# COMP36111: Advanced Algorithms I

# Part 3b: Randomized algorithms - a brief introduction

Howard Barringer

Room KB2.20: email: howard.barringer@manchester.ac.uk

December 2010

# Outline

# To start ...

- Randomised algorithms employ some form of random element in an attempt to obtain improved performance (over worst case performance of deterministic algorithms)
- Sometimes, improvement can be dramatic, from intractable to tractable
- Some loss in reliability of results can occur
- Different runs may produce different results, reliability may be improved by running several times

# To start ...

- Randomised algorithms employ some form of random element in an attempt to obtain improved performance (over worst case performance of deterministic algorithms)
- Sometimes, improvement can be dramatic, from intractable to tractable
- Some loss in reliability of results can occur
- Different runs may produce different results, reliability may be improved by running several times

We will consider two types of randomized algorithms:

Monte Carlo

Las Vegas

# Do you remember Buffon's Needle?

An 18th century approach to approximating $\pi$.

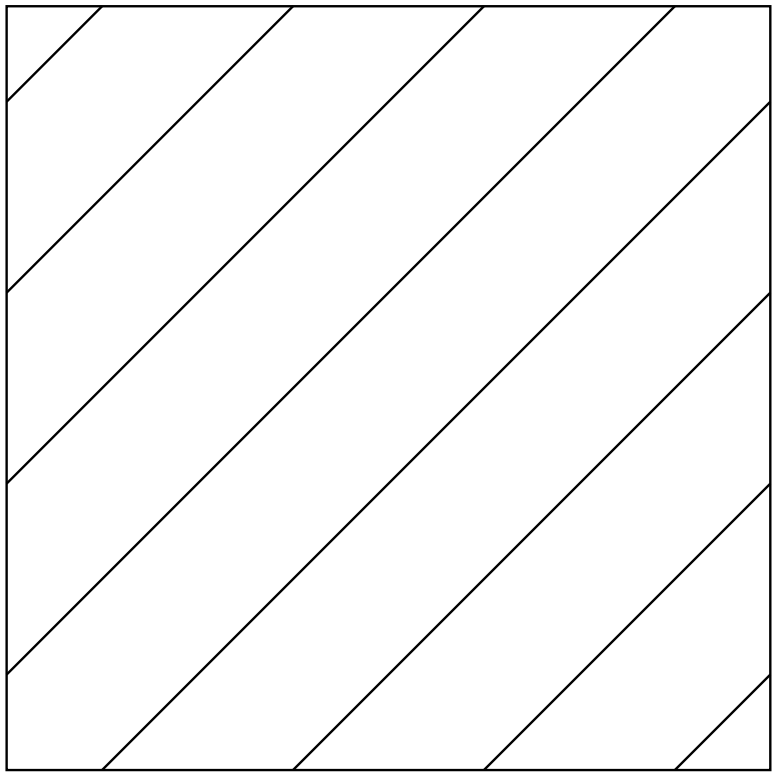
# Do you remember Buffon's Needle?

An 18th century approach to approximating $\pi$.

Drop $N$ needles of unit length randomly (with uniform distribution) over a floor.

Width of floor boards is 2 units.

Some will fall across the boards — the red ones,
some will not — the blue ones.



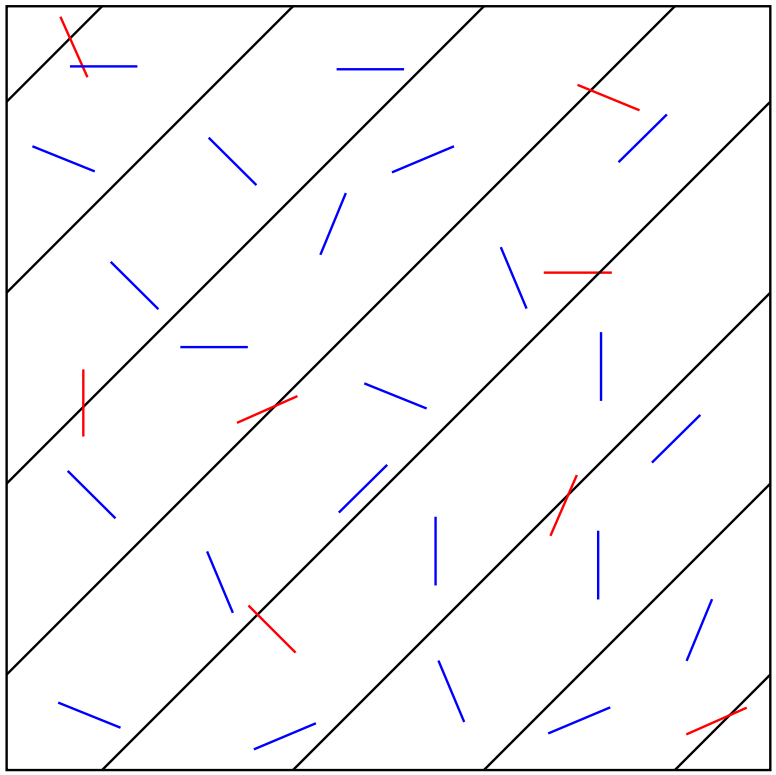
# Do you remember Buffon's Needle?

