EDACC - An advanced Platform for the Experiment Design, Administration and Analysis of Empirical Algorithms

Adrian Balint, Daniel Diepold, Daniel Gall, Simon Gerber, Gregor Kapler, and Robert Retz

Ulm University
Institute of Theoretical Computer Science
89069 Ulm, Germany
{adrian.balint, daniel.diepold, daniel.gall, simon.gerber, gregor.kapler, robert.retz}@uni-ulm.de

Abstract. The design, execution and analysis of experiments using heuristic algorithms can be a very time consuming task in the development of an algorithm. There are a lot of problems that have to be solved throughout this process. To speed up this process we have designed and implemented a framework called EDACC, which supports all the tasks that arise throughout the experimentation with algorithms. A graphical user interface together with a database facilitates archiving and management of solvers and problem instances. It also enables the creation of complex experiments and the generation of the computation jobs needed to perform the experiment. The task of running the jobs on an arbitrary computer system (or computer cluster or grid) is taken by a compute client, which is designed to increase computation throughput to a maximum. Real-time monitoring of running jobs can be done with the GUI or with a web frontend, both of which provide a wide variety of descriptive statistics and statistic testing to analyze the results. The web frontend also provides all the tools needed for the organization and execution of solver competitions.

1 Introduction

Many problems that come from practical applications or from theory are known to be very hard to solve. This means that the time for solving these problems increases exponentially with the size of the input. The class of NP-complete problems is probably the most well known class of such problems. Formerly, proving that a problem was NP-complete meant that the design of a practical algorithm for this problem would be useless because of the estimated exponential time of the algorithm. The situation changed drastically with the development of heuristics, meta-heuristics and approximation algorithms for hard combinatorial problems. The size of the problems that can be solved by these kind of algorithms has increased continuously over the years.

www.manaraa.com

This progress can be seen as the result of a paradigm change from "algorithms are fast if they have a theoretical good upper bound for their runtime" to "algorithms are fast if they are fast in practical experiments". This should not mean that theoretical results are not important any more, but rather that the design of algorithms has become oriented towards practical applications.

With this paradigm change methodologies have also changed a lot. A theoretical analysis of heuristics is not possible in most cases, and has been replaced by an empirical evaluation like the ones used in engineering. Most development of empirical algorithms now follows an engineering scheme like the one in figure 1.

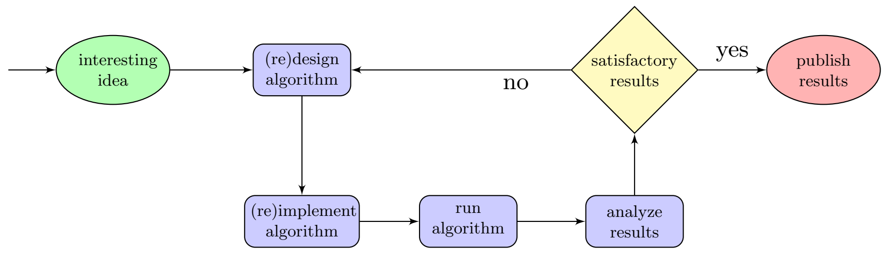


Fig. 1. A typical work flow for the development of empirical algorithms

With the use of new methodologies new problems arise. After the design and implementation phase the algorithm has to be tested and evaluated, which in most cases is a very time consuming task. The first problem that an algorithm designer encounters is the collection and selection of instances on which the solver will be evaluated. A lack of publicly available repositories can hinder this task. Dependent on the set of instances chosen for the evaluation a parameter configuration for the algorithm has to be chosen. This problem can be very often solved by automated procedures like ParamILS [5]. Having the instances and the parameters for the evaluation the user has to choose a computing system. A multi-core computer or a cluster or even a grid can speed up the computations drastically, but at the same time the problem of equally distributing the workload arises, which in most cases is solved by some home brewed scripts. After finishing the computation the results have to be gathered from the computing systems and the important information has to be extracted from the output by some parsing procedures. To find out to what extent the results are satisfactory some statistical tests have to be applied. Comparing the performance of the own algorithm with others demands further elaborated statistics.

The processes of evaluation and analysis are seldom reproducible between different researchers, because of the complexity of the process and the lack of common methods. This is probably the reason why most of the communities working on empirical algorithms periodically organize competitions. The purpose of these competitions is to provide the same evaluation and analysis environment

www.manaraa.com

for all the algorithms. A problem with these competitions is that the underlying evaluation system consists of scripts and databases that are not freely available.

