Decentralized gradient descent:
how network structure affects convergence

Jason Yang, Jun Wan, Hanshen Xiao

Motivation

Suppose several agents want to train a machine learning model:

- each agent has their own training data
- the agents want to train their model on the collective data of all the agents
- no agent wants to release their data to anyone else
    - Ex. these agents could be hospitals, each holding confidential medical data

General Model

- Let agent i's cost function be $f_i(x)$
  - $f_i(x)$ is private to everyone except agent i
- All the agents want to minimize $tf(x)=mean(f_i(x))=1/N*sum(f_i(x))$
- All agents are connected in a graph
  - Every agent has a self-loop to themself

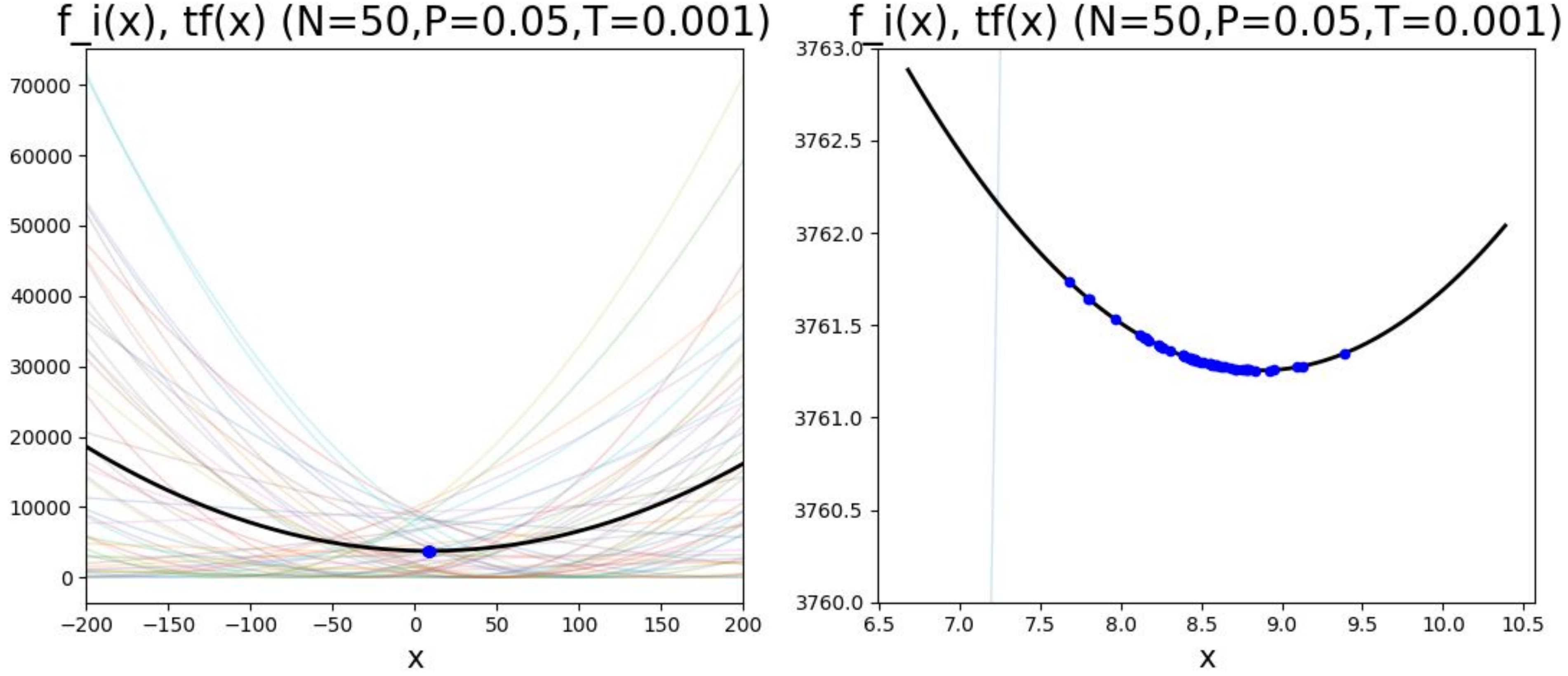
General Model (cont.)

