



IEIIT-CNR



# Randomized Algorithms for Systems and Control: From Monte Carlo to Las Vegas

Roberto Tempo

IEIIT-CNR

Politecnico di Torino

[tempo@polito.it](mailto:tempo@polito.it)



IEIIT-CNR

# Monte Carlo



The journey begins in Monte Carlo...





IEIIT-CNR

# Istanbul

... we make a detour in Istanbul (Byzantium)...





IEIIT-CNR

# Las Vegas



... but our final destination is Las Vegas







IEIIT-CNR



# A Success Story: Randomization in Computer Science



# Randomized Algorithms



- Randomized Algorithms (RAs) are successfully used in various areas, and computer science in particular ...  
... but in systems and control their use is often limited to brute-force Monte Carlo simulations
- Remarkably, randomized methods are not used *systematically* in systems and control
- Development of mathematically rigorous methods, not straightforward use of Monte Carlo simulations



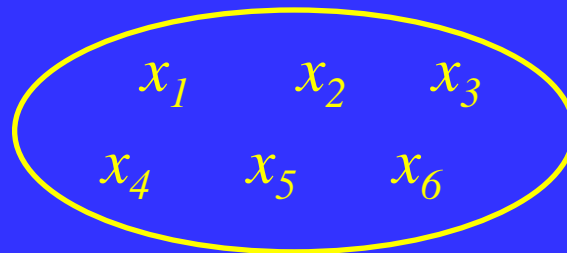


# A Success Story in CS



- Problem: Sorting  $N$  real numbers
- Algorithm: RandQuickSort (RQS)
- RQS is implemented in a C library of *Linux* for sorting numbers
- Sorting Problem

given  $N$  real  
numbers



$S_1$

sort them in  
increasing order

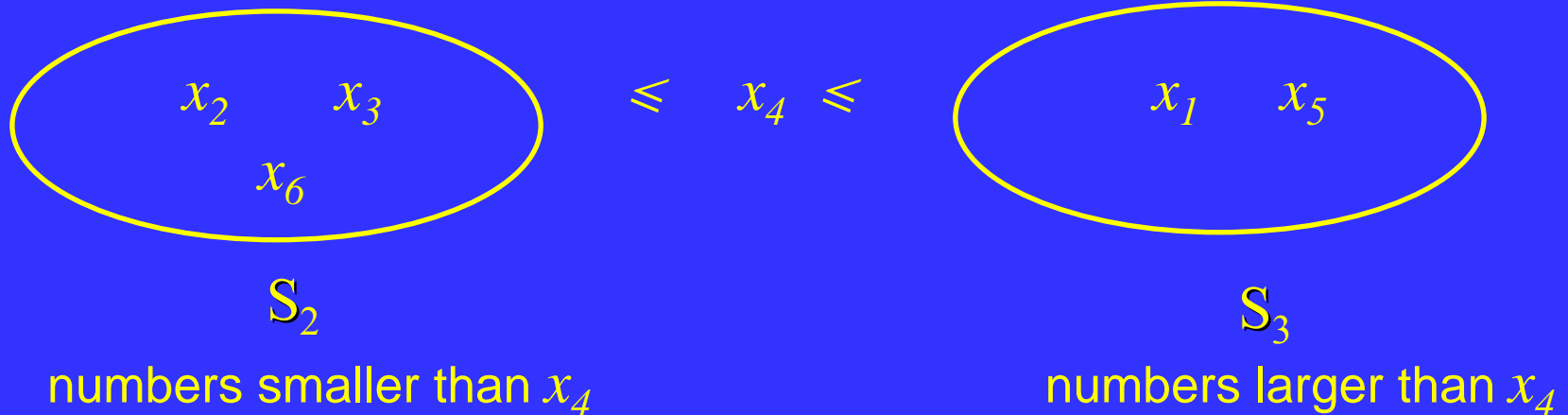


# RandQuickSort (RQS)



RQS is a recursive algorithm consisting of two phases

1. randomly select a number  $x_i$  (e.g.  $x_4$ )
2. deterministic comparisons between  $x_i$  and other  $(N-1)$  numbers





# Running Time of RQS



- Because of randomization, running time may be different from one run of the algorithm to the next one
- RQS is very fast: Average running time is  $O(N \log (N))$
- This is a major improvement compared to brute force approach (e.g. when  $N = 2^M$ )
- Average running time is *highly probable*...  
... we will return to this later



IEIIT-CNR



# Monte Carlo and Las Vegas Algorithms





# Randomized Algorithm: Definition



- **Randomized Algorithm (RA):** An algorithm that makes random choices during its execution to produce a result



IEIIT-CNR

# Randomized Algorithm: Definition

- **Randomized Algorithm (RA):** An algorithm that makes random choices during its execution to produce a result
- Example of a “random choice” is a coin toss

*heads*



or

*tails*





# Randomized Algorithm: Definition

- **Randomized Algorithm (RA):** An algorithm that makes random choices during its execution to produce a result
- For hybrid systems, “random choices” could be switching between different states or logical operations
- For uncertain systems, “random choices” require (vector or matrix) random sample generation



IEIIT-CNR

# Monte Carlo Randomized Algorithm

- Monte Carlo Randomized Algorithm (MCRA): A randomized algorithm that may produce incorrect results, but with bounded error probability





IEIIT-CNR

# Las Vegas Randomized Algorithm



- Las Vegas Randomized Algorithm (LVRA): A randomized algorithm that always produces correct results, the only variation from one run to another is the running time





IEIIT-CNR



# Randomization of Uncertain Systems



# Randomization of Uncertain Systems

- Consider random uncertainty  $\Delta$ , associated pdf and bounding set  $\mathcal{B}$
- $\Delta$  is a (real or complex) random vector (parametric uncertainty) or matrix (nonparametric uncertainty)
- Consider a performance function

$$J(\Delta): \mathcal{B} \rightarrow \mathbf{R}$$

and level  $\gamma > 0$

- Define worst case and average performance

$$J_{max} = \max_{\Delta \in \mathcal{B}} J(\Delta) \qquad J_{ave} = E_{\Delta}(J(\Delta))$$



