4/4/01 IDSIA

The Singularity is Near — ~ Name of New Book By Ray Kurtzweil

:00 :63.35 : On [redacted] Time Share for OZ probs (Lsrch):
Suppose for all OT's that run for time T on this problem, a continuous best, it's proby = P_i. Say we did Lsrch on all OT's w. Time = $P_i T$ spent on each (w. upper limit T) When we spent T on cand$_i$, it would be best, of all cands w. T spent on [redacted] them

.04 [redacted] Try this: for time limit T of OZ problem, spend time $T p_i$ on cand$_i$.
Pick best output. This is it. ~~The result is better than of one had a ...~~
. If one used this w. a clock speed $\frac{1}{p_i}$ times present clock, it would be (as fast as or as good as) (for time T)
.07 the optimum method using normal clock speed for time T.)

.08 → I think .04–.07 is a correct way to do/interpret the Levin TS for OZ. (12.19)

:10 → Hvr, re: (.04–.07) Correct, but Not convincing! Say one has 10 OT's each w. pc = .1, would it not be better to try cand 1 only, for the whole T time?
.12 One advantage of .04–.07: it does "exercise" the other OT's, so that in S.T. phase, one can evaluate them better — But this is not an arg. Toward (short term) "Optimality"!
(Looks very serious!)

(.10) could be related to the "flat pc" problem, in which one has (a set of) many similar cands, so it's better to work on only one (representative) in the set.

129 64
580
132 75
89
10:32 8
4/5/01

Another form of "Lsrch": T is problem time limit for OZ problem: Work on each cand for time T, in order of p_i. This will take jT time, which is < $\frac{T}{p_j}$ of optimum time, T. If we had a clock j times ($< \frac{1}{p_j}$ times) as fast, — then this strategy would
.24 be as good as optimum. Trouble is, one doesn't know when to stop the trials! → (.27)

Note: In the psm in "OSS" for the OZ problem: T ← 2T was not used, i.e. Time Share was not used. — so what was done? was the psm incorrect? [Look at OSS] ; Overegon May have Copy

.27: (.24) Even so, in time jT, one does as good as optimum did in T.

N.B. (Hvr, if one has time T available, one can do k cands for time $\frac{T}{k}$ each; k can be any integer!

Approach: Do the most likely cand. for T time; It will be the best a fraction p_i of the time.
What is mean time for soln of all problems?

I think (.12) is a BIG arg. for using Lsrch for OZ probs

4/6/01 IDSIA

[redacted]

.00: Things to discuss w. Marcus, Juergen.

.01) Mechanics of Updating of GPD: (2) t. 2 parts of GPD: GPD_1, GPD_2.
.03 What GPD_2 is: what we want it to be. — what GPD_1 is & how GPD_2 is obtained from it. → (.14) (=3)

2) What "BIAS" is. Re: "Norma distribution". Any devn. from $\sum_{i=0}^{\infty} 2^{-\ell_i}$ is undesirable bias. → [Not exactly: any deviation from ALP (cc=0) proby ratios is undesirable bias.) — Hvr, for sequential prdn at least, Maximizing $\sum_{i}^{\infty} 2^{-\ell_i}$ would seem to be v. g., but one could do this in a way that gave bias in extrapolation — bias in pc ratios of next symbol(s).

Bad Bias is (perhaps) always introduced by RLP — i.e. by not summing over all codes.
We want to try to avoid systematic bias errors, by studying them. — Lots of room for research in this area. → 104.00 for some General discussn

(.14) (.02) 3) Perhaps concentrate on ①: It seems like the "Guts of the system."
∴ GPD_1 is the thing that has to be "updated"
Re: "Finding short codes" as updating technique! Go into details of backtracking in coding a sequential corpus, reason for retaining as many good codes as possible (for Proxy Revision)
.18 That BAG induction can be done same way, seems likely — but it needs
.19 to be shown. → 105.00

4) The "factor of 2" argt. — Why Lsrch could (w/ love.) be close to optimum.
(~~Hvr, the argt. of 102.10 is disturbing!~~) [redacted] (But note 102.12)
{How GPD can implement (almost) all learns.
1) Defn of All learn.pd.
2) Having any L.a. order any
3)

5) Hilbert's 24th? Kcost of ~~...~~ proof of a theorem wrt. a ~~...~~ a way of listing valid proof trials: so t. ~~...~~ integer N that gives t. no. in which this particular proof of t. theorem occurs. $K(N)$ ($K(N) > \lg_2 N$) is t. Kcost of t. proof. Practical diffty of a proof depends on known theorems & theorys. It is like "Resource limited [redacted] Kcost." It is of interest in [redacted] A.I. & t. "History of Math Science", but not much as a "Mathematical Problem" ← (But one can never be sure!)

4/7/01 IPSIA

BIAS : CREATIVE v.s. Conservative srch

[redacted]

.00 : 86.08 : On general BAYBIAS assoc w. ALP: A "conservative Search" ("non-creative") results in fast solns to easy problems, slower (or no) solns to diffic probs. (because of limited CB) ← INV probs.

for OZ probs: Quick rise to local Max. (or near local Max).

Conservative search means concentration on hi PC trials, very little wts for cands of low PC.

.07 Using random search w. cutoff: $p_i \to p_i^2$ is more conservative; $p_i = \sqrt{p_i}$ is more "creative". → .13

→ A search could start out in a conservative way, then slowly switch to more

.09 wts for low PC trials: this would have advantages of conservative search and advantages of "creative" srch. [Start conservative, slowly become creative. In fact, this maybe what good researchers do: Look for obvious solns — if none found → for more obscure solns. → Poor Researchers get stuck on "obvious" trials.]

→ Creative search usually takes longer to get solns: but when obtained (in OZ probs) they are better than those of conservative search.

.13 (.07) $p_i \to p_i^2$ or $\sqrt{p_i}$ also has perhaps similar effects if Lsrch is used.

.18

.27

.36

.00 ; 103.19 : The arg of GPD, is the probdcrn (say the function, F(x), for INV probs). It has a 2nd arg, — a cand. for soln. — its output is a pd. over the time which

.02 that cand needs for soln. GPD, (probdcrn, cand$_i$, T) → probability. (S)

.03 Or GPD$_1$(prob.dcrn) → [cand$_i$, PD(T)] output is list of cands w. PD for each that soln would occur at time T. Or output maybe [cand$_i$, cum. pd for T (= integral of PD(T) of line .03)].

The corpus [redacted] this [redacted] GPD, is based on: All empirical quadruples (in TM's Life Recall)

.08 [problemdcrn$_j$; cand$_j$, time spent on trial, success or failure.] S/F See (.02) for correspdg. Data form — all cases there are success.

. The quad in .08 has S or F as 4th component; we can add a 4th component, S (success) to .02.

For OZ probs: GPD$_1$ (prob dcrn (Including T), Cand$_i$, T, Goodness) → prob'ty Quadruple

.Corpus: [prob dcrn$_{ij}$ (including T), Cand$_{ij}$, T_{ij}, G_{ij}] Quadruples

~~one GPD for both INV & OZ problems~~ ~~this is what GPD update is~~

Next problem is BAG extrapoln. Usual form is "2 part code" (from MML MDL.)

.15 A machine, M, that contains commonal elements of corpus objects. see 81.28 for how to Apply 2141 to search Grammars.

.16 . Each object is then expressed as a string using concepts in M

The total pc to be maximized (be to be minimized) is cost of M + $\sum$ cost of each object (wrt. M)

.18 (General Example) "Stochastic Grammar" : Any Model like .15 – .18 can be regarded as a Stochastic Grammar.

.20 My "Computer Journal" paper deals w. the form of the model, but doesn't discuss rate of convergence of ALP to correct predictions using the [redacted] models discussed.

Anyway (.15 – .16) / .20 is "the model". The problem then is — how do we best search for short codes of the "BAG"?

A way that would work (but very slowly) would be to try various M types & try to code the corpus in terms of the M. T. There might be a backtracking scheme ~ to that in Sequential extrapoln. (37.09)

[In general, the problem of Grammatical Induction is not in a very good state. Nevertheless, there are many existing methods for discovery of Limited Grammar types. Tho I've worked on this sporadically over the yrs., I haven't really worked on it very hard. Recent trick (ref from Gerry Wolff may be a needed break thru. 2/0 Wolff may have found some things from which a soln. could be generated.

Also, there is a lot one can do in finding regys in Bags that would be very useful for induction, yet would not be good enuf to discover a CFG. Reconstruction and compressive & lang acquisition: A. Life (MIT Press) Spring 2000 Vol 6 #2 p. 125: Effects of Compression on Language Evolution: T. Teal — I have a copy of this paper.

General TRICK : For OT's in General & for other common sets of Prob Solving methods e.g. Taylor for impt. problem classes: List a large bunch of techniques. Factor the set into a search language — based on common concepts, similar xforms, analogy, etc. — perhaps add to search lang so it is Universal. Use search lang to extrapolate the set of prob. solving techniques.

