MIN BACKTRACK

.00: 153,40 ! Another view would be to consider all 1024 10 bit codes: which of these codes to mask out of the corpus? (2) Take to 100 best. Perform first bit of each: consider all 1024, 10 bit expressions — loop to (2), Perhaps use Tz^{-d} as cutoff time where d = level in search tree of the 10 bits into machine (as trial code). Also use more structure OF 1024 codes to reduce repetition of some trial parts.

.05 5/10/01 A serious aspect/difficulty of the "Min Backtrack" system: Say we have stored "Best 10 codes" What we really want to store is the state of the computer system at the end of each of the codes — So each could readily do the production of its continuation. That would work fine! — But to do "Backtracking", we need to save the states of the system at the "Backtrack pts" It is not nearly to save all branch pts. of each code. Its only to save Branch pts that seem critical, i.e. every time TM has spent an additional ↑ seconds on a part, a machine state may be saved — or we have the changes since the last save state. Every K-th storage pt, we store the complete state of the system. Presumably, reconstructing a state from the changes from previously stored states, is much faster that actually running the program. — But this will have to be looked into.

.16

.17  We can store a code as "Lisp pgms are stored. Each node is a computer word. It contains address of Node it came from, & addresses of nodes it goes to — So we can traverse it. Not any way.

For each downward address, we tell which path was taken, also whether the other path: (1) gave wrong bit for code or (2) Timed out before giving bit (gone to bottom) (2) was not tried yet.

Min Backtracking from a particular node implies that that node was wrong or timed out for continuation of corpus. In other cases, backtracking involves moving "up" on the graph until a type (3) branch has been found, and been trying to code the corpus from that branch. With certain "Time-out conditions", we may also decide to spend more time on nodes that have "Timed out".

.27 Actually, all of the "10" best codes can be stored in a single data structure like .17 — .29 We also have to integrate the "state storage" system of .05 — .16 into .17 — .29. I guess what we do — certain nodes have pointers to places where entire state of system is stored, or differences between some more distant, previously stored state is stored.

~~There will be a trade off~~ As one of the no. of code pts. at which the machine state (or difference state) is stored, we will ↑ speed of backtracking, but ↑ other costs of system (memory ~~space~~). So we have to minimize ~ "sum" of time & memory

The time limits will partly vary w. depth in tree: Time limit = $T \cdot 2^{-d}$

d is distance from top of tree.

178.00

.00: 154.40: **Discussn** [redacted] W. Marcus: I started by describing the QA problem: that it was possible to put practically *any* problem into that form.

I desc. the QA prob as about (maybe 100) ≈ 152.00–.01

.04 That there was a set of operators, O_i, that mapped Q's into A's.

(That there was an assoc. stoch Grammar (that I wrote) that extrapolated a large set of initially installed operators into a "Universal set" (All possible operators) —

.07 [redacted] — if necy, I wrote how to add some ops to make it "Universal".

How, it was difficult to describe "updating" on stoch ops, — since I really haven't adapted Z141 to them (?), — Anyway, I switched to Sequential prob. in Z141, in which updating consisted of ① Recomputing PC's ② Defining new Symbols ③ repairing.

So I had this set of maybe 100 (Pams / induction methods), & the Grammar of .04–.07

That the GPD looked at a Q / a time available, T, (& perhaps, recent relevant $[Q_i, A_i]$) & gave a set of pams to work the problem, w. associated P_i. Several of the pams were tried on the problem — partly to get a better PC distribution, but perhaps mainly to "calibrate" = get more info on how these pams worked on this kind of problem.

.08 After working on 1 or more new problems, for which we have solns, I do "Updating": The thing that is updated is mainly GPD: It assigns PC's to

[(Note, I need not ask TM any Q's during training — the only input thing is what PC TM gave for what turned out to be the "correct answer".)]

the various pams, but bases this looking at the Q's. (The pams that yield p's are supposed to give best induction. From the recent degree of success / failure of the pams, the GPD is Modified! This is an induction problem. —

What kind of induction problem is it? Well we have this set of Q's — the PC's given by various of the pams to what turned out to be solns. — So

data output → $[Q_i, [P_{ij}]]$ P_{ij} is pc of A_i given by Cand$_j$ w. input Q_i

$\{Q_i, j(i)\}$ cand$_j$ gave best response to Q_i. ← This is data for induction of

.33 So we want GPD to give $J(i)$ as response to Q_i. ←

or, Since GPD is a stoch operator, we want it to give highest possl. pc to Cand$j(i)$ when Q_i is input to it.

If GPD has OSL, then, hopefully it will apply to Q_i in Cand $j(i)$)

.36 [GPD's problem looks like a QA problem. How does it differ from the original $[Q_i, A_i]$ problem? A_i is replaced by $j(i)$. Is the $Q_i\, J(i)$ problem an easier problem than the $[Q_i, A_i]$ problem?]

157.00

.00: 156.40: So 156.36 -.40 does look a bit like GPD has to solve the original QA problem directly! But actually, not so. GPD's problem is MUCH easier! There are only perhaps 100 regions GPD can drive, but each $j(\xi)$ has to go from Q_ξ to a much larger set of responses — ∴ much harder job. → (.30)

Also this 2 [redacted] step process seems to be what Humans do! First decide what kind of problem solving methods should be tried — after a general perusal of the problem — then application of the chosen methods to the problem.

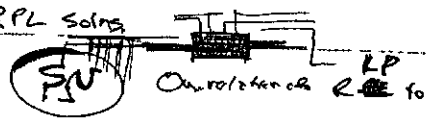
.10 How to update for QA interprocess. TM learns to make various obs on Q_ξ and on Pems: Then empirically learns to correlate them. T. Perry is a very superficial way to do it — it may often work fairly well — but it doesn't show much "Understanding".

Int. diam of updating of 156.18 ff, one can introduce the T into: use how good each Pem did in the known T_ξ.

N.B. Another simple kind of "updating" of GPD is the creation of new cubes/Pems of any utility (i.e. very useful for at least one Q_ξ (or type of Q_ξ).).

N.B. into Q_ξ info, T_ξ must also be included: So GPD knows Q_ξ, T_ξ before it has to give the cond'l distrib. (see ((10-.13) R)↗)

Note that QA problems often have an associated "DATA Pool" to which [redacted] available to all Q_ξ's. The ALP ($\epsilon=\infty$) soln to this problem probably suggests RPL solns



Our relation of RLP to ALP: Putting ALP in form of "All Pems Method", may be Good idea: Some Pems are much better than others: Also $P_E \leq 2^{-k_2}$ form of [redacted] ALP is usually not a good suggestion for how to do RLP.

↑ "All Pems methods" has to work if the [redacted] seq. of interest is very generated by one of the Pems, — of course we assume the coil of the "right" Pem is "reasonably large" ☺.

.30 (.0↗) While at first glance the replacement of $\{Q_\xi A_\xi\}$ prob. by $\{Q_\xi j(\xi)\}$ problem, would seem like not a "Great Break Thru" — it is much in the spirit of My conception of GA work — in which The User has to look at the problem & (eventually) how to represent it, what mutations/crossovers to use, etc. That is "Great Breakthru" would be a method of doing this automatically.

.36 → Another Boost to GA would be the Generation of an initial population based as the result of initial examination of the problem done by the System.

→ I'm not sure my own system does anything analogous to .36.

Well, any work done by the system that makes the problem easier to solve should be rewarded as being in this class.

158.00

.00: 157.40 : I'd want t. updater to create new (cands / stoch / eps) pems for specific problems. [i.e. as a function of Q_i : There is a default d.f. of t. cands (indep of Q .: an unconditional P.D.)

.02 Lets Go thru t. operation of t. system again! We give t. system an induction problem (including T). GPD looks at this problem: Outputs a set of [$cand_i$, P_i]. A cand is any pgm (string) that mite solve t. problem, P_i is GPD's estimate of proby that $cand_i$ will be best for That problem. The output of a cand, is an induction on t. data. It is equivt. to a set of codes for t. data, but more often, consists of a "2 part code" for t. data (or several such codes) (perhaps + an OSL part).

Many ~~...~~ induction problems involve continuing a partial corpus, so perhaps OSL is automatic (?).

Anyway, t. output of t. [redacted] cands is "checkable" : One diffy is that diffnt cands will given use diffnt amounts of t. corpus for their codes. — Diffic to compare.

T. final output of each cand is a pd over possl continus of t. corpus. → Also see summary of ~~159.05–.20~~

To simplify (so we can continue) assume corpus used by all cands are all [redacted] t. same.

Then we can compare cands for each particular problem, wrt t. amt. of compression it gets (in t. available time, T).

So we have t. Guessed output of GPD as to which is t. best cand to use in (problem Q_i) using cc = T.

— And t. apriori assignment. So it is GPD's job to predict which cand is best;

We have data on which cands for which problems were, indeed, "best", so constructing t. updated GPD is a induction problem.

We can update in 2 ways! (1) [redacted] ignore previous GPD & do induction directly on data (2) Use known previous GPD as an apprxn to be modified by t. new data How this last shall be done, depends each induction scheme used. Method (2) is always better if we have t. available CC.

A ~~...~~ third way is to ~~...~~ generate new cands that are likely to be of hy utility, This is done by a Grammar, — but ideally, we should have a grammar for each Q — Giving a pd over all possl cands.

.36 Or, one has the set of cands: Using a finite subset of tested cands for each problem (Q) we get an empirical set of wts. This [cand, wt] info then can be used to induce a Stoch Grammar, which can be used to suggest new ~~...~~ promising cands for t. assoc. Q. → (159.03 spec)

159.00

4/30/01

IDSIA

GE 4230P 49.98 Grm. D 2272 | LO:11

.00 | 158.40 | T. mutter of a new Grammar for each Q isn't too bad; but a grammar for the next Q maybe didn't — but it's closer to what I want!

(previous / Post / past)

T. Peggy is beginning to sound like SOAR!

.03 : 40 . In .36 - .40; There is some confusion: T. (wishes) can be "probability that candidate is best."

or simply the compression (pc ratio) induced by the cond. — a "G" value, which is also a pc.

.05 — on 158.03 again: problems: Q_i; answer A_j. We want a good mapper from Q_i to A_j on past data.

One way: find best F [redacted] $F(Q_i) = A_j$. But F is stochastic operator

so we want [redacted] $F(Q_i) \rightarrow [A_j, P_j]$ Σ^2 P.D. on $\{A_j\}$.

We want [redacted] [redacted]

.09 $j(i) =$ [redacted] $\Rightarrow Q_i \rightarrow A_j$ We want $F \rightarrow \prod$ [redacted] $P_{j,i}$ is max.

$P_{j,i} =$ P.C. assoc. w. correct soln. of Q_i

One way to do this is to have many [redacted] F_k's; have GPS try to guess which is best for a particular problem — or give a set of arb. part. F_k's

GPS's Universal Mind formulation

Updating is in 2 forms: (1) [redacted] assignment of F_k wts. in view of new data.

(2) creation of good new [redacted] F_k's and improvement of old ones (in recursive order!)

.20 .05 - .20 is a [redacted] step process. See diagram of it's 2 step process! 15:7.30

~~Small scale problem: Det. $[Q_i, A_j]$ pairs directly. Find as good a $F(Q_i) \rightarrow [A_j, P_j]$~~

Also, within F's: changes of pc's of symbols; defining new symbols, replacing of all symbols

~~deepest~~

[redacted]

Q_i in .09 we may want to include/[redacted] coding of $[Q_i]$.

.26 ~~To what~~ How may Updating problem be formulated as an induction problem?

.27 . In a simple statement of the problem it seems like the updating problem $\Rightarrow$ an induction problem, but

.28 it is the main problem we are trying to solve: to find good F for $F(Q_i) = A_i$.

.29 Here: It seems that the way F is updated can be improved as TM gets wiser.

149.36

148.25 ← discussed much

[When I first began writing about the TM that did [redacted] induction only (say QA only), I ran into the difficulty of .27, but I quickly drifted into a soln. of .29, & was unable to replicate use of .26!

In .28: A good meta problem would be to speed up the improvement of F.

.37 So Induction says: try to find as many good progs/codes in available time.

Updating says: "try to improve .37" but this sounds very much like .37 itself!

16000

IDSIA

(Rev)

.00 : How does my present System, & understanding of problems differ from state in [Sol (89)] (Israel paper)?

1) I felt t. simple Z141 model was ok. [redacted] (Did I write Manor & forth paper was finished? — I think so, but not sure)
I didn't know about "repacking" — whether this idea will be adequate to deal w. ~~complex~~ "Augmented Z141" w. macro diffic. corps — is uncertain.

2) I was using T&2T Lsrch (no really big difference, hvr, only factor of 2 off) [See 192.18, 23 $\frac{T_{max}}{T_{ll}} = (n-1)\frac{2^{n+1}}{2^{n+1}-1}$] ↳ Could be factor of n, where soln is of length n.

.08 ??→3) I did not ~~do~~ consider effects of up dating Pi's during Lsrch. (hvr, see .28)

• 4) I did not worry about correlations betw. t. Pi of cc subs: (this mite be fixed by Irng. during Lsrch.)

.10 • 5) I felt that Induction was an ordinary OZ problem! ~~[illegible]~~

I had not yet realized that one normally did not try to code t. entire corpus if one's CB was < ∞! (A later approach to this problem was "partitioning the corpus": I'm not sure the relevant to this Q: See 585.21 seems to be a v.g. fairly General Soln.!)

• 6) Related to (3) I didn't really understand that all heurs can't be simulated by "Blind srch" w. enuf inputs. I'm not entirely sure of just why I felt Blind Lsrch could be as good as any heuristic srch. — My present impression is that it is not: that one needs (at least) updating during Lsrch, which makes it Non-Blind, so that it ~~can~~ ~~does~~ mite do as well as Non-Blind heuristic srch. (Most Heur srch is Non-Blind). → See 143.00 for some discn — did I mean "concept"?

I had t. idea that in Lsrch the pc of an "operation" could be a function of [redacted] t. traces of previous trials. While I think this may have been wrong, — rite now I'm not sure that it is nearly wrong — I may be able to see how it could have been rite. → Also (FN#2) of solsy — did I mean FN#4?

.28 → Int. 4th sentence of §4 of Sol(89) (on Lsrch): I said that at later stages of training, inputs to problems could include the "il" traces info from earlier trials. (of this a perhaps clear problems). This is a way of Doing non-blind srch — but I think it's quite "GREEDY". (But this is not, But this is not GREEDY "speeding" "Lsrch" ↑ (ans!))

Hvr, t. method I'm proposing now — (Updating Variation of GPD during Lsrch may also be "GREEDY" by some Amount. → 142.10

.34

.33 • 7) I don't think I had a clear idea as to what t. Pi was, that was guiding t. Lsrch, (i.e., goal cands being tested soln)
As a result, I probably wouldn't have been able to set t. system to Run as described,
But on second thot! I did have a definite model. I had t. set of (problem, best soln) pairs, and I just got a P(· new problem, cands) distribution. } This is a perhaps workable, "Ruff and Dirty" Soln.
The reason I think I dropped this was MCT: It suggested free new, much more complex form, because t. soln. to OZ probs in MCT, involved that approach. Also, t. new approach can use More data in the induction process.

4/21/01 IDSIA **ReV**

or second? or second?
The first "Cure Cancer Prob"
Soln: was ID 19.26-40

(4)

100: 140.90 : In addition to t. "Best" solns, we also use info on "failures".

[Now I understand t. problem better (maybe) & I may be able to use t. older, simple Soln sometimes, if available cc is small.

(8) I didn't have a understanding of Soln. to t. mixed corpus Problem.

9) I didn't understand t. convergence Pm for BAGS : I didn't have a rite Gore for it.

10) On t. Updating problem : In Sol89, I considered Z141 type grammar for updating. I was aware that it could be used for general functions, but I really hadn't worked that much. I considered more general Stoch Grammars, & several data compress algms. — This is close to t. most General soln, but it does run into t. problem of (5)(140.10) — which I was unaware of at t. time.

But see (7)(140.33) for disc. related to UPDATING.

11) I don't think (but I'm not sure) that Sol89 considered that t. Cands could be ~ "Anything": So they might be srch pgms — similar or quite different from Lsrch — So eventually t. effective Srch routines to be as good as any conceivable srch routine → Not Nearly "Lsrch"

-19 12) Sol.89 had t. Scaling diffy : Unclear as to what exact I have now Solved it!

It would seem that GPD being a cond'l pd. would give a special PD for each problem, so that relevant cones would be given by pc's. In general t. pc of components (abstractions/functions) can/should be a function of t. application/use of t. (abs/funct). I think this is an essential idea in dealing w. an impt. aspect of scaling. I.e. t. pc. of an abs should not ↓ much as t. size of t. corpus ↑. Each abs. should be tied to an "area of knowledge" & t. total proby of all abs assoc w. that "area of knowledge" (≡ Domain) should approach about a constant as t. corpus size → ∞. Or maybe / each area of knowl → log N as N (≡ corpus size) → ∞.

.30 13) (4/16/02) I hadn't yet considered "Recognition Algms" — So TM could recognize that a given problem was of a particular kind & should be grouped w. probs of that "kind" in attempting to find a soln. E.g. Math, v.s. Physics v.s. Chemistry v.s. Sol, ect : Various kinds of Math problems : linear Algebra ...

4/21/01

IDS/A

142

73 3/23
142 4/21
71pp
29d
71/29 = 2.448 pp/d.
4/1 : 89
4/21 142
53pp
20d. = 2.65 pp/d.
30 M
22
29 = Sun
30 = Mon
1 = Tues.

.00: ! From Review of 140-142 !

1) on the "Easier Update" idea in Sol 89! (140.33 ff (7))!

for update, easier as data, set of {problems, Cands} Cands being soln. that was found for prob.
Cand is the algm/string/proc. that was finally found by Lsearch for problem.
This update method does not use info on Cands that "failed" or by how much they failed.
So it can't take advantage of "failure" info.
This update scheme is best-suited when one is making history! It may not be
adequate for "updating during Lsearch". (Also, it doesn't work for OZ probs)

.10 2) On 140.28! I had the idea of the problem including data (traces) on previous trials for
present problem and perhaps all other problems. This could, perhaps,
enable the system to emulate being (Blind) blind deaf. It is up to the cand to decide
~~how much info~~ just what info it wants to use:
But this would make the appropriateness of Lsearch questionable! The final result
.15 be dependent on previous trials. If would, then, be a Greedy method. (143.01)

3) Perhaps develop Minimal System; then discuss potential deficiencies; ways
to overcome them. This would be easier to develop/understand.

