Uncertain Prospects

Suppose you have to eat at a restaurant and your choices are:

- chicken
- quiche

Normally you prefer chicken to quiche, but ...

Now you're uncertain as to whether the chicken has salmonella. You think it's unlikely, but it's possible.

- **Key point**: you no longer know the outcome of your choice.
- This is the common situation!

How do you model this, so you can make a sensible choice?

States, Acts, and Outcomes

The standard formulation of decision problems involves:

- a set S of *states* of the world,
  - *state*: the way that the world could be (the chicken is infected or isn't)
- a set O of *outcomes*
  - *outcome*: what happens (you eat chicken and get sick)
- a set A of *acts*
  - *act*: function from states to outcomes

A decision problem with certainty can be viewed as the special case where there is only one state.

- There is no uncertainty as to the true state.

One way of modeling the example:

- two states:
  - s_1: chicken is not infected
  - s_2: chicken is infected
- three outcomes:
  - o_1: you eat quiche
  - o_2: you eat chicken and don't get sick
  - o_3: you eat chicken and get sick
- Two acts:
  - a_1: eat quiche
    - $a_1(s_1) = a_1(s_2) = o_1$
  - a_2: eat chicken
    - $a_2(s_1) = o_2$
    - $a_2(s_2) = o_3$

This is often easiest to represent using a matrix, where the columns correspond to states, the rows correspond to acts, and the entries correspond to outcomes:

	s_1	s_2
a_1	eat quiche	eat quiche
a_2	eat chicken; don't get sick	eat chicken; get sick

Specifying a Problem

Sometimes it's pretty obvious what the states, acts, and outcomes should be; sometimes it's not.

Problem 1: the state might not be detailed enough to make the act a function.

- Even if the chicken is infected, you might not get sick.

Solution 1: Acts can return a probability distribution over outcomes:

- If you eat the chicken in state s_1, with probability 60% you might get infected

Solution 2: Put more detail into the state.

- state s_{11}: the chicken is infected and you have a weak stomach
- state s_{12}: the chicken is infected and you have a strong stomach

Problem 2: Treating the act as a function may force you to identify two acts that should be different.

Example: Consider two possible acts:

- carrying a red umbrella
- carrying a blue umbrella

If the state just mentions what the weather will be (sunny, rainy, ...) and the outcome just involves whether you stay dry, these acts are the same.

- An act is just a function from states to outcomes

Solution: If you think these acts are different, take a richer state space and outcome space.

Problem 3: The choice of labels might matter.

Example: Suppose you're a doctor and need to decide between two treatments for 1000 people. Consider the following outcomes:

- Treatment 1 results in 400 people being dead
- Treatment 2 results in 600 people being saved

Are they the same?

- Most people don't think so!

Problem 4: The states must be independent of the acts.

Example: Should you bet on the American League or the National League in the All-Star game?

	AL wins	NL wins
Bet AL	+$5	-$2
Bet NL	-$2	+$3

But suppose you use a different choice of states:

	I win my bet	I lose my bet
Bet AL	+$5	-$2
Bet NL	+$3	-$2

It looks like betting AL is at least as good as betting NL, no matter what happens. So should you bet AL?

What is wrong with this representation?

Example: Should the US build up its arms, or disarm?

	War	No war
Arm	Dead	Status quo
Disarm	Red	Improved society

Problem 5: The actual outcome might not be among the outcomes you list! Similarly for states.

- In 2002, the All-Star game was called before it ended, so it was a tie.
- What are the states/outcomes if trying to decide whether to attack Iraq?

Decision Rules

We want to be able to tell a computer what to do in all circumstances.

- Assume the computer knows S, O, A
  - This is reasonable in limited domains, perhaps not in general.
  - Remember that the choice of S, O, and A may affect the possible decisions!
- Moreover, assume that there is a utility function u mapping outcomes to real numbers.
  - You have a total preference order on outcomes!
- There may or may not have a measure of likelihood (probability or something else) on S.

