Logistics

- Ask for help! Hoss and I both have office hours that you can come to for help.
- Final exam time: 12:30pm May 7th.
- You should have completed your initial exploration and decided on the questions you want to investigate.
- This week you should be defining the hypotheses you will use to answer your questions.
- You should also be deciding on an experimental design.
- Think about potential extraneous and nuisance variables.
- Look into the Graduate Student Resource Center

https://unmgrc.unm.edu/support-services/individual-consultations.php

Graduate Student Resource Center

Writing

Schedule an individual consultation for assistance with graduate-level writing assignments, including (but not limited to) funding proposals, presentations, articles, and even your thesis or dissertation! Our graduate consultants are trained to provide feedback on content, organization, argument, and structure. The goal is to help you become a proficient writer, so come prepared to engage with our consultants and do not expect a proofreading or editing session.

ESOL Writing

These one-on-one consultations are designed to address specific concerns and questions about academic writing for students who have English as a second language. Using a collaborative approach, the consultant will engage you in reviewing your writing strengths as well as in identifying areas for improvement. The focus of the consultation will be on helping you develop self-directed language learning skills and applying the feedback to future academic writing pieces.

Statistics

GRC Statistics Consultants offer support with quantitative research, including statistical design, analysis, and interpretation. Our consultants can also assist with particular statistics software or recommend additional resources. Students enrolled in undergraduate statistics courses may also use CAPS Math drop-in labs.

Designing Experiments

- What do the benchmarks output?
- Is that the value you what you want to compare your other language?
- What can you do?

Experimental Computer Science

MATTHEW FRICKE

1.0 – SEND CORRECTIONS TO MFRICKE@UNM.EDU

Applications to Computer Science

- Experiments usually show up in CS in:

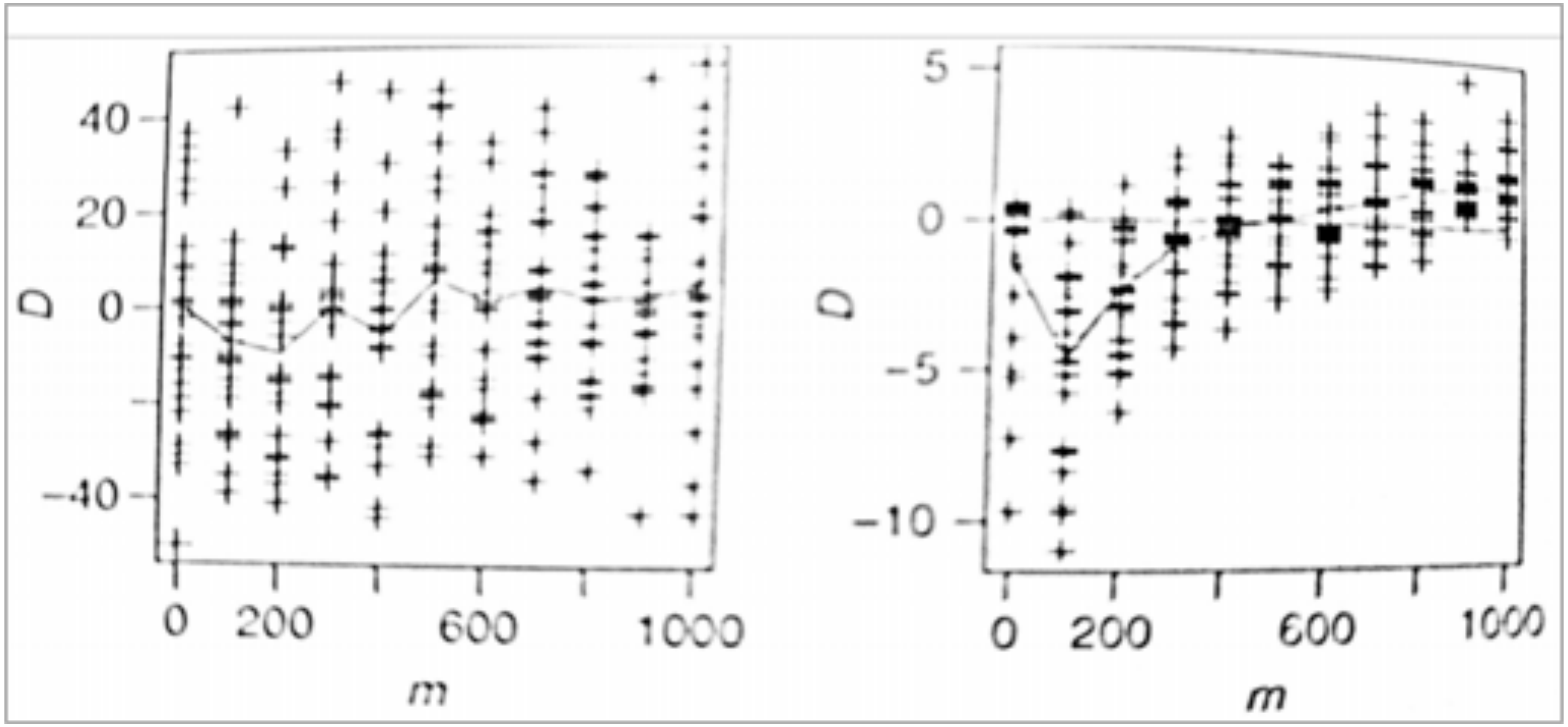
- Algorithm Engineering
- Computer Performance
- Simulation

Success of experiments

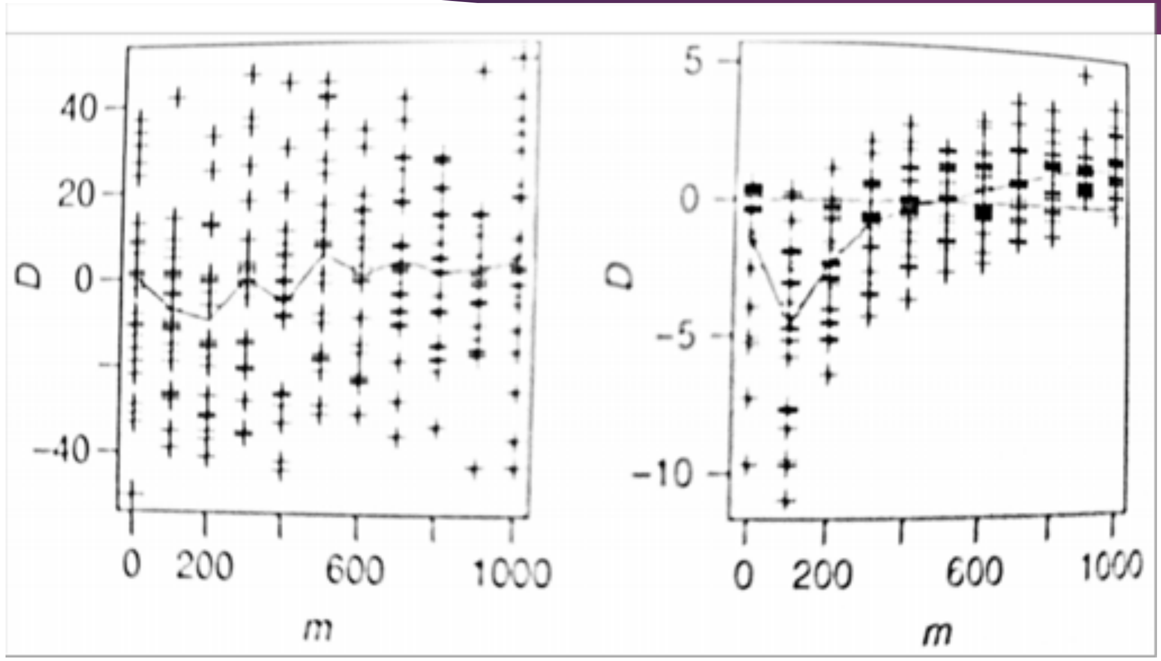
Successful experiments are partly the product of good experimental designs; there is also an element of luck (or savvy) in choosing a well-behaved problem to study

An experiment is not considered negative when it disproves your conjecture: an experiment fails by being inconclusive

Inconclusive versus analysis-friendly data



Inconclusive versus analysis-friendly data



Two experiments to study the average value of a function D(m).

Each column of data represents 25 independent trials at levels m=1, 101, 201, …, 1001
The lines connect the sample means in each column

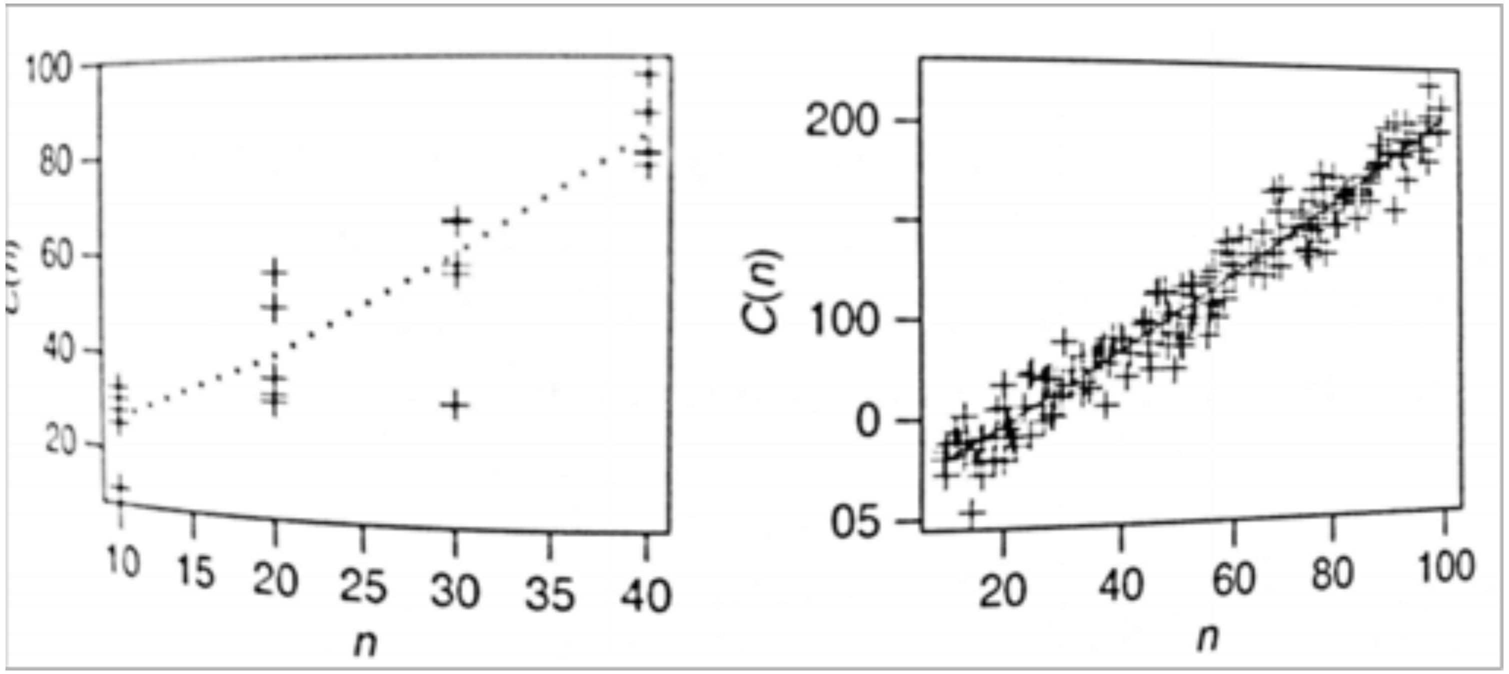
The mean is known to change from negative to positive as m increases
The experimental problem is to *find the crossover point* ***mc*****.**

The sample means have the same theoretical average in both experiments, but the experiments yield different insights with respect to **mc.**

Guideline: For best views of how average cost depends on parameters, work to magnify response and minimize variance

Design for clear views

Sometimes the right experimental design for the problem does not produce easy-to-analyze results. But many designs can be improved to better show the relationship between parameters and performance.

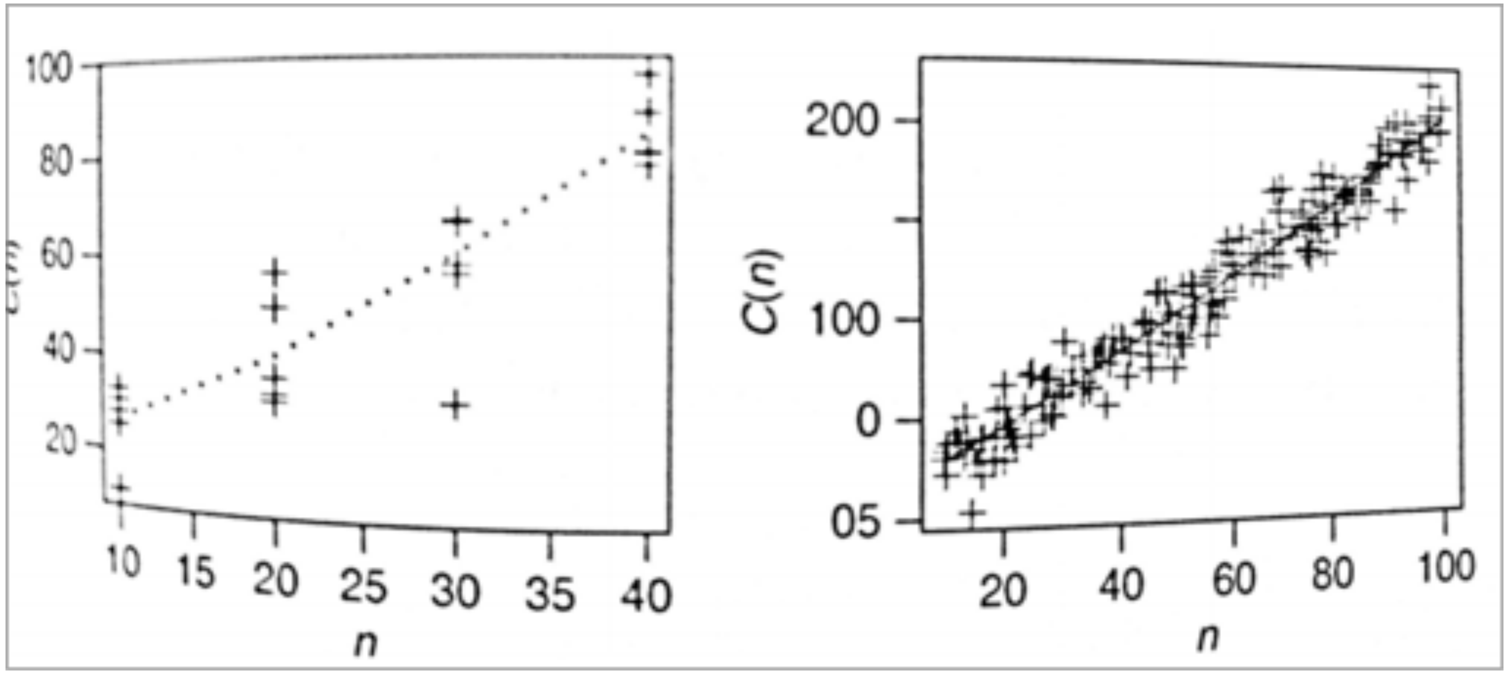


Inconclusive

Strong support for linearity

Design for clear views

Sometimes the right experimental design for the problem does not produce easy-to-analyze results. But many designs can be improved to better show the relationship between parameters and performance.



