Routine High-Return Human-Competitive Machine Learning

John R. Koza
Stanford University
koza@stanford.edu

Matthew J. Streeter
Genetic Programming Inc.
matt@genetic-programming.com

Martin A. Keane
Econometrics Inc.
mak@sportsmrkt.com

Abstract

Genetic programming is a systematic method for getting computers to automatically solve a problem. Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. The paper makes the points that (1) genetic programming now routinely delivers high-return human-competitive machine intelligence; (2) it is an automated invention machine; (3) it can automatically create a general solution to a problem in the form of a parameterized topology; and (4) it has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time.

Keywords: Genetic programming, human-competitive, inventions, parameterized topology

1 Introduction

One of the central challenges of computer science is to get a computer to solve a problem without explicitly programming it. Paraphrasing Arthur Samuel—founder of the field of machine learning—this challenge (1959) concerns

> How can computers be made to do what needs to be done, without being told exactly how to do it?

In his 1983 talk entitled "AI: Where It Has Been and Where It Is Going," Samuel said:

> "[T]he aim [is] … to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."

Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. Genetic programming uses the Darwinian principle of natural selection and analogs of recombination (crossover), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology to progressively breed an improved population over a series of many generations.

This paper makes four points:

(1) Genetic programming now routinely delivers high-return human-competitive machine intelligence (section 3).

(2) Genetic programming is an automated invention machine (section 4).

(3) Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology (section 5).

(4) Genetic programming has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time (section 6).

2 Background on Genetic Programming

Genetic programming (Koza 1992; Koza and Rice 1992; Koza 1994a; Koza 1994b; Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999; Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003; Koza, Keane, Streeter, Mydlowec, Yu, Lanza, and Fletcher 2003) breeds computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of compositions (typically random) of functions and terminals appropriate to the problem.

(2) Iteratively perform the following substeps (a generation) on the population of programs until the termination criterion has been satisfied:

(A) Execute each program in the population and assign it a fitness value using the problem's fitness measure.

(B) Create a new population of programs by applying the following operations to program(s) selected from the population with a probability based on fitness (with reselection allowed).

(i) *Reproduction:* Copy the selected program to the new population.

(ii) *Crossover:* Create a new offspring program for the new population by recombining randomly chosen parts of two selected programs.

(iii) *Mutation:* Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.

(iv) *Architecture-altering operations: C*reate one new offspring program for the new population by applying a selected architecture-altering operation to the selected program.

(3) Designate the individual program that is identified by result designation (e.g., the individual with the best fitness) as the run's result. This result may be a solution (or approximate solution) to the problem.

3 Genetic Programming Now Routinely Delivers High-Return Human-Competitive Machine Intelligence

3.1 Definition of "Human-Competitive"

In attempting to evaluate an automated problem-solving method, the question arises as to whether there is any real substance to the demonstrative problems that are published in connection with the method. Demonstrative problems in the fields of artificial intelligence and machine learning are often contrived toy problems that circulate exclusively inside academic groups that study a particular methodology. These problems typically have little relevance to any issues pursued by any scientist or engineer outside the fields of artificial intelligence and machine learning. To make the idea of human-competitiveness concrete, we say that a result is "human-competitive" if it satisfies one or more of the eight criteria in table 1.

The eight criteria in table 1 have the desirable attribute of being at arms-length from the fields of artificial intelligence, machine learning, and genetic programming. That is, a result cannot acquire the rating of "human-competitive" merely because it is considered interesting by researchers *inside* the specialized fields that are attempting to create machine intelligence. Instead, a result produced by an automated method must earn the rating of "human-competitive" *independent* of the fact that it was generated by an automated method.

Based on this definition, there are now 36 instances where genetic programming has produced a human-competitive result (table 2).

3.2 Definition of "High-Return"

What is delivered by the actual automated operation of an artificial method in comparison to the amount of knowledge, information, analysis, and intelligence that is pre-supplied by the human employing the method?

We define the *AI ratio* (the "artificial-to-intelligence" ratio) of a problem-solving method as the ratio of that which is delivered by the automated operation of the *artificial* method to the amount of *intelligence* that is supplied by the human applying the method to a particular problem.

The AI ratio is especially pertinent to methods for getting computers to automatically solve problems because it measures the value added by the artificial problem-solving method. Manifestly, the aim of the fields of artificial intelligence and machine learning is to generate human-competitive results with a high AI ratio.

