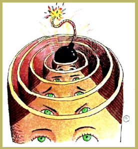
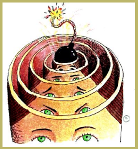


CS2205
Theory of Computation

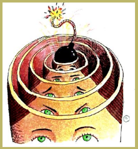
Derek Bridge

"A firm foundation in the theory of a subject is the hallmark of a professional and an excellent defence against technological obsolesence."



1.1. Module Details

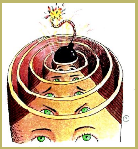
Lecturer:	Dr. Derek Bridge Room 304, Science Building d.bridge@cs.ucc.ie
Credit weighting:	10 credit *optional* module
Content:	Correctness, Complexity and Computability
Prerequisites:	The ability to program; the ability to reason carefully
Lectures:	$2 \times$ 1hr per week
Tutorials:	$2 \times$ 1hr per week (pen-and-paper exercises)
Private study:	2hrs per week (pen-and-paper exercises)
Handouts:	Lecture notes and exercise sheets will be handed out in lectures and tutorials
Course web site:	www.cs.ucc.ie/~dgb/courses/toc.html



Written exam:	3hr written exam; worth 160 marks
Continuous assessment:	2 class tests; worth 40 marks

How to fail:	Skip lectures & tutorials;
	avoid private study;
	cram at Easter;
	expect the exam to be a memory test

How to pass:	Attend lectures & tutorials; practice the skills immediately; expect a problem-solving exam



1.2. The Three Cs

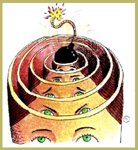
- This module begins with some background material and then proceeds to look at the following three topics.

  Correctness: How can we be sure that an algorithm or program solves a problem? How can we be sure that it terminates and produces the correct output for all of its legal inputs?

  Complexity: How do we measure and compare the complexity of algorithms? By complexity we mean the time and memory space the algorithms consume. What is the inherent complexity of a problem? In other words, what are the minimum resources required to solve that problem. In yet other words, what resources are required by the best possible algorithm? There are problems for which the best possible algorithms require so much time (or space) that those algorithms cannot be executed by any computers of reasonable size in any reasonable amount of time and never will be.

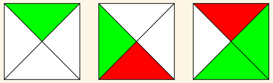
  Computability: Are there precisely-defined problems for which no algorithm is possible? Given a problem, how can one tell whether or not an algorithm is possible?

- We will ask and, where possible, answer these questions about problems, algorithms and programs relative to a particular model of computation. In other words, we will make some assumptions about the capabilities of the computer. We will find in the latter part of the course that these assumptions are not so important. The answers come out the same for an incredibly wide range of different assumptions.



1.3. Tiling Problems

- We will illustrate some of these ideas right now, with a simple example of a set of problems that we could try to solve by computer. The problems we will look at are called *Tiling Problems*.
- One input is a finite set of 1cm × 1cm square tiles, with coloured edges:

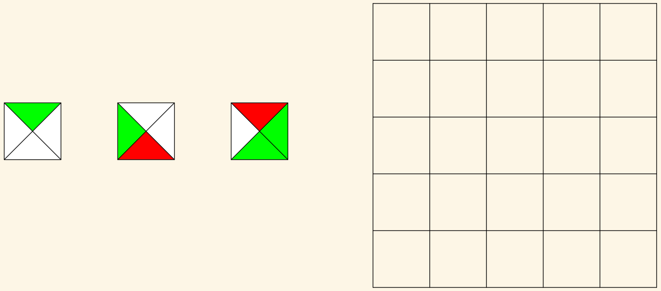


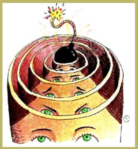
You may assume you have an unlimited supply of each type of tile.

