ENGINEERING TRIPOS PART IIA

ELECTRICAL AND INFORMATION SCIENCES TRIPOS PART I

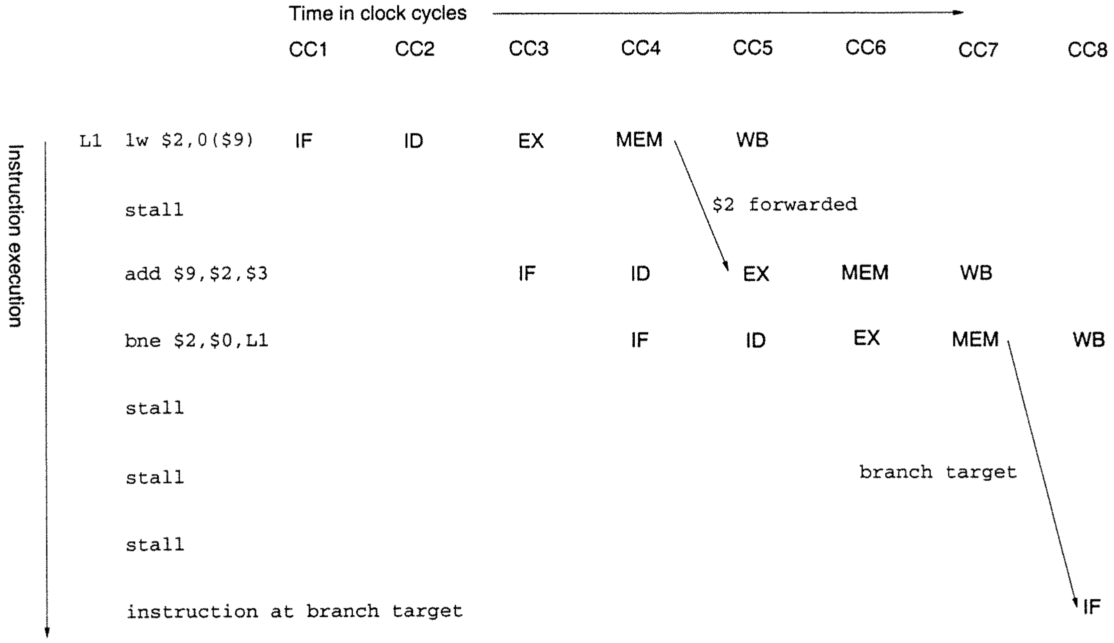
May 2000

Solutions to Paper E6

COMPUTING SYSTEMS

1 (a) There is a data hazard between the `lw` and `add` instructions, and a branch hazard associated with the `bne` instruction. As the diagram below shows, the data hazard incurs a single stall with data forwarding, while the branch hazard incurs three stalls.

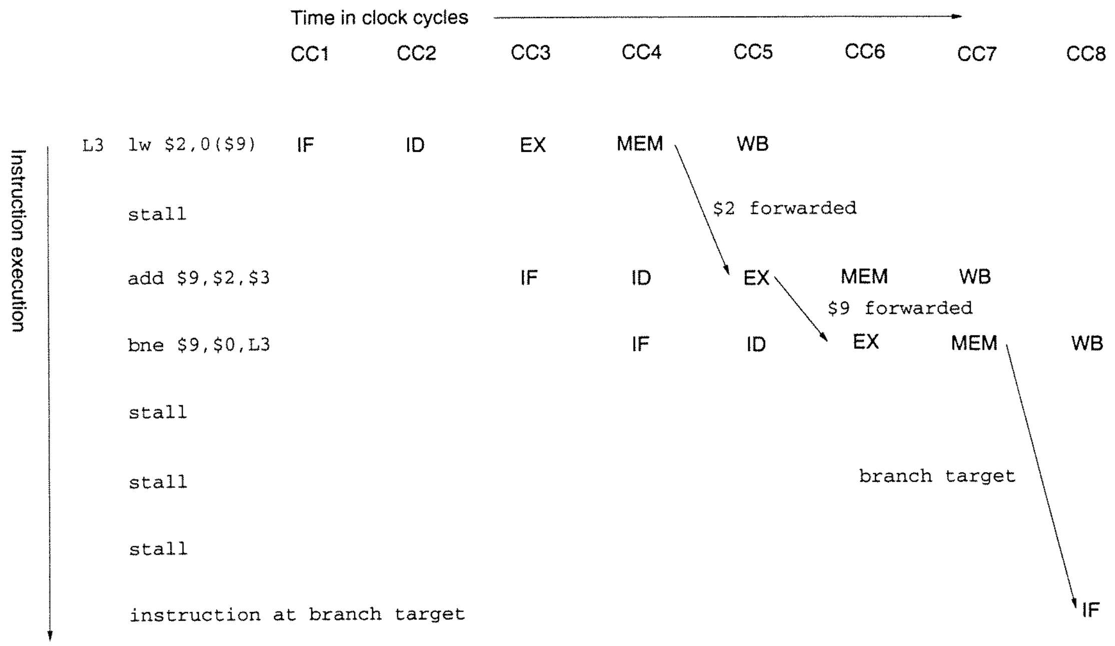
The term "hazard" is used to describe dependencies between instructions which can disrupt the operation of a pipelined datapath. "Data hazards" occur when an instruction requires data before a previous instruction has written it to the register file (or memory). "Branch hazards" occur when the address of the next instruction is required (for instruction fetching) before an earlier conditional branch instruction has been evaluated.