You want a *decision rule*: something that tells the computer what to do in all circumstances, as a function of these inputs.

There are *lots* of decision rules out there.

Maximin

This is a conservative rule:

- Pick the act with the best worst case.
  - Maximize the minimum

Formally, given act $a \in A$, define

$$worst_u(a) = \min\{u_a(s) : s \in S\}.$$

- $worst_u(a)$ is the worst-case outcome for act a

Maximin rule says $a \succeq a'$ iff $worst_u(a) \geq worst_u(a')$.

	s_1	s_2	s_3	s_4
a_1	5	0^*	0^*	2
a_2	-1^*	4	3	7
a_3	6	4	4	1^*
a_4	5	6	4	3^*

Thus, get $a_4 \succ a_3 \succ a_1 \succ a_2$.

But what if you thought s_4 was much likelier than the other states?

Maximax

This is a rule for optimists:

- Choose the rule with the best case outcome:
  - Maximize the maximum

Formally, given act $a \in A$, define

$$best_u(a) = \max\{u_a(s) : s \in S\}.$$

- $best_u(a)$ is the best-case outcome for act a

Maximax rule says $a \succeq a'$ iff $best_u(a) \geq best_u(a')$.

	s_1	s_2	s_3	s_4
a_1	5^*	0	0	2
a_2	-1	4	3	7^*
a_3	6^*	4	4	1
a_4	5	6^*	4	3

Thus, get $a_2 \succ a_4 \sim a_3 \succ a_1$.

Optimism-Pessimism Rule

Idea: weight the best case and the worst case according to how optimistic you are.

Define $opt_u^\alpha(a) = \alpha best_u(a) + (1-\alpha)worst_u(a)$.

- if $\alpha = 1$, get maximax
- if $\alpha = 0$, get maximin
- in general, α measures how optimistic you are.

Rule: $a \succeq a'$ if $opt_u^\alpha(a) \geq opt_u^\alpha(a')$

This rule is strange if you think probabilistically:

- $worst_u(a)$ puts weight (probability) 1 on the state where a has the worst outcome.
  - This may be a different state for different acts!
- More generally, opt_u^α puts weight α on the state where a has the best outcome, and weight $1-\alpha$ on the state where it has the worst outcome.

Minimax Regret

Idea: minimize how much regret you would feel once you discovered the true state of the world.

- The "I wish I would have done x" feeling

For each state s, let a_s be the act with the best outcome in s.

$$\mathit{regret}_u(a, s) = u_{a_s}(s) - u_a(s)$$
$$\mathit{regret}_u(a) = \max_{s \in S} \mathit{regret}_u(a, s)$$

- $\mathit{regret}_u(a)$ is the maximum regret you could ever feel if you performed act a

Minimax regret rule:

$$a \succeq a' \text{ iff } \mathit{regret}_u(a) \leq \mathit{regret}_u(a')$$

- minimize the maximum regret

Example:

	s_1	s_2	s_3	s_4
a_1	5	0	0	2
a_2	-1	4	3	7^*
a_3	6^*	4	4^*	1
a_4	5	6^*	4^*	3

- $a_{s_1} = a_3$; $u_{a_{s_1}}(s_1) = 6$
- $a_{s_2} = a_4$; $u_{a_{s_2}}(s_2) = 6$
- $a_{s_3} = a_3$ (and a_4); $u_{a_{s_3}}(s_3) = 4$
- $a_{s_4} = a_2$; $u_{a_{s_4}}(s_4) = 7$

- $\mathit{regret}_u(a_1) = \max(6-5, 6-0, 4-0, 7-2) = 6$
- $\mathit{regret}_u(a_2) = \max(6-(-1), 6-4, 4-3, 7-7) = 7$
- $\mathit{regret}_u(a_3) = \max(6-6, 6-4, 4-4, 7-1) = 6$
- $\mathit{regret}_u(a_4) = \max(6-5, 6-6, 4-4, 7-3) = 4$

Get $a_4 \succ a_1 \sim a_3 \succ a_2$.