1/8/01 IPSIA

41

~~Spacetime Bags.~~ Space TSQ Bags.

J PEG

.00:105.40: Parts of TM Most in need of Work.

.01 (1) A Complete Dscn. of System, so I can tell what needs work: (I'm close to having that)

(.02) (2) List of Search techniques for OZ, INV probs & just how they are to be implemented.
"factorization" of (OZ) into good Stoch lang. / ? 100.14

(3) Understanding → ≈"Soln" of what parts of the corpus to code for induction ($. [RLP problem] ~~85.00ff~~)

(4) Clear Statement of args about system "Optimality" — just which ~~AIB see work~~.

— also sections in which they are introduced.

Have Glossary of [Acronyms]: GPD, GPD1, GPD2
L search, INV, OZ probs ALP, RLP.
(Climit only used 1 or 2 or 3 times)

79.00 ff is toward an outline of .01

.11 Some topics not mentioned in the intro of 107.00: | OZ probs: Does G(x) have to be "fast"

(1) ~~Exp~~ TSQ's: Their qualities, how to write them.

(2) Initialization of the System: Minimal initialization: — (EOT's).
Non-Min initialzn of the GPD updater; of the set of OZ methods, & set of induction techniques.

(3) Sections: What kinds of problems are solved. (Mention that Induction probs are OZ probs — but they're dealt with in a special section.)

(4) A perhaps better ordering of sections. after introduction —

2 ~~1)~~ Induction for sequences, Bags. — Show that induction for limited resources (RLP) is a "OZ problem." — Also mention that "not always". [ISIS 96 is a good discussion for Sequential induction; Nothing for BAGS is needed! use solution 2 kind of prob. Def]

3 ~~2)~~ We will describe a system that is able to solve 2 kinds of probs. — INV & OZ. (Also Anytime problems which are close to OZ probs) Give many examples of INV, OZ probs.

4 ~~3)~~ L srch for INV, OZ probs: "optimality" meaning: How it's optimum for OZ probs may not be rigorous (early GPD bag) by 80, 09, 102.10

5 ~~4)~~ MCT and Updating Algm for GPD. How GPD is defined; how GPD2 can construct GPD. The troubles at 85.09, 101.30 (Mt Carlo Integration Mite be feasible!)

6 ~~5)~~ TSQ's. CJS: Use of Algebra so TM has "Algebra work" internally — so it can learn to map its natural lang to ≈ NAT lang about Algebra.

7 ~~6)~~ Ability of System to work on/solve any kind of problem describable in a formal langu

more detailed

— NOTE: 106.11 lists some things missing (This far) from this outline of System

.00: A (Complete) Description of The proposed Learning System

Section 1 → See 119.13–16 for some impt ideas to include in section.

§ 1. Introduction: We will describe a system that learns to solve difficult problems by being given a sequence of problems of increasing difficulty.

The system begins with a simple search technique (Levin's Universal Search) and to look for solutions to problems. The search is guided by "GPD" a Guiding conditional probability distribution. The input to GPD (its "conditions") is a description of the problem. Its output is a probability distribution on the set of computer programs that are candidates ("cands") for solutions. The probability associated with each candidate is the probability that it will solve the problem in the least possible time.

Initially the problems are very simple and solutions are quickly found, even though the initial GPD is not very clever in assigning high probabilities to good cands — but as the system matures, these probabilities become better and better.

After the initial problems are solved, the GPD is modified so that the probabilities assigned to successful cands is increased. This is not done by simply incrementing the relevant probabilities, but by modifying the entire GPD — including all information on problem solutions that have been generated during the life of the system.

Using the modified GPD, the system is able to solve the next set of problems — which are more difficult than the first set. The GPD is then modified to accomodate the solutions to these new problems.

This solution of new, more difficult problems, is again followed by integration of the newly solved problems into the GPD. This pairing of problems and integration of solutions continues repeatedly and the system eventually achieves a high level of problem solving capability. → 119.13

Maybe put ≈ 119.13–16 here
119.13–14

.37

Stop here

In following sections we will discuss what kinds of problems are solved, the details of Levin's search procedure, and how the GPD is modified.

There will also be a section on inductive inference techniques, and their application to GPD modification.

This will be followed by a discussion of the capabilities and limitations of the system — the degree to which it is "optimum" — the

108.00

9/19/01 IPSIA

.00! : **On OZ problems:** Does G(x) have to be "fast"? Well, say it's not "fast."

T = 1000 compn. steps & each G takes 10 steps.

Also a Q is : is G "open" to TM ?

Say G is "closed", & we are only able to make 10 trials on G. We then use a kind of Dynamic Prog — Hutter's method perhaps. — I think this is a lot of OZ problems.

For a sophisticated GPD, it looks at this problem & decides its a "dynamic prog" problem & solves it as such. — i.e. it has learned to analyse OZ prob dems & find appropriate solns!

Lsrch.

Cands → candidates.

:00: On Lsrch: First: What are the Cands (ideas?) (for INV or OZ probs)

A candidate (Binary) is a string, which, fed into the ref. computer, along with the prob descr, eventually causes the ref. computer to stop — usually after producing some output. Assoc. w.t. Cand$_i$ is a probability, l_i being the leng of the candidate string. This 2^{-l_i} probability often, but not necc'y $= 2^{-l_i}$.

If the cand strings form a prefix set, but other means can be found to associate (mentioned), assignment is possible

PC's w.t. Cands — whether or not they form a prefix set.

In the present system, the cands are generated by the GPD, — which also assigns pc's to them.

The assignment may or may not be associated with the lengths of the Cands.

Levin Search (L search) is a means for "Blind search" over a set of candidate solutions to problems that is guided by a probability distribution that is associated with the candidates. We will interpret the probability associated with a candidate to be the probability that it will yield the correct solution to the problem. Each candidate, and a description of the problem it is trying to solve, are both finite strings that are the inputs to a 2 input reference computer. The output is a [redacted] solution trial, to be [redacted] evaluated by the problem description.

The most time-efficient form of L search, for solving INV problems, picks a small time T and, in the "first round" spends time $T \cdot p_i$ working on the ith candidate, Cand$_i$. After working on all of the candidates having a p_i greater than a certain small threshold, it has spent $\sum T p_i \le T$ on all of the candidates. It then spends $2T p_i$ on each of the candidates for the "next round", these rounds continue until the [redacted] problem is solved. Suppose that Cand$_j$ is the candidate that solves the problem in the search. If it takes n rounds to solve the problem, we will have spent total time

$T_j = nTp_j$ on [redacted] Cand$_j$, and, since each round takes time T, we will have

$T \le$ spent total search time nT. If we knew in advance

Comment: that Cand$_j$ was the solution, we would have saved time by a factor $nTp_j / nT = p_j$ — so the search technique is inefficient by [redacted] a factor p_j with respect to the shortest solution in the candidate set considered. No!

It is clear that the total search time $nT = \frac{nTp_j}{p_j} = T_j/p_j$

If T_j is the time needed to generate and test the jth candidate then T_j/p_j is its "L cost".
L search obtains the solution of minimum L cost.

Lsrch

.00: : We will present heuristic arguments to the effect that the time shared Lsearch just described is probably the most efficient search technique possible, compared to other blind searches guided by a probability distribution. By "Blind search" we mean that the search is guided only by probabilities that were computed before the search was done. Any information obtained during the search — e.g. by observing the modes of failure of various trials — cannot be used to modify the search.

First consider a search in which each cand. has associated probability p_i, and all $T_i = T_\phi$. Clearly, the most efficient search is to try the candidates in p_i order — largest first.

If the cands have different T_i's and these, along with the p_i, are known in advance, then it can be proved that if we make trials in p_i/T_i order, largest first, our expected search time will be minimized.

A heuristic argument for this: That in picking the cand with maximum p_i/T_i from the remaining untried candidates, we are acting most efficiently, since we obtain the most probability of success, p_i, per time expended, T_i.

In time shared Lsearch that we have described, while we know the p_i's of the candidates, we do not know the T_i's. Instead, if we have spent total time T_i' on cand$_i$ without success, we know that its p_i/T_i is $< p_i/T_i'$ — since T_i' must be $> T_i$. We have for each cand an upper bound, $\frac{p_i}{T_i'}$, on its value of $\frac{p_i}{T_i}$. Since we want the candidate with largest $\frac{p_i}{T_i}$, our best choice is to choose the one with largest upper bound — i.e. largest $\frac{p_i}{T_i'}$. The result of such a choice strategy is that we spend time on candidates so as to keep their $\frac{p_i}{T_i'}$ about equal — which is the strategy used in the time-shared Lsearch that was described.

Space (13.00

4/10/05

IDSIA

.00: ~~Updated the time-based Lsearch that was described.~~

.01 [SN] 1) For integration of GPD₁ → GPD₂ try Mt Carlo integration combined w. Mt Carlo Gaussian Lsrch.