- Another input is a rectangular grid, divided into 1cm × 1cm cells.
- You are asked:

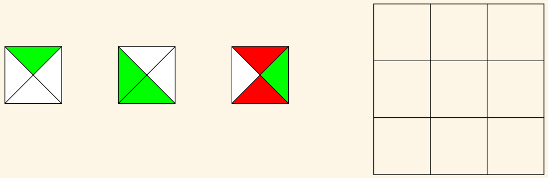
> Without rotating the tiles, can you tile the grid so that every cell in the grid contains exactly one of the given kinds of tile and the colours on all adjacent tile edges are the same?

- Suppose the inputs are these tile types and this grid:





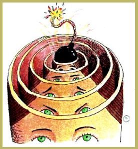
- Then, the answer is YES.
- (In fact, with this set of tile types, we can tile any size rectangular grid.)
- But suppose the inputs are these tile types and this grid:



- Then, the answer is NO.
- When we design the algorithm, we don't know what tile types there are. And we don't know the dimensions of the grid. These are inputs. Our algorithm has to be general enough to handle all legal inputs.
- All that is required is a YES or NO answer. We are not asking the algorithm to come up with an actual configuration of tiles.

1.3.1. Dimensions Known When Writing the Algorithm

- Suppose, contrary to what was said previously, that, at the time we are designing the algorithm, we *do* know the dimensions of the grid.
- E.g. maybe we are told that all grids will be 5×5.
- Then, it turns out that we *can* design an algorithm whose input is a finite set of tile types and whose output is YES exactly when the grid is tilable and NO otherwise.



Module Details
The Three Cs
Tiling Problems
A Classification of . . .

Module Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 7 of 12
Back
Full Screen
Close
Quit

- Furthermore, this algorithm will be reasonably efficient.
- (In fact, it is sufficient to know only one of the dimensions of the grid when we are designing the algorithm. E.g. maybe we know that the grid will be 5 wide, but we don't know how high it will be. In this case, we can still write a reasonably efficient algorithm.)

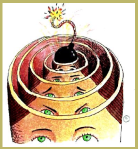
1.3.2. Dimensions Not Known When Writing the Algorithm

- Suppose we do *not* know the dimensions when we are designing the algorithm.
- Then, it is still the case that we *can* design an algorithm whose input is a finite set of tile types and the dimensions of the grid, and whose output is YES exactly when the grid is tilable and NO otherwise.
- But the best algorithm that has been found so far is unreasonably inefficient when given even quite small grids and small sets of tile types.

- Consider an algorithm that assigns a tile to each cell in the grid in turn; when the grid is full, it checks the legality.

		Num. of tile types	
		2	3
Num. of cells	4	16	81
	9	512	19683
	16	65536	43046721
	25	33554432	a 22-digit number
	36	an 11-digit number	a 28-digit number
	49	a 15-digit number	a 34-digit number

Tile Assignments



(The number of microseconds since the Big Bang has 24 digits.)

- Question: is there a way of avoiding some of these tile assignments?

1.3.3. Dimensions Known to be the Integer Grid

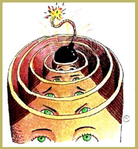
- Suppose, at the time we are designing the algorithm, we know that we must tile the *integer grid*. In other words, the input is still the set of tile types, and we want the algorithm to decide whether, using these tiles, we could tile the infinite grid.
- No algorithm can be written to solve this problem. Even with unlimited time and memory, and irrespective of how clever you are, provably no algorithm can be written to solve this problem. There may be algorithms that can solve special cases, but there is no algorithm that will work for all legal inputs.

1.3.4. Dimensions Known to be the Integer Grid and a Question About Infinite Recurrence

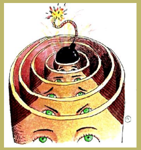
- If the integer grid can be tiled by a particular set of tiles, then it follows that at least one of the tile types must be used infinitely often. Now consider a slight variant of the problem that we have been looking at, in which we additionally ask about the recurrence of a designated tile.
- The problem is augmented:

  > We want a YES if there is a tiling of the integer grid *which contains an infinite recurrence of the first tile type in the set of tile types.*

  To clarify: we know that *at least one* of the tile types would have to recur infinitely often, so we want to know whether some particular tile type (the first tile type, say) is among those that do actually recur infinitely often.

