Towards a Knowledge Base for Performance Data: A Formal Model for Performance Comparison

Hans Degroote
KU Leuven
Department of Computer Science
CODeS & iMinds-ITEC
Hans.Degroote@kuleuven-kulak.be

Patrick De Causmaecker
KU Leuven
Department of Computer Science
CODeS & iMinds-ITEC
Patrick.DeCausmaecker@kuleuven-kulak.be

ABSTRACT

This paper has been motivated by two observations. First, empirical comparison of algorithms is often carried out in an ad hoc manner. Second, performance data is abundantly generated, yet often not efficiently used. This second observation is particularly valid in the presence of evolutionary computing and other metaheuristic techniques. Inspired by these observations, a formal model for performance is introduced wherein the space of possible performances is modelled as a total order. On top of the total order, a quantification of the difference between performances is defined. The model is illustrated by formally defining the ”penalised runtime” criterion for data from the 2014 SAT competition. Finally, the idea of defining questions in terms of a formal performance model is introduced, thereby taking the first step towards a knowledge base for performance data. Regardless of problem domain, the same questions can be answered by the knowledge base, provided performance is measured in a manner compliant with the formal model.

Categories and Subject Descriptors

D.4.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

Algorithm comparison, Performance measurement, Metaheuristics

1. INTRODUCTION

Experimental comparison of algorithms is often carried out in an ad hoc manner. Performance data is abundantly generated nowadays, but its potential remains largely untapped. These two observations serve as motivation for the research presented in this paper: a first step towards a knowledge base for performances. When a set of instances, a set of algorithms and the performance of each algorithm on each instance is input, a number of well-defined questions can be answered. Two examples of common straightforward questions are ”which algorithm is best suited to solve a particular instance?” and ”How much does a particular new algorithm improve on the state of the art algorithm?”. However, more complex questions are definable as well.

To transform questions such as the ones raised into an operational definition, a mathematical model for performance measurement is introduced. Performance measurement is modelled in two stages. At first the user defines a performance function subject to only one condition: performances must be totally ordered. In a second stage the user quantifies the difference between any two performances with a quantification function.

Section 2 introduces a formal model for performance comparison. Section 1.1 defends the claim that this model can be used to improve the reliability of algorithm comparison. Section 3.1 discusses the importance of explicitly defining how performance is measured. Section 3.2 provides an example of such an explicit definition, based on data from the 2014 SAT competition. Section 4 introduces the methodology to extract knowledge from performance data. The relevance of a knowledge base for performance data is argued in section 1.2 by pointing out three situations in which a large amount of performance data is naturally produced, yet often not efficiently used.

1.1 Relevance for algorithm comparison

Stating that a particular algorithm is better than another algorithm is a non-trivial statement. Ideally it is supported by statistical evidence. But even when this statement is supported by statistical evidence, it might not be clear what exactly was being measured in the first place. What does a performance of 1000 mean? Is it better than a performance of 500? Is it twice as good?

This paper provides a methodology to support the statement that a particular algorithm is better than another algorithm, but it requires a more nuanced formulation: ”An algorithm is better than another algorithm on a particular instance set, according to a particular performance function and a particular quantification function”. It is the responsibility of the researcher who carried out the experiment to convince the audience of the rationality of his choice of performance function and quantification function. The researcher also has to convince the audience that the instance set is representative and that the results can be extrapolated to other instances. Smith-Miles et al. [2] have proposed a

GECCO '15, July 11 - 15, 2015, Madrid, Spain
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3488-4/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739482.2768476

methodology for generating a diverse set of instances that can be used in this context.

1.2 Situations where performance data is generated

Three situations where performance data is generated will be discussed: hands-on experimentation, automatic algorithm configuration and algorithm competitions.

Consider an algorithm developer who has implemented multiple solutions strategies. To determine which solution strategy is best, the developer will test each strategy on a set of instances. Multiple solution strategies occur naturally in many modern algorithmic approaches, particularly in metaheuristics. A different genome representation in an evolutionary algorithm defines a new solution strategy. A different neighbourhood structure in a local search algorithm defines a new solution strategy. Modifying any numerical parameter, such as the amount of particles in particle swarm optimisation, or changing the annealing schedule of a simulated annealing algorithm, all define new solution strategies.

The testing process generates a wealth of data. Unfortunately, a lot of knowledge remains hidden because no general knowledge extraction method exists. The developer might not want to spend time investigating, for example, the performance gain achievable by running multiple solution strategies in parallel. Yet the input required to calculate this performance gain is present in the test data.