The system EDACC (**E**xperiment **D**esign and **A**dministration for **C**omputer **C**lusters) overcomes most of these problems. The previous version of EDACC [1] was restricted to SAT-solvers and SAT-instances). EDACC is capable of managing solvers with their parameters, instances, creating experiment jobs, running them on arbitrary computing systems ranging from multi-core computers to large scale grids, collecting the results and processing them. Advanced methods for automatically extracting and archiving information from the results and from the instances are provided for users. EDACC also provides a large variety of statistical tests and descriptive statistics to analyze the results. To make the organization and execution of competitions with EDACC possible, also a competition mode that follows a widely accepted scheme is provided.

The paper is organized as follows. Chapter 2 gives an overview over the system. Chapter 3 describes the methods for extracting information from instances and from the results. The wide range of possibilities for statistically analyzing this information is presented in chapter 4. Chapter 5 describes the competition mode of the system. Some implementation details and related work is given in chapters 6 and 7. Chapter 8 concludes with some outlooks.

2 EDACC - Overview of the main components

A detailed description of the core functionalities of EDACC restricted to the SAT problem was given in [1]. We have considerably extended EDACC to be able to handle arbitrary solvers and instances. All further improvements such as information extraction, statistical analysis and the competition mode, are new features described in this work. To make this paper self-contained an overview of the components of EDACC is given.

Before describing the main components some entities that will be used through the rest of the paper are defined. A *solver* is an implementation of an algorithm that works on some input and has an output. The behavior of a solver is controlled by arbitrarily many *parameters*. A solver together with some fixed parameters is called a *solver configuration*. The input to a solver is called an *instance*. Any information that can be computed from an instance is called an *instance property*. A *computing system* is defined as the computer, computer cluster, or grid on which a solver is tested. When running a solver on a computing system *computational limits* can be imposed (e.g. maximum computation time or maximum memory). An *experiment* is the cartesian product of some set of algorithm configurations, a set of instances, a set of computing systems, and some computational limits. An element of an experiment is a *job*. When the computation of a job is finished it will have a *result*. Any information that is computed from a result is a *result property*.

The main components of EDACC are:

1. database (DB)
2. graphical user interface client (GUI)

www.manaraa.com

3. compute client (CC)
4. web frontend (WF)

The DB is responsible for storing and archiving all the information about the entities defined above. Examples for such information are for solvers the name, version, author, binary, MD5 checksum and the source code. For instances we store the filename, the instance, and the MD5 checksum. The DB also acts as the mediator between GUI, CC, and WF.

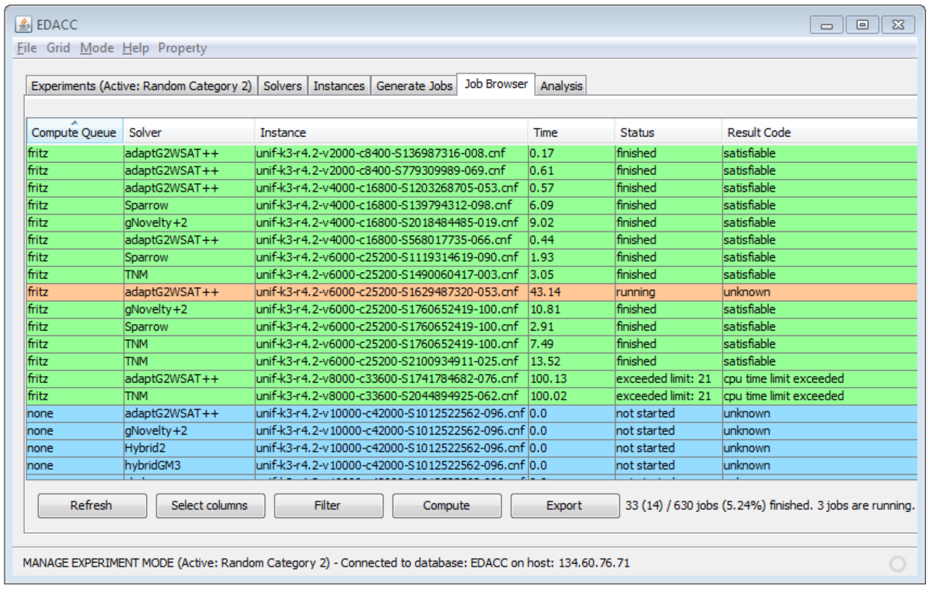
The GUI is split into two modes: *manage DB mode* and *experiment mode*. The first mode provides all the necessary DB-operations e.g. create, remove, update and delete (CRUD) for solvers, parameters and instances. As the number of instances stored in the DB can be very large a categorization of the instances into a hierarchical class model is provided. There are two types of classes: *source classes* and *user classes*. The first one specifies the source of the instances. The second one enables the user to create its own collection of instances from different source classes. The class generation process can be done manually or automatically by using the names and the hierarchies of the directories from where the instances are imported.