→ (4) On Updating during Lsearch: A Cand could have the capability of "Calling" the update algm.
The inputs (params) to the update algm: Time to be expended, corpus to process, (goal of update) —
This last is unclear, but it seems necessary — one (many) general way to specify it would be
to give the problem now being solved. If a cand's have done much updating during Soln.
of a problem, then when the problem is solved, the updating (if any) can be more focused —
less localized to immediate problem area. [4/16/02 — comments: I'm using a "Continual Loop"; What would an "update call"
mean? In Normal Loop, it would be a Wallis-function w. "Update" as a "Side-effect"

(5) Perhaps it would be good to discuss "operator induction". (≡ Q.A induction)
Doing a updating the GPD is an operator induction problem. To show a problem is
an operator induction problem, one needs to show the input set, the output set & the
data set. (A "data set" is not involving but a relation on case counts.
If A is input set & B the output set. Let A × B be the set that is the cart product
of A & B. Any subset of this cart product is a relation. A data set is the Cart product
w. a "case count" associated with each member of that set. Or just take cart product & give case count
for every element. Case count can be: $c \geq 0$. As is case count is a positive
integer. I have no ideas on case counts being neg, real, complex, etc. finite, neg Groups, etc.

IDSIA

NB FN #4 v.s. FN #2: Depends on what Mauus cripts referenced.
Its FN #2 in HTML web page. ← This version is missing 2 few FN's! They were included as "parenthetical Remarks"
Its FN #4 on Hard Copy versions.

.001 142.15 : on t. Fn #2 of Sol(89: 142.22–.24; 142.10–.15
Think of "Various such items": How could they be expressed as "Blind srch vs. "powerful cands"?
I don't know just what I meant by "Blind srch"!

I guess t. "Blindness" of t. srch, is that t. cands, are chosen in some order in advance; a t. order may or may not be further managed by considerations of t cc (so far) spent on each cand. FN #4

Hvr. in FN #2, I allowed cands to use any info found in t srch by previous cands — i.e. traces of trials. What I meant by "Blind Srch" in that FN, is unclear!

[A more relevant Q, of course, is to what extent can I emulate heursrch by suitably "Augmented" Lsrch.]

12.03

.15 Anyway, if "Cands" can look at any previous info in t system, then they can Emulate any (Search Heors) (when can't do more than that?) ← (?) "when can't do more than that! Did Lsrch ... ? .15 could be better than some/all heurs?"

If this (.15) is true, then T. args for Lsrch's optimality are not correct.

If will be best to do a certain "Cand", that generates a set of useful data; but no poss of prob. soln. — then have a bunch of different cands that use this data — each in its own way. The one that might be particularly good if trials are very expensive (so we want to share t. info betw. cands.)

→ What I really probably want, is an Emulation of all poss. Heurs. (.15)

— Th. goodness of a heur/cand, will depend on what trials have been made thus far "NN"

[FN] for convergence of $U^{\text{conditional}}$ in Sol 78 TS: $\sum (X_i - S_i)^2 < k$ $k < k'$
$\sum (X_i - S_i b_i)^2 < k'$; $b_i \to 1$ $b_i > 0.$ $b_i < 1.$

→ So "trials thus far for H" Must be an input to GPD. If I use a very Greedy method, I can start by picking a cand that makes one or more trials. If it fails (via WON-like idea) after a while one tries another cand, t. nature of which depends on cands executed thus far — i.e. traces of those cands.

.35 My picture of how I might solve a problem! I look at problem. From various associations, w/problem dom I get some ideas on how to solve it. I try one or more of them. During the trials, I may generate info that strongly suggests "certain actions" — (these actions could be actions directly oriented toward solving problem or maybe just getting info" to steer future trials.
(α Do action; on basis of results/traces, → loop to (α) until problem is solved.

4/24/05 7:30pm

— see 143.35 for explanation of this "α loop"

.00 (see 143.40): [α] Action_i, obs_i → ~~hoped~~ action_{i+1}, obs_{i+1} → loop to α.

This looks like ob/op algebra!

T. args of action_i are problem_i, all previous traces.

T. ~~[illegible]~~ formalism is t. result of a bit of analysis of 50189, 6W2

.60 Seems to cover All heuristics. (Note 143.35 –.90)

NB, hvr. To go from obs_i → action_{i+1} involves GPD: This function could take some time — maybe a lot! GPD (problem, obs_i) → op_i (= Action_i)

A macro (ob/op) can include ops/obs

.11 In general, t GPD will have "Best op_i", but we may want to do some probabilistic dispersion for "creativity" ← See 104.00

Another poss, is to use "Best op_i" (most likely) each time, but when we get to local max, start Lsrch over rest of cands. — in .11 hvr, presumably, we don't get to "local maximum".

T. above approach is Max GREEDY. "Lookahead" of 1 or more cands could be useful, but we really make no irreversible actions ever, since all past info on ops & obs states is available at all times — (tho TM can't look at all that info — takes too long so we could effectively use "Look Ahead".) — So it would seem that lookahead is unnecy.

On t. other hand, if TM gets to a local peak, it can look at its own history & realize its at a local peak, & get idea of "size" of peak & know how big to jump to get out of it.

Or it might logically analyse situation & figure out how to jump off local peak.

In t. .00 ff approach, t [illegible] updating need not be done very frequently. It can be time shared w. main problem solving w. not much extra cost — or just done periodically.

T. optimum ratio to use ("R") is not clear, hvr. — we may want to spend a very hy % of time "updating" GPD || Q: What is UPDATE Gen?

Soon Perhaps write paper on MCT: This audio clarify t. update process a lot. — Mixed Corpus Thrm.
Also include opns for induction.

4/25/01 **BIG REVIEW**: 140–145 145

IDSIA

01TM

.00: : T. most uncertain part in t. system at 144.00 is that I don't seem to be using probabilistic info! "T. cand most likely to be best of all cands" should be chosen each [redacted] time. (this is prob'istic info!) — ✓ Its t. Best way viz one of Goodenough House Thms.

It has t. same diffy of "correlated cands" as previous approach, but if we chose to execute t. "Best" cand each time, correlations might be less impt. Presumably t. system would develop ways to deal w. "local extrema".

As is, t. system is about t. same as previous/Augmented system, [cands can look at all traces & previous history of t. system.] The op/ob algebra should be made part of t. scheme for updating GPD — i.e. t. algebra produces cands of hy expected utility! it is a good Grammar.

It would seem that if we want to consider cands other than t. "Best" we will run into t. problem of correlations betw. Cands

Perhaps a BIG difrnce betw. 144.00ff & previous! [redacted]

in 144.00ff [redacted] each cand does not nec'ly try to solve t. problem (itself).

.18 Th system (GPD) that first looks at t. problem, effectively "plans" t. attempts to solve t. problem. 3.66 3.66 9.2

→ As a result, t. Gorc for cands is quite unclear. — We could have a Gorc for {t. entire set of cands assoc w. a problem} (along w. t. cands order in which they are to be invoked.) Find Gorc for T. system = cands. Will Gorc seem wrong

This system (.18) can be regarded as a "cand" — (the set of ob/ops that it executes) is t. cand; — T. main diffy here, is that w. this list of cands, one probably wouldn't want to do "Lsrch" Each cand would look at t. traces of traj that it has made. (& possibly at traces of previous cands) COPY w. pic

A cand of this sort can be evaluated (if it is t. first cand to work on t. problem (i.e. that it didn't look at traces of earlier cands) does w. DP We expect that our first cand will win. C.I. Ray

(.32) The reason for Lsrching over other cands is to get data on their "effectiveness" So GPD may learn.

In case of (.32) then we are still interested in "Correln." & in finding

.36 ∼ equivalent classes, and representatives in each class.

It may be best to use t. WON method in dealing w. cands of this sort. (& perhaps any other sort!) — T. main idea being that one sticks w. a cand until one could do better by switching.

4/25/01

IDSIA

.00: 145.40! An impt thing that the WON method does not do! 145.32 (Info for teaching GPD)

To what extent is it possl. to now build a TM using the Sol89; then gradually improve it? (or even start w. Sol 57 (an ind inf. Machine))

.06 Classic (≈ Sol 57) method of for TM: (α) Solve probs by (Srch random (Lsrch perhaps) — rate in order of complexity) using set of primitive funcs. Change pc's of primitives, define new funcs so as to give hy pc to solns. Give new problems (op pt (α)) (α) This was how Sanab tsq's worked.

.09 T. set of probs solvd could be OZ (& perhaps INV) probs. → (.36)

.10 SN An Adequate Problem Set is induction (Time limited), since QA is included, & improvement of GPD is (probably) included. It simplifies f. MCT soln, since OZ problems don't have to be dealt with specifically. (Sol57 did only induction)

.14 So: say we have an induction algm, that is able to take a corpus & prediction pt. problem (& symbol (to be predicted)) & give a proby for that symbol.

Or, say we have part of the corpus "coded" (or ≈ aquired) & we have an augmentn. of the corpus w. a new symbol to be predicted — & it gives a pc for new symbol.

We can compare various algms on the accuracy of their predns (McCarthy Measure).

— Also the time used for each algm! — So we could get Lcost of each Algm.

Each Algm is a {PEM + run time} So this is my old "PD + CC" problem.

It suggests Lcost [redacted] as a GORE. — But note that it/ take may different amts of cc for different problems; The Gore is always for a particular problem set.

So should we add the cc's & Mult the pc for each of the probs solvd?

[Well, say we have a new problem ~ to one solvd by the alg. — but w. very low (cc/pc) for that problem (w.r.t. many Lcost over the problem set) So perhaps use this Lcost as Gore for that problem?]

In comparing 2 induction Algms: If we can clearly tell that one Algm is better in certain Domains than the other — we should combine the 2 Algms to give one that works best in all Domains. T. problem of deciding when to switch between the 2 is an induction problem.

.36 (.09) Q: Just how does .06-.09 differ from Sol 89 (except for use of Lsrch & good understanding of probability? Sol57 only did induction (which is fine according to (.10)

.00! 146.40: On induction TSQ's for ~~the~~ Sol57:

First teach ~~...~~ definitions by example: IS (number, 3), [No / yes] teaches what "number" is.

IS (number, 3) ? ? = yes/no. TM gets pc. of y.e.s.n.

eval (3+4) (1 2 3 4 …) ← feet TM tries to find (a function / functions) that map problem into soln.

So, so far 2 kinds of probs: IS, eval, solve ~~...~~ solve(x, 3x+7) ← =0 is implied

---

1) work on Min backtracking Algm for coding / search w. binary coding.

2) The new idea of (Cands [redacted] having access to all traces up to now) — needs less frequent updating of GPD. — which simplifies Lsch. What disadvantages does it have? It does make the cands more like normal heur srch in A.I.

For the Sol57 system: I had this sort of problems & a function that was able to look at a problem & give solns. (w. hy pc). The function F was given by a "2 part code" (i.e. the OSL ~~modify~~). The first part was a set of functions & [redacted] assoc pc's.

The second part, a dern of the function F (corpus) in terms of the functs in part 1.

.19 This is "Operator induction" (problem in → soln out). After we solve several probs using function F, we give it new problems which F can't solve. Using the operators in parts & their pc's, we try to construct a soln to the new problems & a way to ~~...~~. The ~~...~~ are added to F to produce F_1, which works for the new problems & all problems up to now!

.23 We then try to compress the dern of F_1, by modifying pc's of sub functions and defining new sub functions ← w.o. backtracking. If a new function is defined, we ~~...~~ F_1. If we have time, we may try undefining some functs & defining new ones & reparse. ← Min Backtracking

Hvr, I didn't expect much backtracking. Most "lrng" consisted of building up (defining) more complex functions from simpler functions.

(5) → AH! In the Sarb[redacted] TSQ's: The decision to put all problem solns into memory

.34 was a way of doing OSL! — But note that this technique favors the SCALING problem! If there were too many of these "solns" in memory, their pc's would be very low & they might not be legit OSL codes! → 148.17

The other annoyance about SarbTSQ's was that solns to probs were often only depending that I made! — But this might have been because function F was too small to have

4/25/01 IDSIA.

2, 138
25 198
△ 2/6.

.00:197.40 Many (??) logic regularities (≡ common sub(Trees / Functions))

Within t last (12) 6 mo/or so, I was concerned about this problem of augmenting a function that worked w. a certain set of problems — then we had new problems for which t function no longer worked. (147.19–.23 (see 2))

Re t. method of 147.19–.23, it would be well to have an adjustable amt. of "Backtracking". We would first look at t. "local solns" assoc. w. each subset of probs for which "F" had to be "Augmented". We can find subsets of these that have many common elements, we can "backtrack" on those of they are "recent" — Recency means that t. new features tend to build on one another and that modifying t. defns in them will not screw up [redacted] codes for earlier functions.

So There are at least 4 similar systems for TM.

.16 (1) Sol 57, (2) sol 96–98 (3) Search TSQ (4) Recent modifns/augmentns of (2)

.17 Re: t. Scaling prob. of 147.34; One way out of it (to some extent) is that we want to modify recent Functions F, to become t. new F. Which means we have hy pc for recent F functions in forming new F functions.

How do t. systems of .16 differ

.25 If we have a "only reduction" TM, Then t updating Algm. ≡ same as t. mean problems!! See 159.27 ff for more details on this diffy

.26 (Would WON'T be Relevant?) Does taking "T" into account for all examples Help??

Look into .25: looks right.! We have, say, sequence to be extrapolated. We have a set of Algns (Conds) for which seq. to be extrapolated is input: pc on extrap is output.

→ Possibly, all previous extrapln. probs & (what occured) could be also part of input ≅

Update problem: To reassign wts to Conds; to develop new Conds w. usable wts,

I was thinking of $\Sigma \frac{T_i}{P_i}$ as "expected time assoc w. a cond": T_i is time needed to get P_i ≡ pc of correct soln.

Why not $\frac{\Sigma T_i}{\Pi P_i}$? Any way, we have these Conds a. wts that are regarded

.36 → as a. ass of a (stoch ??. lang.) — of which we want to Grammar } ← This is what "updating" does.

Note: It's not clear that t. predn. is t. wtd mean of predns of all of t. conds!

This is because of correlns betw. t conds.

I, the pc's of t. conds. can be t. wts/assoc. w. their predns.

4/25/01

IDSia

.00 : On 148.25 : It's now unclear as to just what I was thinking about! — See 149.36 Also 159.27 ff

But First derive in some detail other ways to analyse it — so I don't forget. → .15

.02 For seq. extrapolation of limited alphabet: Make defns of subseqs (~Z141)

When new data comes in: ① change freqs ② define new ultras

③ reparse (consider several reparsings w. different symbol freqs —

so when corpus is augmented & freqs change, perhaps different new symbols

will be defined; Also try some backtracking of some depth

(at least 2 or 3 or 4 definitions; reparse & define new symbols, reparse,

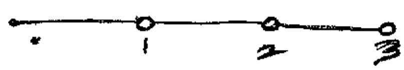
define new symbols, etc.

.02 ff can probably be done for Operator induction system! Again using

(as wanted Z141) Essentially, I am writing a BIG Function

.14 in Z141 "LISP" library. → 160.00

.15 α



$cand_j$ looks at entire corpus$_j$ & time T_j.

& outputs a p.d. for extension of corpus$_j$ in time T

updating: look at the set of ($cand_j$ corpus$_j$, T_j) triplets, & tries to improve them by modifying their parsing, defining new objects, and reparsing. ← looks a lot like .02–.14 ideas

---

.22 β. $Cand_j$ looks at corpus, & T_j : outputs p.d. for extension of corpus.

GPD looks at corpus, and T ; outputs

$[\ cand_j^T,\ p_j]$ ∋ $\sum p_j$ "should be wt. of $cand_j^T$ in predicting corpus"

$cand_j^T$ is a predn algm to operate for time $p_j T$ · (± (such?)

GPD operators by looking at entire past set of problems & deciding, in view of T, what wt. $cand_j$ should have. GPD must do this induction by reducing what wt. $cand_j$ should have based on it's known (Times & trials) of the past. Also how to UP Date GPD — changes its A generate new cands.

This is still not in the spirit of 148.25' — try "not using T"

.36 γ $Cand_j$ looks at corpus : output p.d. for extension ← is ≡ is ??

GPD looks at corpus; outputs $cand_j$ and (p_j which is wt. of p_j)

p_j is simply pr of previous corpus as given by $cand_j$

.38 So system γ (.36) is prob what I was thinking of in 148.25 because GPD output is essentially, to predn: Since cdf grows PEMS (≡ cands) & wts. for them.

.39 Since all PEMS are not considered, the update could generate new (cands / PEMS) & reassign wts. to old ones. One way to generate new cands is to express all old cands

● problem is to devise a good "Summarizing Machine" — but we must also consider time available so as to cut down corpus being coded. – Partitioning problem.

Marcus says that in a brief effort, I could get MAPLE to check my algebra

4/27/01 Idsia 11:40 PM

.00: 149.40 : as (singles) ass in a stoch. Lang. This is difnt. from 149.02, in which what's generated is a (single) stochastic source that generates the corpus. In 149.36 ff. we have many cands — each of which could generate the corpus.

In 149.02, if we considered difnt ways to parse the corpus, we'd have something closer to 149.36 ff.

In general, there are many difnt (kinds of) Pems. Each will have its own updating method. Operator induction & Bag induction will have special updating methods.

In the Bag induction we have a corpus of objects: we have a new partial object & Problem & PDs for its extension. If we have a "sequential corpus" the "objects" can be subsequences of the corpus. We may or may not want to regard this set of sub-sequences as a "BAG"

.14 Thus, it may be that the most General updating method for the set of Pems (is a predn problem.) i.e. Say we have a conditional P.d. — the cond is the string to be extended. In response to this problem, the GPD outputs a set of cands & (assoc.) pc's. This is essentially the solution to the problem: no Lsrch is needed. Updating consists of modifying wts in view of the success/failure of the predictions, — possibly generate new cands → (149.39 – 150.00)

(This is where predn. comes into the updating problem)

.14 ff is about same as 149.36 ff.

149.02 is only one type of PEM — but a very imp. type!

