Complex Algorithm Research and Development
Harder Than Many Think

Version 1.0.1: October 21, 2008

Introduction

An algorithm is a procedure or formula for solving a problem. For the purposes of this article, a complex algorithm is defined as an algorithm that embodies advanced mathematical or logical methods and requires at least one thousand (1000) lines of the C/C++ programming language to implement. The term C/C++ is used intentionally to reflect the reality that object-oriented methods are of limited use in complex algorithms, although the algorithms are often packaged inside an "object" for easy integration into applications.

Complex algorithms are typically implemented as either computer software or in custom VLSI chips (e.g. Application Specific Integrated Circuits or ASICs). Research and development of complex algorithms is a specialized area and differs in a number of ways from most software and hardware development. Remarkably, many computer software business and engineering professionals often underestimate or do not understand the difficulties and scope of complex algorithm projects.

Complex algorithms are already in widespread use in commercial applications. Prominent examples include the video compression algorithms that enable BluRay, DVD Video, YouTube, and many other modern digital video systems. The US DVD Video market is around $25 billion per year (2007). Although limited, speech recognition such as now frequently encountered in telephone help and customer service systems is another example. Other examples include encryption, seismic modeling used in oil and gas exploration, sophisticated financial models, traffic models, and many others.

Complex algorithms may solve a range of major problems confronting the human race including major diseases such as cancer, the need for more and cheaper energy, and so forth. Molecular modeling may enable the design of drugs or systems of drugs that can selectively target and destroy cancer cells based on the identifying characteristics of cancer cells such as chromosomal anomalies, something currently impossible. Electromagnetic modeling software may enable the

successful design and fabrication of working commercial fusion power sources. These are potential trillion-dollar markets. The global annual energy market is over $1 Trillion.

Note: Since I am a software developer, I will focus primarily on complex algorithms developed and implemented as computer software. Many of my comments apply equally well to hardware implementation. Where I have some knowledge and experience, I will make some comments on specific hardware issues.

Complex Algorithms Are Difficult

Complex algorithms are usually quite difficult to develop and often take longer than experience with other software projects would indicate. Although there are exceptions, complex algorithm projects usually take between four (4) months and several years. True research projects in which new mathematical or logical methods are developed are extremely unpredictable and typically take years. Most major scientific discoveries and inventions have taken at least five (5) years.

Complex algorithms frequently involve a tight coupling between different parts of the algorithm. All parts must work together within tight tolerances for the entire algorithm to work. This is similar to mechanical systems such as automobile engines or mechanical clocks. Indeed, implementations of complex algorithms are often referred to as “engines”, partly for this reason. Complex algorithms are often very unforgiving. Even very small errors, getting a single bit wrong, results in the implementation failing. This occurs frequently with encryption where usually every bit must be correct and video compression where even small errors often result in unacceptable “artifacts” in the decoded video. In practical terms, this means that the amount of time spent per line of working code is often significantly larger for complex algorithm projects than other software projects such as web sites, user interfaces, database reporting systems, and so forth

Most commercial software projects involve at most mathematics taught in early high school (9^{th}, 10^{th} grade) in the US. Even advanced high school mathematics such as the solution to quadratic equations is uncommon outside of computer graphics. Complex algorithms in widespread use today typically involve mathematics that is taught in the first and second year of college at a good college or university in the US. A few complex algorithms involve more advanced

mathematics. For example, the Global Positioning System (GPS) uses General Relativity, advanced undergraduate or graduate level mathematics, to determine the location and time correctly. In the future, more advanced mathematics may be needed for pattern recognition and other advanced tasks. Most commercial software developers do not have much experience with mathematical software at the level found in complex algorithms. Research and development of complex algorithms often requires a range of mathematical and logical skills that are not common.

Return on Investment

The return on investment for a successful complex algorithm project can be very high. Complex algorithm research and development is typically done by small teams or individuals. Small teams are the most common. Even a multi-year project, for example five years, with a ten person team (a large team) has a total cost of about $7.5 million (using a total cost per full time employee of $150K/year). A home run can solve a billion dollar or larger problem, bringing in hundreds of millions or even billions of dollars.