# Example - $\mathcal{H}_\infty$ Performance



## ■ $\mathcal{H}_\infty$ performance of sensitivity function

$$\mathcal{B} = \{\Delta: \Delta = \text{bdiag}(\Delta_1, \dots, \Delta_q) \in \mathbf{F}^{n,m}, \sigma_{\max}(\Delta) \leq \rho\}$$

$$S(s, \Delta) = 1 / (1 + P(s, \Delta) C(s))$$

$$J(\Delta) = \|S(s, \Delta)\|_\infty$$

## ■ Objective: Check if

$$J_{\max} \leq \gamma \quad \text{and} \quad J_{\text{ave}} \leq \gamma$$

## ■ These are uncertain decision problems



# Two Problem Instances



- We have two problem instances for worst case performance

$$J_{max} \leq \gamma \quad \text{and} \quad J_{max} > \gamma$$

and two problem instances for average case performance

$$J_{ave} \leq \gamma \quad \text{and} \quad J_{ave} > \gamma$$

- This leads to one-sided and two-sided MCRA



- One-sided MCRA: Always provide a correct solution in one of the instances (they may provide a wrong solution in the other instance)
- Consider the empirical maximum

$$\hat{J}_{max} = \max_{i=1, \dots, N} J(\Delta^i)$$

where  $\Delta^i$  are random samples and  $N$  is the sample size

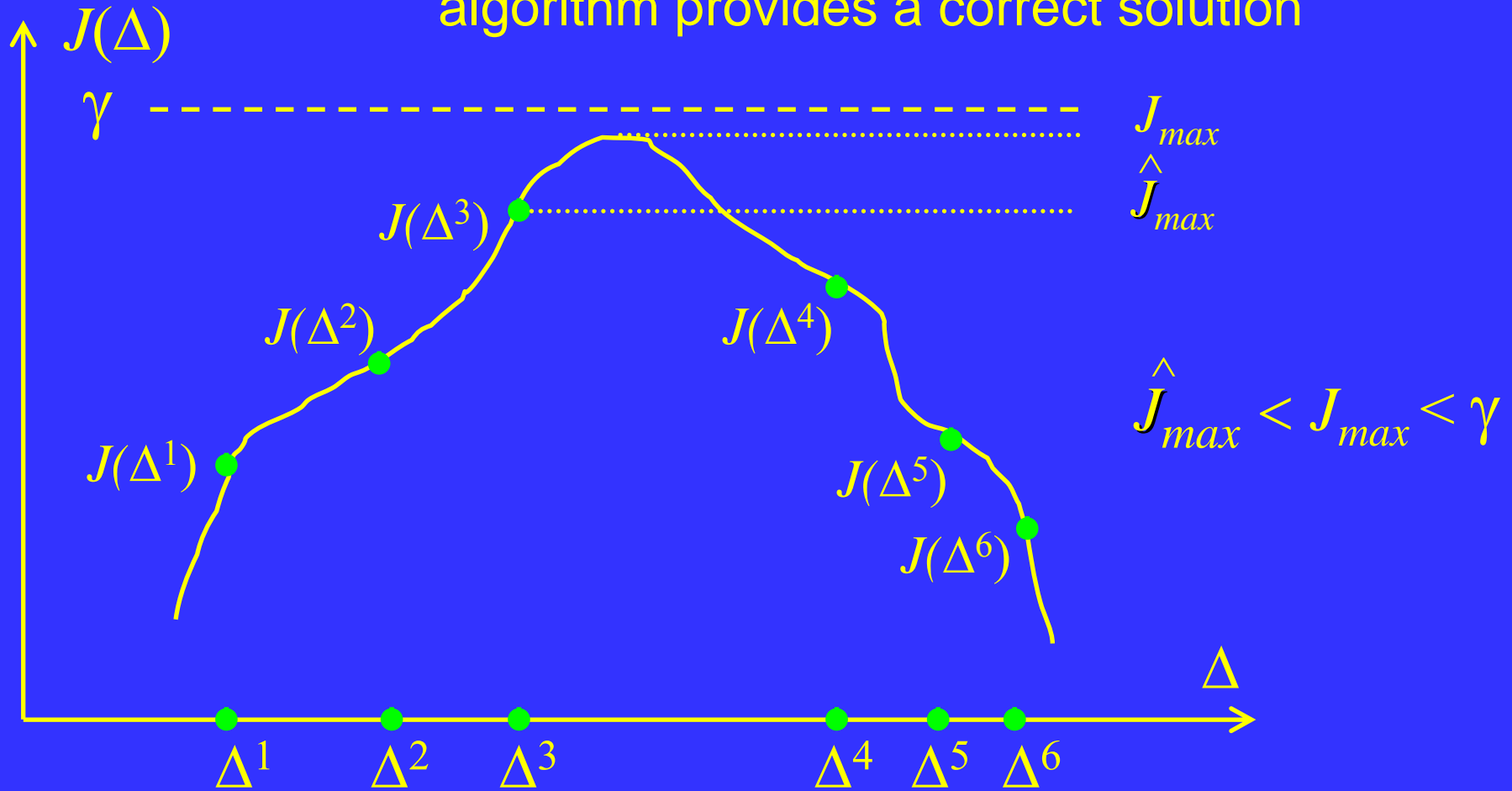
- Check if  $\hat{J}_{max} \leq \gamma$  or  $\hat{J}_{max} > \gamma$



