The Role of Parameterized Computational Complexity Theory in Cognitive Modeling[*]

H. Todd Wareham
Department of Computer Science
University of Victoria
Victoria, BC Canada V8W 3P6
harold@csr.uvic.ca

Abstract

This paper shows how parameterized computational complexity theory is better than previously-used theories of computational complexity, e.g., NP-completeness, at both measuring the power of computational models of cognitive systems and isolating the sources of this power. This point is illustrated with new parameterized analyses of two current constraint-based models in linguistics, Declarative Phonology and Optimality Theory.

Introduction

Over the last fifteen years, a number of authors have applied computational complexity theory (CCT) to evaluate models of cognitive systems, e.g., vision, natural language, memory (Berwick & Weinberg 1984; Barton, Berwick, & Ristad 1987; Ristad 1993; Tsotsos 1990; 1993; Valiant 1994). In part because of various assumptions underlying CCT, critics have questioned the relevance of such analyses (Ramer 1995; Rounds 1991). Recent work (Downey *et al.* 1994) suggests that certain of these criticisms can be addressed by using the theory of parameterized computational complexity (Downey & Fellows 1995a; 1995b); moreover, within this parameterized framework, a more powerful form of CCT analysis which readily exposes the sources of computational power in cognitive models is possible. This paper will more fully develop this framework and present new parameterized analyses of two models in linguistics.

This paper is organized as follows. The first section reviews the basics of CCT and discusses its relevance as a measure of model power. The second section introduces the theory of parameterized computational complexity and discusses both how it addresses certain criticisms of previous CCT analyses and why it is better than classical CCT at assessing the sources of computational power in cognitive models. This discussion is illustrated by parameterized analyses of Optimality Theory and Declarative Phonology. The final section sketches somes promising directions for future research.

[*]To appear, AAAI-96 Workshop Working Notes: *Computational Cognitive Modeling: Source of the Power*

Computational Complexity Theory

Computational complexity theory establishes upper and lower bounds on how efficiently problems can be solved by algorithms, where "efficiency" is judged in terms of the computational resources, e.g., time or space, required by an algorithm to solve its associated problem. As noted by Rounds in his 1991 review, "The presuppositions of an already established theory, such as complexity theory, are perhaps the properties of the theory most easily ignored in making an application" (Rounds 1991, p. 10). With this in mind, the basics of CCT are reviewed in this section.

Perhaps the most important presuppositions alluded to by Rounds are implicit in the definitions of algorithm complexity and efficiency. The *complexity of an algorithm* is a function that summarizes, for each possible input size, the resource requirements of that algorithm over all inputs of that size. This summary can take many forms, e.g., best-case, average-case, worst-case. This paper is concerned with worst-case measures – that is, if $R(i)$ is the resource required by algorithm A to solve input i and I^n is the set of all inputs of size n, then the worst-case complexity of A for value n is $\max_{i \in I^n} R(i)$. This complexity is further "smoothed" by considering its behavior as n goes to infinity. This is typically stated in $\mathcal{O}$ ("big-Oh") notation, which gives the lowest function that is an asymptotic upper bound on the worst-case complexity of the algorithm, e.g., $3n^2 + 10n - 5 = \mathcal{O}(\backslash^{\in})$, $\log_3 n/2 = \mathcal{O}(\log_{\in} \backslash)$. An algorithm is *efficient* if its complexity satisfies some criterion of efficiency, e.g., the complexity function is a polynomial of the input size, and a problem is *tractable* if it has an efficient algorithm.

Computational complexity theory establishes not only what problems can be solved efficiently but also what problems *cannot* be solved efficiently. This is done by appropriately defining a class $\mathcal{F}$ of tractable problems, a class $\mathcal{C}$ such that it is either known or strongly conjectured that $\mathcal{F} \subset \mathcal{C}$, and a reducibility α

between pairs of problems that preserves tractability, i.e., if $X \alpha Y$ and $Y \in \mathcal{F}$ then $X \in \mathcal{F}$. As a reducibility establishes the computational difficulty of problems relative to each other, e.g., if $X \alpha Y$ then Y is at least as computationally difficult as X, it can be used to isolate the hardest problems in a class $\mathcal{C}$ via the notions of hardness and completeness.[1] If a given problem X is at least as hard as the hardest problem in $\mathcal{C}$, then X does not have an efficient algorithm modulo the strength of the assumption that $\mathcal{F} \subset \mathcal{C}$.