The work flow of EDACC usually starts by adding solvers, specifying their parameters, and by adding instances and categorizing them into classes. When all the solvers and instances are available in the DB, the user can switch to the experiment mode. After providing some general information, e.g. a name and description of the experiment, the user can select and configure the solvers to be used in the experiment. There are a lot of solver configuration possibilities e.g. enabling or disabling parameters, automated generation of seeds for probabilistic solvers, linking seeds between solvers, for minimizing the variance, and many more.

Next, the instances to use for the created experiment have to be chosen. This operation is alleviated by the instance classes and by filters, enabling a fast selection process. To restrict the consumption of resources like cpu time or memory different limitations can be imposed on the solvers. If the tested solvers are probabilistic there is the possibility to configure the number of repetitions. After choosing a computation system (for which some basic information has to be provided), the user can generate the jobs for the experiment and the distribution package, which is an archive containing the compute client and a configuration file. The configuration file contains information about the DB connection, the experiment and the target compute system.

Copying the distribution package to the computing system and starting the CC will start the processing of the jobs. The CC consists of three programs: *launcher*, *watcher* and *verifier*. The launcher fetches jobs from the DB and passes them to the watcher, which monitors the use of resources and imposes the desired limitations (At the moment we use the runsolver program from the SAT Competition to achieve this [7]). When a solver finishes, the verifier is used to check the result of the solver, and upon completion the launcher writes all results back to the DB. The verifier is characteristic for each kind of instance and can be replaced or not invoked at all.

المنارة للاستشارات
www.manaraa.com



Compute Queue	Solver	Instance	Time	Status	Result Code
fritz	adaptG2WSAT++	unif-k3-r4.2-v2000-c8400-S136987316-008.cnf	0.17	finished	satisfiable
fritz	adaptG2WSAT++	unif-k3-r4.2-v2000-c8400-S779309989-069.cnf	0.61	finished	satisfiable
fritz	adaptG2WSAT++	unif-k3-r4.2-v4000-c16800-S1203268705-053.cnf	0.57	finished	satisfiable
fritz	Sparrow	unif-k3-r4.2-v4000-c16800-S139794312-098.cnf	6.09	finished	satisfiable
fritz	gNovelty+2	unif-k3-r4.2-v4000-c16800-S2018484485-019.cnf	9.02	finished	satisfiable
fritz	adaptG2WSAT++	unif-k3-r4.2-v4000-c16800-S568017735-066.cnf	0.44	finished	satisfiable
fritz	Sparrow	unif-k3-r4.2-v6000-c25200-S1119314619-090.cnf	1.93	finished	satisfiable
fritz	TNM	unif-k3-r4.2-v6000-c25200-S1490060417-003.cnf	3.05	finished	satisfiable
fritz	adaptG2WSAT++	unif-k3-r4.2-v6000-c25200-S1629487320-053.cnf	43.14	running	unknown
fritz	gNovelty+2	unif-k3-r4.2-v6000-c25200-S1760652419-100.cnf	10.81	finished	satisfiable
fritz	Sparrow	unif-k3-r4.2-v6000-c25200-S1760652419-100.cnf	2.91	finished	satisfiable
fritz	TNM	unif-k3-r4.2-v6000-c25200-S1760652419-100.cnf	7.49	finished	satisfiable
fritz	TNM	unif-k3-r4.2-v6000-c25200-S2100934911-025.cnf	13.52	finished	satisfiable
fritz	adaptG2WSAT++	unif-k3-r4.2-v8000-c33600-S1741784682-076.cnf	100.13	exceeded limit: 21	cpu time limit exceeded
fritz	TNM	unif-k3-r4.2-v8000-c33600-S2044894925-062.cnf	100.02	exceeded limit: 21	cpu time limit exceeded
none	adaptG2WSAT++	unif-k3-r4.2-v10000-c42000-S1012522562-096.cnf	0.0	not started	unknown
none	gNovelty+2	unif-k3-r4.2-v10000-c42000-S1012522562-096.cnf	0.0	not started	unknown
none	Hybrid2	unif-k3-r4.2-v10000-c42000-S1012522562-096.cnf	0.0	not started	unknown
none	hybridGM3	unif-k3-r4.2-v10000-c42000-S1012522562-096.cnf	0.0	not started	unknown

33 (14) / 630 jobs (5.24%) finished. 3 jobs are running.

