Computing Science Group

Dominance: Consistently Comparing Computational Complexity

Ed Blakey
edward.blakey@queens.ox.ac.uk

CS-RR-08-09

Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford, OX1 3QD

Abstract. Though complexity theory strives primarily to categorize *problems* according to their complexity, it is typically the complexity only of *methods of solving problems* that we can directly measure. Specifically, we have an upper bound for a problem's complexity: namely, the complexity of the most efficient known solution method for the problem. In order to improve such bounds, it is desired to consider sets of solution methods that are as complete as possible; but, whilst comparisons of computing systems' respective efficiencies can—via $\mathcal{O}$-notation—readily be made *when the systems conform to the same computational model and when the comparison is with respect to the same resource*, it is not clear how to compare model-heterogeneous sets of solution methods—this imposes a constraint on the size of sets of methods that can be considered, and a corresponding constraint on the quality of bounds on the complexity of problems.

Here, we propose the notion of *dominance* as a way of augmenting the power of $\mathcal{O}$-notation to compare systems' complexity. In particular, our augmentation allows meaningful comparison of the respective complexities, with respect to *different resources*, of computing systems that conform to *different computational models*, thus allowing consideration of larger, model-heterogeneous sets of solution methods and, hence, improved bounds on problems' complexity.

1 Preliminaries

1.1 Computation

Many devices and systems (including, but by no means limited to, digital computers) can be said to *compute*. As long as a system has provision for accepting *input values* and—usually after some form of processing—for supplying *output values*, then it computes; what it computes is the relation—often a function—between input and output values.[1]

For example,

- a digital computer's input values may be typed on a keyboard and its output values presented on a screen (with algorithmic processing—implemented as a program, which is in turn implemented electronically—occurring between input and output);
- a chemical computer's input and output values may be encoded in the concentrations of solutions respectively supplied to and drawn from the system (with chemical reaction occurring between input and output);
- and so on.

[1] An equivalent interpretation of this computation relation is as a *problem*, which is of the form 'given a problem instance (i.e., input value), find the/an answer (i.e., output value) to the question that defines the problem'. Just as we say that a system performs a computation/computes a relation, we say that it solves the corresponding problem.

More generally, computing devices/systems accept input values encoded in the values of manipulable parameters, and supply output values encoded in the values of measurable parameters, of the system[2] (with processing, in the form of the system's evolving in accordance with the laws—be they electrical, chemical, etc.—to which it is subject, occurring between input and output).

1.2 Resource

A small set of 'trivial' (for example, identity- or constant-function) computations aside, computation is not free: the act of computing consumes *resource*. A natural question—when considering computation both theoretically and practically—is,

what types of resource are required (and in what quantities) by a given computational system as it processes a given input value?

We may measure, for example, the *time* for which we have to wait before an output value is supplied, the volume of physical *space* occupied by the system (including any provision for data storage that it may require in processing the given input), or the *precision* with which we must supply input and measure output values in order that the correct computation be performed (see [3]); each is a valid and potentially insightful measure of resource.

We adopt the convention of using upper-case letters A, B, C, etc. to denote resources, which we view as functions

- that depend on the choice of *computational system* (shown as a subscript, which, when understood, is often suppressed), and
- that map an *input value* to the corresponding amount of *resource* used by the system in processing this value.

Hence, where Φ is a computer (that is, a system—any system—that performs computation) and x is an input value for Φ, $A_\Phi(x)$ (or simply $A(x)$) is the amount of resource A consumed by Φ in processing x.

1.3 Complexity

Given a computational system, and considering a specific resource, we may ask how this resource scales. In particular, we may be interested not in the resource used by the system given *one specific input value* (that is, in some '$A(x)$'), but in the resource used *as a function of the size of the input value*[3]—this is what we

[2] There is a side issue that arises when one acknowledges that, in the non-Turing realm of physical computing, the *precision* with which parameters are manipulated and measured can affect the computation; we discuss this, and accordingly introduce *precision complexity*, in [3].