Ascertaining the return of a problem-solving method requires measuring the amount of "A" that is delivered by the method in relation to the amount of "I" that is supplied by the human user.

Because each of the 36 results in table 2 is a human-competitive result, it is reasonable to say that genetic programming delivered a high amount of "A" for each of them.

The question thus arises as to how much "I" was supplied by the human user in order to produce these 36 results. Answering this question requires the discipline of carefully identifying the amount of analysis, intelligence, information, and knowledge that was supplied by the intelligent human user prior to launching a run of genetic programming.

To do this, we make a clear distinction between the problem-specific preparatory steps and the problem-independent executional steps of a run of genetic programming.

The *preparatory steps* are the problem-specific and domain-specific steps that are performed by the human user prior to launching a run of the problem-solving method. The preparatory steps establish the "I" component of the AI ratio (i.e., the denominator).

The *executional steps* are the problem-independent and domain-independent steps that are automatically executed during a run of the problem-solving method. The *executional steps* of genetic programming are defined in section 2 of this paper. The results produced by the executional steps provide the "A" component of the AI ratio (i.e., the numerator).

Table 1 Eight criteria for saying that an automatically created result is human-competitive

	Criterion
A	The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
B	The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
C	The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
D	The result is publishable in its own right as a new scientific result—independent of the fact that the result was mechanically created.
E	The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
F	The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
G	The result solves a problem of indisputable difficulty in its field.
H	The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

Table 2 Thirty-six human-competitive results produced by genetic programming

	Claimed instance	**Basis**
1	Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa "early promise" problem	B, F
2	Creation of a better-than-classical quantum algorithm for Grover's database search problem	B, F
3	Creation of a quantum algorithm for depth-two AND/OR query problem that is better than previously published results	D
4	Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously published result	D
5	Creation of a protocol for communicating information through a quantum gate previously thought not to permit it	D
6	Creation of a novel variant of quantum dense coding	D
7	Creation of a soccer-playing program that won its first two games in the Robo Cup 1997 competition	H
8	Creation of a soccer-player ranked in middle of field of 34 human-written programs in Robo Cup 1998 competition	H
9	Creation of four different algorithms for the transmembrane segment identification problem for proteins	B, E
10	Creation of a sorting network for seven items using only 16 steps	A, D
11	Rediscovery of the Campbell ladder topology for lowpass and highpass filters	A, F
12	Rediscovery of the Zobel "*M*-derived half section" and "constant *K*" filter sections	A, F
13	Rediscovery of the Cauer (elliptic) topology for filters	A, F
14	Automatic decomposition of the problem of synthesizing a crossover filter	A, F
15	Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of amplifier and other circuits	A, F
16	Synthesis of 60 and 96 decibel amplifiers	A, F
17	Synthesis of analog computational circuits for squaring, cubing, square root, cube root, and Gaussian functions	A, D, G
18	Synthesis of a real-time analog circuit for time-optimal control of a robot	G
19	Synthesis of an electronic thermometer	A, G
20	Synthesis of a voltage reference circuit	A, G
21	Creation of a cellular automata rule for the majority classification problem that is better than the Gacs-Kurdyumov-Levin (GKL) rule and all other known rules written by humans	D, E
22	Creation of motifs that detect the D–E–A–D box family of proteins and the manganese superoxide dismutase family	C
23	Synthesis of topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller	A, F
24	Synthesis of an analog circuit equivalent to Philbrick circuit	A, F
25	Synthesis of a `NAND` circuit	A, F
26	Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits	A. F, G
27	Synthesis of topology for a PID (proportional, integrative, and derivative) controller	A, F
28	Rediscovery of negative feedback	A, E, F, G
29	Synthesis of a low-voltage balun circuit	A
30	Synthesis of a mixed analog-digital variable capacitor circuit	A
31	Synthesis of a high-current load circuit	A
32	Synthesis of a voltage-current conversion circuit	A
33	Synthesis of a Cubic function generator	A
34	Synthesis of a tunable integrated active filter	A
35	Creation of PID tuning rules that outperform the Ziegler-Nichols and Åström-Hägglund tuning rules	A, B, D, E, F, G
36	Creation of non-PID controllers that outperform the Ziegler-Nichols or Åström-Hägglund tuning rules	A, B, D, E, F, G

The five major preparatory steps for genetic programming require the human user to specify

(1) the set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) for each branch of the to-be-evolved computer program,

(2) the set of primitive functions for each branch of the to-be-evolved computer program,

(3) the fitness measure (for explicitly or implicitly measuring the fitness of candidate individuals in the population),

(4) certain parameters for controlling the run, and

(5) a termination criterion and method for designating the result of the run.