- Each agent i has a random initial value $x_i(0)$ in round 0
- In round k:
  - Every agent i sends their $x_i(k-1)$ to all their neighbors j
  - Every agent i sets $x_i(k) \leftarrow F(S_i(k)) - T^* \nabla f_i(x_i(k-1))$
    - $S_i(k)$: set of values agent i received in round k
    - F: some aggregate function over a set, ex. Mean, median, trimmed mean
    - T: step size
    - Compare to standard gradient descent: $x_i(k) \leftarrow x_i(k-1) - T^* \nabla f_i(x_i(k-1))$

Initial Model

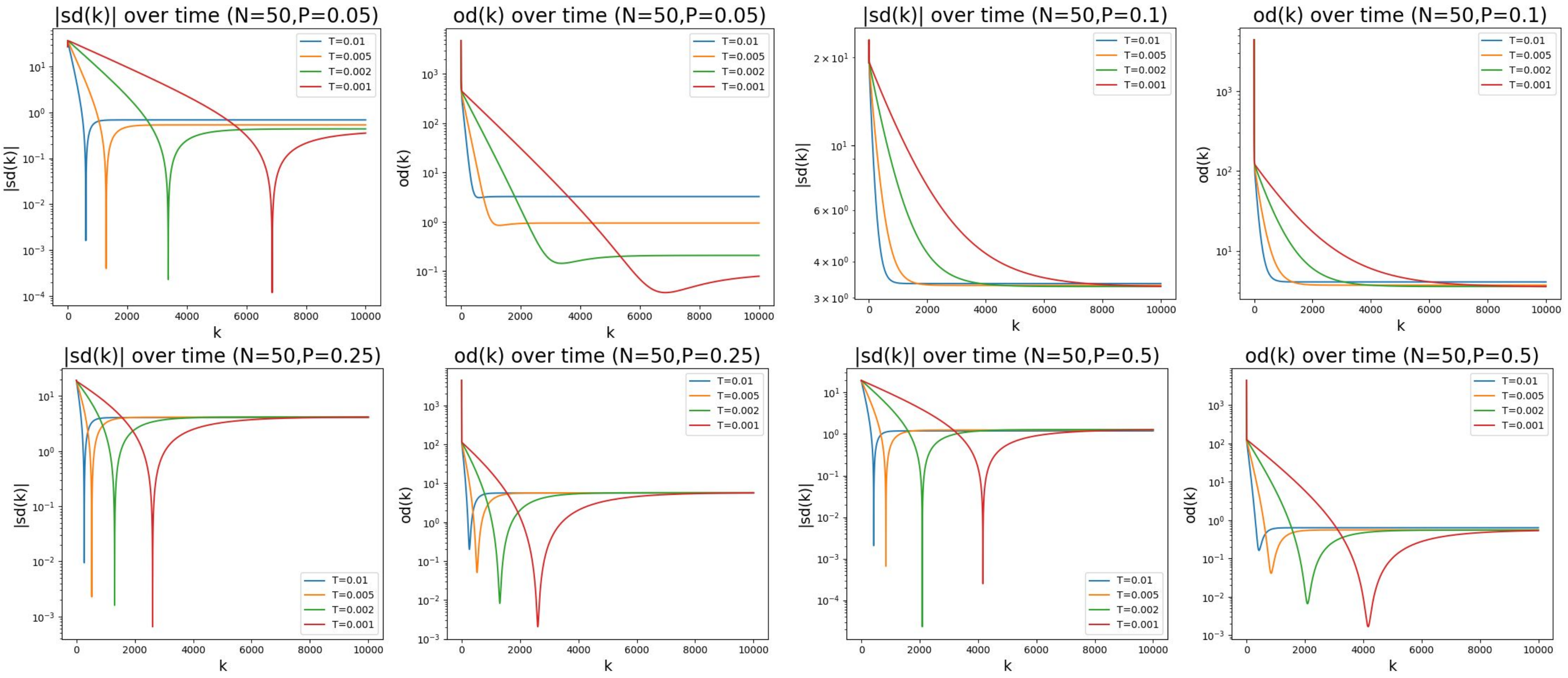
- $f_i(x)$ is of the form $(a_i x - v_i)^2$ for $x \in \mathbf{R}$
- $a_i \in [0,1)$, $v_i \in [-100,100]$, $x_i(0) \in [-200,200]$ uniformly random
- We consider random graphs
  - every edge has probability $P \in \{0.05, 0.10, ... 0.95, 1\}$ of being made
  - We repeatedly generate random graphs until we have one that is connected
- F is the mean
- N fixed to 50
- $T \in \{0.01, 0.005, 0.002, 0.001\}$
- We focus on two quantities of the DGD:
  - $sd(k) = mean(x_i(k)) - argmin_{\mathbf{R}}(tf)$
  - $od(k) = mean(tf(x_i(k))) - min_{\mathbf{R}}(tf)$
- We arbitrarily end DGD at 10000 rounds

