ARTICLES

COMPUTATIONAL COMPLEXITY AND THE SCOPE OF SOFTWARE PATENTS

Andrew Chin*

ABSTRACT: Recent developments in patent law, most notably the effective nullification of the Supreme Court's 1972 *Benson* decision excluding mathematical algorithms from patentable subject matter, have attempted to reflect an increasingly sophisticated approach to computer science and technology. Despite this, the patent system has continued to disregard computational complexity, an issue of central concern to computer scientists and of strategic importance to U.S. information technology policy. This Article proposes a development of patent scope doctrine that would introduce the issue of computational complexity into patent infringement analysis, thereby encouraging more efficient algorithm design, enhancing public benefits from complementary improvements in computer hardware, and strengthening the institutional competence of the patent system.

CITATION: Andrew Chin, Computational Complexity and the Scope of Software Patents, 39 Jurimetrics J. 17–28 (1998).

The past twenty-six years have not been kind to *Gottschalk v. Benson.*[1] Developments in computer science and the software industry have blurred the analytical boundary between "mental processes"[2] and computer hardware,

---

* Andrew Chin, B.S. (Texas), 1987; D. Phil. (Oxford), 1991; J.D. (Yale), 1998, is presently a law clerk to Judge Henry H. Kennedy, Jr. of the U.S. District Court for the District of Columbia, andrewchin@iname.com. The views expressed are those of the author only. He would like to thank Stephen Kahn and Mark Lemley for helpful suggestions.

1. 409 U.S. 63 (1972).

2. *Id.* at 67–68. According to Chisum, the "mental steps" doctrine originated in *Ex parte Meinhardt*, 1907 Comm'n Dec. 237 (cited in 1 DONALD S. CHISUM, CHISUM ON PATENTS § 1.03[6], at 1–78.1 (1990)).

rendering incoherent the Supreme Court's rationale for excluding mathematical algorithms from patentable subject matter under 35 U.S.C. § 101.[3] Thus, in the 1994 *Alappat* decision,[4] the Federal Circuit held that mathematical algorithms were patentable subject matter because the claims as a whole were directed to a machine,[5] and the method disclosed had a "useful, concrete, and tangible result."[6] This summer, in *State Street Bank*,[7] the court further held that the fact that an algorithm's result was numerical would not be an obstacle to patentability.[8] Consequently, careful applicants for software patents can now circumvent the rule of *Benson* by drafting means-plus-function claims directed to a programmed computer[9] or software fixed on digital storage media.[10]

Many commentators have characterized the trend toward patentability of mathematical algorithms as reflecting a growing understanding of how computers work[11] and a greater appreciation of the "useful" work computers do.[12] Despite this

---

3. The statutory categories of patentable subject matter are process, machine, manufacture, and composition of matter or useful improvement thereof. 35 U.S.C. § 101.

4. *In re* Alappat, 33 F.3d 1526 (Fed. Cir. 1994) (en banc).

5. *Id.* at 1543–45.

6. State Street Bank v. Signature Fin. Group, 149 F.3d 1368, 1373 (Fed. Cir. 1998).

7. *Id.*

8. *See id.* at 1375.

9. *See, e.g.*, Lawrence Kass, *Computer Software Patentability and the Role of Means–Plus Function Format in Computer Software Claims*, 15 PACE L. REV. 787, 866–68 (1995) (advising practitioners to draft software means-plus-function claims divided into interrelated means). Gregory Stobbs has suggested that the "common thread" of the patentable subject matter cases is human control; that is, a claim will be found unpatentable under 35 U.S.C. § 101 if and only if there is no human control over the existence of what is claimed. *See* GREGORY A. STOBBS, SOFTWARE PATENTS 328–29 (1995). This rule also guides practitioners toward drafting claims directed to programmed computers or software fixed on media rather than abstract mathematical algorithms.

