The Local to Global Principle

John Mount*

November 11, 2009

Abstract

We describe the "the local to global principle." It is a principle used to break algorithmic problem solving into two distinct phases (local criticism followed by global solution) and is an aid both in the design and in the application of algorithms. Instead of giving a formal definition of the principle we quickly define it and discuss a few examples and methods.

Contents

1 Introduction

A common vain hope of computer scientists and algorithm designers is that a domain expert has already "boiled down" a problem to a precise, but unsolved, algorithmic core. On this point the mathematician Gian-Carlo Rota wrote:

> One of the rarest mathematical talents is the talent for applied mathematics, for picking out of a maze of experimental data the two or three parameters that are relevant, and to discard all other data. This talent is rare. It is taught only at the shop level.[Rot97, "A Mathematician's Gossip"]

We describe a useful tool for designing algorithmic applications and solutions which we call "the local to global principle." The local to global principle is the method of deriving applications and solutions by specifying "local" (and deliberately myopic) heuristics, critiques and methods followed by using a powerful general method to "globalize" this specification into a complete solution.

There are many important problem solving prescriptions and methods of thought already systematically described and taught:

*email: mailto:jmount@win-vector.com web: http://www.win-vector.com/

- Bacon's "New Organon" and Mill's principles of inductive logic.[Mil02]
- Feynman's genius method.[Rot97, "Ten Lessons I Wish I Had Been Taught"]
- Reductionism (top down and bottom up).
- Divide and conquer.[CLRS09]
- Forward deduction, backwards induction.
- Root Cause Analysis.
- Polya's heuristic and conjecture and prove patterns [Pol71, Pol54a, Pol54b]
- Doron Zeilberger's "Method of Undetermined Generalization and Specialization." [Zei95]
- Zbigniew Michalewicz and David B. Fogel's presentation of evolutionary algorithms.[MF00]

The local to global principle is more of an organizational pattern than "computer aided technique" as no one specific species of software or family of notation is required.

The local to global principle can be identified in a number of previous important applications, but it is not currently an identified principle.[1] The principle is very general, so any succinct description of it is going to be painfully vague. Instead, we explain the principle by discussing some example applications and methods. For each of our example applications we deliberately use a different globalization technique. The effective algorithmist or practitioner must in fact come to each problem already familiar with a reasonably large set of already known local and global techniques, so we conclude with some appropriate fields of study and preparation.

The local to global principle is divided into two parts: local encoding of the problem followed by a globalization step that uses the encoding. The guiding feature of local encodings is that they are usually easy to compute from the data at hand. Any extension that looks like enumeration, search or optimization is best left to the global step. The local step is essentially the translation of your problem into an abstract language that is ready for the globalization step. In contrast globalization methods are often "off the shelf" in that once you abstract and encode the particulars of your problem you can look for pre-existing useful methods or software to finish your solution. The idea of globalization is to find a best overall or global compromise between competing local criteria. The local step does not so much have to avoid conflicts but instead "price them." There is also an important trade-off that sophisticated local techniques allow the use of simpler globalization methods and more powerful globalization methods allow the use of simpler local techniques.

2 The Examples

To demonstrate the breadth of the local to global principle we choose a diverse collection of example applications: web page link analysis, natural language processing and machine learning. For each example application we will set up the problem, introduce a reasonable set of local criteria and pick an appropriate globalization technique. We will favor finishing each example without describing the globalization technique in detail, as this would distract from our point and is best left to the given references. These examples are previously solved problems, our contribution is demonstrating the shared underlying principle.

[1] The pre-existing practice that comes cloesest to the local o global principle is found in operations research where encoding a problem to be solved by an optimizer is a central technique. We claim the natural statement of the local to global principle is more general than *always* encoding constraints for a particular optimizer (in particular globalization is not always optimization).

2.1 Web Page Link Analysis

For our first example application we demonstrate web page link analysis in the form of the famous PageRank score.[PBMW98]

One of the many good ideas leading up to the early Google search engine was the design of a non-text based measure of importance or interestingness of web pages. A search engine that could fold "interestingness" or popularity into its notion of relevance could better sort important pages into the search user's view. When the web got so large that there were many pages that were exact matches to any common user query popularity became a critical consideration. A link based notion of popularity exploits what is important about the web (the link structure, for example see [Kle97]) and avoids having to depend on a lot of natural language understanding technology. This technique also uses authority outside of the given page, so has some hope at being resistant (though not immune) to web-spam.

Taken all at once, the task of designing a score of page importance is a daunting task. However, by working in stages (as the local to global principle prescribes) we can quickly derive interesting scores including the famous PageRank score. We start with the idea that popularity (or the amount of web traffic a page receives) is (loosely) correlated with importance. So for our first approximation step we decide to try to estimate popularity (or web traffic) and use this estimate as our importance score. Accurately estimating web traffic is itself a hard problem and a big industry (just a few of the major companies involved in this are: Google/Urchin, Quantcast, Nielsen, comScore, Alexa, Hitwise and LookSmart). For our second approximation step we are going to try and estimate popularity from the link structure[2] of the web (using no other measurements or historic data) and use this as our score. This link based estimate is unlikely to completely reproduce real web surfing patterns, but it is very interesting in its own right and has been proven in the market to be a useful score.