Sample test set of f_i and DGD: line, 10000 rounds



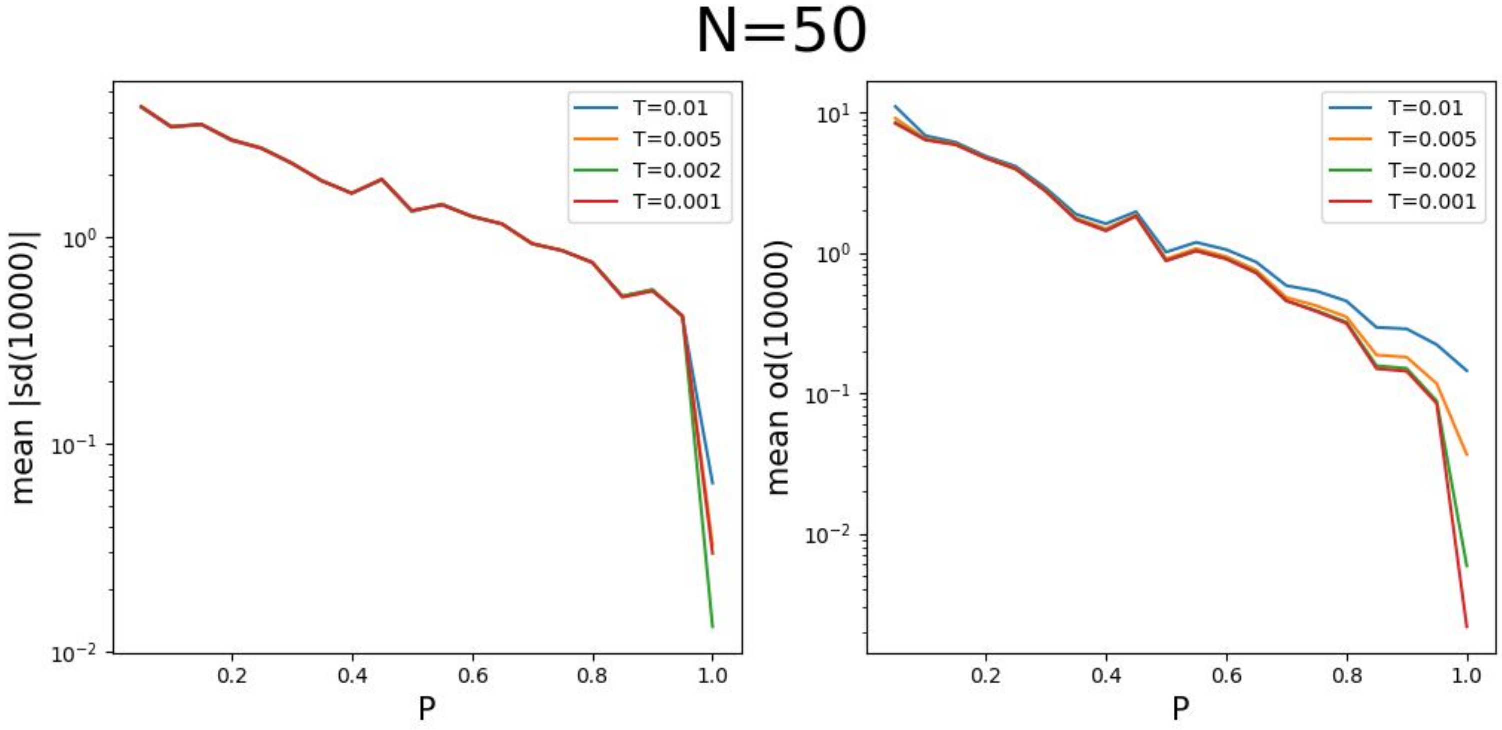
Sample DGDs for various P

DGD converges for various P and T in 10000 rounds



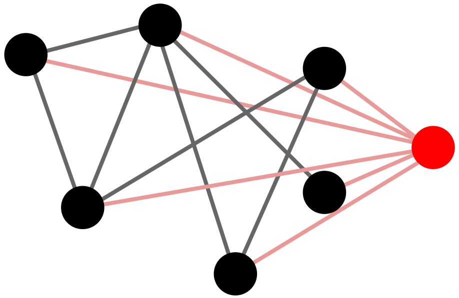
Mean |sd(10000)|, od(10000)

For each (T,P), test DGD on 100 test sets



Adversary

- There are *A* corrupt agents added to graph
  - Can send anything they want to worsen the DGD
- We assume each corrupt agent:
  - Is connected to all honest agents
  - Has exact knowledge of the DGD algorithm



