Theorem (informal statement): There are no "extendible methods" in David Chalmers's sense unless $P = NP$.

Explication: In his paper, "The Singularity: A philosophical analysis," David Chalmers defines an "extendible method" as a method of improving intelligent programs that can "easily be improved, yielding more intelligent systems." I will try to make this notion more precise, at the cost of draining some of the excitement out of it. However, if the more precise definitions make the the theorem true then the excitement will be viewed as a mirage, I believe.

So let's take the classic joke that "AI is the search for polynomial-time solutions to NP-complete problems" as a way of thinking about what it means to be an intelligent program. AI programs tend to take polynomial time on some fraction of "interesting" problems that are all in a problem class C that is NP-complete; such that it is unlikely that a more useful characterization of that fraction of C can be found.[1] C might be the SAT problems that result when classical-planning problems are encoded using the "planning as satisfiability" paradigm (Kautz and Selman 1992).

Let's idealize a successful program as one that solves every solvable problem of some size (or smaller) in a *fixed* amount of time. A "solvable" problem is one with a Yes answer. (We're working with decision problems here.) That is, there is a time T and space S such that for every solvable problem of size N_0 or less, the program R solves the problem in time at most T using space at most S. For larger solvable problems it might take more time, but it always does return; for problems with answer No it might never halt at all; but in any case it never halts with the wrong answer. Of course, there is only a finite set of problems of size $\leq N_0$, so we're used to definitions like this turning out to be trivial, but we will be allowing the problem size to grow, and that should make most of the sense of triviality dissipate, because the space used will not be allowed to grow exponentially.

To proceed: I will take a program-improvement *method* for R, in Chalmers's sense, to be a technique or suite of techniques that (a) can be applied by a manageable group of competent programmers (some of whom may, in the future, themselves be intelligent programs); and (b) yields a new version of R that works on bigger problems in C. The new version may involve hardware improvements as well as algorithmic ones, so when I say "program" below I have in mind a combination of hardware and software. A method is parameterized by a level of effort, which we'll measure as time

[1] Of course, a realistic program actually contains multiple subprograms that can be characterized this way.

spent (at some standard level of person-hours/day). We'll pretend, just to simplify some sentences, that a method does not have stochastic results; if the universe were rewound and the method tried again with the same level of effort, it would always produce exactly the same successor program. In other words, the method is an algorithm. This may seem preposterous, but it seems like a reasonable formalization of a technique that can be applied indefinitely, as Chalmers requires.

To wax formal once more: we'll say that a method is $\langle \mu, \nu \rangle$-*extendible on program* R *and problem class* C (modulo the parameters T, S, and N_0) if it produces a series of programs R_1, R_2, etc., such that R_k solves C-problems of size kN_0 in time T, using space $O(k^\nu)$; and development of R_k from R_{k-1} takes time $O(k^\mu)$.[2]

Just to be clear, we expect the time to apply a method to be years, whereas the constant bound T must lie in the range of seconds, hours, or perhaps a few days at the most for R_k to be practically useful. T never changes from stage to stage of R's evolution; what changes is the size of problem that can be solved in that time.

To show that my formalization of the idea of extendible methods is not completely vacuous, I will give a concrete example. Moore's Law (Wikipedia 2012), whether justifiably or not, is often taken to be the claim that computers double in speed every $T_M = 18$ months, and that's how I will take it. If this really is a "law" that will hold true forever, then it supports the following method for improving programs: Our programming team subscribes to *Computerworld*. Then, to improve R_{i-1} to R_i, they wait some multiple of T_M until the issue arrives announcing that a fast enough computer is in the stores. They buy it and run their existing program on it.

Observation: Let E be a program for testing whether a nondeterministic computation will succeed by generating the exponentially many possible traces of the computation. To be precise, suppose the number of traces is $O(2^{bn})$, where n is the problem size and b is a constant, and the size required to store each trace is $O(n^a)$, where a is a constant. E uses some iterative-deepening scheme to explore the search space, so that for solvable problems it is required only to store the polynomially-size ($O(n^a)$) execution trace of the first trace it finds that ends with Yes. Choose T and S so that when the program is run on the fastest computer within the team's budget on any

[2] $R_0 =_{\text{def}} R$. It might be possible to shorten the time by investing more resources. But, as the maxim says, time is money. You can convert time to dollars and thence to other resources by investing some cash or writing grant proposals and waiting for the government to collect more taxes. So I'll just sweep all resources into one bin labeled "time."

problem of size $\leq N_0 = 1$, it finishes within time T and space S. Then if Moore's Law is a law, the *Computerworld* method is $\langle 0, a \rangle$-extendible applied to E if the team waits for time interval bT_M to port their program. (Here $\mu = 0$, $\nu = a$.)