It may be that the cands/pems being used are (hyly) correlated — in which case — do we really want to use as final P.d., Raw wtd. sum? Perhaps the "correls" are a way to assign extra (or) (apport) wt. to the pems? What Are [illegible] Correlated cands & their wts are used as (data) ass of a stoch. Grammar — (the wts. used for ccc SSZ) (or revise those?) → will it give reasonable extension of (cands/wts)?

For predn, we may want the just use the "Best Pem". Other Pems are randomly tested to keep the Good lineage of cands [illegible] "Adequate (Variety) Diversity".

Perhaps list a bunch of Pems/cands & their updating methods; See 166.32 for a good idea on how to integrate these 2 methods into a "Grammar"

(.33) 1) The Pem of [illegible] (augmented) = 141 ; 149.02 & its detailed updating scheme; See (12)

2) "Regressn" predn. Several linear, non-linear coefs; Number & type of coef. increases w. SSZ. — (Thus we can use many sets of coefs in ||)

3) Neural nets (feed forward) : for n.l. predn. Use as few wts as possl. (feed forward, recurrent, Self Org Maps)

4) The update scheme that operates by recognizing new types of problems & devising Special Solns for them.

& Later, the problem identifiers & soln types are integrated into those used by previous corpus.

5) Using stoch grammar to generalize (successful) cands (see (12)) → See 151.03

4/27/01 IPSAL

151

See 166.32 for a new idea on how to make a "Grammar" for these 15 IA's (Induction Algms).

.00: 150.40 : (SN) In those systems ~~that are~~ in which cands are Evaluated by Them successfully predicting only part of the corpus, there is a ~~[illegible]~~ Tremendous loss of usable info!

Try to find way to Use lost info.

.03: 150.40 : 6) GA prodns of SGA & GP etc. Various Mutation/crossover methods

7) General Compression Schemes, Lempel-Ziv, & others.

8) Modeling Source of Data. — Info about source of data/problem)

9) General ALP. Coding w/ Backtracking. Direct trials of codes. Use T+ZT tree limits Use T·Pi time limit for each branch breadth during backtrack

10) OSL (and 2,3, t-shot lrng), case based lrng,

11) ID3 ("Decision Trees")

~~12) Grammars of various kinds (see) (CFG, CSG, stochastic, UM grammars, finite state grammars~~ Hidden Markov Models

13) Explan based lrng? Explanation Based lrng

~~(14) Predict int sequences by modeling data w/ usual set of Math Models~~ → 166.32 for analysis of how to make such Grammar to encompass these 14 methods of induction.

(15) ~~Vapnik's~~ characteristic system → See 228.01 for Hesterlink

.13 We also want an algm that looks at an induction problem & decides which of the ≈13 methods should be used on it — (what wts). This can be Lsrch. If we want to get info on how each of the methods works on various problems, we will have to "Calibrate them" on lots of problems — a very big cc operation.

To make a grammar for the ≈13 methods, first get each as much as possible — possibly if poss. merge several of them — form several merged methods of induction — then try to "Merge" them.

I'm confused about 2 kinds of "lrng" & how they interact.

① First: GPD learning, Modifn: Changes rel. wts of the cands on (or for an input problem).

② Second: Modifn. of the Cand set.

Since in ① GPD gives wts to All poss. cands, ② is, in theory, imposs. — no new cands can be [illegible] imagined/created: In practice GPD will initially give many (most) cands pc's too low for us to consider those cands. "Discovery" of a new "v.g." form amounts to discovering a very hi pc deriv (or set of derns) for the "new" form.

Probly best thing to do would be to do both ① & ② at the same time.

.32 In my defn. of the General induction problem, I didn't include Time needed for soln. So defn. of an operator induction problem would be to Give {the Question, Time for soln.}.

.35 Also are (Q, A) induction problems ~~at~~ a universal set? Is the updating problem (including time) of this type? Is the updating

See 166.32 for a new way to help make "Grammar" for set of IA's Induction Algms.

4/28/01
IDSIA

.00: 152.40 : For QA probs: We are given a {Q_i, ?, T} for data, a set of [Q,A] pairs
.01 closely related to the present problem; Also, a large bunch not so closely related to present problem.
The ALP "solution" is that of (Sol 99)(2 kinds ...) — requiring T=100.
+400
T. soln for is a set of [(Pems)] (= conds), (there are of greater induction pems, Pri.
(Updates are to recent set of relevant QA's, to present Q & T.), P_i'. — Pr is
.05 prob. that pem will have best soln in time T. → (.23) → (154.00)

.06 (EN) Pems are of various forms wrt TC. In Many of them: put in data, a fixed time elapses,
depending on Q & data; then output (pd on A's ... Most likely first). Giving more time
would not help.
Another approach: extend to RL? : which would better do more rates as CC
.12 expanded

→ T. problem of induction now seems to have about all of the diffy's
of an INV/OZ problem solving TM : (of 140.00 − 141.40) — which is O.K. — It suggests I'm not "cutting corners"
Also, the Diffs of 140/141 should all be examined for relevance to Pure Induction Problem.
.15 → Or consider Z(41 type Pems: They can give a quick response, using PC's from last update,
or, they can look for new definitions and perhaps response at perhaps look for more data ....
.17 But for some forms of Z141, this process eventually terminates,
— For ALP, of course, it does not.
In .15-.17 we may consider part of the process to be "Updating". — So perhaps in general,
t. boundary betw "prediction" & "updating" is unclear. — during "Updating" one does as well as possible in the time given,
<Perhaps Relevant is the concept of "Summarizing Machine"> — if more Updating is needed, it's done during the problem soln.
Hwr. in 140.08 (updating during Lsrch), t. boundary betw updating & predn begins to Blur!
.23 In t. system of .00-.05 — T. data is in 2 parts ① [Q_i A_i] closely related to present Q,
② [Q_i', A_i'] more distantly related to present Q. Part ② is, presumably included in
t. pems by t. updating process, which creates the "appropriate" "Summarizing Machine".
If we omit OSL, Part ① can also be integrated into t. Pems (by t. update algm) into
.27 t. "Summarizing Machine" → 154.00

T. Time param in induction probs. can be very uncertain! Perhaps treat it as
an "Anytime" problem — see .06 − .12

CODING BY BACKTRACKING

:00 : (129.06) (29:27) : This is § 4.1.1 of "T. report" ! T. problem here is just how to do **Backtracking**, when t. no. of ll codes for t. corpus is < k (k is 100, 1000, 10k or whatever)

We want to do "minimal backtracking" — also limit on time to do it. Perhaps go back b steps, in all paths, looking for codes for t. augmented corpus. Use 2 cost to terminate non-converging trials.

.04 Each time we augment t. corpus by 1 bit, t. no. of codes → k/2 — I'm not so sure of this ! — see (.21) — so we have to backtrack to find k/2 new codes. [Normally, we don't have to go further back than we did when in t. last time we augmented corpus by 1 bit $T = 2^{-\ell_i}$] ← This is very import — if true !

The time, T to use for t. PsT computation bounds, is unclear. We have 2 params to chose to get t. k/2 new codes : **b & T**

Keeping k large, makes it more likely that we will find ~ k/2 codes — i.e. t. no. of new codes we will find will fluctuate. If k is bigger, it is less likely that the fluctuations will kill us ("Gambler's Ruin"). Also, probably large k may mean we tend to have shorter codes (= more backtracking) — More accuracy, better compression.

Perhaps chose T in view of how much time we have for t. process of coding. We can take b large enuf to find ~ k/2 codes,

So T & K are what we chose.

Remember: our goal is to get best compression for a given T. (By "T" I mean average time spent per bit coded.) — It probably translates to something like $T \cdot 2^{-\ell_i}$ = time limit.

.21 In general, I'm not at all certain about t. k → k/2 w. each bit coded ((.04)).
It would be nice if I could experiment w. this. Maybe try w. < TRANC, — say a pushdown stack machine.

But I should be able to at least write a pgm to do t. backtracking for a given T, k, b. It might mark, if for each of t. k/2 codes, I found 2 new ones, [illegible] (after augmenting t. corpus by 1 bit).

.29 Another tack : Do "slightly unsgreedy parsing" of t. corpus. First, codes for next symbol (Greedy). Code for next n symbols (NGreedy n). Select "local code" w. max compression (= no. bits out / no. bits in)

.31 Another tack: Try k codes for t. next 10 bits; pick k best. Retain only first bit of each code: loop to (×) [Bar going only 1 bit to end of each loop is to keep t. memory from being too large — by retaining all k best codes in memory — no need to retest most of t. stuff]
In .31, be sure to retain parallel codes. — keep best k! codes for corpus "thus far".
Is .29, or .31 equivalent to "Limited backtracking"?
Note: In both .29 & .31 we have to use a T (or a $T \cdot 2^{-\ell_i}$) time limit for each trial.

When we get a code that codes t. last bit of t. corpus but extends several bits into t. future : be sure to retain this code — it could be v.g. On t. other hand, t. "future" may be available (& will be, when we are coding a sequential corpus) — so if this occurs, usually we can tell if t. code bits "t. future" & reject it if it doesn't.

—155:00

152.27
.001 152.03 : Try to write out just how the system of 152.00-05 works.

"Part(E)" large set of older QA pairs. | "Part(1)" new smaller subset of QA pairs | new Q, T.

I'd like to use Bean soup in defined ways as an example, but

It is not directly applicable to operator induction.

Perhaps: in regard to first discuss operator induction & some possible models for it.

for Operator induction:

(1) Is {0, 0'} algebra much used in the system?

(2) Zipf can be used to compress Q & universe & A universe.

.09 We can then look for "Markov Models" implications" in which a symbol in Q's grammar & probable

.10 possible predecessors a symbol in A. ("Causality")

.11 Also Note: If we do find "objects" consisting of symbols in Q & A, then we get a

default bandeau model. $P(Q,A)$ rather than $P(A|Q)$: Recall $P(Q,A) = P(A|Q) \cdot P(Q)$

In .09-.10, defining $\frac{P(A)}{P(Q)}$ for $\binom{A}{Q}$ gives — so we could do all of

get $Q(Q,A)$ then get $P(A|Q) = P(Q,A)/P(Q)$ — But this is not what

we do when we do "Operator induction".

Consider induction problem: $(\text{Solve}, X+1=0)$, "Solve" has been defined by many QA's.

$(\text{Solve}, X+1=0)\,(-1)$: we want an operator that maps $X+1=0$ to -1 via certain "Solve",

So "Solve" defines an operator Inv that maps $X+1=0$ to -1.

So certain parts of the Q will enable certain sets of operators that have the

rest of the Q (or some other part of the Q) as input & the Answer (or part of

the answer) as output.

So the T.S. Q is arranged so that TM knows that trumpet we want to use the

"Solve" operator on "X+1=0". Now: How did TM learn the details of the "Solve"

operator? (How does TM learn to solve eqs from Examples?)

Note that in the QA problem, T. comes often unsupervised in addition to the {QA} set,

a larger "data pool", D.

For a preliminary analysis, consider Zipf prodn-model:

for final report, use QA model.

MIN BACKTRACK

.00: 153,46 ! Another view would be to consider all 1024 10 bit codes: which of these codes is most like parts of corpus? Take to 100 best. Retain first bit of each: consider all 1024, 10 bit extensions — loop to (2), Perhaps use Tz^{-d} as criterion where d = level in sorted list of 10 bits into machine (as trial code). Also use more structure of 1024 codes to reduce repetition of some trial parts.

.05 5/10/01 A serious aspect/difficulty of the "Min Backtrack" system: Say we have stored "best 10 codes". What we really want to store is the state of the computer system at the end of each of the codes — so each could readily do prediction & continuation. That would work fine! — But to do "Backtracking", we need to save the states of the system at the "Backtrack pts" It is not nearly to save all branch pts. of each code. Its only the Branch pts that seem critical, & every time TM has spent an additional ↑ seconds on a path, a machine state image is saved — or we have the changes since the last save state. Every K storage pts, we store the complete state of the system. Reasonably, reconstructing a state from the changes from previously stored states, is much faster than actually running the pgm. — But this will have to be looked into.

.16

.17 

We must store a code of "Lisp" pgms are stored. Each node is a computer word. It contains addresses of Node it came from, & addresses of nodes it goes to — so we can traverse it. Not any way.

For each downward address, we tell which path was taken, also whether it is a true path: (1) give wrong bit for code or (2) Timed out before giving bit (gone to bottom) (3) Was not tried yet.

Min Backtracking from a particular node means implies that that node was wrong or timed out for continuation of corpus. In other cases, backtracking involves moving "up" on the graph until a type (3) branch has been found, and then trying to code for corpus from that branch. With certain "Timeout conditions", we may also decide to spend more time on nodes that have "Timed out".

.27 Actually, all of the "10" best codes can be stored in a single data structure like .17 – .29 We also have to integrate the "state storage" system of .05 – .16 into .17 – .29. I guess what we do — certain nodes have pointers to places where entire state of system is stored, or differences between some more distant previous state is stored.

Here will be a trade-off As one ↑ the no. of code pts. at which the machine state (or difference state) is stored, we will ↑ speed of backtracking, but ↑ other costs of system (memory). So we have to minimize a "sum" of time & memory.

The time limits will partly vary w. depth in net: Time limit = $T \cdot 2^{-d}$ "d" is distance from top of net.

178.00

4/30/01

IDSIA

GE 4281 | LO:11
49.96
GBM.D 2272

.00/158.40! T. matter often new Grammar for each Q isn't too bad; but a grammar for t. next Q may be diff't — but it's closer to what I want!

(previous / Post / Past)

T. Poggio is beginning to sound like SGA!

.03:40 In .36-.40; There is some confusion! T. $(w_1 w_2)$ can be "probly" that cond. is best.

or simply t. compression (pc ratio) induced by t. cond. — a "G" value, which is also a pc.

.05 [on 158.03 again! Problems: Q_i; answer A_i. We want a good mapping from Q_i to A_i on past data.

One way: find best F s.t. $F(Q_i) = A_i$. But F is stochastic operator

so we want $F(Q_i) \rightarrow [A_j, p_j]$ i.e. P.D. on $\{A_j\}$.

We want F → UTM-like program

.09 $j(i) =$ ⇒ $Q_i \rightarrow A_j$ We may $F \Rightarrow \prod_i P_{j,i}$ is max.

$P_{j,i}$ = p.c. asso. w. correct soln. of Q_i

One way to do this is to have many F_k's; have GPD try to guess which is best for a particular problem — or give a set of wts for the F_k's

GPD is uniformly 1st Alg. formulation

Updating is in 2 forms: ① Re-assignment of F_k wts, in view of new data.

② Creation of good new F_k's 2nd improvement of old ones (in prescribed order!)

.20 .05-.20 is a 2 step process. See discn of 157.30 2 step process!

~~Small-scale problem: Dot. $[Q_i A_i]$ pattern directly. Find as good a $F(Q_i) \rightarrow [A_j, p_j]$~~

Also, within F's: changes of pc's of symbols; defining new symbols, replacing of all symbols

2nd pass.

Q: in .09 we may want to include/ (not coding of $[Q_i]$.

.26 How may Updating problem be formulated as an induction problem?

.27 · In a simple statement of problem it seems like t. updating problem is an induction problem, but

.28 it is t. main problem we are trying to solve: to find good F for $F(Q_i) = A_i$.

.29 Here It seems that t. way F is updated can be improved as TTM gets wiser.

149.36

148.25 ← discussed when

[When I first began writing about t. TTM that did induction only (say QA only), I was into t. difficulty of "27, but I quickly drifted into a soln. of .29, & was unable to applied near .26!

In .28: A good meta problem would be to speed up t. improvement of F.

.37 So Induction says: try to find as many good progs/codes in available time.

Updating says: try to improve .37" but this sounds very unlike .37 itself!

(1600

4/29/01

IDSIA

.00: 157.40: I'd want t. updater to create new (good progs / pgms) for specific problems. (i.e. as a function of Q_i.

t. a.p. of t. cands (in dist of Q_i: an unconditional P.D.)

.02 Lets Go Thru t. operation of t. system again! We give the system an induction problem (including T).

GPD looks at this problem: Outputs a set of [cands, P_i]. A cand is any pgm (string) that

might solve t. problem., & it is GPD's estimate of probly that cand will be best for

that problem. The output of a cand, is an induction on t. data. It is efficient.

4/30/01
IDSIA

.00: 159.40: $\binom{159.40}{149.14}$ → 149.02 — If just describes how the normal update algm works: It is not clear that this need to lead to any good algms. — Nor does it look like self-improvement.

((from 149.22 & 149.36 one gets the idea that including T into may be an ptr in defining True Self improvement. But I have to check that.

1) In T. recent (gross) Augmentation of Sol 89, the cards could be any pgm — including any such scheme — not nearly Lsrch. This seemed like a system that could be creatively good: So "Self improvement" could be positively regenerative. (Positive Regenerative)
Can I work something like this into present "Induction only" system?
Look at the Augmented Sol 89 system (W. "12 or 13" post Sol 89 features) & see whether I can add useful features from it, to the Induction only system.
Remember that Induction is an "OZ problem" — But instead of Lsrch, maybe use UOV soln.

At any pt in the History of TM, it will be using some algm. for induction — for finding short codes of corpus (approximately "short codes"). If we can show that the problem of improving this Algm. is a Induction problem or an OZ problem or some other kind of problem that the system can usefully work on — we have it made!

.18 5/1/01 AHHA! T. problem of (S.I.) Self-improvement should be formulatable as a QA problem. If if it can't we don't understand what we're doing! If we can do so, we simply start w. good enough heurs to do a reasonable tsq. — Hopefully, leading to intelligence state in which it can work on (S.I.)
— This would be true, even if we had a special Mode for (S.I.)
In which case we'd need a MC Thm, to merge data from QA probs & S.I. mode.
.23 (DEF) So I have, as before, the need to formalize the problem of S.I. In Augmented Sol 89 (≡ AS89)
It was an OZ problem (or a modified OZ problem?).
We should be able to "tell" TM "just what we want solved" in the case of the S.I. problem.
.26 Well, basically, the problem of induction is (≈) to get max $\sum 2^{-l_i}$ for available Time.
We have some Algms to do this. We want some Algms that do it better.
The "≈" in line (26) has to fixed, hrr. — THO it might be possible to have a QA machine with $\sum 2^{-l_i}$ ~~[struck]~~ force. The reason is: When we give TM QA problems we would always have to put them into following form: Old corpus of QA's: already updated. New corpus of QA's: Needs updating: we want to improve TM's updating algm.
.35 Another thing in amendments: For many QA probs, we have a large Data pool common to all QA probs. The amt. of Time TM should spend compressing (coding) this pool, is unclear. ——→ (162.01)
.37
.38 We can (effectively, not nearly optimally) divide our inducting algms into 2 parts:
① One that chooses ~~[struck]~~ sub corpus, ② that does $\sum 2^{-l_i}$. It is

T_i

.00! 160.40! easy to compare algms with their effectiveness in part 2 — just compare them w. same sub corpi. To compare w. ① & ② together, we have to use the (T_i) limits that have been used in actual problems. We can use $\prod_j p_j$ as a score; where p_j are the (normalized) normed pc's of the correct A_j's. There was a problem with initialization — or "getting started" when the device was self-improving. I don't remember exactly what the problem was, but the soln. was to use 100% of corpus for a suitable "starting period", then let TM decide on corpus size for subsequent use.

