Natural Design by Computer

Peter J. Bentley

There's a question that is guaranteed to frustrate me. It's: "what is your current research?" or sometimes (if the questioner knows me), "what are you up to these days?" or (if the questioner doesn't know me), "what do you do for a living?"

Like any academic, I have long-winded answers to all such questions. But however much I talk of topics such as evolutionary computation and evolving designs, or swarm intelligence and music composition, or artificial immune systems and intrusion detection, or computational development and *growing* solutions to problems, or evolvability and embodiment in an environment with rich physics, I rarely get my underlying message across. The reason for this is that I've been asked the wrong question. The question should not be *what* am I doing, but *why* am I doing it. In this keynote seminar I shall pretend that someone has asked me this second question, and I'll answer in (approximately) the following way:

Our technology is extremely clever, but it falls down in certain areas. One area is robustness – our designs are created with the expectation that they will be made correctly and will be used appropriately. If either of these expectations turn out to be false, then the item will fail. Another area is adaptability – most of our designs are never intended to cope with unexpected circumstances, so if something unforeseen occurs then they fail. Yet another area is self-design – or rather, the lack of it. We really have little concept of our technology designing itself or building itself outside science fiction. And another area is efficiency – our designs are as efficient as we know how (or wish to) make them. They cannot become more efficient by learning the intrinsic properties of their constituents – indeed, typically it is those intrinsic properties that ultimately cause the technology to fail. They can't even feed themselves – if their power source is failing, they cannot find energy from their environments, even if they're standing next to a power socket or a gas pump.

Natural designs may not be the result of conscious thought, but they are the result of a few billion years of experience. Nature has discovered that life must be robust – it is quite common for mistakes to be made during construction, and for organisms to be seriously damaged. Her solutions must cope, ensuring maximum survivability. Life also tends to be precocious, seeking out unexpected circumstances with alarming regularity. So organisms are highly adaptable, from the fish out of water to the fox in the city. They may not always survive, but they try very hard to keep functioning. Nature is an automatic design process. Life designs itself through evolution, and it ensures maximum efficiency of all of its constituents. It uses carefully tailored molecules as computers to store and execute biological programs, nanorobots (cells) to build, maintain and repair structures, and a million other developmental tricks to create staggeringly complex forms by gathering and processing materials from the environment.

The motivation behind much of my research is to understand these differences, and figure out which of the tricks behind nature's solutions we can use in our designs. There are obvious benefits to health, safety and lifestyle that such technology could bring: faulty transport systems that would self-repair or adapt, technology evolved to treat us like its children and protect us at all costs, helpful tools that do their jobs maintenance free, gathering their own energy as they work. With over three billion years of experience, we still have quite a bit of catching up to do with nature. I've spent a few years thinking about these kinds of things. I'll spend the rest of the seminar talking about a few of the ideas I have. They're to do with evolution, development and complexity.

Let's start with evolution. The field of evolutionary computation has shown that we can use evolution in our computers to solve problems. With the right kind of representations, we can optimise existing solutions or even permit the evolution of novelty from scratch. We also know a

little about the "character" of evolution. It's a very forgiving technique, able to find solutions to a wide range of nasty problems. Unlike some algorithms, an evolutionary algorithm can usually cope with vast search spaces, noisy objective functions, and even errors and bugs in its own implementation. It also has some nice properties of fault-tolerance (given a sufficiently flexible genetic representation and operators). Researchers such as Adrian Thompson (and some of my more recent work on gene regulatory networks) have shown that evolution has a natural tendency towards producing solutions that try to keep working even after damage. The reason is the mechanism of evolution itself. Because evolution finds solutions through random mutations and crossovers, once it has found a "perfect" solution, all it can do is damage that solution in subsequent generations. By forcing evolution to continue "beyond perfection," (open-ended evolution is helpful here) it is forced to protect that solution from itself, and make it tolerant of faults (which may also mean making it more efficient as well).

But the news is not all good. It turns out to be quite difficult to find the right kinds of genetic representations for evolutionary algorithms. If you get it wrong, then evolution will struggle to evolve anything at all. Even if you get it right, there is still a major problem of scalability. Evolve a simple genotype and you'll end up with a simple phenotype. To get a really complex phenotype, the corresponding genotype may need to be so complex that evolution can never find it. The answer to this problem seems to be *development*.

Development (or embryology, or ontogenesis) is essentially a non-linear mapping process from genotype to phenotype. It treats the genes as instructions on how the phenotype should be constructed (instead of a direct plan of the phenotype). When the instructions are carried out, the result is a complex phenotype from a simpler genotype. The danger with this kind of mapping process is: first, that it will often degrade evolvability, making it even harder to evolve anything, and second, that the very process of creating the developmental processes adds so much knowledge to the system that (critics would argue) you've practically solved the problem yourself. What is needed is a developmental processes that is both evolvable and that does not actively add any new knowledge into the system. Instead of knowledge, *capabilities* must be added. And one of the crucial capabilities is an ability for evolution to learn and incorporate its own knowledge into the developmental process. For example, if the solution requires the creation of a module and the reuse (and possibly modification) of the module, then the evolutionary developmental system needs to be able to "discover the idea" of modules and their reuse and modification for itself. If we add knowledge into the system, by modifying representations to include modules and their use, then we are partially solving the problem. If the ultimate intention is to have systems able to design, maintain, and repair themselves and so on, we must be looking at methods of enhancing capabilities but reducing problem-specific knowledge. If it can't even work out the idea of a module for itself, then it doesn't have a hope of discovering more complex ideas.

Finally (and this is a newer subject for me), ideas of *embodiment* seem to be becoming more important. All natural solutions are strongly embodied in their environments: they have many *perturbatory channels* through which they affect and are affected by their surroundings. This leads to *structural coupling* where, over the course of evolution, the structures of organisms are designed to fit together, the design of each being intimately linked. So one reason why life evolved to reach the complexity and efficiency that it has, is because every organism is embodied within a rich and complex environment. In our computers, most of our evolving solutions are in paltry environments, if you can even call a fitness function an environment. In the few examples where more complex environments are introduced (Thompson's evolvable hardware, Sims's virtual creatures), we do see a significant increase in complexity and novelty from evolution.

This article is just an outline of the more detailed presentation I shall give. But, to summarise, I (currently) believe that, in our quest for better technology, we need to consider evolution, development, and embodiment within a rich environment. That's why I'm doing my current research. You'll have to ask me exactly what I'm doing afterwards.