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
L1 lw $2,0($9)	IF	ID	EX	MEM	WB			
stall								
add $9,$2,$3			IF	ID	EX	MEM	WB	
bne $2,$0,L1				IF	ID	EX	MEM	WB
stall								
stall								
stall								
instruction at branch target								IF

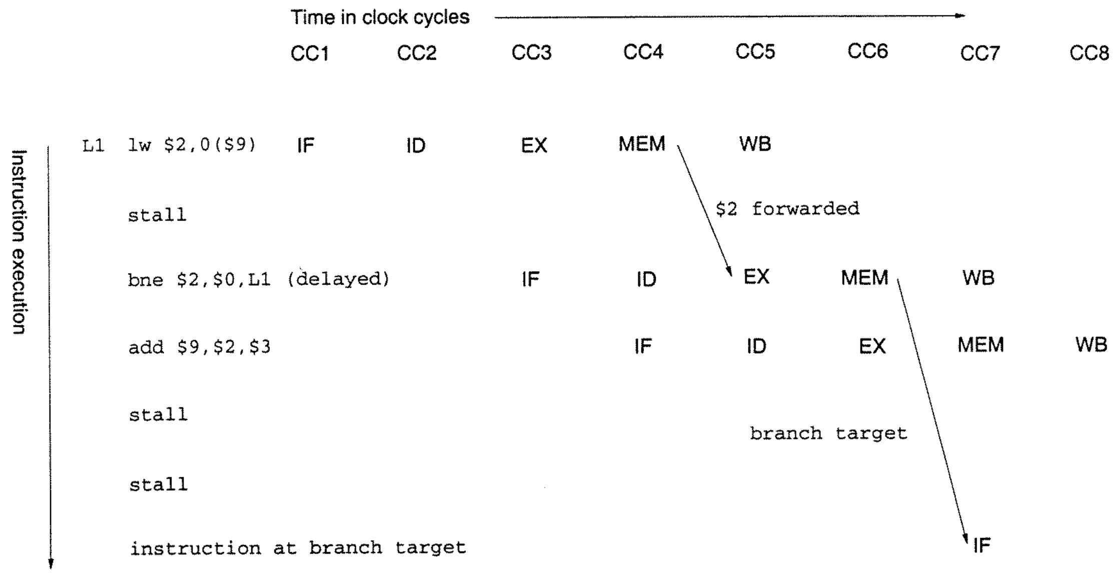
There are therefore 4 stalls in total, and 3 instructions, so the code segment takes $7n$ clock cycles to execute, plus 4 more at the end of the loop for the pipeline to clear.

(b) There are data hazards between the `lw` and `add` instructions, and between the `add` and `bne` instructions. There is also a branch hazard associated with the `bne` instruction. As the diagram below shows, the data hazard following the `lw` incurs a single stall with data forwarding, while the branch hazard incurs three stalls. No stalls are required for the other data hazard with data forwarding.


Time in clock cycles
CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8
Instruction execution
L3 lw $2,0($9) IF ID EX MEM WB
stall
$2 forwarded
add $9,$2,$3 IF ID EX MEM WB
$9 forwarded
bne $9,$0,L3 IF ID EX MEM WB
stall
stall
branch target
stall
instruction at branch target IF

There are therefore 4 stalls in total, and 3 instructions, so the code segment takes $7n$ clock cycles to execute, plus 4 more at the end of the loop for the pipeline to clear.

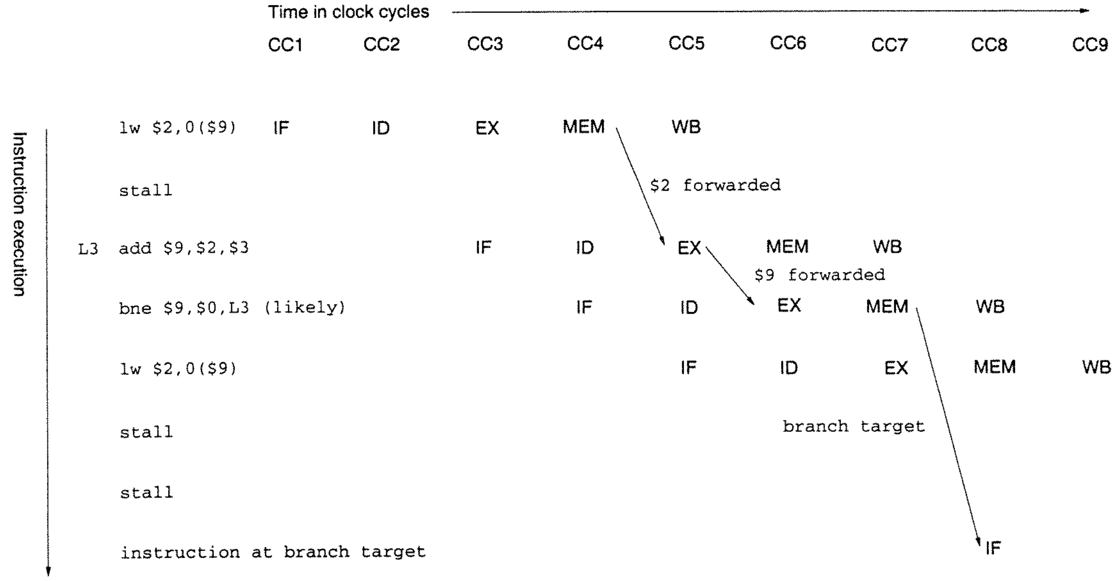
(c)