2) .01 might be used to deal w. correlations betw. the p_i — in INV's in OZ in particular

3) For INV ~~...~~ a/o OZ problems: Correlation might be dealt w. as a "WON" (What to work on Next?) problem: After we've unsuccessfully worked on a problem for a while using candi, candi's ~~...~~ p_i could ↓ (via the S.I. ~~...~~ (updating)) pgm.

4) Could "Planning" (AND/OR nets; WON) be integrated into Lsrch s.t. p_i are changed via S.I.? As of my present approach to Lsrch — it looks like the "OR" net ~~...~~ for WON that I "solved".

→ 5) The args for optimality of Lsrch (working on cand w. hyest $\frac{p_i}{T_i}$) on INV probs do not obviously transfer to OZ probs!

6) Probly best to write up report in view of present developments & just write list of "Bugs" i.e. "Bugs". Buglist

.19:(102.08): 7) For the OZ method of 102.04–.07. Say $cand_k$ is the cand that in time T gets ~~...~~ for best $G(k)$ of all cands considered. Then if we use 102.04–.07 for time $\frac{T}{p_k}$ we will do as well as the best alm ($cand_k$) does in time T. What we want to do is Get GPD to assign as hy a p_k to $cand_k$ as poss!.

If GPD assigns p_k to $cand_k$, is that best we can do? (Maybe (in short term) No! viz objection 102.10 — But maybe in "Long term" 102.12)

.28:101.29 8) We can evaluate $\int_{-\infty}^{+\infty} dG\, p_1'(G) \cdot \prod_{i=2}^{\infty} P_i(G)$ ← This can be done for INV as well as OZ problems! by ~~picking a set of $P_i(G)$ values~~ Generating a set of ~~...~~ G_i values by Mt Carlo $P_i(G)$, then picking the i ∋ G_i is max.

This gives a Mt Carlo d.f. over the $cand_i$'s which can be used in time shared Lsrch, to tell

.31 which cand will be given a fixed ΔT of time.

In fact This is easy to do if the $P_i(G)$'s are uncorrelated. If they are correlated (and they are) I don't yet know how to deal w. it — first, I don't know how to discover & express it as an equation/data; If I did perhaps the forego. Mt Carlo method would work.

How it will work when p_i's change during the search is unclear — it will work but I suspect it it will not be as good as the strategy of 62.19 + (...) → (116.00 spec.

Lsrch

.00: (Spec 111.40) : That time-shared Lsearch might be near optimum for blind search is suggested by the following arguments. In blind search, all of the information to be used in the search, ~~is contained in the guiding probability distribution~~ can be contained only in heuristic search techniques for reordering the random trials of the candidates. Any reordering of the trials can probably be implemented by suitable modification of the guiding probability distribution — i.e. assigning high probability to trials that are to be made earlier.

If we do not use "blind search," then the information obtained in each trial can ~~choices of trials changes as~~ influence the subsequent ordering of trials. This Non-blind search can be implemented by a modification of Lsearch, in which the guiding probability distribution is modified by any trial. In this case we use time shared Lsearch as before, but we work on (spend time) as before on that candidate that has maximum P_i/T_i'. This work reduces its P_i/T_i'. We then continue working until this candidate no longer has maximum P_i/T_i'. We then work on the new candidate of maximum P_i/T_i'. We continue in this way, working on the candidates of maximum P_i/T_i', until a solution is found.

While this technique is clearly superior to blind search, it has (so far) not been possible to make strong arguments for its optimality.

We have described the application of Lsearch to Inversion problems. For time limited optimization problems, the technique is very similar,

Perhaps modify and say it later: However even for blind search, it is not clear as to what extent this technique is near optimum.

Time limited Optimization problems are described by a function, $G(x)=y$ that maps strings to real numbers, and a time limit τ (tau), for the solution time. As with Inversion problems, we have a conditional guiding probability distribution that take as inputs the problem

Lunch

.90 : 113.40 : description, $G(x)$, τ, and a candidate for solution $cand_i$. Its output is the probability that $cand_i$ will, when inserted into the reference computer, along with $G(x)$ and τ, will produce an output x in time τ, such that $G(x)$ is greater than that produced by any of the other candidates.

Summit: We will describe two methods of doing Lsearch for time limited optimization problems. They are very

We will first describe a method for doing Lsearch on time limited optimization problems that is very similar to that used for Inversion problems.

We are given the time limit τ. We divide it by perhaps 10 giving $T = \tau/10$ and do 10 "rounds", by working for a time $T p_i$ on $cand_i$, over all of the candidates whose probability > a certain threshold. We keep records of the final $G(x)$ score of each candidate for each total work time ($j*T$, $j = 1, 2, 3, \cdots 10$).

At the end of time $10T = \tau$ we will have (at least) one $cand_i$ with maximum $G(x)$.

A second technique is a special case of the first: instead of doing 10 rounds it just does one round with $T = \tau$, and obtains the same "best candidate".

The first method obtains the same results as the first, but uses much more memory to store partially tested candidates. The advantage (if any) of the first method, is that if we allow the Guiding Probability distribution to be updated during the search, the first method gives us better opportunity to switch between candidates as their probabilities change.

~~Both techniques yield the result that~~ In both methods, $cand_i$ spends $p_i \tau$ looking for a solution. If $cand_j$ really gives the best $G(x)$ value in time τ, then, since we spent time $T p_j$ on $cand_j$, the system is slower ~~by a factor of p_j~~ than the best possible candidate, by a factor of p_j.

As with the Inversion problems, if we use blind search this may be the best that can be done, if all of our heuristic information is in a guiding probability distribution that is invariant during the

Lunch

.00 11(4) : search. If we allow the Guiding probability to change during the search, — to allow information gained during the search, to be used to modify probabilities later in the search, then the system will be certainly improved, but, as with Inversion problems, the optimality of the system [illegible] is not (yet) certain.

The L search with variation of probabilities during the search, would use the "many rounds" method of search and would operate much like the technique proposed for Inversion problems.

Perhaps make more explicit as to just what T_i' & T_i are) — what times are counted? is time to generate & store cands, counted? Is time for G(x) or F(x) computation counted? / O.K.

~~For both kinds of problems, the "working time" associated with a candidate~~ begins with ~~the time between its insertion into the reference computer and the time for the computer to stop~~ does not include time used by the Guiding Probability distribution to generate the candidates, nor does it include testing the output of candidate program.

[4/13/01] Assoc w. each prob are 3 costs: 1) ~~Generate~~ Cost of Generation of Cand by GPD (also cost of Generating p_i of cand.)
2) Cost of running Cand to obtain ~~cand~~ ~~trial soln~~ cand soln. 3) Cost of testing Cand soln.

My impression: I want to include 2 & 3 in T_i "in $\frac{T}{p_i}$". Both occur ~~after~~ p_i is available. Even if 3) is common to all cands, it should be included — Consider case in which 2) & 3) & p_i are known for all cands. Then $\frac{T}{p_i}$ order is correct w. $T = 2) + 3)$.

My impression that if 1) is very costly, it will not ~~be generated by~~ GPD then, in general, I may have to look at the entire generation & testing procedure to be sure of what prices are allocated to what decisions —
What about ~~T~~ ~~2T~~ Lsrch? — we repeatedly generate Cands.

→ 121.00

IDSIA

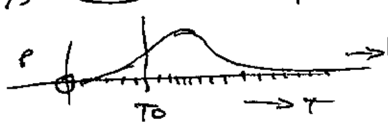
(SNs)

.00: Spec 112.40: 9) On implementing the t.S. M+Carlo search of 112.28–.40.

In t.S. search, we need a lot of memory to store partial solns for each Cand'z'. We use a Disc to store most stuff. RAM is ~~m~~ a Holds, A, B, while calcns are being done on A, B is being swapped w. the Disc.

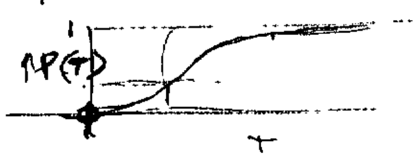
In t. M Carlo method, we generate a large set of trials specifications: this is Bag of Cand indices (≡ "Names") obtained by the MC pgm of 112.28–.31. Then one by one, we swap sections of the Disc onto RAM & spend the specified ΔT's on each of the proper cands. Since the name of a Cand is an integer, we can easily tell which "swap section" of the disc it is on.

.15 HA! In t. fur[?]ory, we can easily add a [?] from: In INV probs from

.16 th. GPD for Cand'z',  $\int^{\infty} dT\, P(T) \equiv 1$.

we know that Cand'z' has been run for time T_0 w.o. success. This fact can be used to modify t. p.d. used to determine how often it has min time of all of t. cands.

.18 In .16, given t. p.d. of ~~p~~ f[?] v.s. completion time, — how do we get a Mt Carlo d.f. for T from it? well use a cumulative pd!