Effect of Transformations

Proposition Let f be an ordinal transformation of utilities (i.e., f is an increasing function):

- maximin(u) = maximin$(f(u))$
  - The preference order determined by maximin given u is the same as that determined by maximin given $f(u)$.
  - An ordinal transformation doesn't change what is the worst outcome
- maximax(u) = maximax$(f(u))$
- $opt^\alpha(u)$ may not be the same as $opt^\alpha((u))$
- $regret(u)$ may not be the same as $regret(f(u))$.

Proposition: Let f be a positive affine transformation

- $f(x) = ax + b$, where $a > 0$.

Then

- maximin(u) = maximin$(f(u))$
- maximax(u) = maximax$(f(u))$
- $opt^\alpha(u) = opt^\alpha(f(u))$
- $regret(u) = regret(f(u))$

“Irrelevant” Acts

Suppose that $A = \{a_1, \ldots, a_n\}$ and, according to some decision rule, $a_1 \succ a_2$.

Can adding another possible act change things?

That is, suppose $A' = A \cup \{a\}$.

- Can it now be the case that $a_2 \succ a_1$?

No, in the case of maximin, maximax, and opt^α. But …

Possibly yes in the case of minimax regret!

- The new act may change what is the best act in a given state, so may change all the calculations.

Example: start with

	s_1	s_2
a_1	8	1
a_2	2	5

$$regret_u(a_1) = 4 < regret_u(a_2) = 6$$

$$a_1 \succ a_2$$

But now suppose we add a_3:

	s_1	s_2
a_1	8	1
a_2	2	5
a_3	0	8

Now

$$regret_u(a_2) = 6 < regret_u(a_1) = 7 < regret_u(a_3) = 8$$

$$a_2 \succ a_1 \succ a_3$$

Is this reasonable?

Multiplicative Regret

The notion of regret is additive; we want an act that such that the difference between what you get and what you could have gotten is not too large.
There is a multiplicative version:

- find an act such that the ratio of what you get and what you could have gotten is not too large.
- usual formulation:

  your cost/what your cost could have been

  is low.

This notion of regret has been extensively studied in the CS literature, under the name *online algorithms* or *competitive ratio*.

Given a problem P with optimal algorithm OPT.

- The optimal algorithm is given the true state

Algorithm A for P has *competitive ratio* c if there exists a constant k such that, for all inputs x

$$\text{running time}(A(x)) \leq c(\text{running time}(OPT(x))) + k$$

The Object Location Problem

Typical goal in CS literature:

- find optimal competitive ratio for problems of interest

This approach has been applied to lots of problems,

- caching, scheduling, portfolio selection, ...

Example: Suppose you have a robot located at point 0 on a line, trying to find an object located somewhere on the line.

- What's a good algorithm for the robot to use?

The optimal algorithm is trivial:

- Go straight to the object

Here's one algorithm:

- Go to $+1$, then -2, then $+4$, then -8, until you find the object

Homework: this algorithm has a competitive ratio of 9

- I believe this is optimal

The Ski Rental Problem

Example:

- It costs \$$p$ to purchase skis
- it costs \$$r$ to rent skis
- You will ski for at most N days (but maybe less)

How long should you rent before you buy?

- It depends (in part) on the ratio of p to r
  - If the purchase price is high relative to rental, you should rent longer, to see if you like skiing

We'll come back to this problem in a future homework.

The Principle of Insufficient Reason

Consider the following example:

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
a_1	9	9	9	9	9	9	9	9	0
a_2	9	0	0	0	0	0	0	0	9

None of the previous decision rules can distinguish a_1 and a_2. But a lot of people would find a_1 better.

- it's more "likely" to produce a better result

Formalization:

- $u_a(s) = u(a(s))$: the utility of act a in state s
    - u_a is a random variable
- Let $\overline{\Pr}$ be the uniform distribution on S
    - All states are equiprobable
    - No reason to assume that one is more likely than others.
- Let $E_{\overline{\Pr}}(u_a)$ be the expected value of u_a

Rule: $a \succ a'$ if $E_{\overline{\Pr}}(u_a) > E_{\overline{\Pr}}(u'_a)$.