Ideally, a complexity-theoretic analysis of a problem is not just a one-sided quest for either algorithms or hardness results. Rather, it is an ongoing dialogue in which both types of results are used to fully characterize the problem by showing which restrictions make that problem tractable and which don't (Garey & Johnson 1979, Section 4.1).

There are many flavors of CCT, the most familiar of which is the theory of NP-completeness (see (Garey & Johnson 1979; Johnson 1990; Papadimitriou 1994) and references). In computer science, which is interested in the machine-independent asymptotic behavior of algorithms, the simplifications implit in the use of worst-case measures and $\mathcal{O}$-notation are acceptable. However, as is pointed out in the next section, it is precisely such assumptions that make CCT analysis questionable when dealing with the machine-specific bounded-power computations underlying cognition.

Computational Complexity as a Measure of Model Power: Pros and Cons

To justify a measure M of the power associated with a computational model, one must adequately address the following four questions:

1. Why is M a good measure of model power?
2. What are the sources of power in a model under M?
3. How is the power of a model measured under M?
4. How are the sources of this power isolated?

In this section, the adequacy of computational complexity as as measure of model power is assessed under the criteria implicit in these questions.

Let the power of a computational model be the computational power required to solve the problem addressed by that model. Within this framework, the power of a model is measured by the computational complexity of its associated problems, and the sources of this power are the various mechanisms within this model that generate this complexity. For example, the power of a model of object recognition by the human visual system would be the computational complexity of the problem of object recognition under that model, and the possible sources of this power would be the various mechanisms invoked by this model to recognize objects, e.g., layout of the visual field, superimposed structures of neurons (Tsotsos 1990). There are several good reasons for using computational complexity to assess the power of cognitive models:

1. Computational complexity is asymptotic, and hence implicitly states how a model's resource requirements will "scale up" as input size increases. Such considerations are not trivial, as many attempts at extending solutions originally proposed for small inputs to inputs of realistic size have shown (Tsotsos 1993).
2. Computational complexity is machine-independent, in two senses:
   (a) $\mathcal{O}$-notation abstracts away from resource-usage details due to implementation on a particular machine; and
   (b) Complexity classes defined under a variety of different machines (including circuit-based models analogous to actual neural hardware (Maass 1994; Parberry 1994; Siegelmann & Sontag 1995)) have been inter-related and are expressed in a common hierarchy (Johnson 1990).

   The latter point is particularly important because a complexity result relative to one machine for a particular resource can often be readily translated into results relative to other machines for different resources.

Note that the use of computational complexity does not preclude analysis of model power in terms of such aspects as task representations and the like – rather, it just places all such analyses within a common framework in which their results can be compared.

How, then, can CCT be used to assess model power in practice? Much current computational work in cognitive science already focuses on deriving algorithms that replicate certain behaviors of a given cognitive system. Such "proof-of-concept" algorithms show what can be done relative to a particular machine and input size, and are thus valuable complements to examinations of and experiments on how the cognitive system of interest is actually implemented in real organisms. However, in light of known bounds on brain structure and processing capacity (Tsotsos 1990; 1993), equally valuable insights can be derived via complexity-theoretic hardness results which show what *cannot* be done over a wide range of machines and input sizes. Indeed, as argued in (Ristad 1993; Tsotsos 1990), both algorithmic upper-bound and hardness lower-bound results are necessary to delineate what types of models of cognitive systems are tractable relative to known biological limitations, and hence may be implemented in real organisms.

The following synthesis of Ristad's "language complexity game" (Ristad 1993) and Tsotsos's levels analysis (Tsotsos 1990) shows how such a CCT analysis of a cognitive system might operate in practice:

[1]Recall that a problem X is $\mathcal{C}$-*hard* if for all problems $Y \in \mathcal{C}$, $Y \alpha X$; if X is also in $\mathcal{C}$, then X is $\mathcal{C}$-*complete*.

1. Define a computational model of that system at the lowest possible level of abstractness. This level depends on how much is currently known about the system and its neural implementation; for example, as the neural structures underlying language are not nearly as well known as those for vision, models of linguistic systems must (for now) be defined much more abstractly than those for visual systems.
2. Repeat until a tractable model is found:
   (a) Add a new system behavior / neural limitation to the current model.
   (b) Use CCT to derive a new model that is consistent with proposed behaviors / limitations and whose associated problems have the lowest computational complexity.