Inconclusive

Strong support for linearity

The underlying function is the same in these two experiments. But the experimental designs differ in the range, spacing, and number of levels of n, and the number of random trials

Design for clear views

Do's and don'ts for designing better experiments (with a goal of clear views)

- Do run more trials: variance in a data sample is inversely proportional to sample size.
- Do expand the range of n values: if the response of C(n) to n is small compared to variance, try magnifying the response by increasing the range of n levels.
- Do "right size" the data sample. Avoid experiments that produce too many or too few data points to be analyzable.
- Do prefer narrow performance indicators: one that focuses closely on one component of algorithmic performance. Simple relationships are generally easier to model and analyze.
- Focus on one thing at a time if at all possible.

Design for clear views

Do’s and don’ts for designing better experiments (with a goal of clear views)

- Don’t summarize prematurely: the right choice of statistics depends on distribution properties of the sample

- Don’t use “lossy” performance indicators. Report measurements that maximize the information content of each trial

    Suppose the experimental goal is to study a ratio R=X/Y

    If the test program reports both, X and Y, R can be calculated

    If it reports only R, other useful quantities for data analysis like X-Y or (X-Y)/X cannot be calculated

Guideline: Design your experiments to maximize the information content in the data: aim for a clear view of simple relationships

Variance reduction techniques

- If the data will not cooperate despite your best design efforts, consider a technique for reducing the variance.
- A variance reduction technique (VRT) modifies the test program in a way that reduces variance in the measured outcomes, on the theory that less variance yields better views of average case costs
- Variance can always be reduced by increasing the number of trials, but this is not always feasible.
- If the goal of the experiment is to understand variance as it occurs naturally, do not apply VRTs

Terminology: Design Point

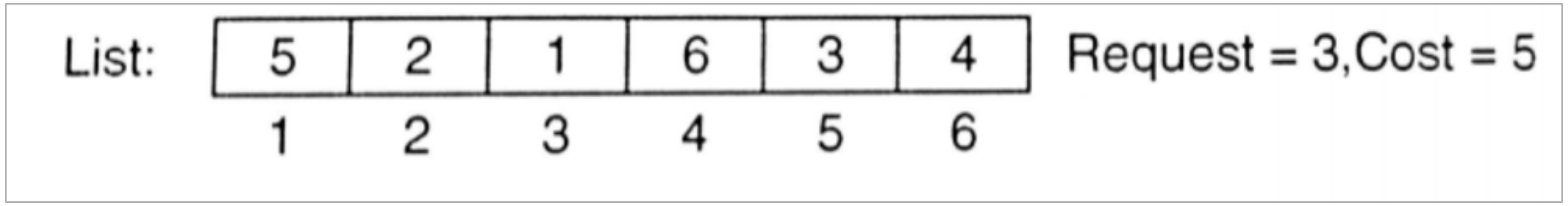
- When running experiments various input values the combination of inputs is called the

Design point

Case study for self-organizing sequential search rules

The self organizing search problem is to maintain a list of **n** distinct keys to service a series of **m** requests for keys

The cost of each request is equal to the position of the key in the list (linear search from the front)



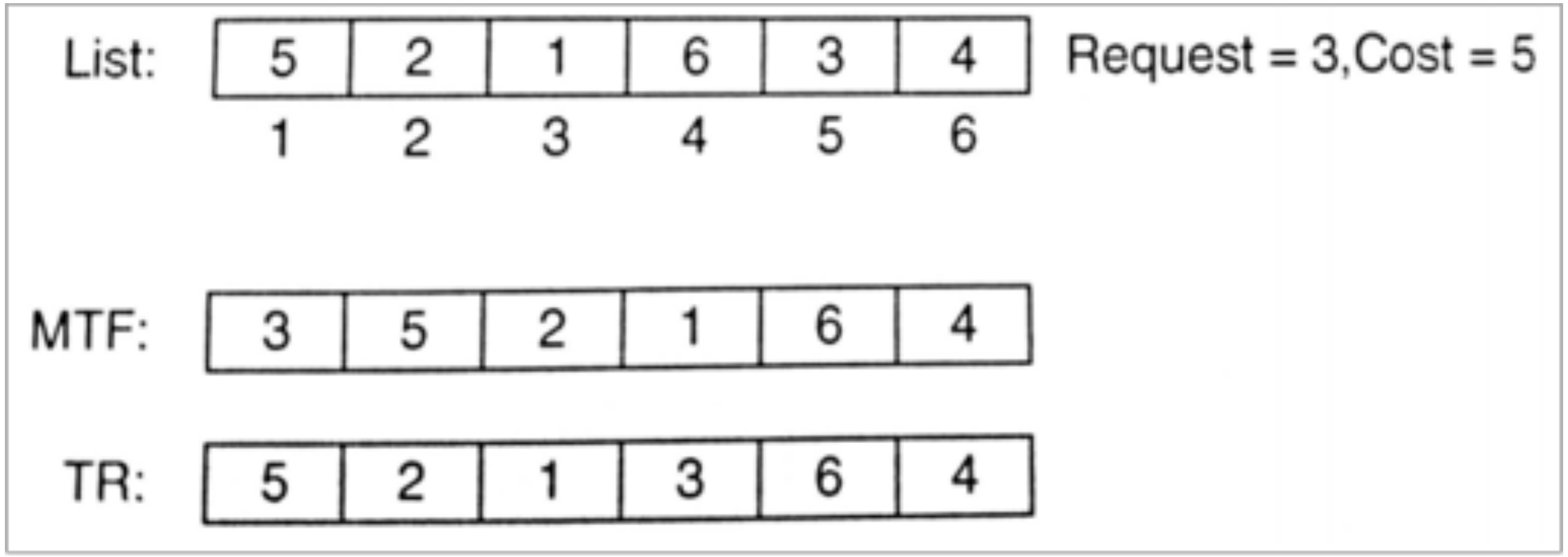
The list is allowed to reorder itself by some rule that tries to keep frequently requested keys near the front to reduce total search cost

What is a possible null hypothesis?

Case study for self-organizing sequential search rules

Move to Front (MTF): After key k is requested, move it to the front of the list

Transpose (TR): After key k is requested, move it one position closer to the front by transposing with its predecessor



Analyzing the rules

Suppose requests are generated independently at random according to a probability distribution $P(n) = p_1, p_2, p_3, \ldots p_n$ defined on n keys

The request cost for key k in position L[i] in a given list is equal to its position i

The average list cost for list L depends on the requests costs and request probabilities for each key:

$$C(L) = \sum_{i=1}^{n} i * p_{L[i]}$$

Analyzing the rules

The average cost of a rule is the expected cost of the m*th* request, assuming that L is initially in random order and that the rule is applied to a sequence of random requests generated according to distribution P(n)

Let $\mu(n,m)$ denote the average cost of MTF, where ***n*** is is the number of keys and *m is the number of the lookup.*

Let $\tau(n,m)$ denote the average cost of TR

The experiments measure costs for requests drawn from Zipf's distribution Z(n) defined over the integers 1, …, n

The probability that key k is requested next is given by $p(k) = \dfrac{1}{kH_n}$

Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table, proposed by linguist George Zipf.

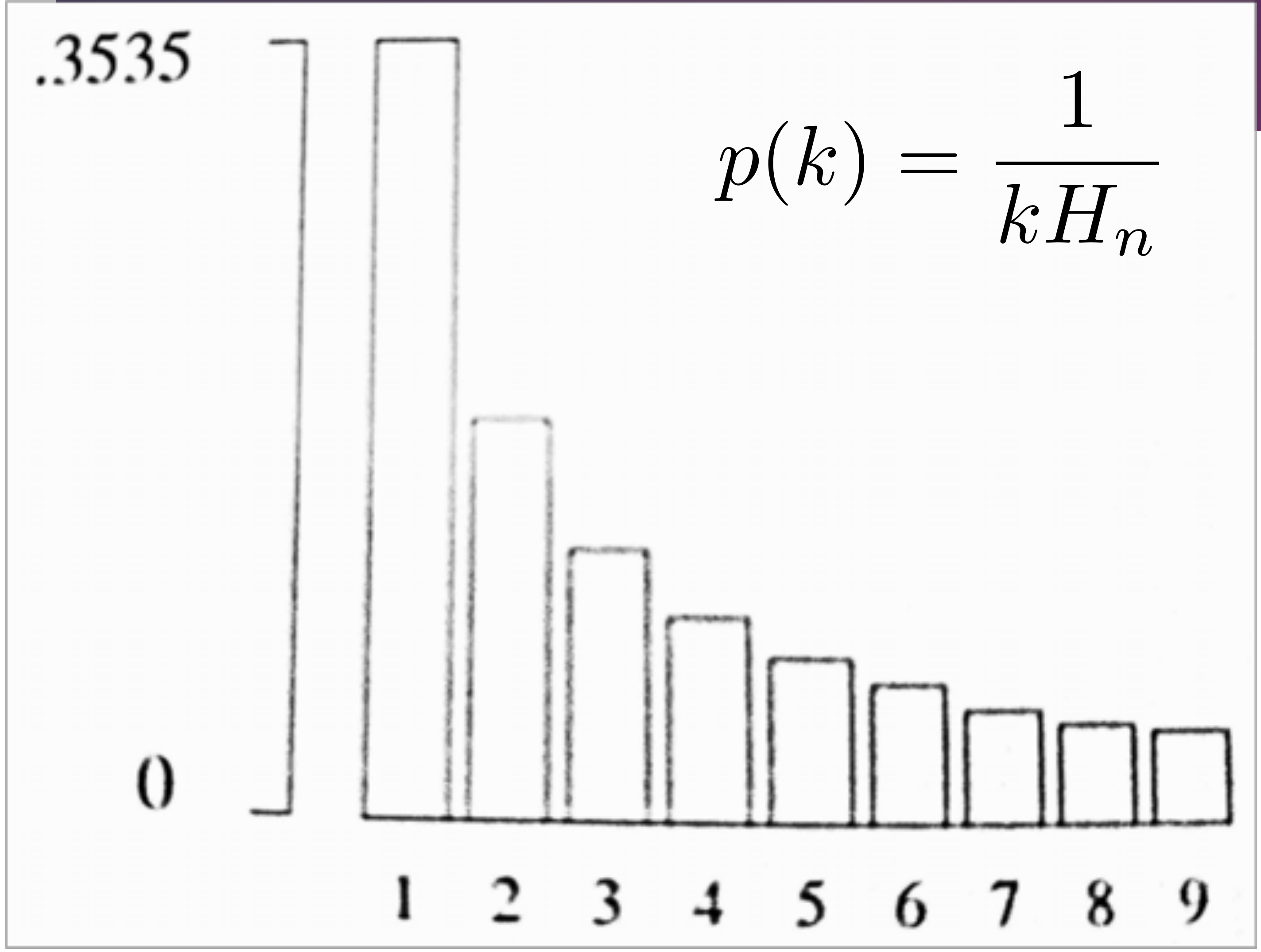
Aside Harmonic Series

Zipf’s law models the frequency of words (or keys in this case) as inversely proportional to the harmonic series.

$$p(k) = \frac{1}{kH_n}, \text{ where } H_n = \sum_{k=1}^{n} \frac{1}{k}$$

Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table, proposed by linguist George Zipf.

Zipf's distribution for n=9



$$p(k) = \frac{1}{kH_n}$$

Where H$_n$ is the n*th* harmonic number (multiplying by H$_n$ scales the probabilities so they sum to 1)

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

Exploration

There are no known formulas for calculating $\mu(n, m)$ and $\tau(n, m)$ under this distribution, so we develop experiments to study these average costs.

Exploration

There are no known formulas for calculating $\mu(n, m)$ and $\tau(n, m)$ under this distribution, so we develop experiments to study these average costs.

- For each random trial the code generates an initial list L that contains a random permutation of the keys.
- Then it generates a random sequence of keys lookups according to the Zipf's distribution.
- For each key it looks up the request in the list, records the cost, and reorders the list according to the rule
- At the end the program reports the cost of the m*th* request

Setting up our experiment

```
SequentialSearchTest(n, m, R, trials)
    For ( t=1; t<= trials, t++ )
        L = randomPermutation (n)
        For ( i=1; i<=m; i++ )
            K = randomZipf(n)
            For ( j=1; L[j] != k; j++ )
                Cost = j
                reorder(R, L, j)
        printCost(R, t, n, m, cost)
```

n = distinct keys
m = requests
R = MTF or TR

t random trials at each design point (n,m)

Setting up our experiment

The experiment runs t random trials at each design point (n,m)

The random variate $M_i(n,m)$ denotes the cost of MTF reported in the *ith* trial at this design point

The sample mean at a design point is the average of t outcomes

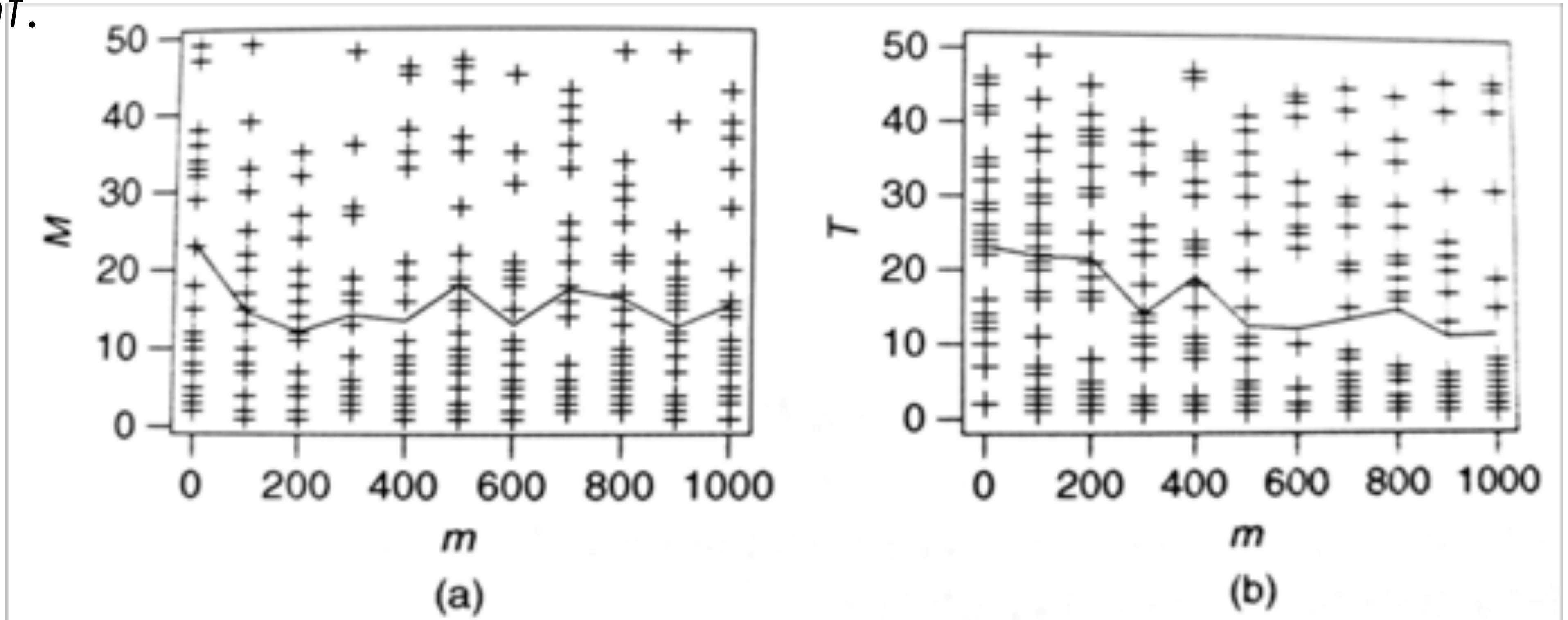
$$\bar{M}(n,m) = \frac{1}{t}\sum_{i=1}^{t} M_i(n,m)$$

The expectation $E[M_i(n,m)] = \mu(n,m)$

The variate $M_i(n,m)$ is an estimator of $\mu(n,m)$

The first experiment

Mi(50,m) and Ti(50,m) in 25 random trials at each design point n=50 and m= 1, 101, 201, …, 1001. The input sequences are *independent*.



