LevinSearch79 — Review of Levin 1979

Ray Solomonoff

Oxbridge Research
Mailing Address: P.O.B. 400404, Cambridge, Ma. 02140, U.S.A.
prettyvivo@gmail.com http://raysolomonoff.com

1.25.79 Rev (Levin) page 1

This will review recent results in "Levin 1979"; ($\sim$ 20.01 to 37.01: Especially 31.25 to 37.01). I will try to refer to exact region so that an expected explanation will be available if necessary.

This is a plan for a preliminary TM (Thinking Machine) — A description of the training sequence and just how the training sequence is solved. It includes *some* discussion of TM_2. In particular the *search routine* is treated exactly.

The corpus itself is an *ordered* sequence of induction problems. Each problem is a sequence of symbols ($\equiv$ to "sub-corpus"). The problem in each case is to find a short code (and/or many short codes or many longer codes) for the sequence or an "honest" way to assign a high aprip to that sub-corpus. (an "honest" probability assignment is one that can be backed up by codes).

[.15] This general problem *is* near *"adequate"*: in the sense that a solution to it would be readily transformed into a solution to all other induction problems — and therefore probably just about all scientific problems, including non-induction problems like NP problems. (see "Levin 1979", 90.33 [all similar numbers over 10 refer to "Levin 1979"]).

How the problem is solved: For each problem ($\equiv$ sub-corpus), we want to find an algm (algorithm) that takes the entire sub-corpus ($\equiv$ sc) as input (this is *not* a sequential input; the machine knows when the sub-corpus ends), then presents as output a set of induction codes and/or an honest aprip to the sub-corpus. If M is a Universal Machine, then M (q, sc) = $s_1, s_2, s_3, \ldots$; q is the description of algm, sc is the finite sub-corpus of interest, $s_1, s_2, s_3 \ldots$ are various short induction codes. The algm M(q, .) looks at the *entire* sc and tries to find short codes for it. Examples of algms that do this are: linear (or any other) regression, clustering, etc. (see 33.20 for a brief list of examples). In the case of linear regression, the algm makes a correlation matrix (which looks at the entire sc) which it uses to obtain the linear coefficients and σ^2 for Gaussian predictive coding.

Sometimes there will be ambiguity as to how to divide a code for the sub-corpus into a "q" ($\equiv$ algm description) and the rest of the sub-corpus description.

[marginal note: pc's (pc ≡ probability cost) of concepts depend on pc of corpus as coded by these concepts.]

e.g. in linear regression: a code (set) would be:

(L.R.), name of algm (Lin. Reg.)	3, no of coeffs	0.11, σ^2	1.2, 01.5, 2.6, the 3 coeffs	seq. of Gaussianly coded errors.

"L.R., 3" (3 = number of coefficients ≡ m) would give a single aprip to the sub-corpus and a set of codes clustering about the shortest code. "L.R." alone, would give a sequence of aprips: one for each value of m — these would be summed to get the total aprip.

Usually, we will choose the *shorter* algm's description, because this usually reduces total search time (see 35.30–37.01; 35.01–.28 is also helpful; however note 98.01).

A uio umc (universal input–output universal machine) is used (≡ process): so that the theorem about $\sum$cc <= $\mathrm{cc}_i/\mathrm{pc}_i$ is true. (cc is computation cost) *Search Technique*: The algms are tried in order of $\approx 2^{-\mathrm{q}_i}/\mathrm{T}_i \approx \mathrm{pc}_i/\mathrm{cc}_i$ — Where $2^{-\mathrm{q}_i}$ is the aprip of the i^{th} algm; and T_i is the computer cost of creating the i^{th} algm and using it to obtain the pc of the sub-corpus with respect to it.