MANAGE EXPERIMENT MODE (Active: Random Category 2) - Connected to database: EDACC on host: 134.60.76.71

Fig. 2. A snapshot of the job browser within the experiment mode of the GUI, while monitoring the progress of an experiment.

There are no limitations on how many CC's are running at the same time. If the computing system is a computer cluster or a grid, then the CC can be run on all nodes to increase throughput. If the nodes have multi-core CPU's the client can make use of this by starting multiple jobs on a node. Crashes of parts of the computing system will not affect the processing of the experiment, because failed jobs are computed by other CC's. A nice feature worth mentioning is that instances or solvers can be added and deleted during computation, without having to stop the CC's. When a CC finishes a job it writes the results (e.g. CPU time, output of solver, watcher and verifier) back to the DB and picks another job until all jobs are completed. During the computation of the jobs the job browser from the GUI or the WF enables real-time monitoring of the jobs (see Fig. 2). When all jobs of an experiment are finished, the user can extract information from the results and from the instances, and use it for descriptive statistics or statistical tests, that can be performed within the GUI or WF. These features are described in detail in the next chapters.

3 Information extraction

To analyze the results of an experiment different kinds of statistics can be used. The more information about the experiment's results and instances are available, the more powerful these statistics can be. To make the analysis more easy for the user, EDACC supports a variety of information extraction mechanisms. All

www.manaraa.com

the information extracted through these mechanisms can be saved in the DB and used for statistics or can be exported.

We differentiate between two kinds of information, depending on the source. Any information that can be computed from the input instances is called an *instance property (IP)*. All other information is called a *result property (RP)*. The sources of RP's are: the parameters of the solver, and the outputs (stdout, stderr) of the solver, launcher, watcher and verifier.

Most of the information researchers are interested in is present in some output file, and can be easily extracted by a parser procedure. However there are a lot of information, e.g. the "hardness" of an instance or the "qualitiy" of a solution, that requires advanced information processing. To cover both of these scenarios, we provide two major mechanisms to extract IP's and RP's : by an internally defined parser, that can work with regular expressions, or by an external program.

Before starting to extract information, the user has to define the properties in the EDACC GUI by specifying the name, value type, description, source and the regular expression or external program. The value type of the property can be chosen from several predefined types like boolean, integer, float or string. To make the information extraction as flexible as possible, the user is also able to define further types and also to specify if the property has multiple occurrences. If the property's computation mechanism is an external program, the user has to provide a binary and a parameter line to run the program. The stdout output of the program is then interpreted as the value of the property.

Properties are stand-alone entities, and do not require the existence of instances or of results. Starting the computation of a property creates a link between the property and the instance or the result. The link contains the value of the property.

The computation of properties can take a long time, depending on the complexity and size of the input. To take advantage of current multi-core computer architectures EDACC can parallelize the computation of properties.

3.1 Instance properties

Instance properties can be computed in the manage DB mode of EDACC and are independent of the existence of an experiment. Information about instances can be also parsed from the instance filenames. This can be very useful when the filename encodes different properties. After their computation, IP's can be displayed within the GUI, or can even be used to filter instances, according to certain values of a property. This feature can be very useful when selecting the instances for an experiment. Further, all computed IP's are available for use in the WF.

3.2 Result properties

Result properties can be computed in the experiment mode of EDACC and assume the existence of an experiment. Most of the RP's, excepting those com-

www.manaraa.com

puted from solver parameters, can be computed only when a job is finished and the output files of solver, launcher, watcher (and verifier) are available in the DB. The computation of result properties can be started during the computation of an experiment because EDACC will take only finished jobs into consideration. Thereby preliminary analysis of the results and their properties is possible. Computed RP's can be displayed in the result browser or can be used in the WF. There are some predefined RP's within EDACC that do not have to be computed: the result time (the time it took to compute the result) and the parameters of a solver.

4 Analysis and statistical evaluation

Through its information extraction mechanism, EDACC provides a lot of information about an experiment. Having all this sort of information in the same DB we have extended the GUI and the WF to provide also descriptive statistics and statistical tests. This can be for example used to measure the performance of algorithms, to find out correlations between some properties of the results or to simply have a graphical representation of the results. This enables the user to directly analyze the results without having to export the data, and then process them within a statistical program.