Let $T_i(n,m)$ and $Var(T(n,m))$ denotes the cost of TR reported in the *ith* trial at this design point (M_i and $Var(M(n,m)$ are defined analogously.

Result: For any non uniform request distribution such as Zipf, TR has lower asymptotic cost, but MTF reaches its asymptote more quickly

There is a crossover point m_c such that

when $1 < m < m_c$ $\mu(n,m) < \tau(n,m)$

when $m_c < m$ $\mu(n,m) > \tau(n,m)$

We are also interested in the sample variance, a statistic that describes the dispersion of points away from their mean

$$Var(M(n,m)) = \frac{1}{t}\sum_{i=1}^{t}(M_i(n,m) - \overline{M}(n,m))^2$$

Let $T_i(n,m)$ and Var(T(n,m)) denotes the cost of TR reported in the *ith* trial at this design point (Mi and Var(M(n,m) are defined analogously.

Known: for any non uniform request distribution such as Zipf, TR has lower asymptotic cost, but MTF reaches its asymptote more quickly

There is a crossover point m_c such that

when $1 < m < m_c$ $\boxed{\mu(n,m) < \tau(n,m)}$

when $m_c < m$ $\boxed{\mu(n,m) > \tau(n,m)}$

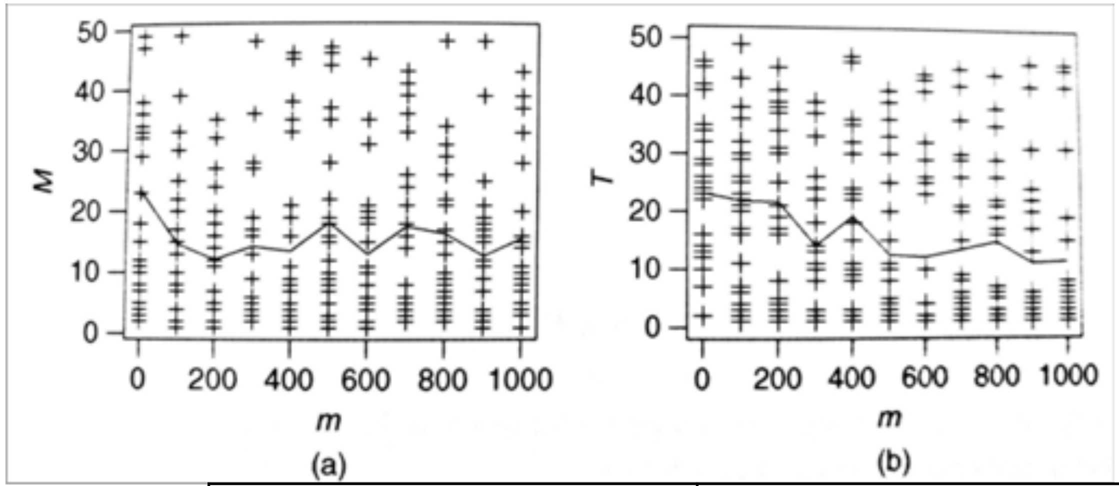
Our first experimental goal is to locate m_c

We are also interested in the sample variance, a statistic that describes the dispersion of points away from their mean

$$\boxed{Var(M(n,m)) = \frac{1}{t}\sum_{i=1}^{t}(M_i(n,m) - \overline{M}(n,m))^2}$$

The first experiment

Mi(50,m) and Ti(50,m) in 25 random trials at each design point
n=50 and m= 1, 101, 201, …, 1001



Since the confidence intervals overlap, we cannot say with any certainty whether at some *m*

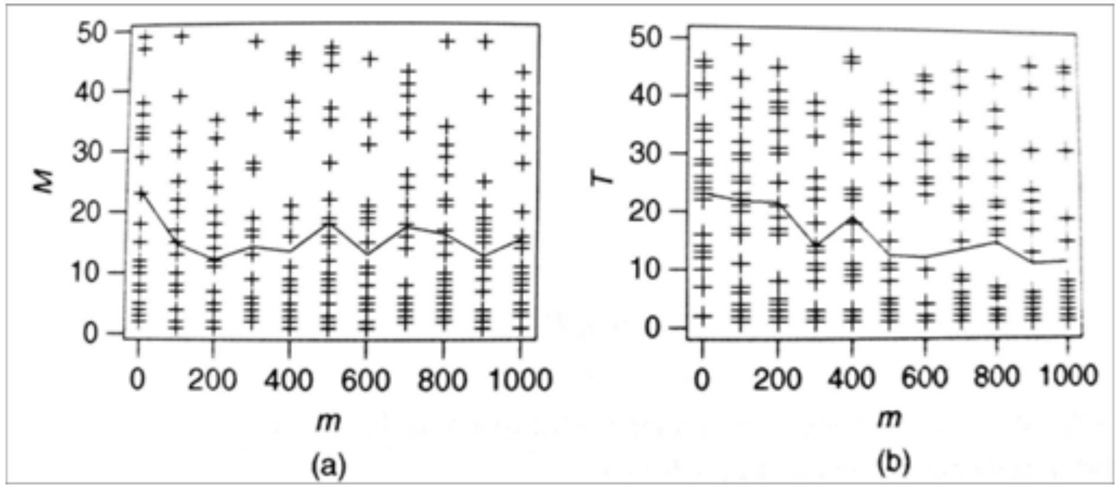
$$\mu(n,m) < \tau(n.m)$$

	Mean	**Var**	**95% Conf.**
MTF	15.6	159.75	[10.64, 20.56]
TR	11.4	195.83	[5.91, 16.89]

The first experiment

We failed to reject the null hypothesis that there is a crossover point. Never mind actually finding that point.

Looking at the experimental results and the overlapping confidence intervals what could be a problem here?



Variance Reduction Techniques (VRT)

It is possible that the large sample variance is keeping us from getting a clear view.

A couple of methods we will examine are:

Common Random Numbers (CRN)

Conditional Expectation (CE)

Common Random Numbers

Common random numbers VRT can be applied when:

1. The goal of the experiment is to compare the differences in costs of two (or more) algorithms.
2. There is a reason to believe that the costs of the algorithms are positively correlated with respect to some random variate in each trial.

Measuring cost differences in paired trials with matching random variates should yield outcomes with less variance than measuring differences using independent random variates

Common Random Numbers

In other words if two random variables are correlated then the difference between those variables will have a lower variance than either alone:

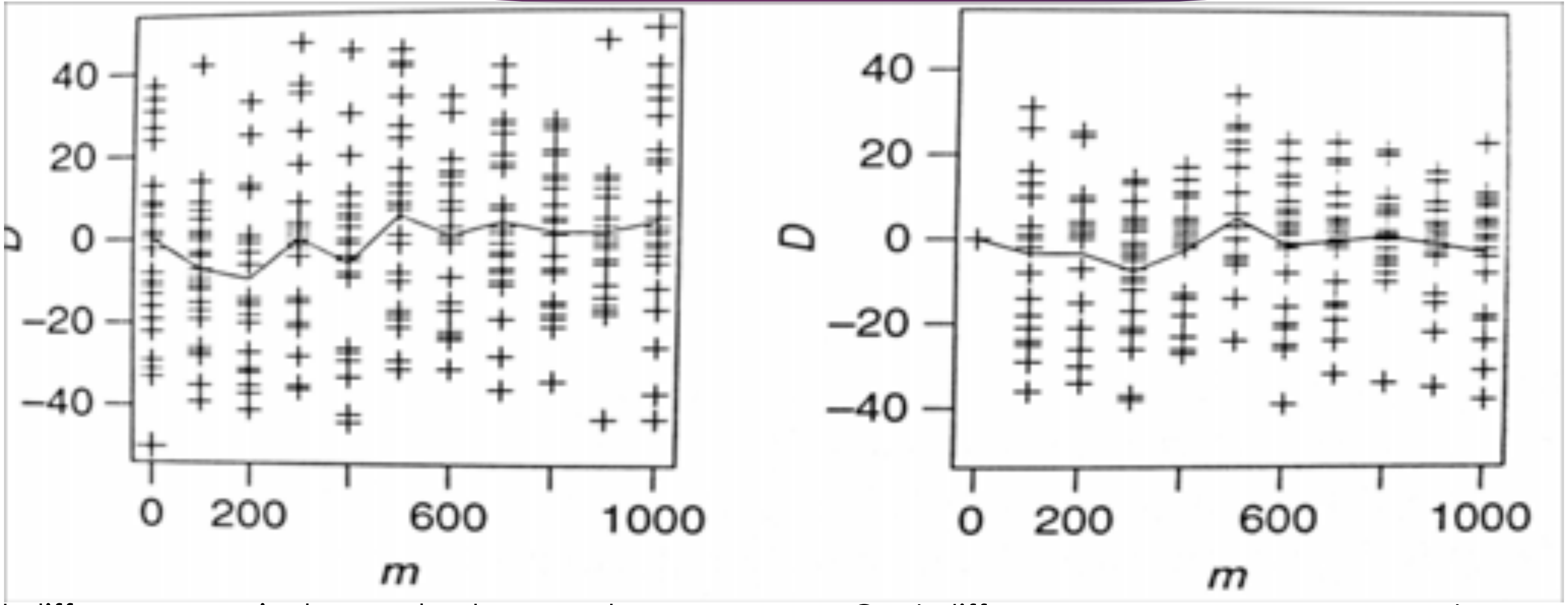
$$\mathrm{Var}(X - Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) - 2\mathrm{Cov}(X, Y)$$

So we redefine our response to be the difference in performance as opposed to the plain performance.

And we compare the response of X and Y on the **same input**. This is how we introduce the correlation. (This is why it is called *common* random variables)

Common Random Numbers

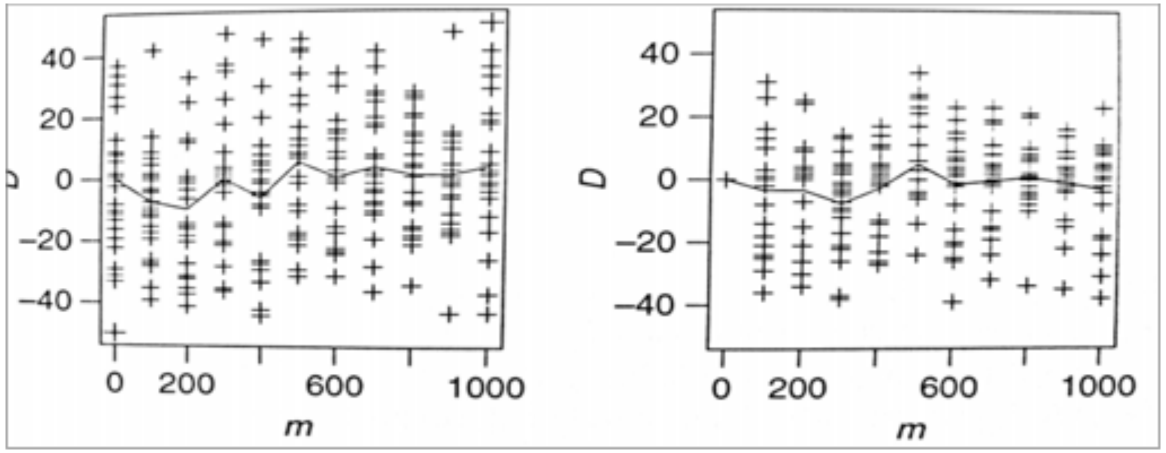
$$D_i(n,m) = M_i(n,m) - T_i(n,m)$$



Cost difference on independent request sequences   Cost difference on common request sequences

Common Random Numbers

$$D_i(n,m) = M_i(n,m) - T_i(n,m)$$



Even though the sample mean is negative in the second experiment, this **VRT is not enough to obtain a definitive answer** about m_c, since the 95% ci contains both negative and positive values. But this experiment needs fewer additional trials to shrink the range to answer the question

	Mean D(50,1001)	**Var(50,1001)**	**95% Conf.**
Independent	4.20	456.42	[10.64, 20.56]
CRN	-3.00	206.75	[-8.64, 2.63]

Common Random Numbers

Can be used when the performance of two tests subjects is positively correlated with respect to some random variate, compare performance in paired trials with identical values for that variate

Common Random Numbers

Common Random Numbers is considered an easy variance reduction technique but has pitfalls.

For example, if the correlation is negative between X and Y then the variance will be increased instead of decreased.

Conditional Expectation

Based on the fact that for any random X and Y

$$\mathrm{E}(\mathrm{E}(X|Y)) = \mathrm{E}(X)$$

$$\mathrm{Var}(\mathrm{E}(X|Y)) = \mathrm{Var}(X) - \mathrm{E}(\mathrm{Var}(X|Y))$$

Conditional Expectation

Based on the fact that for any random X and Y

$$\mathrm{E}(\mathrm{E}(X|Y)) = \mathrm{E}(X)$$

$$\mathrm{Var}(\mathrm{E}(X|Y)) = \mathrm{Var}(X) - \mathrm{E}(\mathrm{Var}(X|Y))$$

Therefore,

$$\mathrm{Var}(\mathrm{E}(X|Y)) \leq \mathrm{Var}(X)$$

Conditional Expectation

Based on the fact that for any random X and Y

$$\mathrm{E}(\mathrm{E}(X|Y)) = \mathrm{E}(X)$$

$$\mathrm{Var}(\mathrm{E}(X|Y)) \leq \mathrm{Var}(X)$$

Now suppose we wish to estimate E(X) and we know E(X | Y) is a function of Y (call it g(Y)).