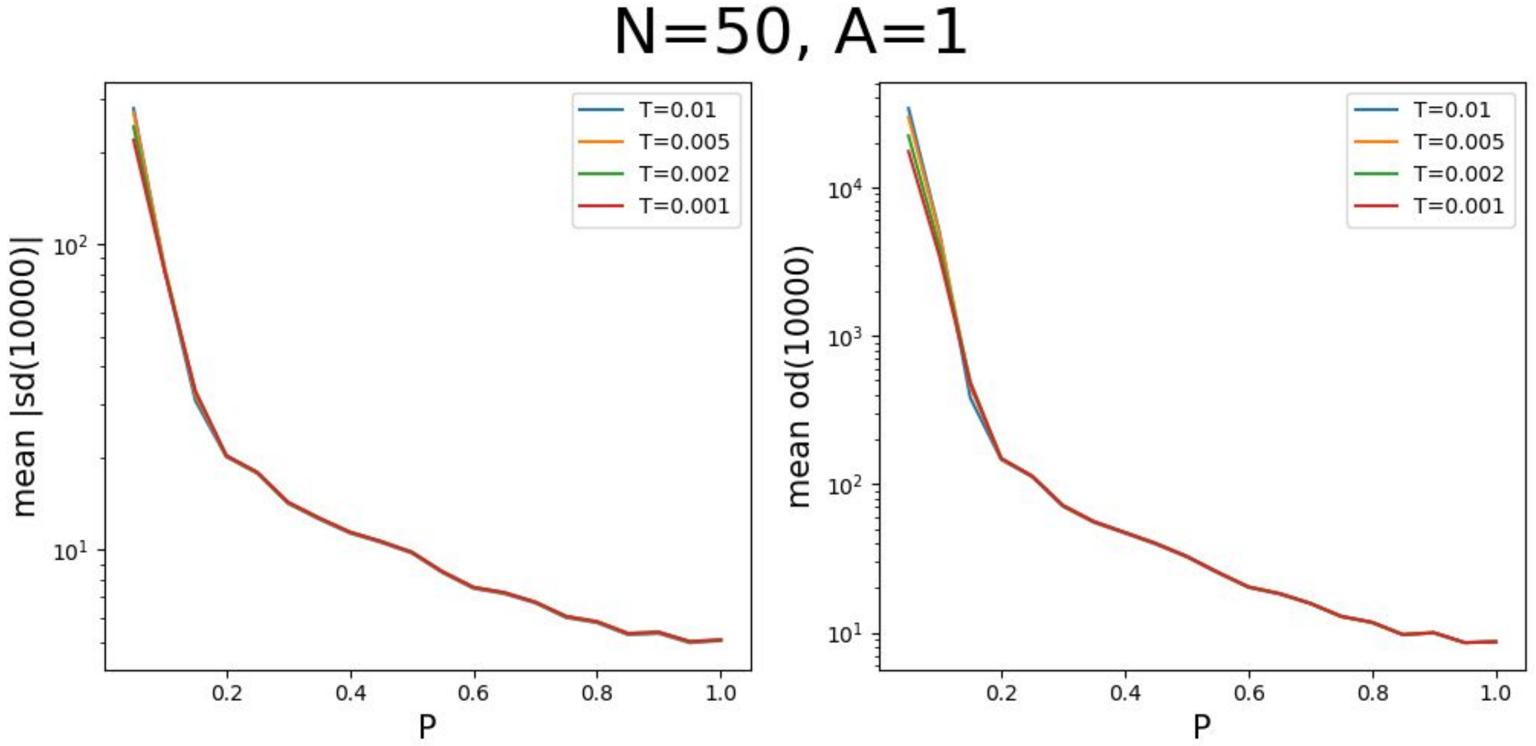
N=6, A=1

1 corrupt agent

- Naturally the adversary wants to send very high or very low values to the honest nodes in order to throw them off
- → Change F to trimmed mean [1:-1] (i.e. remove lowest and highest values)

Mean |sd(10000)|, od(10000): 1 corrupt agent

Corrupt agent always sends super high value (1000000)