The information that can be used for analysis is stored in the DB within IP's and RP's. We differentiate between two scenarios in which analysis is performed. Analysis of a single solver or comparison of two or more solvers. We also have to differentiate between single runs or multiple runs of a solver on the same instance. If multiple runs are available, the information used for statistics can be chosen by the user from median, mean, all runs or only a single specified run.

To improve the statistical methods the user has also the ability to select the instances used for the analysis. For example when analyzing the results of SAT solvers on random instances containing 3-SAT, 5-SAT, and 7-SAT instances, the user might be only interested in 3-SAT. This can be performed by choosing only the 3-SAT instances for the analysis. Instance selection is provided for all methods.

A RP distribution plot (see Fig. 3) and a nonparametric kernel density estimation is provided for the analysis of the results of a single solver on an arbitrary instance by means of an arbitrary RP. To analyze the results of a solver on all instances (or a selection) the user can use scatter plots. The compared information can be an IP with a RP, like for example number of variables vs. CPU time or two RP's, like memory-usage vs CPU-time. Beside the scatter plots we also compute the Spearman rank correlation coefficient and the Pearson product-moment correlation coefficient.

A scatter plot (see fig. 4 for a run time comparison) together with the two mentioned correlation tests is provided for the comparison of two solvers by means of an arbitrary result property. When the comparison is limited to one instance we also provide RP distributions comparisons together with a Kolmogorow-Smirnow two-sample test and a Mann-Whitney-U Test (Wilcoxon

www.manaraa.com

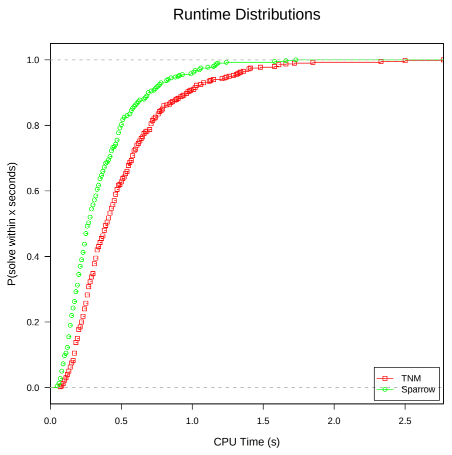


Fig. 3. Comparison of the runtime distribution of two solvers.

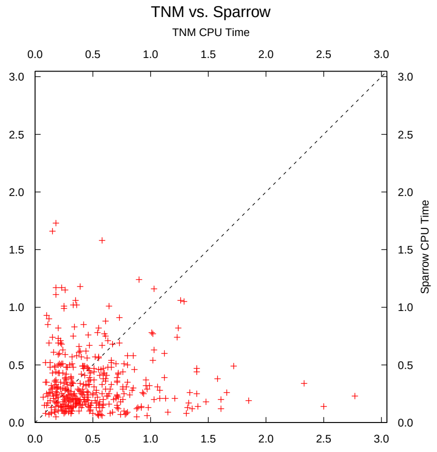


Fig. 4. Scatter plot to compare the runtime of two solvers.

rank sum test). The RP distribution comparison plot can be also done for all solvers but without the tests.

A well founded comparison of the performance of two solvers can also be done with the help of a probabilistic domination test by means of an arbitrary RP. Within this tests instances are split into three categories. The first category contains the instances where the first solver probabilistically dominates the second one. The second one contains the instances where the second solver probabilistically dominates the first one and the third category contains the instances where no probabilistic domination can be found because of the crossing of the RP distributions.

Analyzing one result property for one or more solvers can be done by a box plot or by a cactus plot (number of solved instances within a given amount of the RP's see Fig. 7) .

Finally EDACC can export the generated plots in a huge variety of file formats including vector graphics. To support third-party analysis tools IP's and RP's can also be exported to the widespread csv-format.

5 EDACC - Competition Mode

Solver competitions can be an incentive for researchers to implement new ideas, to improve existing solver and spark interest in the field. Recurring competitions can show the progress in the development of solvers by comparing new solvers with reference solvers from previous competitions. They can also help to identify challenging instances for state-of-the-art solvers. The results of such a competition can be used by researchers to identify the strengths and weaknesses of solvers and instances and to guide further development.

www.manaraa.com

There are several competitions in the field of empirical algorithms, for example the "SAT Competition"[6][7], the "SAT-Race"[8], the "SMT-COMP"[9] or "CASC" [3]. Running such competitions is an organizational challenge and comes with the inevitable need for tools to make it possible to run dozens of solvers on a huge set of instances in a multi-computer environment and then retrieve and process the results for competition purposes. The competitions mentioned above do have such internal tools and web interfaces, but to our knowledge they are not publicly available. To make the organization of competitions to everybody possible, (who has the computational resources) we decided to extend EDACC to be able to provide all required functionalities for the organization of competitions.

