1 Tutorial problems

We'll be using the following decision problems:

PRIME
Instance: A positive integer n.
Problem: Is n prime?

FACTOR-BOUND
Instance: A pair of positive integers m and n.
Problem: Does m have a divisor d such that $1 < d \leqslant n$?

Whenever one has a property, X, of languages, its "co-"property, co-X holds of a language L exactly when $\Sigma^* \setminus L$, i.e., the complement of L, has property X.

1. *Why are co-***P** *and* **P** *the same set of languages?*

   If a language, L, is in **P** then there is a deterministic polynomial-time algorithm that, given a word w, always halts and answers "yes" if $w \in L$ and "no" if $w \notin L$. Switching the answers shows that the complement of L is also in **P**, i.e., that **P** and co-**P** are the same set of languages.

2. *Please tell me if you have determined whether or not co-***NP***and* **NP** *are the same set of languages.*

   That would be great – it's an open problem. The issue now is that if a language is in **NP** it means the genie can provide us with a certificate that proves a given word belongs to the language (which we can verify in polynomial-time). But, to check that it's not in the language would (at least in the most obvious approach) require considering all possible certificates and checking that they are invalid. But the set of possible certificates is certainly exponentially large – so that's not good enough.

3. *Show that* PRIME *is in co-***NP***. In fact, though this is by no means obvious,* PRIME *is in* **P**.

   The complement of PRIME is COMPOSITE and a certificate for an integer n to be composite is just any proper factor d (i.e., $1 < d < n$ and n is a multiple of d). We can easily check the validity of a certificate in polynomial time (even when we

write the integers in binary rather than unary – as we're obliged to do) since the basic arithmetic operations are easily seen to be in **P**.

4. *Show that* FACTOR-BOUND *is in* **NP**.

   For a positive instance of FACTOR-BOUND the genie can just tell us an appropriate d and we can check the details.

5. *Show that* FACTOR-BOUND *is in co-*$\mathbf{NP}$ *(hint(?): this requires knowing that* PRIME *is in* **P***).*

   The pair (m, n) is a negative instance for FACTOR-BOUND if and only if all of m's prime factors are greater than n. So, a prime factorisation of m allows us to check that. That is, we ask the genie to write down a prime factorisation of m. Since PRIME is in **P**, we can check that the factors given are prime, that their product is m and that all are greater than n.

6. **PSPACE** *is the class of all decision problems that can be solved on a Turing machine with a polynomial bound on the use of space (that is, there are some constants A and c such that if the input is of size n, the tape-head never moves more than An^c cells away from the left-hand end of the tape).*

   CLOBBER *is a two-player game played with black and white stones occupying (some finite set of) squares within an infinite grid. One player, Bella (she/her) controls the black stones and the other, William (he/him) controls the white ones. A move for Bella is to take one of her stones, move it to an adjacent square occupied by one of William's stones, and remove his stone from the board. William moves similarly, but using his stones to clobber Bella's. The game is over when one of the players is unable to make a move (because there are no pairs of stones of opposite colours next to each other), and the player who made the last move wins. Consider the problem:*

   WIN-CLOBBER
   Instance: A CLOBBER position
   Problem: Does Bella have a winning strategy moving first?

   Show that WIN-CLOBBER *is in* **PSPACE**. *Does it seem likely to be in* **P**, **NP**, *or co-*$\mathbf{NP}$*?*

   (This is a general argument about games of this type – nothing special about CLOBBER)

Since each move in a game of CLOBBER decreases the number of stones by one, the length of a CLOBBER game is certainly bounded by the number of stones present. Furthermore, all stones always occupy positions that were occupied in the initial position, so any position can be recorded in space proportional to the size of the original game.

The reachable positions from the initial one can be thought of as a finite tree (we're only concerned about space and not time, so if the same position is reachable in two different ways there's no problem with considering it twice). At the leaf vertices of this tree (positions where no moves are possible – also called *terminal positions*) we know who the winner is (the player who made the last move) so we can imagine labelling them all with B or W according to who won.

Now at any node in the tree if all of its children have labels and it is Bella's turn then, if any one of the children is labelled B, we can label that node B (she has a good move - to a node labelled B), while if all are labelled W we must label that node W too (any move by Bella can be countered). There's a corresponding set of rules for labelling nodes when it's William's turn.

But the tree is exponentially big! We can't store the whole tree. Fortunately, we don't need to – because we can do a depth-first search to label the root node. Consider the pseudocode (using 0 and 1 for the player labels)

Algorithm 1 $\mathrm{Label}(n, p)$ (label node n with player $p \in \{0, 1\}$ to play)

```
if n is terminal then
  return 1 − p {In a terminal position, the player whose turn it is has lost.}
end if
for all children n′ of n do
  if Label(n′, 1 − p) = p then
    return p {n′ is a good move for p}
  end if
end for
return 1 − p {p has no good move}
```

The extra storage needed is in the list of children of a given node (so that we can

execute the "for all" statement) and the call stack of the algorithm (so that we can make the recursive call inside that loop). So the total amount of space required is bounded by the product of :

- the maximum number of moves available at any point,
- the maximum possible length of a game, and
- the space required to represent a position.

Since all of these are clearly polynomial (in fact linear) in the size of the description of the initial position, this algorithm runs in polynomial space, i.e., WIN-CLOBBER is in **PSPACE**.

The general rule is that tree search problems that can be accomplished by depth-first-search are usually in **PSPACE**.

It seems unlikely that WIN-CLOBBER is in **P**, **NP**, or co-**NP** because the only (obvious) method for determining whether Bella has a winning strategy does require investigating most (or all) of this tree which is exponentially large in the size of the position in general.

In fact it turns out that even the solitaire problem for Clobber,"can a position be reduced to a single stone", is **NP**-complete. As of 2009 the complexity status of the two-player version is unknown, but it's believed to be **PSPACE**-complete.