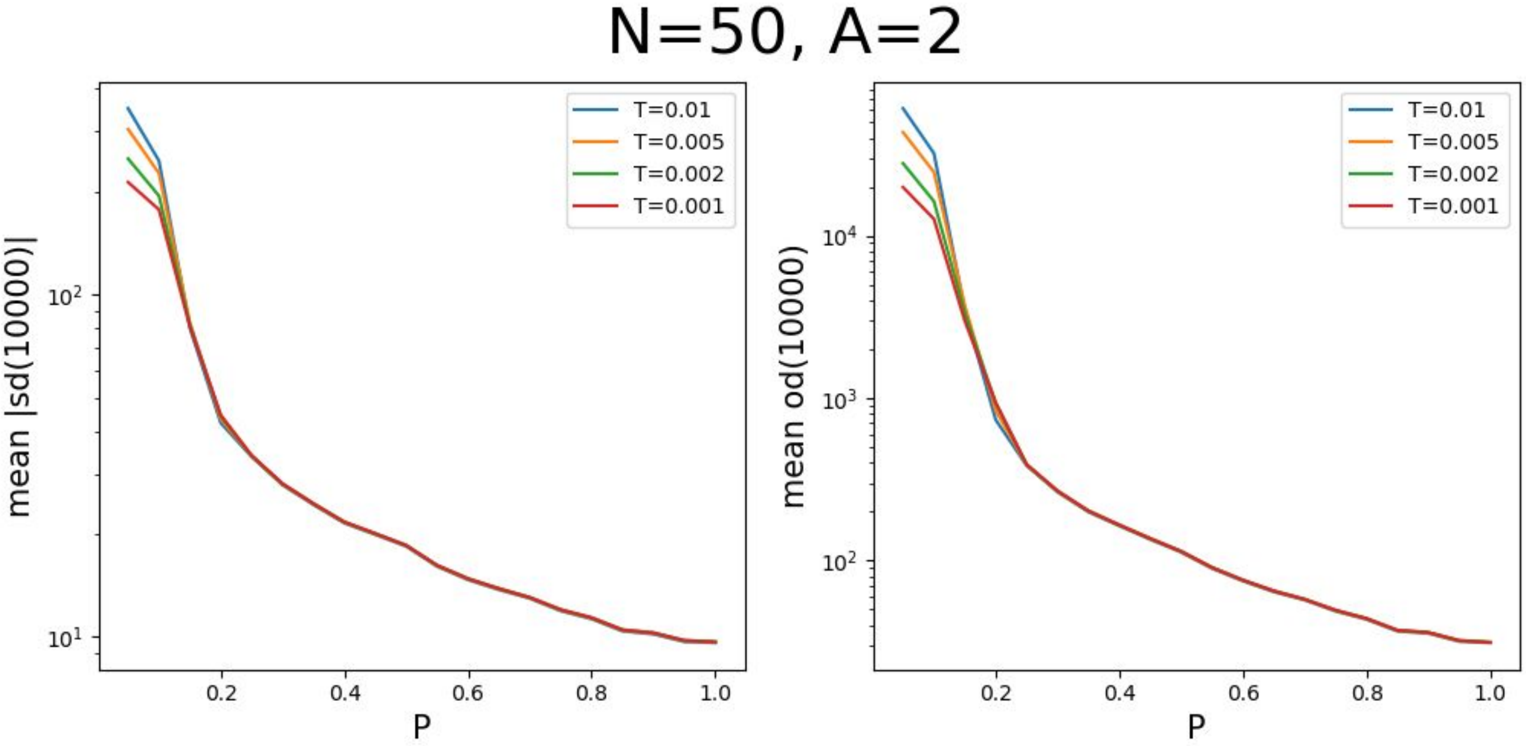


2 corrupt agents

- F now trimmed mean [2:-2] (remove lowest 2 values and highest 2 values)
- During $x_i(k) \leftarrow F(S_i(k)) - T^* \nabla f_i(x_i(k-1))$:
    - If $|S_i(k)| \leq 4$, replace $F(S_i(k))$ with $x_i(k-1)$

Mean |sd(10000)|, od(10000): 2 corrupt agents

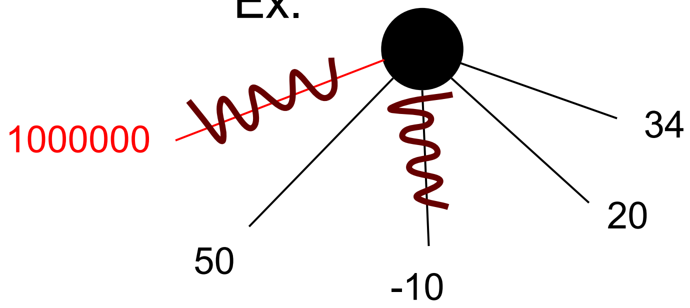
Both corrupt agents always send super high value (1000000)