An 18th century approach to approximating $\pi$.

Drop $N$ needles of unit length randomly (with uniform distribution) over a floor.

Width of floor boards is 2 units.

Some will fall across the boards — the red ones,
some will not — the blue ones.



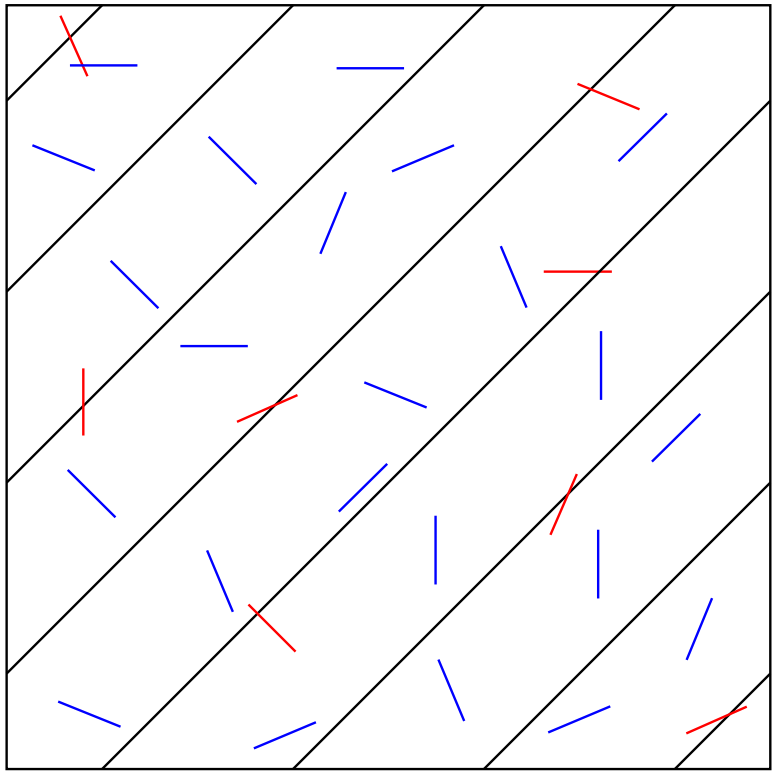
# Do you remember Buffon's Needle?

An 18th century approach to approximating $\pi$.

Drop $N$ needles of unit length randomly (with uniform distribution) over a floor.

Width of floor boards is 2 units.

Some will fall across the boards — the red ones,
some will not — the blue ones.



The ratio $\frac{\text{red}}{N}$ tends to $\frac{1}{\pi}$ as the $N$ tends to $\infty$ .

# Outline

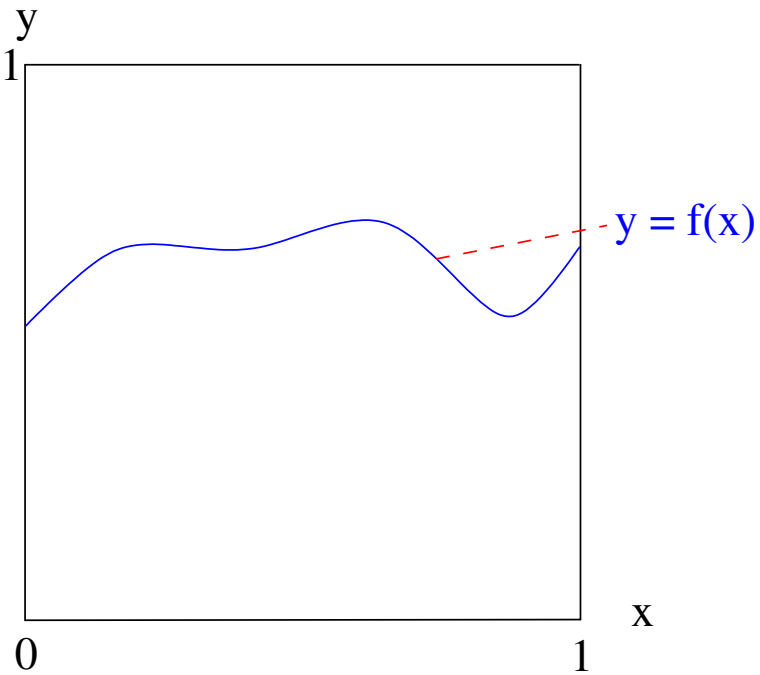
# Monte Carlo algorithms – characteristics

These are randomized algorithms which may produce incorrect results with some small probability, but whose execution time is deterministic .