Problem: this approach is sensitive to the choice of states.

- What happens if we split s_9 into 20 states?

Related problem: why is it reasonable to assume that all states are equally likely?

- Sometimes it's reasonable (we do it all the time when analyzing card games); often it's not

Maximizing Expected Utility

If there is a probability distribution Pr on states, can compute the expected probability of each act a:

$$E_{\Pr}(u_a) = \Sigma_{s \in S} \Pr(s) u_a(s).$$

Maximizing expected utility (MEU) rule:

$$a \succ a' \text{ iff } E_{\Pr}(u_a) > E_{\Pr}(u_{a'}).$$

Obvious question:

- Where is the probability coming from?

In computer systems:

- Computer can gather statistics
  - Unlikely to be complete

When dealing with people:

- Subjective probabilities
  - These can be hard to elicit
  - What do they even mean?

Eliciting Utilities

MEU is unaffected by positive affine transformation, but may be affected by ordinal transformations:

- if f is a positive affine transformation, then MEU(u) = MEU($f(u)$)
- if f is an ordinal transformation, then MEU(u) $\neq$ MEU($f(u)$).

So where are the utilities coming from?

- People are prepared to say "good", "better", "terrible"
- This can be converted to an ordinal utility
- Can people necessarily give differences?

We'll talk more about utility elicitation later in the course

- This is a significant problem in practice, and the subject of lots of research.

Minimizing Expected Regret

Recall that a_s is the act with te best outcome in state s.

$$regret_u(a, s) = u_{a_s}(s) - u_a(s)$$
$$regret_u(a) = \max_{s \in S} regret_u(a, s)$$

Given Pr, the *expected regret* of a is

$$E_{\Pr}(regret_u(a, \cdot)) = \Sigma_{s \in S} \Pr(s) regret_u(a, s)$$

Minimizing expected regret (MER) rule:

$$a \succ a' \text{ iff } E_{\Pr}(regret_u(a, \cdot)) < E_{\Pr}(regret_u(a', \cdot))$$

Theorem: MEU and MER are equivalent rules!

$$a \succ_{MEU} a' \text{ iff } a \succ_{MER} a'$$

Proof:

1. Let $u' = -u$
   - Maximizing $E_{\Pr}(u_a)$ is the same as minimizing $E_{\Pr}(u'_a)$.
2. Let $u^v(a, s) = u'(a, s) + v(s)$, where $v : S \rightarrow I\!R$ is arbitrary.
   - Minimizing $E_{\Pr}(u'_a)$ is the same as minimizing $E_{\Pr}(u^v_a)$.
   - You've just added the same constant ($E_{\Pr}(v)$) to the expected value of u'_a, for each a
3. Taking $v(s) = u(a_s)$, then $E_{\Pr}(u^v_a)$ is the expected regret of a!

Representing Uncertainty by a Set of Probabilities

Why is probability even the right way to represent uncertainty??

Consider tossing a fair coin. A reasonable way to represent your uncertainty is with the probability measure $\Pr_{1/2}$:

$$\Pr_{1/2}(\mathit{heads}) = \Pr_{1/2}(\mathit{tails}) = 1/2.$$

Now suppose the bias of the coin is unknown. How do you represent your uncertainty about heads?

- Could still use $\Pr_{1/2}$
- Perhaps better: use the set

$$\{\Pr_a : a \in [0,1]\}, \text{ where } \Pr_a(\mathit{heads}) = a.$$

Decision Rules with Sets of Probabilities

Given set $\mathcal{P}$ of probabilities, define

$$\underline{E}_{\mathcal{P}}(u_a) = \inf_{\Pr \in \mathcal{P}} \{E_{\Pr}(u_a) : \Pr \in \mathcal{P}\}$$

This is like maximin:

- Optimizing the worst-case expectation

In fact, if $\mathcal{P}_S$ consists of *all* probability measures on S, then $\underline{E}_{\mathcal{P}_S}(u_a) = \mathit{worst}_u(a)$.