The search is done by a "partial" (as in "partial recursive functions") listing of all algms in order of their pc_i's ($\approx 2^{-\mathrm{q}_i}$ if the shortest description of it is much better than other descriptions). This listing must be "partial" since certain descriptions will not describe actual prediction algms – i.e. they will not take finite time to compute the pc of the sub-corpus. [Note that the set of induction algms (≈ cpm's) is not recursively enumerable].

[.10] If one has a total cc available of T_a one *discards* an algm as soon as its $2^{-\mathrm{q}_i}/\mathrm{T}_i$ becomes $< 1/\mathrm{T}_a$, so only $2^{-\mathrm{q}_i}/\mathrm{T}_i > 1/\mathrm{T}_a$ are accepted.

If there is a very good algm with description q_0 and its $\sum$ cc is T_0 for the sc, then this algm will be discovered in a search for when the search parameter $1/\mathrm{T}_a < 2^{-\mathrm{q}_0}/\mathrm{T}_0$ is used.

If search parameter T_a is used, $\mathrm{T}_i < \mathrm{T}_a 2^{-\mathrm{q}_i}$ (for all trials used) (.10) The total search time is $\sum \mathrm{T}_i$ which is $< \mathrm{T}_a \sum 2^{-\mathrm{q}_i} < \mathrm{T}_a \approx \mathrm{T}_0 2^{\mathrm{q}_0} = \mathrm{T}_0/\mathrm{pc}_o$. Since $\sum_i 2^{-\mathrm{q}_i} = \sum$ of pc's of description < 1. Also by Kraft inequality since the q_i are descriptions of finite objects (including stop instructions) and therefore form a prefix set. In cases of interest, I think pc_0 will be *large* – say $> 2^{-10}$. So $1000\mathrm{T}_0$ will not be a very long search time.

Since $\mathrm{T}_0 2^{\mathrm{q}_0}$ is not ordinarily known, we will chose a certain T_a as a search parameter, and complete that search; then use $2\mathrm{T}_a$ and complete *that* search; then use $2^2\mathrm{T}_a$ and complete that search; etc. This doubling technique will at

most make total search time $2 \times T_0 2^{q_0}$. (See 29.01–.10, 29.20–29 and 33.35-34.20 for discussion of shortcuts).

[.30] ———————————————

[.32] Construction of the initial language used to describe the Algms:
This is done by listing a large number of algms that have been very useful in predicting the kinds of sub-corpi of interest. (See 33.20ff for a few examples of such algms). We then try to "factor" these algms into a set of concepts that can (most compactly, minimally) express all of the algms. Each of the concepts devised is assigned a pc, such that the corpus of algms being used is expressed most *compactly*.

[.36] Updating old concept pc's. and assigning *names* and pc's to newly created concepts:
This is done by considering the entire set of sub-corpi and various codings of this entire corpus. In a code of an entire corpus, it is easy to assign optimum pc's to concepts. These pc's are the observed probabilities of those concepts in the various environments in which they occur.

Ideally these pc's should be updated after each new sub-corpus is worked. Practically, it may be better to update less frequently.

Heuristics: How the system finds them automatically (see 32.15–.35, 95.06, also 36.01–37.01). In our corpus, we try to arrange it so that the search thresholds for the various sub-corpi are $\approx$ equivalent. This could be done by having all sc's

[.04] the same length and all search thresholds, T_a the same. Possibly one could

[.05] have T_a increase with sc length in a suitable manner if sc length *did* vary. Another possibility is to allow the "equivalent." T_a to vary over the sc's, but *slowly* change.

The idea here is that the pc's of concepts and of Algms, depend on how effective those abss are in prediction, subject to the constraint that the final use of the algm constructed *must have* cc $<$ Threshold. This automatically gives high pc to abss that create or help create algms that (a) give high pc to sc's, (b) do this (a) with a cc that is $<$ Threshold. These (a), (b) conditions are for a *fast* device that is good for *prediction*. The speed of implementation condition makes it a kind of "heuristic". That the lengths of sc's and the T_a's be the same, is necessary, so that information on concepts obtained from different sc's will be *comparable*.

