surround the argument values given to SATFC by double-quotes.

One can also run

```
> ./bin/SATFC --help
```

to get a list of the SATFC options and parameters.

SATFC 's output mainly comprises informative logging and three result lines. Here is an example where we ask SATFC to pack a single station 13933 into channels 14 to 29:

```
> bash bin/SATFC -DATA-FOLDERNAME "interference-data/fcc-interference-data/" -DOMAINS "
    13933:14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29"
07:27:47.765 [main] INFO  SATFCFacadeExecutor - Initializing facade.
Found default library SATFC/clasp/jna/libjnaclasp.so.
07:27:47.809 [main] INFO  SATFCFacadeExecutor - Solving ...
07:27:47.809 [main] WARN  SolverManager - Requested data from interference-data/fcc-
    interference-data/ not available, will try to add it.
07:27:52.580 [main] INFO  SATFCFacadeExecutor - ..done!
SAT
0.017
{13933=16}
07:27:52.580 [main] INFO  SATFCFacade - Shutting down...
07:27:52.580 [main] INFO  SATFCFacade - Goodbye!
```

Most of the informative logging starts with a times stamp, the thread generating the logging, the log level and the class generating the logging. It can be useful to understand a buggy scenario or to monitor a SATFC run. The three main lines of interest are the `SAT`, `0.017` and `{13933=16}` lines, which correspond to the run result, run time and witness assignment, respectively.

3.4.1 *Parameters & Options*

In addition to the required arguments discussed above, command-line SATFC also exposes the following optional parameters:

- `--help` – display SATFC options and parameters, with short descriptions and helpful information.
- `-PREVIOUS-ASSIGNMENT` – a partial, previously satisfiable channel assignment. This is passed in a string similar to the `-DOMAINS` string: the station and previous channel pairs are separated by ':', and the different pairs are joined by ','. For example, "`1:14,15,16;2:14,15`" means pack station 1 into channels 14,15 and 16 and station 2 into channels 14 and 15.
- `-CUTOFF` – a cutoff time, in seconds, for the SATFC execution.

> **Warning**
> SATFC was optimized for runtimes of about one minute. Thus, components might not interact in the most efficient way if cutoff times vastly different than a minute are enforced, especially much shorter ones. Moreover, cutoff times below a second may not always be respected.

- `-SEED` – the seed to use for any (non-CLASP) randomization done in SATFC .
- `-CLASP-LIBRARY` – a path to the compiled "`.so`" CLASP library to use.
- `-SATENSTEIN-LIBRARY` – a path to the compiled "`.so`" SATENSTEIN library to use.
- `--log-level` – SATFC 's logging level. Can be one of `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE` (listed in increasing order of verbosity).

4 SATFC-SERVER

> **Warning**
> SATFC-SERVER is an efficient in-memory representation of a constantly expanding cache. As such, it has *very large* memory requirements. While the particular memory requirement will depend on exact usage, as a ballpark estimate we typically run the SATFC-SERVER on a machine with 32 GB of RAM available.

There is an entirely optional addition to SATFC called SATFC-SERVER . SATFC-SERVER provides a way to store results from problems that SATFC solves and use them to solve future problems. It is intended for long term, heavy users of SATFC . SATFC-SERVER is web app which uses the Spring Boot framework. In order to run this component, you need to have a Redis server running. We recommend using a dedicated Redis server for this purpose.

Ready-to-go releases of SATFC-SERVER are available from the SATFC repository. It is packaged similarly to SATFC . You can start the server by running

```
> ./SATFCServer --redis.host=<redishost> --redis.port=<redisport> --constraint.folder=<
    constraintfolderpath>
```

The server needs to parse the redis database before it can be used, and this may take a while. It is ready for use when you can hit `<host>:<port>/satfcserver/health` or when you observe a log line similar to

```
INFO [ca.ubc.cs.beta.stationpacking.webapp.Application] Started Application in 3.878
    seconds (JVM running for 4.309)
```

4.1 Parameters & Options

The command line parameters are summarized below:

- `--redis.host` The port that the server runs on. Defaults to localhost
- `--redis.port` The port that the server runs on. Defaults to 6379
- `--constraint.folder` A path to a folder which contains station configuration folders. See Section 4.1.1 for more info.
- `--server.port` The port that the server runs on. Defaults to 8080
- `--logging.level.ca.ubc.cs.beta.stationpacking` The logging level for SATFC classes. Defaults to INFO

Some additional parameters related to server configuration and logging are exposed through Spring Boot. A larger list of parameters is available in the Spring Boot docs. Note that many of them are not applicable to SATFC-SERVER .