Decision rule 1: $a >^1_{\mathcal{P}} a'$ iff $\underline{E}_{\mathcal{P}}(u_a) > \underline{E}_{\mathcal{P}}(u_{a'})$

- maximin order agrees with $>^1_{\mathcal{P}_S}$.
- $>^1_{\mathcal{P}}$ can take advantage of extra information

Define $\underline{E}_{\mathcal{P}}(u_a) = \sup_{\Pr \in \mathcal{P}} \{E_{\Pr}(u_a) : \Pr \in \mathcal{P}\}$.

- Rule 2: $a >^1_{\mathcal{P}} a'$ iff $\overline{E}_{\mathcal{P}}(u_a) > \overline{E}_{\mathcal{P}}(u_{a'})$
  - This is like maximax
- Rule 3: $a >^3_{\mathcal{P}} a'$ iff $\underline{E}_{\mathcal{P}}(u_a) > \overline{E}_{\mathcal{P}}(u_{a'})$
  - This is an extremely conservative rule
- Rule 4: $a >^4_{\mathcal{P}} a'$ iff $E_{\Pr}(u_a) > E_{\Pr}(u_{a'})$ for all $\Pr \in \mathcal{P}$

For homework: $a \geq^3_{\mathcal{P}} a'$ implies $a \geq^4_{\mathcal{P}} a'$

What's the "right" rule?

One way to determine the right rule is to characterize the rules axiomatically:

- What properties of a preference order on acts guarantees that it can be represented by MEU? maximin? …
- We'll do this soon for MEU

Can also look at examples.

Rawls vs. Harsanyi

Which of two societies (each with 1000 people) is better:

- Society 1: 900 people get utility 90, 100 get 1
- Society 2: everybody gets utility 35.

To make this a decision problem:

- two acts:
  1. live in Society 1
  2. live in Society 2
- 1000 states: in state i, you get to be person i

Rawls says: use maximin to decide
Harsanyi says: use principle of insufficient reason

- If you like maximin, consider Society $1'$, where 999 people get utility 100, 1 gets utility 34.
- If you like the principle of insufficient reason, consider society $1''$, where 1 person gets utility 100,000, 999 get utility 1.

Example: The Paging Problem

Consider a two-level *virtual memory system*:

- Each level can store a number of fixed-size memory units called *pages*
- *Slow memory* can store N pages
- *Fast memory* (aka *cache*) can store $k < N$ of these
- Given a request for a page p, the system must make p available in fast memory.
- If p is already in fast memory (a *hit*) then there's nothing to do
- otherwise (on a *miss*) the system incurs a *page fault* and must copy p from slow memory to fast memory
  - But then a page must be deleted from fast memory
  - Which one?

Cost models:

1. charge 0 for a hit, charge 1 for a miss
2. charge 1 for a hit, charge $s > 1$ for a miss

The results I state are for the first cost model.

Algorithms Used in Practice

Paging has been studied since the 1960s. Many algorithms used:

- LRU (Least Recently Used): replace page whose most recent request was earliest
- FIFO (First In/ First out): replace page which has been in fast memory longest
- LIFO (Last In/ First out): replace page most recently moved to fast memory
- LFU (Least Frequently Used): Replace page requested the least since entering fast memory
- ...

These are all *online* algorithms; they don't depend on knowing the full sequence of future requests. What you'd love to implement is:

- LFD (longest-forward distance): replace page whose next request is latest

But this requires knowing the request sequence.

Paging as a Decision Problem

This is a dynamic problem. What are the states/outcomes/acts?

- States: sequence of requests
- Acts: strategy for initially placing pages in fast memory + replacement strategy
- Outcomes: a sequence of hits + misses

Typically, no distribution over request sequences is assumed.