Then g(X) has the **same expected value** as X but **a smaller variance.**

Conditional Expectation

Also called conditional Monte Carlo, conceptually splits an experiment into two phases:

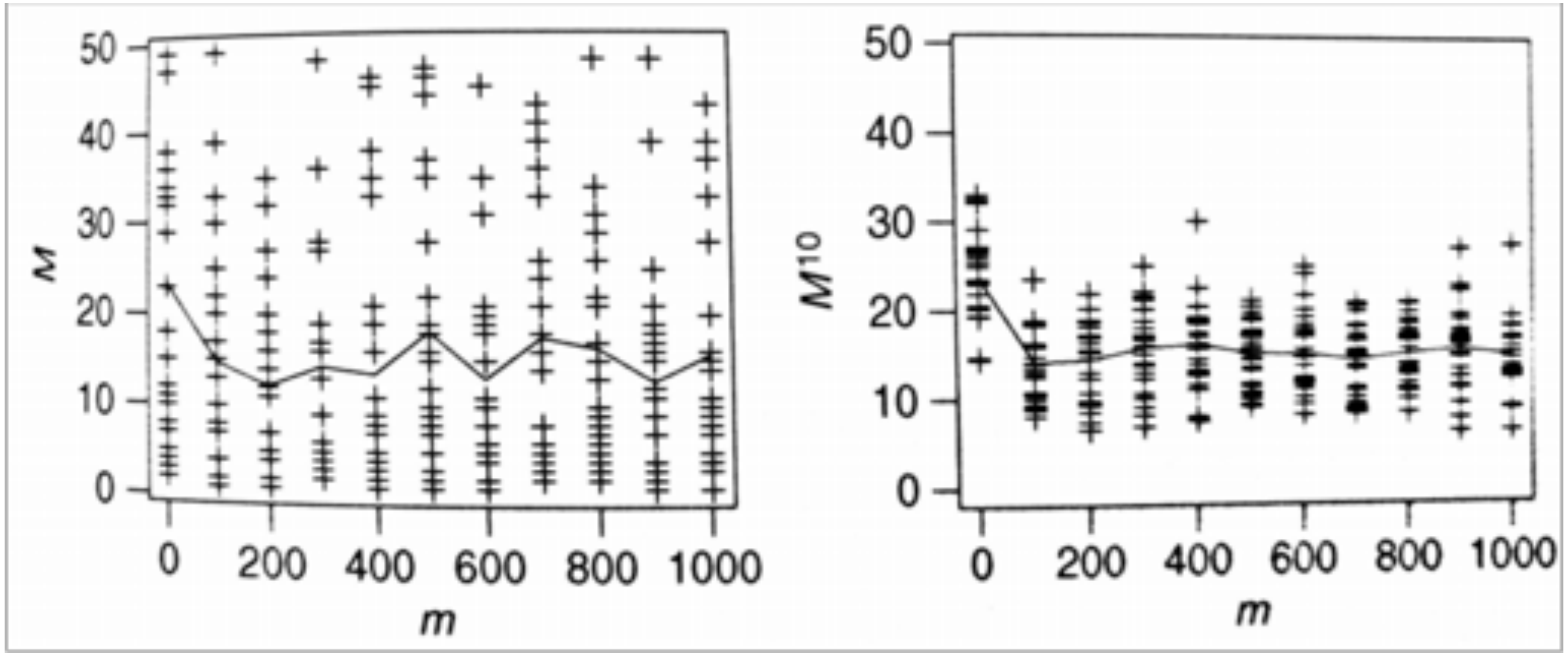
1. Generate a random state S
2. Generate a random sample of the cost of that state

Using this approach for MTF consists of a phase that generates a sequence of m-1 requests to obtain a random list order L (the state after m-1 requests), followed by a second phase that generates **r** random requests and reports their cost (without reordering the list)

The new variate $M^r_i(n,m)$ reports the average costs of r requests on the same list instead of just one. Our g(X) is this average.

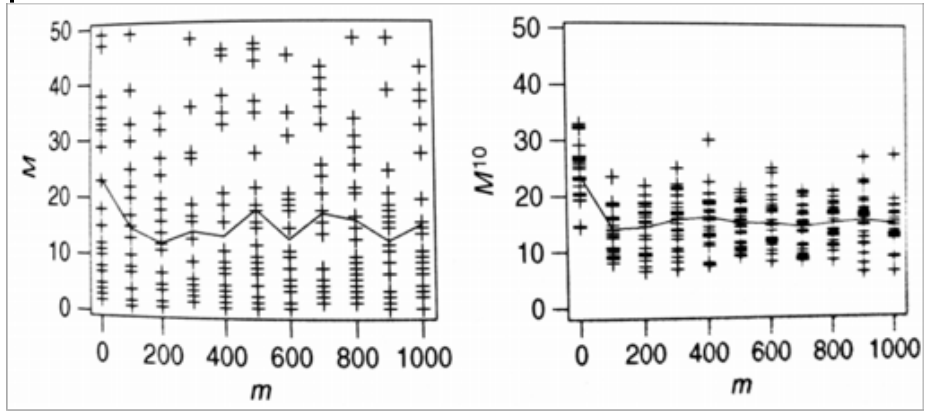
Conditional Expectation

Panel 1 shows $M_i(50,m)$ and panel 2 shows $M_i^{10}(50,m)$ in 25 random trials each



Conditional Expectation

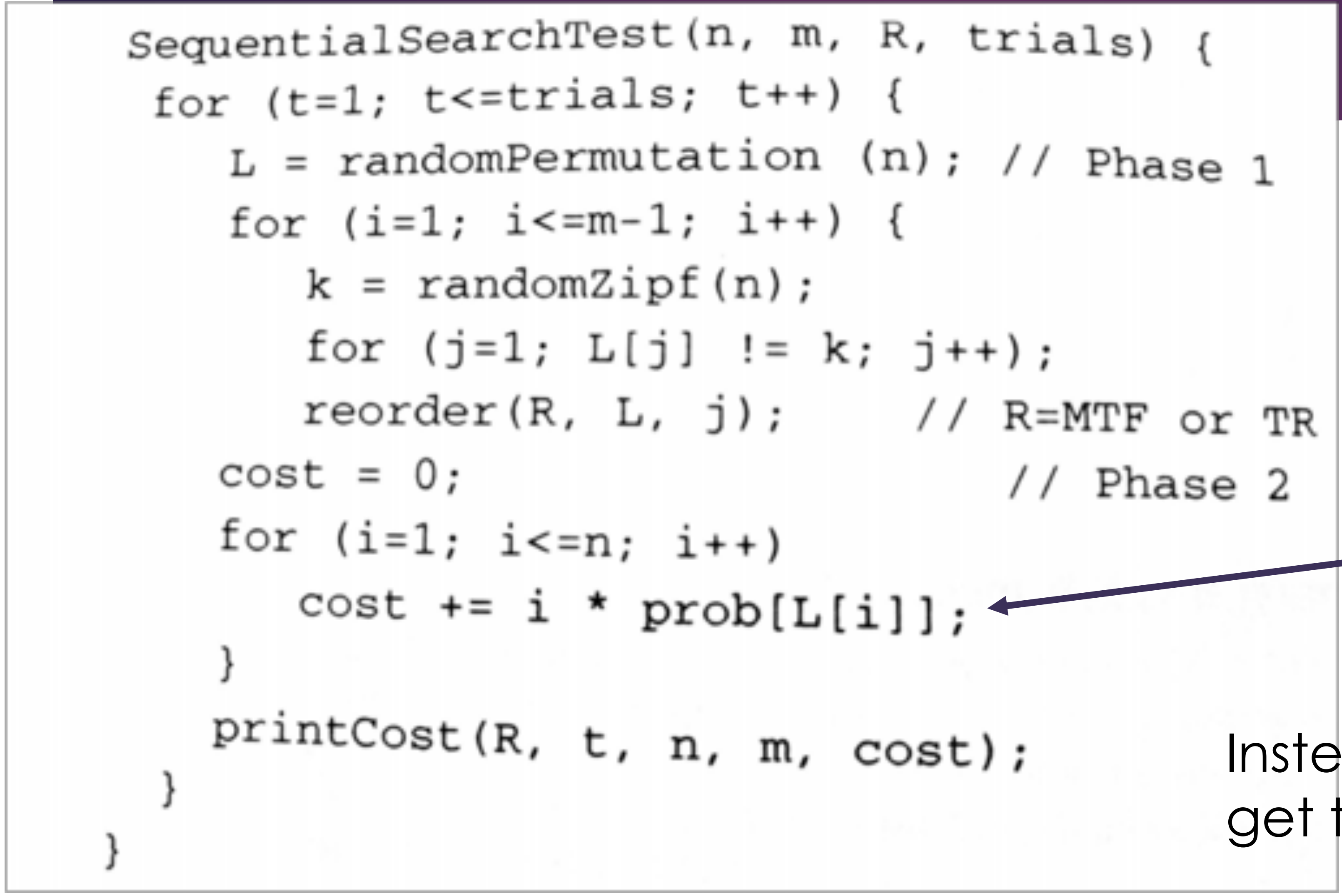
Panel 1 shows $M_i(50,m)$ and panel 2 shows $M_i^{10}(50,m)$ in 25 random trials each



Variance is 9.07 times smaller in this table; on average variance will be 10 times smaller; on average the range in confidence intervals will shrink by a factor of sqrt(10)

	Mean	**Var**	**95% Conf.**
$M_i(50,1001)$	15.60	159.75	[10.65, 22.55]
$M_i^{10}(50,1001)$	15.00	17.61	[13.35, 16.65]

Conditional Expectation – with averaging

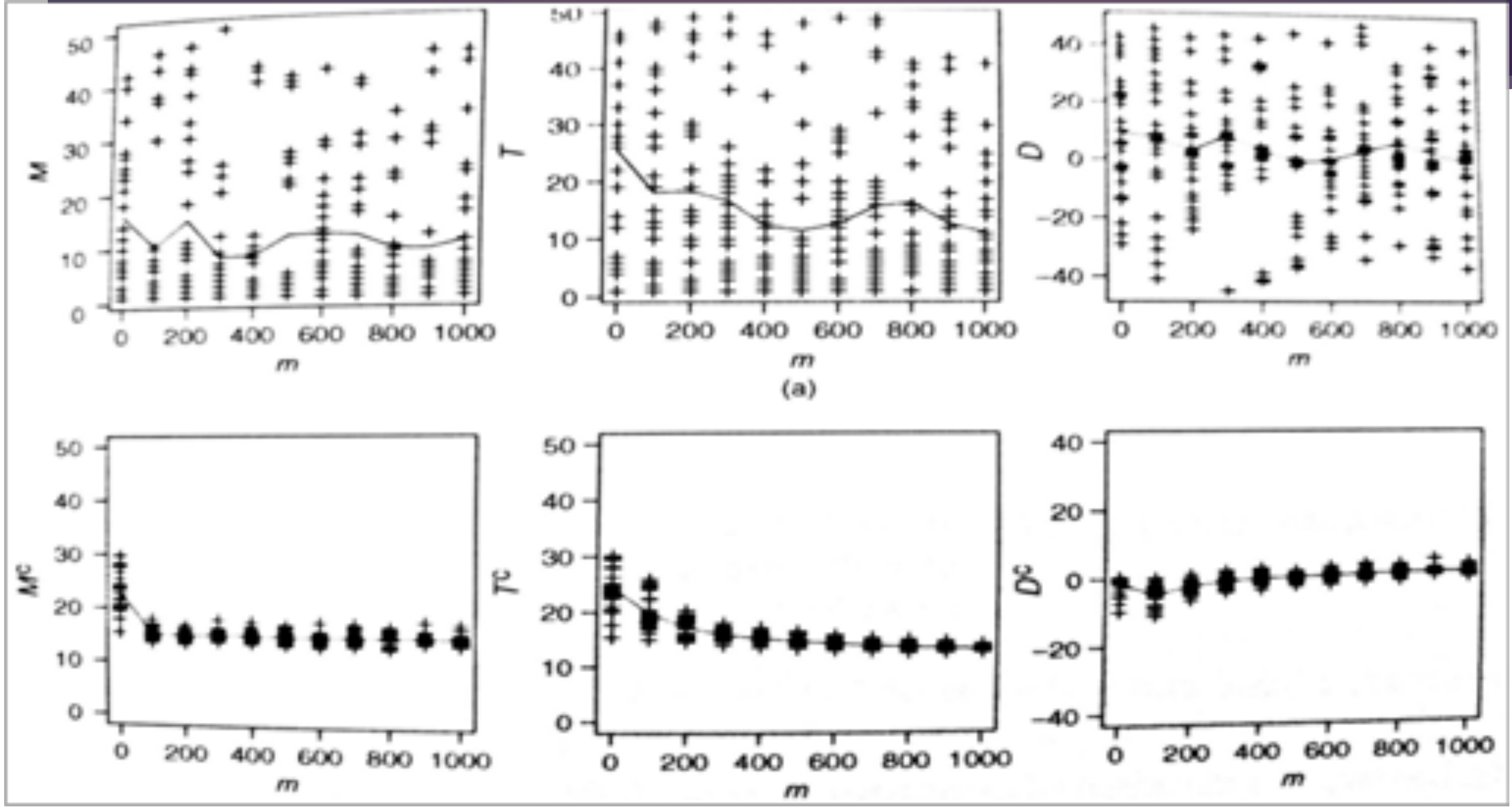


```
SequentialSearchTest(n, m, R, trials) {
 for (t=1; t<=trials; t++) {
    L = randomPermutation (n); // Phase 1
    for (i=1; i<=m-1; i++) {
       k = randomZipf(n);
       for (j=1; L[j] != k; j++);
       reorder(R, L, j);      // R=MTF or TR
   cost = 0;                          // Phase 2
   for (i=1; i<=n; i++)
      cost += i * prob[L[i]];
   }
   printCost(R, t, n, m, cost);
 }
}
```

$$C(L) = \sum_{i=1}^{n} i * p_{L[i]}$$

Instead of 10 random searches, get the average cost for list L

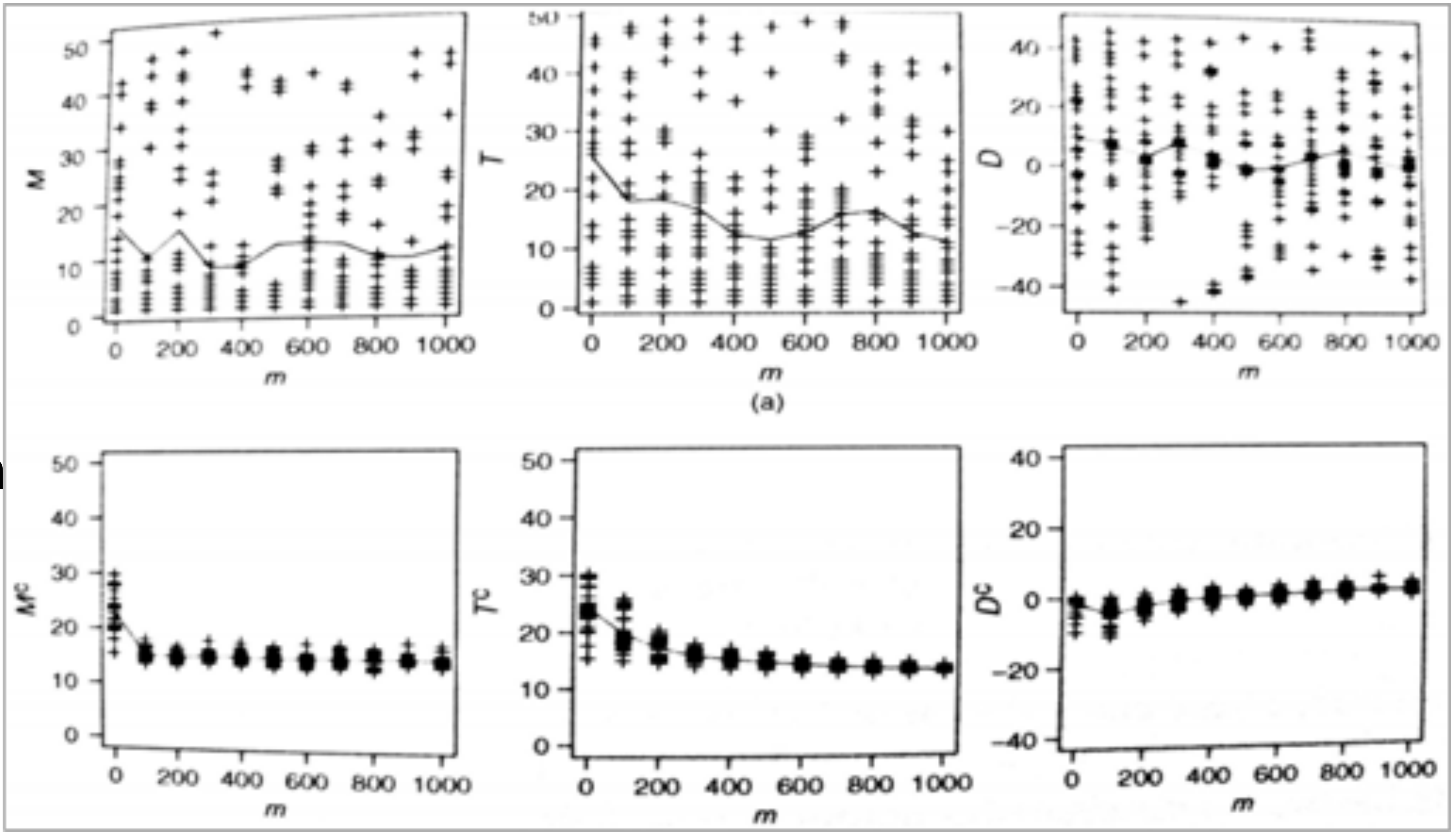
Conditional expectation with exact costs