# One-Sided MCRA: Case 1



algorithm provides a correct solution

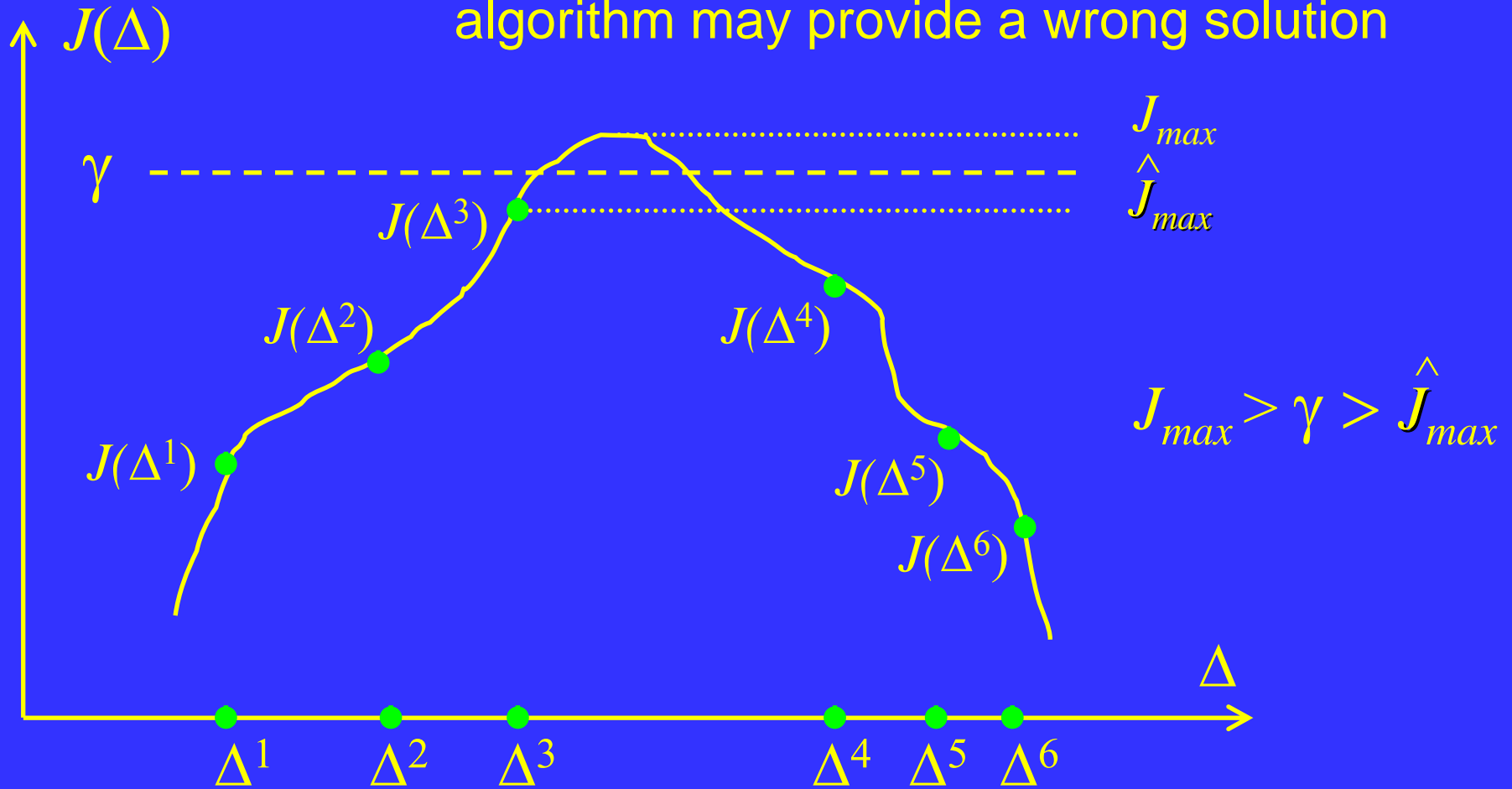




# One-Sided MCRA: Case 2



algorithm may provide a wrong solution





- Two-sided MCRA: They may provide a wrong solution in both instances
- Consider the empirical average

$$\hat{J}_{ave} = \text{ave}_{i=1, \dots, N} J(\Delta^i)$$

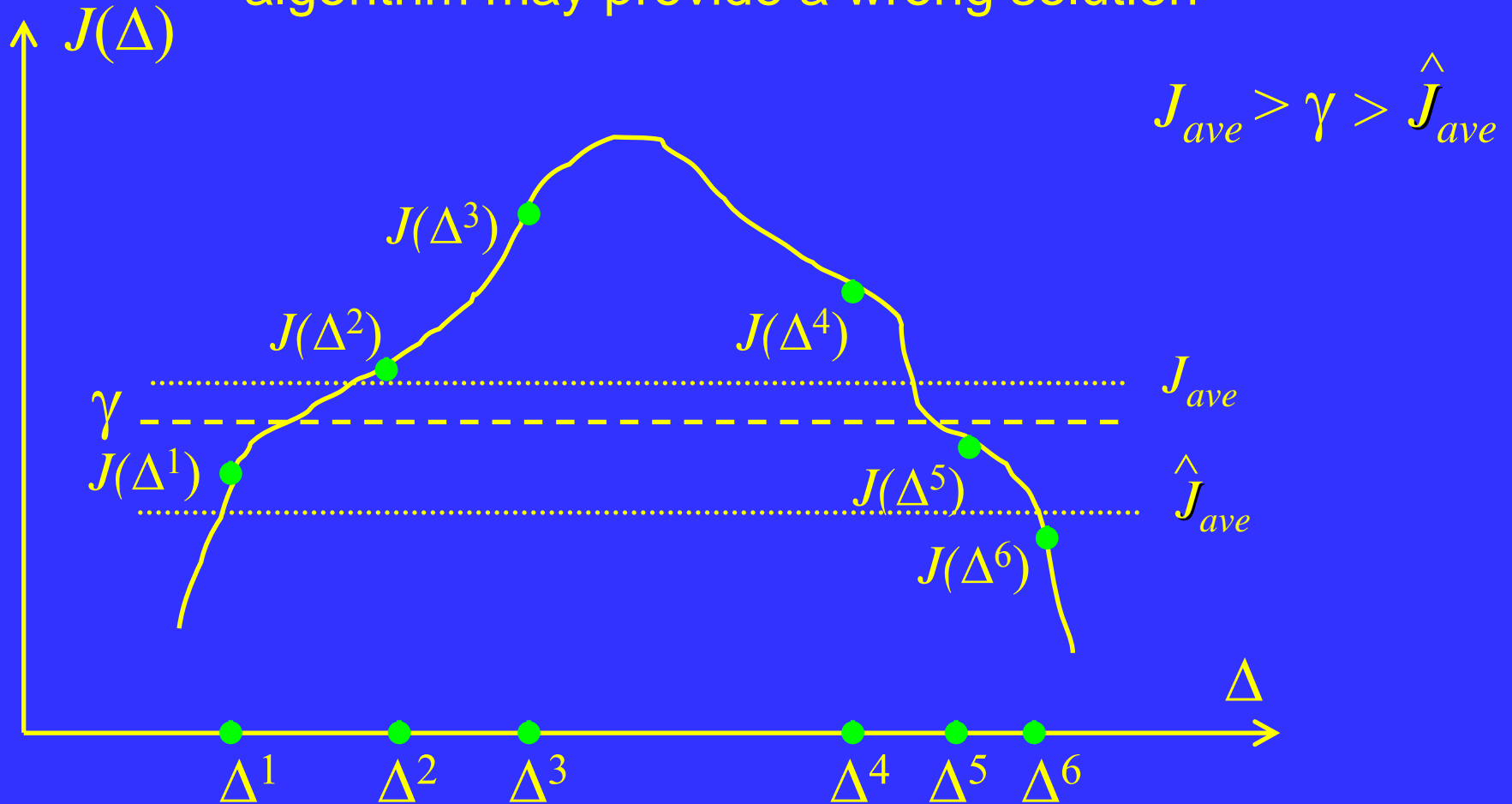
where  $\Delta^i$  are random samples and  $N$  is the sample size