- Return = $100 M / $7.5 M = 13.3 (small home run)
- Return = $1 B / $7.5 M = 133 (big home run)
- Return = $1 T / $7.5 M = 133,000 (off the charts)

The greatest opportunities and the greatest risks lie in areas that require development of new mathematical or logical methods; that is true research. New complex algorithms can be converted very rapidly to commercial software products, even in a matter of months, as happened with new video compression algorithms in 2003.

Research and Development

The commercial software industry focuses overwhelmingly on "technically feasible" projects. Many venture capital firms explicitly claim to only invest in proven, technically feasible projects. Similar thinking pervades the commercial software industry. Where complex algorithms are concerned, technically feasible means proven algorithms for which working prototypes exist somewhere. The working prototypes are usually computer programs, often slow, that successfully implement the algorithm. These are frequently prototypes in the C or C++ programming languages, although Java is becoming more common (see the discussion of software engineering below). Thus, most commercial projects in the complex algorithms arena

involve such tasks as porting algorithms to a different platform (for examples, Unix to Windows), optimizing the algorithms for a new platform, integrating the algorithms into an application program such as a media player, converting a prototype into a production system, and so forth. Most research scientists would call these activities "Development" and not "Research" or "Research and Development".

The commercial software industry follows a widely accepted rule to avoid projects that are not technically feasible, meaning true research projects. Nonetheless, the rhetoric of the commercial software industry, both aimed at unsophisticated investors and customers is the opposite. Terms like research, science, and research and development are used routinely to describe commercial software development activities. Many companies make statements that either explicitly claim or imply that the company has a large R&D group engaged in true research. Note that rhetoric aimed at sophisticated investors such as venture capitalists is often the opposite, which can be quite confusing.

Historically, the commercial software industry has relied heavily on government sponsored research programs such as the Defense Advanced Research Projects Agency (DARPA) and the National Aeronautics and Space Administration (NASA) for the true research in software. Many types of software and specific software products can be traced back to government sponsored research programs. Some well known examples include the Internet, originally a DARPA project, and the World Wide Web which grew out of research projects at CERN and NCSA. Many other examples exist. Essentially all speech recognition software is derived from research sponsored by DARPA, especially projects at Carnegie-Mellon University. Nonetheless, industry rhetoric often invokes the image of private inventors in garages, the Wright brothers, and similar images of "free enterprise" and individual initiative. Typically, the putative inventor such as Tim Berners-Lee or Marc Andreessen is emphasized and the relevant government research program ignored or downplayed. Often there is little or no progress in commercial software if the relevant government research program is unable to make progress. This is most evident in pattern recognition and artificial intelligence, where progress has been very slow or non-existent.

Many government research programs are afflicted by a single "right way" that is pursued to the exclusion of all others. If this right way is good, then there is steady progress in the associated commercial software field. DARPA in particular relies upon periodic contests pitting

different methods against one another. This has repeatedly resulted in a single approach that showed early promise taking over a field. A contest of this type during the 1970's resulted in the so-called Hidden Markov Model (HMM) based speech recognition approach replacing essentially all speech recognition research on a global scale. Essentially all major speech recognition research groups, many directly funded by DARPA, pursue some variant of the HMM algorithm. Yet the performance of the HMM algorithm continues to be quite limited after 30+ years.

Most major commercial opportunities in complex algorithms require a company to fund and undertake genuine research, a difficult task that few companies understand. Artificial intelligence, speech recognition, cures for major diseases such as cancer or working fusion energy sources require substantial research.

How Does Research Differ from Development?

Commercial software development is usually unpredictable. Software projects frequently involve unexpected problems and usually take substantially longer than planned. Nonetheless, technically feasible commercial software development projects are more predictable than true research projects. Often if one takes a conservative cost and schedule estimate and multiplies this by a factor of three to four, one gets the actual cost and schedule of the project. A common joke with a great deal of truth is: *to get the real schedule multiply the official schedule by PI (3.14) for running around in a circle*. Because the project is technically feasible it can certainly be completed. Massive cost and schedule overruns (such as factors of ten) can usually be explained by incompetence or severe political problems.