10. *See In re* Beauregard, 53 F.3d 1583, 1583–84 (Fed. Cir. 1995) (noting Patent Office's finding that "computer programs embodied in a tangible medium, such as floppy diskettes, are patentable subject matter under 35 U.S.C. § 101"); United States Patent and Trademark Office, Examination Guidelines for Computer-Related Inventions, 61 Fed. Reg. 7478, 7482 (1996) ("[A] claimed computer-readable medium encoded with a computer program defines structural and functional interrelationships between the computer program and the medium which permit the computer program's functionality to be realized, and is thus statutory.").

11. *See, e.g.*, Maria T. Arriola, In re Alappat *and Beyond: A New Approach to the Patentability of Mathematical Algorithms and Computer Programs in the United States?*, 5 FED. CIR. B.J. 293, 309 (1995) ("The *In re Alappat* court's revivification of the new machine argument reflects technical reality."); James R. Goodman et al., *The* Alappat *Standard for Determining that Programmed Computers are Patentable Subject Matter*, J. PAT. & TM. OFF. SOC., Oct. 1994, at 771 (explaining the scientific theory in support of the *Alappat* statement that "programming creates a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed").

12. *See, e.g.*, Robert C. Scheinfeld & Lawrence T. Kass, *A Computerized Business*

new sophistication, however, the recent Federal Circuit decisions adopt a simplistic definition of "useful": computer software is deemed useful if it "constitute[s] a practical application of an abstract idea."[13] This definition has the merit of following the letter[14] (though not the spirit[15]) of Supreme Court precedent, but it fails to account for the rapid developments in the complementary technology of computer hardware. The resulting patent regime is misaligned with the policies at the heart of the Patent Act: it rewards mathematical algorithms that do not substantially advance the art of computing, and discourages innovation along lines that would most rapidly push forward the frontiers of science and technology.

It did not have to be this way. In his famous 1986 article advocating the patentability of algorithms, Donald S. Chisum predicted that extending the patent system to software would encourage "the formulation of distinctly new and superior techniques" for unlocking the potential of technological advances in computer hardware.[16] The public would benefit because "[b]etter algorithms enable computers to be used more efficiently and effectively."[17]

These aspirations for the patent system suggest that the definition of "useful" software in the patentability inquiry should have turned in part on the efficient exploitation of technological advances in computer hardware. It is surprising, then, that the extensive literature on software patents thus far has failed to acknowledge the relevance of computational complexity to the patentability analysis.

Now that *State Street Bank* has opened the § 101 floodgates to every piece of software that has "practical application," it is necessary to look to patent scope doctrine, rather than subject matter analysis, to align the software patent system with national priorities. Fortunately, it is within patent scope doctrine that a straightforward solution exists.

This Article describes a doctrinal development that would establish asymptotic computational complexity as an issue of central importance in determining the scope of software patents, without disturbing existing case law. Specifically, I propose that the reverse doctrine of equivalents should allow as a defense to

---

Method is Patentable Subject Matter, N.Y. L.J., Aug. 6, 1998, at 1 (concluding that *State Street Bank* simplifies and focuses the test for patentability of a mathematical algorithm on its "practical utility").

13. *See State Street Bank*, 149 F.3d at 1373 (citing *In re Alappat*, 33 F.3d at 1544 and Arrhythmia Research Tech. Inc. v. Corazonix Corp., 958 F.2d 1053 (Fed. Cir. 1992)).

14. *See* Diamond v. Diehr, 450 U.S. 175, 182 (1981) ("It is for the discovery or invention of some *practical* method or means of producing a beneficial result or effect, that a patent is granted, and not for the result or effect itself.") (emphasis added).

15. *Benson* has never been overruled. *See generally* Pamela Samuelson, *Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 EMORY L.J. 1025 (1990) (arguing that *Benson* should still be regarded as good law).

16. Donald S. Chisum, *The Patentability of Algorithms*, 47 U. PITT. L. REV. 959, 1014 (1986).

17. *Id.*

software patent infringement those improvements in computational complexity that are superlinear in the parameters of the problem solved by the underlying algorithm.[18] The Article offers four independent rationales for such an approach. First, the modern understanding of the nature of computation requires some nontrivial efficiency standard for patentability. Second, since the beginning of the computer age, asymptotic complexity has been the most significant organizing principle in the design and analysis of efficient algorithms, generating a vital body of prior art. Third, a reverse doctrine of equivalents based on asymptotic complexity would encourage efficient algorithm design, thereby aligning the software patent system with national strategic priorities relating to computing, information processing, and problem solving. Finally, an asymptotic complexity standard would provide coherence to patent scope doctrine in a technological field that demands the highest possible degree of consistency and rigor.