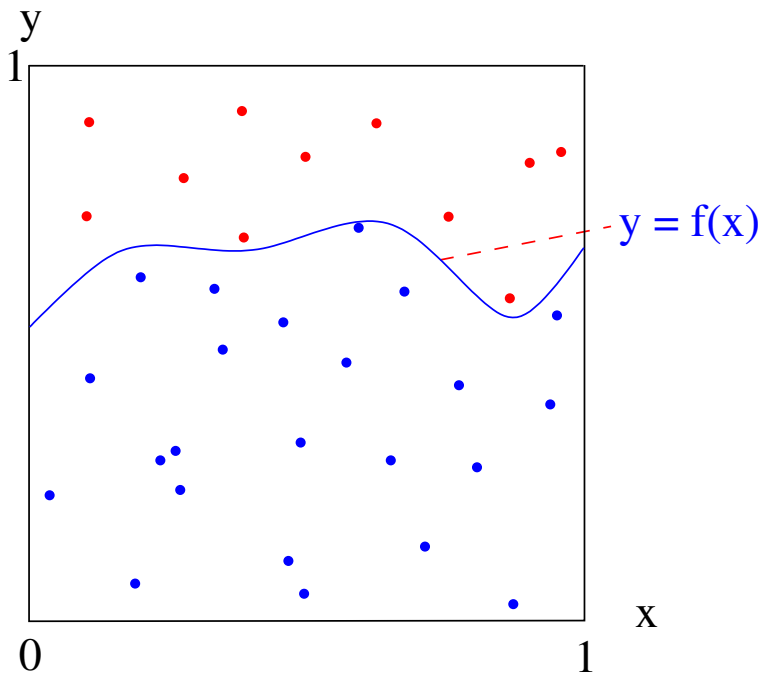
If such an algorithm is run multiple times with independent random choices each time , the failure probability can be made arbitrarily small — at the cost of the running time .

# Monte Carlo numerical integration

Given: for $x \in 0..1$ that $f(x) \in 0..1$



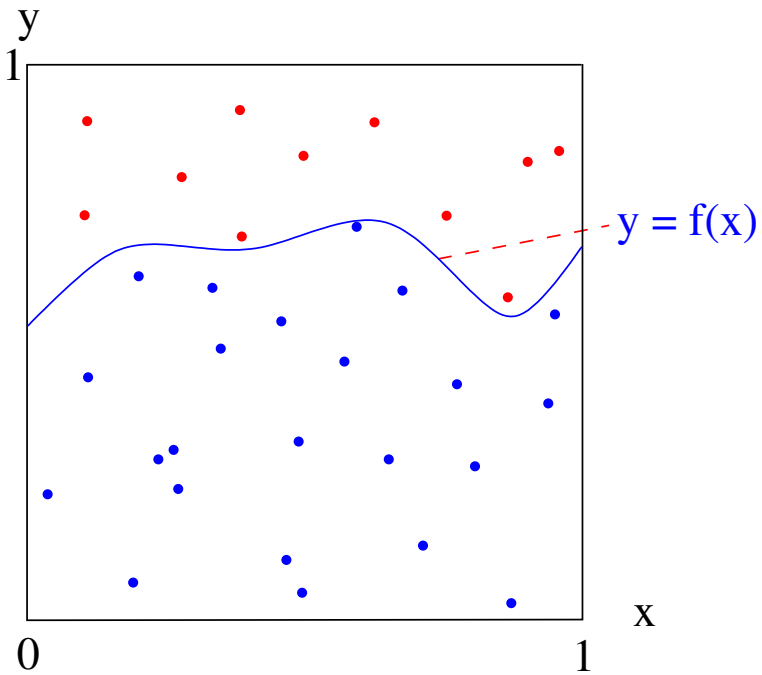
# Monte Carlo numerical integration




Given: for $x \in 0..1$ that $f(x) \in 0..1$

Randomly pick $N$ points in the 0-1 square.

# Monte Carlo numerical integration




Given: for $x \in 0..1$ that $f(x) \in 0..1$

Randomly pick $N$ points in the 0-1 square.

Count the number $k$ of points $(x, y)$ st. $y \leq f(x)$ .

The ratio

$$\frac{k}{N}$$

will approximate

$$\int_0^1 f(x)dx$$

# Randomized integration algorithm

```
integrate(f,N) =
  { k = 0
    for i = 1 to N do
      { x = random(0,1)
        y = random(0,1)
        if y =< f(x) then k += 1}
    return k/N ---- as a real! }
```

# Randomized integration algorithm

```
integrate(f,N) =
  { k = 0
    for i = 1 to N do
      { x = random(0,1)
        y = random(0,1)
        if y =< f(x) then k += 1}
    return k/N ---- as a real! }
```

Repeated runs of this can be used to give increased accuracy of the result.

# Another random approach for integration

As an alternative for $\int_a^b f(x)dx$ , estimate the average height of the curve through random sampling of $x \in a..b$ , then multiply by the integration range $b - a$ .

```
altIntegrate(f,N,a,b) =
  sum = 0
  for i = 1 to N do
      {  x = random(0,1) * (b-a) + a
         sum = sum + f(x) }
  return (sum/N) * (b-a)
```