True research is extremely unpredictable. Many true research projects simply fail. The researchers are unable to find the solution. For example, to date (2008), essentially all attempts to decipher human speech have failed in close to a century of attempts. Substantial research efforts at Bell Labs, MIT, and other institutions have failed to determine why certain sound spectra correspond to the different sounds in English and other languages. Even in successful research, estimates are often way off. For example, the mathematician Johannes Kepler made a bet in 1600 that he could determine the orbit of Mars in eight (8) days. His discovery of the elliptical orbit of Mars and other planets, one of the most important and difficult discoveries in scientific history, took five frustrating years in which every attempt to solve the problem failed until he found the answer in just a few days

in 1605. This process in which long periods of little or no progress are punctuated by sudden unpredictable leaps forward is typical of true research, especially major scientific discoveries or technological inventions.

True research, especially major scientific discoveries or inventions, usually involves a very large amount of trial and error. Often, after many failures, there is a leap or leaps in which a new approach or concept is tried which unexpectedly solves the problem. Most major scientific discoveries or inventions took somewhere between five (5) and twenty (20) years. This is significantly longer than the time frame of typical commercial software industry and venture capital funded projects. In many cases, one is talking about five to twenty years of failure followed by a "breakthrough", as in Kepler's case.

In my experience, people involved in commercial software development are often unaware that they have little or no experience with true research. The misleading rhetoric of the computer software industry often leads people involved in commercial software development to think that they are engaged in the sort of true research conjured up by iconic names like Einstein or the Wright brothers. This undoubtedly leads to many bad decisions and frustrating experiences.

The Importance of Rapid Prototyping

History records many remarkable instances when an individual or small team succeeded in making a major scientific discovery or invention on a very small budget, sometimes beating far better funded competitors. Major scientific discoveries or inventions almost always involve a large amount of trial and error. Discoverers or inventors who managed to make a major discovery or invention on a shoe-string budget usually found a very fast, inexpensive way to perform the many trials and errors required to make a major discovery or invention.

For example, James Watt is remembered for inventing the separate condenser steam engine, a major conceptual leap that turned the steam engine from a niche device used in coal mining to a major driver of the industrial revolution. The Newcomen steam engines of Watt's time were huge expensive house-sized engines. Watt however built and experimented with tiny scale models built from inexpensive wood, copper, and other materials. This enabled him to perform hundreds of trials and errors that led to the breakthrough concept of

the separate condenser that radically improved the Newcomen steam engine.

Octave Chanute and the Wright brothers, his proteges, conducted research and development of gliders constructed of cheap wood and canvas. The gliders were flown at low altitude on soft sand beaches, first in Gary, Indiana near Chicago where Chanute lived, and later at Kitty Hawk in North Carolina. This meant that the inevitable damage from crashes was limited and easily repaired. The pilots did not die from the crashes as was common with other early would-be aviators. By delaying work on the expensive engines until last (they planned to buy a commercial off-the-shelf engine), they avoided the enormous cost involved in repairing or replacing an engine after each crash. This enabled Chanute and the Wright brothers to eventually succeed where better funded efforts such as Hiram Maxim and Samuel Langley failed.

Progress in aviation and rocketry today is quite slow, almost flat-lined since 1970, in part because the cost of a single trial, especially a new high performance engine, has become extremely high, easily in the millions if not billions of dollars per prototype engine and vehicle. In rocketry and other high performance engines, the prototype engine and vehicle are often destroyed during each trial. Internet entrepreneur Elon Musk of PayPal fame encountered this problem with his SpaceX startup as have many other Internet and software entrepreneurs attracted by the dream of space travel.

In algorithm research and development today, rapid prototyping tools such as Matlab and Mathematica (see below) speed up and reduce the cost of the many trials and errors required in true research. This is very important because the number of trials and errors is usually very large.

The Importance of Conceptual Analysis

Most major scientific discoveries and inventions usually involve a large amount of conceptual analysis expressed in words and pictures (often hundreds of thousands of words). It is common to find lengthy verbal discussions of the issues combined with rough sketches or drawings of concepts. For example, Octave Chanute wrote an entire book *Progress in Flying Machines* containing his lengthy verbal analysis of the problem of flight. This book outlines his successful research plan to develop working powered flight. It contains several rough drawings, as is common in major breakthroughs, and only a few brief calculations. The mathematician Johannes Kepler devoted much of his

book, now known as *New Astronomy*, to a lengthy conceptual analysis of the problem of planetary orbits which was critical to his resolution of the problem.