First experiment

Conditional expectation with exact computation of average list cost

We can locate m_c somewhere between m=4001 and m=5001
More experiments focused in that region, together with the common random numbers VRT could be applied to get a tighter estimate



	M(50,1001)	T(50,1001)	D(50,1001)
Before VRT	15.6 [10.6, 20.6]	11.4 [5.9, 16.9]	4.2 [-4.2, 12.6]
After VRT	14.6 [14.3, 15.0]]	12.2 [12.0, 13.4]	1.5 [1.1, 1.9]

Conditional expectation with exact costs

when the computation of an average cost can be split into two parts, add extra work to reduce or eliminate variance in the second part

Guideline: Consider a variety of variance reduction techniques, including control variates, antithetic variates, stratification, poststratification, and importance sampling

More guidelines on data analysis

- Location and dispersion are the **yin and yang** of data summaries; do not report one without the other
- Apply logarithmic transformation, or more generally a power transformation to impose symmetry in a skewed data sample
- Consider these properties when deciding how to summarize a data sample: symmetry, outliers, skew, bi- or multimodality, kurtosis, and data censoring
- Calculate confidence intervals for your sample means
- Use bootstrapping (or related resampling methods) to generalize inferential methods to non-normal distributions and non-standard statistics

More guidelines on data analysis

- Different graphs give different views of relationships in data. Try many views to learn the full story
- Let the trend in residuals guide the search for a better model of your data
- Any data analysis technique applied to answer an asymptotic question about a finite data set must be considered a heuristic that provides no guarantees.
- Insight rather than certainty must be the goal

Computer Performance

Measuring performance

Basic characteristics of a computer system that we typically need to measure are:

- A count of how many times an event occurs
- The duration of some time interval
- The size of some parameter

Examples:

- Count how many times a processor initiates an input/output request
- How long each of these requests takes
- Number of bits transmitted and stored
- Number of times a prediction is correct

Measuring Computer Performance, A practitioner's guide
By David J. Lilja

Performance metrics

If we are interested specifically in the time, count, or size value measures, we can use that value directly and a performance metric

Other times we are interested in normalizing event counts

> For example, normalizing time measured to a common time basis to provide a speed metric such as operations executed per second

This kind of metric is a **rate metric** and it is calculated by dividing the count by some normalizing factor (e.g. seconds)

Choosing an appropriate performance metric depends on the goals for the specific situation and the cost of gathering the necessary information

Performance metrics

There are many metrics that have been used, for example: MIPS, MFLOPS, and others are invented for new situations as they are needed

Not all metrics are good, as they can be misleading. A good metric should be:

- **Linear:** the value of the metric should be linearly proportional to the actual performance. Not all metrics satisfy this proportionality requirement (e.g., logarithmic metrics such as dB scale used to describe intensity of sound)
- **Reliable:** if a system outperforms another by the specific metric, then this should be reflected in the actual performance (e.g., MIPS is unreliable)
- **Repeatable:** the same value of the metric is measured every time the same experiment is performed

Performance metrics (cont)

Easy to measure: if a metric is not easy to measure, it is unlikely that it will be used, and the more likely it will be determined incorrectly

Consistent: the units of the metric and its precise definition are the same across different systems and different configuration of the same system (e.g., MIPS and MFLOPS are not consistent)

Independent: there is pressure from manufacturers to design their machines to optimize the value obtained for a particular metric and to influence the composition of the metric to their benefit

Examples of performance metrics

- Clock rate, e.g., 250 MHz but ignores complex interactions w. memory
- MIPS, millions of instructions executed per second = n / (timeN x 10^6)
- MFLOPS, millions of floating point operations per second
- SPEC, set of integer and floating point benchmark + methodology
- Execution time, time to execute a program (problems, context switching)
- Accuracy, percent of correct instances
- Precision, how many selected items are relevant?
- Recall, how many relevant items are selected?
- Other: latency, bandwidth, throughput, response time

Speedup and relative change

Speedup and relative change are useful metrics because they normalize performance to a common basis

Speedup of system 2 with respect to system 1 is the value S(2,1) such that R2=S(2,1)*R1 where R1 and R2 are the speed metrics for systems 1 and 2. Then system 2 is S(2,1) times faster than system 1

Relative change expresses performance as a percent change, relative to the performance of another system. Delta(2,1) = ((R2 - R1) / R1) * 100

Speedup and relative change, example

Example of calculating speedup and relative change using system 1 as the basis

System x	Execution time	Speedup	Relative change
1	480	1	0
2	360	1.33	+33
3	540		
4	210		

Speedup and relative change, example

Example of calculating speedup and relative change using system 1 as the basis

System x	Execution time	Speedup	Relative change
1	480	1	0
2	360	1.33	+33
3	540	0.89	-11
4	210	2.29	+129

Means versus ends metrics

Most metrics measure what was done, whether or not it was useful

What makes a metric reliable is that it accurately and consistently measures progress towards a goal (means-based metrics vs ends-based metrics). Example

```
For ( i = 1; i < N; i++) s = s+ x[i] * y[i];
```

Executes N floating point addition and multiplication operations, if time to execute one addition is `ta` and multiplication is `tm`. Total time is `T1=N(ta+tm)` cycles

Execution rate is E`R1 = 2N / N(ta+tm) = 2/(ta+tm) FLOPS/cycle`

Means versus ends metrics

But, there is no need to perform addition or multiplication for elements with value 0

```
For ( i = 1; i < N; i++)
     If ( x[i]!= 0 && y[i] !=0 ) s = s + x[i] * y[i];
```

If the conditional requires tif cycles, the total time to execute is

`T2 = N(tif + f(ta+tm)),` f is fraction of N where x[i] and y[i] are nonzero

ExecRate E`R2 = 2Nf/N(tif+f(ta+tm)) = 2f/tif+f(ta+tm) FLOPS/cycle`

Means versus ends metrics

`ER1 = 2/(ta+tm) FLOPS/cycle, ER2 = 2f/tif+f(ta+tm) FLOPS/cycle`

If tif= 4, ta=5, tm=10, f=10% and processor clock is 250 MHz (1 cycle in 4 ns). Then **t1=60N** ns and t2=N[4+0.1*(5+10)]*4ns, **t2=22N ns**

- The **speedup** for program 2 relative to 1 is S(2,1) = 60N/22N = **2.73**
- Execution rates R1= 2/60= **33 MFLOPS** and R2=2(0.1)/22 = **9.09 MFLOPS**

Even though we have reduced the total execution time from 60 to 22, the means based metric (MFLOPS) shows that program 2 is 72% slower than program 1

The ends-based metric shows that program 2 is 173% faster

Average performance and variability

Computer performance is multidimensional. It can be misleading trying to summarize it with one single number - think specialization cases

But, humans continue to want a simple way to compare different systems

The use of mean values can be useful to perform coarse comparisons. But it is important to understand how to correctly calculate an appropriate mean value and how to recognize when a mean has been calculated incorrectly or being used inappropriately

Indices of central tendency

- Sample mean. Given n different measures $E[X] = 1/n \sum x_i \quad i=1:n$
  - Gives equal weight to all measurements ~ problems with outliers
- Sample median, reduces skewing effect of outliers.
  - Found by ordering all of the n measures. The middle value is the median
- Sample mode. Is just the value that occurs more frequently
  - Best index for categorical data

Mean uses all of the sample data, but it is sensitive to outliers. Mean and Mode do not use all of the available information, but are less sensitive to outliers

Other types of means

If you decide the mean is the appropriate index of central tendency for the current situation, you must decide which type of mean to use

Characteristics of a good mean: Depending on the actual meaning of the measured values, the resulting mean value may not make any sense

- If time values are averaged together, the resulting mean should be directly proportional to the total weighted time. So if total execution time were to double, so would the value of the corresponding mean
- If a rate metric is calculated by dividing the number of operations executed by the total execution time, a mean calculated with rates should be inversely proportional to the total weighted time. If total execution time were to double, the mean of rates should be reduced to 1/2

Which mean to use?

- Assume that we have measured execution times of n benchmarks on the same system: $T_i,\ 1 \le i \le n$
- Assume total work performed by each benchmark is constant (executed F flops)
- This workload produces an execution rate for program i of $M_i = F/T_i$ flops

Arithmetic mean

- Mean execution time, $T_a = 1/n \sum T_i$
  - Result is directly proportional to the total execution time
- Mean execution rate, $M_a = 1/n \sum M_i = \sum (F/T_i)/n = F/n \sum 1/T_i$
  - Result is directly proportional to the sum of inverse execution times. But we need a value that is inversely proportional to the sum of times

Which mean to use?

Harmonic mean $x_h = n / \sum(1/x_i)$

Execution time $T_h = n / \sum(1/T_i)$

- Result is obviously not proportional to the total execution time

Execution rate $M_h = n / \sum(1/M_i) = n / \sum(T_i/F) = Fn / \sum(T_i)$

- Result is inversely proportional to the sum of times

Harmonic mean is appropriate for summarizing rate measurements

Which mean to use?

The geometric mean is the nth root of the product of the n individual xi values

$xg = (\prod x_i)^{1/n}$

It is the appropriate mean for summarizing normalized numbers and for summarizing measurements with a wide range of values, since a single value has less influence on the geometric mean than on the arithmetic mean.

It maintains consistent relationships when comparing normalized values regardless of the basis system used to normalize measurements

Which mean to use?

Program	System1	System2	System3
1	417	224	134
2	83	70	70
3	66	153	135
4	39,449	33,527	66,00
5	772	368	369
Geom. mean	587	503	499
Rank	3	2	1

Which mean to use?

Program	System1	System2	System3
1	1.0	0.59	0.32
2	1.0	0.84	0.85
3	1,0	2.32	2.05
4	1.0	0.85	1.67
5	1.0	0.48	0.45
Geom. mean	1.0	0.86	0.84
Rank	3	2	1

Normalize with respect to system 1

Which mean to use?

Program	System1	System2	System3
1	1.71	1.0	0.55
2	1.19	1.0	1.00
3	0.43	1,0	0.88
4	1.18	1.0	1.97
5	2.10	1.0	1.00
Geom. mean	1.17	1.0	0.99
Rank	3	2	1

Normalize with respect to system 2

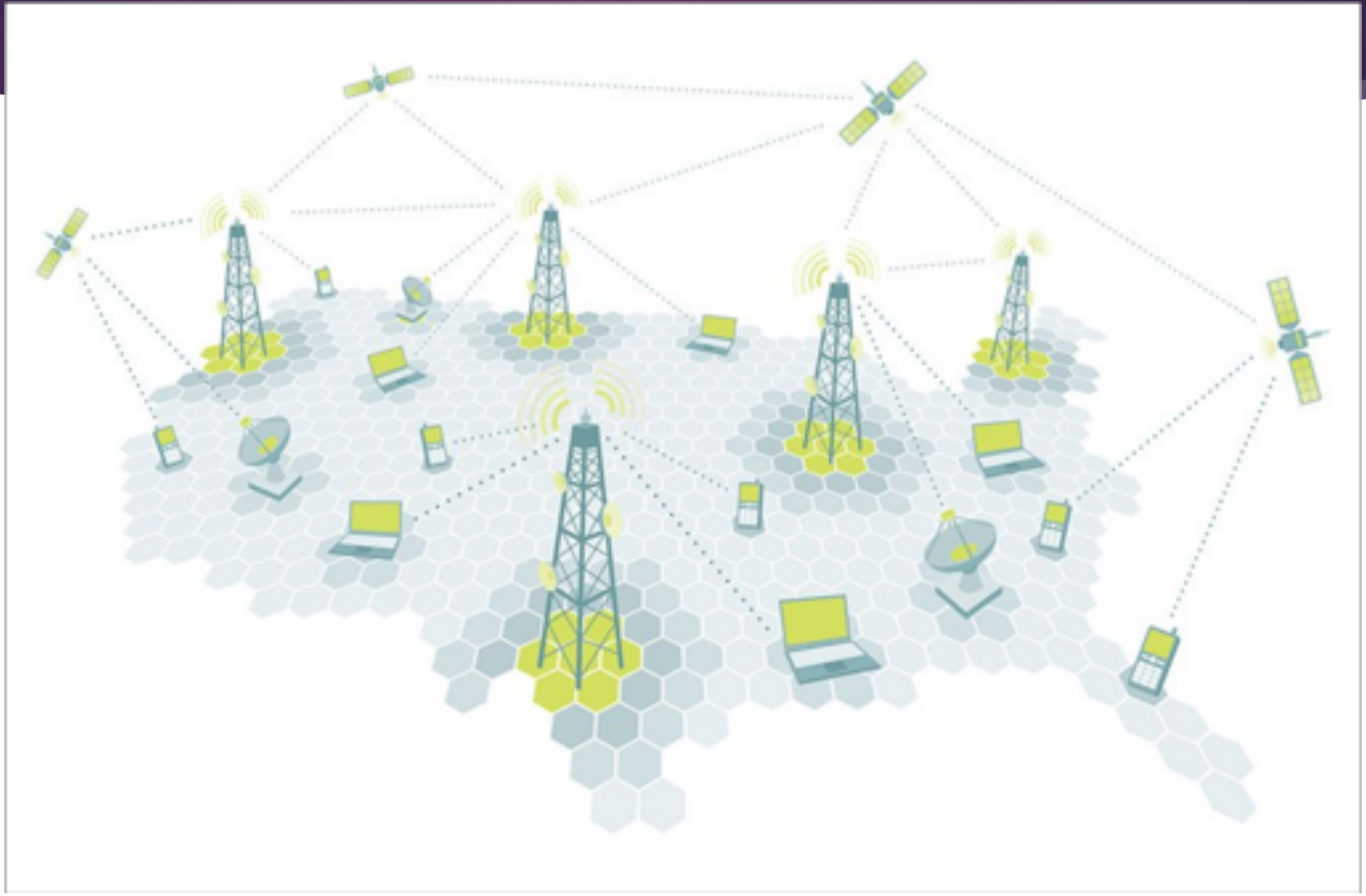
The geometric mean produces a consistent ordering of the systems being compared, but it is the wrong ordering. Geometric mean is not appropriate for summarizing times or rates