It is fair to say that only a *de minimus* amount of "I" is contained in the primitive ingredients of the to-be-created computer program (the first and second preparatory steps), the problem's fitness measure (the third preparatory step containing the high-level statement of what needs to be done), and the run's control parameters and termination procedures (the fourth and fifth preparatory steps).

In any event, the amount of "I" required by genetic programming is certainly not greater than that required by any other method of artificial intelligence and machine learning of which we are aware. Indeed, we know of no other problem-solving method (automated or human) that does not start with primitive elements of some kind, does not incorporate some method for specifying what needs to be done to guide the method's operation, does not employ administrative parameters of some kind, and does not contain a termination criterion of some kind.

In view of the high amount of "A" in the numerator and the small amount of "I" in the denominator, we can see that the AI ratio is high for the results in table 2 produced by genetic programming..

3.3 Definition of "Routine"

Generality is a precondition to what we mean when we say that an automated problem-solving method is "routine." Once the generality of a method is established, "routineness" means that relatively little human effort is required to get the method to successfully handle new problems within a particular domain and to successfully handle new problems from a different domain. The ease of making the transition to new problems lies at the heart of what we mean by "routine."

Virtually all controllers are built from the same primitive ingredients (e.g., integrators, differentiators, gains, adders, subtractors, and signals representing the plant output and the reference signal). Once these primitive ingredients are identified, new problems of controller synthesis can be handled merely by changing the statement of what needs to be done. Thus, after solving one problem of automatically synthesizing both the topology and tuning of a controller (say, item 27 in table 2), the transition to each new problem of controller synthesis (say, item 23) involves providing genetic programming with a different specification of what needs to be done—that is, a different fitness measure.

In making the transition from problems of automatic synthesis of controllers to problems of automatic synthesis of circuits, the primitive ingredients change from integrators, differentiators, gains, and the like to transistors, resistors, capacitors, and the like. The fitness measure changes from one that minimizes a controller's integral of time-weighted absolute error, minimizes overshoot, and maximizes disturbance rejection to one that is based on the circuit's amplification, suppression or passage of a signal, elimination of distortion, and the like.

4 Genetic Programming Is an Automated Invention Machine

There are now 23 instances where genetic programming has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention. Specifically, there are 15 instances where genetic programming has created an entity that either infringes or duplicates the functionality of a previously patented 20th-century invention, six instances where genetic programming has done the same with respect to an invention patented after January 1, 2000, and two instances where genetic programming has created a patentable new invention.

5 Genetic Programming Can Automatically Create Parameterized Topologies

Genetic programming can automatically create, in a single run, a general (parameterized) solution to a problem in the form of a graphical structure whose nodes or edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables. We call such a solution a *parameterized topology*.

In a parameterized topology, the genetically evolved graphical structure represents a complex structure (e.g., electrical circuit, controller, network of chemical reactions, antenna, genetic network). In the automated process, genetic programming determines the graph's size (its number of nodes) as well as the graph's connectivity (specifying which nodes are connected to each other). Genetic programming also assigns, in the automated process, component types to the graph's nodes or edges. In the automated process, genetic programming also creates mathematical expressions that establish the parameter values of the components (e.g., the capacitance of a capacitor in a circuit). Some of these genetically created mathematical expressions contain free variables. The free variables confer generality on the genetically evolved solution by enabling a single genetically evolved graphical structure to represent a general (parameterized) solution to an entire category of problems. Genetic programming can do all the above in an automated way in a *single* run.

The capability of genetic programming to create parameterized topologies for design problems is illustrated by the automatic creation of a general-purpose non-PID controller (figure 1) whose blocks are parameterized by mathematical expressions containing the problem's four free variables, namely the plant's time constant, T_r, ultimate period, T_u, ultimate gain, K_u, and dead time, L. This genetically evolved controller (figure 1) outperforms PID controllers tuned using the widely used Ziegler-Nichols tuning rules (1942) and the recently developed Åström and Hägglund tuning rules (1995) on an industrially representative set of plants. The authors have applied for a patent on this new controller. For details, see Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003.

