Two Letters on Ways to Solve Problems
Letter 1

Ray Solomonoff

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

1. Try to define the problem clearly. Remove irrelevant parts of its description. If the problem is hard to define, make this definition be the new "Top Goal". If problem definition remains very difficult, ask "Do I really want to solve this problem?" — Perhaps I shouldn't be spending time on it. Another possibility is that the problem itself is not well definable at all, that it is meaningless — that it is part of a so–far unsuccessful attempt to break a larger problem in well–defined parts. It might be best to go back to the larger problem and try to break it into pieces in another way, or simply try to solve it without breaking it up.

2. If it is an optimization problem (most problems *are*), make a clear statement of what is to be optimized. Could the goal be defined if one had an infinite amount of time and computer power to realize it? [I have found this to be one of my strongest heuristics — if one is unable to do this then it is strong indication that one doesn't understand the problem].

3. State the problem in clear, operational form. Again, inability to do this is strong indication that you don't understand the problem.

4. Devise "study problems" and try to solve them. A good "study problem" is similar to the original problem. Solving it will suggest ideas, concepts, useful for solving the original problem. Study problems might be analogous to the original problem.

5. Express problem as an AND, OR network of problems. "Divide and Conquer" — means set of AND problems. "Equivalent problems" are OR problems. A difficult problem can often be expressed as a Boolian network of AND and OR problems. If there are N problems in the net, then deciding which problem to work on next, takes an amount of time proportional to N. In general, there may be several alternate ways to break a problem into parts.

6. Guess at a solution, then try to prove it is correct or incorrect. Understanding why certain trial solutions failed can help find correct solutions.

7. Try specific numerical examples for more general problem. An approximate solution can give some insight on what general solution looks like.

8. Generalize the problem. This sometimes makes it easier to solve by getting rid of inessential features.

9. Specialize the problem. Solving several special cases can give insight on the general case.

10. Is the problem solvable? Try to prove that it is not. Is it possible to prove that a solution exists? Is the problem solvable but requires too much computation time?

11. Can the problem be expressed as a "Hill climbing problem"? If so, is the topology smooth enough to make "steepest descent" feasible? Consider less "greedy" methods. For many non–linear optimizations the Levenberg–Marquardt method is very good.

12. Can the problem be configured as a GPS (General Problem Solver) problem — with a "Goodness of criterion" that is a vector — all components of the vector have to be zero for a solution. When using Genetic Algorithms to solve such problems, the fitness vector is usually converted to a single scale. This loses the individuality of the original vector components. Better to use a different mutation/crossover algorithm for each component.

13. If the problem is well known and unsolved, try to reformulate it in novel ways: Transform it into areas in which you have unusual expertise.

14. If you are a machine the idea of 1) "Defining the problem clearly" is not useful. The problem has already been formalized and stripped of what the user thinks are irrelevant data. This "irrelevant" information can be very useful for problem solving. The "context" of a problem, (both local context and extended context) should be included as auxiliary information. They suggest methods of problem solving, through associations. If this information is not given to the machine, it will have to build up this information through its own experience — which can be very time consuming. The user should try to include as much auxiliary information as possible.

15. Some time ago, you suggested that one might make interesting discoveries by listing various physical effects and considering the Cartesian product of several such lists. Each conjunction of effects could suggest an invention, practical application or scientific breakthrough. Zwicky developed something like this in attempting to list all possible solutions to problems and investigating all of them. Gunkel extended this idea by making many

lists of ideas. One problem with the Gunkel approach is that there are too many possibilities to investigate. My idea is to associate a probability with each list element (so total probability of each list is 1). Each element of the Cartesian product, then has the product of the probabilities of its elements. We can then search the "nodes" in order of these probability products. Usually when people do Gunklish searches, they pick a few elements of high likelihood from each list, and combine them. This is a very wasteful method of doing trials and is not nearly as good as trials in strict probability order. Algorithmic Probability is a more general, more exact way of doing these searches.