Before further discussing this proposed refinement of the *Alappat-State Street Bank* doctrine, it is worth noting that Chisum's policy arguments and the Federal Circuit's decisions remain controversial. Several scholars, most notably Pamela Samuelson, have raised serious objections to software patents.[19] In view of the growth and creativity exhibited by the software industry under the copyright regime of the 1970s and 1980s, Samuelson concludes that "patent protection is not necessary for the software industry to thrive."[20] Samuelson also criticizes the institutional incompetence of the patent system in evaluating computer program-related innovations.[21]

---

18. My proposal would require the courts to exercise the reverse equivalents doctrine much more actively than they have in the past. *See* Ethyl Molded Products Co. v. Betts Package, Inc., 9 U.S.P.Q.2d 1001, 1026 (E.D. Ky. 1988) ("The reverse doctrine of equivalents, although frequently argued by infringers, has never been applied by the Federal Circuit."); Phillips Petroleum Co. v. United States Steel Corp., 673 F. Supp. 1278, 1350 (D. Del. 1987) (noting that reverse doctrine of equivalents is rarely successfully asserted), *aff'd*, 865 F.2d 1247 (Fed. Cir. 1989). This Article presents compelling reasons for them to do so.

There are other ways to perform the reverse equivalents calculus. *See, e.g.*, Timothy J. Douros, *Lending the Federal Circuit a Hand: An Economic Interpretation of the Doctrine of Equivalents*, 10 HIGH TECH. L.J. 321, 332–45 (1995) (arguing that analysis should consider level of research investment and commercial viability). However, the ultimate purpose of any such analysis must be to "ensure[ ] that all the purposes of the doctrine of equivalents, in light of the goals of the patent system, are served." *Id.* at 350. This Article argues that these purposes are best served by a doctrine of software patent scope that takes into account the asymptotic complexity of the underlying algorithms. *See infra* Part III.

19. *See, e.g.*, Pamela Samuelson *et al.*, *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308 (1994); *see also* Rafael X. Zahralddin, *The Effect of Broad Patent Scope on the Competitiveness of United States Industry*, 17 DEL. J. CORP. L. 949 (1992).

20. Samuelson, *supra* note 15, at 1135–36.

21. *Id.* at 1138–40 (criticizing Patent Office and Court of Customs and Patent Appeals). Samuelson also argues that the risk of submarine patents and the cost of cross-licensing are prohibitively expensive in the software field. *Id.* at 1136–38. However, these arguments

The consideration of computational complexity in the determination of patent scope effectively limits the patentability of algorithms and goes some way toward answering these objections. Even in a thriving software industry, the patent system can serve to encourage the development not just of *more* software, but of *more efficient* software. Also, efficiency standards for patentability will improve the patent system's institutional competence by requiring the Patent Office to analyze advances in software technology in the same way as computer scientists do. Computational complexity theory thus has much to offer both sides of the software patent debate.

I. THE NECESSITY OF A MINIMUM EFFICIENCY STANDARD

The Supreme Court's statement in *Benson* that a mathematical algorithm is akin to a "phenomenon of nature" and therefore is not patentable[22] focused the early debate over the patentability of software on the epistemological nature of mathematics.[23] Advocates of software patents argued that mathematical algorithms were invented by the human mind,[24] while the courts maintained that mathematics consisted of discovered laws of nature.[25] In 1994, however, the *Alappat* court pre-empted the debate by construing *Benson* narrowly, limiting its holding to inventions that represent "*nothing more* than a 'law of nature,' 'natural phenomenon,' or 'abstract idea.'"[26] Thus, by drafting claims directed to machines rather than disembodied mathematical concepts, a patent applicant need not weigh in on the question of whether the underlying mathematics was discovered or invented.

