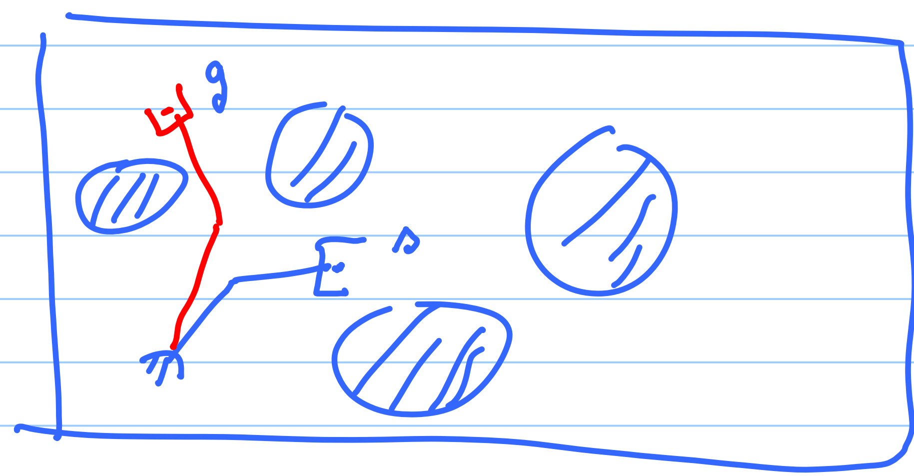
Note Title 1/10/2012

Lecture 2

Q : How difficult (precise complex) is the path planning problem?



BACKGROUND: How do we measure complexity of a "problem"

Two steps

1) Complexity of an algorithm

2) Complexity of a problem is measured by the "best" known (unknown) algorithm that can solve it.

1) Complexity (Run time) of an algorithm : Big "O" notation

<u>Def :</u> $f(n) = O(g(n))$ iff $\exists$ $c, n_0 > 0$ such that

$$|f(n)| \leq c\,|g(n)| \quad \forall\, n > n_0$$

$$f(n) = 5n^2 + 3n + \log(n)$$

$$g(n) = n^2$$

$$\Rightarrow O(f(n)) = O(n^2)$$

<u>simple example:</u> Sorting numbers

<u>INPUT:</u> $A[1:n]$ is given (array of #s)

$$[a_1, a_2 \ldots a_n]$$

<u>Prob:</u> arrange these #s in descending order

```
for i ← 1, n
    for j ← 1, n
        if A[i] < A[j]
            temp = A[j]        }
            A[j] = A[i]        } swap A[i] + A[j]
            A[i] = temp        }
        end if
    end for
end for
```

$O(n^2)$

$(3+1)n^2$

$f(n) = 4n^2$

Two classes of problems

why?

1

best known alg.

are polynomial in n

$O(n^k)$ k is known

P

2

best known alg

are exponential in n

$O(2^n)$

$\Downarrow$

run time increases drastically as a func. of n

even for low values of n (10–100), run time can become quite large (days)

<u>Ex:</u> unit op. (assignment, addition) is say 1 msec.

$n = 10 \quad 2^{10} \rightarrow 1.024$ sec

$= 18 \quad 2^{8} \cdot 2^{10} \rightarrow 256$ sec

$= 26 \quad 2^{8} \cdot 2^{18} = 256 \times 256$ sec

≈ 15 hrs

Examples of class 2 :

(a) Set partition (b) Boolean Satisfiability

a) Given a set $S = \{ l_1, l_2 \dots l_n \}$ of n numbers. Can the set be partitioned into two disjoint subsets S_1 + S_2 :