This preference for optimal solutions does not imply that evolution operates by optimizing the form or function of biological systems; rather, it is a methodological convenience like Occam's Razor which allows investigators to simultaneously restrict the space of hypothesized models and order them for further consideration.

The ultimate goal of such an analysis is to iteratively refine abstract models of both a cognitive system's behavior and neural architecture into a set of algorithms that replicates the observed system behavior and can also be implemented in natural biological hardware. However, there are several problems with using CCT as it is currently defined to achieve this goal:

1. Natural cognitive computations have properties that are incompatible with various assumptions underlying CCT (Berwick & Weinberg 1984; Ramer 1995; Rounds 1991; Tsotsos 1990). Though the sizes and structural complexities of cognitive inputs are in principle unbounded, they are in practice (and thus, in the sense that the brain deals with them computationally) relatively small and simple. Hence, the worst-case inputs which dominate CCT efficiency may not actually be processed by the brain. Moreover, when comparing algorithms running on a specific bounded-power machine such as the brain, implementation details *do* matter, and the machine-invariance induced by $\mathcal{O}$-notation is no longer useful or acceptable. Hence, schemes which search for models that are CCT-tractable need not converge on models that are biologically realistic.
2. Even if the complexity-theoretic conception of tractability is biologically realistic, current CCT analyses say only whether a problem is tractable or intractable. They cannot indicate which aspects of a problem are responsible for intractability, or show the numerical forms in which this intractable behavior manifests itself.

To summarize the situation in terms of the questions asked at the beginning of this section, computational complexity is an appealing definition of model power, and allows a natural definition of the sources of this power; however, CCT as defined to date cannot adequately measure this power or isolate its sources.

One solution to this dilemma is to abandon CCT analysis altogether. However, one may instead choose to attack the root of the objections raised above by creating new theories of computational complexity that mitigate the effects of particularly inconvenient assumptions. One such theory is presented in the following section.

Parameterized Computational Complexity Theory

One of the most important flavors of classical CCT, NP-completeness theory, was inspired by the need to show that certain problems cannot have polynomial-time algorithms (Garey & Johnson 1979); parameterized computational complexity theory was inspired by the following similar need. Most computational problems have input that consists of one or more items; for example, an object-recognition problem might have as input a grid of observed light/dark values and a set of patterns corresponding to 2-D projections of known objects. Call each such item in the input an *input parameter.* When it can be proved that a problem X cannot have a polynomial algorithm, e.g., X is NP-hard, that problem will exhibit one of two algorithmic behaviors relative to any particular input parameter k (called, in this context, a *selected parameter*):

1. An algorithm can exist for X whose running time is non-polynomial in k but polynomial in all other input parameters, e.g., $k^{k^2}n^2m$; or
2. All algorithms for X require either that k is non-polynomial in conjunction with at least one other input parameter, e.g., n^km^2, or that k is polynomial but other input parameters are non-polynomial, e.g., 2^nm^2k.

The former kind of algorithm is preferable if k has small values in typical instances of a problem, e.g., when $k = 10$ and $n = 1000$, $2^kn^3 = 10^{12} << 10^{30} = n^k$. However, as noted above, classical theories of computational complexity are insensitive to the distinction noted above – they can say only that a problem does not have polynomial-time algorithms, and are silent on whether this non-polynomial behavior can be isolated relative to particular input parameters.

Parameterized computational complexity theory (Downey & Fellows 1995a; 1995b) frames this issue via the following definitions:

Definition 1 *A* parameterized problem *is a set $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a fixed alphabet. For a parameterized problem L and $y \in \Sigma^*$, the* fixed-parameter problem $L_y = \{x | (x, y) \in L\}$.

Note that the second component of elements of L corresponds to the selected parameter defined above. Given a problem P, call the parameterized problem defined relative to selected parameter k of P the *k-parameterized version of P.*

Definition 2 *A parameterized problem L is* fixed-parameter tractable *if there exists a constant α and an algorithm A to determine if (x, y) is in L in time $f(|y|) \cdot |x|^{\alpha}$, where $f : N \mapsto N$ is an arbitrary function. Such an algorithm A is a* fixed-parameter (f.p.) *algorithm for L.*