This early focus on the epistemology of mathematics may have obscured a second fundamental objection to software patentability that is even more specific

---

implicate the much larger question of whether the patent term should vary across different industries, and will not be addressed here.

22. *Benson*, 409 U.S. at 67.

23. For a summary of the debate in the mathematical and philosophical literature, see, for example, ROBERT PATRICK MERGES, PATENT LAW AND POLICY 95–97 (1992).

24. *See, e.g.*, Chisum, *supra* note 16, at 980.

25. *See* Irah Donner, *Patenting Mathematical Algorithms that "Embrace" Mother Nature*, COMPUTER LAW., May 1992, at 1, 9 and n.128 (criticizing the *Iwahashi* court's finding that "mathematical algorithms . . . like laws of nature, are not patentable subject matter" as "preoccup[ation] with the mathematical language and with the notion that mathematics always represents laws of nature") (quoting *In re* Iwahashi, 888 F.2d 1370, 1374 (Fed. Cir. 1989)). Donner is also critical of the case law between *Benson* and *Iwahashi*. *See id.* at 4–9.

26. *In re* Alappat, 33 F.3d 1526, 1544 (Fed. Cir. 1994) (emphasis added); *accord* Donner, *supra* note 25, at 3–4 ("Even if we assume that the . . . law of nature rules were applied in *Benson*, these rules do not require all mathematical algorithms to be unpatentable. Rather, *Benson* only held that algorithms which represent laws of nature or truths are not statutory subject matter.").

to the field of algorithmic inventions: the fact that since 1931, there has been enabling prior art specifying a method for performing every possible computer algorithm. Specifically, for any finite set of computers bounded by finite time and space, it is possible to construct a universal algorithm that simulates every program that could run on any of said computers, also within finite (but much greater) time and space.[27] Admittedly, such a construction would be prohibitively time- and space-consuming for almost all practical purposes. Nevertheless, given any problem that is amenable to computer solution and unbounded computational resources, the universal algorithm will eventually solve it in finite time and space.

This abstract theoretical result has practical consequences for the patentability of software because claims directed to computer programs should never be interpreted so broadly as to read on the universal algorithm. To avoid such an interpretation, the scope of software patent claims must be limited by time resources (measured, for example, in instruction cycles), space resources (measured, for example, in bytes of RAM), or underlying models of computation (for example, computer architecture and basic instruction set).

Because the universal algorithm requires so much time and space, the efficiency hurdle it sets for patentable software is de minimis. Nevertheless, the formal necessity of an efficiency standard identifies computational complexity as an issue of analytical significance under the Patent Act.[28] The patent system must respond to this issue, either by requiring software patent claims to delimit their own efficiency standards, or by establishing a judicially created doctrine that considers efficiency in construing software patent claims. This Article advocates the latter approach—specifically, the establishment of an asymptotic efficiency standard for the reverse doctrine of equivalents—for reasons that I will now elaborate.

II. ASYMPTOTIC COMPUTATIONAL COMPLEXITY

Computer scientists state the efficiency of an algorithm by describing its asymptotic computational complexity.[29] For example, consider the problem of sorting numbers on a computer that can perform only two basic steps: comparing two numbers; and if the two numbers are out of order, interchanging them. The

---

27. The proof of Kurt Gödel's famous Incompleteness Theorem, *inter alia*, establishes a one-to-one correspondence between the integers and the set of all possible algorithmic computations (known in formal logic as "recursive processes"). *See Recursive Functions* (visited Sept. 16, 1998) <http:// www.mtnmath.com/book/ node56.html>. Thus, "one can with a single recursive process simulate all recursive processes." *Id.* This universal recursive process is fully specified by the details of Gödel's proof, which have been studied by several generations of computer scientists. *See* KURT GÖDEL, ON FORMALLY UNDECIDABLE PROPOSITIONS OF PRINCIPIA MATHEMATICA AND RELATED SYSTEMS (B. Metzer trans., 1962); ERNEST NAGEL & JAMES R. NEWMAN, GÖDEL'S PROOF (1958).

28. 35 U.S.C. § 102–03 (1994).

