Parametric Search (Megiddo '83)

general technique to reduce problem to decision problem

compute t^* unknown

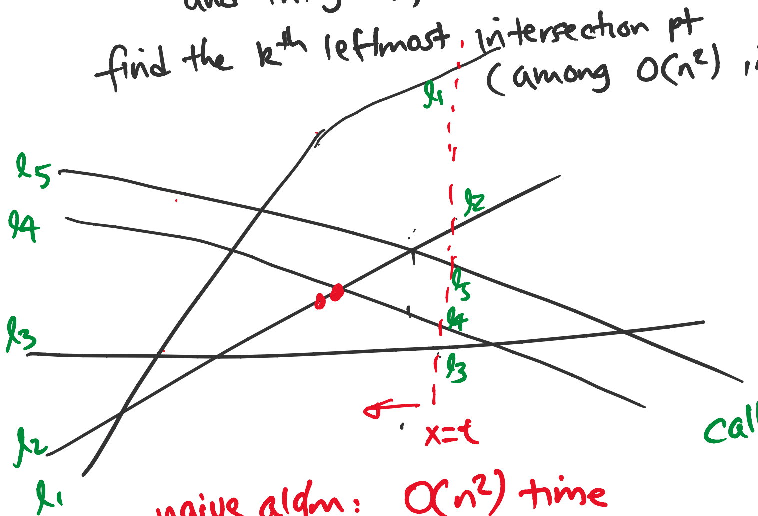
given value t, decide whether
$t^* < t$
$> t$
$= t$

let T_D = time for solving decision problem

If t^* lies in finite universe U,
can solve orig. problem in
$O(T_D \log U + U \cancel{\log U})$ time
by binary search.

But what if U is too big?

Ex1 Given n lines in $\mathbb{R}^2$, $\ell_1, \ldots, \ell_n$ $\quad \ell_i : y = m_i x + b_i$
and integer k,
find the k^{th} leftmost intersection pt (among $O(n^2)$ intersections)



$k=6$

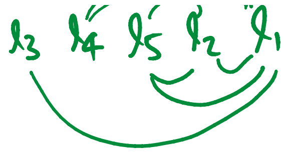
called slope selection problem (by duality)

naive algm: $O(n^2)$ time
better?

Decision Problem: given t,
decide whether there are $\geq k$ intersections to the left of $x=t$

$\Downarrow$

reduces to sorting



reduces to sorting
+ counting inversions in a permutation }

$\Rightarrow$ $O(n \log n)$ time

But can't apply binary search to solve orig prob.

Ex2 exact nearest neighbor search in $\mathbb{R}^d$

Decision Problem: given query pt q & value t,
is nearest neighbor dist $\le t$?
i.e. does ball(q,t) contain a pt?
radius

ball range emptiness reduces to halfspace range emptiness in $\mathbb{R}^{d+1}$

$\tilde{O}(n^{\lceil d/2 \rceil})$ space, $O(\log n)$ query time
$O(n)$ space, $\tilde{O}(n^{1-\frac{1}{\lceil d/2 \rceil}})$ query

Many more exs...

The Technique.

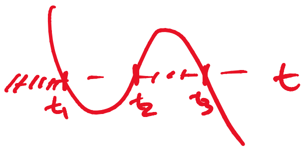
let $\mathcal{A}(t)$ be the decision alg'm with T_D time

idea - simulate $\mathcal{A}(t)$ for $t = t^*$
but t^* is not known!
at some pt, $\mathcal{A}$ will make a comparison that depends on t^*

(e.g. is l_i below l_j at $x = t^*$?
i.e. $m_i t^* + b_i \le m_j t^* + b_j$
i.e. $t^* \le \frac{b_j - b_i}{m_i - m_j}$ $(m_i > m_j)$

(in CG, comparisons usually reduce to testing signs of const-deg polynomials)

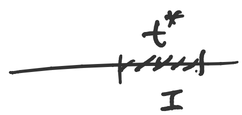
(in CG, comparisons usually reduce to testing signs of const-deg polynomials)
which reduces to comparing t^* with const # of values)



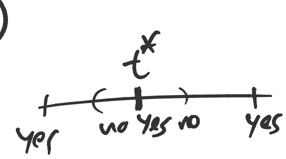
How to resolve the comparison?
by calling the decision alg'm

$O(T_D)$ steps to simulate,
each costing $O(T_D)$ time

$\Rightarrow$ $O(T_D^2)$ time



at the end,
get an interval I s.t.
$\forall t \in I$, $\mathcal{A}(t)$ has same as $\mathcal{A}(t^*)$



$\Rightarrow$ I is $\{t^*\}$. $\square$

Thm If decis. prob can be solved in T_D time,
then orig. problem can be solved in
$O(T_D^2)$ time.

Ex nearest neighbor search
$\tilde{O}(n^{\lceil d/2 \rceil})$ space, $O(\log^2 n)$ query time.

The Technique Refined:

Suppose there is a parallel decision alg'm $\mathcal{A}_{par}$
that requires Π_D processors and
τ_D time

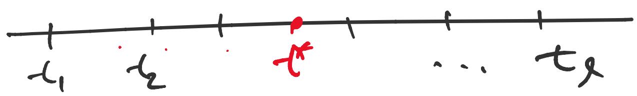
e.g. sorting $\rightarrow$ $O(n)$ processors
$O(\log n)$ time.
in PRAM
(or "AKS network")

idea. Simulate $A_{par}(t)$ for $t = t^*$

at each time step,

A_{par} will make $O(\Pi_D)$ comparisons that depend on t^*

(e.g. $t^* \overset{?}{\le} t_1$, $t^* \overset{?}{\le} t_2$, ..., $t^* \le t_\ell$) $\ell = O(\Pi_D)$



resolve all $O(\Pi_D)$ comps
by $O(\log \Pi_D)$ calls to
the (sequential) decision alg'm A !

Thm If decision prob. has sequential alg'm with T_D time
& a parallel alg'm with Π_D processors
& τ_D time,

then orig prob. can be solved in

$$\boxed{O\left(\left(\Pi_D + T_D \log \Pi_D\right) \cdot \tau_D \right)} \text{ time}$$

Ex slope selection

$T_D = O(n \log n)$

$\Pi_D = O(n)$
$\tau_D = O(\log n)$ } by parallel sorting

$$\Rightarrow O\left(\left(n + n\log n \cdot \log n \right) \cdot \log n \right)$$

$$= \boxed{O(n \log^3 n)} \text{ time}$$

Ex nearest neighbor $\tilde{O}(n^{1-\frac{1}{\lceil d/2 \rceil}} \log^c n)$ query time

Ex nearest neighbor
$O(n)$ space, $\tilde{O}(n^{1-\frac{1}{\lceil d/2 \rceil}} \log^c n)$ query time

Rmk Cole '87 improves to
$$O\left((\Pi_D + T_D) \cdot (T_D + \log \Pi_D) \right) \text{ time}$$
in some cases

(idea - at each step, use 1 call to resolve half of comps ..)

$\Rightarrow$ $O(n \log^2 n)$ time for slope selection

Rmks - only need to parallelize steps that depend on t

$\mathcal{A}_{par}$ doesn't need to solve decision problem
$\leftarrow$ it can decide membership in any finite universe containing t^*

Disadvantages:
- need parallelization ←
- hard to implement
- extra logs ←

Simpler alternatives?