Then, using a random no R betw. 0 & 1, (do $P^{-1}(R) = T_R$.) I'm not sure this is right!

(How do we get t. cumulative DF & its inverse?)

"Look at this later"

GOOD! One way: take curve of .16 Along T axis, integrate. Every time $\int$ increases by .01, make a mark on T axis. This will divide the axis into 100 sections of equal probability. The 100 mid pts of t. sections are 100 T values of = probty. If we later find out as in .15, that T must be > a certain T_i', we just elim all of t. 100 T, below that value, (say r pts are lost) we then picks random integer betw r & 100 to chose a Mt. Carlo T value

(Section 5) → When inserted into the reference computer along with the problem description, will give an output that solves the problem faster than any other candidate

.00: **UPDATING The GPD.** (103.02, 105.00–.40)
Guiding Probability Distribution (GPD).

The GPD is a conditional probability distribution with 2 arguments:
The problem description and the candidate for solution. For INV problems, the output is the probability that the candidate will produce

GPD (problem description, cand$_i$) = Probability that
M_R (problem description, cand$_i$) = x, F(x) = y and time for the M_R computation is minimal for all candidates that solve F(x) = y.

For OZ problems, we have the same two arguments. The problem description is the function G(x) and the time limit Tau. The output is the probability that when the problem description along with cand$_i$ is inserted into the reference computer, its output at time tau will be an x such that G(x) is greater than that produced by any of the other candidates considered.

The GPD is obtained from another conditional probability distribution GPD_1. GPD_1 has 4 arguments:
The first is the problem description
The second is cand$_i$
The third is the time spent by cand$_i$ on the problem.
The fourth is, for INV problems the symbol "S" (meaning "success")
for OZ problems, it is the best value of G(x) obtained in time T, by cand$_i$
The output is the probability that the first 3 arguments will produce the fourth.

GPD_1 is an empirical probability distribution. Its data takes the form of a BAG. The induction technique that produces the probability distribution GPD_1 from this BAG, is described in Sol( ) "Two Kinds of Probabilistic Induction". It is a type of OZ problem that is solvable by the system.

.00: 117.40 The BAG data elements are of two kinds; one kind from INV problems and one kind from OZ problems. The data from INV problems takes the form of a quadruple of objects that describe a trial solution to a problem that has occurred.

(1) problem description
(2) cand j
(3) Time spent on trial: T
(4) S if successful, F if not successful.

For OZ problems we have the same first 3 elements, but the last element is the value of G(x) obtained for cand i working on the problem for time T.

Each solution trial contributes to this data set. Each quadruple is an element of a BAG of data.

Since induction is an OZ problem, the system is able to take the data BAG and produce the corresponding probability distribution, GPD_1.

GPD is produced from GPD_1 by integration. Consider GPD and INV problems.

For a given problem, each cand has a GPD_1 distribution giving probability of a solution in time T. We want, for GPD the probability that cand will have the smallest T of all candidates.

If $P_1'(T)$ is the probability that $cand_1$ will solve the problem with time T, and $P_i(T)$ is the probability that $cand_i$ will take longer than T to solve the problem, then the probability that $cand_1$ will have the shortest solution is

$$\int_0^\infty dT \cdot P_1'(T) \prod_{i=2}^{\infty} P_i(T)$$

i.e. Integral [0, infinity, dT, P'_1 (T)* Product {i=2, infinity, P_i (T) }]

(sq. … inc. … sq … curly … curly)

Try to put it on one line.

Write it in this form, but leave room for me to write on the paper so it can be Xeroxed. sub 1 GPD_1

If the arguments of GPD describe an OZ problem and an associated candidates we integrate over the values of G(x) achieved by the various candidates, rather than the T of INV problems,

119.00

3431
51

11 (w. Goncalves)

.00 : 118.40 : It may be difficult to approximate this integral. There is, however, a new Monte Carlo technique that generates GPD from GPD, in a form that makes Lsearch easy to implement. (perhaps describe in appendix) — Mention the re-report. ≥ (44 22 53 1147 1148) 144 50 83

Perhaps more impt. at present, than .00 : Give a better idea as to how the particular OZ problem (that GPD is) is solved by the system.

Perhaps show how the BAG example works: illustrate the "codes" to be minized? (sum to be maxzd) ? → But perhaps this should have been shown in Section 2 "on INDUCTION". For doing Bay induction, maybe use quotes from "Q 2 kinds of Prob. ind".

→ 107.00 In Introduction

.13 Also, early in report: Mention that Lsrch is not much good by itself that Lc is too large that (slow) models (long) of pi's is accury.

Desc 2 levels of prob solving Ruff (pi assignment by GPD) } in introduction → 107.00

.16 fine (By Lsrch)

.17 For 2 .17 The system may be regarded as having two levels of problem solving — for first, rough approximation the GPD that looks at a problem and gives a set of guesses at possible solutions by giving a probability distribution over candidate solutions, probabilities to guesses. At a finer level, Lsearch uses the so probabilities to do a detailed search for a solution. → (108.03)

.26 We have defined the GPD as having two inputs: the problem description and a solution candidate — its output is the associated probability. In practice we will use a form of GPD that is mathematically equivalent to this, but more amenable to Lsearch. This GPD has only one input — the problem description. Its output is a list of candidates, paired with their associated probabilities. The pairs are roughly in order of probability — largest first. — maybe combining curves of .00.02

The Guiding Probability Distribution (GPD) can be realized in two equivalent forms. [redacted] One takes the form of a function of two arguments: the problem description and the candidate — its output is the probability associated with that candidate. A second functional form, which is more amenable to Lsrch, has only the problem description as argument. Its output is ...

In our discussions of GPD, we will sometimes use one form, sometimes the other 121.00

1/12/01 IDSIA

TM General ! BAR GRAPHS: Gabriel KRON (Tensor analysis of Networks).

.00: 4/11/01 Jean Thoma (U of Waterloo) gave a talk on BAR-Graphs:

This is a way to represent Electrical ckts as analogs of Electrical ckts, so its easy to compute their behavior. In this it is ~ to Krons "Tensor Analysis of Networks".
The "Analogs" have been worked out in many domains — Mechanical (of course), but also chemical and thermodynamics, (≡ continuous params. also Non-Discrete parameters & Non-linear stuff) Kron has done it for Gen Relativity (perhaps ??).

There is a pgm "20 sim" that computes behavior of bar graphs. —
Perhaps SPICE could be given appropriate modules to do simulating for every thing that BAR Graphs can do — or that TAN (Kron's "Tensors") do.

This would greatly expand the utility of Koza's Ckt analysis pgm via Genetic Pgmng. It also suggests that Koza's "Problem domain" for his Electrical Ckt networks, is very large, much more general than supposed.

A criticism of Koza's results has been that he had to figure out this rather "non-intuitive", "very special", way to represent his problems. It appears that the method maybe much more General.

4/12/01
810A
112 49?
67
125 53
80

Lsrch

.00:15.40 : In comparing Lsrch to "Heuristic blind srch", it sounds is a "straw man": All it can do is apriori 'list cand in order to try. Lsrch uses more info so it's much more likely to be better. If we allow Heur' Blind srch' to use info on trial times for trial cutoff (at same trial only) — it's likely that Lsrch is better.

.04 One may now non-blind heur search as having a new ordering of trials, after each trial (the new ordering being modified by each trial.). After each trial, compare NBHS (Non-blind Heur srch) w. ALS ALsrch (Augmented Lsrch: that allows changes of proby during srch). Assume they are both in the position of Blind srch v.s. ordinary Lsrch, at the first (or any later) trial. — And it would seem that

.09 Lsrch would be at least as good that situation.

Doffs

ABCDE abc

~~P = of Lsrch. Heuristy If we do not use P~~

Perhaps have (cands & problem) instructions machines for INV problems; output is X and S or F; for INV, outputs X and G(X). So maybe its useful for Inv. probs, if F is one output X is not needed for other output (?) — for "updating info".

.04 – .09 suggests a way to look at (NBHS Heur Srch) v.s. ALsrch. Perhaps an impt. difference betw. the 2 is that NBHS is able to make quick (cheap) decisions in resolving near (and distant) past trials. It would seem that it's harder (more expensive) to do updating of GPD. [redacted] This sort of thing

→ perhaps find way for GPD to do fast "Micro updating" — It would seem to be "legal", since can be done in ALS (or maybe even Ordinary Lsrch) inside a cand;

Since a cand can be any pgm — learn to be any pgm.

At any rate, it would seem that ALS would speed-up (perhaps a lot) the aquisition of these arbitrary srch methods by the cands.

I'd like to find (a) mechanism(s) for TM to be able to find (Make it more likely for TM to find) cands of arbty [redacted] Power) — It is my impression that the present model does make this possi. — perhaps encourages it — But a TSQ oriented in that direction would certainly speed up things a lot!