Time in clock cycles
CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8
Instruction execution
L1 lw $2,0($9) IF ID EX MEM WB
stall
$2 forwarded
bne $2,$0,L1 (delayed) IF ID EX MEM WB
add $9,$2,$3 IF ID EX MEM WB
stall
branch target
stall
instruction at branch target IF

A smart compiler could reorder the instructions to schedule an independent instruction in the branch delay slot. For the code segment in (a), the add instruction could occupy the branch delay slot. The number of stalls is reduced to three, so the code segment takes $6n$ clock cycles to execute, plus 4 more at the end of the loop for the pipeline to clear.

A similar optimization is not possible with the code segment in (b), since the **bne** instruction uses the result of the add instruction: these two instructions cannot be interchanged.

(d) A smart compiler could take advantage of the *branch likely* instruction to optimize the code in (b) as follows:



		CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
	lw $2,0($9)	IF	ID	EX	MEM	WB				
	stall									
L3	add $9,$2,$3			IF	ID	EX	MEM	WB		
	bne $9,$0,L3 (likely)				IF	ID	EX	MEM	WB	
	lw $2,0($9)					IF	ID	EX	MEM	WB
	stall									
	stall									
	instruction at branch target								IF	

The `lw` instruction from the *next* iteration of the loop occupies the branch delay slot. As long as the branch is taken, the `lw` will execute as usual. When the branch is not taken (after n iterations), the `lw` will be flushed from the pipeline after the EX stage. Note how the usual stall following the `lw` has been absorbed into one of the branch stalls. There are now only 3 instructions and 2 stalls inside the loop. The code segment will therefore take $5n$ clock cycles to execute, plus 2 for the start-up code (the first `lw` and its stall), plus another 4 at the end of the loop for the pipeline to clear.
Examiners comments: This was a popular question. It tested the candidates understanding of pipeline hazards and data forwarding. Some candidates understood the basic concepts but could not apply them. Other candidates were more successful in solving the particular problem required. The question produced a good spread of marks.

2 (a) [Bookwork] SIMD stands for *single instruction stream, multiple data streams.* A SIMD parallel architecture comprises a number of simple processors which execute identical instructions with different data under the control of a central sequencer. SIMD machines are effective only when dealing with *data parallel* tasks.

MIMD stands for *multiple instruction streams, multiple data streams.* A MIMD parallel architecture comprises a number of processors, each of which fetches its own instructions and operates on its own data. This is the most general form of parallelism. The processors are often off-the-shelf microprocessors.

The SIMD model is not currently popular as a general-purpose multiprocessor architecture, for two main reasons. First, it is inflexible: many potential applications do not exhibit data parallelism. Secondly, SIMD machines use special-purpose processor units and cannot take advantage of the significant performance and cost advantages of mass-market microprocessor technology.

In contrast, small, bus-connected MIMD machines incorporating off-the-shelf microprocessors are extremely flexible and cost effective. Recent mass-market microprocessors come with much of the logic for bus snooping built-in. Larger MIMD machines can also use off-the-shelf processors interconnected through a suitable network.

SIMD architectures are still first choice for certain special-purpose applications which are highly data parallel and require a limited set of operations. For example, many high-performance 3D computer graphics cards include an element of SIMD parallel processing: each processor performs the same operations on a different set of polygons to build up the rendered image on the screen.

(b) [Bookwork] Write-invalidate is usually preferred to write-update for bus efficiency reasons. Multiple writes to the same cache block with no intervening reads require multiple update broadcasts in an update protocol, which is highly wasteful of the limited bus bandwidth. In contrast, the invalidate protocol requires only a single invalidate broadcast on the first write. Minimizing bus traffic is of paramount importance in this sort of architecture, since the bus is usually the bottleneck, limiting the number of processors which can be installed in the machine.