The knowledge base presented in this paper can be used as a data management and knowledge extraction tool for algorithm evaluation. If the algorithm developer transforms his raw experimental data to a performance function and a quantification function, he or she can immediately query the system for information. It is no longer necessary to manually analyse, in the worst case, a spreadsheet with raw performance data.

When the space of solution strategies consists of different parametrisations of a single algorithm, automatic algorithm configuration can automate the testing process. Automatic algorithm configuration techniques search the space of possible parameter values for an optimal configuration. A more formal definition as well as an overview of the state of the art can be found in [1]

Algorithm competitions result in vast amounts of performance data generated in identical experimental conditions. Still, the wealth of information that this data can provide remains largely unexplored. In [3], Xu et al. perform some analyses on data generated during the 2011 SAT competition. They formulate interesting questions related to measuring the value of a solver in algorithm portfolios and automatic algorithm selection techniques, but do not provide formal definitions. These questions could be added to the knowledge base, as discussed in section 6.

2. MODEL

2.1 Comparing performances

Let $\mathcal{I}$ be a finite set of instances of some problem domain. Let $\mathcal{A}$ be a finite set of algorithms. Let pf be a performance function, describing the performance of each algorithm on each instance. The domain of pf is the Cartesian product of the sets of all instances and all algorithms. The image of pf is some set $\mathcal{P}$ consisting of all possible performances. Using these notational conventions, a performance function can be defined as in definition 1. $pf(a, i) = p$ should be read as "the performance of algorithm a on instance i is p.

Definition 1. $pf : \mathcal{A} \times \mathcal{I} \to \mathcal{P} : pf(a, i) = p$

The question which of two performances is better lies at the heart of a lot of research in empirical computer science. Ensuring that each two performances are comparable is done by imposing a total order on the image of the performance function. This entails that performances can be compared using some binary relation '$\leq$'. Stating that '$p_1 \leq p_2$' is stating that performance p_1 is worse than or equally good as performance p_2. A total order must uphold three properties for all elements a, b and c of the set on which it is imposed:

- $a \leq b \vee b \leq a$ (totality)
- $(a \leq b \wedge b \leq a) \Rightarrow a = b$ (antisymmetry)
- $(a \leq b \wedge b \leq c) \Rightarrow a \leq c$ (transitivity)

Totality ensures that all performances are comparable to each other. Antisymmetry ensures that for every two distinct performances, it is possible to say which is better. Transitivity ensures consistency.

2.2 Measuring performance difference

Modelling performances as belonging to a totally ordered space permits comparison, but it remains impossible to quantify performance differences. Quantification becomes possible by mapping each performance to a unique real number. This mapping is defined by a quantification function qf, with as domain a performance space and as image the real numbers. Any quantification function must adhere to the following two properties:

- $qf(p_1) =_n qf(p_2) \Leftrightarrow p_1 = p_2$
- $qf(p_1) \leq_n qf(p_2) \Leftrightarrow p_1 \leq p_2$

Quantifications are ordered according to the natural ordering of the real numbers. $\leq_n$ *and* $=_n$ refer to this natural ordering of the real numbers. The first property states that each performance must be mapped to a unique real number. The second property ensures consistency between the ordering of performances and the ordering of their quantifications.

The difference between two performances p_1 and p_2, according to a particular quantification function qf, is equal to $|qf(p_1) - qf(p_2)|$. Note that it is the user's responsibility to ensure that this difference has a quantitative interpretation. Section 3.1 expands on this remark.

3. DEFINING PERFORMANCE AND QUANTIFICATION FUNCTIONS

3.1 Def ning a correct function

It is important to keep in mind that answers to queries to the knowledge base, of which some examples are given in section 4, provide information for one specific performance function and, when relevant, for one specific quantification function. Defining a new performance function corresponds to defining a new way of ranking performances.

Consider a decision problem with time limit. Some algorithms might excel at solving instances within the time limit.

Other algorithms might obtain solutions more quickly on average, but might not solve as many instances as the first kind of algorithms. To focus on algorithms of the first kind, a binary performance function can be defined: either an instance is solved, or it is not. All algorithms that solve an instance within the time limit solve the instance equally well, regardless of the actual time taken. To focus on algorithms of the second kind, a continuous performance function can be defined. An algorithm that solves a particular instance performs better than another algorithm that also solves the instance, if the second algorithm needs more time than the first. Both approaches to measuring performance are valid. Preferability is situational.