29. *See generally* THOMAS H. CORMEN ET AL., INTRODUCTION TO ALGORITHMS 23 (1990).

simplest way of sorting n numbers on such a computer, "bubble sorting," yields an algorithm with a running time of about n^2 steps.[30] In 1983, however, three Hungarian computer scientists announced that they had invented an algorithm capable of sorting n numbers in no more than $cn\log n$ steps, where c is a known constant.[31] They reported this result to the computer science community as "an $O(n\log n)$ sorting network,[32] because they considered the value of the constant c to have very little significance in the context of what they had achieved.

Computer algorithm design and analysis has been directed toward asymptotic measures of computational complexity since at least the early 1960s.[33] Today, more than 70,000 academic publications exist on the asymptotic computational complexity of computer algorithms.[34] Courses emphasizing asymptotic computational complexity of algorithms are a standard part of the training of virtually every professional computer programmer, via the undergraduate and graduate computer science curriculum. In short, the use of asymptotic computational complexity to describe not only the efficiency of computer algorithms, but their overall value to the computer science community, is an industry standard.[35]

Of course, the patent system is not obligated to follow the industry standard. Congress is free to design a patent scheme to promote the progress of the useful

---

30. *See* DONALD E. KNUTH, SORTING AND SEARCHING (1973).

31. *See* M. Ajtai et al., *An O(n log n) Sorting Network, in* PROCEEDINGS OF THE 15TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING 1 (1983); *see also* CORMEN, *supra* note 29, at 653 (finding the value of c to be in the thousands).

32. *See* Ajtai, *supra* note 31, at 1.

33. By 1965, computer scientists identified as tractable mathematical problems for which a polynomial-time algorithm was known, using asymptotic measures of computational complexity in their analysis. *See* A. Cobham, *The Intrinsic Computational Difficulty of Functions, in* INTERNATIONAL CONGRESS FOR LOGIC, METHODOLOGY, AND PHILOSOPHY OF SCIENCE 24 (Yehoshua Bar-Hillel ed., 1964); *see generally* J. Edmonds, *Paths, Trees, and Flowers*, 17 CAN. J. MATH. 449 (1965) (introducing concepts of "good" and "bad" algorithms defined in terms of asymptotic computational complexity). The use of asymptotic measures of computational complexity probably dates back to the earliest computers. This work naturally led to the cataloging of algorithms using a taxonomy based on asymptotic computational complexity (still a current practice). *See* J. Hartmanis & R. E. Stearns, *On the Computational Complexity of Algorithms*, 117 TRANS. AM. MATH. SOC'Y. 285 (1965).

34. For a partial bibliography listing more than 74,000 articles, see <http://theory.lcs.mit.edu/~dmjones/hbp/ipl/ipl>. Asymptotic complexity measures are also reported in the standard source books for computer programmers. *See, e.g.*, WILLIAM H. PRESS ET AL., NUMERICAL RECIPES IN C (1992).

35. Thus, asymptotic improvements in computational complexity are recognized, at least by the computer science community, as conceptual breakthroughs of the kind that have traditionally warranted broad patent scope under the pioneer invention doctrine. *See* Christina Y. Lai, *A Dysfunctional Formalism: How Modern Courts are Undermining the Doctrine of Equivalents*, 44 UCLA L. REV. 2031, 2040–41 (1997) (describing early development of the pioneer invention doctrine).

arts[36] by rewarding software inventions according to its own rational criteria. The nation's science and technology priorities, however, would be best served by a patent law that recognizes the critical importance of asymptotic computational efficiency in software engineering.

III. COMPUTATIONAL EFFICIENCY AS A NATIONAL PRIORITY

Technology policy should advance the state of the art.[37] In computing, the state of the art is measured by the range and size of problems that can be solved using available resources.[38] A computer program advances the art if it enables a computer system to solve a previously unsolvable problem. Patent scope doctrine aligns itself with this policy when it rewards software that advances the art in this way.[39]

---

36. *See* U.S. CONST. art. I, § 8.

