Lecture 22. Hard Problems

- **Important properties of algorithms**

  ***Finite*:** **Guarranteed to terminate**

  ***Deterministic*:** **Always produces the same output for the same input**

- ***Efficient*** **algorithms execute in times that are no more than** ***polynomial*** **in the size of their inputs,** N

  N, N^2, $N + N^4$, **etc.**

- ***Inefficient*** **algorithms execute in times that are at least** ***exponential*** **in** N

  2^N, 10^N, $N!$, **etc.**

- **Some apparently simple problems have no known efficient solutions**

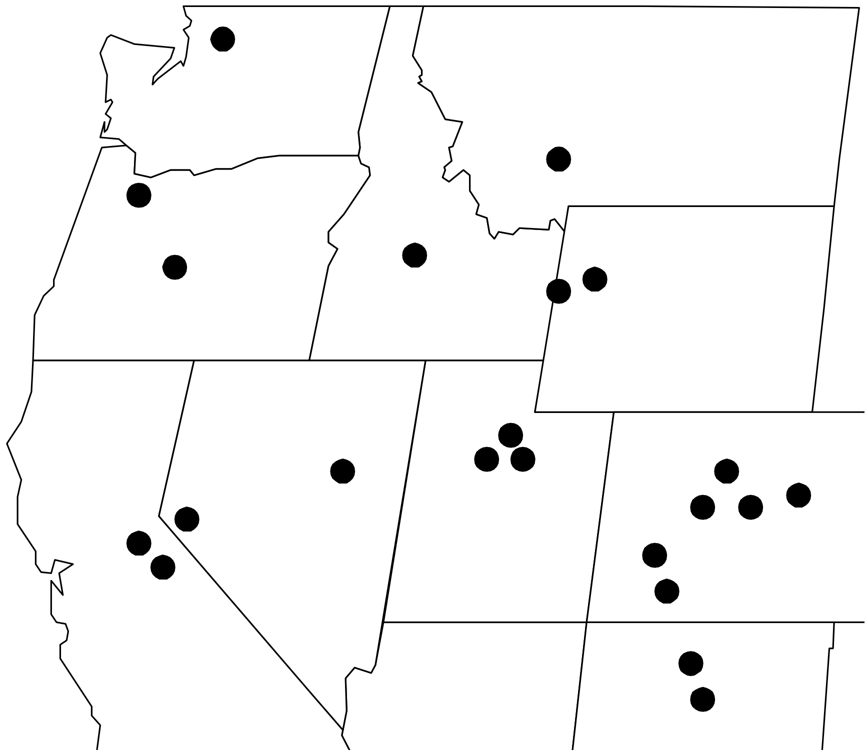
Traveling Salesman	**Find the minimum-cost tour of** N **cities**
Scheduling	**Schedule** N **jobs of varying length on two machines to finish by a given deadline**
Sequencing	**Arrange** N **4-letter fragments cut from a long string (with overlaps) into the original string (DNA sequencing)**
Satisfiability	**Assign true/false values to** N **logical variables so that a given logical formula is true**

The Traveling Skibum Problem

- **Visit *N* ski areas in the order that minimizes cost, e.g., distance**
- **To find an optimal tour, try all of them**

```
void visit(int k) {
        if (k == 1)
                checklength();
        else {
                int i;
                for (i = 0; i < k; i++) {
                        swap(i, k - 1);
                        visit(k - 1);
                        swap(i, k - 1);
                }
        }
}
```

visit(n);



- **Takes *N*! steps; no computer can run this for $N = 100$, because $100! \approx 10^{157}$**
- **Use *<u>heuristics</u>* to get good, but not optimal solutions, to hard problems**

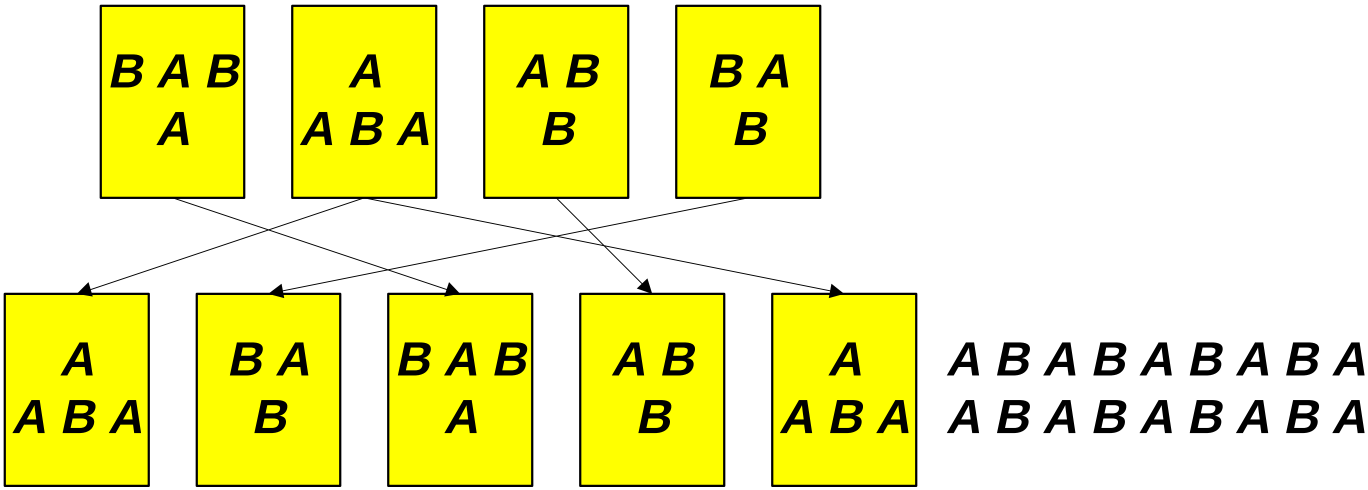
  TSP: Choose the ‘nearest neighbor’ as the next ski area on the tour
- **Hard problems can be your friends: Use encryption to send secret messages**