- Check if  $\hat{J}_{ave} \leq \gamma$  or  $\hat{J}_{ave} > \gamma$



# Two-Sided MCRA: Case 1

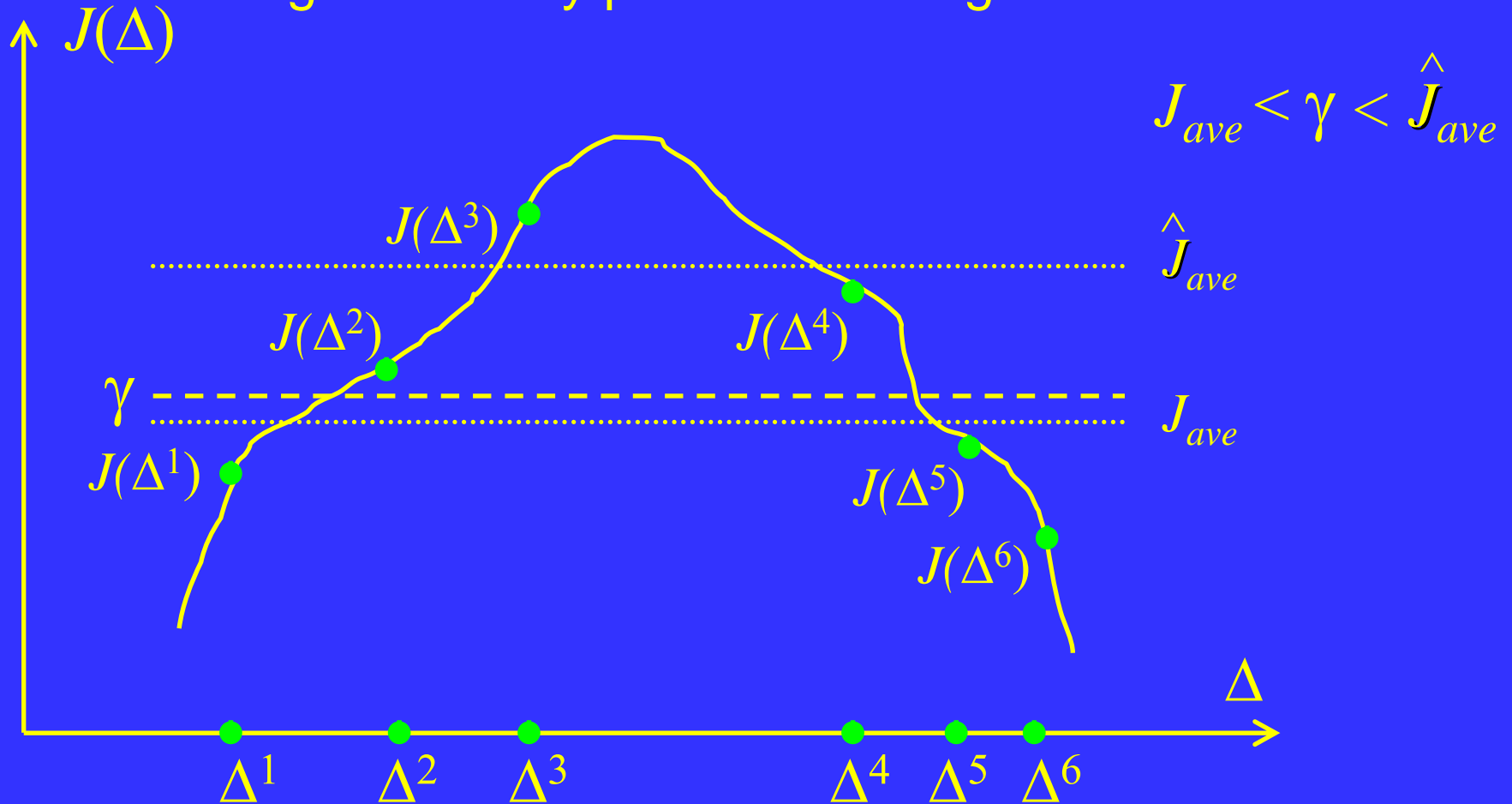
algorithm may provide a wrong solution





# Two-Sided MCRA: Case 2

algorithm may provide a wrong solution





# Las Vegas Randomized Algorithms

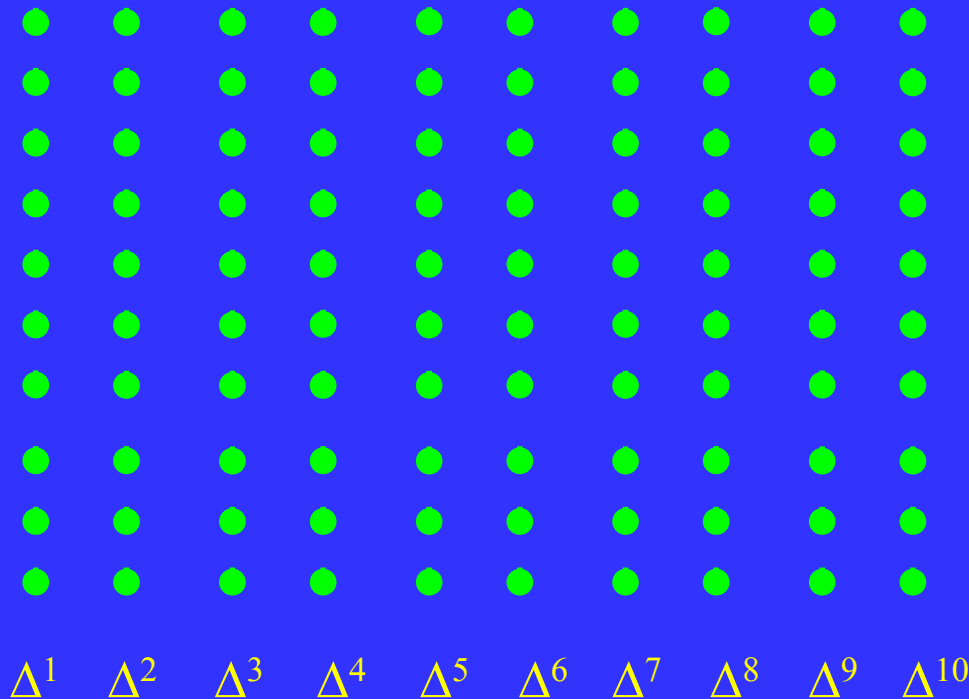
- We also have zero-sided (Las Vegas) randomized algorithms
- Las Vegas Randomized Algorithm (LVRA): Always give the correct solution
- The solution obtained with a LVRA is probabilistic, so “always” means with probability one
- Running time may be different from one run to another
- We can study the *average* running time