We first started by analyzing the existing competition systems to find out their commonalities and to identify interesting or missing features.

From an abstract point of view all competitions have:

1. static web pages to provide information about rules and the course of events
2. user administration to control the access to the results
3. an execution system to run solvers and manage the results
4. dynamic web pages to present the results

As necessary, interesting or missing features we have identified:

1. Plausibility and verifiability of the steps taken in all competition phases by providing participants real-time access to all relevant information.
2. The results have to be reproducible, which means all required information (e.g. starting command, seeds, input files, output files) should be easily accessible through a web interface.
3. Various forms of presentation of the results with cross linking and filtering.
4. Different graphical presentations of the results, including interactive elements such as clickable points in plots that lead to detailed information.
5. All graphical presentations are exportable both as image and as numerical data.
6. Descriptive statistics and statistical tests for analysis of the results.
7. Clean encapsulation of the ranking system enabling easy implementation of new ranking systems.

We have extended the WF of EDACC to provide together with the GUI and CC all of these features. Further we have added a phase system (see Fig. 5) to specify the course of events during a competition. The phases also specify which actions should be taken by whom and control the access to the various information.

Next we are going to describe the organization of a competition with the EDACC WF by describing each phase, and pointing out the interesting features that are provided. The access control to different kinds of information (e.g. own results, all results, statistics, etc.) can be configured by the organizers for each phase individually, according to their competition policies. Through the description of the phases an exemplary access control is given.

In the first phase the organizers of the competition define the competition categories (which actually can be seen as sub-competitions). A category is defined

www.manaraa.com

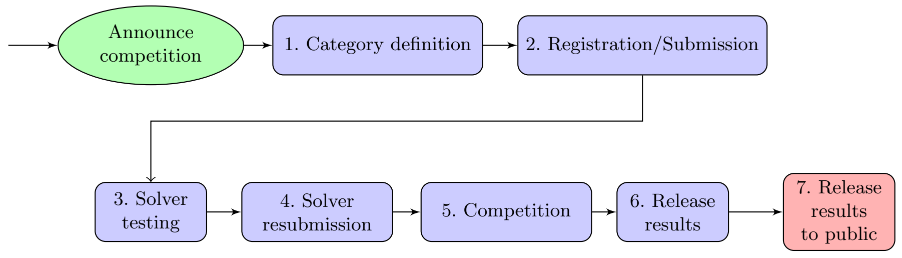


Fig. 5. The phases of a competition

by the instances it will contain and should give the competitors an orientation where to submit their solvers. In EDACC, each category will be represented by an experiment. In this phase competitors have access only to general information, rules and the schedule. The WF provides containers for these static web pages.

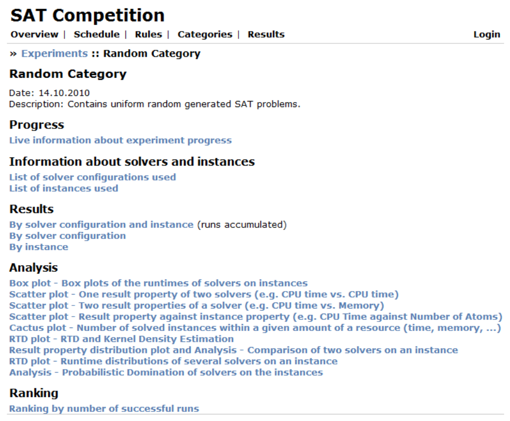
In the second phase, competitors are requested to create an account for the web interface. After login they can submit their solvers (i.e. source code or binary), which are directly saved within the DB. They have to provide detailed information about their solvers like the parameters and the competition category where the solver should participate. Instances can also be submitted by specifying the origin, type and the category it would suit best. Submitted instances will be then available to organizers in the EDACC DB. During this phase competitors have no access to other competitors' solvers nor instances. The WF together with the DB provides the necessary access controls.

The solver testing phase is used to ensure that the submitted solvers are able to run on the computing system of the competition. Within the EDACC GUI organizers create test experiments, corresponding to each of the competition categories. Creating this experiments is straightforward, because solvers and instances are already in the DB. Each solver will be tested in all categories it was submitted to. The experiments are then run on the competition computing system with the help of the CC. Competitors have the possibility to real-time monitor their solvers through the WF (results of other solvers are not visible). Registration and submission of solvers or instances is no longer possible within the WF. From this phase on results are accessible in several forms [1]:

1. By solver configuration: The results for all instances computed by a solver configuration.
2. By instance: The results of all solver configurations.
3. By solver configuration and instance (if multiple runs are allowed): multiple jobs of each solver configuration on an instance are accumulated and some descriptive statistics like the minimum, maximum, median and mean runtime displayed.