37. This policy is implicit in § 102 and § 103 of the Patent Act: the patent system refuses to reward claimed inventions that fail to advance the art in a nonobvious way. *See* Patent Act, 35 U.S.C. § 102–03 (1994).

38. This definition of the state of the art in computer technology is supported by our classical understanding of computers, as well as the objectives of practitioners in the field. See, e.g., FUNK & WAGNALLS STANDARD ENCYCLOPEDIC DICTIONARY 278 (1968) (defining computer as "any of various machines equipped . . . for the high-speed performance of mathematical and logical operations, or for the processing of large masses of coded information"); MICHAEL R. GAREY & DAVID S. JOHNSON, COMPUTERS AND INTRACTABILITY: A GUIDE TO THE THEORY OF NP-COMPLETENESS 7–8 (1979) (assessing effect of improvements in computer technology on "the largest problem instance solvable in one hour" using algorithms with varying asymptotic complexities).

In the context of the technology policy underlying intellectual property law, the value of a piece of software as a factor of production lies in its power to solve a particular set of problem instances. This problem-solving imperative is perhaps most vividly illustrated by the U.S. government's strategic computing programs. Strategically significant computational problems, characterized by the size of the data to be processed and the complexity of the problem to be solved, include molecular modeling, human genome mapping, weather forecasting, pharmaceutical testing, fluid dynamics, signal-image processing, environmental quality modeling, and electronics. *See Department of Defense Scalable Software Initiative* (last modified Jan. 28, 1998) <http://www.hpcmo.hpc.mil/Htdocs/CHSSI/chssi-intro.html>.

39. *See* John R. Thomas, *The Question Concerning Patent Law and Pioneer Inventions*, 10 HIGH TECH. L.J. 35, 97 (1995) (arguing that "pioneer status" inquiry governing patent scope doctrine should turn on social impact of invention and goals of federal regulatory policies); *cf.* Riley M. Sinder et al., *Promoting Progress: The Supreme Court's Duty of Care*, 23 OHIO N.U. L. REV. 71, 79 (1996) (interpreting limitation of software patents to hardware-directed claims in *Diehr* and *Alappat* as reflecting concern that patentees would "repress the [abstract] problem-solving efforts" of later programmers). However, the subsequent *Beauregard* case and Patent and Trademark Office guidelines suggest that the hardware requirement may be satisfied without reference to a particular computer architecture, and thus may exacerbate the tension between software patents and abstract problem-solving. *See supra* note 10.

The patent term is at odds with this problem-solving imperative. Depending on the scope of a software patent, the patentee may have the right to exclude others from using the most efficient methods available for solving a problem. For example, if the optimal algorithm is patented, then there is a nonempty set of problem instances that will not be solved without the patentee's consent. This social cost must be balanced against the eventual social benefit that will accrue when the algorithm enters the public domain.

A complicating factor in performing this balancing is the ongoing improvement of computer hardware technology, which continually changes the set of tractable problem instances independently of advances in computer software. However, under the standard assumption that hardware advances, relative to time and money resources, can be modeled by exponential increases in processing speed and memory capacity,[40] the cost-benefit analysis leads to an asymptotic computational complexity standard for patent scope.

To see this, note that the assumption of exponential improvement implies that the number of basic computation steps that can be performed with a given set of resources will increase by some constant factor c during the course of a fixed patent term. Suppose that a patent has recently issued for a piece of software that solves a certain problem P. Suppose further that the underlying algorithm A_0 solves all instances of P of size n within $10n^3$ basic computational steps. Consider two new algorithms A_1 and A_2, which have running times of $100n^2$ and n^3, respectively.

Because algorithm A_1 has a faster asymptotic running time than A_0, software based on algorithm A_1 should be held not to infringe the patent by the reverse doctrine of equivalents. This is because no matter what the value of c is, there are problem instances that can be solved now with algorithm A_1 that could not be solved even at the end of the patent term with algorithm A_0.[41] To find infringement

---

40. The federal government's own strategic computing initiatives are oriented toward an exponential model of technological development in computer hardware. *See, e.g.*, *Department of Defense Scalable Software Initiative*, *supra* note 38 ("There are clear trends emerging today in the high performance computing market that the Department of Defense recognizes and is positioning itself to exploit. These trends include an exponential growth in the processing speed of microprocessor technology."); *Accelerated Strategic Computing Initiative Program Plan* (last modified Aug. 22, 1997) <http:// www.dote/osd.mil/ifte/ ASCI.htm> ("ASCI will stimulate the U.S. supercomputing industry to develop high performance supercomputers with speeds and memory capacities 1000s of times greater than current models and 10s to 100s of times greater than anticipated based on current trends in supercomputer development").