# Another random approach for integration

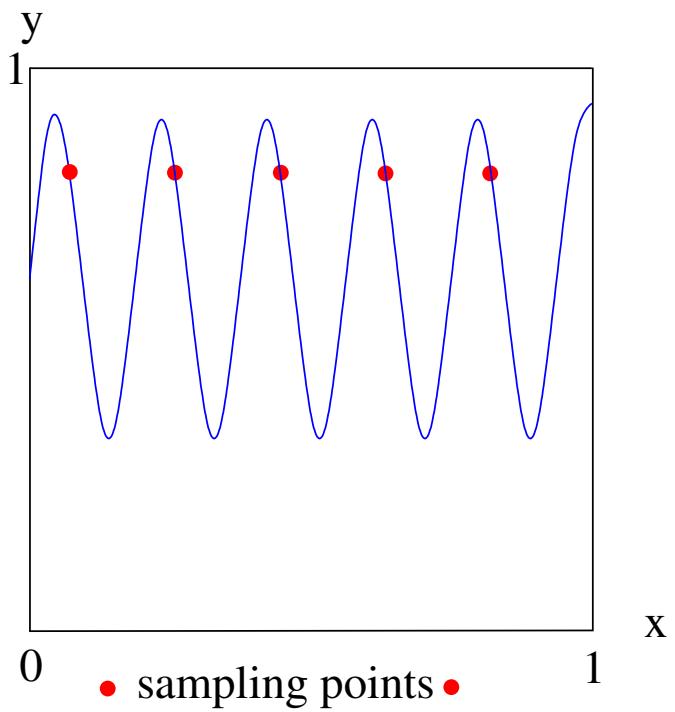
As an alternative for $\int_a^b f(x)dx$ , estimate the average height of the curve through random sampling of $x \in a..b$ , then multiply by the integration range $b - a$ .

```
altIntegrate(f,N,a,b) =
  sum = 0
  for i = 1 to N do
      {  x = random(0,1) * (b-a) + a
         sum = sum + f(x) }
  return (sum/N) * (b-a)
```

Again, a deterministic sampling generally converges faster ...

# But ...

Deterministic sampling algorithms can be fooled through bad sampling.




Which won't occur with the randomized version...

# Note, though ...

In general, the above techniques do not provide fast convergence.

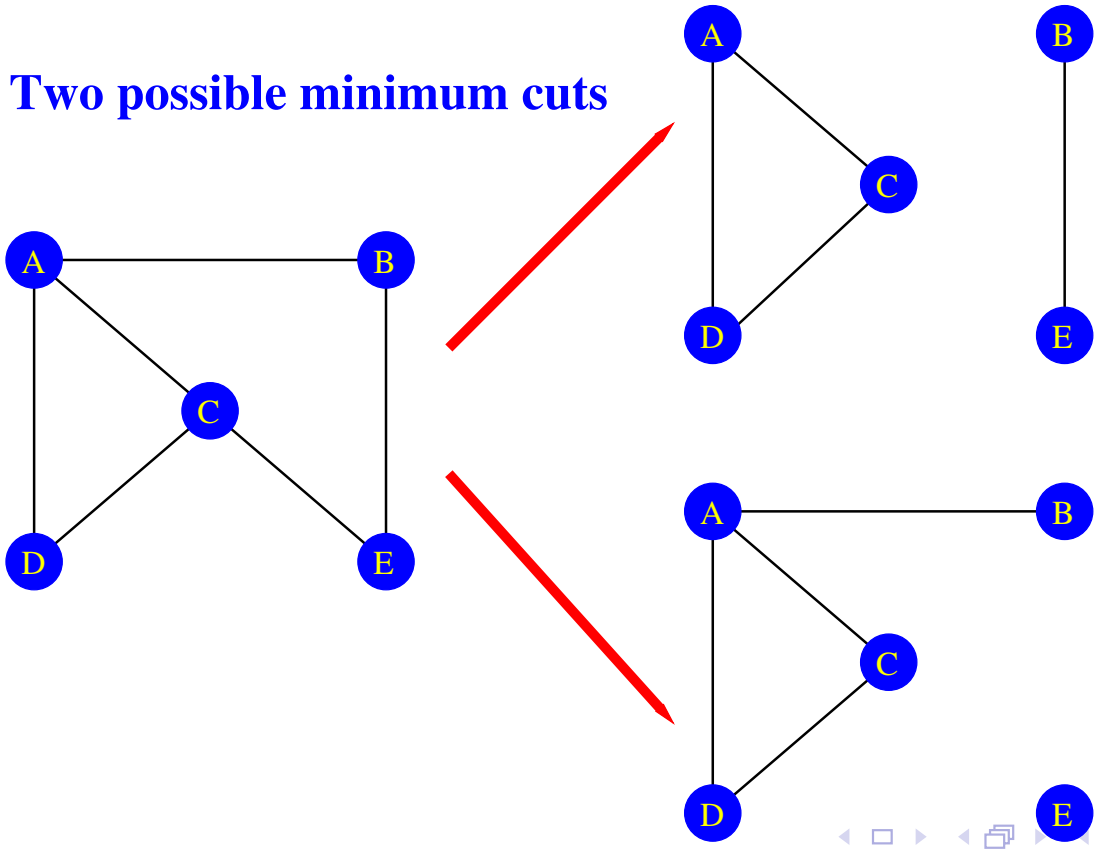
For simple integrals, standard approximation methods, such as Simpson's or the Trapezium method, are better.

For multiple integrals in higher dimensions, randomized methods can be effective.

# Minimum cut of a graph - unweighted

Given a graph $G = (V, E)$ , a cut in $G$ is a set of edges $C \subseteq E$ whose removal from $G$ results in a graph with two or more distinct components.

A minimum cut of $G$ is a cut in $G$ of minimum cardinality.



# A simple randomized algorithm for min-cut

The idea: given a graph $G = (V, E)$ , we randomly contract edges until two vertices remain, then the set of edges between the two vertices is a cut in $G$ .