At some point, these verbal analyses are refined into precise technical drawings in the case of mechanical inventions and specific mathematical expressions in the case of mathematical discoveries like Kepler's. However, the verbal and visual analysis appears to be critical in many discoveries and inventions and usually comes first. It is likely that this sort of verbal and visual analysis will be essential to solve many problems such as artificial intelligence and pattern recognition.

Historically, this conceptual analysis was considered a part of philosophy. Much of the classical training in Greek philosophy and mathematics probably provided important training in this conceptual analysis. The discovery of new mathematical expressions of practical use strictly by the symbolic manipulation and the highly abstract thought favored by the famous mathematician David Hilbert and his school at the University of Gottingen in the early 20th century seems to be rare. This is specifically mentioned because Hilbert's extremely abstract approach to higher mathematics has come to dominate mathematics and theoretical physics in the 20th century.

Some Famous Flops

Complex algorithm research and development is a treacherous area. There have been numerous flops and fiascoes over the years. It is easy to misjudge the technical feasibility of projects. There is a long history of exaggerated claims for complex algorithms that emulate aspects of human intelligence such as speech recognition. There has been enormous success in data compression over the last few decades. Nonetheless, there is a long history of exaggerated claims for advanced in data compression. Video and other data compression involves complex algorithms that are difficult to evaluate. *Caveat emptor!*

The Pen computing fad of the early 1990's is an example of a famous flop. The most prominent of these firms was Jerry Kaplan's GO, described in his book *Startup*. GO and similar firms' business plans hinged on handwriting recognition, an unsolved problem in pattern recognition. Kaplan actually devotes only a few pages of his book to the handwriting recognition problem.

Another notorious example is the speech recognition firm Lernout and

Hauspie which collapsed in a giant financial scandal with court cases and allegations of massive fraud. Again, the success of Lernout and Hauspie's business depended on the solution of the speech recognition problem which remains largely unsolved even today.

Note that many apparently sophisticated investors invested many millions of dollars in both GO and Lernout and Hauspie, even though a modicum of research would have revealed the poor state of handwriting and speech recognition technology at the time.

Some Famous Successes

Video and audio compression is one of the most successful areas in complex algorithms. Technologies such as VideoCD, DVD, MP3, and BluRay all incorporate sophisticated audio and video compression algorithms.

A major breakthrough in video compression reached the market in 2003, embodied in H.264, Windows Media 10, Flash Video, and other video standards and products. Prior to 2003 the bitrate for usable, loosely VHS quality video was about one (1) megabit per second. In 2003, the new video technologies achieved a bit rate of around 275 Kilobits/second, often with close to DVD quality with proper tuning of the compression. This was a truly major advance, a rare technological leap forward. This enabled YouTube and other forms of Internet/web video over DSL connections.

The bottom line is that complex algorithm research and development can be done, but it is difficult.

Software Engineering

There are significant differences between software engineering for complex algorithm research and development and mainstream software development. As mentioned above, complex algorithms often involve a tight coupling between parts of the algorithm that makes development more difficult and tedious than most software development.

It is often easier to research and develop complex algorithms using tools such as Matlab, Mathematica, AXIOM, or Maxima (formerly known as MACSYMA). These are scripting languages similar to Python or PHP. They usually have implicit variable declaration and/or conversion. They are usually "weakly typed" languages and break

many textbook rules of “good” software engineering. They include comprehensive, well-integrated libraries of mathematical, numerical, and statistical functions. They usually have a data type known variously as a *list*, *vector*, or *matrix* that handles sequences of numerical or symbolic data. These tools are sometimes referred to as *computer algebra systems* (CAS), although this is really only one subset of their features.

Adding Two Vectors in Mathematica

```
A = {1.0, 2.0, 3.0};  (* A is a Mathematica list *)
B = {1.1, 0.0, 4.0};
C = A + B
Out[1]={2.1, 2.0, 7.0}
```

Adding Two Vectors in C/C++

```
#include <iostream.h>
double A[3] = {1.0, 2.0, 3.0};  // A is a C/C++ array
double B[3] = {1.1, 0.0, 4.0};
double C[3];
int index;
for(index = 0; index <3; index++)
{
C[index] = A[index] + B[index];
}
cout << “{“ << C[1] << “,” <<
C[2] << “,” << C[3] << “}” <<
endl;
```

Note that there is a *vector* class template in the C++ Standard Template Library (STL) with somewhat similar properties to the *lists* in Mathematica. The above comparison is for illustrative purposes. Even using the STL classes, it is usually much easier to research, develop, and test algorithms in these tools than using traditional compiled, strongly typed languages such as C/C++, Java, or <*insert your favorite programming language here*>. However, these tools are slow and require large amounts of memory. This is a significant drawback. Once an algorithm is developed, it is often necessary to convert the algorithm to a fast compiled language for performance reasons. This is easier if the target fast language has good libraries of mathematical, numerical, and statistical functions.