# Discrete Bounding Set



Consider random uncertainty  $\Delta$ , a *discrete* bounding set  $\mathcal{B}$ , given pdf and performance function





# The Las Vegas Viewpoint



- Consider discrete random variables
- The sample space is discrete and  $M^N$  possible choices can be made
- In the binary case we have  $2^N$
- Finding maximum requires ordering the  $2^N$  choices
- Las Vegas can be used for ordering real numbers
  
- Example: Revisiting RQS



# RandQuickSort (RQS)

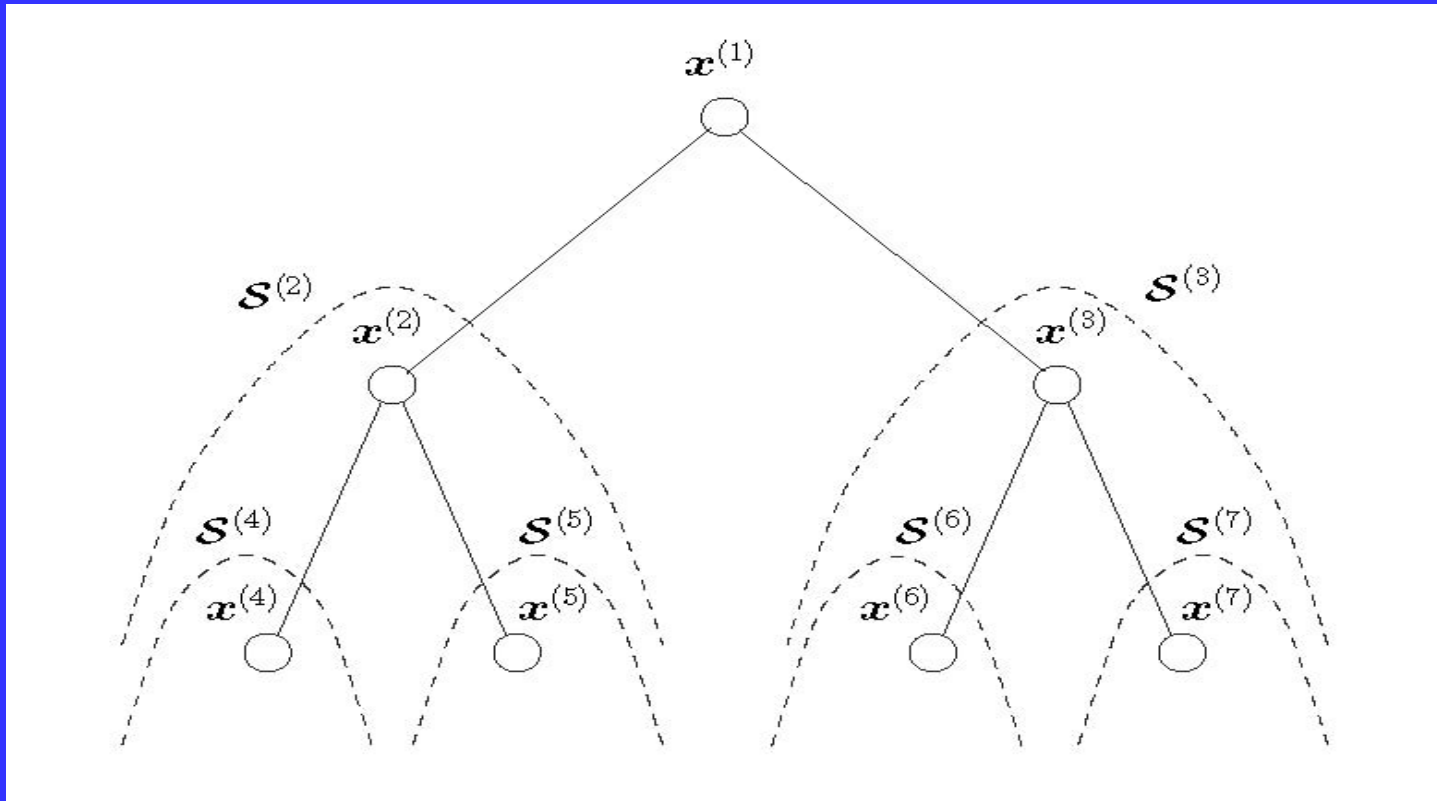


- The idea is to divide the original set  $S_1$  into two sets having (approximately) the same cardinality
- This requires finding the *median* of  $S_1$  (which may be difficult)



# RQS: Binary Tree Structure

We use randomization at each step of the (binary) tree





# RQS: The Las Vegas Viewpoint



- Average running time is  $O(N \log(N))$
- This running time holds for every input
- The average running time holds with probability at least  $1-1/N$
- Hint: Use the so-called Chernoff bound to prove this
- Improvements for RQS to avoid achieving the worst case running time  $O(N^2)$



# Find Algorithm



- Find Algorithm: Find the  $k$ -th smallest number in a set
- Basically it is a RQS but it terminates when the number is found
- Average running time of Find is  $O(N)$



# Complexity Relaxation



- If  $N$  is too large (e.g. when  $N=2^M$ ), we may want to consider only a subset of  $K$  samples out of  $N$
- This leads to a one-sided Monte Carlo which gives a suboptimal, but more efficient, solution
- Close connections with *Ordinal Optimization*<sup>[1]</sup> having the objective not to find the maximum value, but the value which is within the top  $N$ -th percent (for given  $N$ )
- **Conclusion:** Ordering between elements is easier than finding their values

[1] Y.C. Ho, R. Sreenivas, P. Vakili (1992)



# Continuous versus Discrete Sample Space



- The underlying problem may be continuous or discrete
- For *Lyapunov stability* the original problem is continuous, but it is equivalent to another discrete problem
- For *consensus* the original problem is discrete (binary), e.g. Byzantine Agreement



IEIIT-CNR



# Lyapunov Stability Analysis and Synthesis



# Uncertain Systems (State Space)



- Consider the uncertain system

$$\dot{x}(t) = A(\Delta) x(t) + B u(t)$$

and state feedback

$$u(t) = K x(t)$$

with  $x(0) = x_0$ ,  $x \in \mathbf{R}^n$ ,  $u \in \mathbf{R}^m$ ,  $A(\Delta) \in \mathcal{A}$