Repeat the following steps until only two vertices remain.

- choose an edge uniformly at random
- merge the vertices at the end points of the edge
- remove any self loop so introduced

# A simple randomized algorithm for min-cut

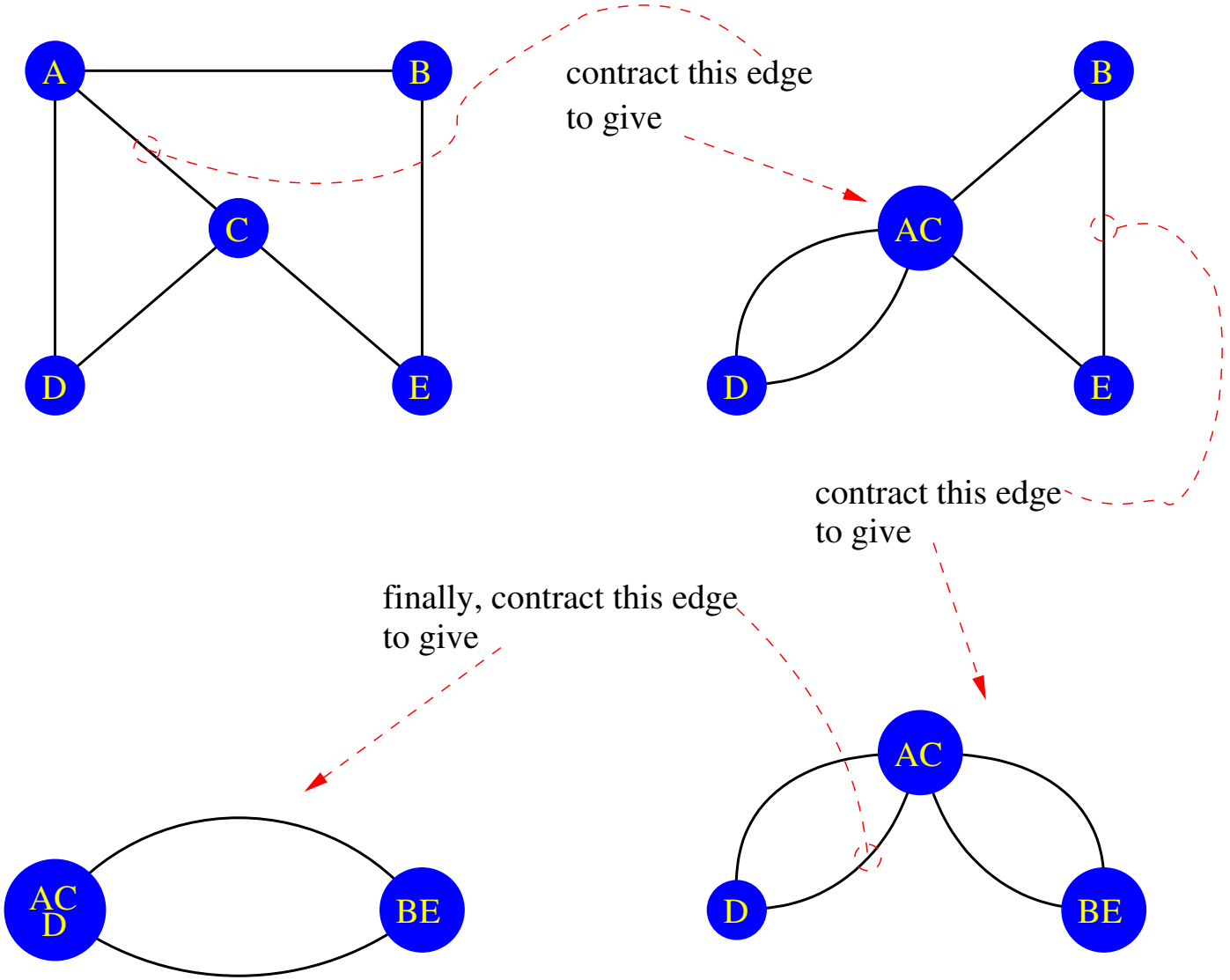
The idea: given a graph $G = (V, E)$ , we randomly contract edges until two vertices remain, then the set of edges between the two vertices is a cut in $G$ .

Repeat the following steps until only two vertices remain.