4.1.1 *Constraint Folder*

You can use a single SATFC-SERVER instance even if your problems come from different data folders (see Section 3.1.3 for more information on data folders). To make this work, SATFC-SERVER expects your data folders to be organized in a particular way. SATFC-SERVER takes as input a path to a folder that contains all of your data folders (and nothing else!). For example, if you have two data folders, named *Data_Folder_1* and *Data_Folder_2*, then your directory structure might look like:

```
constraint_sets
├── Data_Folder_1
│   ├── Domain.csv
│   └── Interference_Paired.csv
└── Data_Folder_2
    ├── Domain.csv
    └── Interference_Paired.csv
```

and you would pass in the path to the *constraint_sets* folder as your argument to `---constraint.folder`.

4.2 Modifications to running SATFC

Once the server is up and running, you can run SATFC as before with the following changes. If you are running from the command line, you need to add the following parameters: `--serverURL <serverhost>:<serverport>/satfcserver`. If you run SATFC using the facade, then you need to add the following lines to when you instantiate the facade:

```
SATFCFacadeBuilder satfcFacadeBuilder = new SATFCFacadeBuilder();
/* initialize builder normally here
* ...
*/
// add caching info
satfcFacadeBuilder.setServerURL(<serverURL>)
// finish like normal
SATFCFacade satfc = satfcFacadeBuilder.build();
```

4.3 Dealing with memory errors

You may need to configure the JVM memory settings to increase the maximum heap size available to the SATFC-SERVER . This can be done with by opening up the SATFC-SERVER launch script and modifying the line starting with DEFAULT_JVM_OPTS, for example:

```
DEFAULT_JVM_OPTS='"-Xmx16g"'
```

would set the allowable memory consumption to 16 gigs. If adding more memory is not possible, then you can delete cache entries from the underlying redis and restart the server. By sending an HTTP *POST* request to the `<host>:<port>/satfcserver/filterSAT` endpoint, the SATFC-SERVER will prune entries that contain redundant information. These redundant cache entries will be deleted from the backing redis as well. Note that this filtering operation can take many hours and will take time proportional to the size of the cache. While the filtering operation is ongoing, the SATFC-SERVER should not be answering other requests.

4.4 A note on allocating resources

The SATFC-SERVER can handle multiple requests concurrently. However, the number of concurrent requests is limited by the underlying hardware. For example, if you run the SATFC-SERVER on a machine with 16 physical cores, then you should not expect to handle more than 16 concurrent requests without

experiencing slowdown. The SATFC-Server has no built in mechanisms for scaling horizontally, and it is expected that users are running a single SATFC-Server .

4.4.1 *Metrics*

A small number of metrics are available via `<host>:<port>/satfcserver/metrics` and `<host>:<port>/satfcserver/met`

5 CUSTOMIZING THE PORTFOLIO

> **Warning**
> This is an advanced and largely undocumented feature. Unless you have a compelling reason to alter the portfolio and are willing to read and understand the SATFC source code, it's probably better to stay away and use the defaults.

SATFC 's portfolios are specified in *YAML* files. There are default, provided files that ship with SATFC and will run without user intervention. However, you can override the *.yaml* file being used by calling the

```
satfcFacadeBuilder.setConfigFile("/pathtosomefile.yaml")
```

method on the `SATFCFacadeBuilder`. The main reasons to want to alter a portfolio are to change the parameter configurations that are given to each solver, or to take greater advantage of multiple cores. The format for these *.yaml* files can be inferred from a close read of the `YAMLBundle` class and looking at the files that ship with SATFC ; further documentation is unavailable at this time.

6 WORKING FROM SOURCE: BUILDING SATFC

This section is for anyone that wishes to develop SATFC . It can be skipped otherwise. SATFC uses the Gradle build system for dependency resolution. To compile from source and build your own command-line executable version of SATFC , check out the source code from and execute the following:

```
> cd <SATFC repository>
> ./gradlew installDist
```

This builds and packages SATFC in `<SATFC repository>satfc/build/install/`, and may be a lengthy process as it (possibly) installs Gradle, downloads all of SATFC 's dependencies (from external repositories) and builds the project. Note that it can sometimes fail while trying to download dependencies - restarting the process fixes this.

SATFC uses Lombok annotations to streamline its source code. We suggest you install the Lombok plugin for your favorite IDE. Otherwise, it possible to "delombok" the source code, but that will make your code completely incompatible with SATFC 's pipeline.

7 ACKNOWLEDGMENTS

Steve Ramage deserves a special mention as he is indirectly responsible for much of SATFC 's quality, via his package AEATK, one of the main libraries in SATFC that was used throughout its development for various prototypes, as well as different other miscellaneous features. He also was a constant source of technical help and knowledge when it came to the software design of SATFC .

We would also like to acknowledge various members of the Auctionomics team. Ilya Segal provided the main idea behind the pre-solver used in SATFC . Ulrich Gall and his team members offered much useful feedback after working with SATFC during its early stages.