# Interval and Vertex Matrices

- Uncertain matrix family  $\mathcal{A}$  is interval
- That is,  $a_{ik}$  ranges in the interval for all  $i, k$

$$|a_{ik} - a_{ik}^*| \leq w_{ik}$$

where  $a_{ik}^*$  are nominal values and  $w_{ik}$  are weights

- Define the  $N = 2^{n^2}$  vertex matrices  $A^1, A^2, \dots, A^N$

$$a_{ik} = a_{ik}^* + w_{ik} \quad \text{or} \quad a_{ik} = a_{ik}^* - w_{ik}$$

for all  $i, k = 1, 2, \dots, n$



# Common Lyapunov Functions

- Given matrices  $A^*$ ,  $W$  and feedback  $K$ , find a *common quadratic Lyapunov function*  $P > 0$  for the system

$$\dot{x}(t) = (A + B K) x(t) \quad \text{for all } A \in \mathcal{A}$$

- Find  $P > 0$  such that

$$\mathcal{L}(P, A) = (A+BK)^T P + P (A+BK) < 0 \quad \text{for all } A \in \mathcal{A}$$

- Equivalently, find  $P > 0$  such that

$$J(P, A) = \lambda_{\max} \mathcal{L}(P, A) < 0 \quad \text{for all } A \in \mathcal{A}$$



# Lyapunov Stability of Interval Systems

- Quadratic Lyapunov stability analysis and synthesis of interval systems are NP-hard problems
- In principle, they can be solved in one-shot with convex optimization, but the number of constraints is exponential
- We can use relaxation (e.g.  $\pi/2$  Theorem<sup>[1]</sup>) or randomization

[1] Yu. Nesterov (1997)



# Randomization Approach

- Various randomized algorithms have been recently proposed for finding a probability one common solution  $P \succ 0$  of the Lyapunov equation<sup>[1]</sup>

$$\mathcal{L}(P, A) = (A+BK)^T P + P (A+BK) < 0 \quad \text{for all } A \in \mathcal{A}$$

- Efficient randomized algorithms are based on gradient iterations or ellipsoid method

[1] B. Polyak, R. Tempo (2001)



# Las Vegas Randomized Algorithm

- Due to convexity, it suffices to study  $\mathcal{L}(P, A) < 0$  for all vertex matrices<sup>[1]</sup>
- We need to perform randomization of the  $N = 2^{n^2}$  vertices (in the worst case)
- If we select the vertices in random order according to a given pdf, we have a LVRA
- Question: Do we really need to check all the vertex matrices ( $N = 2^{n^2}$ )?

[1] H.P. Horisberger, P.R. Belanger (1976)



- **Answer:** It suffices to check “only” a subset of  $2^{2n}$  vertex matrices<sup>[1]</sup>
- This is still exponential (the problem is NP-hard), but it leads to a major computational improvement for medium size problems (e.g.  $n = 8$  or  $10$ )
- For example, for  $n=8$ ,  $N$  is of the order  $10^5$  (instead of  $10^{19}$ )

[1] T. Alamo, R. Tempo, D. Rodriguez, E.F. Camacho (2007)



# Diagonal Matrices and Generalizations

- Transform the original problem from full square matrices  $A$  to diagonal matrices  $Z \in \mathbf{R}^{2n,2n}$
- It suffices to check the vertices of  $Z$
- Extensions for  $\mathcal{L}_2$ -gain minimization and other related LMI problems
- Generalizations for multiaffine interval systems



IEIIT-CNR



# Probabilistic Sorting of Switched Systems



# Sorting of Switched Systems



- Consider Lyapunov equations

$$\mathcal{L}(P, A) = (A^i)^T P + P A^i \quad \text{for all } i = 1, 2, \dots, N$$

- The objective is to sort these  $N$  Lyapunov equations according to their degree of stability (decay rate) using a common  $P > 0$  previously computed
- Motivations: Deciding which systems are more stable than others is useful information for the controller



# LVRA for Matrix Sorting

- The sorting operation should be performed quickly because we are switching between  $N = 2^{2n}$  systems
- This requires finding a LVRA which provides a *matrix* sorting for the  $N$  equations  $\mathcal{L}(P)$
- Matrix version of RandQuickSort is developed<sup>[1]</sup>
- **Technical difficulty:** The equations may be not completely sortable because of sign indefiniteness

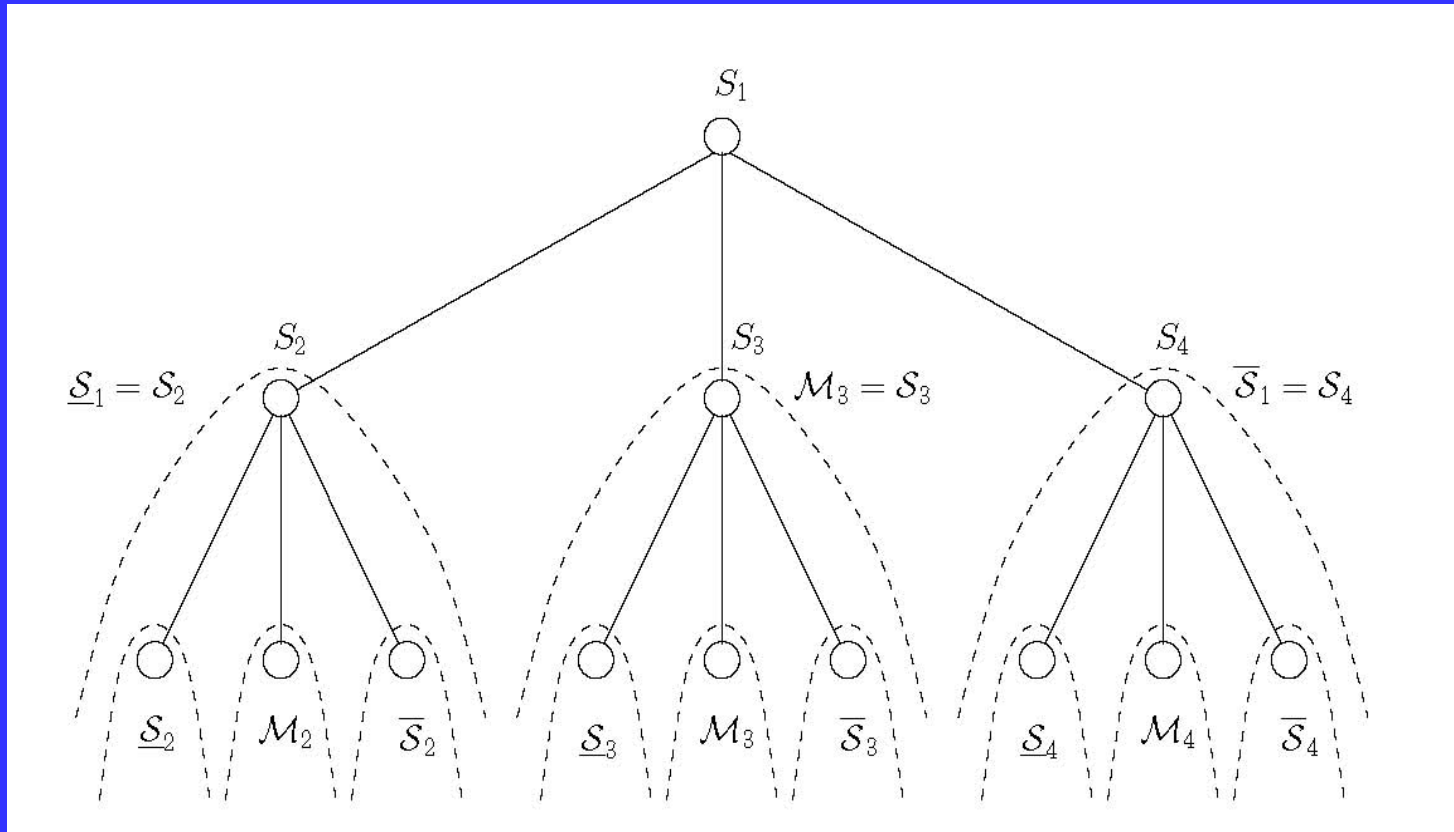
[1] H. Ishii, R. Tempo (2007)