Part of the heuristic concept is the TM_2 concept. Here the machine (TM_1) is allowed to look at good codings of the entire corpus($\equiv$ all sub-corpi) and tries

to find regularities in the construction of algms that have been most useful. Any such regularities can be used to increase the pc of existing algms and their sub–concepts. This operation is a TM_2 function, but it can be performed by TM_1, if the "sc" code is regarded as "just another sc of TM_1". (A possible difficulty is that this code will not, ordinarily, be of a length comparable to the other sub-corpi — in which case see .04–.05 for an idea) ...

[.25] *Another important idea* is that of "learning" as the search for a general sc progresses. We try $algm_1$ and it gets final sc pc of p_1 We try $algm_2$ and it gets final sc pc of p_2 and so on for $algm_3$ ... From this information we may be able to get ideas for new algms that are more likely to have *very high* final sc pc — I.E. we use the degree of success of algms 1, 2, 3 to modify the aprip (≡ pc) of the subsequent algm trials. (36.01–37.01 discusses this idea some: 36.25–37.01 in particular). This process is equivalent to a *new algm*, with (presumably) higher δ total pc per δ cc than others — so we give this new algm high pc (≡ small —q— for its description q). However this "learning" idea isn't yet too clear in my mind and I haven't properly formalized it for analysis (See 78.01 for better discussion of this: also 71.01–71.35 to some extent).

[.30] I conceive of this system as being part of a PMTM. (Partial Matching Thinking Machine) I suspect that many of the concepts needed to develop statistical predictors like linear regression will come from fields of "experience" that are not included in the usual statistical sub-corpi — so ≡ PMTM is *needed*. The ways in which these other modes of PMTM can be implemented using the formalism of this note, have not been worked out — also how information is transferred from one mode to another.

See 95.06 for discussion of "production" v.s. "search" heuristics.

General Discussion: *the form of the prediction problem* used seems adequate (1.15). *The idea of the sub-corpus not* being merely "sequential" is important. *The search method* using
pc of (algm) / (cc of algm's evaluation of sc)
is very good and important. (2.01–.30) — It makes it possible to get good results with short total search times.

That *heuristics* are automatically included in the formalism is important (3.01–.40) — Though this part is a bit vague in my mind.

This system *seems* like it might take care of heuristics as well as prediction without any special devices — looks like what I've been looking for for *many years*!

Some Major Problems!

(1) Devising an initial set of concepts that are complete (≡universal) so that all possible algms can be partially listed. (2.32) — also the concepts should give good aprips to good algms.

(2) The stuff on how heuristics are automatically implemented (3.01–.40) is

not so clear and hasn't been adequately analyzed or formalized. We definitely want to make sure that all possible heuristics *can* be worked into the system.

(3) How to implement PMTM using the present formalism (3.30–3.40 is a preliminary discussion). I'm not sure an actual PMTM is needed — but many of the sc's must be from fields superficially distant from the statistical prediction sub-corpi.

(4) The ways in which, say, linear regression (in various of its forms) could be discovered, and the training sequence needed has not been gone into at all — this is related to (3). (3) and (4) also tie in with (1) somewhat, and also (5).

(5) The general problem of devising training sequences for this TM ((1), (3), (4), are relevant).

(6) The problem of how T_a (or some other search parameter) must vary as the sc length varies (3.01–.05). Possibly it is not the *length* of the corpus but it's final pc (function of [corpus length times redundancy of corpus]), that should determine T_a? Perhaps the way information on a general concept derived from different sc's is combined, can use "weights" dependent on the respective sc lengths.

(7) The IPC problem of how to construct real machines that will compute solutions to these problems with minimum cc. It is likely that a factor of ≈ 1000 in speed up is possible. With technologically doubling of cc/$ every three years, say, this factor of 1000 amounts to $\approx$ *thirty years* of hardware improvement!