- If a distribution were assumed, you could try to compute the strategy that minimized expected cost
  - utility = $-$cost
- But this might be difficult to do in practice
- Characterizing the distribution of request sequences is also difficult
  - A set of distributions may be more reasonable
    - There has been some work on this
  - Each distribution characterizes a class of "requestors"

Paging: Competitive Ratio

Maximin is clearly not a useful decision rule for paging

- Whatever the strategy, can always find a request sequence that results in all misses

There's been a lot of work on the competitive ratio of various algorithms:

Theorem: [Belady] LFD is an optimal offline algorithm.

- replacing page whose next request comes latest seems like the obvious thing to do, but proving optimality is not completely trivial.
- The theorem says that we should thus compare the performance of an online algorithm to that of LFD.

Theorem: If fast memory has size k, LRU and FIFO are k-competitive:

- For all request sequences, they have at most k times as many misses as LFD
- There is a matching lower bound.

LIFO and LFU are not competitive

- For all ℓ, there exists a request sequence for which LIFO (LRU) has at least ℓ times as many misses as LFD

- For LIFO, consider request sequence

$$p_1, \ldots, p_k, p_{k+1}, p_k, p_{k+1}, p_k, p_{k+1}, \ldots$$

  - Whatever the initial fast memory, LFD has at most $k+1$ misses
  - LIFO has a miss at every step after the first k

- For LFU, consider request sequence

$$p_1^\ell, \ldots, p_{k-1}^\ell, (p_k, p_{k+1})^{\ell-1}$$

  - Whatever the initial fast memory, LFD has at most $k+1$ misses
  - LFU has a miss at every step after the first $(k-1)\ell$ $\Rightarrow 2(\ell-1)$ misses
    - Thus, $(k-1)+2(\ell-1)$ misses altogether.
    - This makes the competitive ratio

$$[(k-1)+2(\ell-1)]/(k-1)$$

    - Since ℓ can be arbitrarily large, the competitive ratio can be made arbitrarily large.

- Note both examples require that there be only $k+1$ pages altogether.

Paging: Theory vs. Practice

- the "empirical" competitive ratio of LRU is < 2, independent of fast memory size
- the "empirical" competitive ratio of FIFO is ~ 3, independent of fast memory size

Why do they do well in practice?

- One intution: in practice, request sequences obey some *locality of reference*
    - Consecutive requests are related

Modeling Locality of Reference

One way to model locality of reference: use an *access graph* G

- the nodes in G are requests
- require that successive requests in a sequence have an edge between them in G
- if G is completely connected, arbitrary sequences of requests are possible
- FIFO does not adequately exploit locality of reference
  - For any access graph G, the competitive ratio of FIFO is $> k/2$
- LRU can exploit locality of reference
  - E.g.: if G is a line, the competitive ration of LRU is 1
    - LRU does as well as the optimal algorithm in this case!
  - E.g.: if G is a grid, the competitive ration of LRU is $\sim 3/2$

Key point: you can model knowledge of the access pattern without necessarily using probability.

Example: Query Optimization

A decision theory problem from databases: query optimization.

- Joint work with Francis Chu and Praveen Seshadri.

Given a database query, the DBMS must choose an appropriate evaluation plan.

- Different plans produce the same result, but may have wildly different costs.

Queries are optimized once and evaluated frequently.

- A great deal of effort goes into optimization!

Why is Query Optimization Hard?

Query optimization is simple in principle:

- Evaluate the cost of each plan
- Choose the plan with minimum cost

Difficult in practice:

1. There are too many plans for an optimizer to evaluate
2. Accurate cost estimation depends on accurate estimation of various parameters, about which there is uncertainty:
    - amount of memory available
    - number of tuples in a relation with certain properties
    - ...

- Solution to problem 1: use dynamic programming (System R approach)
- Solution to problem 2: assume expected value of each relevant parameter is the actual value to get LSC (Least Specific Cost) plan.

A Motivating Example

Claim: Assuming the expected value is the actual value can be a bad idea . . .