One can also research and develop algorithms directly in a fast programming language such as C/C++ or Java. This avoids

conversion costs, speed, and memory issues. However, it is often *much* easier to do research and development using a tool such as Matlab or Mathematica.

The leading algorithm research and development tools are:

- **MATLAB** Matlab is widely used in the commercial world, especially in digital signal processing.
- **MATHEMATICA** Mathematica is widely used in government sponsored research and development and academic research. It has a following in Wall Street finance and economics.
- **AXIOM** Axiom is free, open source, with a Berkeley style license. AXIOM was started in 1971 and has over 300 man years of work integrated into it.
- **MAXIMA** Maxima is free, open-source, with a GNU license.

Many fast programming languages have been used for complex algorithms. The most popular are probably:

- **ANSI C** ANSI C is almost universally available for all processors. It is simple, efficient, with small memory needs and high speed.
- **C++** C++ is object-oriented. It often has larger memory needs than C and can be slower.
- **Java** Java is compiled to byte-codes, but is approaching C/C++ in speed. It can be slower and less efficient. It can be easier to reverse engineer.

The dream algorithm R&D tool would be similar to Matlab or Mathematica but could be compiled to fast, efficient binaries similar to ANSI C and would be available for all platforms. An integrated GUI builder similar to Visual Basic and integrated network support would be helpful. The biggest single weakness of all kinds of scripting languages is that they are slow and cannot be compiled. For compute-intensive complex algorithms this can be a very significant problem. Of scripting languages, only Visual Basic 6 appears to have solved the problem of producing a compiler that can produce binary executables with similar performance to C/C++.

These algorithm research and development tools are not, of course, a substitute for thought, creativity, and the extensive conceptual analysis frequently required for major advances. Trial and error alone, without insight, rarely succeeds.

Conclusion

Complex algorithm research and development can be done successfully. Some great successes exist. Nonetheless, it is not easy and many things can go wrong. The project scope is significant. Project feasibility is difficult to assess. Genuine breakthroughs are unpredictable and take time. The return on investment for a home run can be five to one-thousand times the original investment.

About the Author

John F. McGowan, Ph.D. is a software developer, research scientist, and consultant. He works primarily in the area of complex algorithms that embody advanced mathematical and logical concepts, including speech recognition and video compression technologies. He has many years of experience developing software in Visual Basic, C++, and many other programming languages and environments. He has a Ph.D. in Physics from the University of Illinois at Urbana-Champaign and a B.S.in Physics from the California Institute of Technology (Caltech). He can be reached at jmcgowan11@earthlink.net.

References

Some Complex Algorithms

- http://www.chiariglione.org/mpeg/ (MPEG compression, one of the great success stories)

- http://www.videolan.org/developers/x264.html (x264 is a free, opensource h.264 video encoder)

- http://cmusphinx.sourceforge.net/ (The Carnegie Mellon Sphinx Project, an open-source speech recognition engine)

- http://www.itk.org/ (National Library of Medicine Insight Image Registration and Segmentation Toolkit)

Algorithm R&D Tools

- http://www.mathworks.com/ (Matlab)
- http://www.wolfram.com/ (Mathematica)
- http://www.axiom-developer.org/ (AXIOM)
- http://maxima.sourceforge.net/ (Maxima)

Books and Articles

StartUp: A Silicon Valley Adventure, by Jerry Kaplan, Houghton Mifflin Co, Boston, 1995

“How High-Tech Dream Shattered in Scandal at Lernout & Hauspie”, by Mark Maremont, Jesse Eisinger, and John Carreyrou, Wall Street Journal, December 7, 2000

New Astronomy (Nova Astronomia), by Johannes Kepler, Translated from the Latin original by William H. Donahue, Cambridge University Press, Cambridge, UK, 1992