$$S_1 \cup S_2 = S$$
$$S_1 \cap S_2 = \phi$$

$$\sum_{l_i \in S_1} l_i = \sum_{l_j \in S_2} l_j$$

b) Given n literal x_i, $i = 1, n$

$x_i \in \{0, 1\}$

<u>Boolean Sentence:</u> $\bigwedge_{i=1}^{k} C_i$ <u>CNF</u>

<u>Clauses</u> $\longrightarrow$ $C_i = \bigvee x_j(i)$

$$C_1 = (x_1(1) \vee x_2(1) \vee \bar{x}_3(1))$$

<u>Prob:</u> Does $\exists$ an assignment of literals x_i such that

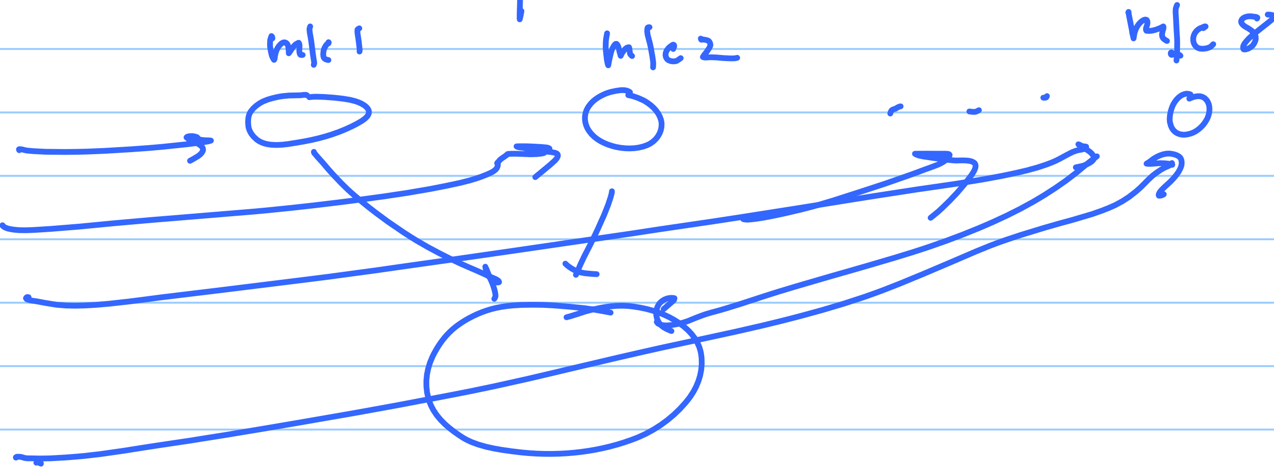
the Boolean Sentence is true.

"<u>polynomial verifiability</u>": if a "solution" is proposed, one can (certificate)
verify if it is true or not
in polynomial time.

special machine: imagine we can have arbitrarily large parallelism implemented:

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
⋮		
1	1	1

8



Non-deterministic m/c: Boolean satisfiability runs in poly time

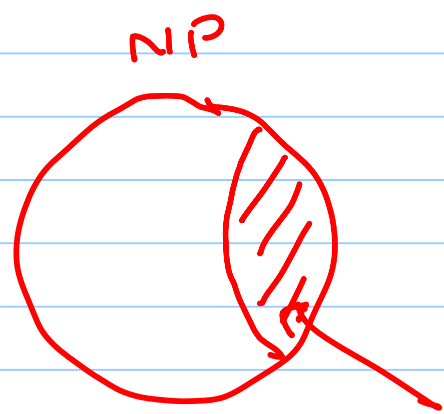
$\boxed{\text{NP problems}}$

Non-deterministic poly

$P \subseteq NP$

$NP \overset{?}{\subseteq} P$

open Q in CS.



"Cook's theorem"

"hardest problems in NP"

Notion of "polynomial reducibility":