Consider a query that requires a join between tables A and B, where the result needs to be ordered by the join column.

- A has 1,000,000 pages
- B has 400,000 pages
- the result has 3000 pages.
- *Plan 1:* Apply a sort-merge join to A and B.
  - If available buffer size > 1000 pages ($\surd$ of larger relation), join requires two passes over the relations; otherwise it requires at least three.
  - Each pass requires that 1,400,000 pages be read and written.
- *Plan 2:* Apply a Grace hash-join to A and B and then sort their result.
  - if available buffer size is > 633 pages ($\surd$ of smaller relation), the hash join requires two passes over the input relations.

- ◦ Also some additional overhead in sorting.

If the available buffer memory is accurately known, it is trivial to choose between the two plans

- Plan 1 if $>$ 1000 pages available, else Plan 2

Assume that available memory is estimated to be 2000 pages 80% of the time and 700 pages 20% of the time

- Plan A is best under the assumption that the expected value of memory (1740) is the actual value
- But Plan B has the least expected cost!

If utility $= -$running time, then LEC plan is the plan that maximizes expected utility.

- Is this the right plan to choose?
- If so, how hard is it to compute?

Computing Joins: The Standard Approach

Suppose we want to compute $A_1 \bowtie \ldots \bowtie A_n$:

- Joins are commutative and associative
- How should do we order the joins?
- System R simplification: to join k sets, first join $k-1$ and then add the last one.
  - Don't join $A_1 \ldots A_4$, $A_5 \ldots A_9$, and then join the results
  - Order the relations, and then join from left.

A *left-deep plan* has the form

$$(\ldots((A_{\pi(1)} \bowtie A_{\pi(2)}) \bowtie A_{\pi(3)}) \ldots \bowtie A_{\pi(n)})$$

for some permutation π.

- How do we find the best permutation?

The System R Approach

Idea:

- Assume a fixed setting for parameters
- Construct a dag with nodes labeled by subsets of $\{1, \ldots, n\}$.
- Compute the optimal plan (for that setting) for computing the join over $S \subseteq \{1, \ldots, n\}$ by working down the dag

Theorem: The System R optimizer computes the LSC left-deep plan for the specific setting of the parameters.

Computing the LEC Plan

We can modify the standard System R optimizer to compute the LEC plan with relatively little overhead.

Key observation: can instead compute the LEC plan for the join over S if we have a distribution over the relevant parameters.

- Divide the parameter space into "buckets"
  - Doing this well is an interesting research issue
- Assume a probability distribution on the buckets.
- Can apply the System R approach to compute the LEC plan at every node in the tree.

Theorem: This approach gives us the LEC left-deep plan.

- This approach works even if the parameters change dynamically (under some simplifying assumptions)

Is the LEC Plan the Right Plan?

The LEC plan is the right plan if the query is being run repeatedly, care only about minimizing total running time.

- The running time of N queries $\to N\times$ expected cost of single query.

But what if the query is only being used once?

- Your manager might be happier with a plan that minimizes regret.

Other problems:

- What if you have only incomplete information about probabilities?
- What if utility $\neq -$running time?
  - Consider time-critical data.
- Our algorithms work only in the case that utility $=$ $-$running time

Some Morals and Observations

1. Complexity matters
   - Even if you want to be "rational" and maximize expected utility, finding the act that maximizes expected utility may be hard.
2. It may be useful to approximate the solution:

   If you want to compute
   $$\Sigma_{i=1}^{n} \Pr(X = i) f(i)$$
   and f is "continuous" ($f(i)$ is close to $f(i+1)$ for all i), then you can approximate it by
   - partitioning the interval $[1, \ldots, n]$ into contingu-ous sets $A_1, \ldots, A_m$,
   - taking $g(j)$ to be some intermediate value of f in A_j
   - computing $\Sigma_{j=1}^{m} \Pr(X \in A_j) g(j)$

   This is what happens in computing expected running time if there are i units of memory.
   - Computing a reasonable approximation may be much easier than computing the actual value

3. Sometimes variance is relevant

   - Managers don't like to be surprised
   - If the same query takes vastly different amounts of time, they won't be happy
   - Apparently, ATMs are slowed down at 3 AM for that reason

   Problem: what utility function captures variance??