(8) In Real World, the corpus is not broken down into sc's of about equal length. Just how can this system deal with Real World data? (79.01–.15 for ideas on TM constucting its own training sequences from Real World data.)

1.25.79 Rev (Levin) page 5

This will cover Levin 79, 37.02–79.40. The *main* thing reviewed is 37.02–54.40; then 71.01–79.40. 53.01–54.40 reviews 37.02–52.40, to *some extent*; it is on a pure induction machine; it is a sort of review of stuff up to that point. 75.01–.24 reviews much of 56.02–74.40. 76.01–.28 reviews also.

N.B. 46.01–47.40 discusses "hueristics": also importance of using "historic" information on cc and pc of algms that were *actually used*.

I'll start with what I think is the major development since 37.01:

My present impression is that the device of Revision 1.01–4.40 may, indeed, be an adequate "study problem", in the sense that, *very many problem types can be expressed* as *either "NP problems" or "Gray NP problems"*. [NP problem is: $U(\alpha, x, .)$ is given: U is a umc. We have to find a y of length not much $> |x| \ni U(\alpha, x, y) = 0$.] α is the description of a fixed algm. x is some string of arbitrary but finite length. The time to compute $U(\alpha, x, y)$ is comparable to $|x|$. For a Gray NP problem, the conditions are similar but $U(\alpha, x, .)$ is a real number and we want to find a y that

[.16] maximizes it, or gives a largest value in the cc available for the search.] Finding a good induction code for a corpus, is a "Gray NP problem" (However,

see 6.02).

[.17] For $TM_1 = TM_a$, some of TM_1's problems will have to be, to both (a) Get a good code for a corpus, (b) Get a sequence of highest probability extrapolations of the corpus. Devising an algm to do this with an arbitrary corpus is a (Def.) *Gray NP problem* ($\equiv$ GNP problem). See 50.18 for a

[.20] discussion of the problem and 50.37–.40, 52.01–.05 for what looks like an adequate solution. However, problems like .17–20 *are* of the NP and GNP type, so *fine*!

[.25] Note that for the NP problems defined by $U(q_i, \alpha_i, \cdot)$; we want to find an algm description $q_i \ni y = U(q_i, \alpha_i, x_i)$ is a solution. I *think* this is a better way than to try to find $y = U(q_i, x_i)$, because in the $U(q, \alpha, x)$ problem, we want to find a q that is relatively *constant* for all α's; we want a general problem solving technique that can look at both α and x and then try to find a solution. The q_i's will then not change much with i (Important: This reduces search time, since q_{i+1} will be a "short distance" from q_i therefore the "δq" needed will be small say < 10 bits. It is *this* 10 bits that occurs as $m^{2^{10}}$ (Time to compute q and test y).

[.30–Def.] Some important kinds of problems that are (G)NP [(G)NP = NP or GNP] :

(1) Hill Climbing Problems — either direct search for optimum (history of 1) or History of a few or History of h $\gg 1 - >$ [See 78.02–.12 for discussions] $< -$ So this includes sequential optimization problems — the hardest type of scientific problem that I've thought about much. — In solving optimization, the solution is *an algm* for making trials – the algm basing the trials on past trials' success and is able to do exponents, because of the "averaging " structure of the GORC. Finding a solution to an (or a set of) equation(s) could be regarded as a GNP problem. Each closer approximation is higher G. Usual hill climbing methods *need* not be used — ordinary equation solutions methods can be used.

2) Finding a good Grammar for a corpus. [Note: Search for such a grammar will, if done well, involve noting what trials in the past for *this* sc_i have worked badly and well, and *why* they worked badly or well — so it *could* be a more advanced type of search such as discussed in 78.02–.12.]

3) Finding Solutions to ordinary NP problems, like (a) find $y \ni Axy = xi$: solve linear and non-linear equations in 1 or more unknowns; perhaps problems in "Number Theory", etc. An entire very interesting training sequence. could be constructed from *NP problems alone*.