Proof: E_k must solve problems of size k in time T. Program E_{k-1} could explore $O(2^{b(k-1)})$ traces in time T, but E_k gets to run on hardware $O(2^b)$ times faster, so it can explore $O(2^{bk})$ traces in time T. Development of program k takes constant time, so $\mu = 0$. The space grows proportionally to the space required for the longest trace, so $\nu = a$. QED

We can now state and prove the target theorem:

Theorem: If there is an extendible method for R (modulo all the parameters), then C, the class of problem solved by R, is in P.

Proof: This is pretty obvious, but let's go through the argument. We'll demonstrate a polynomial-time algorithm for solving problems in class C. Consider a problem of size n. Let R_{k_1} be the last version of R produced by the method, which solves all problems of size $k_1 N_0$. What we need is version $k_2 = \lceil \frac{n}{N_0} \rceil$. If $k_1 \geq k_2$, then the current version of R can solve the program in time T. Otherwise, the extendible method must be applied $k_2 - k_1$ times first, costing time bounded by the following sum:

$$\begin{aligned} t &\leq \sum_{i=k_1+1}^{k_2} c_i k^\mu \\ &\leq c_{\max} k_2 k_2^\mu \ (c_{\max} = \max_i c_i) \\ &= c_{\max} k_2^{\mu+1} \\ &= O(n^{\mu+1}) \end{aligned}$$

Once version k_2 of R is available, we can run it, taking time T. So the total time is $T + O(n^{\mu+1}) = O(n^{\mu+1})$. QED[3]

Corollary: If there is an extendible method, then $P = NP$.

Proof: Extendible methods as defined operate on programs to solve NP-complete problems. If one of these problems is in P, then they all are.

Corollary: Unless $P = NP$, Moore's Law is not actually a law. I hope this doesn't come as a shock.

Discussion:

[3] I would point out that the time required to develop a new version of the program can be amortized over the many times we will then use the wonderful new version; but we're in the middle of a *reductio* to a *quasi-absurdum*, so never mind.

The proofs above may read like a parody of such things, and to some extent that's the way they're intended. The very idea of extendible method cries out for parody. One might suppose that my requirement that a method be repeatable *forever* is part of this parody, but in fact it's of the essence in Chalmers's semi-formalized argument that AI++, an ultra-intelligent artificial entity, will come about. After all, if a method can be applied only k times, then one can dispense with talk of methods and just focus on R_k, the final product of all the extension. I suspect that part of the pleasure one gets from contemplating methods is that we can describe some of them, whereas we have no idea what R_k might look like.[4]

It would be nice if we could weaken the requirement that an AI system solve *all* problems below a certain size. Here's one way to do that: Let C_N be all the C-problems of size $\leq N$, Let θ be some constant $> 1/2$, the desired probability threshold for our program to work (0.9 would be nice). We adopt the following notation: A program R *solves* C*-problems of size* N *for practical purposes* iff such that given a randomly chosen stream of solvable C_N-problems, the probability R will solve the next one in time $\leq qN^\alpha$, using space $\leq rN^\alpha$ is $\geq \theta$. We'll abbreviate "solve for practical purposes" to *pp-solve.*

Now we can envision an extendible method increasing N (not necessarily linearly as we required above) while leaving α and θ the same. As the program R evolves to version R_k, exactly which fraction of problems it solves can change, so long as the fraction stays above the threshold. In fact, if there are problems of size N_k that go unsolved, the fraction of problems of sizes $< N_k$ that are solved must increase to keep the overall number $> \theta$. This all should sound plausibly like the way AI programs do evolve in practice.

Unfortunately, there is no nice theorem to conclude with. There is unlikely to be one involving P and NP. One could probably invent classes P_θ and NP_θ of problems that can be pp-solved in polynomial time without or with the help of nondeterminism. But, since no one's ever heard of these classes, the conjecture that $P_\theta \neq NP_\theta$ lacks intuitive oomph.

– Drew McDermott, 2012

References:

[4]The one apparent exception I'm aware of is the "Computerworld" method supported by Moore's Law. In this case we think we know exactly what R_k looks like, because it's just R_0, but the missing piece, exactly what-size problems the ultimate version will solve, is everything.

David Chalmers 2010 The Singularity: A philosophical analysis. *J. of Consciousness Studies* **17**(9–10), pp. 7–65

Henry A. Kautz and Bart Selman 1992 Planning as satisfiability. *Proc. European Conference on Artificial Intelligence (ECAI)*, pp. 359–363

Wikipedia 2012 Moore's Law. `http://http://en.wikipedia.org/wiki/Moore%27s_law`