Intuition for DGD behavior under adversary

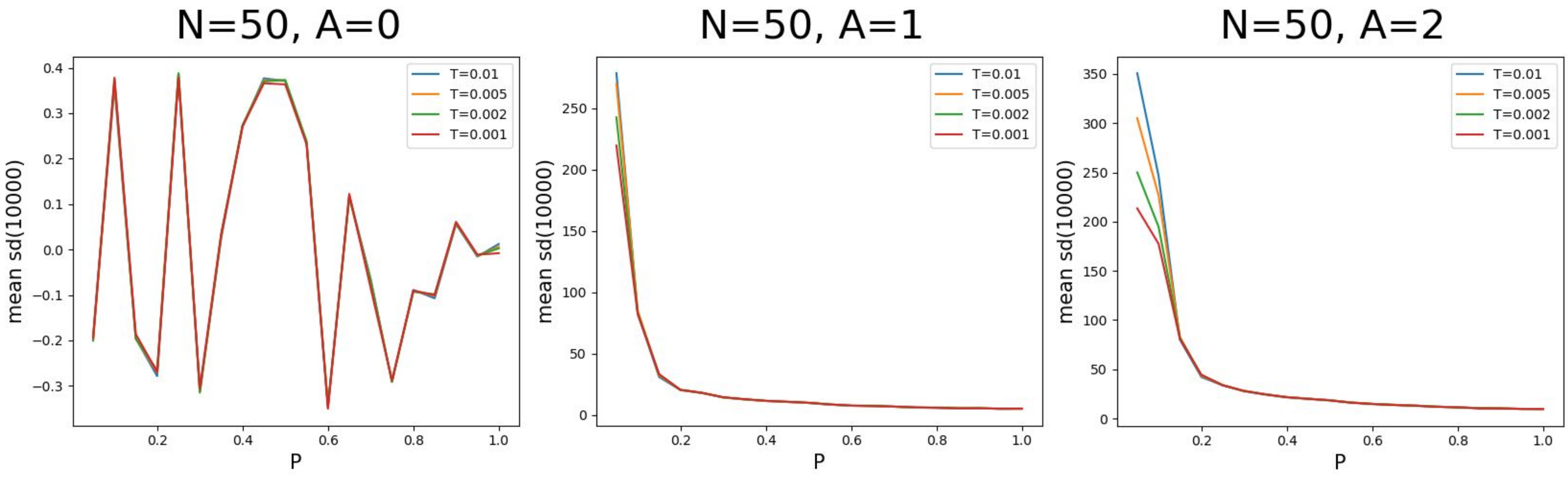
- Ex. A=1, adversary always sends super high value
  - Each honest agent trims highest and lowest value
  - → trims adversary's value, but also lowest value of neighboring honest node
  - → honest agents' $x_i(k)$ get skewed to higher values

Ex.



Mean sd(10000): 0, 1, 2 corrupt nodes

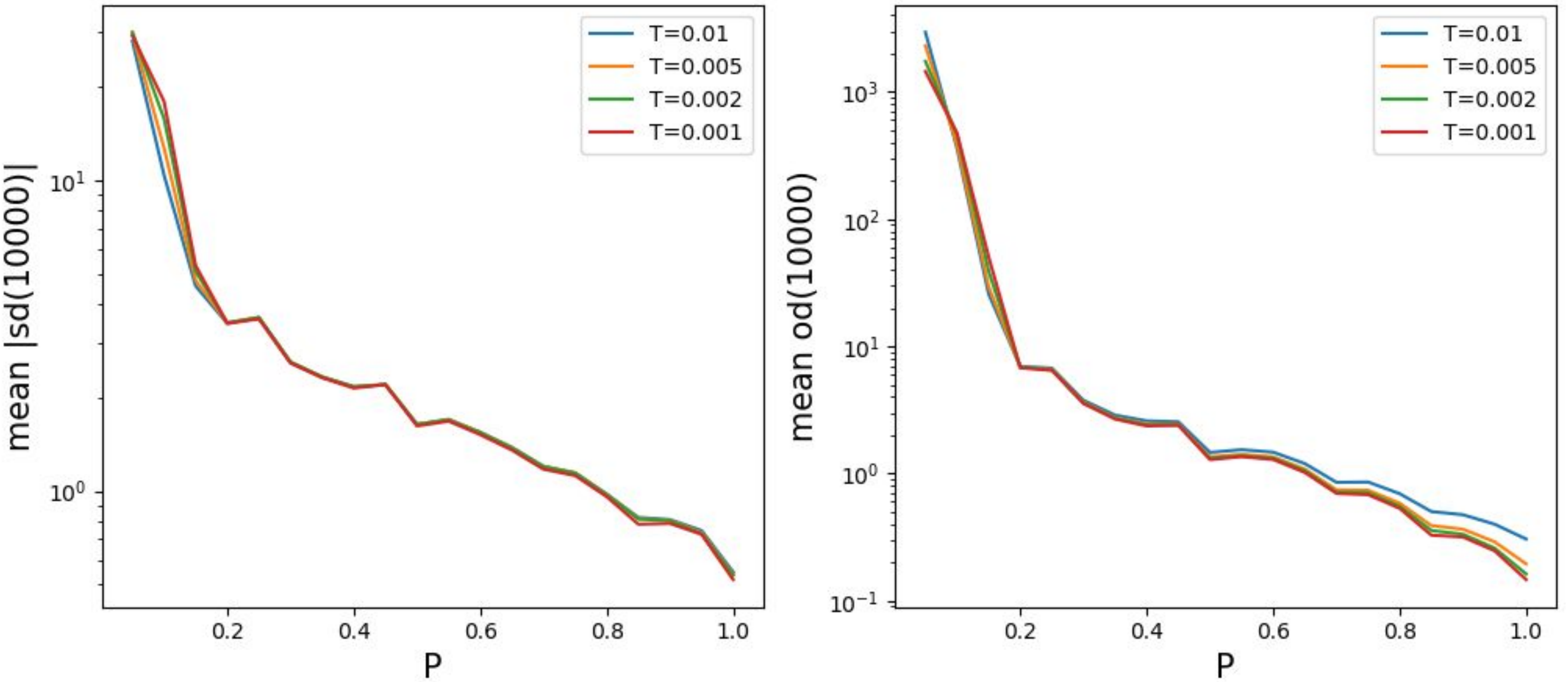
- A=0: mean sd(10000) close to 0
- A=1, 2: sd(10000) always +