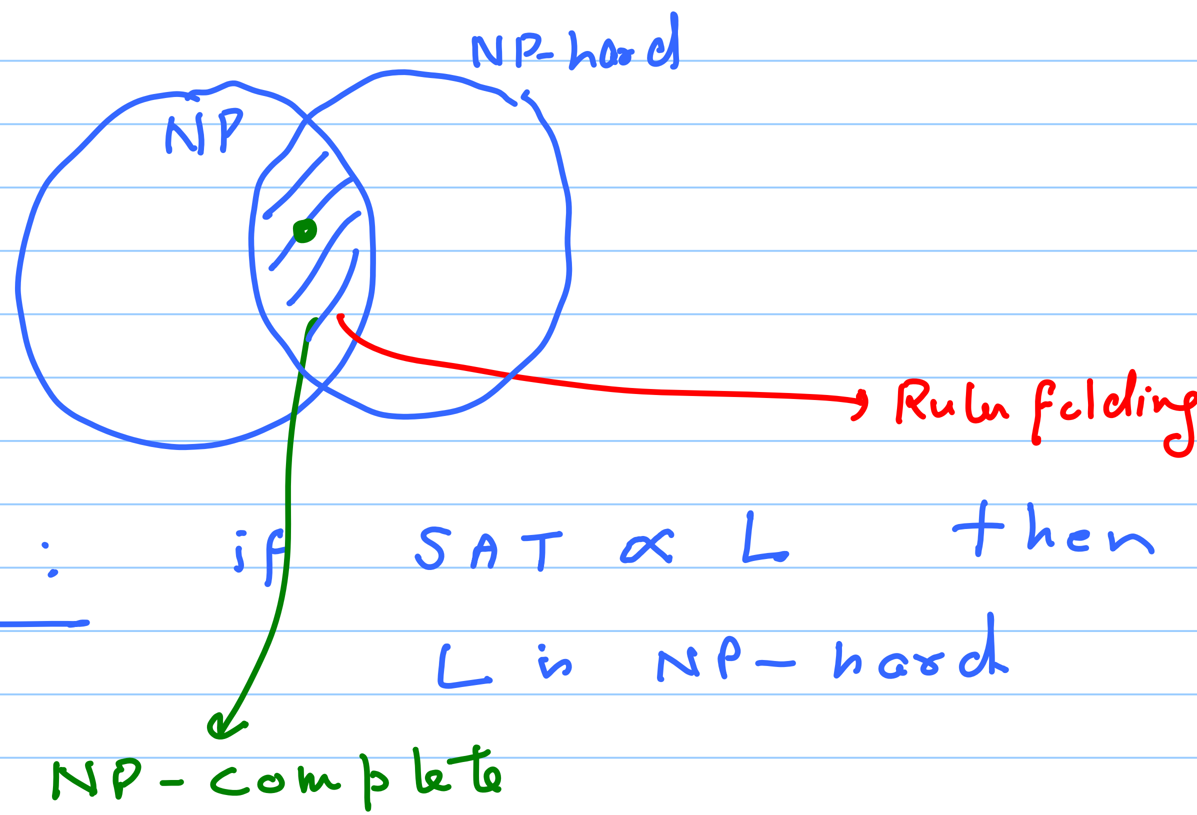
Let L_1 & L_2 be two problems.

$L_1 \propto L_2$ or (L_1 reduces to L_2)

<u>iff</u> there is a way to solve L_1 by a deterministic poly algorithm using a deterministic alg. that solves L_2 in polynomial time.

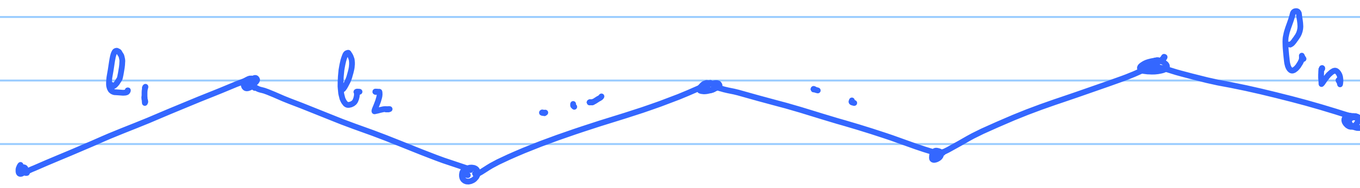
$\Rightarrow$ L_2 is at least as hard as L_1

"Cook showed that Boolean Satisfiability is the hardest problem in NP"



<u>NP-hard:</u> if $SAT \propto L$ then L is NP-hard

Ruler folding problem:



Given this ruler, and a +ve integer k.

Prob: Can this ruler be folded in length $\leq k$?

NP - Complete !!