See Lev 53.01–54.50: this is a *pure induction* TM: It is a kind of Review of the TM contemplated. ———————————————————————

[.02:5.16] [Side Note]: I'm not sure that GNP problems are so "naturally" worked: Though it *would* seem that if one considers the problem to be that of selecting an optimization algm, and "q" is the algm description, and T(q) is the time needed to "test" algm q, then $2^{q_0}T_{(q)}$ seconds are needed to find the algm, q_0. In the case of $U(q_i)$ being a optimization algm, testing time for each trial, can be rather large. Also the test result is not White or Black, but Gray. Finding induction codes is a *Gray* problem of this type.

See top of Page 5 of this review for list of partial reviews of the material of interest: *Read* these reviews. Then, try to characterize my "New Approach" as exactly as possible.

(Brief) Important New Characterizations of Proposed Approach:

(1) Breakup of corpus into sub-corpi (≡ sc's): this is so that: (a) Coding of sc's is easier because they are smaller. (Coding of the corpus as a whole is *much* more difficult than the sum of coding the parts [≡ sc's]). (b) One obtains partial Feedback Faster.

(2) The "ordering" of the sc's into a "training sequence". This is so regularities observed in earlier parts can be used to help code later parts.

[Without both (1) and (2) coding of large corpi is usually "TransComputable". (1) is possible without (2), but is not very good without (2): (2) is not possible without (1) however.]

(3) In coding a sc, we do *not* try for the most general type of code, but rather one of the form: [code of a cpm] ⌢ [Probability codes of sc with respect to that cpm]. Here, the codes of the cpm's can be relatively short (say ≈ 10 bits, with respect to codes used for the previous sub-corpi), even though the probability of sc with respect to that cpm is *very* small. We can code the cpm by using a single string. (Chaitin showed that $\sum$ pc of such finite objects is within constant factor of $2^{-\text{kcost}}$) ... The probabiity of the corpus with respect to the cpm can be computed directly by the cpm (*or* occasionally by summation, $\sum 2^{-\text{kcost}}$). (See 98.01 for *serious difficulties* with this idea!).

Sc's can be either *induction problems, NP problems or a more general type of optimization problem* — providing they are solvable by a short code (10 bits ... or perhaps up to 20 bits — depending on amount of time needed to generate the cpm (or whatever) and the time to check or evaluate the cpm (or whatever)). [Total expected cc of search is $\widetilde{<}2^q$ (cc of generating and checking an adequate solution). However, note difficulties with, say, *linear regression*]

(4) cc/pc Search: This, in general, may not be an optimum search strategy, but it may be acceptable/adequate. Later (see (5) ≡ 7.01) we can devise better search strategies. (On non-optimality of cc/pc search: 72.01–.40 summarized by 75.01.24).

---

(5) An understanding of how $TM_1 = TM_2$ is to be implemented: Just how it is a generalization of the "total problem"; and what the "total problem" *is*. How $TM_1 = TM_2$ may be improved — and just what it is "an approximation" to. Criticism of simple cc/pc search. Suggestions for better forms of search that *can* be practically implemented. Form of what *may* be the best possible search strategy. [71.01–.40, 79.16–20; 77.01–79.15, 76.01–.28]

[.12] [Side Note] One thing that I am not at this moment clear about: The details of just what TM does its cc/pc search on. Does it try to find (a) an $algm_i(X_i)$ (where X_i is sc_i to be coded), whose output is a code of legitimate probability of X_i, or

[.15] (b) An $algm_i \ni algm_i\ (X_j\ )$ (where X_i is sc_i to be coded) has the proper outputs for all $j \leq i$. [See 8.01–.40 for what looks like an adequate model — it's "b"]. I *think* I analyzed this to some extent within the last week or two. 5.25 (of revision) discusses this a little. [date 3.25.79: the solution of (b) is that the *latest* $algm_j$ (i is not necessarily changed for each sc_i) is our best bet for a TM_1 at time j: $algm_j$ *need not* work well for the beginning of the corpus, since $algm_j$ is a *search* algm, and the solutions to early part of the corpus have already been found. $algm_j$ is more the "best bet" for a good search algm for the remaining (or near) future of our corpus.].