RLP does involve selecting parts of the (Sometimes small) of the corpus to code (Soup (25000 ...) ... problem) PP

4/13/01 IDSIA

.00 : ~~123.40~~ 121.40 : A Major Q is whether the present system (w. reasonable initialization) & a reasonable TSQ could learn enuf to really begin serious self improvement.) Also note the more general system of 123.25

So best draw up a minimal system, w. ~~comments~~ suggestions on improvements & comments about weak pts., etc.

The present "report" should be a strong step toward such a "minimal system".

4/14/01 IDSIA

Misc

.00 1) An early part of human/animal training: to be able to decide what parts of envt. are relevent to the present problem: ∴ what parts to code, first.

2) In present approach of TM being written up: I feel that it is close to isomorphic to how such (is possibly better/more General): that I should be able to explain how to solve any problem I can solve as Moore Such in. I know/knowable heurs — ∴ I should be able to get present system to do it — w. perhaps small additions/modifications.

.09 3) In using Ag Lsch (ALS): we work on cand w. largest $\frac{P_i}{T_i'}$ & $\uparrow T_i'$.
$\frac{P_i}{T_i'}$ $\frac{P_i}{T_i}$ (since $T_i' < T_i$) so $\frac{P_i}{T_i'}$ is an upper bnd for $\frac{P_i}{T_i}$.
If we assume any a priori pd for $\frac{P_i}{T_i}$, then if one has an upper bnd on the $\frac{P_i}{T_i}$ of each cand, then the one w. largest upper bnd, has largest expected value of $\frac{P_i}{T_i}$ — so work on that first. While this seems reasonable, I'm not very happy w. it!

4) I think an impt. weak pt. of argt. about optimality of ALS for both INV & OZ probs is that the p_i are correlated — in OZ probs — these are the [illegible] p_i's of OT's — & they tend to be highly correlated. — This Lsch model as drawn to date does not consider correln.

.25 5) A very General Prob-solving system that can get to arby level of intelligence w. suitable TSQ & enuf time:
a) Set of problem types solvable is universal. (QA problem form a universal set.) ("subset of") (OZ probs form a universal set)
b) Ability to know (learn) when one PS method is better than another.
c) Ability to take parts of good PS methods & use them to form new trial PS methods.
Try to formalize this idea; make the concs. very precise.
d) The problem, then, is to get the system on the "royal road" as fast as possl.
e) The reason a, b, c are "Universal" is that the machine can usefully work on problem of SI. GA is an adequate set because QA is an OZ problem (I guess: But unclear now as to whether all induction probs are OZ probs!) — Because of the what part of corpus to code "problem" (?) — Any problem can be framed as a QA problem.

Section 4.1 [redacted] or 3.1 : How Problems are Solved.
or : CANDIDATES

100: ~~104.40~~ :

The [Cands] have been dscbd as finite strings with associated Pi's accrd by GPD. Just what does output of GPD look like and how does it get that way?

In "steady state" after f-system has been running for some time with a good TSQ, the set of [Cands] for a problem will be any ptrs ppms or (derns of computer ppms) we might use a variety of LISP, (or any other functional lang) (as it has a universal pp F) because all strings Fns. symbols of the language are meaningful. This incr increases "hit rate" a lot!

[It may not be necy to have recursive definitions. One can generate a rather large set of useful functions (about all functs usd in Science) w.o. recursion — we would have loops, hvr. (Maybe always simple Do loops w. indices of specified range) Examine (Cleave's book: work by Z Rosa Peter) — simple characterized prim. Rec. functs]

[Anyway, the cands will be (usually) simple functions of a set of functions that the System has found "useful". Functional langs make this easy to do.

GPD looks at a problem dscn, and assigns Pi's to all possl. cands. — One simple way it could do this: assign probs to each of Pi "defined functs".

.23 Useful functions are commonly used functions! One Way to find them: look for common subtrees in the trees that define useful functions. Use ~~ALP to determine~~ "compression" to determine if a definition

.26 is "Useful" & give it a [redacted] Pi.

.27 [[It would seem that the forgy. hunt for "regularitys" would not cover all possl. regs: We'd like some More General Algms!]] (I think I'll be solving Pi's into Pass yr or so. I don't think I really solved any Regs, hvr.) → 128.24, 127.00 also, 126.09

Z(4.1) Is only one of many used in L search for induction → 128.24

.29 In the forgy. we have to be careful to include OSL. (The Z(4) includes OSL's the forgy. is a form of Z(4.)) → note 126.03-.06

One impt induction force not included in forgy (apparently): Analogy — Similarity of structure (?) → The we can get this by ~~similarity in~~ noting common subtrees; Also, by modifying a tree so it's equivalent to itself but uses more common subtrees. } "good":

129.00 on § 3.1 or §4.1!

126.00

(distillation) Corpus reduction problem

T. (Backtrack) Corpus Cutting Problem

Cutting, Partitioning, Specification, Range; Processing, Sectioning, Selection, (reduction); Focusing, Backpack, Discard ... rejects, (Windowing)

.00 : On the problem of [85:01] [Theorem in Basic paper: Why induction is not just finding codes $\geq \Sigma 2^{-l_i}$ is max]

.01 GPD has as input the Induction problem domain: Corpus & "what is to be predicted" & Time available.

.02 Output is Cand$_i$, P_i: Cand$_i$ gives method of soln. Usually it will do this by telling what part of corpus to code, & how to get ~~most~~ max $\Sigma 2^{-l_i}$ in [redacted] available time.

The relevant Goe is based on (post hoc) — how hi a pc was given to what actually occured.

.05 So this trick will work on past corpus only. But more generally, the cand$_i$ can be

.06 any pgm that looks at [Corp, what is to be predicted, & time available] & produces a pd on what is to be predicted. It's a (conventional) Stoch operator problem. As in the past } See (.34) for a ≈ soln.

we solve it in the usual way by adding lengths of codes of d-cruns.

T. apparent difficulty w. .05 ff is that the stoch operator problem is again an "induction problem" — So we are (apparently) back where we started! Maybe not quite, because we are now in position where this is the only induction problem we need to solve — Its a GPD for all induction problems. Also, ~~it~~ we may be able to use a recursive definition ~~defn~~, since we are able to solve the problem in many cases (~~...~~ Rosom which we know how to code the entire corpus). — But "recursion equation" doesn't seem to reduce the complexity of the ~~...~~ defined object! | ~~...~~ .01–.02 (.06)

Central to soln. of this difficulty, is the Algm. that goes from [Corpus T, ~~...~~ what needs to be predicted] → to a pd on what is to be predicted. There are many such algms; we want a pd on these that is periodically updated. It would seem that a pc for such algms would be the app of the algm (initially ~~...~~ $\approx 2^{-l_i}$) mult by the pc of what actually occurd.

.27 T. algm. need not be simply an ~~...~~ "address" (location indicator) of part of the corpus.

.28 It could be a specification of what fraction of time to spend on what parts

.29 of the corpus. — It could even be an [non-RLP type of prediction!]

.30 [.28 could be easily done ~~...~~ (fraction of available time spent on each sub corpus) by using several "corpus cutting" algms" in ∥ ~~...~~ assigning ~~...~~ pc's to them — which end up being fraction of ~~...~~ available "Time limit" spent on them.

(.34) So a first approxn. to a soln. of (.06). Each algm. simply specifies what section of the corpus to code, w. max $\Sigma 2^{-l_i}$ criterion mult by pc. of the "cutting" algm. The (Specification, partitioning) Cutting algm ~~...~~ is a function of {t, Problem dom, corpus}. We obtain the pc of a cutting algm by applying it to all past problems & multiplying ~~...~~ all assoc. pc's

.39 A serious problem is updating the (cutting / reduction) algms; w. a given time (limit), how ~~...~~

.40 ~~...~~ many past cases should one consider? One might apply the algs to itself "dist alg"

.00: 125.40: to decide this. For a small corpus, "radiction" ("distillation" .. .. algm) would do whole corpus: — as size of corpus ↑, the algm would begin to specify smaller parts of the corpus —

.02 which would continue as the corpus grew.

.03 Perhaps the "best pred pc" would be a rig. Gave for pred in general; It may include OSL, which RLP may not! (but Z141 does include OSL)

.06 (Actually, the [redacted] (OSL) effectiveness of RLP v.s. Z141 v.s. (MML) MDL) is something I'm not at all [redacted] certain about. I don't think it's a hard problem, hvr. → 133.00

.09: 124.29: I've been thinking of "coding & recoding": so we end up at an early stage w. a Lisp function, and we want to code this function more compactly.

.11 → A functional lang. seems ideal for direct applicn. to "operator induction". One way to do it: we have a "common area" w. lots of definitions. Then each I/O is coded in the lang. using whatever set of the common defns, it desires. Is this the most genl form of operator induction?? In my usual operator induction, I have a 2 input mchne. One input sets up the machine (corresponding to "containing the definitions" — The other input ≡[Ii] uses the defined machine as an I,O device — w/ (extra) aux input giving perhaps "corrected errors", for the direct I/O operation of the common machine on the [Ii] set. Perhaps a 3 input machine ① for common machine data.