[3] We require, then, a *size function* σ that maps input values to non-negative reals (the input values' 'sizes'); for example, if our input value is a natural number n expressed in binary, then we can take as its size the number of bits, excluding leading zeros (or, as is sufficient for virtually all complexity-theoretic purposes, the approximation $\log_2(n)$ to this number of bits).

mean by the *complexity function* corresponding to the resource. Specifically, for computer Φ and size function σ, the complexity function $AC_{\Phi,\sigma}$ (corresponding to a given resource A) evaluated at n is $AC_{\Phi,\sigma}(n) := \sup\{A_\Phi(x) \mid \sigma(x) = n\}$ ('Φ's and/or 'σ's are often suppressed when understood).[4] So, just as A, B, C, etc. stand for types of *resource*, AC, BC, CC, etc. stand for types of *complexity*.

1.4 $\mathcal{O}$-notation

The use of asymptotic, especially $\mathcal{O}$-, notation is widespread in existing complexity theory; it provides a suitable language for discussing the large-scale behaviour of complexity functions in that it allows abstraction of the relevant information. To say that an algorithm takes $6n^2 - 2n + 10$ milliseconds to process an input value of size n is to make an implicit assumption about the *implementation* of the algorithm (that it uses a processor of a specific speed, for example); a better processor may execute the same algorithm in $3n^2 - n + 5$ milliseconds. Since it is typically the time complexity of the *algorithm* itself, rather than of any of its physical implementations (with specific-speed processors and so on), that we wish to measure, the relevant information is that the time complexity is *quadratic* in n; more exactly, the complexity theorist is often interested only in the fact that, *but for a finite number (or bounded set) of exceptional values of* n—which necessarily all occur where n is less than some finite threshold—, the time complexity behaves quadratically. This is precisely the sort of condition captured by $\mathcal{O}$-notation (in this example, we have that $TC(n) \in \mathcal{O}\left(n^2\right)$, where T is the resource of *time*), as is evident from its definition:

Definition 1. *Let* $\mathcal{O}(f(n))$ *be the set of all functions* $g(n)$ *such that there exist a threshold* n_0 *and constant* c *such that, for all* $n > n_0$, $|g(n)| \leq c\,|f(n)|$.

(In addition to the notation's ignoring arbitrary details such as processor speed, its appropriateness is further bolstered by speed-up theorems, etc.[5])

Below, we introduce a means—namely, dominance—of comparing in a fair and meaningful way the respective complexities of computational systems; notably, the systems may be instances of *different models of computation* (Turing machines, random access machines, analogue/quantum/chemical comput-

[4] This definition of $AC_{\Phi,\sigma}$ in terms of the *supremum* of a set of amounts of resource reflects the fact that, whilst there may be several different amounts $A_\Phi(x)$ of resource corresponding to various input values x of the same size n, we are interested in finding an amount $AC_{\Phi,\sigma}(n)$ of resource sufficient *for any* input value of size n. (We are tacitly assuming the perfectly reasonable condition that computation can proceed when more resource than is necessary is available: specific values of A_Φ and $AC_{\Phi,\sigma}$ can be thought of as *lower bounds*. Should there be a need to model the notion of 'too much of resource type A'—an *upper* bound—, then a new resource, '$-A$', bounded from below, can be introduced.)

[5] For example, if an algorithm Φ is such that $TC_\Phi(n) = f(n)$, then, for all real numbers $\epsilon > 0$, there exists an equivalent algorithm Ψ such that $TC_\Psi(n) = \epsilon f(n) + n + 2$. Hence, we can disregard all but the order k of a polynomial time complexity function: '$TC(n) \in \mathcal{O}\left(n^k\right)$' captures the relevant information (see Sect. 2.4 of [5]).

ers, etc.), and may be compared with respect to *different resources* (time, space, precision, ink, etc.). This notion of dominance relies on $\mathcal{O}$-notation.

2 Dominance

2.1 Motivation

$\mathcal{O}$-notation makes immediately possible like-with-like comparisons: given two computers Φ and Ψ[6] and a resource A, we may have either

1. that $AC_{\Psi}(n) \in \mathcal{O}(AC_{\Phi}(n))$ (in which case Φ can be said to consume no less of A than Ψ does), or
2. that $AC_{\Phi}(n) \in \mathcal{O}(AC_{\Psi}(n))$ (in which case Ψ consumes no less of A than Φ does).

