PLANET PACKING REVISITED

DANA RICHARDS
Fairfax, Virginia

Ross Eckler discusses a problem in his article "Planet Packing" in the May 2001 Word Ways: given a list of words, such as the names of the planets, how efficiently can they be packed into a single string of characters so that each word on the list can be read off left to right (but not necessarily contiguously)? He hypothesizes "there is no guarantee that any algorithm will end up with a minimum string." Since the design and analysis of algorithms has been my area of research for some 25 years, this caught my attention. Informally, an algorithm is a terminating procedure that could be coded as a computer program. (However, the procedure in the "Planet Packing" article does not contain enough tie-breaking rules to qualify as an algorithm.)

Perhaps the most surprising fact in this area is that we can prove strong results about some problems, in the sense that we can make definitive statements about the performance of any algorithm that solves a problem (including any algorithm that will be discovered in the future). In the most approximate terms, we say a problem is *tractable* if it can be solved in polynomial time, *intractable* if solving it requires more than polynomial time, or *incomputable* if no algorithm exists that solves the problem. Incomputable problems, further refined into *undecidable* and *not recursively enumerable,* are beyond the scope of this article, though there are puzzle-related topics in that direction. When we say polynomial time, we mean there exists an algorithm that always halts after n^k steps, for some constant k, where n is the size of the input.

The packing problem above does have an algorithm that solves it. We know that there exists a solution that is a string of length less than or equal to the sum of the lengths of input words. We can enumerate all such strings, lexicographically by increasing length. This clearly takes finite time so an algorithm exists. But is the problem tractable or intractable? The algorithm just given takes exponential time, but there might exist a polynomial time algorithm. The procedure sketched in "Planet Packing" certainly is more sophisticated, but its run time is unknown.

NP-Complete Problems

Very few (natural) problems are known to be intractable. However, the theory of NP-completeness is the next best thing. Starting around 1970, this theory has grown in importance, so that today every computer scientist has some understanding of it. It is based on some arcane topics in nondeterministic automata theory; to give all the relevant definitions takes several hours of lecture time. ("NP" stands for "nondeterministic polynomial time".) Surprisingly, though, it is of immense practical significance.

The importance of the theory is based on two observations: (1) all NP-complete problems are equally hard to solve, being either all tractable or all intractable, and (2) of the thousands of known NP-complete problems, not one is known to be tractable. The practical significance is that most of these thousands of problems are real problems, for which a polynomial time algorithm would have economic value. Hence the stakes are very high, but all researchers on all of these problems have failed. This is taken as strong circumstantial evidence that these NP-complete problems are all intractable.

The 1970s saw a flurry of activity that culminated n the book *Computers and Intractability* by Garey and Johnson (Freeman, 1979). While much work has appeared since then, this remains the definitive book. It contains a lengthy appendix listing hundreds of problems. In appendix A4.2 we find (slightly paraphrased):

> [SR8] Shortest Common Supersequence
> Instance: A finite set R of strings from a finite alphabet and a positive integer K.
> Question: Is there a string w, of at most K characters, such that each string x in R is a subsequence of w?

A *subsequence* of a string is a subset of the characters that are read left to right but are not necessarily contiguous (as opposed to a *substring* which must be contiguous). Note that this is phrased as a yes-no question, even though we really want to simply ask for the shortest common supersequence. The theory of NP-completeness requires that we (artificially) recast everything as yes-no questions. Note that if the yes-no version is intractable then actually finding the shortest common supersequence must be intractable too.

Clearly this is the same as our packing problem above. Hence, since it is NP-complete, we feel certain that it is intractable. Why would planet packing be found in a serious computer science book? It turns out to be an important problem with applications to data compression, DNA sequencing, and job scheduling.

Coping with NP-Completeness

OK, so we know our packing problem is NP-complete and hence probably intractable. What do we do? The first practical thing is to abandon all hope of solving the problem with a fast algorithm that always gets the optimal answer. If we want the exact answer we can use so-called dynamic programming techniques to get exponential time algorithms that are somewhat better in practice than the naïve exhaustive approach mentioned above. (For the supersequence problem, many approximation algorithms are based on repeated merging, always merging the two current strings that cause the least increase.) Otherwise, there are two avenues to pursue.

First, we can hope that our list of words has certain properties that will make our problem tractable. Unfortunately, it is known that even if every word on our list only has three letters the problem remains NP-complete. (However, if every word in the input has at most two letters it can be readily solved.) It is known that if we abandon the 26-letter alphabet in favor of a two-letter binary alphabet the problem remains NP-complete. (The unary alphabet is too trivial to discuss.) What if the input contains only two words? That can be solved in quadratic time and hence is a tractable problem. (In fact, in this case it is the dual of the better-known longest common subsequence problem.) In fact, an input of k words with a total of n characters can be solved in about n^k steps.

Second, we can use an approximate algorithm that will deliver a good but not optimal answer, such as sketched in "Planet Packing". Such an algorithm is rated according to its approximation factor α, where it is guaranteed to not produce an answer that is more than α times the optimal answer. So, for our problem, if we have an approximation algorithm with α=3 it will output a feasible string that is not longer than the true shortest common supersequence by a factor of 3. In my files I have a series of papers giving better and better approximation algorithms with α decreasing over time: 3, 2.88, 2.83, 2.79, 2.75, 2.66, 2.59 and 2.5. (The last appears in a paper by Z. Sweedyk, SIAM J. Computing 29 (1999), pp 954-986.) Clearly a lot of effort has gone into this problem. These algorithms all run in linear time. (The problem is actually known to be MAX SNP-hard which is a stronger result with technical implications about the inability to get much better approximations.)

Why Do We Care?

If you are still reading by this point, you are probably desperately hoping for some big payoff for recreational logology. I would not have written all of this if the planet packing problem was the only relevant problem. In this age when logologists are increasingly sloughing off questions with "let the computer do it," it is important to be reminded of the computational limitations of these problems. For example, in the Garey and Johnson book we see that word square construction is an NP-complete problem (problem [GP14]). Consider the problem of determining if there exists a single word that "crashes" with every word on a list; this is NP-complete (problem [SR12]). The longest common subsequence of a list of words is also NP-complete (problem [SR10]). The list can be made longer, especially if logologists tried to prove the problems peculiar to their discipline were hard.

Of course not everything is hard. For example, finding shortest word ladders, even when drawing from extremely large dictionaries, is a comparatively easy shortest-path computation that is not only tractable, but the known algorithms are very fast. (On the other hand, finding the longest word ladder of distinct words is equivalent to a longest path computation which is NP-complete!) When relying on computers it is important to be alert to these differences.