Now the problem is to try to estimate the popularity of a web page from the link structure of the web. We claim: we can generate a useful (but not necessarily accurate) estimate of web traffic from the web's link structure alone. Consider Figure 1 where we have a universe of three web pages A,B and C that link to each other in the pattern illustrated by what is called a graph[3]

[2]By "link structure" we mean which web pages link to which other web pages.

[3]Remember, a graph is diagram consisting of nodes and edges (here depicted as arrows).

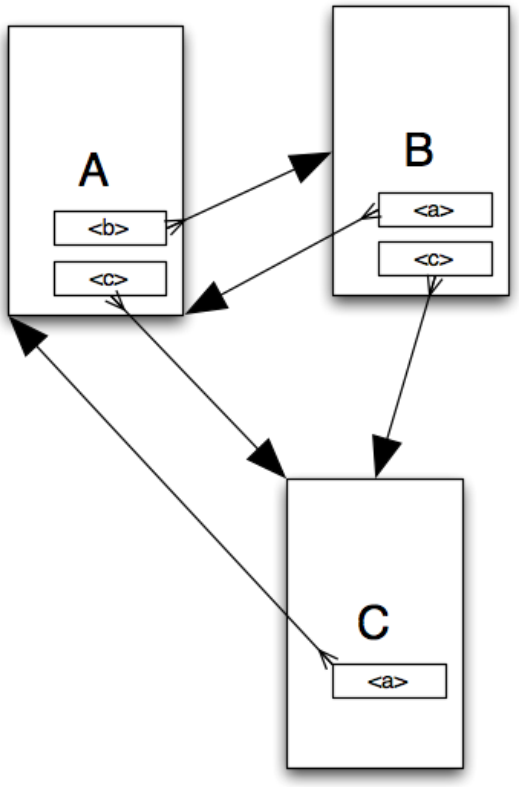


Figure 1: A set of Mutually Linked Web Pages

In Figure 1 we can consider each link to another page as evidence the other page is interesting or popular. One idea is to simulate a very simple web surfer who clicks on the links on a page uniformly at random. This is called "the random surfer model" and even a model this simple allows us to read some useful information from the link structure of the web. For instance, we could ask what fraction of their time the random surfer spends on each web page, with an eye to the idea that the pages the random surfer visits more often are the more important ones. Let $p(A)$ denote the proportion of time the random web surfer spends on page A (and define $p(B)$ and $p(C)$ similarly). While we do not know any of $p(A), p(B)$ or $p(C)$ we can derive some relationships between them by inspecting the link graph:

$$\begin{aligned} p(A) &= \frac{1}{2}P(B) + P(C) \\ p(B) &= \frac{1}{2}P(A) \\ p(C) &= \frac{1}{2}P(A) + \frac{1}{2}P(B). \end{aligned}$$

The first equation is just reading from the graph that: all visits on page-A must come from pages B and C, half of the visitors on page-B continue on to A and all of the visitors on page-C continue on to A. The second and third equations are the appropriate summaries of how traffic is routed to pages B and C. We can insist that $P(A) + P(B) + P(C) = 1$ as we want these numbers to represent the fraction of time the random web surfer spends on each page. A more sophisticated model would add more features[4] to get a more useful result.

[4]For example the model could account for:

It turns out we have already encoded enough local rules to completely determine $P(A), P(B)$ and $P(C)$. In this example application an algorithmist already familiar with linear algebra [Str76] would recognize these local conditions as "a system of linear equations." Solving even web-scale systems of linear systems is considered easy with modern techniques and modern computers. For our small example example the solution is: $p(A) = \frac{4}{9}$, $p(B) = \frac{2}{9}$, and $p(C) = \frac{3}{9}$. The role of the local steps was to reduce a new problem (estimating the importance or popularity of web page from the link structure) to something with its *already known* known techniques (like solving a linear system as illustrated in Figure 2).

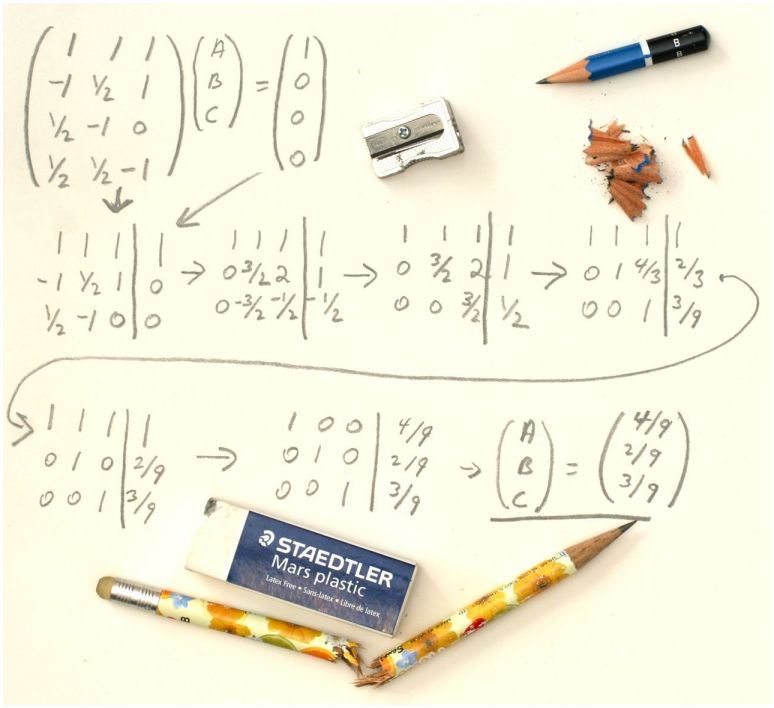


Figure 2: Linear Algebra Solution: As Taught in School

So page-A is the most important page by the PageRank measure.