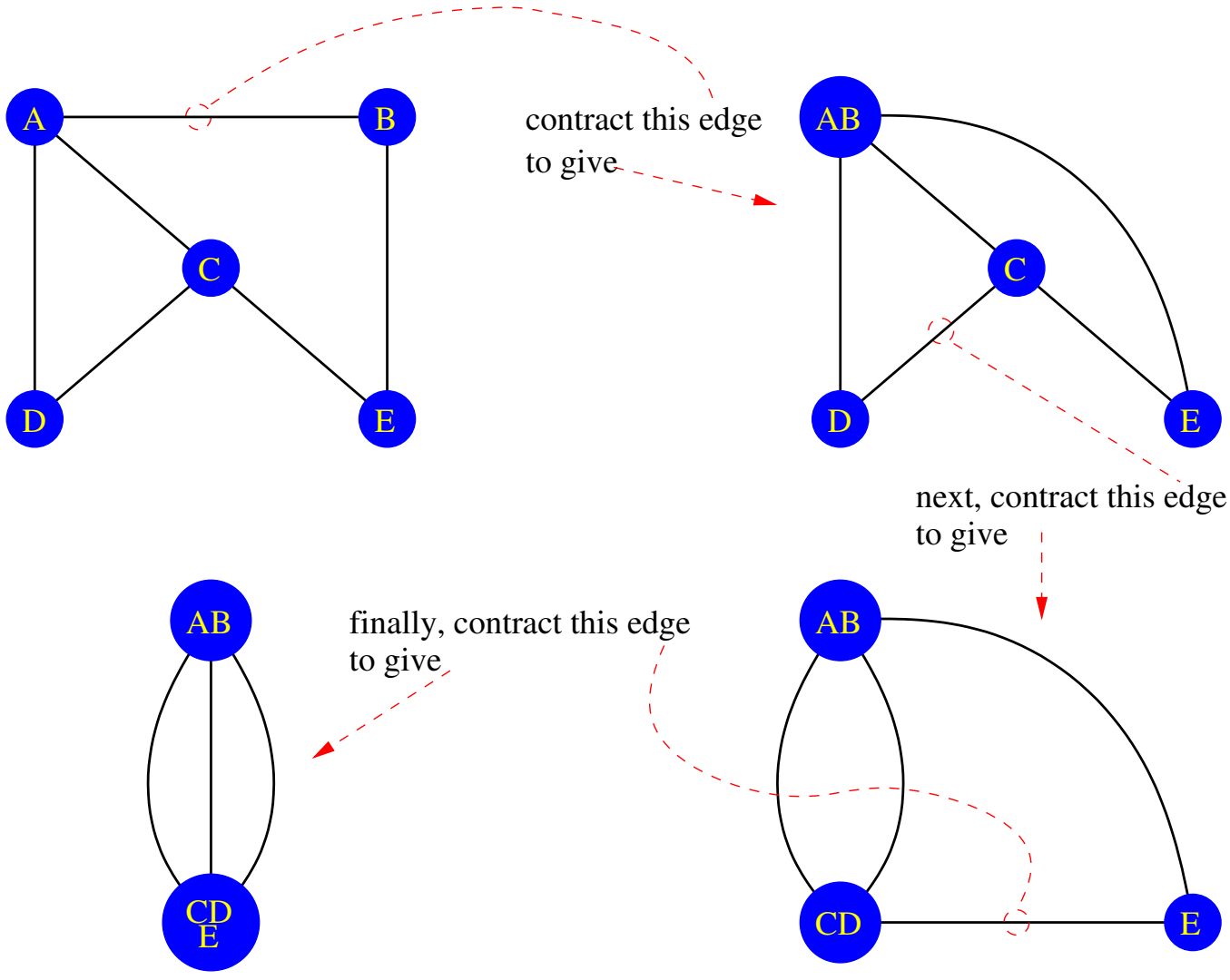
- choose an edge uniformly at random
- merge the vertices at the end points of the edge
- remove any self loop so introduced

Will this algorithm find a min cut?

# Random min-cut example



# Random min-cut example - different choices



# Random min-cut example - analysis

Given a graph with $n$ vertices.

Let $C$ denote a particular min-cut of size $k$ .

Let $\varepsilon_i$ denote the event of not picking an edge of $C$ on the $i^{\text{th}}$ step.

$Pr(\bigcap_{i=1}^{n-2} \varepsilon_i) \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}$ .

Thus, probability of finding a particular min-cut is $\geq \frac{2}{n^2}$ .

We could repeat this algorithm $\frac{n^2}{2}$ times, using independent random choices — the probability that the min-cut is not found in any of $\frac{n^2}{2}$ repetitions reduces to become $< \frac{1}{e}$ .

Further repetition will reduce the probability further.

# Random min-cut example - analysis

Given a graph with $n$ vertices.

Let $C$ denote a particular min-cut of size $k$ .

Let $\varepsilon_i$ denote the event of not picking an edge of $C$ on the $i^{\text{th}}$ step.

$Pr(\bigcap_{i=1}^{n-2} \varepsilon_i) \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}$ .

Thus, probability of finding a particular min-cut is $\geq \frac{2}{n^2}$ .

We could repeat this algorithm $\frac{n^2}{2}$ times, using independent random choices — the probability that the min-cut is not found in any of $\frac{n^2}{2}$ repetitions reduces to become $< \frac{1}{e}$ .

Further repetition will reduce the probability further.

This is an example of a Monte Carlo algorithm.

# An improved randomized min-cut algorithm

The above randomized min-cut algorithm can be used to contract a graph of, say, $n$ nodes down to one with $t$ nodes.

Importantly, any min-cut $K$ survives with probability $\Omega((\frac{t}{n})^2)$ .

This inspires an improved min-cut algorithm (Karger and Stein, 1996) which

- first performs two independent contractions to give two graphs $H_1$ and $H_2$ , each with $\lceil 1 + \frac{n}{\sqrt{2}} \rceil$ nodes — this bound ensures the min-cut survives with probability $\geq \frac{1}{2}$
- then recursively computes the min-cuts of $H_1$ and $H_2$
- and selects the smaller of the two min-cuts.