The foregoing Gore for S.I. of the QA TM, may be more or less clear; How to integrate it (via MCT) with the production Q.A. problems must be examined.

→ So perhaps the QA TM is more or less defined! The ratio of S.I./update time has to be user selected.

.15 O.K.! So it's a QA system! In steady state! TM has an operator F (or a set of operators F_i): GPD looks at latest problem (usually a set of recent QA's) $[Q_i A_i]_{new}$
any? e.g. one operator. Say 1 operator, F: TM has to integrate $[Q_i, A_i]$ into F. (→ Modify F)

One way is minimal Modn of F! So F has same responses for all previous Q's,

.19 Well, F is a stoch operator! we have already coded all of the old QA's; we

.20 just have to code how $[QA]_{new}$, we do this using the old defns & freqs used for the old corpus.

More generally we have $[\Sigma Q_i A_i]_{old}$, $[QA]_{new}$. We want as good a code for the total corpus as possl. For small "T", .19–.20 should be used. For larger T, Some Backtracking is used. Since the corpus isn't sequential (in the simple sense — the sub sets of QA's have been grouped, usually by the user — in groups now "problem sets") — "Backtracking" is not defined by a scalar "look back" measure.

TM has some sense of "clumping" of the corpus — that is used ① in backtracking ② in the Partition problem of deciding how much of the corpus to code. (because of finite T) (① & ② are closely related)

TM has an Algm that is able to look at the $[QA]_{new}$ & decide what other sub corpi it is "like": which it should be "coded with" when T is small. For very large T various subcorpi are merged and TM looks for better codes for these "Macro" subcorpi.

So at any of the times TM has Algms to do the things — to look at T & Partition (or decide what partitioning to be), & algms for coding sub corpi.

162.01

5/1/01

IDSIA

25 9/5 40/32/72 40/32/72

.00: 157.40 : To update those Alpms, use the Gore(S) of 160.3

.01 | 160.37 Consider the problem of 160.35: (large Data pool is part of the QA corpus)

Try to see how I would use such a subcorpus for helping answer Q's.

I guess we need some (at least) minimal compression coding of the Data pool before we can use it at all,

It must have symbols/words in common w. the QA subcorpi.

.05 (DEF) We can Index the Data Pool (DP) so we can more easily find sections related to various QA subcorpi: (perhaps) partition DP so the partitions belong to partitions in the [QA] subcorpi.

[5/3/01] Is If my method of working WON is only ok for monotone ↓ G(T), then this the early ↑ section of G(T)'s "special problem". If I can't work it, give it to TM: A TM of this sort (using WON) could only do problems in this manner, w. large T, (or use Lsrch) — so we start out giving only those kinds of probs, until TM solves that problem.

IN GENERAL: If I have a prob solving method that works only for T > a threshold, To, then give TM T > To to work on finding methods to solve that problem w. T < To. Perhaps this idea can be applied to many situations.

I This idea has also been used in getting TM to solve the problem of

.20 how to partition a corpus for short "T" assignments. → (163.00)

5/7/01
IDSIA

00:162.20 : As is, I have phrased the Up Adaptive problem for a QA TM as an OZ problem. A QA TM can work on any problem in which I can "teach" TM the concs needed to define the problem — e.g. for Humans, define the problem using Math notation. Then teach TM to understand each of the concs used in the Math notation.

Actually, this is not much worse than having TM w/ Built-in routines for working OZ problems, because ① To work on such problems, it would be well for TM to understand the (Human) defn of OZ problems, in its internal lang. ② TM would probably need defns of the concs to usefully work on the problem ③ If the defining concs are hard for TM to learn, then TM is not ready (TSQ-wise) to work on the problem.

④ By defining OZ probs in this way for TM, TM is less restricted in its set of possible solns, than if I had a special OZ solving Algm that TM had to use. (One same Algm of this sort could enable TM to use any OZ solving technique it could possibly devise.)

⑤ I could use this method to get TM to work on the "Speed OZ" problem of $G(X) = G_1(X) \cdot F(t)$, where G_1 is a score on soln, X, & T is time needed to test proposed X, & F() is a known (to TM) monotone ↓ funct.

To get TM to learn defns well, we could also provide negative examples. — We want to test TM's understanding of a defn. by giving it lots of tests of weird cases, to make sure it understands the defn. This same problem occurs w. Humans. (2:37/8)

In General, we should be able to teach TM any legit Math concepts. If we cannot do this, then the concs we are trying to teach are "extremely interesting" & possibly "meaningless". ★

5/13/01 IDSIA

REV

164

.00 : General Picture of QATM!

1) Desc. what it does: What is the QA problem, various forms, why it is a useful form.)

Some forms a) [$Q_i A_i$] given: new Q asked.

b) Corpus is divided into [$Q_i A_i$]old [$Q_i A_i$]new, Q ← related must go [QA]new.

.04 c) Corpus as in a) or b) but also contains a subcorpus D — a "data pool" (say an encyclopedia or a dictionary or newspapers or reports) or ...)

It is a useful form because probably any well defined problem can be put into this form: TM would first have to learn to understand the concepts in the Question. While this would seem like a formidable barrier to practical application of the system — it is not. To solve a problem of any complexity at all, a major hurdle is to understand the words as well as possible, the concepts used in describing the problem. Educating TM in these concepts should be an important part of the TSQ that precedes the problem.

2) How does it do it? i.e. QA induction : a) Steady state operation of mature system; b) Early operation c) initialization.

a) Steady state: Given large corpus including Data pool (or not) and a new problem Q_0! — an optional And a time limit T.

At any time in TM's history, it has a function F(Q) that maps Q's into a P.D. on possible answers. Usually this P.D. will be in the form of an ordered list of [A_i, P_i] with the A_i's of highest P_i being earlier in the List of (A_i, P_i) pairs.

For small T, and simpler Q's, ... F has already been updated "a" so the reply involves little new computation. || For more complex Q a/o larger T; The reply process is more complex! The corpus will be partitioned into 2 parts ① The part most relevant to Q (This will include partitioned one of D (.04) as well.) ② The rest of the corpus → TM has to learn how to do this partitioning: Do big Gore fast. trying — Tell how this done

Part ① will be large if T is large: For very large T, it may include all of the corpus,

As a function of Q_0, TM will apply a relevant induction algm or set of induction algms to ① to produce P.D. on Answers, [A_i, P_i]

It TM uses ... TM is given ... Ans from standard techniques of search of A_i for small T, Then for large T and larger T: we parodie ... updating A_i's P_i: first it may do ... on TS (1) ...

5/3/01

IDSIA

&

1956 – 1975
Harmony

.003 : If F(Q) is the function used for single Q's & small T, then the processing of complex Q_0 w/ large T involves a kind of "updating" of the F() algm.

N.B. In explaining this, go thru the process for small corpus, & D = Λ. Then do it for larger corpus & D ≠ Λ.

what is D? "DATA Pool" 164.04

For small corpus & T that is large for that corpus (This is the case in early training of TM), when the entire corpus is assumed to be relevant to the Q_0, For very early trng, TM has only a rough, a priori ordering of induction algms to be used — This ordering will be indep of Q_0. The algms & their initial wts/pc's given by the designer. [perhaps give list of ~ 12 algms of (150.33ff) (288.00) to give idea as to what I mean by an inductive algm]

The induction algms will differ considerably in the ease with which they can be "updated". E.g. Say TM has been given a set of [QA] & a Q & it has used induction algm Alg_0 to obtain the answer list. Depending on the induction algm, we will retain in memory, critical parameters that describe the solution to that problem.

We give TM the correct answer to the last problem, & a new set of [QA] & a new Q_1. For certain induction algms, there will be standard methods to "update" the parameters of soln to the old list of QAs, in view of the augmented list and the new Q_1.

At the other extreme will be induction algms that treat each modified corpus as an entirely new problem to be solved, involving no "carry over" of info from previously solved problems. There will be various degrees of "ease of update" — depending on the induction algm.

Also varying much with inductive algms is the way the update proceeds as a function of the amt. of time available. Often, (but not always) increasing T will mean the amount of corpus to be considered in the updating.

IDSIA

IA's

.00: 165.40 : Q. by reader You are using this large list of Ind. Algms that has been created by the Sci. Community over the yrs. to solve various problems! Just what does your system contribute?

Several things: (1) [redacted] When one is given an induction problem, it is not clear as to which ind. algm should be used — how to apply it, how much of the corpus to apply it to in the available time. The system develops a function that does these things.

.06

(2) Most induction systems (in the Mach. Lrng community) do not use info found in one problem to be used in another. The proposed system is supposed to bring to bear, any and all info obtained in solving [redacted] previous problems,

(4) When the system has gained enuf expertise in the proper Domains, it is able to usefully work on the task of self-improvement. This will involve devising new induction algms or modifying (improving) old ones.

Defining (Discovering)

(3) Because of (2), (the system is able to do incremental learning" and) [redacted] can be brot to a hy degree of problem solving skill, by a suitable Trg. Sqe. — [redacted]

R

Q2: Often difrnt algms will be used in difrnt domains. If these domains are inductively related, how... Mixt Crps Theorem. MCT able to deal w. the fact that difrnt ind. Algms were used in the Domains? .32

(1.5) ~~The QA format enables focus for the sq. A very large range of problems can be put into the QA format and are thereby accessable to the system.~~

The QA format for problems is clearly general and enables the system to work on a very great variety of problem types

SN On comparing 2 PD's that have difrnt pc & differnt cc's (i.e. "T") for various problems. Occasionally, one PD will be uniformly better than another, in all ways: i.e. for any T value (PD1) will give a larger P value for a given corpus than (PD2) (? ≡ for all corpi?)

— But more commonly, one PD will be better for a given T for a given Domain (≡ set of corpi) than another PD. This kind of comparison betw. corpi is very useful, since we can then decide which is better to use for a particular (problem, T) pair.

.06R Seems to be an impt Idea that I should (eventually) work on.

.32 Re (.06R) — In comparing the 12 induction Algms of 150.33ff, one way to compare them is on how they are updated & "improved". e.g. for a given corpus, a Predn algm will be characterized by certain params — as the corpus is augmented, these params will change. Also w. any corpus, we may introduce new params a/o discard old — this could involve "parsing" or an analog of "parsing".

Good!

Various elements in difrnt induction algms can be made to correspond in view of correspondences in their update Algms. This could be a key to making a "Grammar" that covers many of the 12 induction algms of 150.33ff.

Another pt. of comparison of difrnt ind. systems: What parts correspond to 2^{-RL}?

→ line .05 - .06 explains this.

.00 : 166.40 : [151.13 — .40] discusses ways to generalize / interpolate ; "15" induction methods

In my discussn, there has been little or no consideration of How the data is "coded" by the ind. system (whatever that means!): [It may be possl. for the DATA to suggest new induction systems. ← its not clear how I can do this!] In Alp (& many ind systems) we often have 2 part codes: We want to ↑ pc of the product of the 2, but ↑ pc of either by (1) finding

.05 new codes for the system descrn (2) w/o new codes for the descrn of the corpus in terms

.06 of the system. (Also Note) [ALP supposed to be able to express all 15 induction methods in a "common format". If true, this should enable us to see common elements, similarities, differences, betw. the 15 methods.]

.07 Go-thru 164.00 — 166.40 : (Perhaps rewrite it) : Just whats missing from

.08 the descn. of the system? — what parts are "vague descr." &/or need much work? (.23)

Q: What do we do if >1 answer is correct? Say $Q_1 A_1$ & $Q_1 A_2$ are both correct pairs: Then we mite give both to $Q_1 A_1$ & $Q_1 A_2$ as part of the corpus. A Q mite be how much wt. to give these 2 data's, — full or "½"?

Q: What if none of the answers are correct? — Well this could be because TM was unable to get even near the soln, so it assigned very low pc to it — or because there was no soln.

If its the second case, then its not a good problem! In the first case, it means TM wasnt prepared to solve it — or T was too small or C_j's too large, so TM would need more training before giving the problem again. Or TM should try a diffrnt induction scheme

.23 : (.08) A simplified version of the system of 164.00—166.40 :

Inb. Input is a [QA] set. The problem is Q_0, T. (T = Time allowd for soln.) | we can also deal w. Q_0 w.o. time "anytime problems".

— No!

Say we are in steady state: The system has a (long seq. of QA pairs [Q,A]) & It's given a new Q_0, T. We send Q_0, T to f( ) f(Q_0, T) gives (p_i ≥ p_{i+1})

Def: As output, a set of IA's (Induction Algorithms), w. assoc. pc's (p_i) [IA_i, p_i] in rough pc order.

TM then uses IA_1 for time $p_1 T$ on Q_0 & outputs a pd on possl. A_j's.

" " " " IA_2 " " $p_2 T$. " " " " " "

IA_3 $p_3 T$

We then "sum" these pd's by adding. Say IA_i for time $p_i T$ gives [$A_{j(i)}$, $P_{j(i)}$]

(A bit uncertain / A bit "weak") The sum of is [A_j, $\sum_i P_{j(i)}$] I guess normalize the df. so we can use it for feedback.

this is "the Answer" ; we may take the A_j of max P_j as the answer, but how one uses the Df in Answers depends on the OS r's applicn.

IDEA

.00: 167.40: After we tell TM what its answer is, we give TM a time T′ to use for updating. Say TM has used 3 IA's (in attempting) to solve this problem. We will then update each of these 3 IA's.
Diff't IA's vary in the way they do updating, but typically, the function F will depend on a set of parameters that depend on the set [OLD Corpus $\sum Q_i, A_i$] (this does not include the recent Q_0, A_j). These params might be the frequencies of occurrence of various symbols used to describe the IA. A minimal update would change these params, in view of the new datum, Q_0, A_j. More extensive updates can be done if T′ is large enough. These could involve defining new symbols — which adds new parameters, and evaluating the params associated with the newly defined symbols. Again, if there is enough time there can be a reparsing of the IA desc, in view of the new symbols.

It might be best to have a special section on updating: Tell how updating is done for [redacted] sequential corpus, Then outline how it's done for a corpus that is a [redacted] large function composed of many sub-functions. 116 75 67 121 65 63

5/5/01 Perhaps start out report w. a long section on Induction. Treat first, seq. pred., bag pred., then Operator Induction (OP'ind) & first do CB < ∞. Then treat CB < ∞ — giving various problems & tentative solns. Seq Ind Bag Ind OP Ind.

.19 After this preamble, the desc of QATM should be easy → .31

(SN) Make List (Bibliog) of Bugs, Defects in system. Then try to order them w.r.t. importance.

.22 (SN) Perhaps if all G funcs are monotone ↓ in certain way then Lsrch is optimum. Superficially, this seems to be false. Work/unit time spent on a cand is $\propto \left(\frac{dG}{dt}\right)^{-1}$ ($=\frac{1}{\text{slope}}$). So t. ratios of t. slopes of G funcs of all cands would have to be invariant, thus — So all G funcs are t. same except for dilation in T direction.
This would mean $\int_0^\infty G(t)dT$ would be diff't for each cand — it would be $\propto \left(\frac{dG}{dt}\right)^{-1}$ (here the 's must be for all cands)
Hvr. in earlier work on WON, I vaguely remember finding Lsrch to be soln to WON under certain circumstances! It may be that second ½ of line (.22) is wrong.

,31 (.19) Tho t. problem of SI is related to induction, it's not exactly an induction problem. It does seem to be an OZ problem, & I do have reasonable Goodness fnct. As such, it is "solvable" by Lsrch, but probly better by WON. — T. main difficulty in both approaches is correctly being cands. — A Tentative "soln" is t. "partitioning" problem — subsets of cands, w/ "representative" for each sub-set.

Hvr, .31 seems to not be a critical problem. TM will not spend time on it until it is "rather smart" — able to work diff't Math probs, diff't OZ probs. We should be able to "explain" to TM what t. problem is. (169.00)

5/5/01 | 169

IDSIA

.001, 168.40 Is there any "updating" problem related to Ses's (168.30 - 90) SI problem? Or just t. standard Induction update(s)?

Another aspect of t. system that seems to depend on Lsrch (i.e. on "sequential index" of cands) is choice of a IA for a given problem. Here, use of Lsrch is not optimum in a narrow sense, — but enables us to get more empirical data on what IA's are good for what problems, Tho it does seem to do this test, I'm not at all sure it does it in any "Optimum" way (or even much good at all!)

So 2 main pts./ of system as of now ① Choice of IA (viz "F") ② SI [redacted],

Tho ② "covers" ①, we will need a good soln to ① well before SI can be invoked.

Re: ① we do have t. function of which looks at problem & gets PD on which is t. best IA. — A standard induction problem — over choice a IA to do F (in early TM) then later for later TM without problem of improving F — It does look like an "OZ" problem.

→ It may well be that TM could be very smart w.o. any SI. ←

So perhaps SI could be a later trim to t. System. [redacted]

[redacted] .22 So getting F( ) initialization & Improvement is an impt/ initial problem. Another impt initial problem is [redacted] a set of IA's that are fairly good, & in forms that TM can readily use.

→ An impt. diffy w. .22 is that it is an OZ problem — & if we use Lsrch to solve it, it's not very efficient, because of correlations between pc's of cands, Hrr. even if we solve .22 inefficiently, we can still end up w. a v.g. soln. — if all that "inefficiency" means is "slow".