.20 ② for Ii input ③ for aux info to get output from Ii & the "machine".

Another very impt form of Induction is Bag induction in the "Z hmg of prob ind" paper. we have a 2 input machine as in .11–.20 but the [Ii] input is constant (≡Λ). Using (functlangs "Lisp") we have a function defined by the lang, & we have to put in various inputs to get the Bag members.

.27 One way to get ideas on how to "fill in" this domain of what occurs: to get TM to "do" a TSQ, & see just how I would do the TSQ & how TM mite do it & how this reflects on the problems of detailed implementation of .09ff.

——— Perhaps another way to think of (.27): Z141 and a functional langs & common subtrees in functions give a certain kind of induction. So try them on a TSQ! See what other heurs humans seem to use, then see how these heurs can be expressed by modifns of Z141 what we want is for the set of things we look for to be equivalent to a universal lang. — so, in theory, any regy could eventually be found.

4/15/01
IDSIA

.00: 126.40 : On the Q of [124.27–.29]: is augmented Z(4) (124.23–.26) able to find all regularities in a corpus? The usual way to find regularities! Start w. small corpus — use Lsrch to find short codes. Then use Z(4) to code common elements in short codes for different parts of corpus.

Or just use Lsrch when corpuses short. Somehow, use Lsrch to add on codes [redacted] to augmented corpus. When a new chunk of corpus occurs, we first try a simple addition to the code for the old corpus. If this does not work, we backtrack & try recoding a larger chunk of the corpus — using perhaps Lsrch & perhaps retaining as many of the old definitions as possible. We keep backtracking until we find a suitable code.

.11 When a corpus is augmented, we try to continue using the old definitions — & we update pc's of the defns. We may make new defns. Backtracking consists of undefining some things, defining new things and reparsing old (& new) corpus wrt the new defns. (Also, if a corpus augmentation results in some new defns, the old (& new) corpus should be reparsed.

While the forgo seems to talk about sequential corpi — the approach also works for Bag induction, in which we sequentially add elements to the Bag.

I think all this is pretty much what Science does. Theory/revision (Big) involves backtracking, & occasionally massive recoding w. new defns & reparsing. The "backtracking" can involve recoding of non sequential parts of the corpus. The "distilling" (partitioning) of the corpus (125.00 ff) is something TM learns to do — it involves corpus partitioning — & Backtracking may often involve partitioning the corpus to decide what "parts" to recode.

The foregoing "Backtracking" method is only one (Most popular) method of Theory Revisions; there are probably others. A Big "adjustable parameter" in the method is the partitioning of the corpus. (≡ "distilling"). As in the distilling problem, we have to find a (usually unstructured) sub-corpus of stuff relevant to the recent "Disturbance" ("Anomaly") [An anomaly is something that "contrary" doesn't "fit" as well as previous stuff (I'm not clear on exactly what this means) (how to formalize it) (Explicate it)

.35 (SN) Q: Are "Sci laws" same as "Definitions"? In a sequential corpus, a "Scilaw" might be: "the seq. abc is always followed by f(a,b,c)". This is of some import, because many regularities take the form of "Sci laws". Consider the seq "abcd." Since abc is usually followed by f(a,b,c), we invent the object abc, f(abc). To define it we write, f, a,b,c. "f" is much more likely than d. Or write a,b,c,"f"; the function f is very likely after abc.

128.00

4/16/01

IDSIA

.00:27.40: I'm not at all clear on how this is done by defns/parsing.. : Just select t. codes.
Much of science involves finding things that are (usually partially) functions of other "things" or Sets of "things". In simple Z141, we make defns of objects by combining other primitives or previously defined objects. In scient. laws, we could define a descr. what we do in this way, but usually we don't descr. it that way. Also t. sc. law "objects" are not "compact" — t. parts may be separated from one another by "not-relevant" parts of t. corpus.

(SN) A part of t. scientific process that I've not looked at! When a sci law is suspected but SSZ is too small for verifn., we do "experiments" — i.e. we obtain more data: This augments t. corpus in a very special way — oriented toward verifn. of t. suspected "law". It is t. kind of action that must be taken by a T.M. that specifically is able to interact w. RW. I think I can set up a [redacted] simple INV/OZ problem TM to work probs of that kind.

SN Marcos' "Paint a Good Picture" problem: A formulation as a OZ problem, we are allowed 20 trials: these are paintings presented to "Teacher" who gives each a score. The "20" is not the "time limit parameter"; instead, we are allowed a total of 10 hrs for completion of painting. The "cohn" is t. T limit. This problem would be solved probably as Marcos solves it — as a Dynamic Prog problem — but I think Marcos doesn't know how to get a practical soln. Another formulation: No limit on no. of trials, but each trial cost 20 minutes; so In 10 hrs we could have 30 trials max unlikely, but no heads completion. Goal is max score. Score may be total, or for t. best painting thus far.

.24:27.00,40 Re: Use of Z141 for induction: OK, but it doesn't seem related to Lsrch. .25 → Unless we regard Z141 as "just another O.T. for induction problems", Perhaps that's it! Z141 is merely 1 of t. induction O.T.'s used by t. system. (Because thing is used for any modelled corpus, the Z141 method tends to be "biased".)
.26 Another O.T. for induction would be straight Lsrch for direct codes. For augmented corpus, use that Lsrch w. backtracking of nodes. (This can be faster than usual, if we retain many codes of a corpus). — See 87.09 ff also 87.27 - .32
Note that one may be able to retain at all times, t. best 1k or 10k or look codes for t. corpus! — This may not require very much memory, because much of these codes is common: T. codes themselves will be a binary tree & I suspect that t. best 1k codes will have much in common.

4/16/01

IDSIA

p 70: 3/17
p 128: 4/6
see p. 58 ed.

Section 4.1 (maybe) to follow portion updating.

.00 spec 24.40: We have discussed the system on a rather abstract level, giving a general picture of its operation, but not much detail. This section will ~~discuss~~ show [redacted] how the system might solve induction problems — by describing some of the candidates used for such problems.

.06 4.1.1 Use of direct Lsearch for sequential prediction.

Here we use a Universal Machine with ~~unidirectional~~ unidirectional I/O and a bidirectional work tape or Random access memory. We want to find short codes for the training string, S. The string "a" is considered to be a code for string "b", if Ram machine produces an output with prefix "b", when it is given input "a";

OMMIT

We start out by finding all codes for the first bit. We save all of these codes, and use extensions of Ram to code the second bit. Again we save all of the codes and use extensions of Ram to code the third bit. This continues until we have saved n codes (n maybe 1000 or more) — at which point the system saves only the shortest n codes for the corpus. — We continue to code sequentially as before.

~~During most of the coding~~ We find codes sequentially, using Lsearch to terminate trials that take too long. Say we have coded up to the n^th^ bit. We save all codes for the n bit prefix of the corpus, unless we have more than k codes — in which case we save only the shortest k codes. As we augment the corpus by a bit, we try all continuations of the k codes in storage, and retain the shortest k codes that work.

This process continues until the entire corpus is coded.

This technique is not particularly fast, but it will get fairly "unbiased" codes — and perhaps relatively short codes if k is large enough. For existing computers, k values of 1000 are not difficult to obtain and probably much larger values can be used. The k codes can be usually stored somewhat compactly because of their tree structure — since the codes will have many sub ~~trees~~ in common. — 153.00

.27

4.1.2: Again we code a sequential corpus, using primitive symbols in some (not necessarily binary) alphabet. We look for the adjacent pairs of symbols that occur most unusually high frequency and define this adjacent pair by the symbol, alpha. Using the formalism of Sol 64 part II, this gives compression of the corpus. Using the primitive alphabet augmented by the symbol alpha we again look for adjacent pairs of most unusually high frequency and define it as beta. We then reparse the sequence using the symbols alpha and beta to give maximum compression. We continue to find new symbols and reparse the corpus, until no more compression is possible.

This scheme was described in Sol 64 part II, but it did not work because ~~reparsing feature, it did not work~~ the reparsing idea was absent: it lacked the reparsing mechanism.

.00: ! From some papers by Gerry Wolff, who used it effectively to English text that had no interword spaces, insert interword spaces in . . .

The technique is fairly general and could be applied to many different kinds of induction problems. Functions in a functional language (such as LISP) can be coded as trees. A set of functions can be extrapolated by defining common subtrees, then rewriting the functions using the newly defined subtree. The resultant set of functions is then searched for new common subtrees to be defined, and the set of functions is reparsed using all of the defined subtrees.

This continues as with the sequential extrapolation problem. The set of primitive and defined functions, with suitable associated probabilities, then constitute an extrapolating stochastic grammar.

The method is applicable to evaluation of "goodness of fit" of context free (or context sensitive) grammars to a corpus, but at present, I don't know of any good methods for finding candidate grammars to evaluate. — It may be that Wolff's reparsing trick can be used to help solve this problem.