Parameterized computational complexity theory encompasses both a set of techniques for deriving f.p. tractable algorithms and an appropriate set of classes for proving f.p. intractability. Within this theory, class $\mathcal{F}$ defined in the first section of this paper corresponds to the class FPT of fixed-parameter tractable problems, and class $\mathcal{C}$ is one of the members of the W-hierarchy, a set of classes $\{W[1], W[2], \ldots, W[P]\}$ defined by successively more powerful solution-checking circuits (see (Downey & Fellows 1995a; 1995b) for details). These classes are related as follows:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P]$$

It is conjectured that all inclusions in this hierarchy are proper. Hence, no $W[t]$-complete problem is f.p. tractable unless all problems in $W[t]$ are f.p. tractable. Over one hundred problems from areas as diverse as VLSI design, molecular biology, and robotics have been classified within the W-hierarchy (see the Parameterized Complexity Home Page at `http://www-csc.uvic.ca/home/mhallett/research.html`).

Essentially, a W-hardness result for a k-parameterized version of a problem P suggests that the non-polynomial resource requirements of P are not just a function of k. As a selected parameter may actually be composed of one or more input parameters, such results can show which groups of input parameters in a problem are responsible for non-polynomial behavior. This gives parameterized analyses the following two advantages over regular CCT analyses:

1. The effect of assuming unbounded-size inputs is reduced by focusing on how various input parameters contribute to the asymptotic behavior. This also suggests, to a limited degree, how this behavior will be affected by bounding these parameters to small values.
2. Such analyses give a qualitative measure of the contribution of an input parameter (and thus any algorithmic mechanisms associated with that parameter) to the problem's general complexity, e,g., the lower the parameterized complexity of the k-parameterized version of a problem, the more that problem's general complexity depends on k.

The first advantage mitigates the effects of one of the assumptions of CCT discussed in the previous section; the second allows CCT analyses to "dissect" the mechanisms underlying cognitive models and thus isolate the sources of power in these models. Two examples of such model "dissections" are given in the following section.

A Parameterized Analysis of Two Constraint-Based Models in Linguistics

Phonology is the area of linguistics which studies regularities in the mapping between deep (mental/lexical) and surface (spoken/phonetic) representations in natural language. Motivated in part by the intractability inherent in the ordered content-sensitive transformation rules of classical generative grammar, several recent proposals have replaced such rules with unordered computationally trivial constraints that must be satisfied by surface representations (see (Prince & Smolensky 1993) and references). Unfortunately, even though these constraints are simple, their interactions can be a source of unforeseen computational difficulties. In this section, parameterized analyses are used to gauge the power of various mechanisms within two such constraint-based theories, Optimality Theory and Declarative Phonology.

In both theories, the input is a set of constraints and a set of candidate surface representations, and the goal is to find the subset of the given candidates that best satisfies the given constraints. The theories differ in how candidates are evaluated relative to these constraints:

1. In Declarative Phonology (DP) (Scobbie 1992), a successful candidate must satisfy all constraints.
2. Optimality Theory (OT) (Prince & Smolensky 1993), in contrast to DP, assumes that there is a fixed total order on the constraints from least to most important. Constraints are evaluated one at a time from most to least important under this ordering; at any point in this evaluation, only those candidates that have the minimum number of violations for the current constraint are allowed to proceed to the next constraint for further evaluation. A successful candidate is one that survives all such constraint evaluations.

Note that at least one candidate will always be successful under OT but that no candidates may be successful under DP.

Following (Bird & Ellison 1994; Ellison 1994), the versions of DP and OT examined here formalize both the given set of candidate representations and each given constraint as deterministic finite-state automata (DFA) (see (Hopcroft & Ullman 1979) for definitions). Each candidate representation DFA will be restricted to accept (and hence generate) strings of a particular length. Furthermore, to investigate certain restrictions on the form of constraints, each constraint will be represented by a *contextual CDFA* (CDFA) which simultaneously judges acceptance or rejection of each substring of length k in a given candidate string, for some fixed constant k. Call this k the *context-length* of that CDFA. In the case of DP, a candidate string x satisfies (violates) a constraint C if each (any) k-length substring of x is x is (not) accepted by the CDFA corresponding to C; in the case of OT, the number of unaccepted substrings is the number of constraint violations. This yields the following problems:

DECLARATIVE PHONOLOGY DERIVATION (DP-DERIVE)
Instance: A candidate string DFA C and a set of constraint CDFA A.
Question: Is there a string x such that x is accepted by C and x does not violate A_i, $1 \leq i \leq |A|$?