# An improved randomized min-cut algorithm

The above randomized min-cut algorithm can be used to contract a graph of, say, $n$ nodes down to one with $t$ nodes.

Importantly, any min-cut $K$ survives with probability $\Omega((\frac{t}{n})^2)$ .

This inspires an improved min-cut algorithm (Karger and Stein, 1996) which

- first performs two independent contractions to give two graphs $H_1$ and $H_2$ , each with $\lceil 1 + \frac{n}{\sqrt{2}} \rceil$ nodes — this bound ensures the min-cut survives with probability $\geq \frac{1}{2}$
- then recursively computes the min-cuts of $H_1$ and $H_2$
- and selects the smaller of the two min-cuts.

It can be shown that this improved algorithm runs in $O(n^2 \log n)$ time, using $n^2$ space.

Furthermore, it will find a min-cut with probability $\Omega(\frac{1}{\log n})$ .

# Outline

# Las Vegas algorithms – characteristics

These are randomized algorithms which never produce incorrect results, but whose execution time may vary from one run to another .

Random choices made within the algorithm are used to establish an expected running time for the algorithm that is, essentially, independent of the input.

# Las Vegas algorithms – characteristics

These are randomized algorithms which never produce incorrect results, but whose execution time may vary from one run to another .

Random choices made within the algorithm are used to establish an expected running time for the algorithm that is, essentially, independent of the input.

Some literature distinguishes:

- randomized algorithms that always return the correct result — Sherwood
- from those that if they return a result, it is always correct, but may not return a result — Las Vegas

# A randomized Quicksort algorithm

For an input of size $n$ , Quicksort has:

- a worst case time behaviour of $O(n^2)$ , but
- an average case time behaviour of $O(n\log n)$

The difficulty is determining the pivot point — very uneven splits can be obtained.

Interestingly, $O(n\log n)$ behaviour can be achieved.

# Randomized Quicksort

Given a set $S$ of $n$ numbers.

1. If $\mid S \mid \leq 1$ , output the elements of $S$ and stop.
2. Choose a pivot element $y$ uniformly at random from $S$ .
3. Determine the set $S_1$ of elements $\leq y$ , and the set $S_2$ of elements $> y$ .
4. Recursively apply to $S_1$ , output the pivot element $y$ , then recursively apply to $S_2$ .

# Informal analysis

Randomized quicksort has expected time (averaged over all choices of pivots) of $O(n\log n)$ .

Assume we sort the set and divide into four parts, the middle two contain the best pivots.

Each is larger than at least $\frac{1}{4}$ of the elements and smaller than at least $\frac{1}{4}$ of the elements.

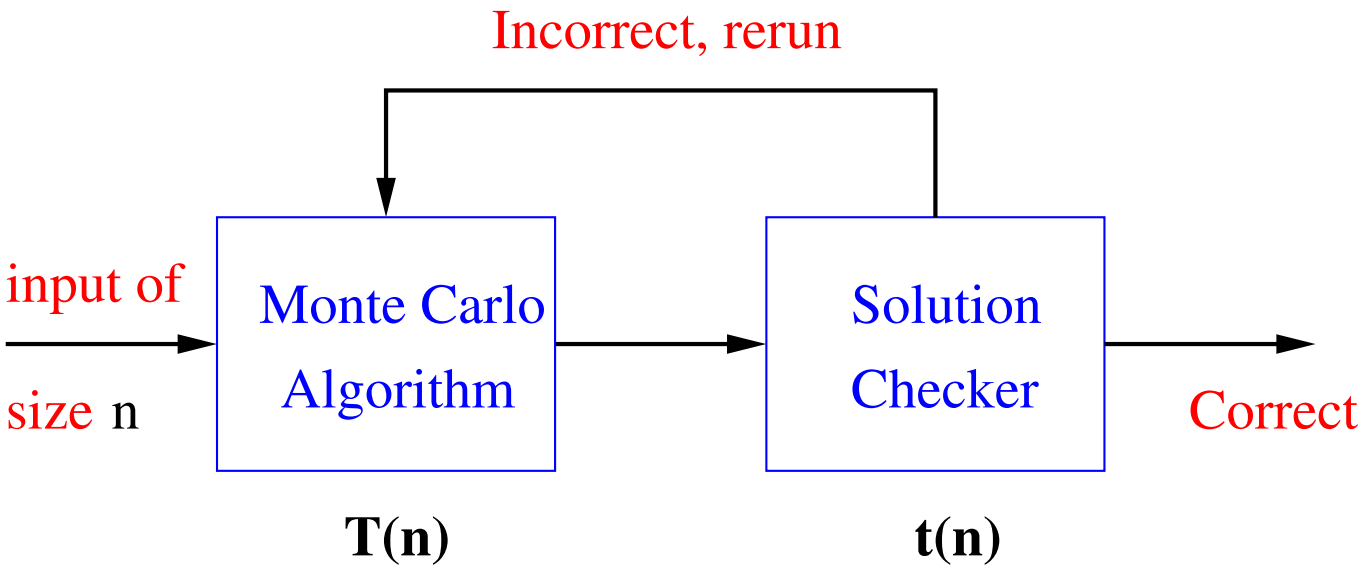
Choosing an element from the middle two means we split at most $2\log 2n$ times.