(c) Increasing the cache block size can lead to a higher miss rate if a write-invalidate coherency protocol is used. Suppose processor A writes word X. Also suppose that processor B's cache contains a copy of the block containing X: this block is invalidated when A writes. Now suppose processor B wishes to read word Y, which is distinct from X but in the same cache block. The read will miss, since the cache block has just been invalidated by A's write, even though word Y was not written by A. This cache miss would not occur with one-word blocks. This phenomenon is known as *false sharing.*

(d) If it takes t μs to process the incoming messages, the polling scheme can cope with inter-message intervals as low as $(1.6 + t)$ μs, whereas the interrupt scheme can only cope if the inter-message interval remains above $(19 + t)$ μs. The interrupt scheme cannot, therefore, be used to process successive messages in a single burst. The polling scheme, however, is wasteful of processor time between bursts: polling every 10 μs would consume 16% of the available processor time, even when there are no incoming messages. This suggests a hybrid approach. The processor should use the interrupt scheme for the first message, then poll every 10 μs until there are no further messages to process. It should then re-enable the interrupt mechanism. The network interface will need to buffer at least two messages while the interrupt service routine is being initiated.

Examiners comments: Another popular question. It tested the candidates understanding of multi-processor shared memory architectures. Most candidates produced a sensible solution to the problem posed in part (d). Clear explanations of the issues in parts (b) and (c) were less common.

---

3 (a) Because while certain resource-bound tasks are being executed (e.g. getting requests for CDs across the network, playing a CD etc.), other threads can be running, doing other work (e.g. handling another request, starting another CD player). With a single-threaded approach, only one thing can be done at once within the same process.

(b) A race condition can occur if a thread is preempted after it has checked the value of a in_use flag but before it has set the value of the in_use flag to reflect the fact that the corresponding CD is now taken. See the illustration below:

```
void request_play_cd(Track t, Location l) {
  int n=0;

  while (TRUE) {
    if (!in_use[n]) {

<========== THREAD PREEMPTED HERE

      in_use[n] = TRUE;
      cd[n].play(t,l);
      in_use[n] = FALSE;
      return;
    }
    else {
      n=n+1;
      if (n == NUM_CHANGERS) n=0;
    }
  }
}
```

Another thread can run and can see that that CD player has not been marked as busy and will start to use it as well.

The unexpected behaviour will manifest itself as one CD track starting to play momentarily and then being replaced by another track. The track that started playing will not start playing again. It will appear to the requester of the first track that the system has lost that track. All these stages are illustrated below. (NB this is only considering 2 thread problems — more threads could be involved in the race).

```
Thread1                         Thread2
========================================
check in_use
CD1 currently free
===========OS PREEMPTS==============>
                                check in_use
                                CD1 currently free
                                mark CD1 busy
                                play track on CD1
<===========THREAD BLOCKS============
mark CD1 busy
```

```
play track on CD1 (cut off currently playing track)
```

(c) You would use a semaphore mutex (initialized to 1) to only allow one thread to check and set the `in_use` flag at a time.

```
Semaphore mutex(1);
...
void request_play_cd(Track t, Location l) {
  int n=0;

  while (TRUE) {
    mutex.WAIT();
    if (!in_use[n]) {
      in_use[n] = TRUE;
      mutex.SIGNAL();
      cd[n].play(t,l);
      in_use[n] = FALSE;
      return;
    }
    else {
      mutex.SIGNAL();
      n=n+1;
      if (n == NUM_CHANGERS) n=0;
    }
  }
}
```

(d) `request_play_cd` is unnecessarily wasteful of processor time because it is *busy waiting* within the while loop. Waiting threads are going round and round waiting for a CD to become free. What we want to do is to send waiting threads to sleep and wake them up when a CD is free. This can be done using an additional semaphore, initialized to the number of resources (CD units). Only that many threads are allowed into the bit of code — the rest are sent to sleep. Whenever a thread leaves, a waiting thread is woken up by the OS.

```
Semaphore mutex(1);
Semaphore cd_semaphore(NUM_CHANGERS);
...
void request_play_cd(Track t, Location l) {
  int n=0;

  cd_semaphore.WAIT();
  while (TRUE) {
    mutex.WAIT();
    if (!in_use[n]) {
      in_use[n] = TRUE;
      mutex.SIGNAL();
      cd[n].play(t,1);
      in_use[n] = FALSE;
      cd_semaphore.SIGNAL();
      return;
    }
    else {
      mutex.SIGNAL();
      n=n+1;
      if (n == NUM_CHANGERS) n=0;
    }
  }
}
```

Some of the above code is now redundant because we are guaranteed to get allocated a CD player first time round the loop — because only 5 threads are allowed in.
Examiners comments: This question tested candidates understanding of semaphores. It was a straightforward question and several candidates achieved high marks. It was also attempted by some candidates who had little understanding of the topic and this dragged down the average mark.

---

4 (a) Candidates must demonstrate an ability to abstract functionality and an understanding of how to formulate a class interface.

```
class pollution_collector {

  public:
    void send_reading(Double pollution_level,
                      char *zone);
    Double current_reading(char *zone);
    Double average();
    Double max();
    Double min();

  private:
    Zone zones[NUM_ZONES];
}
```

(b) Information hiding is the ability to restrict information that users of a class can gain access to to a "need to know" basis. For example, in the class definition above some components are within the public part of the class. This means that any program using the class can access these components. Other parts are in the private part. These cannot be seen outside of the class itself.