In this example application the local step was setting up the system of linear equalities (which are easy to derive from the web link graph) and the global step was solving the entire system for the final scores (which were not obvious). You spend most of your time encoding the problem and then use a known technique (in this case solving a linear system) to finish the solution.

2.2 Natural Language Processing

Our next example application is natural language processing [Cha96, Cha97]. Speech recognition (the alignment or transcription of recognized intelligible segments of sound to written text) is an important problem in natural language processing. An example problem is the need to find the most likely text matching a sequence of sounds such as is shown in Figure 3.

- surfers entering and leaving the model
- link odds that vary where they are on a page
- surfers staying on a page proportional to how much text is on the page
- matching known traffic and click behavior where we have such data.

For simplicity we will just stick with the example given example.

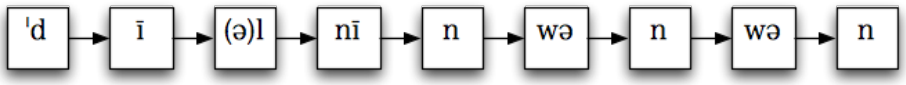


Figure 3: A Sequence of Sounds

Consider Figure 4 (which shows a bad transcription) and Figure 5 (which shows a good transcription).

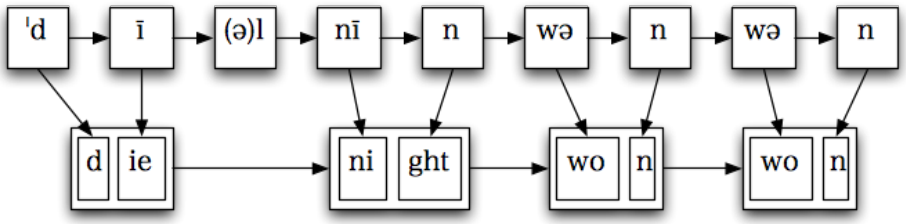


Figure 4: A Bad Transcription

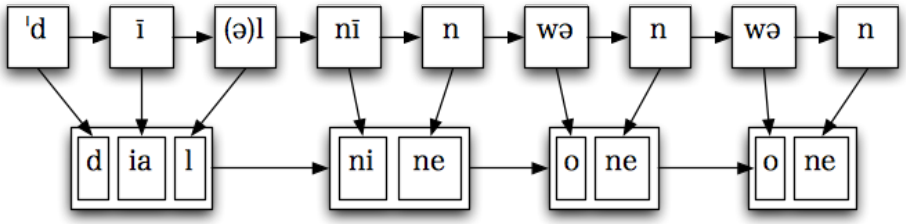


Figure 5: A Good Transcription

Our claim: we can (given access to training data, and this is the age of data [HNP09]) solve this problem with a local step that is a set of simple criticisms of proposed transcriptions. A good starting point is a database of previous sounds to text transcriptions. This database allows the construction of a series of tables that give the historic frequency (or probability) of all of the following:

- Prior probability of each sound
- Probability of each sound given the immediately previous sound
- Prior probability of each word
- Probability of each word given the immediately previous word
- Which combinations of word fragments are legitimate words
- Probability of each sound being assigned to each word fragment (syllables, phonemes and so on).

These tables encode a "speech model" (the rules involving sounds only), a language model (the rules involving text or words only) and the linkage between the two models. These models are deliberately simple in that they capture only local interactions (like probability of a word given the word before it) but no long range interactions (like subject predicate agreement).

Each box, nested box and arrow on our diagram represents one possible local critique. For each item in our diagram (again, the boxes and arrows) we can use our tables to assign a goodness or plausibility score. For instance bad word to word transitions (like "won" → "won") will be rare in our historic tables so, just looking up probabilities from the tables (or, better, using the logarithms of probabilities) gives as a "plausibility score" that prefers known patterns of language. Then a score for the overall transcription can be derived by multiplying all of the local scores together. These local scores (though simple) already have encoded enough evidence to prefer the good transcription to the bad transcription *without* requiring any deep knowledge of speech, text or the meaning of the text. This is because the bad transcription has a series of obvious flaws such as: unlikely sound to word fragment assignments and unlikely word to word transitions.

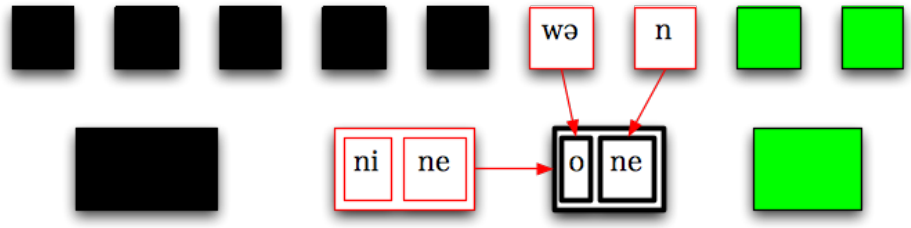


Figure 6: Naively Extending a Partial Transcription

For example consider Figure 6 where a naive solver is in the process of considering selecting the word "one" as the third word to fill in. The *only* local critiques they need to consider are:

- how likely the word "one" is in general (call this $P[one]$)
- how likely the word "one" is to follow the word "nine" (call this $P[one|nine]$)
- how likely the letter sequence "o" is given the sound "wə" (call this $P[o|\text{wə}]$)
- how likely the letter sequence "ne" is given the sound "n" (call this $P[ne|\text{n}]$).