4.1.3. There are many techniques for finding regularities in data. Each of them can be regarded as a candidate for a suitable induction problem. One way to use them would be to consider each of them to be a "Black box" (not transparent Solomonoff systems), and assign each an initial equal probability to each of these candidates.

After trying these candidates on very many problems, the GPD would acquire some ability to look at a problem and assign a probability distribution of candidates to it.

A better method would be to use the same candidate techniques, but not as "Black boxes" but as acceptable sentences in a stochastic language. The elements of the grammar would be functions or constants common to some of the techniques. A grammar of this sort could assign probabilities to new trial techniques constructed by the grammar, as well as the original techniques from which the grammar was constructed. The stochastic grammar serves as an extrapolating method for the objects that gave rise to it.

.00: 130.40 This scheme for extrapolating cjud techniques can be used to expand any set of problem solving techniques initially installed into the system by the designer. In some cases certain elements may have to be added to the grammar so that the language described by the grammar is a (Turing) Universal

.04 set of computer programs. This would be particularly important in the set of candidates

.05 for updating the GPD

§4.1.4 d? Initialization of the system:

Initially, the system knows how to do Lsrch for INV and OZ problems.

comment: The candidates set for all Inv. problems and all OZ problems (but one) can be the same!

Its list of candidates for updating the GPD should be as good as possible. Sections 4.1.1, 4.1.2 and 4.1.3 suggest some techniques for getting a good set of candidates.

For all other problems — INV or OZ, the initial set of candidates is not as important. Any complete set of programs to a (Turing) universal machine can be used.

To speed up the machine's development, however, it would be well for the system designer to initially install problem solving techniques for various problem domains as described at the end of Section 4.1.3.

Usually the resultant speedup of development will be associated with "BIAS" in the types of problems it can and cannot solve.

.00: #31.40 : What needs to be done on the system? P107 is introdu...

1) I really need a better understanding of just how it works problems! How it reacts to a TSQ

2) The Updating Algm. certainly needs work. I have formally desc t. problem, as an inductive problem, but t. details of how it solves t. extrapolation of this bag of quads, is not altogether clear to me.

.09 • 3) The ~~partitioning~~ corpus partitioning problem of 125.00 still needs a lot of work, but I think I'm on a very promising path. 126.00 – 126.06 (133.01 – .23) Rev

.12 4) 127.27 This Section Generalized backtracking & its relation to the partitioning problem of 125.00 may be complete.

5) How Problems can be solved by (logical / mathml. reasoning): The conds can be any kind of pgms — pgms to do logical reasoning are certainly included. They would be more likely if TM had been given a TSQ in a relevant area.

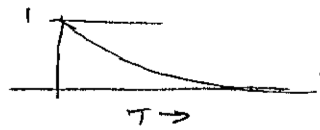
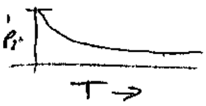
.19 • 6) The problem of Correlated conds! (134.00 start), 134.23 – .40 § 135.09, .26

• 7) TSQ Design

• 8) Augmented Z141 as a main induction system! It non-universality (126.09 – .40, 127.01 – .40, 128.40 128.26 (.25) in particular!
Here we regard Augmented Z141 as "just another Cond. for induction" We give TM many such conds, & give a (universal) stoch. grammar of which these conds are ass; Then we are able to universally extrapolate this set of conds.

4/18/01 IDSIA

.00 ; 133.40 ! On the eq. $\mathcal{P}_1 = -\int_0^\infty p_1'(T) \prod_{i=2}^{\infty} p_i(T)\, dT$!

p_i  or p_i' 

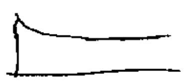
In either case $\prod_{i=2}^{\infty} p_i(T)$ starts at 1 for T=0 and very quickly drops to ≈ 0.

The effect is that about all cands will have only a very small $\mathcal{P}_i$

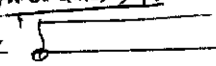
If we look at it from the Monte Carlo model perspective; each MC trial will have a cand w. max T, but while some cands may have higher P_i than most, most of the wt. of the $\mathcal{P}_i$ will be in the "∞" cands that are not very promising.

$\mathcal{P}_j = -\int_0^\infty \frac{p_j'(T)}{p_j(T)} \prod_{i=1}^{\infty} p_i(T)\, dT$. Also the S for all cands, has the factor $\prod_{i=1}^{\infty} p_i(T)$ — which

Zooms down to zero very fast.  for small T.

Another possy is that $\prod_{i=m}^{\infty} p_i(T)$ looks like  for some "not so large" value of m.

.18 i.e. for $i \geq m$ the comps/cands aren't much good — they tend not to solve the problem at all. — So we mainly have to deal w. $\prod_{i=2}^{m-1} p_i(T)$, which does not drop down so rapidly

For almost all cands p_i looks like  i.e. no possy of solving problem. So we have a very limb set w. $P_i < 1$ for much of the range of T (!)

.25 Another possl. way to deal with this — which seems fairly realistic!

The cands are correlated. That for i = m to ∞ ← hyly correlated

.27 For the present problem, for i < m, there are about 10 (or less) cands that are really indep — so we need only include them in the $\prod_{i=2}^{m}$ product. Then If we could do this, the implementation of $\int p_1'(T) \prod p_i(T)\, dT$ would be much speeded up.

But finding the indep cands is a new, difficult job.

However, finding them would overcome a serious criticism of this model — i.e. that the cands are not indep — as assumed in the model.

.36

Spec 135.26 135.09

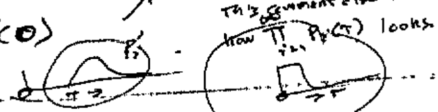
→ 135.00

4/18/01 IDSIA

.00 : 134.40 : Another approach would be to simply consider only the top 20 or 30 cands.

Another idea: Maybe better, non-cl. — Compute GPD directly from same data I was using for GPD₁. The Goal for this GPD could be its ability to successfully "postdict" the (Gmax's / Tmin's) of the corpus of quals.

In the original eq. the S may have been essentially determined by $\frac{d}{dt}\rho_1'(T)$ at $T=0$ ≡ the initial slope of $\rho_1'(T) = \rho_1''(0)$ — perhaps this is OK!?!? [not exactly $\rho_1''(T)=\rho_1''(T)=0$: usually ...] — so probably not so simple! — This comment also affects how $\prod_{i\geq 2} \rho_i(T)$ looks.



.09 : 134.40 Another Q: if they are correlated: does this make any difference? Yes, it does: If a bunch of say 10 cands are essentially the same, we would do much better by spending T/10 on 1 cand rather than T on each one!

It would be nice if we could devise "equiv classes" of n cands, & select a "representative" from each class, & work on the reps. only.

The eq $\int \rho_i'(T)\prod_{i\geq 2}\rho_i(T)\,dT$ is not at fault: it gives same results as MC calc.

I think the real problem is correlations. How can GPD detect and express the correlns. betw. cands in a way that is useable? → .26

(SN) put the eq. in better form. If $\rho_i(T)$ is and the prob. and $Q_i(T) = \int_T^\infty \rho_i(\tau)\,d\tau$ is the prob that cand will take longer than T to solve the problem

(.26) 134.40 [redacted] A ≟ non-el. "Soln" to the correln. problem: As is, GPD outputs a set of cands in response to a problem ("inquiry"). It could also (or "instead") output a set of "much smaller" less correlated [— Going just how this could be non-el.] cands (w. same pc's — or modified pc's: The pc's are automatically renormalized in most versions of L.sch, but some cands will "represent" more cands than others will "represent") and the "Reward" (Goal) for this kind of behavior on GPD's part, will be higher Pj for the cands that actually "wins".

How to actually find the correln. classes is unclear.

After "representatives" for each class are chosen & worked on, after a while, no success & one or more other cands may be chosen from each class & worked on.

4/20/01

IDSIA

H.W. note on IBM "Blue Gene" (10^{15} ops/sec.) Spectrum July 2000 p44 —

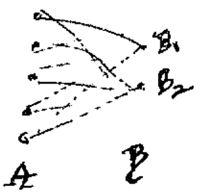
The Correlation (Delta Cands) Problem.

.00 | 135.40 | On a set of problems of ~~apparently~~ N characteristics!
Thirst probability problem: 132.09

.01 1) Dividing up a corpus into [redacted] (statistically indep.) chunks: Backtracking 132.12

.02 2) Dividing up (a set of cands for a particular problem) into a set of somewhat indep subsets. 132.19

Juergen has a ~ situation, in Neural Nets, in which he wants B_1 & B_2 to ~~independently~~ express "independent" aspects of ~~the~~ what's happening in layer A.



In the case of Indp cands, we want GPD to select (or create) a sub set of cands that represent the sub set, a clear as well/better than as any element of the sub set.

[redacted] GDRC is ~~by~~ may — post-problem PCi for the cand. that succeeded.