A random choice will only choose from these middle parts half the time, but this is good enough to give an average call depth of $2(2\log 2n)$ .

And hence the expected time of $O(n\log n)$ .

# From Monte Carlo to Las Vegas!

Since (our) Las Vegas algorithms are Monte Carlo ones with an error probability of zero, we can construct Las Vegas ones from Monte Carlo.




If the success probability is $p(n)$ , what's the expected run time?

# Analysis ...

Let the number of iterations be denoted by the random variable $X$ .

Independent choices are made on each iteration and $X$ is said to be geometrically distributed.

Given success probability $p(n)$ , the expectation $E[X]$ is:

$$\frac{1}{p(n)}$$

.

Hence, the expected run-time of the Las Vegas algorithm is:

$$\frac{T(n) + t(n)}{p(n)}$$

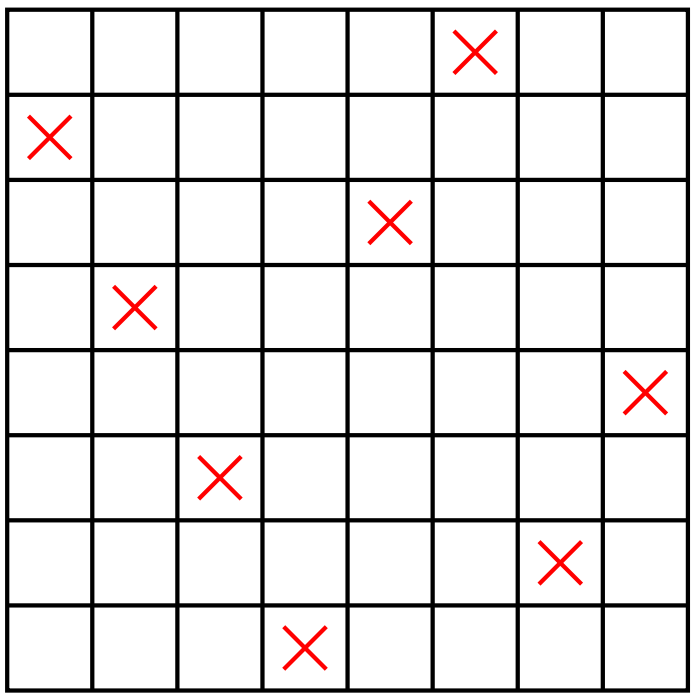
.

# Eight queens problem — another Las Vegas example

Place 8 queens on a chess board so that no one attacks another.

> *Remember: a queen attacks other pieces on the same row, same column and same diagonals.*

A possible solution:



# Backtracking algorithm

Place first queen in top-left corner.

Assume now the situation with some non-attacking queens in place.

If $< 8$ queens on board, place a queen in the next row — if it attacks a queen, move along the row one square at a time. If no position is possible, move the queen in the immediately preceeding row on one square. If no position is suitable, move the queen in the next row back, etc..

This algorthm actually returns a solution after examining $114$ of the $2057$ possible states.

# A Las Vegas approach – non-backtracking

Assume that $k$ rows, $0 \leq k \leq 8$ , are successfully occupied by queens.

If $k = 8$ then stop with success.
Otherwise, proceed to occupy row $k + 1$ .

Calculate all positions on this row not attacked by existing queens.
If there are none, then fail.
Otherwise, pick one at random , and continue to next row.

Note, there is no backtracking, the algorithm simply fails if a queen can't be placed.

BUT this can be repeated, and will consider a probably different placement.

# A Las Vegas approach – non-backtracking

Assume that $k$ rows, $0 \leq k \leq 8$ , are successfully occupied by queens.

If $k = 8$ then stop with success.
Otherwise, proceed to occupy row $k + 1$ .

Calculate all positions on this row not attacked by existing queens.
If there are none, then fail.
Otherwise, pick one at random , and continue to next row.

Note, there is no backtracking, the algorithm simply fails if a queen can't be placed.

BUT this can be repeated, and will consider a probably different placement.

Amazingly, this is better than the backtracking algorithm.

# Random eight queens — cont.

The probability of success for the random placement is $0.1293\ldots$ , i.e. approximately $1$ in $8$ times, just by guessing.

The expected number of states explored can be calculated to be around $56$ , compared with $114$ for the deterministic algorithm.

Further improvements can be made by combining random placement with some backtracking, first fixing some queens at random and then completing the arrangement via backtracking.

When the first two rows are occupied at random, then the expected number of states explored to find a solution becomes just $29$.

# Outline

# Further Reading on randomized algorithms

- Algorithmics: Theory and Practice. G. Brassard and P. Bratley, Prentice-Hall, 1988.

  *Introductory chapter on the topic.*

- Randomized Algorithms. R. Motwani and P. Raghavan, Cambridge University Press, 1995.

  *An excellent comprehensive account of the topic. A very good source book on approaches to using randomization in algorithms.*

- A new approach to the minimum cut problem. D. Karger and C. Stein, Journal of the ACM, Vo. 43, No. 4, July 1006, pp 601-640.

  *Only for the very brave — a briefer and slightly easier account of this algorithm is contained in Chapter 10, section 10.2, pages 289–295, of Motwani and Raghavan's above book.*