So the local plausibility of the fill-in word "one" is: $P[one] \times P[one|nine] \times P[o|\text{wə}] \times P[ne|\text{o}]$. We will call this the critique of "one" in position 3 and write as $C_3(w_2, one)$ where w_2 is the word known to be in position 2. Similarly we can generate all of the possible critiques $C_1(w_1)$, $C_2(w_1, w_2)$, $C_3(w_2, w_3)$, $C_4(w_3, w_4)$ and the overall criticize of a sequence $w_1w_2w_3w_4$: $C_1(w_1) \times C_2(w_1, w_2) \times C_3(w_2, w_3) \times C_4(w_3, w_4)$ from our pre-computed tables of probabilities. Notice for all of these critiques only the immediately previous word and the nearby sounds were used to determine the plausibility of the word we are attempting to fit in. Instead of using these critiques to directly fill in a possible solution (or using search) we will package up these critiques (in the form of the $C_i()$) and pass them on to a powerful separate globalization step called Dynamic Programming [Bel57].

The globalization or finding of a best overall transcription is not trivial even though our score is simple. This is because the overall *best* sequence could depend on clever non-local fill-ins (like deliberately picking a less likely first word to allow a later favored transition to a fantastically good third word). Dynamic Programing does not fill in the transcription from left to right, but instead uses a table of scores derived from the left to right arrows and the $C_i()$. In our example Dynamic Programming consists of building a table of information as shown in Figure 7. Let i represent the word position we are working looking at (so i ranges from 1 to 4) and let w be a variable that ranges over every word in the dictionary. Our table is indexed by i and w and when filled in $T(i, w)$ stores what the highest "plausibility score" of a partial sequence of words where words 1 through i have been filled in and the i-th word is w.

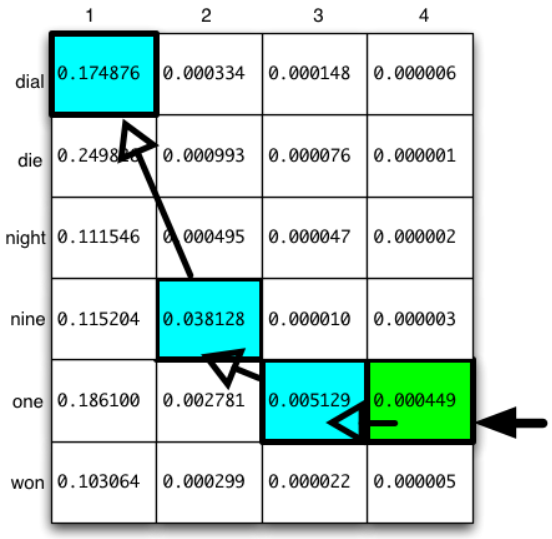


Figure 7: Dynamic Programming: Back Chaining in $T()$ for a Solution

If we already had this magic table $T()$ we could find a best possible sequence by "back chaining." We start by finding a fourth word (w_4) such that $T(4, w_4)$ is maximal (in this case "one"). We then find a best third word (w_3) by enumerating all words and picking w_3 such that $T(3, w_3) \times C_4(w_3, w_4) = T(4, w_4)$. We continue back until we had found words w_2 and w_1 to get a complete best sequence. Notice that we work from right to left (backwards) and except for the starting step we pick each word to match the calculation we are trying to un-roll, not to be the maximal entry in the column. For instance we pick $w_1 = dial$ even though it does not have a the highest score, but because $T(1, dial)C_2(dial, nine)C_3(nine, one)C_4(one, one) = T(4, one)$ is the maximal complete chain.

Of course, we don't start with the table $T()$ already filled in- so we need a procedure to build it. This procedure is the heart of the Dynamic Programming method (for more examples see: "Introduction to Algorithms" [CLRS09]). Notice that $T(1, w)$ can be filled in for all w just by plugging in words and computing the critiques $C_1(w)$ (i.e. $T(1, w) = C_1(w)$). Once all the $T(1, w)$ are filled in we can fill in the the $T(2, w)$ with the general (and slightly trickier) formula:

$$T(i+1, w) = \max_v T(i, v)C_{i+1}(v, w)$$

as we illustrate for $T(2, nine)$ in Figure 8.