.30 → A more Serious Criticism would be t. non-universality of t. set of Models used for F( )

.31 — & perhaps of More importance: That all info obtained in t. TSQ isn't properly applied to

.32 t. soln. of t. F( ) (induction modeling) problem | See 270.30—.37 for Resoln. of .30—.32 | ABCDE → See 259.40 for one universal Model: 258.09 ff also looks universal, t. counter example of 253.21 is wrong — see 267.00 ff for why that Model

So just draw of a picture of t. system a s of now & co. info. & deficiencies || May be Universal.

.34 Note that F( ) has to do 2 things! ① decide what parts of t. corpus are relevant to

.35 t. problem ② decide on what wts. to use for t. IA's. Later (@260) I think that it would be good to have t. IA's choose t. subcorpi (Summary/Summarizing) 7/19/01 → Now it seems better to do this: .34—.35.

.36 (N.B.) for each sub-corpus, each IA may have a "summary machine" that so that t. amt. of time needed to make a predn. is indep of t. sub-corpus size (except for OSL aspect). If we want to "update" this induction for a given IA, we can have default (amounts/depth) of updating: ① width of trees of symbols ② More symbols ③ reparsing.

In General, each IA will have "summaries" for a set of sub corpi. For a given

(170.00)

IDSIA

:00:169.40 Post, say the prog a reasonable subcorpus C to base the induction on a IA, cand; . We look at the subcorpus that IA has "enumerated". We pick the largest one which contains C. (If possible — if no (summarized) (subcorpus) of IA contains C, then we try a different IA — that may have more experience in the relevant domain.

.04 → (172.04)

.07 A main Q is: "Can this TM learn practically any thing?". In line with In know. 578fn2, all heurs can be derived by suitable fsqs. & suitable operators on past data. (This I need to flesh out this idea in more detail). Also, that fn. said that all heurs could be expressed by modules of a search pd in Lsrch. (But this seems to require updating Lsrch, so the CJS $\frac{I_i}{pc_i}$ is not really true.)

Args like .07 were important in making it likely that the device of sol 89 would work well w. "adequate fsq".

For the recent QA TM, I'm thinking of mainly #2 IA's: (AZ141) Augmented Z(41) & "[struck out]" Minimal Backtrack [struck out] Search".

With AZ141, the search technique is somewhat "greedy". — It's less greedy if I save many poss parses of the corpus; Also if I look for [struck out] subtrees of size [struck out] > 2.

I think important Sol 89

Re: Sol 89 fn 2: It would seem that any heurs should be discoverable by noting regularities in (the soln, & traces) of previous problem trials (not merely only successful trials). The implementation of such heurs, however, would often use info from previous trials on a problem: which is not allowed in Lsrch. However, each "trial" (cand) can be a srch technique that can look at previous trials within itself — so a [struck out] any heur can be realized by a cand of this sort. There will be some (a lot?) of "wastage" since the cand will be blind to trials made by other cands & will have to repeat (many) of those trials. = "Z141 tie doubling"

.30 [5/6/01] On the non-universality of AugZ141: (= AZ141)! (Say we considered (common) subtrees of any size): Would AZ141 recognize $A^4+A^3+A^2+A+1$ (expr) could be simplified by $(A^5-1)/(A-1)$?,

.31 well actually this can be written as $\sum_{i=0}^{4} A^i$; which is longer

.32 or $(1+\epsilon)^N \approx e^{\epsilon N}$? The problem here is not of the cost but a term we still only # n in size.

better able to combine w. other terms to make "simplifies".

We might try AZ141 in "Minimal Backtrack coding".

But, I really have to look at various kinds of "regys" & see how AZ141 deals w. them or can be modified, (further augmented) to deal with them. AZ141 might be able to deal w. the regys of .31 & .32 by being given a suitable TSQ

172.00

5/6/01

IDSIA

Z141

.00: : Re, use of Z141 for determining "spaces" in "despaced" English text. I think that in general, Z141 "wouldn't know when to stop". If the corpus were large enough "Thoman" would be one word, & various other commonly occurring combinations of words would also come out as one word. A child learning a language often makes "errors" of this sort.

I complained about Wolff's method not telling ~~...~~ "when to stop". My method would tell, but usually too late! Almost all of the things it gives as spaces, would be spaces, but it would miss things like

IDS/A — D → R
Domain → Range

Perhaps use word "Domain" instead of "Subcorpus".

.00: 170.40: Could AZ141 be regarded as a special case of "Min backtracking"?

T. (Apparently) biggest deficiency of "S89+" = using Lsrch, i.e. correlations of pc's of cands. Would this amount to an acceptable "constant factor" or would it grow w. corpus size, — So it was a kind of "Scaling problem"?

.04: 169.36–170.04 (Earlier reference): f(prob,T) → of (QATM) goes from (string,T) to a pd. on [IA_z]

In updating f( ) we could, minimally, update wts (pr's of IA_z -).

Hvr, the IA_i themselves will have to be updated as well, i.e. they will be updated wrt different subcorpi. So when f chooses a IA for a problem, that IA already has a specific subcorpus that it summarizes (to various degrees of completeness).

So after f assigns a IA_j to a [redacted] problem, we know the relevant subcorpus of IA_j and we could spend more time updating it if T was large enough — i.e. new defns, reprocessing etc.

There is, hvr, a more "Global" updating, that is more time consuming & is done only occasionally;

.15 This tries to "link" various parts of the corpus that have not been "linked" before.

If we only have one IA (e.g. AZ(41)) then f( )'s main work is to choose the [redacted] subcorpus! There will be several of these available: each one having had its own total T expended on it — so f( ) will somehow make this choice.

.20 (Assume for only one IA (i.e. AZ(41)) When we do the "global updating" (.15), we would look for common defns in the various subcorpi (where the subcorpi overlap, we have to be careful about SSZ (necessary case counts) of various symbols). I think of "Analogies" being "similarities of 'objects' in structures" between apparently disparate parts of the corpus. (I'm not sure this covers all kinds of Analogy that I'd want to cover). When I actually do a TSQ, (or several disparate TSQ's) I will want to see if TM can really recognize analogies of various kinds (or any other weird regularities) — And the tsq. can be designed to help TM recognize such regys!

Certain kinds of "structural similarity" are represented by "common subtrees" in larger functions. These common subtrees have different kinds of arguments in the different subcorpi (≡ "Domains"). (So "Analogies" between "Domains".)

.34 Well: .15 may be "not too bad" if we only use one IA (AZ141)! — But if we use > one IA — (say Min backtracking — or any of the IA's of 150.33ff) then each IA will have its own set of subcorpi & it may be not diff to update a given IA wrt its set of subcorpi, — But mixing updated IA's of different kinds, may not be so easy! — How to do — The different IA's could have partially overlapping domains (≡ subcorpi). How to do "global updating" like .15 ff & .20ff between disparate IA's, is unclear.

5/7/01

~~IDSIA~~

Juergen

:..00 : : 5/7/01: Juergen writes "why not analyse binary seq. w. predn. of each successive bit, w. update Algorithm etc." (I lost this letter in STD)

1965–2000
Boot Health
23.5%/yr.
S&P ~ 11%/yr.

Also Reply to Marcus!
Note 156.00!
(Discn. w. Marcus)

Actually, there are a large no. of EA's for predn. of a binary seq. (G.e.o part of Sept of 150.33). I have a few additions:

1) ~~one too~~ For a ~~n bit~~ predn. of an n bit seq., it looks at the most recent K bits and asks how many times they have occurred in the past (k = 1...n-1). For each k value, there is a prediction and we take a suitable wtd. average of these predictions.

It is similar to LZ & a system proposed by Riss. ~~I don't know~~ ~~why~~ Suspect this is better for prediction — LZ & Riss systems were designed for memory economy; mine is not constrained in this way.

2) Another is an update of the system described in S64b pp — It compresses a corpus by defining new symbols composed of adjacent pairs of original (and previously defined) symbols. As described, the system of S64b ~~would~~ does not work — but it ~~might~~ will probably work with periodic refreshing ~~the recent refinement~~. The system is also applicable to more complex kinds of prediction, such as extrapolation of functions defined by a "functional language" like "Lisp". It is useful in the Inductive Question-Answer System. "functional Language"

3) Another is the "Minimal back tracking Algorithm". [153.00] [155.00] — Also described in 4.1.1 of "report". I have a good idea as to how this works but have not worked it out in any detail — as I have for the previous 2 systems. [178.00]

.22

IDSIA

≈ ALP

.00: 17240: Re: Mixing info/prodns from various IA's; Perhaps having a common format would
.01 facilitate this: e.g. All induction must be expressable as "compression",

SN On "Parsing" Once TM has solved a problem, it knows at least one useful parsing —
i.e. the parsing assoc. w. that solution/path

.04 In the "589+" TM, problems were solved in 2 stages; GPD looked at prob & GB
.05 & gave "Rough soln" which was a PD over possl. soln methods,
.06 Was could have a second process (not nec. Levelwise) — that look at this list of
soln. methods (& possibly correlational info — either than obtained either empirically/theoretically
.09 a/o by logical reasoning) & decides which one to try, in what order & how
.10 much cc to spend on each. Have TM (developing process) in SI model

The r. .04-ish model was used in 589+; it was not really used in QATM —
QATM uses several IA's, but in a different way. — They are not really regarded as
"alternate ways to solve a problem", but are combined by weighting their Bayesian
prediction Pd's. This last, assumes "independence" of IA's (tho it may be that
we don't really need "independence" to use this "wtd mean" — that is a very rough
move wt. to IA's that have "many dems" (or many Σ "dems").

Another Big difference bet QATM & 589+: IN QATM (& all induction
problems) we have the problem of "Mixing" several IAS: In 12, 34, 40, 17, 00-01
problems) we have the problem of dealing w. common cases common to several IA's, of Domains of different IAS
— of dealing w. overlapping, etc.

In 589+, we have alternative Methods of Problem soln. We do not "Mix" them.
(Tho, the problems are induction problems — we should!)

Well, OK.: Lets see how far we can go w. QATM!

We start out w. a corpus of QA's ("No data pool yet"); This is "compressed" via an
Operator function $H(Q \xrightarrow{T} [Q_i])$ we use as Goal: Max:

$(p_{cot} O) \cdot \prod_T P(O(Q_T) \to A_T)$ or $(p_{cot} O) \cdot \prod_i P(O(Q_i \to A_i))$

Also, (for simplicity) assume Rest for every Q_T's, we use: entire corpus for prodn.
Also, say there is only One induction algm.

As we augment the corpus w. 1 or more new QA's, the operator, O, is "updated" by
varying its params (freqcs usually), introducing new nonns (= symbols), & reparsing.
Reparsing can occur even when no new symbols are added, but when the "case counts" of old symbols changes.
The parsing device we could work O.K. if ⊕ the IA was good ⊕ TM was large enough for good induction
⊕ the TSQ was good enough.

5/8/01

IDSIA

Tu 8 W 9 Th 10

7:12 new sig. 7:15 done w Rubsig.
7:18 " "

[See .12]

.00: 17440 : T, next jump in complexity [redacted] may be large:

.01 a) we add more IA's. (still, we use all of t. corpus for all problems). ancT

.02 → So we have 2 stages of operation: F( ) looks at t. problem (& "new" data (QA pairs) if any)

No! F( ) looks at problem after all IA's have predicted: see (.12)

& out put a set of pi's for all of t. IA's. ; They tell "which" IA's was likely best for that problem, for that T value. F is supposed to give high pi to t. Best IA for that (problem, T, pair).

The Score of F is t. product of t. pj's (for t. j th problem) : pj is t. prob assigned to t. IA that "did best" of all t. IA's tested for time T. (appreciable cc)

So if we test 10 IA's [redacted] for each problem, This is an appreciable cc)

This is t. cost of obtaining a good FC).

When we do long T updating, to find concs common to problems that look dissimilar, we update each IA separately! There is no interaction betw. them.

.12 → If an IA is used on a problem, then it has to work all QA's in t. corpus.

What this means, is that All of t. IA's have to be used on all of t. problems, every time!

If we have 10 IA's then we take 10 times as long to solve problems: [redacted]

[redacted] After a while, FC) is of some value in [redacted] guessing which IA will be best:", so we get better preds. But FC) is used after all t. IA's

.18 have been used for time T each.

b) Like .01, but t. subcorpus for each IA can be different. I originally had FC) decide what part of t. corpus to use for each IAi, T value — (But perhaps t. IA's could do this by themselves.) (Rather than have FC) do it.) At any rate, I had F look at t problem & T, & assign a Pi to each IAi & an assoc. subcorpus for each IAi. IAi would work for time T pi on t. problem using t. assigned sub corpus. If T is small, IAi would simply update its old prob evaln — T. "corpus size" assigned to IAi would not determine how long it took to get a result —

If T was long enuf IAi would do lots of "updating" on its corpus.

.27 At each pt. in time, each IAi will have worked on a certain subcorpus for a certain total amt. of time. → It will have a certain expected pc. for preds of "legit" "extensions" of that corpus.

.29 So we have these triplets : (IAi, Sub corpusi, Ti, Pi) Pi is t. Geometric mean pc of correct responses

.30 for the (IAi, Sub corpusi) pair. This [IAi] with its "summary", contains all t. info that IA has at any particular time

In t. [redacted] "Simplified" QATM of .00-.18 ; we had all these IAi's operating on t. entire corpus. Say, for each IAi we consider only those problems in which t. pc assigned to t. "correct" answer was > .7, say (or t. upper 30 percentile of all of its predn. pc's, or whatever).

Those problems could be regarded as "its assoc corpus", & we could just give that IAi that corpus to predict & see if it did as well as when t. rest of t. data was present!

The "Simulation Game" (.14)

.001 175.40: [SN] On "Methodology": Seems that whenever the going gets rough on a particular model, I quickly find a new model that seems more promising! This continues indefinitely: I never finish any models!

Well, I did more or less "finish" Z141 & AZ141. I don't know how far I got on WON.

The last 2 models were S89+ & (recent) QATM.

S89+ was an analysis of Bugs in S89 & ways to overcome them.

QATM was meant to be a simplified (yet truly adequate) version of S89+ — but it seems not to be so simple! — Since it worked fewer types of problems than S89+ & S89+ was supposed to work all QA problems — it is clear that the diffys in QATM are also diffys in S89+! —

→ But S89+ may give a different way of looking at QATM's problems.

.14 [5/8/01] [SN] (But Related to QATM): Say we wanted [redacted] TM to create some stuff that was

.15 indistinguishable from Bach, or Kant, or Voltaire, etc. How would we make it clear that we wanted something "Original", yet "very similar"? This has (perhaps) to do with "Sub-corpi" & "Summary Machines" assoc. w. various IA's & Subcorpi. For this problem, could we use negative examples as well? [redacted] This problem of "indistinguishable" brings up the Turing test". (◯)

In .14—.15 [redacted] my mind is not entirely clear on just what I want!

.30 175.29—.30 contains a picture of QATM's "info content" (Note 175.30 (2R) in particular) Is this what I want? : is it adequate? Also note that for T→∞ I'd want each of the IA's to have the entire Corpus as their individual Domains. Also note: The T's of 175.29 is not adequate to descr. the info sort of IA's: The T's will have been broken up into pieces of various sizes during TM's problem solving life, so certain parts of the sub corpus will have had much more time spent on them than others. How impt. is this? Say I actually had the detaild breakdown of each T's into the amt. of time spent on each Sub-corpus. Is this an adequate summary (Complete Learn) of the info in TM at a particular time? If it is an adequate summary, Does TM have enuf info to solve the kinds of problems I want to Give it?

Well, in actual use, each IA could have some "cross coupling" w. other IA's, obtained during "long term Updating". I don't yet know the form of this "interaction": At present, I'm considering something like a "common format" for all IA's, (say ALP). It might be a lot clearer as to what's going on if I had a clearer notation of this "integrated notation".

.35 [My impression: that each IA emphasizes diff kinds of regularities: By giving them hi pc &/o by programming tricks that give better $\frac{pc}{cc}$ for certain regys.

Also, certain IA's are not universal, so they give pc=0 to many regys.

If we had CB=∞ for each IA; this would correspond to a apriori assoc w. each IA.

(But, because CB<∞, this simple view is inadequate. My impressn: that these IA's correspond to my "Generalized PD" — which includes cc for each prodn. Also those IA's have been partially updated, so their PD's have been Modified.

177.00

5/9/01 IDSIA

IA = "Induction Alg"

.00: 176.40! In view of 176.35–.40: How is "mixing" (of IAs) done? Presumably, each IA has its own special (short) ways of coding certain regularities.

. Try Mixing for CB=∞ ; then for CB=∞ for one IA; CB<∞ for others:

.03 Then CB<∞ for both. (Do I have how to do CB<∞ for even one? ☹☹.

.04 ≈ 176.35 again: For each T (≤CB), each IA amounts to a certain epnpd. — to some extent. (However, each IA has been "updated" to some extent — so it's a "summarizing machine" — T: effects on the idea of .04, is not clear.)

Perhaps a "summarizing machine" is too extreme a term. It's the same old IA, but it has to be applied to a new problem using a test in CB. When this is done, this IA will be much biased by the past corpus it has been updated (w.r.t. "on")

.11 As for "Mixing" of "formally updated IA's (completely updated would mean CB=∞)

The "common format" must be simply the production pc's (PDF) assigned by a IA for its problems in its "assigned" corpus"

.14 Say we have several IA's w. partially overlapping subcorpi, & we want to test on a new problem: How to wt. their predns?

.16 In .14: We have a bunch of subcorpi: one has all IA's worked on it. Others have subset of IA's worked on them: To decide w. mix: for each subcorpus try to find a set of cheap obs that would enable to tell which of the IA's has best pc for that sub-corpus. This need not be 100% accurate, but it would produce a sort of MIXED IA's of distinct "total pc for the various corpi".

.20 Given a set of obs, one must use "Decision trees" (& IDS) to decide between IA's for a given problem/sub-corpus.

Each "Mixed" IA would consist of a (perhaps) distinct IA for each of the subcorpi.

No! → Actually if there are only 10 obs, they would have to have mean pc of 5.1 ! ← No! only if they were Mut. exclusive — Probably not.

— So they couldn't be very bad! Well Not so! The domains of the obs themselves must be quite large; the distance between them & the decision as to whether any ob exp applies must be to a particular sub-corpus: Even these could be of low pc!

However, these "obs" will, typically be same as, or constructed of obs used in the IA's themselves — tho they are being used in a "META" way; here, it's not clear how the pc of an ob within a IA is related to the pc of that ob that is used to "control" that IA.