The analysis in the paper does not require that developments in computer technology strictly follow an exponential curve; it follows from the weaker assumption that processing speed and memory capacity will increase by some (unknown) constant factor during the patent term.

41. Given two algorithms, the algorithm with faster asymptotic running time will be faster overall when the problem instance is sufficiently large. In this example, algorithm A_1 will be faster than algorithm A_0 provided that $n > 10$.

would be to allow the patentee to forbid the public from practicing an art (solving these large problem instances) that the patentee will not enable at any time during the patent term.

The same cannot be said[42] for algorithm A_2. If it turns out that $c > 10$, then advances in computer hardware technology will bring algorithm A_0 up to speed at some point during the patent term. At that time, the patent disclosure will enable the solution of the same set of problem instances that can be solved now with algorithm A_2. If P-solving software based on A_2 is found to infringe the patent, the public will face a delay in practicing an art that the patent disclosure will only later enable. However, such a broad reading of enablement falls well within established doctrine.[43]

Ultimately, an asymptotic computational complexity standard embodies a forecast about the progress of software's complementary technology, that is, that improvements in computer hardware will enable the art promised by constant-factor improvements in software efficiency. As Robert Merges and Richard Nelson point out, the decision as to how this technological forecast influences patent scope is better made by the courts in interpreting congressional intent than by a patent examiner.[44] In discerning Congress's forecasts for computer technology, the best available evidence suggests that Congress would adopt the standard assumption of continuing exponential increases in processor speed and memory capacity.[45] Therefore, a reverse doctrine of equivalents based on an asymptotic computational

---

42. I am not arguing that algorithm A_2 should be found to infringe the patent; A_2 may be found non-infringing on independent grounds. The point is that the efficiency defense outlined in this paper should be available only to software that provides more than a constant factor improvement in efficiency.

43. Enablement must be established only as of the date of filing. *See, e.g.*, *In re* Hogan, 559 F.2d 595, 606 (C.C.P.A. 1977) ("To restrict [a patentee] to the . . . form disclosed . . . would be a poor way to stimulate invention, and particularly to encourage its early disclosure. To demand such restriction is merely to state a policy against broad protection for pioneer inventions, a policy both shortsighted and unsound from the standpoint of promoting progress in the useful arts, the constitutional purpose of the patent laws."); *see also* Phillips Petroleum Co. v. United States Steel Corp., 673 F. Supp. 1278, 1287, 1292 (D. Del. 1987), *aff'd*, 865 F.2d 1247 (Fed. Cir. 1989) (sustaining patent claims covering forms made possible by later technologies).

44. "The rule [that broad claims may be allowed] puts the burden of *disproving* enablement on the examiner. The rationale is that any other rule would leave claim scope too much in the hands of individual examiners and their technological forecasting abilities. Narrowing is left to the courts in particular infringement suits." Robert P. Merges & Richard R. Nelson, *On the Complex Economics of Patent Scope*, 90 COLUM. L. REV. 839, 849 (1990).

45. See *supra* note 40 and accompanying text; *see also National Defense Authorization Act for Fiscal Year 1997*, H.R. CONF. REP. NO. 104–724 (1996) (noting "exponential improvement in commercial hardware and software").

complexity standard would best satisfy the congressional purposes behind the patentability of software.