The reason that information hiding improves robustness of system components is that other components can only interact with your components in the way you designed them to. In the example above, the implementation of how the zones are stored is an array. If this was not hidden then the fields of the array could be set by a calling program in a way the programmer never intended. Errors can occur due to unanticipated side-effects.

(c) To add exceptions requires 3 additions:

(i) Exceptions must be added to the class interface

```
class pollution_collector {
  public:
    ...
    class unknown_zone { }; // Exception
    ...
}
```

(ii) Code to signal exceptions must be added to the class implementation in the appropriate place

```
  if (!index = get_zone_index(zone)) {
    throw unknown_zone();
  }
```

(iii) The users of the object must handle exceptions (see answer to d).

```
  try {
    obj->send_reading(pl,z);
  }
  catch (pollution_collector::unknown_zone) {
    printf("Unknown zone\n");
  }
```

Exceptions improve the design because:

- Error handling is managed at the type system level — rather than being handled in an ad hoc system or user-dependent way.
- Exceptions are easier for a programmer to understand — because they can have meaningful names, like functions.
- Exceptions are named and can be parameterised — similar to function calls. They are subject to type checking, enabling errors to be trapped by the compiler.

- Exceptions can be handled in a uniform way, along with lots of other sorts of error or run-time condition.
- Returning error codes requires both parties to understand the calling conventions. Because exceptions are a standard mechanism, they get around this problem.
- If a calling function doesn't handle the exception, it is passed up to the next level until it is handled. With ad hoc error handling, this is not guaranteed.

(d) Candidate must show they understand the concept of creating object instances and invoking methods.

```
pollution_collector *pc;

try {
  pc = new pollution_collector();
  cout << pc->average() << endl;
}
catch (pollution_collector::unknown_zone) {
  printf("Unknown zone\n");
}

...
```

Examiners comments: This question tested candidates understanding of the software engineering principles associated with information hiding and exception handling. It was unpopular, but well answered by some candidates. As with question 3, there was a broad range in the quality of the answers.

---

5 (a) (i) The minimum error rate classifier simply assigns the test data sample to the class with the highest posterior probability. For each class i this is computed using Bayes' Theorem

$$P(\text{class}_i|\mathbf{x}) = \frac{P_i p(\mathbf{x}|\text{class}_i)}{\sum_{j=1}^{2} P_j p(\mathbf{x}|\text{class}_j)}$$

Since the denominator is independent of the class find (taking logs of the above and substituting for a Gaussian)

$$\arg\max_i \left\{\ln P_i - 1/2 \ln|\mathbf{\Sigma}_i| - 1/2(\mathbf{x}-\boldsymbol{\mu}_i)'\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right\}$$

(ii) The decision boundary is values of $\mathbf{x}$ when $P(\text{class}_1|\mathbf{x}) = P(\text{class}_2|\mathbf{x})$. So in this case defined by values of $\mathbf{x}$ such that

$$\ln P_1 - 1/2 \ln|\mathbf{\Sigma}_1| - 1/2(\mathbf{x}-\boldsymbol{\mu}_1)'\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) = \ln P_2 - 1/2 \ln|\mathbf{\Sigma}_2| - 1/2(\mathbf{x}-\boldsymbol{\mu}_2)'\mathbf{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)$$

or

$$(\mathbf{x}-\boldsymbol{\mu}_1)'\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)-(\mathbf{x}-\boldsymbol{\mu}_2)'\boldsymbol{\Sigma}_2^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)=-2\ln\frac{P_2}{P_1}\sqrt{\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_2|}}$$

which may be rewritten as

$$\mathbf{x}'(\boldsymbol{\Sigma}_1^{-1}-\boldsymbol{\Sigma}_2^{-1})\mathbf{x}-2(\boldsymbol{\mu}_1'\boldsymbol{\Sigma}_1^{-1}-\boldsymbol{\mu}_2'\boldsymbol{\Sigma}_2^{-1})\mathbf{x}+2\ln\frac{P_2}{P_1}\sqrt{\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_2|}}+\boldsymbol{\mu}_1'\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1-\boldsymbol{\mu}_2'\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2=0$$

(b) Gaussian mixture distributions can model arbitrary data distributions given enough mixture components. This includes (with diagonal covariance mixture components) multimodal distributions; correlated distributions and non-symmetric distributions (a diagram can be used to illustrate this to advantage).

The number of parameters needed will be $2dM + M - 1$ where d is the dimensionality of the data and M is the number of mixture components.

A full covariance model can model correlations directly (still assumes basic Gaussian structure and symmetric, unimodal). It has a symmetric covariance matrix with $d.(d+1)/2$ so the overall number including the mean is $d.(d+3)/2$. So this is not so powerful and for large d (10's or greater) then number of parameters involved can be very high and a Gaussian mixture approach (or feature reduction) is preferred.

The training of full covariance matrices is simple (non-iterative) compared to training Gaussian mixture models.