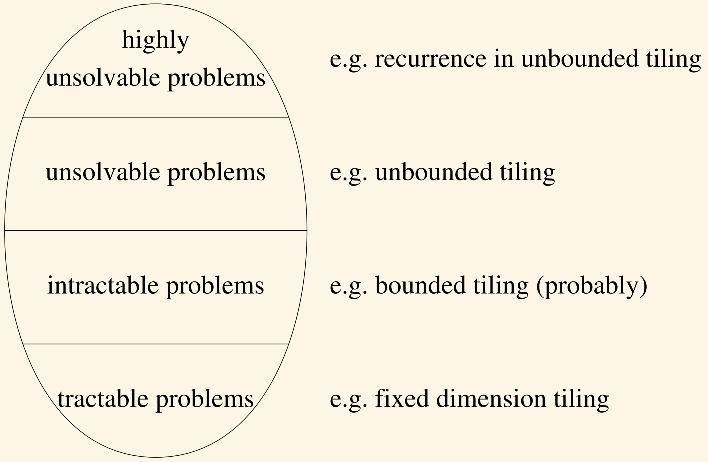


- No algorithm can be written to solve this problem.
- But this problem is provably, in some sense, even less solvable than the previous one! This might seem counter-intuitive. The previous problem couldn't be solved. Neither can this one. But this one is, in some sense, even worse! In fact, there is an infinite hierarchy of levels of unsolvability! You might be relieved to hear that we're not going to get bogged down in this kind of stuff in this module.

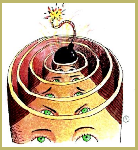


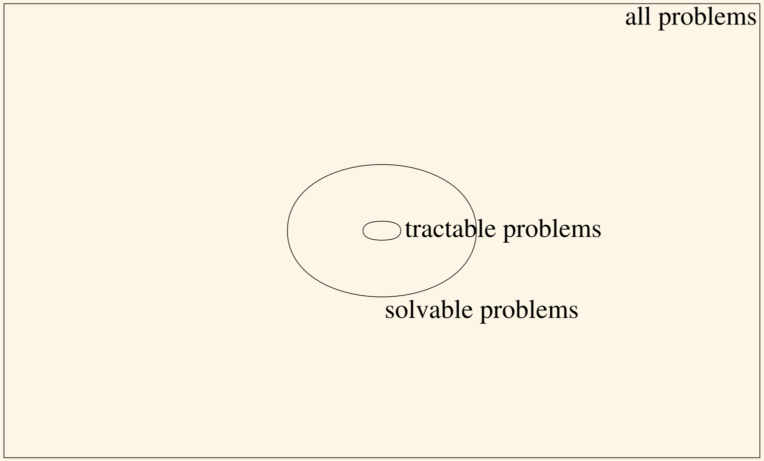
1.4. A Classification of Problems

- This lecture has given you an overview of four classes of problem.


highly unsolvable problems
e.g. recurrence in unbounded tiling
unsolvable problems
e.g. unbounded tiling
intractable problems
e.g. bounded tiling (probably)
tractable problems
e.g. fixed dimension tiling

- The set of solvable problems is an infinitesimally small subset of the set of all problems, and the tractable problems are a minute subset of the solvable problems. The next diagram is suggestive of these facts.

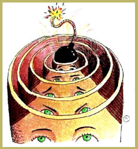




Acknowledgements

This module draws heavily on Harel's excellent book [Har92]. The discussion of tiling problems, along with several of the diagrams, are based on material in Chapter 8 of that book. The Venn diagram is based on one in Chapter 3 of [GL82].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.



References

[GL82] L. Goldschlager and A. Lister. *Computer Science: A Modern Introduction*. Prentice-Hall, 1982.

[Har92] D. Harel. *Algorithmics: The Spirit of Computing*. Addison-Wesley, 2nd edition, 1992.