We may, further, have both 1 and 2 (in which case Φ and Ψ consume A equally[7]), or, finally, neither 1 nor 2 (in which case the respective consumptions by Φ and Ψ of A are incomparable). This induces a pre-ordering of complexity functions based on 'A-efficiency'.

So, we know how to make like-with-like ('A-with-A') comparisons; however, they are not necessarily *relevant*. For example, we note in [3] two methods for finding the greatest common divisor of two given natural numbers:

- an algorithm (Euclid's Algorithm), which has time and space complexities logarithmic in the input values, and precision complexity[8] constant in same; and
- an analogue system, which has time and space complexities constant in the input values, and precision complexity cubic in same.

We could describe the methods respectively as logarithmic- and constant-time (and infer, by 'A-with-A' comparison, that the latter is more time-efficient); but it is intuitively more insightful to describe the latter as a *cubic-precision* (and less overall-resource-efficient), rather than *constant-time*, method, since the former description focuses on the more relevant (i.e., $\mathcal{O}$-dominant) resource. This notion of 'most relevant resource' is now formalized.

[6] In practice, the computers will typically perform the same computation/solve the same problem; else, little meaning can be attributed to a comparison of their respective complexities (one may, prima facie, assume that such comparison *is* meaningful, that it says something, for example, about which computation is harder to perform, but a computation's/problem's complexity is defined in terms of an *optimal*—not *arbitrary*—solution method's (i.e., computer's) complexity; comparison of *problems* is better achieved using the notion of *reduction*).

[7] This should be interpreted as equality 'modulo irrelevant details'—see Sect. 1.4.

[8] Precision complexity is discussed in [3]; for present purposes, we need note only that precision is a resource required during some (non-Turing) computations, and that precision complexity is the corresponding complexity function.

2.2 Definition

In light of Sect. 2.1, the intent of our notion of *dominant resource* is that the corresponding complexity function should $\mathcal{O}$-dominate all other resources' complexity functions; those resources corresponding to dominated complexity functions are negligible, in the sense that their asymptotic contribution to resource requirements is negligible. Accordingly, we tentatively make the following definition.

Definition 2 (provisional; see Definition 3). *A* dominant resource *for a computing system Φ is a type A of resource such that, for any resource type B, the B complexity BC_Φ is in $\mathcal{O}(AC_\Phi)$.*

There are two aspects of this definition that require modification. First, a resource's dominance is over *every* other resource: it is not enough to show, for example, that precision complexity $\mathcal{O}$-exceeds time and space complexities in order to show that precision is dominant according to Definition 2; rather, precision complexity must be shown to $\mathcal{O}$-exceed time, space, and *all other conceivable resources'* complexities, whereas attempting even to list such resources is futile. Instead, we redefine below dominance *relative to a set of resource types.*

Secondly, the definition does not for every computing system imply existence of a dominant resource (much as we should like one), since there exist pairs of functions f and g such that $f \notin \mathcal{O}(g)$ and $g \notin \mathcal{O}(f)$. We weaken accordingly the definition of dominance (essentially from '*AC* $\mathcal{O}$-exceeds all other complexity functions' to '*AC* $\mathcal{O}$-exceeds all other complexity functions with which it is $\mathcal{O}$-comparable').[9]

Definition 3 (to replace Definition 2). *Let Φ be a computing system, and let $\mathcal{R}$ be a finite, non-empty set of resource types for Φ. An* $\mathcal{R}$-dominant resource *for Φ is a type of resource $A \in \mathcal{R}$ such that, for any resource type $B \in \mathcal{R}$ such that $AC_\Phi \in \mathcal{O}(BC_\Phi)$, we have that $BC_\Phi \in \mathcal{O}(AC_\Phi)$.*

Given this notion of ($\mathcal{R}$-) dominance, it is natural to introduce the following complexity classes.