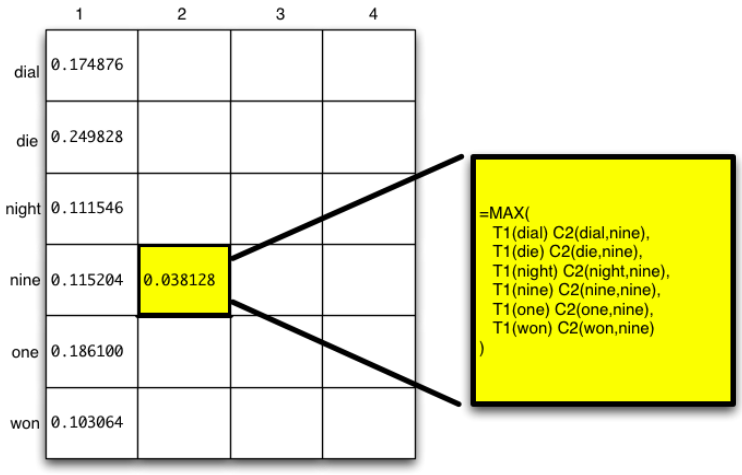


Figure 8: Dynamic Programming: Building the Table $T()$

The magic of the Dynamic Programing technique is: by being careful to not store too much in the table $T(i, w)$ we avoid an explosion in record keeping that would render the method inefficient. Dynamic Programming exploits the small dependence structure encoded in $C_i()$ (each box in our diagram depending on only a few arrows) and as we have shown can find "clever" solutions (such as taking a sub-optimal first word to get better transitions into preferred later words). For those who want more detail on solving this problem we recommend [Cha96] (as our goal is not to fully explain Dynamic Programming, but to demonstrate how it could be applied to the transcription problem as a pre-packaged globalizer).

In this example the local step was the graph link based critiques and the globalization step was Dynamic Programming. The separation of concerns from the local scoring to the globalizing step is a strength of the local to global principle.

2.3 Machine Learning

Our final example application is machine learning. Machine learning is loosely defined as computer programs that adapt or learn from data. Thomas Mitchell helps distinguish this activity as a specialty of artificial intelligence that concentrates on "well-posed learning problems." [Mit97] Trevor Hastie, Robert Tibshirani, Jerome Friedman emphasize the relation to statistics (versus more traditional symbolic AI) [TH09]. A simple demonstration can be found in [Mou09b].

Machine learning is perhaps the strongest example of the local to global principle and is inspired by the work of Kristin P. Bennett and Emilio Parrado-Hernandez [BPH06]. In hindsight many machine learning algorithms (each of which has had a turn at being "the most exciting breakthrough ever" for a while) can be seen as the pairing of a performance criterion (which we call a local criterion as it applies to one specific set of parameter values at a time) and an optimization method (what we have been calling the globalization step). The work of Bennett and Parrado-Hernandez calls this distinction out and shows how it is not productive to present machine learning systems as unique named monolithic units, but instead to consider how to break them into an objective function and an optimizer. This allows both choice of better optimizers (such as replacing the inferior method of gradient descent method wherever it occurs) and for explicit control of important concepts such as hypothesis regularization and control of over-fitting (which some algorithms claim to achieve by deliberately using early exit from a an inferior optimizer).

At a "30,000 feet level" we can build a table of common machine learning techniques and name what is commonly used to implement their local and global steps. When a machine learning algorithm

is defined by what conditions are meant to be true at the optimum we are no longer bound by details of the original implementation and can examine fix and improve the components.[5] Table 1 is a crude summary of a wide selection for machine learning algorithms that may be more likely to offend everybody than just offend somebody. But this is also the point: it is the algorithmist's job to think fluidly (beyond given names and provenances) and to invent scaffolding to convert partial analogies into practical correspondences.

Machine Learning Method	Local Criterion	Globalization Method
Linear Regression [BF97]	square error	Linear Algebra
Linear Discriminant Analysis [Fis36]	square error	Linear Algebra
Logistic Regression [Kom08]	logit penalty	Newton's Method
Perceptron [BRS91] [BD02]	error rate	error based update
Naive Bayes [MK00] [Mar61] [Lew98]	frequency tables	arithmetic
Nearest Neighbor [AC06] [IM99] [AI06]	Kernel Methods	enumeration, projection
Decision Trees [BFSO84]	information theory	partitioning
clustering [CV05]	square error	partitioning
MaxEnt [Gru00] [GD04] [Ski88]	entropy penalty	Newton's Method
Neural Net with Back Propagation [Hus99]	sigmoid penalty function	Automatic Differentiation, steepest descent
Winnow [KWA95]	error rate	multiplicative error based update
Boosting [FS99] [Bre00] [CSS02] [TTV08]	weighted errors, data re-weighting	Conjugate Gradient
HMM [KCVM04]	probability penalty	Gibbs Sampler
Latent Dirichlet Allocation [BNJ03]	KL divergence	Variational Methods
Support Vector Machine [Joa98] [STC00]	L1 Margin, Kernel Methods	Quadratic Optimization

Table 1: Various Machine Learning Techniques

This table is a necessarily crude summary. For example: notice that several known techniques can not even be distinguished from each other by the local and global columns of the table.

There are a few points we would like to make. Back propagation was considered unique to Neural Nets for quite a while because it was so entwined with the technique it was not recognized as the simple application of Automatic Differentiation [RC96] that it is. Support Vector Machines (SVM) are remarkable for their uniform very good choice of component methods (maximum L1 margin objective regularization, Kernel Methods [STC04] and sophisticated optimization methods [Joa06]). Many of the machine learning methods that SVM outperforms become again competitive when they adopt some of SVM's technologies (especially using kernel methods to produce synthetic features).

Beyond these points we invoke a "globalizers are pre-packaged" principle and leave the discussion of machine learning and optimization to our reference: [BPH06]. In this example the local step is a per-example score or penalty and the globalization step is optimization.