For Each IA, we already have obs over from them on & off — This is how the various sub-corpi have been defined! We want to sharpen these obs so that they are better in deciding which IA is able to be used for each problem: A "Decision-Tree" method would be relevant here.

My impression of "Decision Trees" is that it is not such a meta "IA", but it may be adequate for a "start" — to get TM smart enough to work on the problem level.

.37 "Decision trees" has been applied to! → 179.08, Also Note 211.21; also 214.00

(SN) In GENERAL, there is this trade-off between ① an initially smart TM. & ② the need for a larger, carefully designed TSQ. One can emphasize ① or ② → 179.00

5/11/01
IDSIA

Juergen
→ 173.22

MIN BACKTRACK

:00: 155.40: Memory Management : Store only the last r layers of the net! These correspond to the (Most recent) r bits of codes.

Maybe not: store r layers before the shortest code, ?

Or: for each of the k codes, store r layers back from the code of the entire Corpus thus far".

Use a "circular file of length r" to accommodate this info. — So address of x^{th} bit in code is like $\boxed{X \bmod r}$

As the codes get longer (As the corpus grows), we drop various data on various net nodes, & take up storage for new nodes.

The $k-1$ codes stored can't be much longer than the Shortest code, because they'd get so little wt.

T. "Original idea" was to store data on all r layers back of the current "bottom" (fig. 155.17L) layer. — using a "circular file" to reject layers as the code continued. This would work ok for one code, but integrating all of the codes into a single net seems more diff, if we store k codes in that net.

Say we store r previous nodes of each of the k codes. There will be much overlap, so much storage should be saved.

Hvr, "Garbage Collection" & assignment of already "expired" addresses to new nodes — I'm not sure about how to do this.

Also, I'm not quite sure about the Backtrack algm; even w.o. the "storage" problem. 155.17 does tell how to Backtrack minimally. For each code, one backtracks until one legal code is finally found. We can continue the backtracking on the same told code until we find more legal codes.

After some thot! A simpleminded RLP fore may ≤ 2^{-kr} — hvr, for a finite sequence & finite time, this could be very "Greedy". One might want to retain many codes that are much different from the shortest code, that are not much longer than the shortest code — for Diversity of models. — So when one backtracks, one has a greater variety of possys.

≈ SN OSS (opt seq. srch) had a method of exploring a code tree for L srch, that was "complete". I think it may have amounted to "chose a zero whenever its possl. — otherwise chose 1" — But that aspect of the srch does not immediately concern us.

5/12/01 IDSIA

06 05 12 03 Mo 06

97 98 99 00 01 02 03 — Tu Th

:00: 177.40 : (SN cont.) either way can give an "Adequate" TM. Marcus Meuthing that w. very poor ①, one can always show that "in theory" an adequate ② exists — i.e. the TSQ that tells how to construct the very smart TM.

So perhaps consider the problem of designing a "Minimal" initial TM for which the TSQ's would be reasonable & not too hard to design. Already QATM is perhaps "not BAD". I have this initial set of IA's that can be arbitrarily "clever." — So I could, in theory, put the entire desc. of a complete TM in one of the IA's.

.08: 177.37 : 177.16 — .37 seems like a reasonable soln. To the "Mixing" problem of 177.00 — .03 and 176.55 — .90 : The idea is: I have this ∑ QA } corpus: I want to code it using this set of IA's: A "particular coder" can be a Ob that looks at any specific (QA$_i$, TI) & assigns a IA to it. This results in a pc for each QA & ∴ a pc for the entire corpus. This ob is the FC) function of 175.02. ⌈Hvr, I had FC) assign pc's to each of QA's, rather than simply (Black/white) Boolean choices.

Compare the [redacted] Boolean v.s. PD ("Gray") methods of choice.

.15 The Boolean method, does provide [redacted] pc for the corpus & if we include the pc of the Ob, FC), we have a legit. pc of code for the corpus, & we want to arrange so that (the pc of FC) mult by the pc it gives the corpus) is Max.

.18 For "Gray" FC), the pc assoc. w. each Q$_j$A$_j$ will be $\sum_i P^j_{IA_i} \cdot IA_i(Q_jA_j)$?? See 211.00 & 211.10

.19 $P^j_{IA_i}$ is the pc assigned by FC) to IA$_i$ wrt problem Q$_j$A$_j$

.20 IA$_i$(Q$_j$A$_j$) is the pc assigned to correct answer A$_j$, when Question Q$_j$ is input.

So viewed as .18 — .20, it would seem that the Gray FC) is also part of a legit coder for

.22 the corpus in the fashion of .15 — .17

.24 An item missed in the "Gray" — That each IA's output for a particular Q$_i$A$_i$, depends on what previous corpus members it has coded. Usually, if a IA has done a large sub corpus, it will "smarter" & get higher pc's for [redacted] future prob solns. In general, we will want to "exercise" IA's that have been given "fairly hy" pc's by FC) — This has 2 purposes ① to "update" those pc's over large a relevant sub-corpus as possl.

② To give FC)'s update more info to evaluate efficacy of any particular

.32 IA for any particular problem. → 180.00 spec

.33 On updating FC) (the "Gray" FC)): we may want to use .18 — .22 to get the pc of the corpus wrt a given FC) ; we mult this by the pc of the FC) — this product is the Gove we want to Maxz.

.35

.36 An "Adequate" Algm for updating FC) ! We want FC) to choose the IA that gives max pc to the correct A$_j$ of each of the Q's: This is equivalent (except for factor of pc of FC) ?) to the maxzn. of .33 — .35. Also, hvr., it can be formulated as a pure Induction problem (solvable by QATM) followed by integration. (This is like very much "like" GPD, → GPD in Sol89+)

As part of the info used for induction, TM may need to know the sub-corpus & IT assoc. w. each prob of each IA.

150.00

Spec 179.32 →
.00: 179.40 ⇒ When we Give a IA a larger s.corpus than it is really most relevant to, The IA should (ideally) be able to pick out t. sub-sub corpus that is relevant to a problem that the IA is good at solving. Th. fact that t. s.corpus of a IA has "noise" in it (noise = problems that this IA is not much good at solving) is something that the IA should lrn to deal w.

179.36–.40 is then, perhaps the final nail in the whole QATM system! It expresses t. F( ) updating problem as a Z QA problem. Hvr, I have to Go over 179.36–.40 more carefully. e.g. is it nec'y to take the pc of F( ) into account (i.e. t. pc of its deriv.)?

When QATM is asked a Q, it first uses t. "best" IA only on t. problem. If there is more time, it might update that IA & give a diff't response. Later when F( ) is updated, Other IA's will be applied to t. same problem.

[5/13/01] T. problem of F( ) is To obtain in time T_1 an estimate of what pc, IA_i will have for a given QA problem, for T_2 ($T_2 \gg T_1$). If $T_1 = T_2$, then F( ) simply tries IA_i on the problem for time $T_1 = T_2$. ; If $T_2 = 10T_1$, we could try IA_i on t. problem for time T_2, but this would be a slowdown of ~ ×10 (but more accurate, i.e. we have more "accurate info" on IA_i. If we only had 10 IA's then it would probly be best not to use F( ) at all, but try each IA_i on t. problem for t. time T_2 assoc. w. that problem.

Q: perhaps I'm getting this wrong! T_2 may be time available to solve t. problem, but updating IA_i can take much longer! On t. other hand, an evaln. of a IA_i that has not been updated, is of marginal value. A possy would be to try all IA_i's for t. time given in t. problem defn, but only update t. say 10 best IA_i's for this problem.

→ More likely! What Humans Actually do: F( ) is used for a first ruff guess as to which AI_i's are relevant. Then each of t. "Relevant" ones is tried a bit, then T. estimate of which is best, is revised, & then more T is spent ... etc.

(.28) For a prelim model perhaps use only one IA: AZ141 (possibly further augmented?) It may be quite possl. to write useful TSQ's & get interesting behavior from such device. Some impt. problems: ① How much time to spend on "updating" ② What sub corpus to use for a particular problem?

For prelimy work, I can decide for TM, about ①: ② is relevant if I suddenly give TM a large batch of data (as like an encyclopedia), & it must find which parts are relevant to a given Q.

It may be possl. to design a QATM that does not initially have problem ②; That for all problems given, one wants to code t. entire corpus. Would it be possl. to train a machine of this kind (using its normal input) to work on problems like ②?

[SN] I've been thinking of QA's as being close to Questions & Answers as a Government
.35 would think of them. Hvr, as far as t. formalism I'm using, TM can learn
.39 any kind of inductive relation betw "Q" & "A". Thinking of it this way may make it
.40 easier to get QATM to work on a great variety of tasks.

5/13/01

IDSIA

SCALING: Soln to an Impl. Aspect : (.53)

.00 : 180.40! Some kinds of info "Q" can have that would help TM "learn".

1) Labeling as to the area of Science a/o Math that seems relevant: eg. Geometry, Algebra, Chemistry,

2) Labeling as an Optimization problem, "IN" as in TSP, or "OZ" or "Anytime".

(NB) → in (2), It's not clear just how we convey to TM the idea of "optimum".

I may want to look at some Cases needed for this "idea" & found TSQ's could enable these operations by TM.

If I do decide to do 180.28! write up a clear statement of just what choices were made at this BRANCH pt., so I know what to "Backtrack to", or just "Return #0" when I Recall I've gotten far enough on 180.28

.10 Look at the Buy list of 5 89 (140.00 - 141.40 i.e. What new developments occurred 1989 to 2001) (150.53)

What is (was) on "OTher IA's"?, etc.

Some Uncertainty Uncertainty about AZ141! It is an operator/function (language. What is perhaps least clear is how "updating" is done, We have a function that solves all problems up to now. We devise an NOW-function for the latest problem — Then we have to recompute for each problem type so that the new function can be in Baked when needed. In General, the Members will consist of an OB (recognition part) and a set of OPs (parameterized problem).

Also the Party, etc for non-probabilistic functs: How is the idea extended to

.21 probabilistic functions/operators? (.27)

Also go back to S 89 & see just what I was doing! (Note .10) ↳ a (141-142)

(SN In retrospect, the existence of FC) (an initial guess/approx of the soln. to problem) Was the very crucial part of my Methodology. In dropping that, am I really Reverting to the older (PURE) S 89? — look at (140-141)

.25: (.21) The Updating of Z141 is simpler because of its sequential nature. Updating AZ141 is difficult because we have to identify the class of Q's to which particular "A" (answer) class is appropriate.

Any way: Some approaches to this problem:

(1) Write out just how it seems to be done in Search TSQ's (2) Just how I seem to be doing it when I do "Algebra" problems (3) How my views on more General Q A probs than Algebra — even 180.38-.40 This may be a source of much more General Solns. than AZ141 updating problem.

.33 (SN) In "Search TM" At each Trans pt. in the search for "the function", the set of all possible sub-functions is determined by a FC) — like function of "the Situation". This FC) is periodically updated.

It may Prob which MAY be enough to Solve (one aspect of) Int. SCALING Problem

.00: 181.40: AZ141 Looking for repeated sub-trees: Not so difficult!

.01 A(α, B, β, γ) : A = B are functions: We now have NUM = 3 count of A, 2, B. function A, second argt, B.
If A commutes btw first 2 args, we count this as A, 1, B.
A(B, B, β, γ) is counted in a special way.
.01 ↑ no. of "funcs" by a factor of 2 (or 3 or however many args they have.
For funcs w. many non-symmetric args, this can be troublesome!

.09 1) Recursion as hyper order Grammar(?) (Grammar Grammar)
2) In AZ141, Given Funct HC) works up to now; Given new Q&A pair in which HC) does not work well, How to "Modify" HC) to fit new data. We can put this in a Corrupting event, & other kinds of problem events to get clues on Solns.
3) See .17

*I should "add on" could be CFG, Must be could be done if Grammar (was infinite)'s a CFG.

.13 I Guess T. Major problem now is Update of AZ141 (≈ Saarb Corpus)
see 181.21, 182.09

.17 Re: .09, .11: "A Curve fitting event". Q is a number, k; A is H(k); we don't know HC).
We have any set of k, H(k) (= H(k)) pairs, to try to approximate to HC).
If we approximate HC) by $\sum_{i=1}^{n} a_i f_i(x)$, Ran as we ↑ ss2 (≡ ↑ Size of Corpus)
we modify to as ↑ occasionally ↑ n.

→ More Generally, It's the problem of Operator Induction; I'm working on 180.28;
I'm assuming all of corpus is relevant.

.25 (SN) On "corrections" to the output of a stoch op. model.
Correctness
T. output of the stoch op. can be in 2 parts: one, directly from the OP &
two, the correction codes. If most output has no correctness the pc of a correction will be low.
Alternation (1 type output), a 0 or 1 is expected. "0" means no correct, "1" means
correction. If corrections are wrong, 0 has pc near 1 & 1 has low pc.

.32 SHOW: On updating Saarbrücken (ANL); Watch carefully how I work
those problems. When a "New" problem comes in — what parts do I immediately
recognize as "New"? — so I can recognize this new type of problem?
I really need to know the "OB" part of my problem soln.!
Note also, that I must design the System so it is able to Learn these kinds of discovery
Techniques from a "reasonable" TSQ.

183.00

.00 : 182.90 : OK so, say TM is able to do a + b → ~~out~~ (a, b, sum) [above "a + b": input symbols; above "sum": computed code]

we have this pgm : H(3, +, 7) → 10.

.02 we have a new problem "8, −, 3" : (we want to minimally modify H : but the real problem is to [margin note: one way to do ?]

obtain a small pgm that does both 3+7 & 8−3.

One way is to find a small pgm for "3+7"

from " " " " "8−3" & somehow combine them — a standard

way is to use an "OB" Part in disjunction [reliably] between "3+7" & "8−3" — then use

this OB to control which pgm is used. Later, we can further compress the resulting

pgm because the pgms for 3+7 & 8−3 are very similar.

T. fuzzy, would probably work OK. : Another (apparently) OK direction is (.02)

"Min modifn" usually means if PH is a short (PBM) code for H( ), we want

min modifn of PH : The meaning of "min modifn" depends critically on the reference machine —

So the "reference machine" must be periodically "updated" (modified in view of data —

— up-to-now.) One defect of Mutation as a GA strategy is that the Mutations

are always static. [redacted] (usually) mutations work on the code for an object

rather than the objects themselves, (which is o.k.) — Also, the form of the mutation

should be such that increases in G (fitness) are more likely. Similarly, the details of

crossover should also change "Adaptively".

In alg. complexity terms, "min modifn" means: P_x is a short pgm for x.

M_r is a 2 input UMC: $M_r(P_x, s)$ will give small modifns of x if |s| is small).

perhaps $M_r(P_x, \Lambda) = x$. Or y is close to x if

$ALP(x,y) \approx ALP(x)$. Like $P(y|x) \cdot P(x) = P(x,y)$ -

so $P(y|x) = \frac{P(x,y)}{P(x)}$. y is close to x if $P(y|x)$ is close to 1.

the object x, y is uniquely decomposable into x, y.

I'm a bit confused about $K(y|x)$! x is original object; y is slightly augmented object

we have a short code for x ; we want a short code for y. (Short code for y).

T problem is made more difficult if we have a stoch operator for a sub corpus

[QA] & we add to it, & we want to get a short code for the augmented corpus.

5/16/01 Perhaps Write Good Review of the update problem : Just what I know about

soln. — what is [redacted] unsolved? Make list of solns & possible directions

to go to find solns.

IDSIA:

Summary of DMPT ideas

.00: 183.40 ○ One more idea about TSP design: That if the % of the soln. .01 of a Env. problem is (p) it takes time T to recognize & verify an acceptable soln, then Lsrch takes $\in \frac{T}{p}$

○ Another big idea was the "2 part soln" of problems:

.04 1) F( ) looks at problem, assigns #'s to possible solns. (F() is a cond'l pd.)

2) Lsrch over possible solns.

.07 ——— That updating F( ) is a kind of problem for system can work on/solve.

.07 proved to be more difficult than anticipated!

That QA problem, If I have an operator for the [QA] corpus and I add a new Q, A, I have ideas on how to solve it incremental [QA] corpus, but may be seen a bit A.M. (i.e. 182.13 182.32

○ Another big idea (perhaps close to .04)! That a tsp. is a way of the p (of .01) so that the problem is practically solvable by Lsrch. —

○ That somehow, (a (II) search heuristics could also be discovered by suitable tsp's

.19 { (b) could be expressed as modification of p (of .01) for Lsrch. b may not be exactly true, since it seems to require modification of p for any Lsrch. — but anything can be regarded as "a way to do a srch" — although can be a cond in Lsrch, so "b" (.19) would be exactly true.

This is SS9 FN2

ON BACKTRACKING: In actual science (say physics), when a "backtrack" occurs, it is almost never necessary to apply the new law to all of the old cases. We know the old law was within ε% of correct for all old data! The new law has to be within ε% of old law for all those cases which can be "tested" Analytically.

.31 (2) I had that idea about "How backtracking/updating" can other modes of backtracking/updating be put into that form? (Hm, note that D does sum to a problem with the idea — I don't have a good overall sense. (other than $\sum 2^{-d} = \infty$.)

Some Q is about (.38) (1) How many codes to save (2) How far back to backtrack (Answer by considerations of Accuracy & Consistency) for each of the "falsified" codes? (3) Should I return some low PC codes for "diversity's sake"? (4) What "Time out" criterion to use for each "backtrack" attempt.

On (2): If the shortest code is of length L, backtrack to length L − α, where α is some constant.

Q: When backtracking, should not the distance one goes back depend on how (185.00)

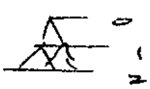
5/17/01

IDSIA.

- .001: 184,90 : longest outputs from a (partial) code? Ok: say the best code has length L, the corpus has length C^r: $C^r_z - L = C_m$: C_m is the compression obtained by best code.
Perhaps examine backtrack branches only if $C_i - L_i > C_m - $ some α.
[redacted] L_i is (partial) length of i^{th} backtrack; C_i is length of (longest) corpus coded by L_i.
α is how much worse than "Best" we will tolerate.

Q's about Storage Algm. How much to save? How deep in past to save?

.08 So, If the shortest code is of length L, then all codes can backtrack to length $L - \alpha$ at most. Codes in general shouldn't be much longer than L — say $L+5$. So we save at most $5 + \alpha$ layers for each code. We save all info from $L - \alpha$ length & up. We save the entire graph of codes for layers $L - \alpha$ & lower.



the Layer no. = code length.