Definition 4. *Let Φ and $\mathcal{R}$ be as in Definition 3.*

- *For $A \in \mathcal{R}$ and a function f, let $\mathbf{C}_\mathcal{R}(f, A)$ be the class of problems for which there exists a computing system Φ with $\mathcal{R}$-dominant resource type A such that $AC_\Phi \in \mathcal{O}(f)$.*
- *Let $\mathbf{C}_\mathcal{R}(f) = \bigcup_{R \in \mathcal{R}} \mathbf{C}_\mathcal{R}(f, R)$.*

[9] Note, however, that it would be an immense surprise to the author were the definition without this modification not to suffice for actual, practical computing systems—f and g as described are necessarily fairly contrived and unnatural.

2.3 Discussion

Dominance formalizes a resource's relevance when considering a computation: resources that are dominant impose the asymptotically greatest cost, to the extent that non-dominant resources may be disregarded as irrelevant.[10] Further, we have defined complexity classes that categorize problems according to cost in terms of relevant (that is, dominant) resource.

As we note above, $\mathcal{O}$-notation allows comparison of computers' respective complexities with respect to like resources ('A-with-A' comparison). This notation *together with the notion of dominance* allows meaningful comparison of computers'[11] respective complexities with respect to *different* resources: as long as the resources are dominant for their respective computers, it is meaningful to compare (according to the $\mathcal{O}$-ordering) the corresponding complexities; though not necessarily 'A-with-A', such comparisons are nonetheless 'relevant-with-relevant', and, hence, like-with-like (albeit at a higher level of abstraction). For example, letting T, S and P stand for the resources of time, space and precision respectively, we have

- that both T and S, but not P, are $\{T, S, P\}$-dominant for the former of the greatest common divisor methods of Sect. 2.1, and
- that P, but neither T nor S, is for the latter.

Since the cubic PC_{latter} $\mathcal{O}$-dominates the constant TC_{former} and SC_{former}—a 'relevant-with-relevant' comparison—, we have that the former method is (at least when considering only these three resources) the more efficient.

We have, then, a framework in which can be made meaningful and consistent comparisons of computation-model-heterogeneous sets of computers; the framework's classes can accommodate instances of various models of computation, and provide structure according to cost in terms of various resources. This model heterogeneity offers an immediate advantage: a *problem's* complexity, which is the most commonly sought object in complexity theory, is bounded above by the complexity of the most efficient *solution method for the problem* (algorithm, physical system, or similar that solves the problem); the ability to consider model-heterogeneous—and, hence, larger—sets of methods allows lower minimal complexity of methods, and so tighter upper bounds on the complexity of problems.

Note that dominance as we have defined it is by no means specific to Turing machines: it is defined relative to the resource(s) considered, which may cater for any computational model(s).[12] Therefore, the notion is compatible with,

[10] Necessary care during this disregard takes the form of constraints upon what constitutes a resource. Though such constraints are outside the scope of the present paper—here we define dominance relative to sets of *given resources*, which we assume to be sensibly constrained—, Blum's axioms offer a possible starting point for the constraints' definition; we shall explore this elsewhere.

[11] These computers may, furthermore, be instances of different computational models.

[12] This is of particular interest given the great research and practical activity in non-standard computation—see [1], [4], etc.

and offers complexity classes defined in terms of, arbitrary theories of resource; this work may—in fact, we intend it to—form part of a modular framework of complexity (see [2]).

Acknowledgements. We thank Bob Coecke and Joël Ouaknine (at Oxford) for their support, supervision and suggestions; and participants of Unconventional Computing 2007 and the Second International Workshop on Natural Computing for their encouraging feedback and discussion. We acknowledge the generous support of the EPSRC; this work forms part of project EP/G003017/1.

4.vii.2008

References

1. Adamatzky, A. (editor): *International Journal of Unconventional Computing*. Old City Publishing (2005 onwards)
2. Blakey, E.: *A Model-Independent Theory of Computational Complexity; Price: From Patience to Precision (and Beyond)*. Available at `http://users.ox.ac.uk/~quee1871/transfer.pdf`.
3. Blakey, E.: *On the Computational Complexity of Physical Computing Systems*. Unconventional Computing proceedings (2007) pp. 95–115
4. Hoare, C. A. R.; Milner, R. (editors): *Grand Challenges in Computing*. The British Computer Society (2004)
5. Papadimitriou, C.: *Computational Complexity*. Addison-Wesley (1995)