(c) (i) The log likelihood, $\mathcal{L}$ for N data elements $x_1 \cdots x_N$ for the 2 component Gaussian with mixture weights c_1, c_2 (assuming 1-dimensional data)

$$\mathcal{L}=\sum_{n=1}^{N}\ln p(x_n)=\sum_{n=1}^{N}\ln\left[\sum_{j=1}^{2}p(x_n|j)c_j\right]$$

and using the standard formula for a Gaussian distribution with mean and variance elements μ_j, σ_j^2

$$p(x|j)=\frac{1}{(2\pi)^{1/2}}\frac{1}{\sigma_j}e^{-\frac{1}{2}(\frac{x-\mu_j}{\sigma_j})^2}$$

(ii) Using Bayes' theorem in the form

$$P(j|x)=\frac{p(x|j)c_j}{p(x)}$$

where $P(x|j)$ is the posterior probability of mixture component j we get:

$$\frac{\partial\mathcal{L}}{\partial\mu_j}=\sum_{n=1}^{N}P(j|x_n)\frac{(x_n-\mu_j)}{\sigma_j^2}$$

and

$$\frac{\partial \mathcal{L}}{\partial \sigma_j^2} = \sum_{n=1}^{N} P(j|x_n)\frac{1}{2}\left[\frac{(\mu_j - x_n)^2}{\sigma_j^4} - \frac{1}{\sigma_j^2}\right]$$

For gradient descent we minimise $-\mathcal{L}$, so the update formulae at the $(t+1)^{\text{th}}$ iteration become

$$\mu_j[t+1] = \mu_j[t] - \Delta \left.\frac{\partial \mathcal{L}}{\partial \mu_j}\right|_{\mu_j[t],\sigma_j^2[t]}$$

$$\sigma_j^2[t+1] = \sigma_j^2[t+1] - \Delta \left.\frac{\partial \mathcal{L}}{\partial \sigma_j^2}\right|_{\mu_j[t],\sigma_j^2[t]}$$

where Δ is the step size.

Problems with this technique are:

1. The value of Δ is required to be small (so as to achieve stability), but large enough for rapid convergence.

2. The variance may go negative if directly implemented. Possible to overcome by optimising the log of the variance.

Neither of these is a problem for EM, though mathematicians worry about the convergence rate

Examiners comments: This question tested the candidates understanding of decision boundaries and Gaussian mixture models. Most candidates answered sections (a) and (b) reasonably well. However section (c) was very poorly answered, reflected in the low average mark.

---

6 (a) Principal components are an ordered set of orthogonal directions that account for most of the observed variability in the data.

Any vector can be expressed as a linear combination of the principal components (since they form a basis set). Truncation of the expression and representing the vector by the projections along the first principal components gives a data reduction that still represents most of the observed variability.

The principal components can be found by performing eigenvector analysis on the scatter matrices (either total scatter or average within-class scatter).

In this case we have

$$\Sigma = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

To find the eigenvalues solve

$$\begin{vmatrix} 4-\lambda & 2 \\ 2 & 4-\lambda \end{vmatrix} = 0$$

gives

$$12 - 8\lambda + \lambda^2 = 0 \text{ or } \lambda = 2, 6$$

For the eigenvector U_1 for the first principal direction ($\lambda_1 = 6$)

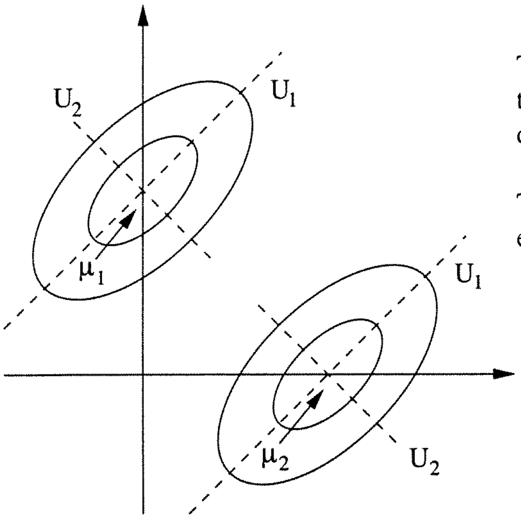
$$\begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} 6 \\ 6x \end{bmatrix}$$

$$U_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and for the second eigenvector U_2 ($\lambda_2 = 2$)

$$\begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} 2 \\ 2x \end{bmatrix}$$

$$U_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$



The eigenvectors λ_1 and λ_2 give the variance in the principal directions U_1 and U_2 .

The ellipses are contours of equal probability density.

(b) The Fisher discriminant is found by maximising the ratio of the projected means (in the direction of the Fisher discriminant) to the projected average within-class scatter matrix.

