CATS 2.0 User Guide

Galen Andrew

1 Compiling CATS

You will probably have to edit the Makefile to get CATS to compile on your system. If there are compilation problems, check the library and include directories in the Makefile. The current Makefile is configured to compile in Linux. To compile on UNIX, remove the `-DLINUX` flags. To compile on Windows systems, remove the `-DLINUX` flags and add `-DWIN32` and `-D_WIN32`. These instructions are also in the Makefile.

To solve linear programming problems, CATS uses a free library called lp_solve 4.0 by default, which is included with the distribution. If you have access to CPLEX 8.0, we strongly recommend using it in place of lp_solve 4.0. This requires only commenting out one section of the Makefile and uncommenting another. Precise instructions are included in the Makefile.

2 CATS command line

2.1 Required Parameters

CATS requires three pieces of information in the command line: the type of distribution, the number of bids, and the number of goods.

A single built-in distribution from which all instances are to be generated is selected with the `-d` flag and can be any of the following: `arbitrary, arbitrary-npv, arbitrary-upv, matching, paths, regions, regions-npv, regions-upv, scheduling, L1, L2, L3, L4, L5, L6, L7,` and `L8`. The distributions are defined in [1]. The optional suffixes `-npv` and `-upv` available for the regions and arbitrary distributions specify that valuations are to be drawn from either a normal or uniform distribution, respectively. If a suffix is omitted, the valuation distribution is selected at random for each instance.

The `-uniform_hybrid` flag can be used in place of the `-d` flag to produce instances by first choosing a built-in distribution uniformly at random for each instance,[1] then generating from it. The `-dist_dist` flag, described in section 3.1, can also be used in place of `-d`.

The number of goods and bids must also be specified for each run. If every instance is to have the same number of goods and bids, the `-goods` and `-bids` flags are used to provide these numbers. It is also possible to choose the numbers of goods and bids separately for each instance from a uniform distribution over a specified range. This is done with the `-random_goods` and `-random_bids` flags. Both are followed by two integers specifying the minimum and maximum values of the range.

If the `-default_hard` flag is used (see section 3.5), all of the above parameters take on default values and are not required. A specific distribution can still be chosen, but the number of bids and goods, if they are (redundantly) entered, must be 1000 and 256, respectively, since our hardness models are based on this problem size. The 2.1 release of CATS may allow variable problem sizes.

[1]L1 and L5 are excluded because it is impossible to generate large numbers of non-dominated bids with them. Regions and arbitrary are given equal weight to the other distributions before a valuation type is chosen.

2.2 Optional Parameters

CATS can be given several optional parameters that are listed and described here, with their default values:

- `-n` [1]: number of instance files that you want CATS to generate
- `-seed` [taken from clock]: random seed
- `-seed2` [taken from clock]: random seed 2: used only for normal distributions
- `-bid_alpha` [1000]: multiply prices by this before rounding, when `-int_prices` is set
- `-num_weighted_samples` [50]: number of samples to take in generating from distribution over features (see section 3)

2.3 Optional Flags

The following optional flags also alter CATS' behavior. The ones with arguments have no default values.

- `-cplex`: write a CPLEX-compatible .lp file as well as a CATS data file
- `-no_dom_check`: turn off removal of dominated bids. The default CATS behavior is to keep generating bids until the specified number of *non-dominated* bids have been generated.
- `-int_prices`: write integer prices in bids instead of real-valued prices
- `-filename`: set filename prefix for generated instances
- `-no_output`: do not write unnecessary output to the screen
- `-random_parameters`: choose distribution parameters uniformly from their ranges independently for each instance
- `-model_filenames`: file containing filenames of polynomially defined distributions over distribution parameters and/or polynomial functions of instance features for weighted sampling (see section 3)
- `-model_file_help`: print help screen for model file features
- `-no_normalization`: turns off normalization of all polynomially defined distributions. If the polynomials do not represent proper probability density functions over their ranges, behavior is undefined.
- `-default_hard`: weight problems for hardness using the built-in hardness model (see section 3)
- `-help, -h, -?`: print a usage screen. If a distribution is selected with `-d`, help specific to that distribution is also printed.