Program	System1	System2	System3
1	417	224	134
2	83	70	70
3	66	153	135
4	39,449	33,527	66,00
5	772	368	369
Total time	40,787	34,362	66,798
Rank	2	1	3

Experimental algorithms

BASED ON MATERIAL FROM: A GUIDE TO EXPERIMENTAL ALGORITHMICS, BY CATHERINE C. MCGEOCH

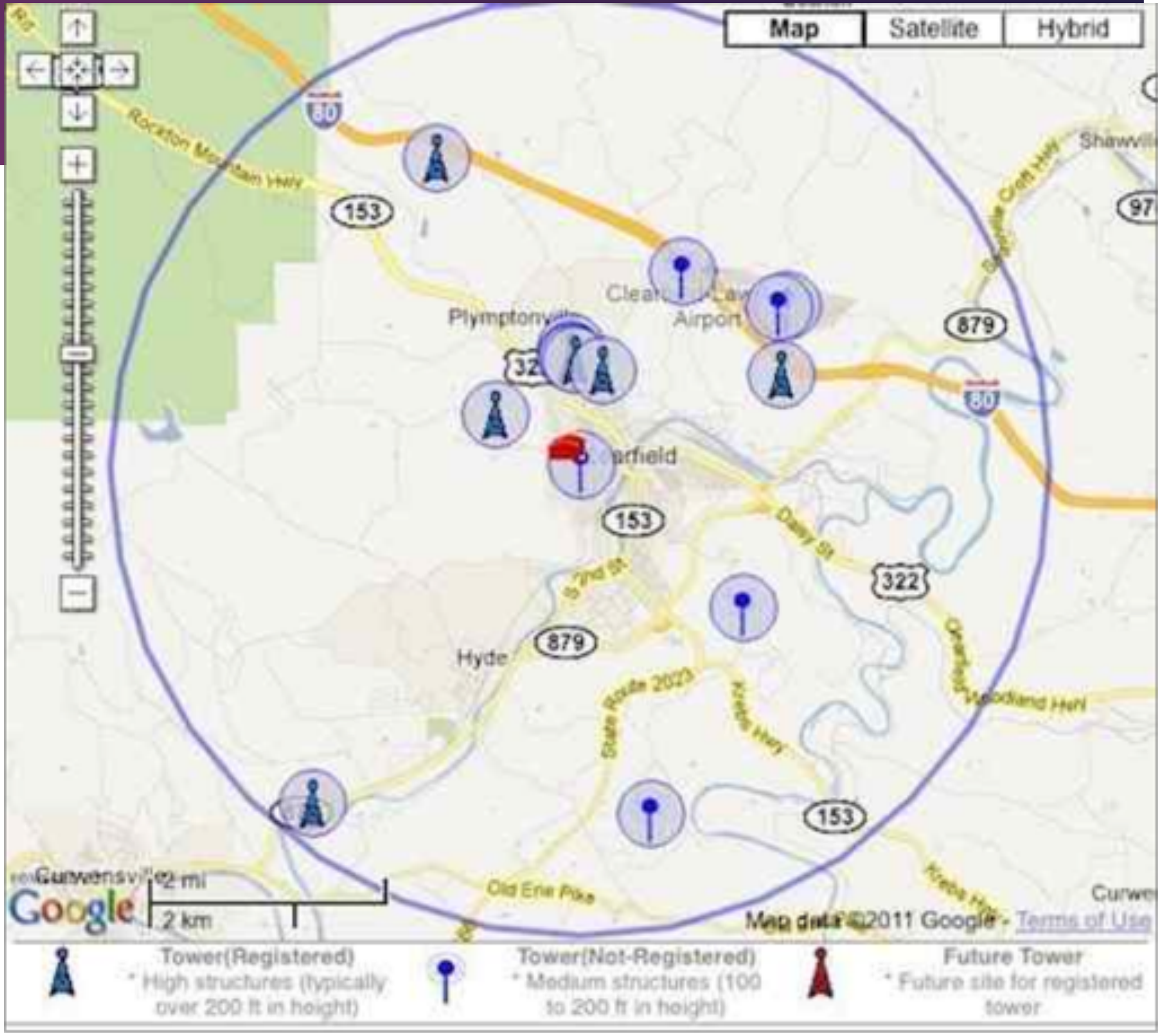
Example: telecommunications network



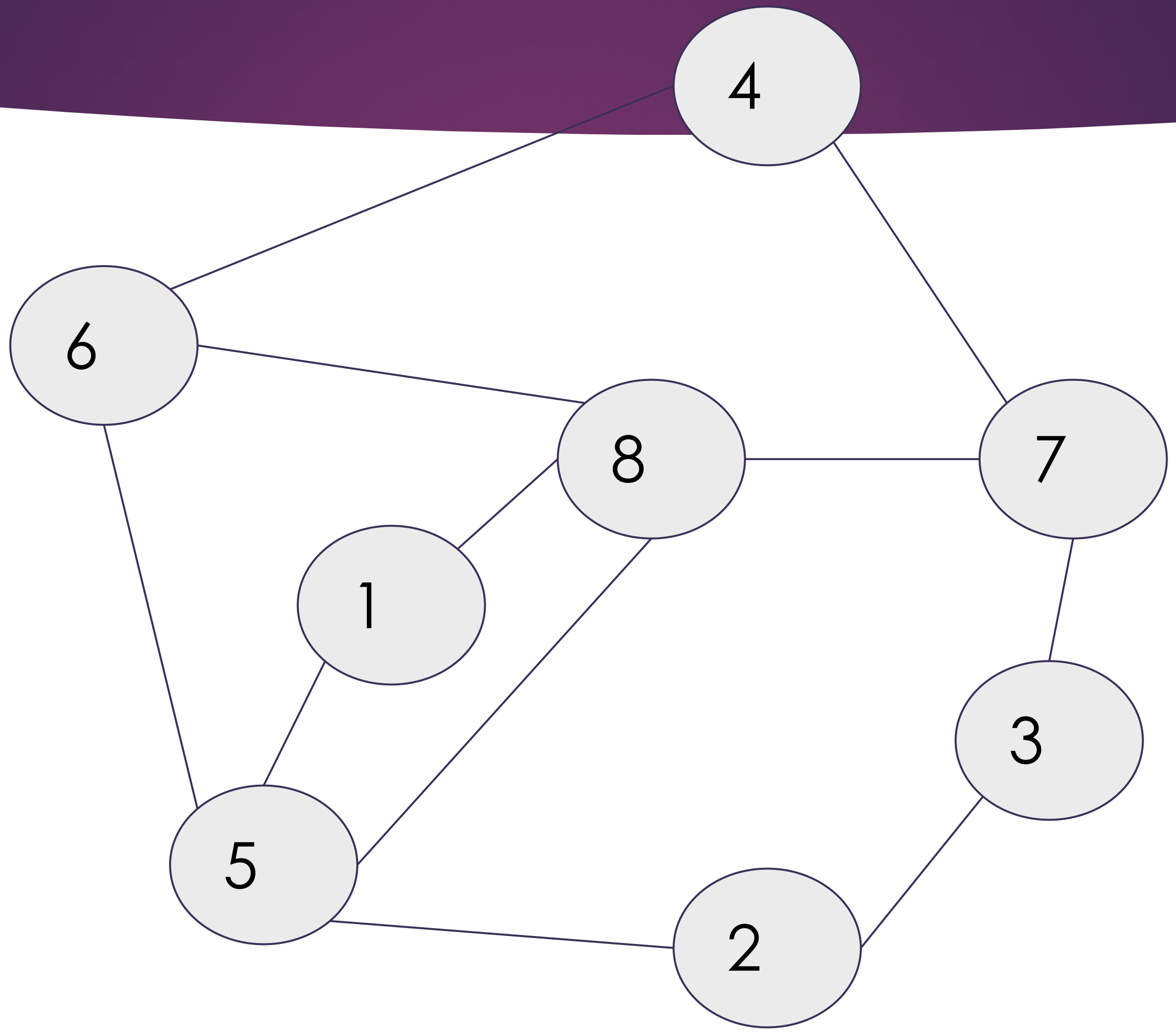
Example: problem definition

There are a number of cell towers in a given region

Towers within a radius need to be assigned a different broadcast frequency, so that nearby towers do not interfere with one another


Map
Satellite
Hybrid
Plymptonville
Clearfield Airport
Clearfield
Hyde
Curwensville
Tower(Registered)
* High structures (typically over 200 ft in height)
Tower(Not-Registered)
* Medium structures (100 to 200 ft in height)
Future Tower
* Future site for registered tower
2 mi
2 km
Map data ©2011 Google - Terms of Use

Example: can be modeled as a graph



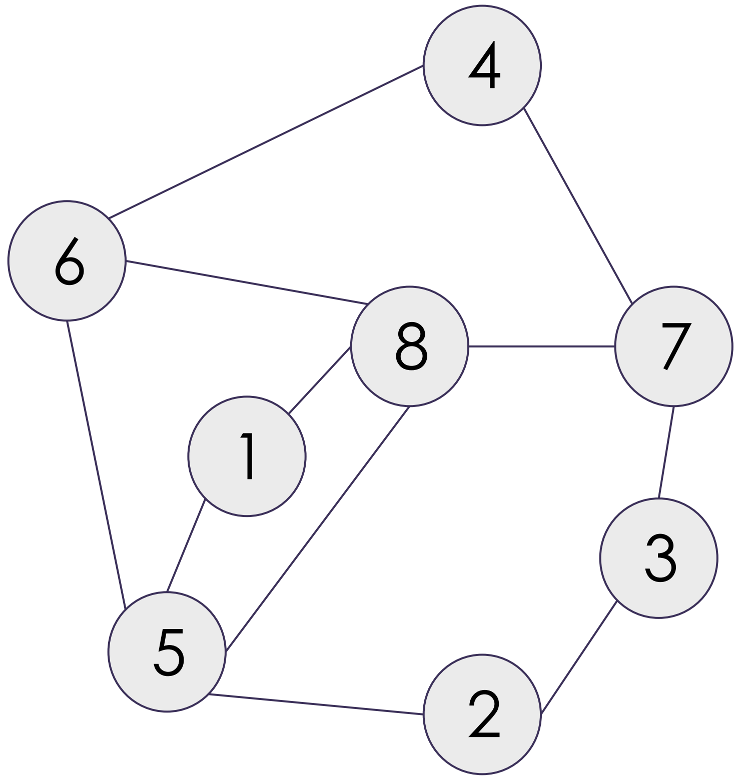
Greedy approach

```
Greedy(G)
    For (v=1; v<=n; v++)
        For (c=1; c<=m; c++)
            If (G.checkColor(c,v))
                G.assignColor(c,v)
                Break
    Return G.coloring
```

Graph coloring, greedy

V = {1, 2, 3, 4, 5, 6, 7, 8}

c = { red, yellow, green, blue, magenta, cyan }

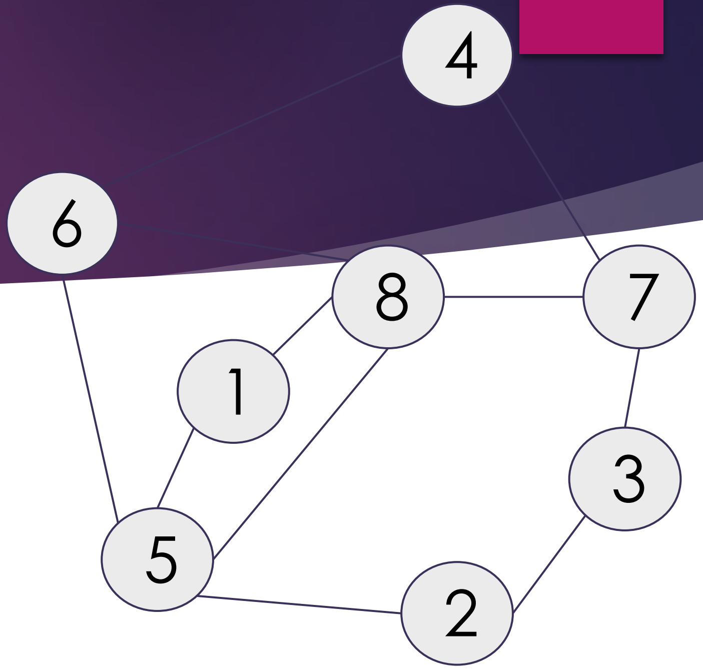


Graph coloring, greedy

V = {1, 2, 3, 4, 5, 6, 7, 8}

c = { red, yellow, green, blue, magenta, cyan }

Color count = 4



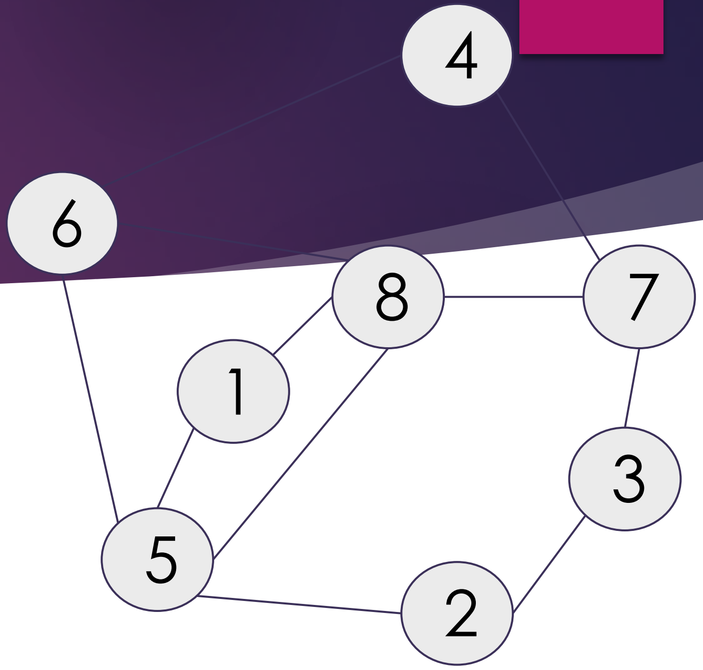
1	2	3	4	5	6	7	8
red	red	yellow	red	yellow	green	green	blue

Graph coloring, greedy

V = {8, 7, 6, 5, 4, 3, 2, 1}

c = { red, yellow, green, blue, magenta, cyan }

Color count = 3



8	7	6	5	4	3	2	1
red	yellow	yellow	green	red	red	yellow	green

Random approach

```
Random(G,I)

    For (i=1; i<=I; i++)

        coloring = Greedy(G.randomVertexOrder())

        If ( coloring.colorCount < bestCount)

            bestCount = coloring.colorCount

            bestColoring = coloring

    Return bestColoring
```

Questions

How much time do these approaches take on average, as a function of `vertices, edges, I`?