Using a different quantification function can also alter results. Consider again a decision problem with time limit. Assume the performance function is continuous and that less time taken results in better performance. A straightforward quantification can be defined by considering the value of the runtime. However, improving runtime from 5 seconds to 2 seconds might be more relevant than improving runtime from 1000 to 997 seconds. A linear quantification function cannot represent this. Some logarithmic function might be better. The ranking of performances remains constant, regardless of which of the two quantification functions is used, because ranking is determined by the performance function. But, the difference between two performances might have changed, which will influence the answers to more complex questions that require information from the quantification function.

3.2 An example: SAT

To illustrate the model presented in section 2, it will be applied to a specific problem domain. The example uses data of the "Sequential, Application SAT+UNSAT" track of the 2014 SAT competition. This data is available from the website of the SAT competition. [1] The SAT competition is a recurring competition for algorithms that solve the boolean satisfiability problem. The algorithms' goal is to either prove that a boolean formula is satisfiable or that it is not satisfiable, within a certain time limit.

The algorithm set $\mathcal{A}$ contains more than 30 algorithms. The instance set $\mathcal{I}$ consists of 300 instances. This dataset represents millions of CPU-seconds, yet uploading this readily available data into a knowledge base requires a trivial amount of time.

The SAT competition website reports the results in two files: an answer file and a runtime file. Data in the answer file takes one of four forms. If the instance is proven satisfiable within the time limit, "SAT" is reported. If the instance is proven not to be satisfiable within the time limit, "UNSAT" is reported. If the algorithm executed normally, but failed to obtain an answer, "time limit exceeded" is reported. In case of an error, "unknown" is reported. Data in the runtime file takes one of two forms. If the instance is proven to be satisfiable or proven not to be satisfiable within the time limit of 5000 CPU seconds, the total CPU-time in seconds, with up to three decimal digits, is reported. If the instance remains unsolved within the time limit, or if some error occurred, the maximal runtime (5000.0) is reported.

Data from the two result files is combined to create a performance function. The performance domain $\mathcal{P}$ is defined as follows:

$$\mathcal{P} = [0, 5000] \cup \{\mathit{TIMEOUT}\}$$

The performance function pf is defined as:

$$\begin{aligned} pf : \mathcal{A} \times \mathcal{I} \to \mathcal{P} :& pf(a,i) = \\ & rt(a,i) \;\; [ans(a,i) = SAT \lor ans(a,i) = UNSAT] \\ \mathit{TIMEOUT} &\;\; [\mathit{otherwise}] \end{aligned}$$

Here, $rt(a,i)$ returns the result of algorithm a on instance i according to the runtime file, and $ans(a,i)$ returns the result of algorithm a on instance i according to the answer file. The actual runtime is used when a solution is found. "$\mathit{TIMEOUT}$" is reported when the time limit is exceeded and in case of an error. Note that the interval $[0, 5000]$ implies that the set of performances is infinite. In this example, performances could also be modelled as the union of the singleton $\mathit{TIMEOUT}$ and the finite set of all numbers between 0 and 5000 with up to three decimals, because the raw data is limited to this finite set of numbers.

The total order on $\mathcal{P}$ that will be used is:

$$\begin{aligned} \forall p_1, p_2 \in \mathcal{P} :& p_1 \leq p_2 \Leftrightarrow \\ & (p_1 = \mathit{TIMEOUT}) \\ & \lor (p_1 \in [0, 5000] \land p_2 \in [0, 5000] \land p_1 \geq_n p_2) \end{aligned}$$

The $\geq_n$ in the last line refers to the natural ordering on the real numbers. It can be used because for both p_1 and p_2 is checked that they belong to an interval of real numbers. $\mathit{TIMEOUT}$ is worse than any solution found within the time limit. For solutions found within the time limit, requiring less time to find the solution results in better performance.

The quantification function is based on the PAR_{10} criterion. PAR is short for "penalised runtime". PAR transformations are used to measure performance for decision problems when runtime is limited. They penalise time-outs to better differentiate a time-out from a solution found only just within the time limit. With PAR_{10}, runs completed within the time limit are mapped to themselves and time-outs are mapped to ten times the time limit. The time limit is 5000, resulting in the following quantification function:

$$\begin{aligned} qf : \mathcal{P} \to \{0\} \cup [45000, 50000] :& qf(p) = \\ 0 &\;\; [p = \mathit{TIMEOUT}] \\ 50000 - p &\;\; [p \in [0, 5000]] \end{aligned}$$

The model requires lower quantifications to be worse than higher quantifications. Therefore a $\mathit{TIMEOUT}$ has performance 0. The theoretically best possible quantification achievable with this quantification function is 50000, corresponding to finding a solution instantly.