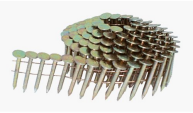
3 Some Methods

The application of the local to global principle is similar to the Feynman "genius method." Feynman's method is to always have in mind a list of problems and a list of solution methods. The genius step is:

[5] When a system is named and defined as an exact set of procedures the system can, by definition, not be improved. This is because with any change in procedure we have a new system that no longer matches the original definition and therefore requires a new name.

anytime you see a new problem or a new solution method to immediately try it against every item from the complementary list. [Rot97, "Ten Lessons I Wish I Had Been Taught"] This deliberate retention and activity greatly increases your problem solving ability. The power of the local to global principle is itself proportional to the number of local methods times the number of globalization strategies. Of course, to even start: the practitioner must already have available a number of candidate local and globalization methods. We list some methods and some guidance on variation and invention.

3.1 Local Methods



Good sources of ideas and analogies for local methods include:

- Introduce a Graph Structure

  A graph structure is a network of nodes connected by edges. Use of graphs was demonstrated both in the natural language processing and web page link analysis examples. We can dress up how we solved these problems and say we used a "Hidden Markov Model", but the real power was we encoded our problem in a simple graph. Some problems (especially those from logic or those involving time) are essentially solved once they are translated out of their original form and into graph notation (for an example see: [Mou00]).

- Appeal to Physical Conservation Laws

  A good example physical law is Kirchhoff's law or conservation of flow. All of the web page link analysis's equations were derived by saying that the attention of at node is essentially the sum of attentions from other nodes (more sophisticated versions of the analysis actually do create and destroy flow, but they do it in a principled way).

- Encode the Problem into an Objective Function

  This method is essentially your declaration that you intend to use an optimizer for the globalization step. In operations research this specific technique has long been the practice (with no disrespect: a very productive part of operations research has been translating different problems into linear programs so the simplex method can be applied, for an example see [Mou09a]).

- Gradient Like Computations

  Includes Gradients, Secants, Lagrangians and other ideas from calculus. Gradients can drive optimizer based globalizers and techniques like Lagrangians are often powerful enough use mere inspection as the globalization step.

- Violation Driven Updates

  This method is particularly effective when your problem is not amenable to continuous optimization. A good example is the Lin-Kernighan heuristic for solving the traveling salesman problem.[LK73] This heuristic looks at subsets of the problem and suggests improving "surgeries" (until no more such improvements are possible).

- Introduction of Symbols

  Often, as with the web page link analysis example, you can not specify specific values for the unknowns, but you can specify relationships. You often can then solve for the symbols or introduce additional conditions and use an optimizer to complete the solution (see for example the maximum entropy method as described in [Ski88]).

- Over Specification

If we anticipate using a global step like search, enumeration, summation or integration then over specification is a good local idea.

For example: consider computing the probability that a fair count flipped 10 times comes up with heads exactly 3 times. The easiest way to perform this calculation is to specify exactly which 3 coins come up heads (the local over-specification) and then sum over all choices of 3 out of 10 coins (the global step). In mathematical notation this is:

$$P[\text{exactly 3 heads out of 10 flips}] = \binom{10}{3} 2^{-10} \approx 0.117$$

or just under 12%.

- Under Specification

  One of the core principles of Dynamic Programming is to forget as much as possible about partial solutions, keeping only partial solution cost and just enough information to extend partial solutions. If you anticipate using something like Dynamic Programming as your globalization step then your goal should be to under specify.

- Tables

  A key step of the natural language processing example was the use of tables of past experience to determine which sounds likely corresponded to which words, which words likely followed each other (and so on). Encoding domain knowledge or expertise as probability tables is a very effective problem solving strategy (especially if the globalization strategy is going to be search or Dynamic Programming). In natural language processing examples tables and statistics are *much* easier to manage than comprehensive rules or grammars.

- Set up as Ranking or Machine Learning Problem

  This tactic is especially appropriate if your solution success metric is counts, frequencies or probabilities (instead of having to always be correct or always be optimal).

3.2 Globalization Methods

The universe of possible globalization methods is very diverse (in particular globalization is not always optimization).

- Search / Enumeration

  Search can be slow, but it is always an option to consider. If your problem translates naturally into a graph structure or your solutions are naturally seen as being composed of small pieces search should be considered. One of the big advantages using the local phase to formally encode your problem's structure and putting search off to the global phase is: you can use advanced search techniques. Once you are freed from your specific problem details it becomes much easier to consider search techniques like branch and bound, A*, game theoretic search and general speed-up techniques like hashing and caching.

- Dynamic Programming

  If your problem has a bit more structure (in that partial solutions summarize and compose easily) then you can likely replace search with Dynamic Programming. The advantage is that Dynamic Programming typically offers an incredible speed up when compared to search.

- Optimization

  If your problem is continuous (involves numbers instead of discrete or categorical decisions), can be encoded as a reasonable objective function (linear, positive definite quadratic) and has reasonable constraints (linear or convex) then you can immediately apply an optimizer as your globalization step. Typical optimization methods include: conjugate gradient, Newton methods, quasi Newton methods, linear programming and quadratic programming.

- Combinatorial Optimization

  If your problem includes a "discrete variables" (that is variables that take on one of fixed set of values instead of values from a numeric range) then you may not be able to apply standard optimization techniques. At this point you may want to use more expensive combinatorial optimization techniques like integer linear programing or constraint satisfaction.