- For a long corpus, the 100 best codes will usually have nothing in common within the last 10 bits. — or from the "longest code" to the "shortest − 5" will tend to have little in common — so we might as well save data on the 100 best codes separately (woops! maybe not! In backtracking, we don't want to end up w. "new" codes that we already have! — A common tree storage could prevent this. —— So this is a part that I should work out.

Hvr, if 2 codes have nothing in common past $L - \alpha$, then they further descendants of these codes will have nothing in common past $L' - \alpha$ ($L' > L$ & is future length of shortest code —— woops! L can ↓ w. time, but usually doesn't ——
If it does, then perhaps its ok. to just backtrack everything & start from the new "shortest code" $L \to L'$; $L - \alpha \to L' - \alpha$.

So assume that L is monotone ↑ in time. If it does ↓, then, say, by 2, then let α ↓ by 2 also (for a while; only until L is above its original size)

Perhaps .08 solves the main problem.

Say we save 100 best codes: 2 or more codes will be "on same subtree" If they have a common node at level $> L - \alpha$; If there is choice of which 100 codes to save, pick those which are in different subtrees. So also ↑ diversity.

- In searching the code, when "Timeout occurs", simply mark that pt as "untested" & go on to next final node on the net. Untested Timeout codes will (usually) be tried w. twice as much Timeout threshold next time, so we will not waste much by reprocessing this way.

(186.00)

.00 18:40 : So, it looks like t. search will be very easy pgm, — just "keep to t. left node if possl."

.02 I may have to work out Storage, but storing only (L−α pts) could be adequate for a demo of a first approx soln. — we retain "100" best codes

Storage : Not needed when moving down in tree — (this is normal computation) But when Backtracking (Going up on tree) storage is needed.

Try Z141 as "t. U10 machine!" See whether this "Backtrack Algm" is at all comparable in speed to t. usual use of Z141. If not, perhaps Modify t. Backtrack Algm

.02 may not be so bad, even for a final system! In fact, it does much simplify t. system; When a GRAND backtrack is called for, we do ≤ 100 "restarts" from node L−α. We then simply do a "keep to t. Left if possl." search for all possl. continuations, until we get w 100 codes for t. corpus. T. Q is, do we need to retain all the info in a address tree structure? — Do we need t. address tree structure at all? Does it serve any useful approach here?

If L, t. length of t. shortest code for t. corpus is by 1 (L → L+1), then L−α → L−α+1, and t. Timeout limit on all t. old nodes doubles — so rerunning those old branches is worth while since the time out limit is doubled for each node.

N.B. : As I'm thinking of it now, t. timeout limit, T is for t. execution of 1 bit of code — indep of whether it is producing output. t. "one bit" is over, when t. machine asks for another bit. If it asks for > α + "5" bits without producing t. entire corpus, we have a failure at that pt.

I have been thinking in terms of "1 bit of t. corpus at a time"! So we increment t. corpus by 1 bit, then try to continue all 10 codes — backtracking when necc, This may be very inefficient!

A more General Criticism is that t. Pgm is not "Min Backtracking" at all, Its backtracking to ... (L−α)th node in all cases. It would seem to be more likely to get short codes, but it would be much more time consuming than "Min backtrack" — (Tho I'm not at all clear on t. details of "Min backtrack"!

.39 ((— Perhaps it was wrong that we go back as little as possl. to obtain a branch of t. code that will realize t. corpus

.00 ! 186.40 ! One diffy of 186.35 ! That it is "pure backtracking" & it requires complete storage of states (or ? ?). One could go thru same old paths, but at double "Timeout".

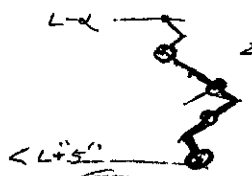
.02 T. way I'm thinking about it; One back tracks by n & looks at all nodes until no codes for

.03 entire corpus are found ; n → n+1, loop to (γ). [start w. n = 1]

This .02 – .03 works conceptually, but the storage of states, remains a problem.

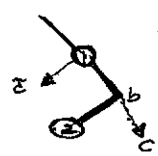
Re: storage of states, etc.

Consider a particular code: → Every several nodes along the code, we store the state or a set of "differences" from a previously / completely stored state.



Say we stored every third node: Then we could just back track to those pts only. We first [redacted]

Start at the beginning & develop down, using suitable TOL (TOL = Time out limits). Presumably the TOL is at least twice TOL used when first developing that region.



Say we back track to pt ① (States are stored at ① & at ②). If we run more logical trials in direction "a", we do so, making a complete search out to L+"5". Then we go to b & see if we can develop c out to L+"5". Both of these developments need not involve remembering anything. — one just develops by taking the left hand branch whenever poss'l.

3:35 P

1.40 / .78 53 !

for "Min" back tracking: first back track 1 on all "100" codes then
" " 2 " "
" " 3 " " etc, until we have a total of

100 [redacted] codes — Or, perhaps Generate > 100 codes but save only the "Best 100"

.27 ("Best" need not be shortest, but "diversity" can be a consideration.)

So: Essentially 2 Methods of BACKTRACKING ! 185.08 – .40

.29 1) Go back to layer L−α at each of the 100 codes, & try to code the corpus, using TOL's that are at least 2× those used in the original search for those codes. No memory is needed: Just "Choose zero branch whenever poss'l.

Q: Are there signif. shortcuts, time savings poss'l.

.33 2) 187.02 – .27 : We backtrack by 1 bit on all "100" codes, then by 2, then 3, etc. until we have "enuf" codes.

(188.00)

5/18/01
IDSIA

Trs 710 750
Col 10

.00: 187.40 : in ① (187.29) There is some diffy in exploring various continuations of the "state of level L − α". Much has to be "remembered"; unless one does linear Lsrch, try 0, 1, 00, 01, etc ~~......~~ not storing states. One could use what corresponds to "|| Lsrch": || Lsrch for codes of a corpus involving potentially, an ~~...~~ of all codes.

It was my impression that || Lsrch was often considerably better than T⇐2T Lsrch — something like a factor ∝ "time of layers of some function" — I don't remember much about the demonstration, hvr.
└ "No. of layers" was length of code → see (.23)

In ② (187.33) There was a diffnt kind of backtracking. The || Lsrch technique seems inappropriate — But if we want to store a very large no. of states, we may be able to do it like I thought for ||Lsrch! ~~...~~ e.g. Use ½ of RAM to store all ~~.....~~ recent (below L−α) states of the code we're working on. ~~.....~~ All states of other "100" codes are on disk: while we work on ½ of RAM, other ½ is being ~~..........~~ stored & reloaded w. states from a difnt code.

[Or, just have 2 cpu's, each w. its own memory, but use common Hard Disc Drive. —
(on the Linux system for 2 cpu's do that?) : well, w. 2 cpus, using T⇐2T, we should be able to double speed to that of || ~~...~~ Lsrch!

— So the main Q is : is || Lsrch actually much faster than 2× ! (T⇐2T?)

.17 ~~...~~ Rite now, it seems that || Lsrch is ~~......~~ ≈ $2\times\sqrt{2} = \sqrt{8}$ times as fast as T⇐2T Lsrch. (T⇐3T is slitely faster than T⇐2T). That the 2 methods do pretty much the same thing — but T⇐2T does it twice.

Well! .17 is wrong! T⇐2T is inefficient in a way that may be much worse than the "doubling" i.e. To do 0110**0** and 0110**1**, it actually ~~does~~ both codes, one after the other. (executes)
||.Lsrch does 0110, saves the state, then continues w. ∅, then continues from that state, w. 1.

(.23) Say, in Lsrch, we are looking at all ~~...~~ codes of length ≤ n.
In T⇐2T search, we execute $n \times 2^n$ bits of code, twice; so ~~.....~~ $n \times 2 \cdot 2^n$
In || Lsrch " " $2^n + 2^{n-1} \cdots = 2^{n+1} - 1 \approx 2 \cdot 2^n$: so n times as fast. — is this a lower Bnd ?
$= (1+2+2^2+\cdots 2^n = 2^{n+1}-1)$
This is all very approximate, since difnt bits require difnt time.
If ~~.....~~ the time for a bit is ∝ its pc, then

No! T⇐2T Lsrch requires time $(2^n + 2^{n-1} \ldots) \times 2^n$ ~~...~~ $(2^{n+1}-1)\,2n \approx 2 \times 2^{2n} = 2 \cdot 2^n \cdot 2^n$ | w. the 2 = $4 \cdot 2^n \cdot 2^n$
Parallel " " " $= 2^n + 2\times 2^{n-1} + 4 \times 2^{n-2} \ldots = n \cdot 2^n$
So 2^{n+2} v.s. n ! or $\boxed{\frac{2^{n+2}}{n}}$ times as fast — (I think this is upper Bnd)

But || Lsrch needs much more memory. Say 2^n times as much — But it can be disk memory.

.23 → A code of length ℓ, requires a total time of $\leq T \cdot 2^{-\ell}$

→ 192.08

I think the Backtrack problem is this: Say we have these "100" codes; say they have nothing in common for ~~...~~ ~~......~~ > c; (c << L−α): How far to backtrack & what L cost to search at? (or what L cost limit to search for?).

One way to think about it; ~~....~~ The known codes have been investigated w. certain L cost bounds! So, in backtracking, we must, say, double those bounds in re-searching.

→ 189.00

IDSIA

(To 207, 180)

.00 : 188.40 : Another way (perhaps related) : When backtracking, we should have record of occurrences "along the code" : which alternative bits were tried/not tried ; Of those tried : Did they fail "absolutely" or fail at some Lcost level (≡ Time out Limit (TOL)) .
This could tell us promising directions to try.

So, one could go back 5 bits & double the Lcost limit.
We already have (probably) diff'rnt Lcosts of the "100" codes. → The Lcost of the shortest code is the lowest.

Another approach! ~~Say our corpus when starts w/ length~~ 15 bits ; we do Lsrch until we get 100 / best codes. α To continue the corpus (say 5 bits more), we use only codes "based on" the "100 best" —— This means we take each of the 100 codes & we consider all sprouts from them that have timed out, & we double the time on those "sprouts". We again pick the "100" best that continue us the desired bits, Then we loop to α

Somehow, this "doubling" of the TOC is not so bad because we somehow have fewer
.14 nodes to try so $\sum p_i << 1$. —— we m'ht do the srch by actual time sharing, & use
.15 the same nominal p_i's as before.

So, in General .14 — .15 sounds good : we could just do a continued Lsrch on the tree, & have some criterion for continually pruning the tree.

At each time pt, each node will either be dead (incorrect output) or not.
If not it will have an assoc. no. that tells how many bits of the true corpus it has coded thus far.
T. total time spent on a node, depends only on its "Depth" (depth = o > L)
Perhaps rate of work on a node should be $2^{(-\text{depth}+\text{length it has coded corpus})}$
well, the/a/h, code, depth & length of corpus : ~~[illegible]~~ (corpuslength − depth) = compression
$\sum 2^{-\text{depth}_i} < 1$, but any bnd on $\sum 2^{(\text{corpus length}_i - \text{depth}_i)}$?
C_i = corpus length$_i$; D_i = Depth$_i$: There aren't many short codes so — converge.

.27 Hvr, say $C_0 - D_0$ is ~5 for some code, & then that code doesn't halt ever.
We keep on putting 2^5 wt. on that trial ! perhaps make Time Share wt. ∝
.29 — $2^{C_i - D_i} \cdot f(C_i)$: F m↑ funct of C_i —— so that's those w/ are doing actual coding could eventually get a reasonable fraction of the wt. — In fact, ~~[illegible]~~
((In fact if f is any ↑ funct, no matter how slow (it should ↑ → ∞ as $C_i \to \infty$)
it will "work" (or at least eventually get rid of the non-halting 2^5.

. 5/20/01 Hvr, consider the "copy" function : $R_0 = C_0$ (no compression) — say it is always one of the codes.
If it is fast — we should always include it in the set of codes we are considering. → Hvr, the "copy" code especially takes little time & completes the corpus quickly, so we spend no more time on it!
In .29 $2^{C_i - D_i} \cdot f(C_i)$: for the identity codes $C_i = D_i$; C_i will usually be > that of any other codes so $f(C_i)$ will eventually dominate, so it looks like .29 by itself is a bad idea.
If we used $2^{C_i - D_i}$ as the only wt., then we run into trouble of .27 : we may accept this trouble ; The value of a very compressive code would be very large, if it worked, so one may want to risk on the possibility of wasting a lot of cc on it.

IDSIA

:00 : 189.40 ; The idea I had on how to do this coding! That I would use "ideal" II Lsrch w. time share; That the fraction of avail. cc spent on each code would be some function of $c_i - D_i$ and c_i. — That this "weighting function" would somehow move along as the c_i's increased. How to get this "Moving along" is unclear — I'd want velocity to be about constant — so it would not slow down because of large no. of codes w. small lengths $S_i - D_i$, that did not "halt".

We could characterize run by its velocity: we pick a velocity for the run & stick to it. A possible use of this: Only work on codes for which $c_i > kt$; where t is real time. If we spent cc on codes $\propto 2^{c_i - L_i}$ would we get into trouble as $c_i \to \infty$? I.e. would the total of $2^{c_i - L_i}$ diverge? It probably depends on k.

If k is small (< 20, say) would $\Sigma 2^{c_i - L_i}$ diverge?

For fixed c, $\Sigma 2^{c_i - L_i}$ will be $2^{c_i} \Sigma 2^{-L_i}$ — which converges.

— But if c_i can have many values, I don't see that $\Sigma 2^{c_i - L_i}$ need converge.

.15 [Another: There is a way to "effectively fix c", so $\Sigma 2^{c_i - L_i}$ is kept about constant) We start at codes of max c_i, then we work backwards until $\Sigma 2^{c_i - L_i} \geq$ a constant we choose $= \Gamma$.

.17 By choosing large Γ, our search is slower but "better". (.33)

SN On time-shared II Lsrch! I had considered using a big disc to store status: There were 2 indep (RAM) memories; while CPU was working on one ½ RAM, the other half was being swapped w. disc.

This could be done easily w. a dual CPU machine, but it costs a factor of 2.

(.22) Another way! w. normal computer! Use very large RAM & do swapping w. disc, then computing in RAM, then swap w. disc, then work on new RAM, etc.

If time for swapping is << time to do compn, then everything is OK. — we lose ≈ no time. There are some wide SCSI discs that [illegible] swap very fast but I don't know how fast, & I don't know how long compns in RAM take: Hvr, I could always ↑ size of T (one spends $T \cdot 2^{S_i - L_i}$ time on each code.) — Total time spent on RAM is ∝ this T, but too large a T means it's much less like "II Lsrch". — At any rate this (.22) looks Very promising!

.33 (.17) If too much time is spent on codes of large c_i, but larger L_i, perhaps we use $T \cdot 2^{(c_i - L_i)\gamma}$, where $\gamma > 1$. But look into this problem more carefully! → see 191.12 — it is not clearly clear when this applies.

*When we come to "end of corpus" (more or less) Use a lot of extra cc than start — this closes of many loops.

At end, make a plot of no. of codes that terminated in time T.

We may be able to impose a < ∞ stop rule.

.39 Also, (particularly for codes w. large c_i) — reject codes w. $L_i > c_i$ or $L_i > c_i + 3$ or 4 +5 or 4 or 3

.40 But for some cases, we will be coding fairly well so $L_i \approx c_i \cdot \alpha$ ($\alpha < 1$, say $\alpha = .5$) ... perhaps

(151.00)

.00: 190.90 : Some single regys. What to do then? — See remark of (.12)
Perhaps a better compression criterion would be $\frac{L_i - C_i}{C_i} = \frac{L_i}{C_i} - 1$, or $1 - \frac{L_i}{C_i}$.

$1 - \frac{L_i}{C_i} = \frac{C_i - L_i}{C_i}$ is the compression ratio: which is expected to be ≈ constant for large sub corpi.

$\frac{C_i - L_i}{C_i}$ ... having "cc wt." ∝ this factor might be good. — We only mutate codes w. C_i ≥ a certain threshold, determined by ≈ 190.15.

For $\frac{C_i - L_i}{C_i} = \phi$ we may have a fair no. of codes. As $\frac{C_i - L_i}{C_i}$ gets < 0, I think the no. of codes rank a lot.

No. of codes might be $\sim 2^{L_i - C_i}$

The problem is how much cc to give to each code as a function of C_i & L_i.

As an interesting study: There will be at least $2^{L_i - C_i}$ "copy" codes for any corpus fragment of C_i bits; using L_i bits of code. We don't want to give codes w. $L_i - C_i > \phi$ much wt.

(.12a) ● Re: Remarks of 190.33 & .39 & .40 : It's not clear that wt. of $2^{C_i - L_i}$ would be inadequate.

— That Q of 190.40 – 191.00 (on $L_i \approx aC_i$, $a < 1$ say $a = .5$) is unclear with

Say $L_1 = 5$, $C_1 = 10$; $L_2 = 10$, $C_2 = 15$ } $C - L = 5$ in both cases. — yet 5,10 would seem to be better since it has more compression/symbol.

One could cover $L_1\ C_1$ to $L_2\ C_2$ by copying 5 bits. — So in this sense the 2 codes are of equal value,
But L_1, C_1 gives one the idea that 5 bits of additional corpus could probably be coded by only 2½ bits of code rather than 5.
So 2 sources of Goodness: $C_i - L_i$ big desired | $\frac{C_i - L_i}{C_i}$ big desired.

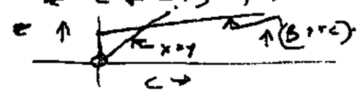
$\frac{C_i - L_i}{C_i^{\gamma}}$: $\gamma < 1$. would mix these 2 goals.

If I used $2^{C_i - L_i}$ only, codes of small C_i automatically get deleted in 190.15 – .17
— so in this sense, big S_i codes are "selected for".

So: a Q is it, $2^{S_i - L_i}$ is used as wt. in 190.15 – .17 will it work? will it scale? —
I.e. as the corpus gets larger, will we still have about the same no. of codes w. high wt. as when corpus was smaller?

● Note: I doubt if one could do much by trial & error psms w.o. a clear understanding of what's going on: A reference machine has to be chosen so it can find the regys in the corpus w. reasonable pc.! Also, if the kernel of the regy in the corpus takes B bits, and the regy gives compression ratio r (bits) of (r ≥ 1), then best code for corpus of length C is B + rC.