This controller's overall topology consists of three adders, three subtractors, four gain blocks parameterized by a constant, two gain blocks parameterized by non-constant mathematical expressions containing free variables, and two lead blocks parameterized by non-constant mathematical expressions containing free variables. For purposes of illustration, we mention that gain block 730 of figure 1 has a gain of

$$\left|\log\left|T_r - T_u + \log\left|\frac{\log\left(|L|^L\right)}{T_u+1}\right|\right|\right| \qquad [31]$$

and that gain block 760 of figure 1 has a gain of

$$\left|\log\left|T_r + 1\right|\right| \qquad [34].$$

If the genetically evolved program contains conditional developmental operators as well as free variables, a different graphical structure will, in general, be produced for different instantiations of the free variables. That is, the genetically evolved program operates as a genetic switch. Each program has inputs, namely the problem's free variables. Depending on the values of the free variables, different graphical structures will result from the execution of the program.

6 Progression of Qualitatively More Substantial Results Produced in Synchrony with Increasing Computer Power

Table 3 lists the five computer systems used to produce our group's reported work on genetic programming in the 15-year period between 1987 and 2002. Column 7 shows the number of human-competitive results (as itemized in table 2) generated by each computer system.

The first entry in the table is a serial computer. The four subsequent entries are parallel computer systems. The presence of four increasingly powerful parallel computer systems in the table reflects the fact that genetic programming has successfully taken advantage of the increased computational power available by means of parallel processing (thereby avoiding a pitfall that often constrains other proposed approaches to machine intelligence).

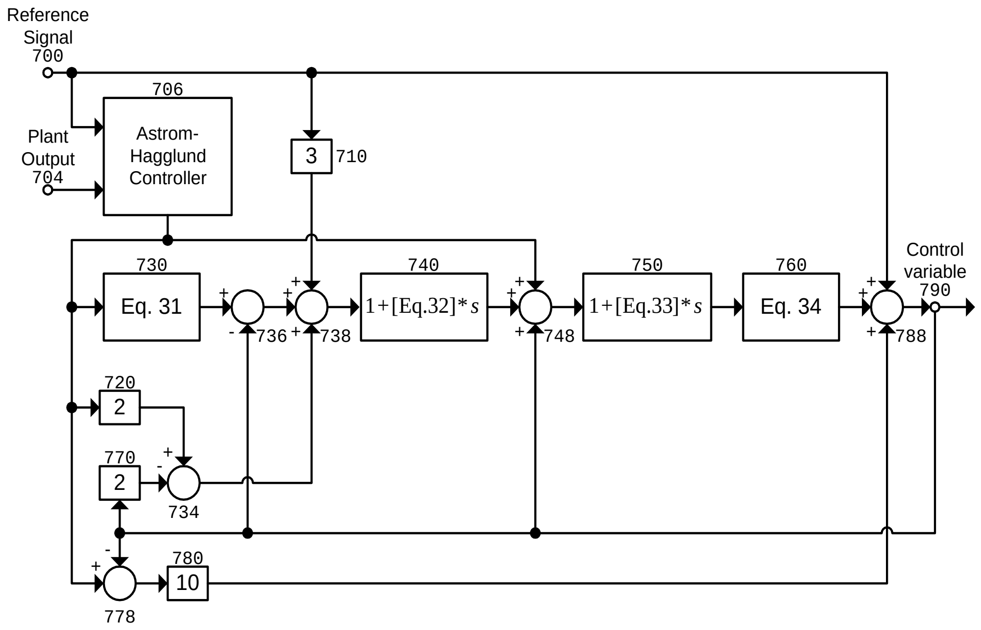

Figure 1 Parameterized topology for a general-purpose controller.

Table 3 shows the following:

- There is an order-of-magnitude increase speed-up (column 4) between each successive computer system in the table. Note that, according to Moore's law, exponential increases in computer power correspond approximately to constant periods of time.
- There is a 13,900-to-1 speed-up (column 5) between the fastest and most recent machine (the 1,000-node parallel computer system) and the slowest and earliest computer system in the table (the serial LISP machine).
- The slower early machines generated few or no human-competitive results, whereas the faster more recent machines have generated numerous human-competitive results.

Four successive order-of-magnitude increases in computer power are explicitly shown in table 3. An additional order-of-magnitude increase was achieved by the expedient of making extraordinarily long runs on the largest machine in the table (the 1,000-node Pentium® II parallel machine). The length of the run that produced the genetically evolved controller described in section 5 of this paper was 28.8 days—almost an order-of-magnitude increase (9.3 times) over the 3.4-day average for runs that our group has made in recent years. A patent application was filed for the controller produced by this four-week run.

