COMP114
Experimental Methods in Computing

Methodologies for Experimental Design

Applications of Experiments in Computing

Testing
(Program Correctness)
Evaluation
(Programs, Algorithms, Systems)

Applications of Experiments in Computing

- These are different activities –
  Different aims
  Distinct methodologies
- This module will mainly deal with experimental issues as arising in “evaluation”.
- For completeness, a brief overview of the main distinction between “testing” and “evaluating” is given.

Experiment design for Testing

Qn: What does “testing a program” aim to achieve?

- Consider a ‘typical’ program, P, e.g. solution to COMP101 exercise.
- Programs are created in order to “do something”
- This aspect raises several questions.

Experiment design for Testing

Q: How is it shown that P does what it should?

A: A description (*specification*) of how the program should behave has been given. If the program "meets the specification" then it is a "correct program".

- Notice that (unless explicitly indicated in the specification) *performance criteria* (run-time, etc.) are not relevant.

Testing Summary

- Program outputs results for all inputs.
- Output returned is what is required by specification.
- Program treats "extreme" or "incorrect" data in a "robust" manner.

➢ Describe minimum levels of testing.

Where do "experiments" come in?

- Input/Output behaviour.
- Checking execution paths.
- ➢ Reason 1: in "non-trivial" cases it is not feasible to check that *every* valid input is treated correctly.
- ➢ Reason 2: it is not feasible to check *every* sequence of steps a program might execute.
- ➢ One solution: perform a number of *experimental tests* with the aim of finding *errors.*
- ➢ Note: Testing can prove that errors are *present*; it *cannot* show they are *absent*.

Experiment design for Evaluating

Qn: How do we deal with assertions such as the following?

a. P is a "better solution" than Q.
b. P produces "good quality" results.
c. Typical users would feel "more comfortable" using P than Q.

Examples

a. One program produces results quicker than another for the same task, e.g. sorting a list of numbers into order.
b. A program for scheduling deliveries of packages to widely separated locations produces schedules whose overall delivery time is minimised.
c. Users prefer Mozilla as a web browser to Internet Explorer or Netscape Navigator.

Evaluation

- The nature of such assertions is quite different from that of “P is correct”.
- These make claims about “quality” of solutions.
- While (a) and (b) may *sometimes* be dealt with by formal analysis – e.g. counting number of operations for (a) – this may be extremely hard in general.
- Claims such as (c) can not be dealt with by any type of “rigorous mathematical proof”.

Experiment to evaluate solutions.

- General approaches

a. Conduct a number of test runs of both programs using identical data. Monitor “time” taken by each.

b. Examine the delivery schedules produced by the program when it is tested on cases for which the “best schedule” is already known.

c. Construct a questionnaire to be completed by users and analyse the responses from an “unbiased” sample of these, e.g. standard opinion poll methods.

Experiment to evaluate solutions.

- Notice that the techniques in dealing with (c) are quite different from those in (a) and (b).
- In Computing fields such as Human-Computer Interaction, the design of such experiments is an important area of study.
- In contrast, experimental studies for (a) and (b) have several features in common.
- We will, in general, deal with experiments related to such “quantitative” issues.

Experimental Methodology – Review

- Rationale – reason for experiment?
- Repeatable – similar results each time?
- Data used – is this “realistic”?
- Analysis of outcome – is this “correct”?
- Scalability – outcome for “large” data?

Rationale for experiment

- Experimental studies aim to present evidence supporting *hypotheses*.
- For example,

That an algorithm is efficient

That its outputs are good solutions

That it “does better” than alternatives

and so on …

Rationale for experiment

- The *reasons* for conducting the experiment must be clearly understood.
- Otherwise,

  Its results may be misinterpreted

  Data used may be unsuitable

  Inappropriate measures may be used
- Careful formulation of the hypothesis being tested is a key aspect of experiment design.

What next? – experimental runs.

- Given a program, P, and hypothesis, H, about P
- the accuracy of this hypothesis is to tested by running a set of experiments.
- In doing this, data on which the program will be assessed must be chosen.

Choice of experimental data.

- Some options –

  Exactly one input case

  Many different inputs (but all same ‘size’)

  Generate random data

  “benchmark” problems

Choice of experimental data.

- Little can be concluded about how well the experiment supports the hypothesis in the first case.
- The second could be reasonable if the hypothesis tested deals with exactly such data.
- The final two methods are often used.

Experimental data – benchmarks

- So called *benchmarks* offer collections of data with which the performance of different approaches to the same problem can be compared.
- Benchmark suites often include data sets that have proven to be particularly challenging.
- This provides a useful standard for assessing performance of algorithms for “hard” computational problems.

Experimental data – random data

- Motivated by idea that “typical program behaviour” – i.e behaviour “on average” – should be estimated using “typical data sets”.
- “typical data” = “generated randomly”
- Widely used as approach to evaluating proposed solutions.

Random data – important issues

- "random" (statistical sense) is not always the same as "typical" (form most likely to be seen in practice).
- Constructing random data sets (with various properties) may be non-trivial.
- Ensuring results obtained are not just "coincidences" – i.e the notions of "significance" and "confidence"
- We will look in more depth at these later in the module.

Analysing Results of Experiment

- Key question: do the results support or conflict with the hypothesis the experiment was set up to investigate?
- "yes" – how is this justified?, how should results be presented convincingly?, were "enough" trials carried out?, etc
- "no" – do the results obtained suggest an alternative hypothesis to test?

Presentation of results

- Describe exact hypothesis considered.
- Method used to investigate it.
- Basis for concluding hypothesis is "reasonable" given the results obtained by experiment.

Presentation of results

- For example –
- Graphical methods

  Bar-charts, Plots

  e.g. for hypotheses about run-time,

  X-axis :– range of data sizes

  Y-axis :– run-time (average/worst/best)
- If random data used then full detail needs to discuss – number of trials, statistical features, how data was generated, justification that "random generation" methods are "reasonable"

An example – random ordering

- Recall from COMP109 that a *permutation* of the numbers

<1,2,3,…,n>

is an ordering of these as

<p1,p2,p3,…,pn>

with each number between 1 and n occurring exactly once. For example,

<3,5,1,4,2,6>

is a permutation of

<1,2,3,4,5,6>

Example continued

- One component used often in experimental studies is a method to carry out the following task:

  Random Permutation

  Input: n (a positive integer)

  Output: [r1,r2,r3,….,rn] (a *random* permutation of [1,2,3,…,n].

Example

- Suppose we have a method which (given n) returns a random integer between 1 and n. For example,

  1 + Sequence.nextInt(n)

  [where Sequence is an instance of the class Random() in java.util.Random]
- How can we use this to construct a method for Random Permutation?

Method 1

1. Given an array int[ ] P = new int[n] with P[i]=i+1 for each i between 0 and n-1:

a. Choose a random integer k between 0 and n-1;

b. If k was already chosen go back to (a)

c. Otherwise make k the next value in the permutation *and* “mark” k as used.

d. Continue from (a) until all values used.

Method 2

1. Given an array int[ ] P = new int[n] with P[i]=i+1 for each i between 0 and n-1:

a. Let m be the number of values still to be chosen – m = n, initially.

b. Choose a random integer k between 0 and m-1;

c. Swap P[k] with P[m-1] and decrease m by 1; (m--)

d. If m>0 then repeat from (a).