   - Variance is a property of a whole distribution, not a single state
   - Need a more complex state space

Complexity Theory and Decision Theory

Let $T(A(x))$ denote the running time of algorithm A on input x.
Intuitively, larger input $\rightarrow$ longer running time.

- Sorting 1000 items takes longer than sorting 100 items

Typical CS goal: characterize complexity of a problem in terms of the running time of algorithms that solve it.

CS tends to focus on the worst-case running time and order of magnitude.

- E.g., running time of A is $O(n^2)$ if there exist constants c and k such that $T(A(x)) \leq c|x|^2 + k$ for all inputs x.
- It could be the case that $T(A(x)) \leq 2|x|$ for "almost all" x

The complexity of a problem is the complexity of the best algorithm for that problem.

- How hard is sorting?
- The naive sorting algorithm is $O(n^2)$
- Are there algorithms that do better?
- Yes, there is an $O(n \log n)$ algorithm, and this is best possible.
  - Every algorithm that does sorting must take at least $O(n \log n)$ steps on some inputs.

Key point: choosing an algorithm with best worst-case complexity means making the maximin choice.

- Choices are algorithms
- States are inputs
- Outcome is running time

Why is the maximin choice the “right” choice?

- In practice, algorithms with good worst-case running time typically do well.

But this is not always true.

- The simplex algorithm for linear programming has worst-case exponential-time complexity, and often works better in practice than polynomial-time algorithms.
- There has been a great deal of work trying to explain why.
- The focus has been on considering average-case complexity, for some appropriate probability distribution.

Choosing the algorithm with the best average-case complexity amounts to maximizing expected utility.

Problem with average-case complexity:

- It’s rarely clear what probability distribution to use.
- A probability distribution that’s appropriate for one application may be inappropriate for another.

It may make sense to consider maximin expected complexity with respect to a set of distributions:

- If we consider all distributions, this gives worst-case complexity
- If we consider one distribution, this gives average-case complexity.

If we can find a well-motivated set of distributions for a particular application, this can be a reasonable interpolation between worst-case and average-case complexity.

As we have seen, considering the competitive ratio is another alternative, that seems reasonable in some applications.

More Thoughts on the "Best" Rule

- Suppose you can randomize over acts:
  - $pa + (1-p)a'$ = do a with probability p and a' with probability $1-p$.
  - You toss a biased coin to decide what to do
- You might expect: if $a \sim a'$, then $a \sim pa + (1-p)a'$
- Not true for minimax, maximax, optimism-pessimism if $u(pa + (1-p)a', s) = pu(a, s) + (1-p)u(a', s)$.

Example:

	s_1	s_2
a_1	2	0
a_2	0	2

According to optimism-pessimism rule $a_1 \sim a_2$ (for all indices α).

- But $a_1 \not\sim \frac{1}{2}a_1 + \frac{1}{2}a_2$ (unless $\alpha = 1/2$).

- $u(pa + (1-p)a', s) = pu(a, s) + (1-p)u(a', s)$ is incompatible with ordinal transformations
  - If $f(x) = x^3$, then $f(1) \neq (f(0) + f(2))/2$.

Recall that minimizing expected regret is affected by the addition of "irrelevant" acts:

- You can add an act a_3 and change the relative order of a_1 and a_2

This suggests that "probability-based" approaches might be better

- Note that you don't necessarily have to maximize expected utility in order to use probability in a sensible way.
- Could use sets of probabilities, as we've seen.

But probability-based approaches aren't a panacea either.

- There is also the problem of where the proability is coming from

Reminiscent of Arrow's Theorem:

- There is no "ideal" way to aggregate choices/make decisions

That doesn't necessarily mean you should give up!