4. FORMULATING QUESTIONS

4.1 Using the performance function

The most basic question to be asked is "What is the performance of an algorithm on an instance?". This question is trivially answerable using this paper's model if "according to a particular performance function" is appended. The answer to this question is the result of applying the performance function to the algorithm and instance.

Because the set of performances is totally ordered, the performance of algorithms on instances is comparable. Therefore questions such as "Is a particular algorithm better than

[1] http://satcompetition.org/2014/results.shtml

another algorithm for a particular instance, according to a particular performance function?" and "Is a particular instance harder than another instance for a particular algorithm, according to a particular performance function?" become answerable.

Best performance on an instance

One of the most basic non-trivial questions is: "What is the best performance achievable on a particular instance, according to a particular performance function?". This best performance is defined in definition 2.

Definition 2. $bestPerformance(i) =$
$p \in \mathcal{P} : ((\exists a \in \mathcal{A} : pf(a,i) = p) \wedge (\forall a \in \mathcal{A} : pf(a,i) \leq p))$

The best performance achievable on an instance satisfies two requirements. First, at least one algorithm must be mapped to it. Second, there exists no algorithm with strictly better performance on the instance.

Note that the best algorithm to solve an instance is not uniquely defined. Only the best performance achievable by an algorithm is uniquely defined. Multiple algorithms might achieve this best performance.

4.2 Using the quantif cation function

The ability to rank performances suffices to answer simple questions. However, a lot of interesting questions require quantification. The most basic question to be asked that involves quantification is: "How large is the difference between two performances?". The most common variant of this questions is probably: "What is the difference between the performance of two particular algorithms on a particular instance?". But, "How much harder is it to solve instance i_1 than it is to solve instance i_2, using a particular algorithm?" is also a possibility. These questions are all directly answerable when a quantification function has been defined.

Algorithm comparison on a set of instances

To enable well-defined comparison of algorithms on an instance set, the difference between two algorithms is defined. Define the difference between algorithms a_1 and a_2 on a set of instances J, for a particular performance function pf and a particular quantification function qf, as in definition 3.

Definition 3.

$$diff(a_1, a_2, J) = \sum_{i \in J} qf(pf(a_1, i)) - \sum_{i \in J} qf(pf(a_2, i))$$

If $diff(a_i, a_2, J)$ is 0, both algorithms perform equally well on instance set J according to the given performance and quantification function. If the difference is positive, algorithm a_1 outperforms a_2. If the difference is negative, algorithm a_2 outperforms algorithm a_1. The size of the difference is the amount by which the better algorithm outperforms the other.

5. CONCLUSION

This paper has proposed a formal model for describing the performance of an algorithm on an instance. The model facilitates precise algorithm comparison by requiring an explicit description of how performance is measured. This explicit description consists of two functions: a performance function to order performances and a quantification function to quantify the difference between performances. It was exemplified that the same problem domain can be represented by different functions and that different functions can yield different results for algorithm comparison. The model was illustrated by formally describing the results of the 2014 SAT competition using the "penalised runtime" performance criterion.

This paper has also proposed a first step towards a knowledge base for performance data. If questions are formulated in terms of a formal model, they are applicable to any problem domain for which a performance and quantification function have been defined. As an example, the basic question of what the best achievable performance on an instance is, was defined in terms of the model presented in this paper. The question which of two algorithms is best on an instance set, and by how much, was also formally defined. A general knowledge extraction process could prove particularly useful in the presence of algorithm competitions and extensive testing: the tedious work of generating the data required to obtain knowledge has already been completed.

6. FUTURE RESEARCH

Two immediate directions for future research exist. First, performance and quantification functions for common problem domains can be defined. If a pair of performance and quantification functions is generally accepted, algorithm comparison can become clearer. Algorithms can even be ranked based on different performance and quantification functions.

A second immediate direction for future research is to define more elaborate questions, thereby extending the utility of the knowledge base. A promising area for which a knowledge base of performances might be useful is that of algorithm portfolios and automatic algorithm selection. Methods for these fields generally require an overview of performance data for a set of algorithms and instances, which is exactly what this knowledge base stores. If relevant questions such as "How large is the potential for algorithm selection?" and "What is the optimal portfolio of algorithms?" are formulated in terms of a formal model, they become quickly answerable for every problem domain.

7. ACKNOWLEDGMENTS

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX. (http://comex.ulb.ac.be)

8. REFERENCES

[1] Hoos, H. H. Automated algorithm configuration and parameter tuning. In *Autonomous search*. Springer, 2012, pp. 37–71.

[2] Smith-Miles, K., Baatar, D., Wreford, B., and Lewis, R. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research 45* (2014), 12–24.

[3] Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer, 2012, pp. 228–241.