(c) For NP problems: see 90.33 for what is probably an *adequate* approach to NP (and $\tilde{}$) problems. Say $\alpha(x, y) = 0$ is the problem: Given α, x to find y: we want $algm_i(\alpha_i, x_i)$ that generates a good trial y_i in a short time, *or* it generates a *solution* y_i in short time. Say $algm_i = U(P_i, \cdot, \cdot)$. P_i will have a short description with respect to the set of previously acceptible P_js. Again we want $algm_i \ni algm_i(\alpha_j, x_j)$ solves all problems for $j \leq i$. Testing these trial p_is can take *much time*, and various sampling methods could be used.

In .15 (b) we also have a similar condition, but by suitable sampling, we can discard a trial $algm_i$ if the pc's of the sc_js tested thus far gives a total bcost > acceptability threshold (sort of mindful of $\alpha_i\beta$ heuristic, in that one saves more time with this trick if one happens to get good guesses *early*). What I *can* do, is list the various possible kinds of probabilities, and training sequences, and methods of treatment, — so I can criticize them and decide on the best ones.

However try reading on what I've written on these probabilities — I think I've forgotten a lot.

[.30] Biblio on the *probability of .12*: 43.28–45.40 (43.28–44.27 is on H.C., but relevant); 48.01–50.40; 52.01–52.40 (both somewhat "talky"); 53.01–54.40 is on *pure induction* machine; 71.01–71.40 $->$ 76.01–79.40 on sub-optimality of the training sequence method; 72.01–72.40 $->$ 75.01–75.24 on sub-optimality of the pc/cc search; 76.01–76.40 summarizes both of these.

77.14–77.40 Discusses and criticizes various possible forms of TM (pure induction TM *only*).

Note: the machine of 77.14, also 53.01–54.40, is purely an *induction* machine: it doesn't do NP problems: *superficially*, the improvement and generalizations of 78.01–79.40; (onto the machine of 77.14) are all toward *pure induction*. Is this *really true*? *Can* I make good generalizations to work NP problems? Note: 90.33 ff *really* deals with the NP problems adequately; *solving induction problems* is

all I need to work on.

Auxiliary question: would the *pure induction* machine of Rev 1.01–4.40; 53.01–54.40 *work*? Narrowly relevant to the question of .12 is Rev. 5.25–5.30; 44.28 ff. Continues on page 8; Levin 88.01 ff *may* be that continuation however. (Looks very likely).

1.25.79 Rev (Levin) page 8

Discussion of how picture has changed since Rev. 1.01–7.40. (the foregoing is almost $\equiv$ to 94.01–95.40 with exceptions noted on (94.20–94.40) R and (94.10–94.12) R).

(0) See 97.01–97.40 for another brief discussion of the desired system.

(1) The corpus is as before, divided up into sc_i's sequentially.

[.03] (2) Around 1.15 I had a different idea as to what $M(q, sc_i)$ was:

[.04] Now I use $algm_j(sc_i, (b)) \equiv U(q_j, sc_i, (b)) -> P_1, P_2 \ldots$ for sc_i, where q_j is a program to simulate $Algm_j$ and P_1... is set of codes for sc_i and or probability distribution. (See 96.04, 96.19–96.20, 96.25–96.28 for what (b) can contain — also see .25 below ($\equiv$ (6)) $algm_j$ is the particular form of TM_1 at "time" j.

[.10] (3) sc_i can be either a short or long sequence. We do *not confine* ourselves to codes of the form (description of cpm_k) (description of sc_i with respect to CPM_k. See 98.01.