3 Advanced Features

CATS 2.0 has several advanced features over CATS 1.0 as described in [1], related to the work of Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden and Yoav Shoham during 2002-2003 [2] [3]. They enable the creation of new distributions by skewing the built-in distributions to emphasize desired regions of the problem space. The features are first described, and an explanation of how to use them in a run of CATS follows. In the discussion, we will refer to the built-in distributions and the uniform hybrid distribution using either constant or uniformly randomized parameter settings as *simple* distributions.

3.1 Arbitrary hybrid distributions

It is possible to generate instances from a user-specified hybrid distribution that, for each instance, first selects one of the built-in distributions with probabilities specified by the user, and then generates a set of bids from that distribution. This is done with the `-dist_dist` flag. The argument to this flag is the name of a file containing a list of numbers specifying the probability of choosing each distribution. The file must contain one number for each of the built-in distributions (excluding regions and arbitrary, exactly 15 in all) in the order listed in section 2.1, separated by whitespace. The regions and arbitrary distributions are omitted because they are redundant: any probability assigned to them can be divided directly into the -npv and -upv probabilities. If the entries do not sum to 1, they will be normalized.

3.2 Specifying Parameter Distributions

Every CATS distribution has from 1 to 7 parameters. These parameters can be used with their default values, can be individually set at the beginning of a run, or can be chosen for each instance uniformly from their ranges with the `-random_parameters` flag. They can also be chosen according to a polynomially defined joint distribution. CATS will interpret the polynomial (which must have the same number of variables as the distribution has parameters) as a pdf and choose the parameter settings for each instance by sampling from it. Unless the `-no_normalization` flag is used, CATS will first normalize the distribution to ensure that it is non-negative on the ranges of the parameters and integrates to one. In fact, CATS will not only raise the polynomial to be non-negative everywhere, it will also *lower* the polynomial, if necessary, to ensure that the minimum is zero. At present, the only way to specify a distribution with minimum value greater than zero is to use `-no_normalization`.

3.3 Specifying Feature Weighting

CATS is equipped with procedures for calculating 32 features of instances that are predictive of instance hardness. It is possible to provide a polynomial function of these features that is used to do weighted sampling from instances. CATS then generates from the *product* of a simple distribution and the feature polynomial, reinterpreted as a pdf. This essentially skews the simple distribution to emphasize instances that are weighted more heavily by the feature polynomial.

If this feature is enabled, for each instance generated, first a certain number of samples are created (the default is 50) according to a simple distribution. Their feature values are calculated, and then one is sampled for output after weighting the samples according to the feature polynomial evaluated at the calculated feature vectors. The number of samples can be changed with the `-num_weighted_samples` flag. There is no way to determine exactly how many samples are necessary for accurate sampling, but it should be likely that a few highly weighted instances are generated within the given number of samples. Several feature polynomials may be entered (up to a maximum of 10), in which case it is their *minimum* that is used as the weight for sampling. The original purpose of this feature was to weight instances according to their estimated hardness when solved using an *algorithm portfolio* [2], which is the minimum estimated run-time of the portfolio's constituent algorithms.

3.4 Interaction of Parameter Distributions and Feature Weighting

Both parameter distributions and feature weighting can be used during the same run of CATS. This will not, as might be expected, generate from the distribution using the parameter model skewed according to the feature model.[2] Problems will still be generated from the product of the distribution with *uniformly* chosen parameter values and the feature polynomial. The purpose of providing parameter distributions in addition to feature weighting, then, is to guide generation toward instances that are more likely to be weighted highly by the feature model. This reduces the number of samples necessary to sample

[2]Generation from this type of custom distribution is not currently supported.

accurately from the weighted distribution. The number of samples taken should be adjusted using the `-num_weighted_samples` flag depending on your degree of confidence in the parameter model.

After generating samples using the distribution over parameters, the weight of each sample is the value of the feature polynomial divided by the value of the parameter pdf at the chosen parameter vector. Similarly, if a hybrid distribution is specified, sample weights will (also) be divided by the probability of that sample's distribution having been chosen, so that the weighting is with respect to the uniform hybrid distribution. Note that even if the parameter model (or hybrid distribution) is inaccurate and guides generation away from the desired distribution based on features, this procedure will still ultimately generate from the correct distribution. Only instead of speeding up sampling, *more* samples would be necessary than if no parameter model were used at all. For a more formal description of the sampling method used, see [3].