If the final 9.3-to-1 increase in table 4 is counted as an additional speed-up, the overall speed-up between the first and last entries in the table is 130,660-to-1.

Table 4 is organized around the five just-explained order-of-magnitude increases in the expenditure of computing power. Column 4 of table 4 characterizes the qualitative nature of the results produced by genetic programming. The table shows the progression of qualitatively more substantial results produced by genetic programming in terms of five order-of-magnitude increases in the expenditure of computational resources.

The order-of-magnitude increases in computer power shown in table 4 correspond closely (albeit not perfectly) with the following progression of qualitatively more substantial results produced by genetic programming:

- toy problems,
- human-competitive results not related to patented inventions,
- 20th-century patented inventions,
- 21st-century patented inventions, and
- patentable new inventions.

This progression demonstrates that genetic programming is able to take advantage of the exponentially increasing computational power made available by iterations of Moore's law.

7 Conclusions

As far as we know, genetic programming is, at the present time, unique among methods of artificial intelligence and machine learning in terms of its duplication of numerous previously patented results, unique in its generation of patentable new results, unique in the breadth and depth of problems solved, unique in its demonstrated ability to produce parameterized topologies, and unique in its delivery of routine high-return, human-competitive machine intelligence.

8 The Future

Looking forward, we believe that genetic programming will be increasingly used to automatically generate ever-more complex human-competitive results.

With 50-gigaflops desktop parallel computers costing about $20,000 expected to be available in late 2003, genetic programming will be increasingly used as an invention machine to generate patentable new inventions.

References

Åström, Karl J. and Hägglund, Tore. 1995. *PID Controllers: Theory, Design, and Tuning.* Second Edition. Research Triangle Park, NC: Instrument Society of America.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido. 2003. *Genetic Programming IV. Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, Lanza, Guido, and Fletcher, David. 2003. *Genetic Programming IV Video: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Samuel, Arthur L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development.* 3(3): 210–229.

Samuel, Arthur L. 1983. AI: Where it has been and where it is going. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann. Pages 1152 – 1157.

Ziegler, J. G. and Nichols, N. B. 1942. Optimum settings for automatic controllers. *Transactions of ASME*. (64)759–768.

Table 3 Human-competitive results produced by genetic programming with five computer systems

System	Period	Petacycles (10^{15}cycles) per day for system	Speed-up over previous row	Speed-up over first system in this table	Used for work in book	Human-competitive results
Serial Texas Instruments LISP machine	1987–1994	0.00216	1 (base)	1 (base)	*Genetic Programming I* and *Genetic Programming II*	0
64-node Transtech transputer parallel machine	1994–1997	0.02	9	9	A few problems in *Genetic Programming III*	2
64-node Parsytec parallel machine	1995–2000	0.44	22	204	Most problems in *Genetic Programming III*	12
70-node Alpha parallel machine	1999–2001	3.2	7.3	1,481	A minority (8) of problems in *Genetic Programming IV*	2
1,000-node Pentium II parallel machine	2000–2002	30.0	9.4	13,900	A majority (28) of the problems in *Genetic Programming IV*	12

Table 4 Progression of qualitatively more substantial results produced by genetic programming in relation to five order-of-magnitude increases in computational power

System	Period	Speed-up over previous row	Qualitative nature of the results produced by genetic programming
Serial Texas Instruments LISP machine	1987–1994	1 (base)	• Toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning
64-node Transtech transputer parallel machine	1994–1997	9	•Two human-competitive results involving one-dimensional discrete data (not patent-related)
64-node Parsytec parallel machine	1995–2000	22	• One human-competitive result involving two-dimensional discrete data • Numerous human-competitive results involving continuous signals analyzed in the frequency domain • Numerous human-competitive results involving 20th-century patented inventions
70-node Alpha parallel machine	1999–2001	7.3	• One human-competitive result involving continuous signals analyzed in the time domain • Circuit synthesis extended from topology and sizing to include routing and placement (layout)
1,000-node Pentium II parallel machine	2000–2002	9.4	• Numerous human-competitive results involving continuous signals analyzed in the time domain • Numerous general solutions to problems in the form of parameterized topologies • Six human-competitive results duplicating the functionality of 21st-century patented inventions
Long (4-week) runs of 1,000-node Pentium II parallel machine	2002	9.3	• Generation of two patentable new inventions