OPTIMALITY THEORY DERIVATION (OT-DERIVE)
Instance: A candidate string DFA C, a set of constraint CDFA A, a total ordering O on the members of A, and a vector $y \in \mathcal{Z}^{|A|}$.
Question: Is there a string x such that x is accepted by C, x is a candidate selected by A under O, and x violates constraint $A_i \leq y_i$ times, $1 \leq i \leq |A|$.

Consider now the following input parameters:

- The number of constraints ($|A|$);
- The length of the candidate strings ($|x|$);
- The size of the candidate string alphabet ($|\Sigma|$); and
- The maximum context-length of any constraint CDFA (c).

Given a problem X with input parameter k, let the k-parameterized version of X be written as k-X, and the version where k is fixed at constant value c be k_c-X.

Theorem 1 DP-DERIVE *and* OT-DERIVE *are NP-complete.*

Theorem 2 $|A|$-$|\Sigma|_2$-DP-DERIVE *and* $|A|$-$|\Sigma|_2$-OT-DERIVE *are W[t]-hard for all* $t \geq 1$.

Theorem 3 $|x|$-c-DP-DERIVE *and* $|x|$-c-OT-DERIVE *are W[2]-hard.*

Theorem 4 $|A|$-$|x|$-c-DP-DERIVE *and* $|A|$-$|x|$-c-OT-DERIVE *are W[1]-hard.*

Theorem 5 $|x|$-$|\Sigma|$-DP-DERIVE *and* $|x|$-$|\Sigma|$-OT-DERIVE *are in FPT.*

Theorem 6 $|\Sigma|_2$-DP-DERIVE *and* $|\Sigma|_2$-OT-DERIVE *are W[P]-hard.*

These results are summarized in Table 1. As $|x|$ and $|\Sigma|$ together cause the lowest observed parameterized complexity, the candidate string set generated by C (or, more correctly, the size of this set) is the major source of computational power in the DP and OT models. These results have two interesting implications:

1. *Contra* (Bird & Ellison 1994; Ellison 1994; Tesar 1995), DP-DERIVE and OT-DERIVE are intractable when the candidate string set is restricted to being a regular language generated by some DFA.

Selected Parameter(s)	$\|\Sigma\|$ Unbounded Value	Selected Parameter	Fixed Constant
–	–	W[P]-hard	W[P]-hard
$\|A\|$	W[t]-hard	W[t]-hard	W[t]-hard
$\|x\|$	W[2]-hard	FPT	FPT
c	W[2]-hard	?	?
$\|A\|$,$\|x\|$	W[1]-hard	FPT	FPT
$\|A\|$,c	W[1]-hard	?	?
$\|x\|$,c	W[2]-hard	FPT	FPT
$\|A\|$,$\|x\|$,c	W[1]-hard	FPT	FPT

Table 1: The Parameterized Complexity of DP-DERIVE and OT-DERIVE.

2. *Contra* the intuition of Tesar (Tesar 1995), who gives a low-order polynomial algorithm for OT-DERIVE when $c = 3$, there is no polynomial-time algorithm for OT-DERIVE when c is not a constant. This makes unlikely the efficient accommodation within OT of proposed constraints that require unbounded-size contexts, e.g., generalized alignment.

Future research into these models should look at further restrictions on the candidate string set and the form of constraints. The latter should initially focus on such aspects of constraint CDFA as the maximum number of states or transitions; however, different formalisms for expressing constraints, e.g., first order logic, should also be examined.

Conclusion

This paper has discussed why computational complexity is a valid measure of model power, and sketched how the techniques of computational complexity theory can be used to guide the process of model refinement. The theory of parameterized computational complexity has also been presented as an alternative that mitigates some of the problems encountered when using currently-available flavors of CCT in the analysis of cognitive systems. This presentation has been illustrated by parameterized analyses of two constraint-based models from linguistics.

Though parameterized complexity is more appropriate for analyzing cognitive systems than previous complexity-theoretic techniques, it still stops short of what will ultimately be required. To handle the bounded inputs and computational abilities of natural neural hardware, CCT needs to both consider more realistic neural models and divest itself of many of its assumptions of machine and numerical invariance, cf. (Barton, Berwick, & Ristad 1987, pages 35–37). Two lines of research seem promising: (1) new theories of parameterized computational complexity based on parallel machines (Cesati & Ianni 1995), and, more provocatively, (2) new theories of computational complexity that eliminate $\mathcal{O}$-notation and deal explicitly with fixed amounts of computational resources (Cai *et al.* 1993; Downey & Fellows 1994).