In one sense, this is what GPD is already doing: it is trying to give hy P's to the cand that is most likely to succeed (BESTG / Back Gr / ~~fastest~~ / Fastest Soln). — Hvr, the way GPD is ~~obtained from~~ GPD, in a way that prevents this "clumping". Ideally, it should be possible for the GPD to do this clumping "on its own" since it is rewarded for ~~...~~ "clumping" — But it is handicapped by its dependence on ~~E~~ GPD, as an intermediary processor of info.

Perhaps re define goal of GPD to produce a set of cands w. PCi's & Processing times w. these wts for time T, is most likely to produce soln.

or ~~a~~ has max expected "G value (expected G means (inversion of Gov!)

Another trick — "Get most likely cand to win" was then try it out: If it fails, we ~~...~~ change GPD & ~~...~~ ask for new "Most likely".

Or we ask GPD for list in advance, of cands in order to try, ~~&~~ for time T each. ~~(for individual problems), → expected~~

Well, say it's Inv problem; we try Cand$_1$ for Time T. It fails this for we now try Cand$_2$, which is now most likely, we work quit for time T ~~...~~ ~~At least~~ ~~Proper Knowl,~~ No success. At this time we info on failure of ~~#~~ cand$_1$ & cand$_2$. for time T each, we have a new GPD — It may say "Go to Cand$_1$ again" or Go to ~~...~~ Cand$_8$.

IDSIA

WON

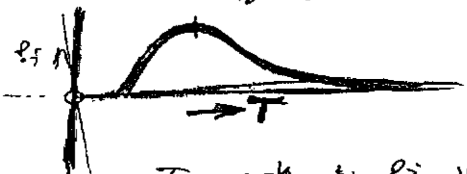
Copydoc
Won foldr

+ correlated Conds problem.

.00 : 136.40 Another approach (for Uncorrelated Conds) is t. **WON** approach!

It would be nice if I could expand t. treatments to deal w/ correlated Conds.

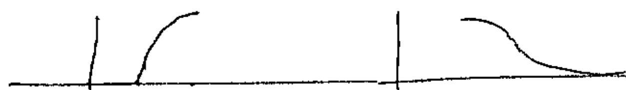
Given t. $P_i(\tau)$ functions of t. conds: a very Greedy approach would work on t. Cond of largest $\cdot P_i$ (at that time).



Unfortly, t. non-monotonicity of P makes this impossbl.

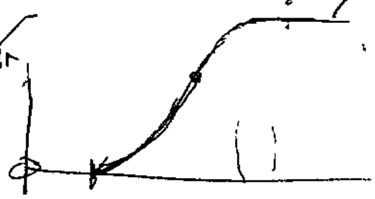
To work, t. P_i would all have to be ↓ (non-increasing) functs of T. (for that method?)

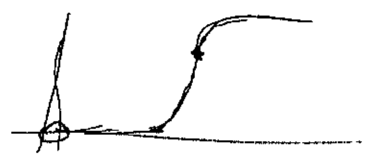
· perhaps divide up T (for each cond)



2 monotonic regions: One & other ↓. (↓ is easy to do — but ↑??)

.12 S(t)/dt





This is a well defined problem, but I may not be just what I want to solve!

If we have time T available, spend τ on $Cond_1$, $T-\tau$ on $Cond_2$; what is probly of success at t. end of time T? Use $\int_\tau^\infty$ & $\int_{T-\tau}^\infty$ an multiply; This is probly of both failing. 1 − (product) = probty of success.

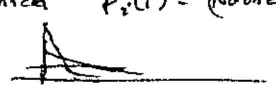
$\frac{d}{d\tau}$ of product: = $P_1(\tau)\cdot\int_{T-\tau}^\infty P_2(x)\,dx - P_2(T-\tau)\cdot\int_\tau^\infty P_1(x)\,dx = 0$

Maybe $\frac{P_1(\tau)}{P_2(T-\tau)} = \frac{\int_\tau^\infty P_1(x)\,dx}{\int_{T-\tau}^\infty P_2(x)\,dx}$ & $\frac{\int_\tau^\infty P_1(x)\,dx}{P_1(\tau)} = \frac{\int_{T-\tau}^\infty P_2(x)\,dx}{P_2(T-\tau)}$

.22 This would be a neat result if true!

(SN) Find t. special case in which $P_i(T) = (\text{Normal})\, e^{-\frac{T}{P_i}}$, Recom I think

Lsrch was optimum



$\frac{P_i}{T}$ $T = n P_i$

After, looking at it more carefully, it Looks like Lsrch could never be t. optimum Strategy!
— But look at my analysis of Lsrch wrt WON — I did do something like it — ~1980

$\int_\tau^\infty P_1(x)dx \cdot \int_{T-\tau}^\infty P_2(x)dx = \min$.

no! $(1-\int_0^\tau P_1)(1-\int_0^{T-\tau} P_2) = \min$ ⇐ Because $\int_0^\infty P_i(x)dx$ does have to = 1.

.36 $1 - \int_0^{T-\tau} P_2 - \int_0^\tau P_1 + \int_0^\tau P_1 \cdot \int_0^{T-\tau} P_2 = \min$.

$\frac{d \cdot 3c}{d\tau} = +P_2 - P_1 + P_1 \int_0^{T-\tau} P_2 - P_2 \int_0^\tau P_1 = 0$

.39 $-P_1(1-\int_0^{T-\tau} P_2) + P_2(1-\int_0^\tau P_1) = 0$

so $\frac{1-\int_0^\tau P_1}{P_1(\tau)} = \frac{1-\int_0^{T-\tau} P_2}{P_2(T-\tau)}$ → 138.00

4/21/01

IDSIA

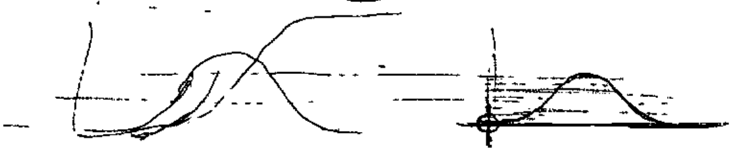
copy & to Won folder

WON

.00! 137.70: (137.39R) is a nice result, it means that $f_i(T) \equiv f_i(T) = \frac{1 - \int_0^T p_i(x)dx}{p_i(T)}$ = constant for all cands worked on.

Numerator is a ↓ funct of T, but denom is not monotonic.

Take reciprocal of .00 ! $p_i(T)$ mult by a monotonic function.

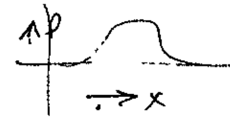


so plot $\frac{p_i(T)}{1 - \int_0^T p_i(x)dx}$

I think the result will look like  about across.

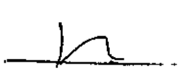
A monotonic ↑ ... to a peak from monotonic ↓ to ϕ.

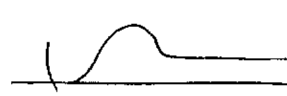
No! only if $\int_0^\infty p_i(x)dT < 1$. If it = 1, the ratio may not → 0.

we want $\frac{p_i(T)}{\int_T^\infty p_i(T)}$ look at $\int_{-\infty}^T p(x)$ vs $p(x)$ 

~~look at this~~

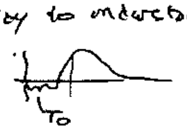
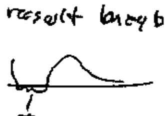
.17 Anyway, If we use eqn. .100, for a cond and all $\int_0^\infty p_i < 1$,
BAD! then we start out by working on all conds until their $f_i(T)$ = some small constant.
This would seem to be Bad because most conds have a ϕ
.20 "startup time", & one would have to do this for all conds (many of which have very poor!) ←

((Whoops Mult  by monotonic ↑ funct can result in a funct that has > 1 Bump! (How, it may be possible to show this does not occur in present case)
.26 Also Note rework of ((137.12-.22) ...))

Even if $f_i(T)$ of .00 looks like 

[The resultant strategy is the same as has objection .17-.20 ...
A possible answer to Objection. $f_i(T)$ of .00 was obtained by looking at derivatives of functs
But this is ok, since I've continuous & have nice derivatives !

I will have to look at this result — it seems Nutty! — Non-intuitive!
— How it does look like a simple result should be available! But note .26

.37 A way to understand how the result may be Bad!
Consider P_1  P_2  If one has $T_0 < T < 2T_0$
available, the best strategy is not to do both P_1 & P_2, but to work on one or the other.
Only if $> 2T_0$ is available, should one consider working on both.

4/21/01 IDSIA

WON What to Work On Next

Copyd to WON folder

.00: 138.40: T-analysis of 138.37-.40 is fruitful way to look at it.

If P_1 looks like  then $\frac{1}{1-.5P_1}$ is 1 until $T_0=0$, then it is $1+2k(T-T_0)^2$

This approach to WON may be v.g., but drop it for awhile. I want go work on "Report".

See WON folder for more recent stuff