3.5 Default Hard Distributions

One of the original motivations for implementing these advanced features was for use with models of algorithm runtime. Thus, provided with CATS are default feature and parameter models of run-time for the three algorithms studied in [2]. The `-default_hard` flag causes CATS to use these models, weighting the problems generated according to the minimum predicted runtime of three different algorithms, in order to produce harder problems. The parameter models are also used to speed the sampling process. Problems can be generated from a single CATS distribution chosen with the `-d` flag, or if no specific distribution is entered, the uniform hybrid distribution over all CATS distributions except for L1, L5, L8 and paths is used.[3] Note that in this case the easiest distributions will be sampled with very low probability. It is not possible to use the `-default_hard` flag with any other hybrid distribution. Also note that because the default feature and parameter models were trained on instances with 256 goods and 1000 bids, this is the only problem size that can be generated with the default models.

3.6 Input of Polynomial Models

Polynomial models (both parameter models and feature models) are input as files that contain the coefficients of the polynomials. The format of the files is a list of numbers separated by whitespace in the following order: the degree of the polynomial, the number of variables, and then the coefficients, which are ordered first according to the degree of the term and second according to the variables in the term, when listed from highest to lowest, with repetition. For example, the second order terms of a polynomial in x, y, and z would have their coefficients listed in the following order: $x^2, yx, y^2, zx, zy, z^2$. Here is an example of an entire third degree polynomial in four variables, where the term is written in place of its coefficient:

3 4
`[constant term]`
x_1 x_2 x_3 x_4
x_1^2 x_2x_1 x_2^2 x_3x_1 x_3x_2 x_3^2 x_4x_1 x_4x_2 x_4x_3 x_4^2
x_1^3 $x_2x_1^2$ $x_2^2x_1$ x_2^3 $x_3x_1^2$ $x_3x_2x_1$ $x_3x_2^2$ $x_3^2x_1$ $x_3^2x_2$ x_3^3 $x_4x_1^2$ $x_4x_2x_1$ $x_4x_2^2$ $x_4x_3x_1$ $x_4x_3x_2$ $x_4x_3^2$ $x_4^2x_1$ $x_4^2x_2$ $x_4^2x_3$ x_4^3

Since there may be many models over parameters (one for each distribution) and features, it would be cumbersome to enter the name of each file on the command line. Instead, a single file that contains the names of all of the model files is passed as an argument with the `-model_filenames` flag. The format of the model filenames file is a list of filenames preceeded by markers specifying the type of model, either `-param_dist` for distributions over distribution parameters, or `-feat_poly` for feature weight polynomials. The `-param_dist` marker must also be followed by the name of the distribution to which it applies, before the filename. Any number of parameter models may be included, even if the indicated distribution

[3]L1, and L5 are omitted because it is impossible to generate large numbers of non-dominated bids with them, and paths and L8 because they hadn't been implemented when we collected the algorithm run-time data.

is not used in the run. For example, here is the included default hard distribution model filenames file. It specifies three feature models, and all parameter models, even if only a single distribution is used in a particular run.

```
-param_dist arbitrary-npv qpstuff/arbitrary-npv-qp.txt
-param_dist arbitrary-upv qpstuff/arbitrary-upv-qp.txt
-param_dist L2 qpstuff/L2-qp.txt
-param_dist L3 qpstuff/L3-qp.txt
-param_dist L4 qpstuff/L4-qp.txt
-param_dist L6 qpstuff/L6-qp.txt
-param_dist L7 qpstuff/L7-qp.txt
-param_dist matching qpstuff/matching-qp.txt
-param_dist regions-npv qpstuff/regions-npv-qp.txt
-param_dist regions-upv qpstuff/regions-upv-qp.txt
-param_dist scheduling qpstuff/scheduling-qp.txt
-feat_poly qpstuff/cass-qp.txt
-feat_poly qpstuff/cplex-qp.txt
-feat_poly qpstuff/lehman-qp.txt
```

4 Additional Resources

For more information about how to use CATS and up-to-date releases, please go to the CATS website at `http://robotics.stanford.edu/CATS/`. Direct any questions, suggestions or problems to Galen Andrew at `galand@cs.stanford.edu`. We hope you find CATS a valuable resource. Good luck with your research!

References

[1] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM EC*, 2000. `http://robotics.stanford.edu/~kevinlb/CATS.pdf`

[2] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *IJCAI*, 2003. `http://robotics.stanford.edu/~kevinlb/portfolio-IJCAI.pdf`

[3] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *CP*, 2003. `http://robotics.stanford.edu/~kevinlb/boosting-CP.pdf`