Are they competitive with state of the art GC algorithms?

On what types of inputs are they most and least effective?

How does `I` affect the trade-off between time and color count in Random?

What is the best way to implement `G.checkColor` and `G.assignColor` ?

Experimental design

Experimental design is a plan for an experiment that targets a specific question

The design specifies what properties to measure, what input classes to incorporate, what input sizes to use, etc.

Terminology

Performance metric: A dimension of algorithm performance that can be measured, such as time, solution quality, space usage, accuracy, precision, recall, sensitivity, specificity, etc.

Performance indicator: A quantity associated with a performance metric that can be measured in an experiment.

Parameter: Any property that affects the value of a performance indicator.

- Algorithm parameter
- Instance parameter (or problem parameter)
- Environment parameter

Terminology

Factor: a parameter that is explicitly manipulated in the experiment

Level: a value (range or subset) assigned to a factor in an experiment

Design point: a particular combination of levels to be tested (combinatorial design)

Trial or test: one run of the test program at a specific design point, which produces a measurement of the performance indicator (the design may specify some number of trials at each design point to account for variance)

Fixed parameter: a parameter held constant through all trials

Noise parameter: a parameter with levels that change from trial to trial in some uncontrolled or semi-controlled way

Selecting input instances

Input instances may be collected from real-world application domains or constructed by generation programs

Stress-test inputs: are meant to invoke bugs and reveal artifacts found in boundary conditions and presenting easy to check cases

Worst case inputs: may be hard or expensive to solve. Used to assess performance boundaries

Random inputs: typically controlled by a small number of parameters and use random number generators to fill in the details

Selecting input instances

Structured random inputs:

- *Algorithm centered generators:* built with parameters that exercise algorithm mechanisms
- *Reality centered generators:* capture properties of real-world inputs

Real instances: collected from real world applications. It may be difficult to collect enough samples for thorough testing

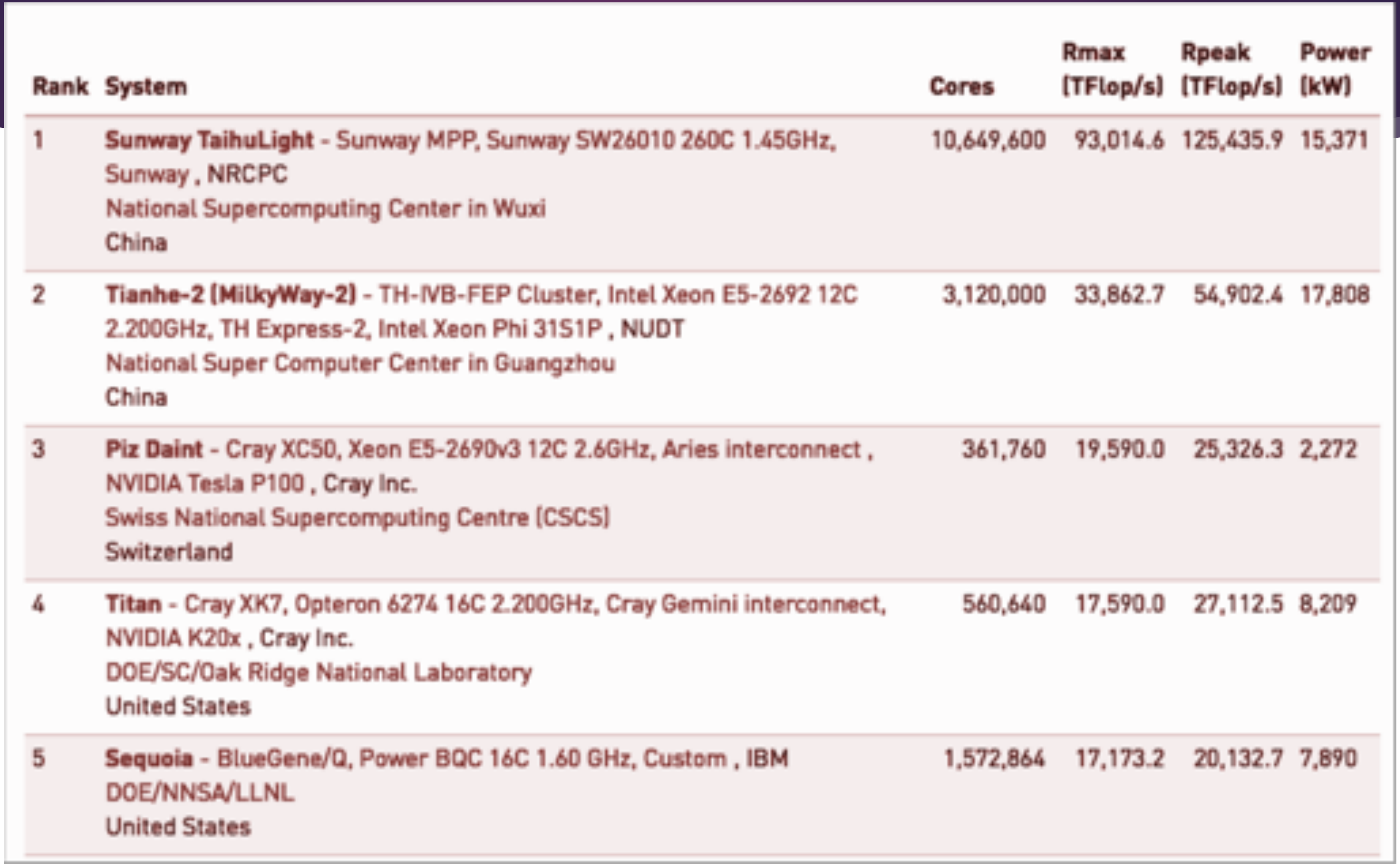
Hybrid instances: combine real-world structures with generated components

Benchmarks and testbeds: produce results that are directly comparable to others

HPC benchmarks

1. HPL - the Linpack TPP benchmark which measures the floating point rate of execution for solving a linear system of equations.
2. DGEMM - measures the floating point rate of execution of double precision real matrix-matrix multiplication.
3. STREAM - a simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel.
4. PTRANS (parallel matrix transpose) - exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.

Top 500



Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

Graph 500 benchmark: search and shortest path

1. Construct a graph from the edge list (timed, kernel 1).
2. Randomly sample 64 unique search keys with degree at least one, not counting self-loops.
3. For each search key:
    1. Compute the parent array (timed, kernel 2).
    2. Validate that the parent array is a correct BFS search tree for the given search tree.
4. For each search key:
    1. Compute the parent array and the distance array (timed, kernel 3).
    2. Validate that the parent array/distance vector is a correct SSSP search tree with shortest paths for the given search tree.
5. Compute and output performance information.

Graph 500

There are six problem classes defined by their input size:

- Toy 17GB or around 1010 bytes, which we also call level 10,
- Mini 140GB (1011 bytes, level 11),
- Small 1TB (1012 bytes, level 12),
- Medium 17TB (1013 bytes, level 13),
- Large 140TB (1014 bytes, level 14), and
- Huge 1.1PB (1015 bytes, level 15).

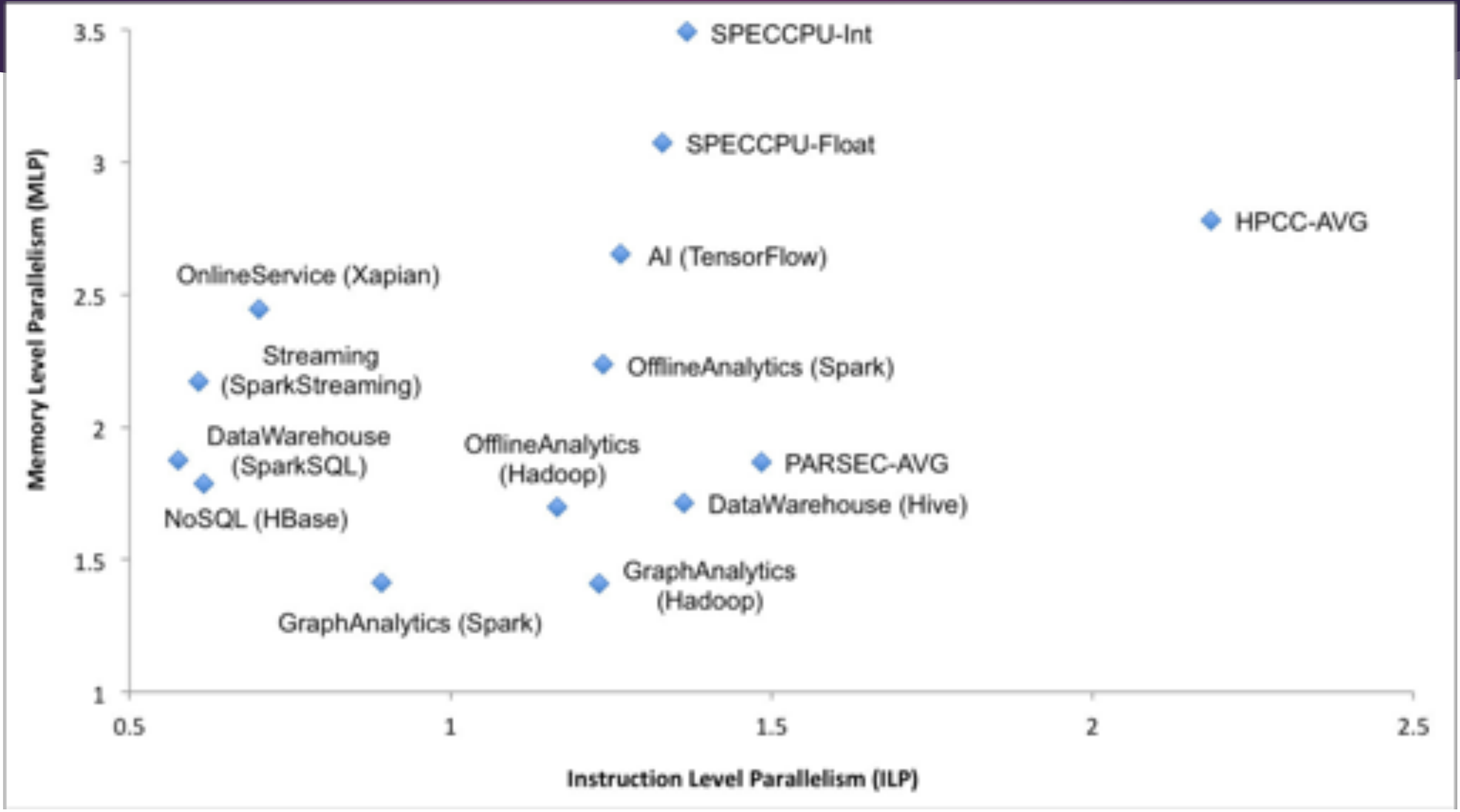
Selecting input instances

The choice of input instances should reflect general experimental goals

To meet goals of correctness and validity:

- Use stress-test inputs and check that random generators really generate instances with the intended properties
- Use pilot experiments to identify, and remove from consideration, instances that are too easy or too hard to be useful for distinguishing competing algorithmic ideas (recall floor and ceiling effects)

Big Data benchmarks



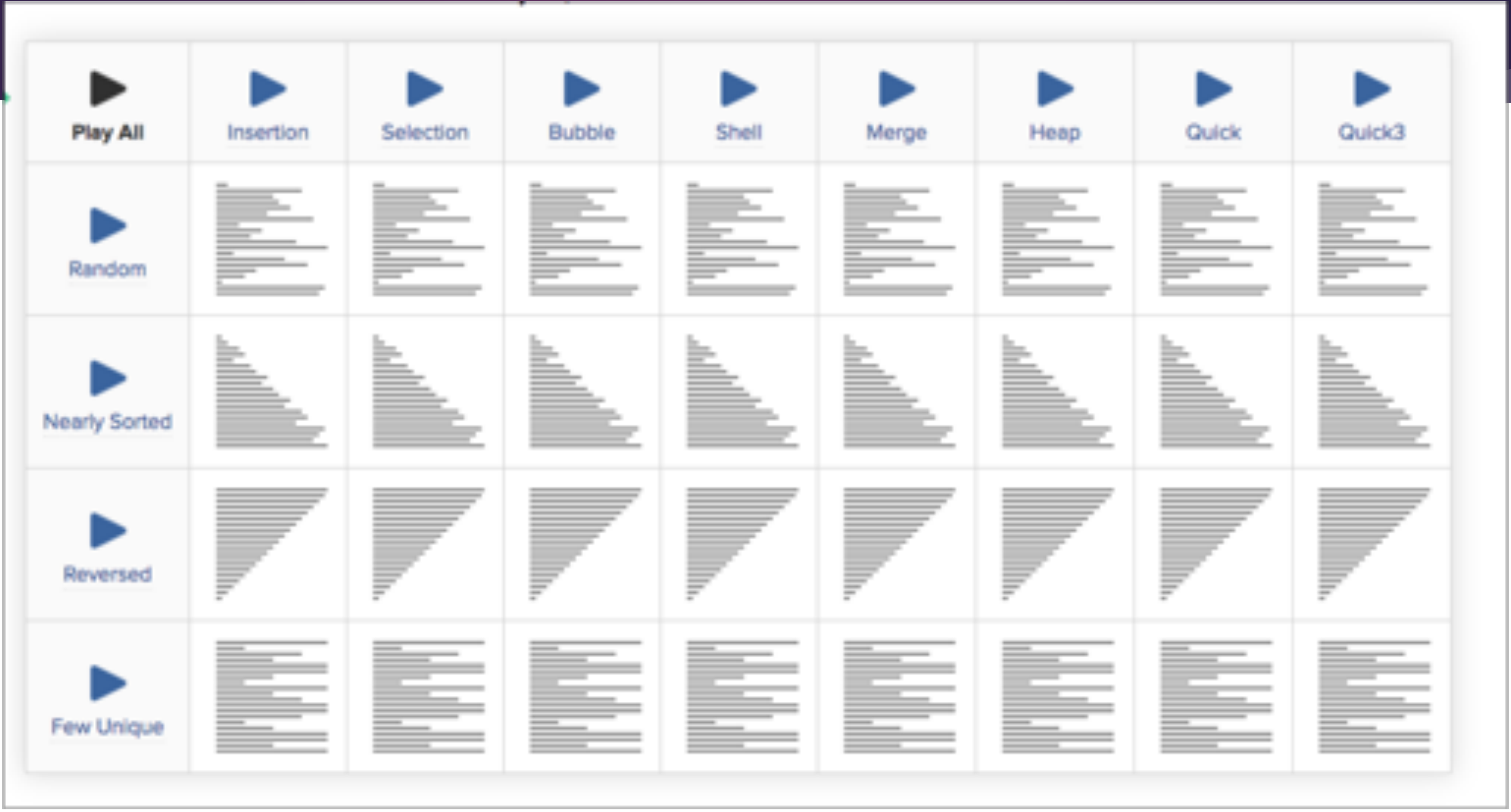
Selecting input instances

For general results incorporate good variety in the set of input classes tested. But avoid variety for variety's sake: consider how each class contributes new insights about performance.