Set Partitioning $\propto$ Ruler folding

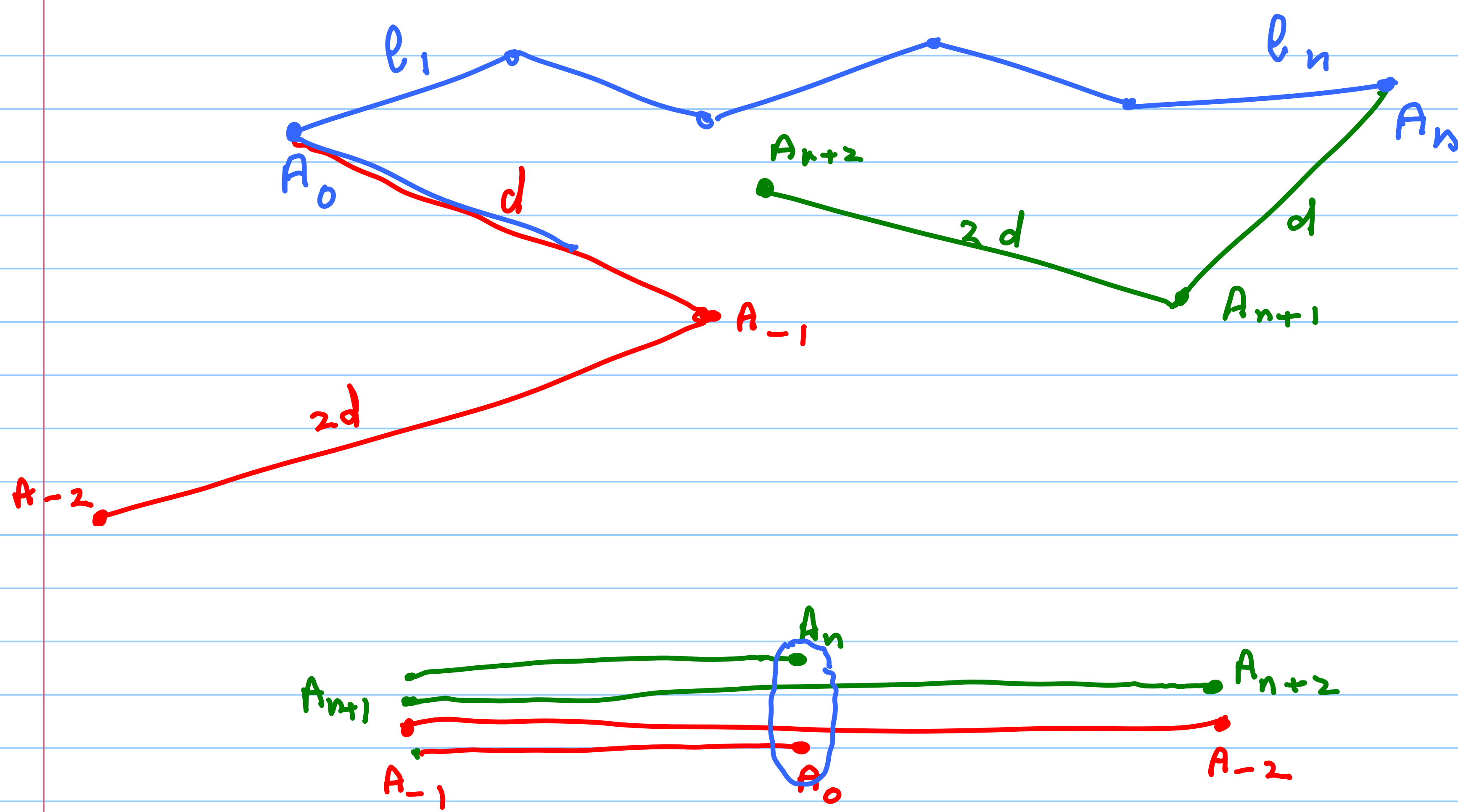
$$S = \{ \ell_1, \ldots \ell_n \}$$

find S_1, S_2 : $S_1 \cup S_2 = S$

$S_1 \cap S_2 = \phi$

$$\sum_{\ell_i \in S_1} \ell_i = \sum_{\ell_j \in S_2} \ell_j$$

Ruler Construction : $\sum_{\ell_i \in S} \ell_i = d$



ℓ_1
ℓ_n
A_0
A_n
A_{n+2}
d
$2d$
d
A_{-1}
A_{n+1}
$2d$
A_{-2}
A_n
A_{n+1}
A_{n+2}
A_{-1}
A_0
A_{-2}

$\Rightarrow$ $\boxed{\text{Rulerfolding problem comp. is exponential in the \# of links of the rules}}$

$\longrightarrow$ Canny + Reif [1992]? $\longrightarrow$ moving obstacles problem

NP-hard $\longleftrightarrow$ exponential is in # of obstacles

Schwartz + Sharir [1981]:

exponential in # of
links of the robot

1) geometric variable : captures the environment complexity

2) # of links / degrees of freedom in the robot $\longrightarrow$ captures complexity of the robot