It is in the direction

$$b = [1/2(\Sigma_1 + \Sigma_2)]^{-1}(\mu_1 - \mu_2)$$

(i) Compute the Fisher discriminant

First find the inverse of the covariance matrix (determinant = 12) and is

$$\begin{bmatrix} 1/3 & -1/6 \\ -1/6 & 1/3 \end{bmatrix}$$

Hence the Fisher discriminant is in the direction

$$\begin{bmatrix} 1/3 & -1/6 \\ -1/6 & 1/3 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

(ii) Discrimination rule takes the Fisher direction b and projects data onto this. This value is then compared with a threshold T to form the decision choosing either class ω_1 or class ω_2

Hence

$$b'x \underset{\omega_2}{\overset{\omega_1}{\lessgtr}} T$$

The threshold can be chosen based on the class priors or the decision cost (and includes the projected average mean difference to average projected variance).

(iii) In this case, the first principal component is *orthogonal* to the direction of the Fisher discriminant and no discrimination between the classes is found by projecting onto the first principal component. Hence while this direction is useful for characterising the variation observed in the data sets it is not useful for discriminating between them.

[Due to the fact that in this case there is a common class covariance matrix with Gaussian class-conditional densities, the decision rule that optimises the Fisher criterion also gives the minimum probability of error]

Examiners comments: This question tested the candidates knowledge of how to choose discriminating directions. The majority of candidates understood the basic details of the two schemes. However the final section of section (b), where the two schemes were contrasted, was not well answered.

---

7 (a) The depth-first search algorithm:

1. Form a queue, Q, beginning with the start node (root).

2. Until Q is empty or goal has been reached:

   A determine if first (front of queue) element is the goal state,

   B if it isn't, remove the first element and replace it with its children, at the front of the queue.

3. If the goal state is reached, return success, else failure.

The algorithms differ in the details of step **B**:

depth-first: extend the deepest node first. If b is the branching factor and d is the depth, then the algorithm has a complexity of order b^d in time, and of is of order bd in memory. Advantage: efficient in the amount of memory required. Disadvantage: results in unbalanced trees.

breadth-first: extend the shallowest node first. The algorithm is of order b^d in both time and memory. Advantage: finds the solution in a minimum number of moves. Disadvantage: prohibitive in memory required if b and d are large. Step B becomes:

B if the first element of the queue is not the goal state then remove it
from the queue, and place its children at the BACK of the queue.

best-first: extend the most promising ('best') node first. This algorithm requires an heuristic evaluation function to be devised, and involves the additional effort of performing the sort in each step. The best-first search is not an optimal algorithm. The amount of pruning depends on the effectiveness of the evaluation heuristic. There is no concept of finding the best path to the goal (see A* search). Step B becomes:

B if the first element of the queue is not the goal state then remove it and replace it with its children. Sort the queue using an evaluation function to put the most promising node at the top.

(b) (i) A* search tree (see next page). Answers should also include an overview of A* search as provided in the notes.

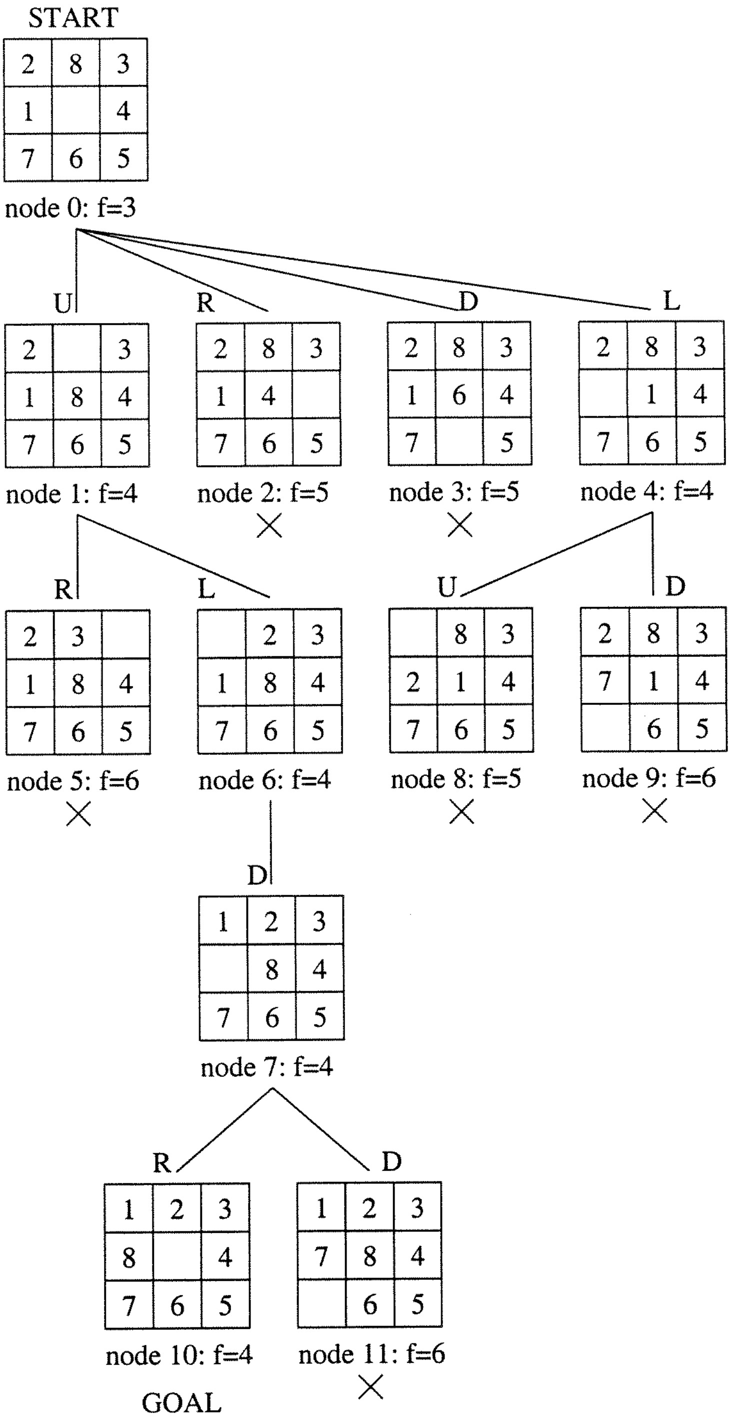
(ii) $f(n)$ is an admissible heuristic if $f(n) \leq g(\text{goal}) - g(n)$. That is to say, $f(n)$ always underestimates the true cost to reach the goal to avoid premature termination of a node.