- Worst case instances provide general upper bounds
- Random generators that span the input space can reveal the range of possible outcomes
- Real world instances from application hot spots can highlight properties of interest to certain communities

More ambitious analysis tend to require more general input classes and tight control of parameters.

Comparison of sorting algorithms



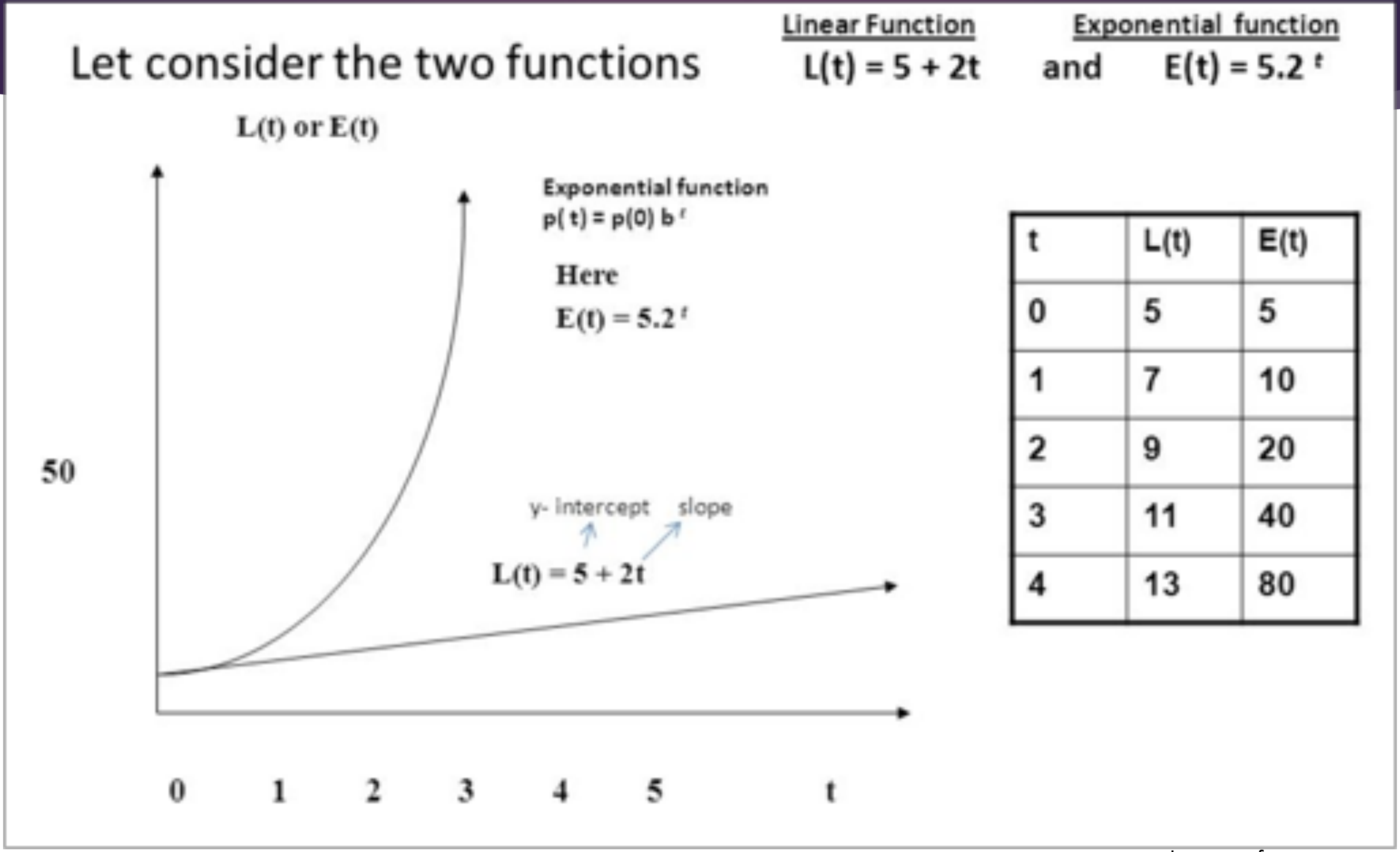
***Guideline:** Choose input classes to support goals of correctness and generality, and to target the question at hand*

Choosing factors and design points

The motivating question in an algorithmic experiment typically falls into one of these four broad categories:

1. Assessment: These experiments look at general properties, relationships, and ranges of outcomes (e.g. find performance bottlenecks, find input properties that affect performance).
2. The horse race: This type of experiment looks for winners and losers in the space of implementation ideas
3. Filling functions: This experiment start with a functional model that describes some cost property and aims to fill in the details (e.g. finding coefficients)
4. Modeling: These experiments are concerned with finding the correct function family to describe a given cost (e.g.is cost linear, exponential, logarithmic?)

Function growth



Let consider the two functions **Linear Function** $L(t) = 5 + 2t$ and **Exponential function** $E(t) = 5.2^t$

t	L(t)	E(t)
0	5	5
1	7	10
2	9	20
3	11	40
4	13	80

Image from
http://slideplayer.com/slide/3481392/

Guideline: *Choose as factors these parameters that are most important to performance, fix the parameters that are least relevant to performance, and let the other parameters vary.*

General assessment

The goal is to identify promising algorithm designs.

Choose performance indicators and factors to highlight the differences between options. It is helpful to determine questions to ask.

E.g., If the task is to compare data structures A, B, C, choose a performance indicator that is common to all three and changes the most when a data structure is substituted

Get a rough idea of the functional relationship between key parameters (especially input size) and algorithm performance.

Doubling experiments

Strategy for quick assessment of function growth: use doubling experiments

(n, 2n, 4n, 8n ..)

If measurements do not change with n, C(n) is constant

If costs increment by a constant as n doubles, then C(n) is O(log n)

If costs double as n doubles, C(n) is linear

To determine if C(n) in O(n log n), divide each measurement by n and check whether the result C(n)/n increments by a constant

If cost quadruples each time n doubles, c(n) in O(n^2)

Assessing convergence

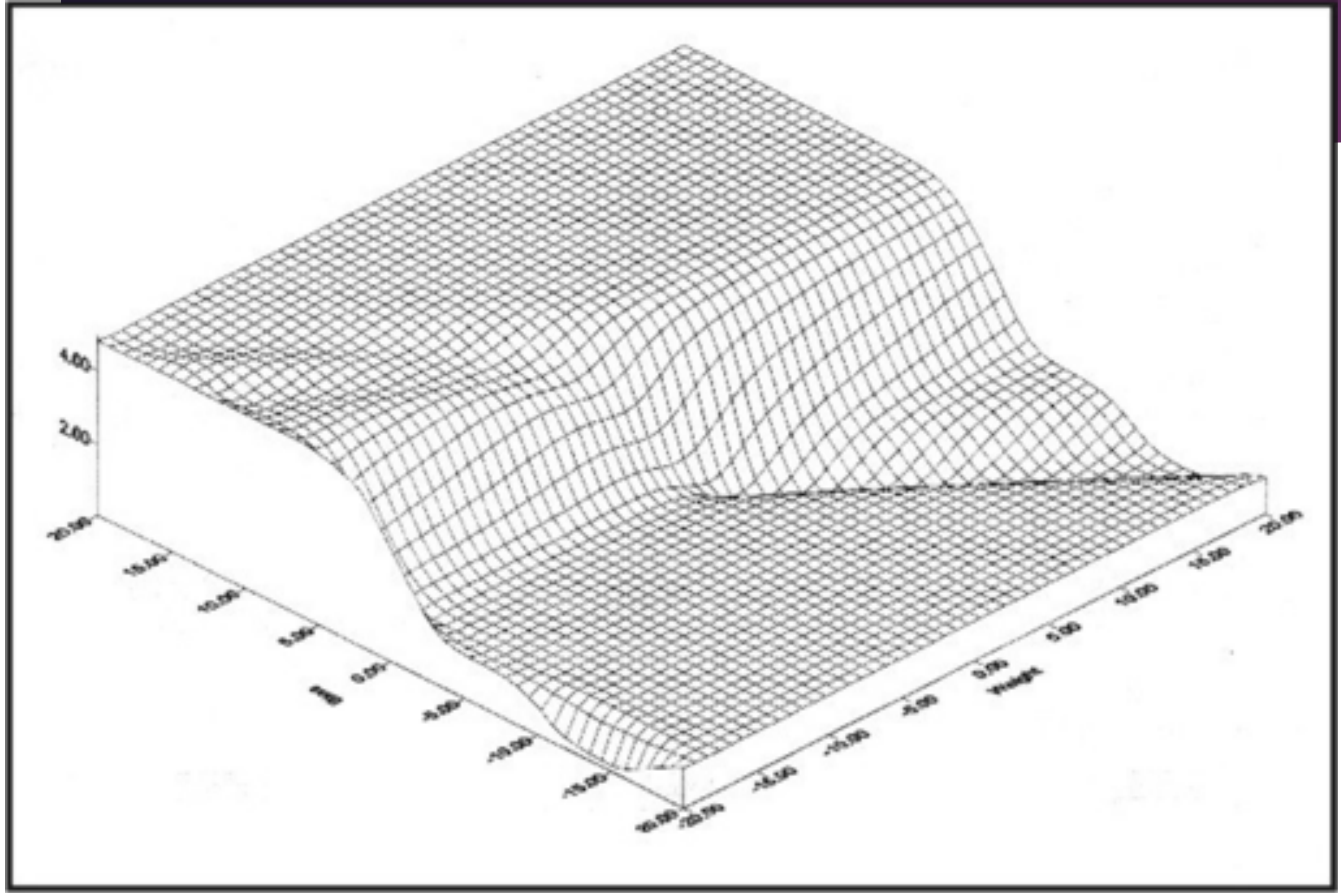
Determine when an algorithm has converged (i.e., the probability of improving / changing further is too small to be worth continuing searching/iterating/optimizing)

A stopping rule is a condition that halts the algorithm

A poorly chosen stopping rule either wastes time or stops the algorithm prematurely

Stopping rules are automatic, platform independent, and usually consider the relative change in cost for two or more iterations

Assessing convergence



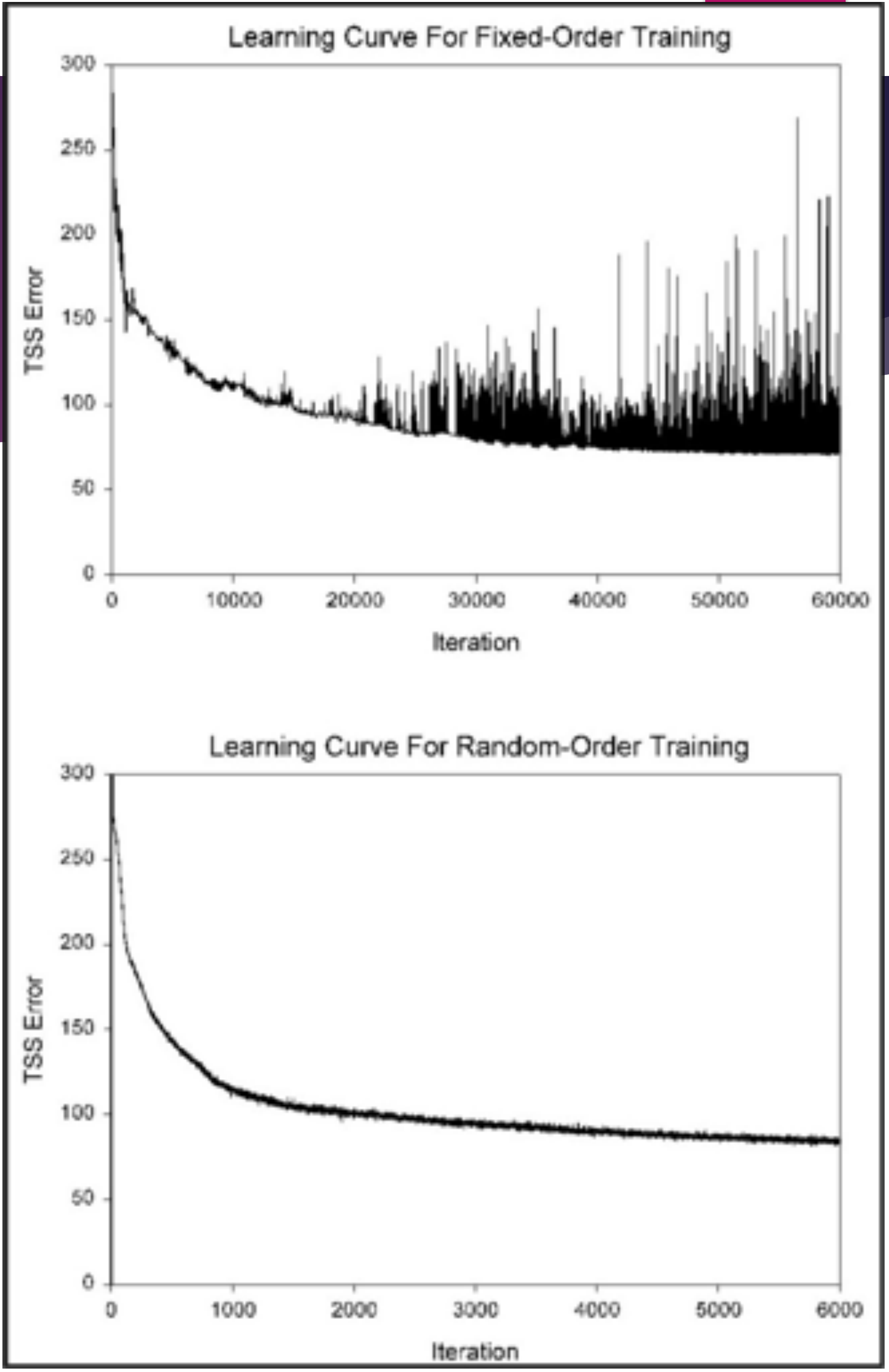


Image from
http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

Guideline: *The problem of analysing a multidimensional function can be simplified by focusing on a small number of one-dimensional functions, ideally with similar shapes*

Guideline: *To study trends and functions, choose design points that exploit what you already know*

Factor analysis (how to reduce experimental designs)

Merge similar factors: if 2 factors have similar effect on performance, treat them as one by restricting the experiment to just the design points ++ and -- (omitting +- and -+)

Use trace data to infer effects of omitted factors

Convert factors to noise parameters: instead of explicitly setting levels for a factor, let the levels vary according to a simple probability distribution

Limit the scope of the experiment by fixing some factors or reducing the number of levels

Guideline: *Full factorial design maximize the information gained from one experiment*

Guideline: *When the experimental design is too big, apply factor reduction strategies to reduce the size of the design with least damage to generality*

Some more guidelines

Leverage the pilot study and the literature to create better experiments

Never assume! Design experiments with built in safeguards against bugs and biases, and be sure you can replicate your own results

Experimental efficiency depends on the speed of the test program, the usability of the test environment, the quality of data returned, and the generality of conclusions

When comparing algorithm design options, choose performance indicators and factors to highlight the differences among the options being compared

Try a doubling experiment for a quick assessment of function growth

Experimental design template

- Question
- Performance indicators
- Factors
- Levels
- Trials per design point
- Design points
- Outputs