[1] An example for the results of a competition can be found at `http://edacc.informatik.uni-ulm.de/`

www.manaraa.com



SAT Competition

Overview | Schedule | Rules | Categories | Results Login

» Experiments :: Random Category

Random Category

Date: 14.10.2010
Description: Contains uniform random generated SAT problems.

Progress

Live information about experiment progress

Information about solvers and instances

List of solver configurations used
List of instances used

Results

By solver configuration and instance (runs accumulated)
By solver configuration
By instance

Analysis

Box plot - Box plots of the runtimes of solvers on instances
Scatter plot - One result property of two solvers (e.g. CPU time vs. CPU time)
Scatter plot - Two result properties of a solver (e.g. CPU time vs. Memory)
Scatter plot - Result property against instance property (e.g. CPU Time against Number of Atoms)
Cactus plot - Number of solved instances within a given amount of a resource (time, memory, ...)
RTD plot - RTD and Kernel Density Estimation
Result property distribution plot and Analysis - Comparison of two solvers on an instance
RTD plot - Runtime distributions of several solvers on an instance
Analysis - Probabilistic Domination of solvers on the instances

Ranking

Ranking by number of successful runs

Fig. 6. View of the WF showing the result phase of a competition.

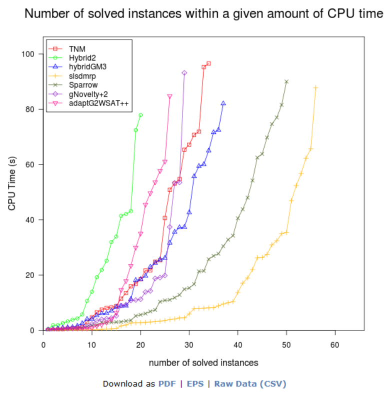


Fig. 7. Cactus plot of the results during the run of a competition.

4. Single result: The result of a single job, including the output of solver, launcher, watcher and verifier and also all result properties that where computed for this result.

During a solver resubmission phase, competitors have the opportunity to submit solver updates if bugs or compatibility issues with the computing system occurred during the test phase. The organizers can then rerun the testing experiments with the updated solvers.

Similar to the testing phase, in the competition phase organizers create experiments based on the competition categories and choose the solvers and instances for each experiment. This task is again accomplished with the help of the GUI. The experiments are then run on the computing system and competitors have the possibility to monitor the results of their own solvers online (and of others if configured so by the organizers).

In the release phase competitors gain access to the results of all competing solvers. Before making the results available to the wide public a ranking has to be calculated. The ranking can either be calculated dynamically by the web application or simply displayed after a manual calculation. We implemented a simple, exemplary ranking using the number of correct results and breaking ties by the accumulated CPU time. Further rankings can be easily encapsulated within the application. Also available in this phase is the complete spectrum of descriptive statistics and statistical tests described in chapter 4. For pointing out interesting results or correlations the organizer have the possibility to extract instance or result properties within the GUI and make them available within the WF.

In a last phase, instances, results and possibly solver source codes and binaries are made publicly available on the web interface without requiring registration.

www.manaraa.com

6 Implementation Details

The first component of EDACC, the DB, requires an user-account on a MySQL 5.1 database with read and write access. The location of the DB plays no role. The needed tables are generated by EDACC itself. The GUI of EDACC is written in Java and is independent of the operating system of the computer. It needs only the Java virtual machine version 6. For the statistical evaluation, the R programming language should also be installed on the computer.

The compute client consists of three sub programs: the launcher, the watcher and the verifier. The launcher builds a DB-connection, and is responsible for fetching the jobs and all necessary files, providing them to the watcher. The launcher is written in C and was tested only on unix-like systems. The watcher starts the solver, and monitors the consumption of resources on the computing system. If some limits are exceeded the solver will be stopped. At the moment we use the runsolver code of Olivier Roussel from the SAT Competition as a watcher. The watcher is a replaceable component in EDACC. The verifier is problem dependent and has to be provided by the user. If the results of the solver can be trusted (e.g. the solver contains a verifier procedure) the verifier can be omitted.

A MySQL proxy is provided to make the execution of the CC on computer clusters possible, where the nodes do not have Internet access, except for a login node. In such a scenario the MySQL-proxy running on the login node provides the DB-connection for the CC's. This feature was tested on several computer clusters.