Unsolvable Problems

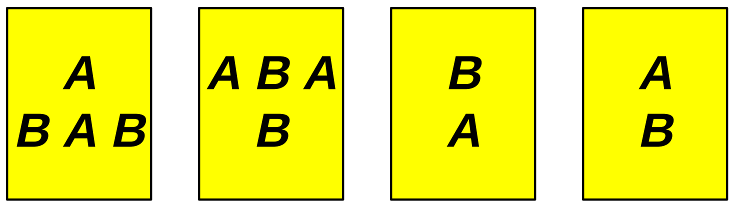
- **Oh oh… Are some problems *unsolvable*?**
- **Example: Post's Correspondence Problem**

  ***N* types of cards, each with a top string and a bottom string**

  Using as many of each card as needed, arrange them so that the top and bottom strings are identical (or say it's impossible)



- **There's no solution for the cards**



- **The bad news: Post's Correspondence Problem is unsolvable; you cannot write a program that determines if there is a solution for a given set of cards**

The Halting Problem

- **Write a C program that**

  Reads another C program, *P*

  Reads *P*'s input

  Determines whether or not *P* loops forever; that is, whether or not *P* halts

```
while (x != 1)
    if (x > 2) x -= 2; else x += 2;
```

7	**5**	**3**	**1**					***P* halts**
8	**6**	**4**	**2**	**4**	**2**	**4**	**…**	***P* loops on even inputs**

```
while (x != 1)
    if (x%2 != 0) x = 3*x + 1; else x /= 2;
```

7	**22**	**11**	**34**	**17**	**52**	**26**	**13**	**40**
	20	**10**	**5**	**16**	**8**	**4**	**2**	**1**	**does *P* halt for all odd integers?**
8	**4**	**2**	**1**						***P* halts**

The Halting Problem, cont'd

- **Theorem: The Halting Problem is unsolvable**
- **Proof by contradiction**

  Assume there is a program, `HALTS(`*P*,*y*`)`, that takes two inputs, a program *P* and its input *y*. If *P*(*y*) halts, `HALTS(`*P*,*y*`)` stops and prints 'Yes'; if *P*(*y*) does not halt, `HALTS(`*P*,*y*`)` stops and prints 'No'

  Build another program, `CONFUSE(`*x*`)`, that takes a legal C program *x* as input. If `HALTS(`*x*,*x*`)` prints 'Yes', `CONFUSE(`*x*`)` loops forever; if `HALTS(`*x*,*x*`)` prints 'No', `CONFUSE(`*x*`)` stops.

  Now, call `CONFUSE(CONFUSE)`:

  If `HALTS(CONFUSE,CONFUSE)` prints 'Yes', `CONFUSE(CONFUSE)` loops

  If `HALTS(CONFUSE,CONFUSE)` prints 'No', `CONFUSE(CONFUSE)` stops

  But `CONFUSE` can't do both! So, `HALTS` cannot exist ▮

- **Maybe C programs are too hard; what about TOY programs?**

  If the Halting Problem can be solved for TOY programs, it can be solved for C

  Use a C compiler to translate C programs to TOY code

- **Ditto for simple, abstract machines — for any machine that can *simulate* others**

More Integers or Reals?

- **Just how many unsolvable problems are there?**
- **A simpler question: Are there more integers or more even integers?**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**…**
0	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**…**

  There's a 1-to-1 correspondence, none missing, so there are as many integers as even integers!

- **Are there more integers or more reals? Try the same technique: Make a 1-to-1 correspondence between integers and reals, listing the reals in *<u>any</u>* order**

0	**0.*<u>1</u>*00100110000100101010010101…**
1	**0.0*<u>0</u>*0100100100101001001000101…**
2	**0.11*<u>1</u>*11111111111111111111111…**
3	**0.000*<u>1</u>*0000000100010001000001 0…**
4	**0.1000*<u>0</u>*000000000000000000000…**
5	**0.11100*<u>0</u>*111000111000111000111…**

  This *<u>diagonalization</u>* shows there's at least one real not on the list! 0.010011… the *<u>complement</u>* of the bits on the diagonal above

  There are infinitely more reals than integers

- **All possible programs correspond to the integers, all possible functions correspond to the reals: *<u>Most</u>* functions are not computable!**

Implications

- **Practical**

  Computing has its limitations; work within them

  Recognize and avoid unsolvable problems

  Recognize hard problems, don't try for optimal solutions

  Use heuristics for hard problems

  Abstract structures reveal much about practical problems

- **Philosophical (Buyer beware: Consult a 'real' philosopher for the truth)**

  We 'assume' that step-by-step reasoning can solve any technical problem

  'Not quite' says the Halting Problem

  Anything that is 'like a computer' suffers the same flaw

  - **Physical machines**
  - **Human brain?**
  - **Matter?**