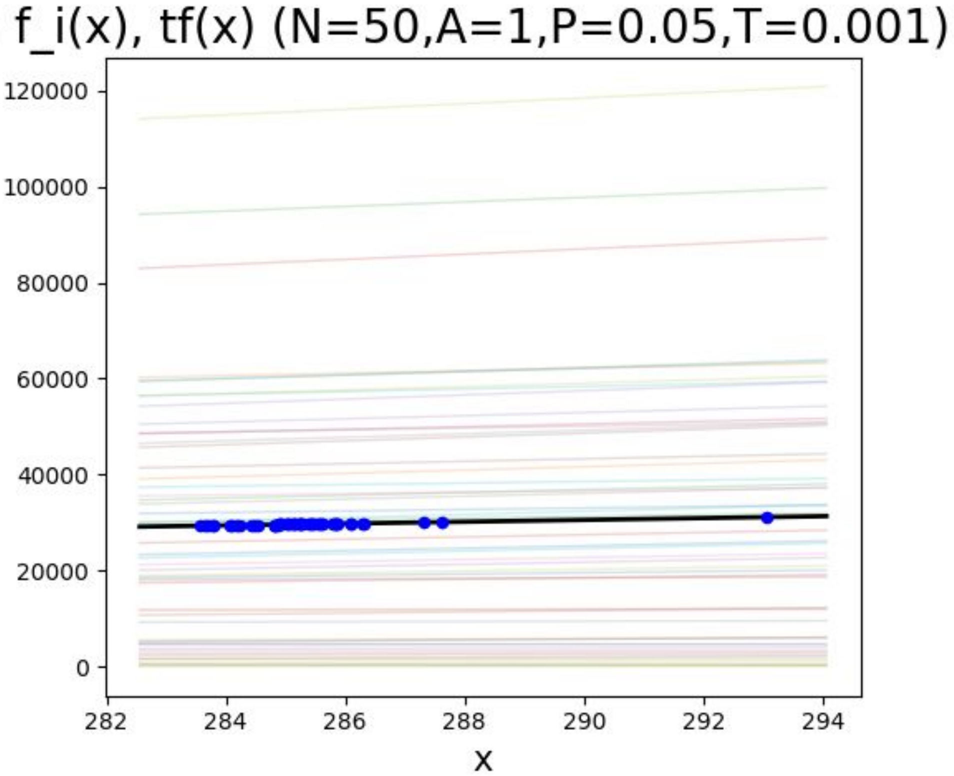
Equivocating Adversary: 1 corrupt node

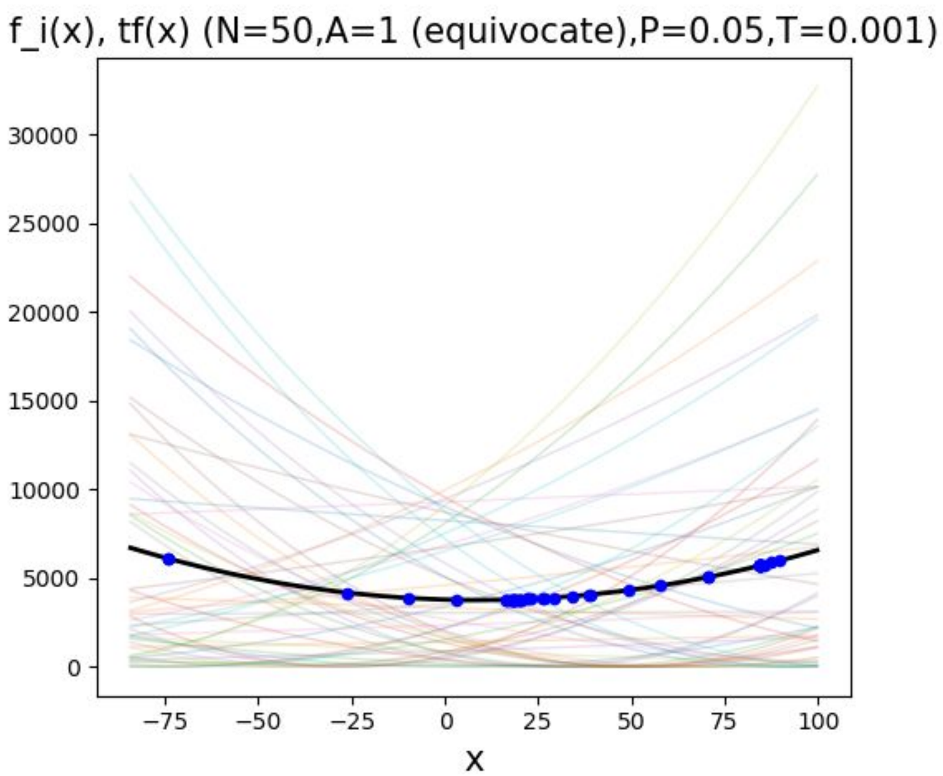
Adversary sends 1000000 to N/2 arbitrarily chosen agents and -1000000 to all other agents

N=50, A=1 (equivocate)



Equivocating Adversary: separation of x_i



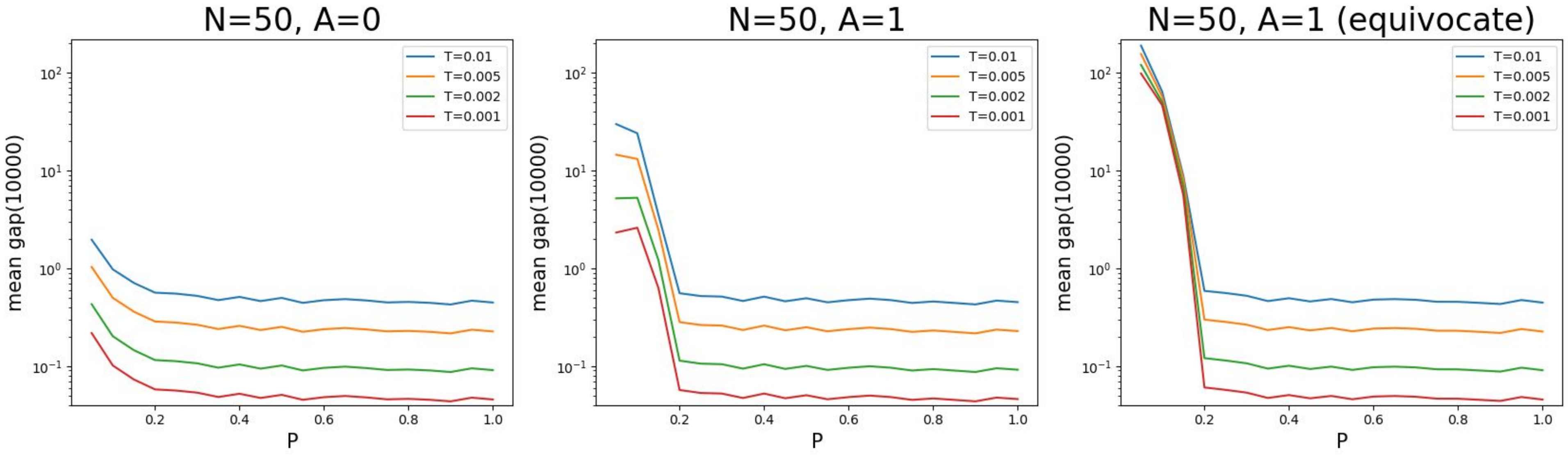


Equivocating Adversary: gap between x_i

- For round k:
  - Let S=sorted([x_i(k) for all i])
  - Define gap(k)=$\max_j(S_{j+1}-S_j)$

Equivocating Adversary: gap between x_i

Equivocation increases mean gap(10000), but only for low P



Conclusion

- Higher P → better convergence
- Normal adversary makes all agents' x_i skew high
  - Higher A → higher x_i
- Equivocating adversary separates agents' x_i only for low P

Future Steps

- Advanced adversary
  - Ex. splitting honest nodes into better groups to equivocate between
- More robust DGD
  - Ex. weighted/adaptively trimmed mean, decaying step size
- Asymptotics of solution error |sd(k)| w.r.t. N, P, A, k
- Multidimensional (nonconvex) functions

Acknowledgments

I would like to thank:

- Jun Wan and Hanshen Xiao for their mentorship
- MIT PRIMES Computer Science for making this project possible
- My parents for supporting me
- You for listening