The web interface for the competition mode is implemented as Python WSGI (Web Server Gateway Interface) application. The application uses a web framework and several open source libraries which are available on most platforms. All competition specific data like user accounts, instance types and the phase of the competition are stored in the central DB. To generate plots and calculate statistics it uses an interface library to the statistical computing language R.

The code of EDACC components is open source and is released under the MIT License (excepting the watcher, which has an GPLv3 license). The code is available at the project site: `http://sourceforge.net/projects/edacc/`.

7 Related Work

We are not aware of the existence of an experimentation system for empirical algorithms that provides all the functionalities of EDACC within the same platform. Parts of EDACC's functionalities are provided by different systems or tools. GridTPT [4] for example supports the testing of SMT solvers and their distribution on computer clusters supporting a master/slave architecture. It is also able to parse information from the output and present some statistics as scatter plots.

The different competitions like [7] and the SMT Competition [9] systems have several tools similar to our WF but they lack the possibility to perform

www.manaraa.com

advanced analysis of the results and are not freely available nor portable to other computing systems.

8 Conclusion and future work

In this work we have introduced EDACC, a platform for the design, administration and analysis of experiments on empirical algorithms. EDACC consists of four major components, the database, a graphical user interface, a compute client and a web frontend. The DB is the central information storage of EDACC and provides the communication link between GUI, CC and WF. The GUI enables the user to manage solvers, their parameters and instances within the DB. It also enables the design and creation of complex experiments and their administration on different computing systems. The compute client performs the computation of the experiment jobs on arbitrary computing systems ranging from multi-core computers to large scale grids. The architecture of the compute client is designed to use the allocated resources to a maximum, increasing the computational throughput. Crashes of parts of the computational system do not affect the processing of experiment jobs, as failed jobs can be recomputed by other CC's. During the computation of an experiment the GUI and the WF provide a job browser to monitor the jobs. They also provide a wide variety of statistical analysis methods like descriptive statistics and statistical tests. For organizing solver competitions the WF provides all necessary functionalities like user administration, and different dynamic web pages for monitoring the course of events. The statistical analysis possibilities are also provided for the competition mode, enabling a fast evaluation of the results.

We think that researchers, that study empirical algorithms, can drastically speed up their experimental and analysis work by using EDACC as their experimental platform.

In the further development of EDACC we plan to integrate an automatic parameter optimizing procedure. Together with the distributed computing possibilities of EDACC, the optimization process could be sped up. We also plan to integrate different priority policies for processing the jobs within an experiment. For the competition mode of the WF an automated compilation of the source code (which is submitted by the competitors) on the computing system is planed.

Acknowledgments: We would like to thank the bwGrid [2] project for providing the test environment, and Borislav Junk and Raffael Bild for the first version of the launcher code. We would also like to thank Geoff Sutcliffe for fruitful suggestions regarding the first version of this paper.

References

1. Balint, A., Gall D., Kapler G., Retz, R., Experiment design and administration for computer clusters for SAT-solvers (EDACC). JSAT Volume 7 (2010), system description, pages 77-82.

www.manaraa.com

2. bwGRiD (http://www.bw-grid.de), member of the German D-Grid initiative, funded by the Ministry for Education and Research (Bundesministerium für Bildung und Forschung) and the Ministry for Science, Research and Arts Baden-Württemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg)
3. Sutcliffe, G.: The CADE-22 Automated Theorem Proving System Competition CASC-22 2010, AI Communications Journal 23 Number 1 pages 47-60
4. Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe and Pascal Fontaine: GridTPT: a distributed platform for Theorem Prover In Proc. Workshop on Practical Aspects of Automated Reasoning 2010
5. Frank Hutter, Holger Hoos, and Thomas Stützle: Automatic Algorithm Configuration based on Local Search In AAAI-07.
6. Le Berre, D., Simon, L.: The essentials of the SAT 2003 competition In Sixth International Conference on Theory and Applications of Satisfiability Testing, volume 2919 of LNCS, pages 452-467.
7. The SAT Competition Homepage: `http://www.satcompetition.org`
8. SAT-Race 2010 Homepage: `http://baldur.iti.uka.de/sat-race-2010/`
9. Barrett, C., De Moura, L., Stump, A.: SMT-COMP: Satisfiability Modulo Theories Competition In CAV'05, LNCS 3576
10. Homepage of the project : `http://sourceforge.net/projects/edacc/`

المنارة للاستشارات

www.manaraa.com