The ongoing nature of this research highlights what is perhaps the most message in this paper – that computational complexity theory is not a finished piece of business, but rather an active area of mathematics that can and does evolve in response to the needs of researchers in both the computing and cognitive sciences.

Acknowledgements

I would like to thank Carl de Marcken, Patricia Evans, Michael Fellows, Michael Hallett, Alexis Manaster Ramer, Eric Ristad, John Tsotsos, and Leslie Valiant for various manuscripts, helpful conversations, and comments on earlier versions of the work reported here.

References

Barton, G. E.; Berwick, R. C.; and Ristad, E. S. 1987. *Computational Complexity and Natural Language.* Cambridge, MA: MIT Press.

Berwick, R. C., and Weinberg, A. S. 1984. *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition.* Cambridge, MA: MIT Press.

Bird, S., and Ellison, T. M. 1994. One-level phonology: Autosegmental representations and rules as finite automata. *Computational Linguistics* 20(1):55–90.

Cai, L.; Chen, J.; Downey, R. G.; and Fellows, M. R. 1993. Advice classes of parameterized tractability. Manuscript.

Cesati, M., and Ianni, M. D. 1995. Parallel parameterized complexity. Manuscript.

Downey, R. G., and Fellows, M. R. 1994. $\mathcal{O}$ NO! A curious document concerning the philosophy of parameterized complexity. Manuscript.

Downey, R. G., and Fellows, M. R. 1995a. Fixed-parameter tractability and completeness I. Basic results. *SIAM Journal on Computing* 24(4):873–921.

Downey, R. G., and Fellows, M. R. 1995b. Fixed-parameter tractability and completeness II. On completeness for $W[1]$. *Theoretical Computer Science* 141:109–131.

Downey, R. G.; Fellows, M. R.; Kapron, B. M.; Hallett, M. T.; and Wareham, H. T. 1994. Parameterized complexity of some problems in logic and linguistics (extended abstract). In Nerode, A., and Matiyasevich, Y. V., eds., *Logical Foundations of Computer Science*, Lecture Notes in Computer Science no. 813, 89–101. Berlin: Springer-Verlag.

Ellison, T. M. 1994. Phonological derivation in optimality theory. In *COLING'94*, 1007–1013.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W.H. Freeman and Company.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley Publishing Company.

Johnson, D. S. 1990. A catalog of complexity classes. In van Leeuwen, J., ed., *The Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, 69–161. Cambridge, MA: MIT Press.

Maass, W. 1994. Lower bounds on the computational power of networks of spiking neurons. Technical report TR94-019, The Electronic Colloquium on Computational Complexity.

Papadimitriou, C. H. 1994. *Computational Complexity.* Reading, MA: Addison-Wesley Publishing Company.

Parberry, I. 1994. *Circuit Complexity and Neural Networks.* Cambridge, MA: MIT Press.

Prince, A., and Smolensky, P. 1993. Optimality theory: Constraint interaction in generative grammar. Technical Report RuCCS TR-2, Rutgers University Center for Cognitive Science.

Ramer, A. M. 1995. Review of Ristad, *The Language Complexity Game. Computational Linguistics* 21(1):124–131.

Ristad, E. S. 1993. *The Language Complexity Game.* Cambridge, MA: MIT Press.

Rounds, W. 1991. The relevance of computational complexity theory to natural language processing. In Sells, P.; Shieber, S.; and Wasow, T., eds., *Foundational Issues in Natural Language Processing*, 9–29. Cambridge, MA: MIT Press.

Scobbie, J. M. 1992. Towards declarative phonology. In Bird, S., ed., *Declarative Perspectives in Phonology*, Edinburgh Working Papers in Cognitive Science, Volume No. 7, 1–27. University of Edinburgh.

Siegelmann, H. T., and Sontag, E. D. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences* 50:132–150.

Tesar, B. B. 1995. *Computational Optimality Theory.* Ph.D. Dissertation, Department of Computer Science, University of Colorado.

Tsotsos, J. K. 1990. Analyzing vision at the complexity level. *Behavioral and Brain Science* 13:423–469.

Tsotsos, J. K. 1993. The role of computational complexity in perceptual theory. In Masin, S. C., ed., *Foundations of Perceptual Theory*, 261–296. Amsterdam: North-Holland.

Valiant, L. G. 1994. *Circuits of the Mind.* Oxford University Press.