- Fixed Point Methods / Iteration

  Fixed point methods are based on the idea: "incrementally improve until there is no incremental improvement possible." If the problem is continuous this is similar to steepest descent. If the problem is discrete then this is similar the Lin-Kernighan heuristic.

- Linear Algebra

  The web page link analysis and optimization examples were essentially solved once we reduced them to linear algebra. If you can write your problem as a linear relationship between unknowns or as the fixed-point of a linear operator (i.e. an x such that $Ax = x$) then you can immediately use linear algebra to solve the problem at very large scale (e.g. web scale).

- Sampling / Problem Kernels

  A very successful line of attack on large problems is to reduce to a smaller problem containing most of the essential difficulty. David Karger has produced a number of effective algorithms for graph cuts and flows using a theory of sampling [Kar98]. Rod Downey and M. Fellows have demonstrated an effective theory of "problem kernels" that finds solution by focusing on smaller sub-problems (on which we can afford to use more expensive procedures).[DF98]

- Amortized Analysis / Economic Mechanism Methods

  Daniel Sleator and Robert Tarjan's ideas of amortized analysis [ST85] allow approximation schemes similar to problem kernels. The method is to approximately optimize by pairing a bunch of unavoidable large penalties (conditions we can't meet) with some accounting credits (say bonuses from other conditions we are meeting very well). We then isolate these paired items and optimize the rest of the problem exactly. The technique often works by showing the approximation can not be too bad because, due to the pairing of large penalties to good credits, there can not be too many large penalties. An informal example is: if it is impossible to pick someplace where all of an office will eat for lunch, perhaps you can solve the problem by paying one person to accept a restaurant they do not like (if the removal of their objection opens up a venue that is acceptable to everybody else).

- Relaxation / Homotopic methods

  These methods involve changing hard constraints to soft penalties (so allowing the constraints to be violated, but at a slowly increasing cost). After such a relaxation the homotopic (or continuous deformation) method is to increase the cost of violation and re-solve to try and get a trajectory of better and better nearly acceptable solutions that point to a possible overall solution.

4 Conclusion

The purpose of this article has been to make more visible an idea we call the local to global principle. This principle is an organizing tool useful both in designing and analyzing a wide variety of applications.

Essentially the whole point of this writeup is to set up enough framework to quickly write down a table of advice such as Table 2 (and for such a table to mean something).

Example	Local Step	Global Step
speech transcription	tables	Dynamic Programming
PageRank	graph structure, linear equations	Linear Algebra
machine learning	objective function	optimization

Table 2: Various Applications, Local Steps and Global Steps

The principle is not universal; not everything can be fit into such a table. For example the local to global decoupling is *not* a feature of the famous EM algorithm [DLR77], which depends on mixing predictions and corrections.

To conclude: the recipe is as follows. If you come to a problem with a large shopping bag of possible ways to build local criteria and powerful globalization procedures then you stand a very good chance of solving the problem quickly. Also, if you keep the local to global principle in mind you are more likely to identify and retain potential local tricks and globalizers when you see them and thus have a larger more nimble set of tools available to solve problems when the time comes.

References

[AC06] Nir Ailon and Bernard Chazelle, *Approximate nearest neighbors and the fast johnson-lindenstrauss transform*, STOC (2006).

[AI06] Alexandr Andoni and Piotr Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions.*

[BD02] Avrim Blum and John Dunagan, *Smoothed analysis of the perceptron algorithm for linear programming*, SODA (2002), 11.

[Bel57] Richard Bellman, *Dynamic programming*, Princeton University Press, 1957.

[BF97] Leo Breiman and Jerome H Friedman, *Predicting multivariate responses in multiple linear regression*, Journal of the Royal Statistical Society, Series B (Methodological) **59** (1997), no. 1, 3–54.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen, *Classification and regression trees*, Chapman & Hall/CRC, January 1984.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan, *Latent dirichlet allocation*, Journal of Machine Learning Research **3** (2003), 993–1022.

[BPH06] Kristin P. Bennett and Emilio Parrado-Hernandez, *The interplay of optimization and machine learning research*, Journal of Machine Learning Research **7** (2006), 1265–1281.

[Bre00] Leo Breiman, *Special invited paper. additive logistic regression: A statistical view of boosting: Discussion*, Ann. Statist. **28** (2000), no. 2, 374–377.

[BRS91] R Beigel, N Reingold, and D Spielman, *The perceptron strikes back*, Structure in Complexity Theory Conference **6** (1991), 286–291.

[Cha96] Eugene Charniak, *Statistical language learning*, MIT Press, 1996.

[Cha97] ———, *Statistial techniques for natural language parsing*, AI Magazine **18** (1997), no. 4, 33–44.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, MIT Press, 2009.

[CSS02] Michael Collins, Robert E Schapire, and Yoram Singer, *Logistic regression, adaboost and bregman distances*, Machine Learning **48** (2002), no. 1/2/3, 30.

[CV05] Rudi Cilibrasi and Paul M.B Vitanyi, *Clustering by compression*, IEEE Transactions on Information Theory **51** (2005), no. 4, 1523–1545.

[DF98] Rod G. Downey and M. R. Fellows, *Parameterized complexity*, Monographs in Computer Science, Springer, November 1998.