# RQS for Matrices: Trinary Tree

We use randomization at each step of the (trinary) tree





# RQS for Matrices: Results



- If the Lyapunov equations are completely sortable, then the expected running time is (the same of RQS)  
 $O(N \log (N))$
- If the Lyapunov equations are not completely sortable, then additional comparisons should be performed
- The worst case number of additional comparisons is  
 $N(N-1)/2$



IEIIT-CNR



# Distributed Average Consensus



# Distributed Average Consensus Problems



- **General problem:** Consider a set of  $N$  agents each having a numerical value
- This numerical value is communicated to the neighboring agents iteratively
- **Consensus:** All agents eventually reach a common value (average of the initial value)
- **Applications:** Sensor networks, load balancing, multi-vehicle coordination, UAVs, ...

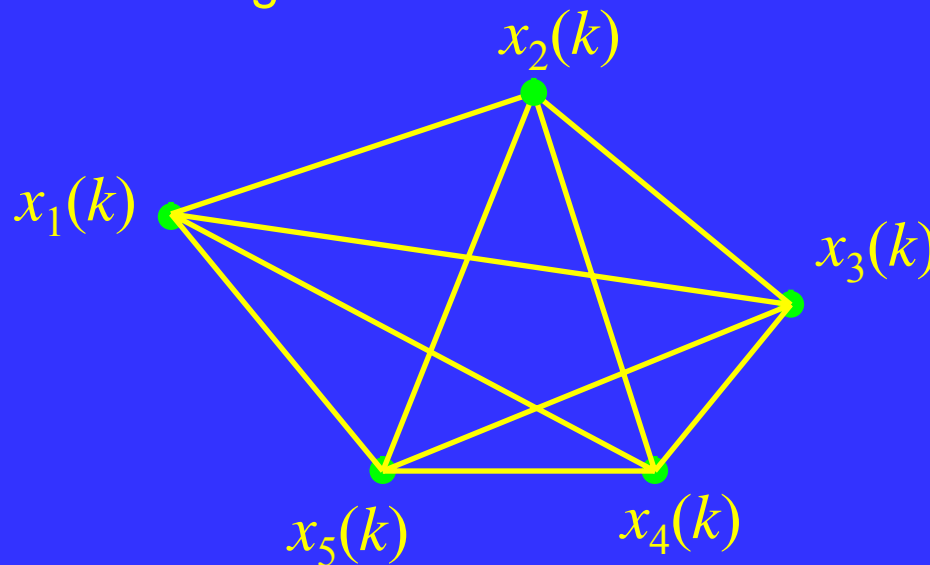


# Graph Formulation

Consider a network of  $N$  nodes and the graph  $(\mathcal{V}, \mathcal{E})$

$\mathcal{E}$  is the set of edges

$\mathcal{V}$  is the set of nodes



graph is undirected  
and connected  
(no directions, no  
self-loops, no  
multiple edges)

at time  $k$  each node  $i$  has a scalar value  $x_i(k)$ ; initial value is  $x_i(0)$



# Distributed Average Consensus Problems



- **Objective:** Derive a randomized algorithm (MC or LV) such that
  1. Nodes update the values  $x_i(k)$  using neighbor information
  2. The values of the nodes converge to the average of initial values  $x_i(0)$
- Three different cases depending on the range of node values: Real, integer (quantized) and binary values



# Real/Quantized/Binary Agent Values

- Real and quantized case are very popular<sup>[1]</sup>
- Monte Carlo (MCMC Markov Chain Monte Carlo) algorithms are developed in these cases
- The binary value case is not really studied in the systems and control community...
- Paradigm: Byzantine Agreement Problem

[1] P.J. Antsaklis, J. Baillieul (2007)