(4) $Algm_j$ can use *any* method of search. At first (small j), pure pc/cc search will be used, using (b) data only ($\equiv$ apri to sc_i). Laater, as $algm_j$ is improved (by TM_2), we will use more general types of search using observations on sc_i to direct the search.

(5) Initially, I will be TM_2 and design various $algm_j$'s and improve them. Since $algm_j$ can guide the search by looking at the whole sc_i, it *can* bias the search — so I want to be careful of this when I am TM_2 — One way to avoid bias is to *avoid* using sc_i observations. [98.20–98.30], however, later, when $TM_2 = TM_1$, there need not be any bias, since TM_j's *goal* is to get good predictions, and any bias would be against this goal.

[.25] (6) In 2.36 we considered *updating* pc's of old concepts, naming new concepts and assigning pc's to them. This *may* all be automatically taken care of by the argument (b). (b) includes any and all information that is *apri* to sc_i, so it can be used to help code sc_i. (b) includes srtms, definitions and concepts and their pc's. It includes these for *parallel* codes of the *previous* sc_i's — these parallel codes *are* useful and contain various high pc/cc abss. Anything associated with the previous sc_i's is legitimate to put into (b), (which is $\equiv$ a library). But how to optimumly *use* the information in (b) is a Question! We have to include various IR tricks to use (b). (b) also includes the *raw* uncoded previous corpus.

(7) We will concern ourselves with *induction* problems *only*. NP problems are a subclass of induction problems (as are *all* problems) and *can* be "included"

in the training sequence *in this form.* [90.33–91.13].

(8) Bibliog on improvements needed: 95.01 [Actually, by defining (b) properly: (See 8.04) this algm_j becomes a *potentially very non–el solution* and overcomes many objections!].

(9) Discussion of heuristics and how they are based on finite CB: 95.06–95.25 (also see Rev 9.10 on heuristics).

(10) The forgoing is $\approx\equiv< 94.01 - 95.40 >$; *exceptions* are noted on (94.10–94.12 (R), 94.20–94.40 (R)).

(11) All of the *sub*problems of optimizing algm_j [such as optimizing TM_2 or finding best way to make observations in real world, or finding good set of *observations* on sc_i, etc.] are regular "induction problems" and can be treated by the usual methods.

---

12) The main *methodological* idea: if there is *any* problem — induction *or* non–induction, write up the solution a human would find: describe the concepts, strings, definitions needed for that solution. Next problem: devise a new problem — $\ni$ these abss, get large enough pc/cc so that they are within our available $\sum$ cc. Next problem: devise new problems which ends up with abss that give the abss needed in Problem_2 high enough pc/cc, etc.

Essentially a "working backwards;" approach. This is *alternative* to trying to devise a good set of primitive abss. and working *up* though a suitable training sequence to get to difficult problems. It is possible to use *both* approaches, however.

[.10] (13) Of theoretical interest and very *important*: On *heuristics*: 95.06–95.25, 32.15–32.35; 31.01–37.01; 78.01 ff; 71.01–71.35. Many more references;

[.17] but an important problem is that of 40.25–40.30; 46.01–47.40. The problem is that various abss have obtained various pc's with respect to various sc_i's using various CB's. ($\text{CB} \equiv \text{T}_0 < 2^{\text{q}}\text{T}_{\text{q}}$). So how should one assign pc's to abss used with a particular CB — which can be different from the CBs that those pc's were obtained with?

40.25–40.30 was a "rough and dirty solution" for a certain kind of TM: I'm not sure how relevant it is to the present TM.

One not bad approach is to express the pc of each abss as a function of CB, so we can directly assign a pc for each possible CB level. While pc of a given abss may *not* be monotonic in CB, (So this function can have a complicated form), — due to pc's of competing abss — it *may* be that the *unnormalized* pc's of various competing abss. can be approximated by *monotonic* functions of CB.