The efficiency of heuristics can be compared by calculating the effective branching factor, b^*, where the total number of nodes expanded $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$. Aim to find $b^* \approx 1$.

A more efficient heuristic is the total, for all the misplaced tiles, of the Manhattan distance from their currant position to the goal position.

Examiners comments: This question tested the candidates knowledge of basic search techniques. Section (a) was standard book-work and generally well answered. Section (b) was also well answered, this was reflected in the high average mark obtained.

A* search tree.



8 (a) Sound rules of inference: the conclusion is true in all cases when the premises are true (i.e. truth preserving).

modus ponens:	if $\alpha \to \beta$ and α then β. $[(\alpha \to \beta) \wedge \alpha] \to \beta$
modus tolens:	if $\alpha \to \beta$ and $\neg\beta$ then $\neg\alpha$ $[(\alpha \to \beta) \wedge \neg\beta] \to \neg\alpha$
resolution:	if $\alpha \vee \beta$ and $\neg\beta \vee \gamma$ then $\alpha \vee \gamma$ $\equiv [(\neg\alpha \to \beta) \wedge (\beta \to \gamma)] \to (\neg\alpha \to \gamma)$

(b) Abduction: if $\alpha \to \beta$ and β then we can infer α. This is not a sound rule. By truth table:

α	β	$\alpha \to \beta$	premises	conclusion	statement
T	F	F	F	T	√
T	T	T	T	T	√
F	F	T	F	F	√
F	T	T	T	F	×

Alternatively:

$$\begin{aligned} S &\equiv ((\alpha \to \beta) \wedge \beta) \to \alpha \\ &= \neg[(\neg\alpha \vee \beta) \wedge \beta] \vee \alpha \end{aligned}$$

which is not always true, thus the sentence S is not valid.

(c) (i) First-order logic.

$$\begin{aligned} &\forall x[\text{pass}(x) \to \text{happy}(x)] \\ &\forall x[\text{study}(x) \vee \text{lucky}(x) \to \text{pass}(x)] \\ &\neg\text{study}(\text{John}) \wedge \text{lucky}(\text{John}) \end{aligned}$$

(ii) Conjunctive normal form.

$$\neg\text{pass}(x1) \vee \text{happy}(x1) \tag{1}$$

$$\neg\text{study}(x2) \vee \text{pass}(x2) \tag{2}$$

$$\neg\text{lucky}(x3) \vee \text{pass}(x3) \tag{3}$$

$$\neg\text{study}(\text{John}) \tag{4}$$

$$\text{lucky}(\text{John}) \tag{5}$$

(iii) To prove a theorem using resolution.

- Negate the theorem to be proved and add the result to the list of axioms.
- Put the axioms into clause form.
- Until the empty clause, Nil, is produced or there is no resolvable pair of clauses, find resolvable clauses, resolve them and add the result to the list of clauses.
- If the empty clause is produced, report that the theorem is true. If there are no resolvable clauses, report that the theorem is false.

(iv) Add negation of goal and look for a contradiction.

$$\neg\text{happy}(\text{John}) \tag{6}$$

Resolve (6) with (1), unify $x1/\text{John}$.

$$\neg\text{pass}(\text{John}) \tag{7}$$

Resolve (7) with (3), unify $x3/\text{John}$.

$$\neg\text{lucky}(\text{John}) \tag{8}$$

Conflict with (5). Hence the clause added to the database was false (i.e. $\text{KB}\wedge\neg\text{happy}(\text{John}) \rightarrow \text{FALSE}$), and we have proved that

$$\text{happy}(\text{John}).$$

Examiners comments: This question tested the candidates knowledge of basic logic. Overall the question was very well done. Most sections were well answered, though some candidates had problems with section (b).