IEIIT-CNR



# Byzantine Agreement Problem (ByzAgr)

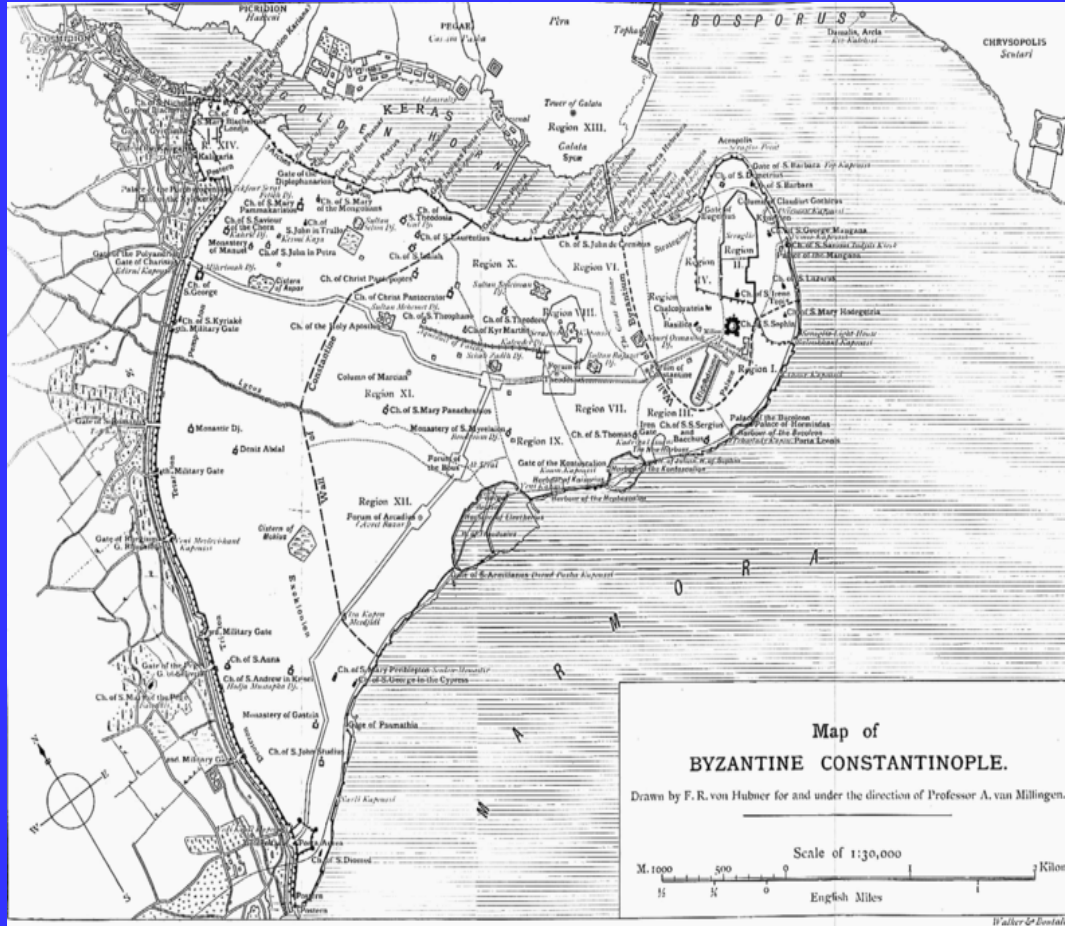


# Byzantium, 1453 AD

- **Some history:** In the year 1453 AD, the city of Byzantium is under siege... powerful Ottoman battalions are camped around the city on both sides of the Bosphorus, poised to launch the final attack...
- The generals, located in several camps, are trying to reach an agreement... they can communicate thanks to the messenger service of the Ottoman Army...
- Messages can be delivered certifying the identity of the sender and preserving its content...



# Byzantium and the Bosphorus





# Loyal and Traitor Generals



- Loyal generals are trying to reach an agreement on when they should launch the final attack, but...
- Traitor generals are secretly conspiring... their aim is to confuse the loyal generals so that an insufficient number of generals is deceived into attacking...



# Byzantine Agreement Problem

(ByzAgr)<sup>[1]</sup>



- Find a distributed consensus algorithm satisfying the following conditions:

1. *Non-triviality*: If all generals have the same input, all loyal generals take a decision equal to the input
2. *Agreement*: All loyal generals should agree on the decision
3. *Limited dithering*: Eventually all loyal generals should come to a decision

[1] L. Lamport, R. Shostak, M. Pease (1982)



# Agents and Communication Mechanism



- *Synchronous*: The agents have synchronized clocks
- *Reliable*: A message is guaranteed to be delivered
- *Authenticated*: Identity of the sender is known
- *Faulty agents*: Hard to identify because they behave maliciously instead of simply crashing; they may collude to maximize damage
- *Point-to-point communication*: Underlying topology is that of a undirected and connected graph





# Distributed Agreement Protocol



- Distributed agreement protocol proceeds in a sequence of *rounds*
- At each round each agent sends a binary message (called *vote*, e.g. attack or retrieve) to the other agents
- Non-faulty agents send the same vote to all the other agents; faulty agents may send different votes to different agents
- Round is concluded when all agents receive a vote



# Randomized Algorithm for ByzAgr



- First randomized algorithm is due to Rabin<sup>[1]</sup>
- There is a global coin toss performed by a trusted party consisting of

*heads*



or



*tails*

- The result of the global coin toss is correctly transmitted to all agents
- All agents equally contribute to the decision
- Majority vote: Count the number of votes received

[1] M.O. Rabin (1983)



# Rabin's Randomized Algorithm

$N_f < N/8$ ;  $N$  is a multiple of 8;  $L = (5N/8) + 1$ ;  $H = (3N/4) + 1$ ;  $G = 7N/8$

input: vote, output: decision  $y_i$ , for each round do

1. broadcast vote, receive votes from all other agents
2. set  $m_i(k) \leftarrow$  majority value (0, 1) received
3. set  $t_i(k) \leftarrow$  number of occurrences of  $m_i(k)$
4. if coin = heads, then set  $\bar{t}(k) \leftarrow L$ , else  $\bar{t}(k) \leftarrow H$
5. if  $t_i(k) \geq \bar{t}(k)$ , then set vote  $\leftarrow m_i(k)$ , else vote  $\leftarrow 0$
6. if  $t_i(k) \geq G$ , then set  $y_i \leftarrow m_i(k)$





- Rabin's algorithm always (probabilistically) gives a correct output
- Number of rounds is a random variable because the algorithm is based on randomization
- The expected number of steps to reach consensus is a constant
- This algorithm is a LVRA



# Randomized vs Deterministic Algorithms for ByzAgr



- Deterministic algorithms requires  $N_f$  steps

- Impossibility Theorem<sup>[1]</sup>

If there is no synchronized clock for all agents and if the processing speed of the agents is different, then no deterministic algorithm can achieve consensus (even in the presence of a single faulty agent)

[1] M.J. Fisher, N.A. Lynch, M.S. Paterson (1985)



- We have defined Monte Carlo (one-sided and two-sided) and Las Vegas Randomized Algorithms
- We have shown applications of these algorithms for continuous and discrete problems
  1. Lyapunov stability
  2. Consensus problems (Byzantine agreement)





- “*Randomized Algorithms for Analysis and Control of Uncertain Systems*” by RT et al, Springer, 2005
- EJC paper by RT and Hideaki Ishii, 2007
- RACT (Randomized Algorithms Control Toolbox) by Andrey Tremba et al, 2007
- For more details, please visit  
<http://staff.polito.it/roberto.tempo/>



IEIIT-CNR

# European Control Conference in Kos



This is the end of the journey...