IV. TOWARD A COHERENT DOCTRINE FOR SOFTWARE PATENT SCOPE

My proposal locates the algorithmic efficiency inquiry in the infringement determination, thereby minimizing the burden on Patent Office procedure. The defendant in an infringement action should have the burden of analyzing the algorithms underlying both pieces of software and demonstrating an asymptotic improvement in efficiency. On the other hand, any uncertainty regarding the complexity of the algorithm underlying the patented software should be resolved against the patentee.[46] Applicants for software patents should therefore take care during prosecution to identify, unambiguously, the basic computational components of their programs.[47] By declaring that a component is basic, a patentee concedes that the design of the component is not novel, but obtains protection against a programmer who tries to avoid infringement by improving the efficiency of the component. Thus, the efficiency inquiry is focused on the claimed invention, and not on routine elements of software design that may be streamlined with little effort.

For example, consider a database algorithm that calls a standard sorting subroutine n times, where n is the size of the database. Suppose that the sorting subroutine requires the execution of $2n^2$ instructions. A patentee would be well advised to identify the sorting subroutine as a basic computational component. Thus, a subsequent programmer could not avoid infringement by substituting the three Hungarians' asymptotically faster sorting subroutine[48] for the one disclosed in the patent, because such a substitution would still call the sorting subroutine n

---

46. This allocation of liability minimizes uncertainty regarding patent scope and is consistent with the presumption of prosecution history estoppel with respect to claim limitations. *See* Warner-Jenkinson Co. v. Hilton Davis Chemical Co., 117 S. Ct. 1040, 1049-51 (1997) (ruling that the doctrine of equivalents allows the scope of a claim relating to a chemical process to be extended beyond the claimed lower pH limit of 6.0, provided that the patentee can overcome the presumption of prosecution history estoppel). The identification of basic computational components would serve the "definitional and public-notice function of the statutory claiming requirement." *Id.* at 1049. As we have seen in Part I, every software patent claim should be construed to imply a limitation on computational complexity so as not to read on the universal algorithm.

47. Patentee's identification of basic computational components serves as notice that the remaining disclosure embodies the principle, or spirit, of the invention for purposes of reverse equivalents analysis. *See* Graver Tank & Mfg. Co. v. Linde Air Products Co., 339 U.S. 605, 608–09 (1950) ("[W]here a device is so far changed in principle from a patented article that it performs the same or a similar function in a substantially different way, but nevertheless falls within the literal words of the claim, the doctrine of equivalents may be used to restrict the claim and defeat the patentee's action for infringement").

48. *See, e.g.*, Ajtai et al., *supra* note 31.

times. On the other hand, the Hungarians in applying for a patent on their sorting algorithm would identify the comparison and exchange operations as their basic steps, consistent with the spirit of their invention.[49]

Following this framework, the scope of a software patent would be informed by the expert testimony of computer scientists who had performed a rigorous analysis of the underlying algorithms. The result would be a coherent doctrine for software patent scope. A patentee would be entitled to the problem-solving art she or he had enabled, with an allowance for constant-factor improvements in computer hardware technology during the life of the patent. The vast academic literature on efficiency in computer programming[50] would become relevant to the patent system that has too long neglected the efforts of leading scientists to extend the reach of computer technology.[51] And, the software patent system would serve the purpose for which it was intended: promoting progress in the useful arts.

---

49. *See id.*

50. *See supra* note 34.

51. Academic literature is underutilized. *See, e.g.,* STOBBS, *supra* note 9, at 196–97 (describing *Dr. Dobb's Journal*, a popular magazine, as a leading source of articles "at the cutting edge of software technology," while failing to mention any academic literature on computer science or software engineering). The heavy reliance during prosecution on previously issued software patents is particularly troubling because this body of prior art does not provide an accurate or accessible picture of the state of software technology. *See* Thomas P. Burke, *Software Patent Protection: Debugging the Current System*, 69 NOTRE DAME L. REV. 1115, 1163 (1994) ("For the majority of software inventions, patent prosecution was never sought. Where software patents have been awarded, the high-art of disguising the software component has made these issued patents invisible to all but the most skilled searchers."); *cf.* Marci A. Hamilton & Ted Sabety, *Computer Science Concepts in Copyright Cases: The Path to a Coherent Law*, 10 HARV. J. L. & TECH. 239, 245 (1997) (criticizing "[t]he legal world's studious avoidance of computer science terminology" in copyright cases).