This is an important idea, if we want to try simulation



So, we for a short corpus, L will be > C. — i.e. we have to discover the regy by random search — only then do is wanted ΔL < ΔC

Another possy is that the regys in short corpus are not discovered all at once, but ≈ "one at a time"

193.00

5/21/01

IDS 18

:00: 158.53 Lsrch : T←2T v.s. L Lsrch | worst case : I. worst case is ...

* For random corpus No shorter codes found : Each bit takes 1 sec. so do :

for T ← 2T : T=1, L=1 done. (L = length of codes = C = length of corpus)

T = 2 · 2^-2 ... I sec. for each code only,

T = 4 · 2^-2 = 2

$n < T \cdot 2^{R} \cdot 2^{-n}$ ← Rounds ... $n 2^n < 2R$

So, really: for corpus of length n, T ← 2T takes ...

since $1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 \cdots n 2^n = \sum_1^n i x^i = x \sum_1^n i x^{i-1} = x \frac{d}{dx} \sum_0^n x^i$

$= x \frac{d}{dx}\left(\frac{x^{n+1}-1}{x-1}\right) = x\left(\frac{(n+1)x^n}{x-1} + \frac{x^{n+1}-1}{(x-1)^2} \times (-1)\right)$

$= x \frac{(n+1)x^n(x-1) - x^{n+1} + 1}{(x-1)^2} = x \frac{x^{n+1} - x^n - x^{n+1} + 1}{(x-1)^2} = \frac{1-x^n}{(1-x)^2} \cdot x$

for x = 2 this is $\frac{1-2^n}{} \cdot 2$

$x \frac{(n+1) x^n (x-1) - x^{n+1} + 1}{(x-1)^2} =$ $\quad 1 + x + x^2 \cdots x^n = \frac{x^{n+1}-1}{x-1}$

$x \frac{(n+1)x^{n+1} - (n+1)x^n - x^{n+1} + 1}{(x-1)^2} = \frac{n x^{n+1} - (n+1)x^n + 1}{(x-1)^2} \cdot x$

for x = 2 $\quad (n 2^{n+1} - (n+1)2^n + 1) \cdot 2 = ((n \cdot 2 - (n+1)) 2^n + 1) 2$

.18 $\quad 2n - n - 1 =$ $\quad \boxed{\sum_{i=1}^n i 2^i} = 2(n-1) \cdot 2^n = \boxed{(n-1) 2^{n+1}}$

N = 10 : x = 0

For J = 1 to N : X = X + J * 2^J : Next

Print X $\quad$ 20 : x = 3.984589 E +7.

Print (N-1) * 2^(N+1) $\quad$ works for 10 : X = 18432

formula was To check numerically — was final ≥

(margin: I Read Parity, Video Socs)

.23 For parallel Lsrch: $1 + 2 + 2^2 + \cdots 2^n = \boxed{2^{n+1} - 1}$ $\quad \frac{(n-1) \cdot 2^{n+1}}{2^{n+1}-1} \approx n-1$ if n >> 1.

$\frac{T_{T \leftarrow 2T}}{T_{||}} = (n-1)\frac{2^{n+1}}{2^{n+1}-1} = (n-1)\frac{1}{1-2^{-n-1}} \approx (n-1)(1+2^{-n-1})$

Maybe not! for T ← 2T ! start w/ T=1 : we found we allow $T \cdot 2^{-1} = \frac{1}{2}$ ... trials;

no trials done! T=2; Trials of length 1 allows $2T \cdot 2^{-1} = 2T$: so all are done.

T=4; Trials of length 2 given time $4 \cdot 2^{-2} = 1$ ∴ not done. Trials of length 1 are repeated.

T=8 " " 2 " " $8 \cdot 2^{-2} = 2$ done

T=16 " " 3 " " $16 \cdot 2^{-3} = 2$ not done!

T=32 " " 3 " " $32 \cdot 2^{-3} = 4$ ∴ done

$R \geq L + \log_2 L$ $\quad T = T \cdot 2^R$

$1 \cdot 2^R \geq L \cdot 2^{-L}$ (← = T)

$L \leq 2^R \cdot 2^{-L}$

$L 2^L \leq 2^R$

$\log_2 L + L \leq R$

L	R
1	1
2	3
3	5
4	6
5	8

5/25/01

IPSa

Sun 8:15p start [redacted]
8:30 p.m.

191.40 :
:00: T. hopes that using t. forgoing ideas (or any other ( )) that t corpus has a dom. that is discoverable in 3 parts: T_i, P_i bits, (i=1/3) ; T_i, P_i bitdom (i=1/3)
That finding it will take time $\sum_{i=1}^{3} (T_i \cdot 2^{P_i})$ rather than time $(\sum T_i) \cdot 2^{(\sum P_i)}$.
→ $\sum_{i=1}^{3} (T_i \cdot 2^{P_i})$

— That t. regys are discoverable sequencially. Most programs in Science (almost)?
require this for feasibility, This is a key of t. concs. to be discovered.

Can coding, discovery, of t. value of a continuous parameter (like, say, t. P_i's of aBorm. D.F.) be done sequencially? — I think so! but perhaps w. less efficiency than if it were done "All at once"!

If there are 3 regys of Lcosts LC_i (i=1/3) then I think its most efficient to derive t. one of smallest Lcost ($\sum T_i 2^{P_i}$) first. (?) — Tho w. my most recent (2011) stch method, I would be deriving all 3 of them in parallel codes, ≈ simultly! — Unless, t. regys had some "natural order" in which they could be derived — say using abss. that "build upon" abss used in previous discoveries. → 195.05

.18

An Apparent Diffy w. t. forgo. system/analysis: T. Search is not "Heuristic" at all — A human using human Heurs would do Much Better!

IS There Any Way I could get Heurs into t. System?

Well: Do I want any heurs in it! — T. system was meant to be one of many

- IA's (Induction Algms) used by a TM system. The heurs of t. system are else where in t. System.

But Anyway: W.o. heurs it does have t. possy of having a lot of "Diversity" & ∴ w. enuf cc, it could get very "Orignal"/"creative" Solns. to problems. — (Like certain Kinds of GA)

T. system is of interest! — It has several interesting properties:

1) T. idea of "Backtracking" (t. 3 variations on this theme) do correspond to "Theory Revision" in Science.

2) The ideas of (193.00–.18) on concs, combinations of concs, sequences of concs. — is relevant to normal "Scientific Discovery".

3) Just why it doesn't have heurs, & other scientific discovery Systems do, is of interest. This (present) system doesn't seem to have any "memory" of what was successful in t. past, & what was not

IDSIA

8:49 to 9:18 P end of loud music (SOS) ← 30 min

.00 : 193.40 : Well, Z141 didn't have heurs; nor did AZ141! Hvr, t. system in which they were imbedded, was able to use heurs. Simple Lsrch was able to use 'some' kinds of heurs, & if we allowed updating of t. Pd during Lsrch even more kinds of heurs were poss!.
[At t. present moment, I don't remember just how far I got in being able to implement heurs in various modifns of t. "S89+" system.]

.06 I think t. system looked like this: Problem "X" comes into system!
F(X) looks at X; outputs a p.d. on Algms that should be good in solving X.
We did an Lsrch on these Algms working on X. During t. Lsrch, t. algm that updates t. PD,

.09 watches what's going on & modifies t. PD's of t. Lsrch (so it's no longer Lsrch but can be

.10 (in a sense that I've defined) ≅ "optimum".) [redacted]

Furthermore, t. updating algm not only wts Algms for solving X, it can create & wt. new

.12 algms for solving X. — Based on [illegible] info in .09. THE "Heurs" are in F(X) & its update Algm.

Well; in .06 – .12 — where do Z141 & AZ141 enter? Well, t. "Algms" for solving X are of t. forms in AZ141 — i.e. functions.
T. language of any Heur I can think of can be phrased in terms of a PD on searching over Algms or searching over final solns for X. (AZ141 can give a pc for any function I can descb.)

.18 → AZ141 gives pd on functs: Given a Bag of functions, AZ141 can induce a grammar.
[While .18 is true, t. mechanics of how it is done I'm not sure of t. "details" ☹]

SN₁ Juergen's interest in t. learning of t. language of binary seq. extrapoln — T. method I have for coding an arby ume. & was/is perhaps relevant to recurrent neural nets!

SN₂ One reason t. technique isn't of much value as is, is that ("every") Lsrch is

.24 not of much value, unless all problems are in a suitable sequential form. (now Hatch shell. end of Red bridge.)
A more useful form of "ume" would be a 2 input ume: One input describes problem. — other input is random; output is pd on soln (or pdm for soln?) of t. problem.

.27 5/27/01 Condition (.24) is rarely (if ever!) satisfied. One way would be to have t. problems be given sequentially, w. markers betw. them ("T. poor man's BAG induction"). "Poor MAN's BAG induction"

SN₃ On coding a sequential corpus w. limited backtrack: Ideally, when a new chunk of corpus comes in, I'd like to be able to recode t. entire augmented Corpus. W. "limited backtrack" I can only recode a small part of it. This would seem to limit t. kinds of induction codes I could use.
Hvr. in t. original Z141, there was not t. trouble of this kind!
For continuous params, I may have proved that if one changed params as one went along t. corpus, or one simply coded t. corpus as a whole: — T. results would be t. same, in terms of total coding cost (ΣPC) of t. corpus. — So in this (continuous) case, "massive Backtrack" is unnecy.

≡ IS there an analogous situation for Discrete cases?

My impression is that in t. continuous case, as t. corpus grows, t. code grows monotonically, w.o. backtrack. Hvr, t. way I think of it, t. Z141 coding "Hyly Parallel" — Any single code would give an approximation of that of t. corpus.
On t. other hand, there may be a simple way to convert any set of binary codes into a single, shorter, binary code, using Arithmetic Coding.

195.00

5/27/01

IDSIA.

10 PM 5/30/01
"B"

7:46 p start music.

+(Binary)+

.00: 194.40 If we have known prefix set w. assoc. pc for each member — can we associate a "pc" w. any subset of that prefix set, in a reversible way? I don't think so!
There may be a way, but it take a fixed add a tiver (at least) code to do this. We translate the pc's obtained by the II codes into a machine that gives, say a code corresponding to $L = \lceil -\log_2(pc) \rceil$ — i.e. the lengthwise code is [illegible].

.05: 193.18: A possl. good feature of the $2^{C_i - L_i}$ wt. — If simple "identity" codes are devised, they will code very fast, so very quickly, C = length of corpus & one will spend no more time on them.

.09: Back to 194.27 ff! Now say one has a (binary) Bern seq. Is there any way to code it sequentially, so that the pc's are always (*) updated?
[Even if one is given the "p" param a priori — how does one sequentially code the corpus? — well: Arithmetic Coding would work.]
well, in sequential coding, we can still use arithmetic coding, provided that codes are obtained, by (say) "Laplace's Rule."
The mechanics of "Arith coding" can be a bit complex, but it's a fixed pgm (a priori), so it can have zero cost — the only real cost is the calculation of the pc's for 0 & 1.
If each "code" is able (by defn) to compute pc's of each possl. binary seq. (actually this is only "part" of a "code"), then all we want is "codes" that assign high pc to the actual corpus. — "Arith coding" becomes an ordinary Artifact!
A way to defn. "pcms" so they give pc's as output: Give several params, & the pc's are functs of these params: Then, as the corpus is scanned, give rules for updating the several params. This would be a complete defn of the PGM!:
E.g. A Bern seq. w. Alphabet of k symbols: the k params are $N_1, N_2 \cdots N_k$; the k case counts for each symbol. When the ith symbol occurs, $N_i \leftarrow N_i + 1$; then calculate pc's via "Lap's rule" (generalized to an alphabet of k).
To do the [illegible] for Z141 would not be trivial! Perhaps parallel codes would correspond to alternative parsings. But what about New Definitions & the needed "reparsing"?
One way to do this is to realize one's only interested in getting good (probably not) models for the entire corpus up to now. — "Good model" includes of course, the pc of the model as well as the pc of the corpus w.r.t. the model.
In Z141, "the defn of the pgm" consists of the pre corpus + (At least!) Theoretically, all possl. parsings are assumed to be considered "ML" in II.
Comparing Z141 to "Min Backtrack types of coding": In both, we try to get a short dcm (≡ hypo) of the known corpus. The 2 methods vary in the ways they do "Backtracking".
Z141 can backtrack by (a) reparsing (b) new defns: — note the alternation of (a) & (b). This may give rise to the equivalent (or actual) multiple parsings (≡ codings) for the corpus.
In M.B.: Backtracking is limited to recoding of "recent" portions of the corpus.
In Z141, the making of new definitions makes it nec'y to recode the entire corpus.
But in Z141, this is not hard to do.
In M.B, recoding the entire corpus would be hard to do.
In comparing Z141 & M.B — perhaps "Backtracking" has diff't meaning.
In M.B. I'm thinking of "Theory Revision" in the sci community.

198.00

IDSIA

Lsrch to OZ problems

.00 : : T. method I devd in OSS: Rev 1985!

~~Say p_i is the pc of the ith algm A_i, that~~

.02 $p_i = M^*(\alpha_i, M, T)$. We only consider pgms that take time $\leq T 2^{-l(\alpha_i)}$ to generate p_i is test M(p_i)

In E time T we can generate & test all pgms p_i ∋ generating and testing them takes $< T \cdot 2^{-l(\alpha_i)}$

.05 We find one w. max M(p_i) of all of these.

$A(M, T \cdot 2^{-l(\beta)}) = M^*(\beta, M, T)$: β is shortest pgm ∋ $M^*(\beta, M, T) = A(M, T \cdot 2^{-l(\beta)})$

Q: do there exist any β ∋ ↑ is true?

Well, we could define $A(M, T \cdot 2^{-l(\beta)})$ to be $M^*(\beta, M, T)$

Another Track: we try .02 → .05

But in general, the system spends time $T \cdot 2^{-l(\alpha)}$ on Algms of length $l(\alpha)$.

.16 In OSS Rev 1985 P2: last 2 ¶'s, we say "$p = A(M, T)$; $G = M(p)$"

In the "Proof" of P13: (last ¶) $p = A(M, T \cdot 2^{-l(\beta M)}/C_A)$.

.18 This looks like an Error in the proof! — Not exactly! : see (.36)

.19 For "Any time" problems, hvr, || Lsrch produces at least as good as A(M) does in time T, but it takes $T \cdot 2^{|A|}$ times as long. A(M) is any "Anytime" Algm. (There is also the "C_A" factor).

Hvr, Perhaps .16–.18 is not an "error". If A really is faster by a factor f, than Lsrch, for that value of T, one way to say this is: $M(A(M, \frac{T}{f})) \geq M(Lsrch(M, T))$

Using T ← 2T method and $M^*(\beta_M, M, T \cdot 2^{-l(\beta M)})$ as the trial $p_{\beta M}$ · for $M(p_{\beta M})$.

$T = T_0 2^n$ for n^{th} round. Eventually we do $M^*(\beta_M, M, T_0 \cdot 2^n \cdot 2^{-l(\beta M)})$

and $n > l(\beta M)$, so for time $< (2T_0)$ A(M, ·) gets a certain value for G(p).

Lsrch gets same value, but it takes $T_0 \cdot 2^n$ time or $2^{l(\beta M)}$ times as long.

For any srch algm A(M, T); if $M^*(\beta_M, M, T) = A(M, T)$; then

~~$M^*(\beta_M$~~ using Lsrch, we get same G as $A(M, T \cdot 2^{-l(\beta M)})$ in time ≤ T.

One trouble w. the "ANYTIME" method of .19 is that I really don't have a good defn. of the Genre of "Anytime". If I "simulate" Anytime by T←2T Lsrch, this is a kind of defn. of Anytime.

(.36) Hvr, It seems clear that in some sense, Lsrch is $2^{-l(\beta M)}$ as fast as the fastest A(·M)". i.e. it takes $2^{l(\beta M)}$ times as long to get same results: We would have to know what A( ) was, before we know what this $2^{l(\beta M)}$ factor was! Hvr, not knowing A( ), the fact that Lsrch is ("only") $2^{+l(\beta M)}$ slower, is of much interest.

But, for any A( ) that we can state, we know $2^{l(\beta M)}$.... Similar to soln. of INV. problems

5/28/01
IDSIA

Lsrch for OZ probs

:00:196.40: Actually, there are several variations of Lsrch for OZ probs:

1) The method of OSS 1985: do trial code β_M: $\beta_M = M^*(M, T\cdot 2^{-\ell(\beta_M)})$ for Time limit $T\cdot 2^{-\ell(\beta_M)}$;
Simply do trials in β_M order (shortest first.): β_M are a prefix set so $\sum 2^{-\ell(\beta_M)} \leq 1$.

2) Do (1) but after each (finite) round do $T \leftarrow 2T$. (This is "Anytime" version of OZ prob)

3) Simply do each β_M; $\beta_M = (M\ T)$ ← NB in β_M order of length. —
Continue for as long as one has time. If one does n trials, it will take nT; $n \leq 2^{\ell(\beta_M)}$. (Thm: $\leq T\ 2^{\ell(\beta_M)}$) } .16
→ For a given T (≡ total time) its not clear what n & T to use for $nT = T$

4) Do "Parallel Lsrch", (but T limit is unclear: This only works for "Anytime" version of lsrch so $P_{\beta_M} = M^*(M)$. Time share $\propto 2^{-\ell(\beta_M)}$.

.14 1), 2) & 4) are probly "O.K." For all of them (i.e. any Lsrch for INV or OZ),
The problem of "correlns" betw. the pc's of the trials, is an "unsolved" problem.

.16 3) Might be O.K. when TM is "Matoro's first few OpCzn Algms. (OT's) are v.g. — use
T that one has for a "3C" limit. But if it is used it first "NEEDS WORK"!

Some notes from II Note

46.02 –.12 ; (30) → 48.01 Not quit very very inst!

15.05 on TM's Goc.
19.26 "cure Cancer" : Review of TM system
51 – 53 : early

136 : out to "convolution problem".

140 – 145 vernan 1999 – 2001

164.00ff demo of QATM

System.

202.02 : stuff for coding

Phys 2 " Not Bad QATM system"

137 : WON (warhor Need)

19.26 – 40 "Cure Cancer".