[DLR77] A P Dempster, N M Laird, and D B Rubin, *Maximum likelihood from incomplete data via the em algorithm*, Journal of the Royal Statistical Society, Series B (Methodological) **39** (1977), no. 1, 1–38.

[Fis36] Ronald A Fisher, *The use of multiple measurements in taxonomic problems*, Annals of Eugenics **7** (1936), 179–188.

[FS99] Yoav Freund and Robert E Schapire, *A short introduction to boosting*, Journal of Japanese Society for Artificial Intelligence **14** (1999), no. 5, 771–780.

[GD04] Peter D Grunwald and A Philip Dawid, *Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory*, Ann. Statist. **32** (2004), no. 4, 1367–1433.

[Gru00] PD Grunwald, *Maximum entropy and the glasses you are looking through*, Conference on Uncertainty in Artificial Intelligence (2000), 238–246.

[HNP09] Alon Halevy, Peter Norvig, and Fernando Pereira, *The unreasonable effectiveness of data*, IEEE Intellegent Systems (2009).

[Hus99] Dirk Husmeier, *Neural networks for conditional probability estimation*, Springer, 1999.

[IM99] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality.*

[Joa98] Thorsten Joachims, *Making large-scale svm learning practical*, Advances in Kernel Methods - Support Vector Learning (1998).

[Joa06] ———, *Training linear svms in linear time*, KDD (2006).

[Kar98] David R Karger, *Randomization in graph optimization problems: A survey*, Optima: Mathematical Programming Society Newsletter **58** (1998).

[KCVM04] Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew Kachites McCallum, *Interactive information extraction with constrained conditional random fields*, AAAI (2004).

[Kle97] Jon M Kleinberg, *Authoritative souces in a hyperlinked environment*, ACM SIAM Symposium on Discrete Algorithms (1997).

[Kom08] Paul Komarek, *Logistic regression for data mining and high-dimensional classification*, CMU CS Thesis (2008), 138.

[KWA95] J Kivinen, Manfred K Warmuth, and P Auer, *The perceptron algorithm v.s. winnowo: Linear v.s. logarithmic mistake bounds when few input variables are relevant*, COLT (1995), 289–296.

[Lew98] David D Lewis, *Naive (bayes) at forty: The independence assumption in information retrieval*, find journal (1998).

[LK73] S Lin and BW Kernighan, *An effective heuristic algorithm for the traveling-salesman problem*, Operations Research (1973), 498–516.

[Mar61] M E Maron, *Automatic indexing: An experimental inquiry*, RAND Technical Report (1961), 404–417.

[MF00] Zbigniew Michalewicz and David B. Fogel, *How to solve it: Modern heuristics*, Springer, 2000.

[Mil02] John Stuart Mill, *A system of logic*, University Press of the Pacific, 2002.

[Mit97] Thomas Mitchell, *Machine learning*, McGraw-Hill, 1997.

[MK00] M E Maron and J L Kuhns, *On relevance, probabilistic indexing and information retrieval*, 1960 (2000), 1–29.

[Mou00] John A Mount, *Automatic detection of potential deadlock*, Dr. Dobbs Journal (2000).

[Mou09a] John Mount, *Automatic generation and testing of un-rolls for profitable technical trades*, http://www.win-vector.com/blog/2007/10/paper-on-stock-trading/ (link), 2009.

[Mou09b] ———, *A demonstration of data mining*, http://www.win-vector.com/blog/2009/08/a-demonstration-of-data-mining/ (link), 2009.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Tery Winograd, *The pagerank citation ranking: Bringing order to the web*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1768 (link) (1998).

[Pol54a] G. Polya, *Induction and analogy in mathematics*, Princeton University Press, 1954.

[Pol54b] ———, *Patterns of plausible inference*, Princeton University Press, 1954.

[Pol71] ———, *How to solve it*, Princeton University Press, November 1971.

[RC96] Louis B Rall and George F Corliss, *An introduction to automatic differentiation*, SIAM: Computational Differentiation: Techniques, Applications and Tools (1996), 1–18.

[Rot97] Gian-Carlo Rota, *Indiscrete thoughts*, Birkhauser, 1997.

[Ski88] John Skilling, *The axioms of maximum entropy*, Maximum Entropy and Bayesian Methods in Science and Engineering **1** (1988), no. 173-187.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), no. 2.

[STC00] Jown Shawe-Taylor and Nello Cristianini, *Support vector machines*, Cambridge University Press, 2000.

[STC04] ———, *Kernel methods for pattern analysis*, Cambridge University Press, 2004.

[Str76] Gilbert Strang, *Linear algebra and its applications*, Academic Press, Inc., 1976.

[TH09] Jerome Friedman Trevor Hastie, Robert Tibshirani, *The elements of statistical learning: Data mining, inference and prediction*, Springer, 2009.

[TTV08] Luca Trevisan, Madhur Tulsiani, and Salil Vadhan, *Regularity, boosting, and efficiently simulating every high-entropy distribution*, Electronic Colloquium on Computational Complexity (2008), 18.

[Zei95] Doron Zeilberger, *The method of undetermined generalization and specialization illustrated with fred galvin's amazing proof of the dinitz conjecture*, http://arxiv.org/abs/math/9506215 (link), 1995.

A Acknowledgement

A thank you to readers who supplied help and comments on earlier drafts.