Intelligent Artificiality: Algorithmic Microfoundations for Strategic Problem Solving

Mihnea Moldoveanu
Desautels Professor of Integrative Thinking
Vice-Dean, Learning, Innovation and Executive Programs
Director, Desautels Centre for Integrative Thinking
Founder and Director, Mind Brain Behavior Institute
Founder and Director, RotmanDigital
Rotman School of Management, University of Toronto

Visiting Professor
Harvard University, Graduate School of Business Administration

Working Paper 19-072

e o
d B
e o
d B
e o

Abstract

This paper introduces algorithmic micro-foundations for formulating and solving strategic problems. It shows how the languages and disciplines of theoretical computer science, 'artificial intelligence' and computational complexity theory can be used to devise a set of heuristics, blueprints and procedures that can help strategic managers formulate problems, evaluate their difficulty, define 'good enough solutions' and optimize the ways in which they will solve them *in advance of attempting to solve them*. The paper introduces both a framework for the analysis of strategic problems in computational terms, and a set of prescriptions for strategic problem formulation and problem solving relative to which deviations and counter-productive moves can be specified and measured.

1. Introduction: Formulating and Solving Strategic Problems Using Computational Language Systems. Strategic problem formulation was recently brought into the focus of inquiry [Baer, Dirks and Nickerson, 2013] in strategic management, with calls for the development of 'microfoundations' that will help us make sense of the social and cognitive bases for defining problems before solving them. The upshot to such inquiry is both deriving prescriptive advice on *which* problem to try solving before 'plunging in' [Bhardwaj, Crocker, Sims and Wang, 2018] and providing a nuanced model of the individual and interpersonal choices involved in strategic problem solving. This emphasis on decoding the 'problem formulation process' has been applied to strategic and operational problem solving from multiple domains [Moldoveanu and Leclerc, 2015] as a generative blueprint for 'strategic innovation' that results from synthesizing spaces of possible solutions by choosing the language and ontology in which problems are framed – rather than simply generating solutions *within* a search space that is either 'given' through conversational habit. This article contributes a set of computational and algorithmic 'microfoundations' to the study of problem formulation and structuration. It leverages the abstraction (conceptualization, ideation and the structuration of thought and blueprinting of action) layers developed by computer scientists and complexity theorists over fifty years to provide both a way of studying and teaching problem formulation and structuration across strategic domains, and to generate a useful abstraction layer for strategic problem solving.

The field of problem definition and structuration has a rich and textured history in the field of artificial intelligence [Simon, 1973] where the need to encode a problem solving procedure as a set of algorithmic steps to be performed by a digital computer requires careful consideration of the definition of the problem (objectives, constraints, the space of possible solutions) and the structuration of the solution search process (accounting for situations in which the process of search changes the space of possible solutions). Even a simple and disciplined application of Simon's conceptualization of problem definition and structuration can significantly benefit the practice of strategic problem shaping and solving by providing strategists with ways of thinking about what they do when they 'think', 'talk', 'argue', and 'decide' [Moldoveanu and Martin, 2009; Christian and Griffiths, 2017; Valiant, 2006]. However, the practice of problem shaping and structuration has made strides in the field of 'algorithmics' (including machine learning and deep learning) In the intervening period [see, for instance, Sedgewick and Wayne 2011; Cormen, Leiserson and Rivest, 2011; Hromkovic, 2003] the field of 'algorithmics' has grown in depth, breadth and explanatory coverage to the point where it can ably provide a comprehensive description language and associated prescriptive prompts for intelligently coping with strategic problems.

Algorithmic thinking and computational thinking: Computational Foundations for Problem Formulation. [Wing, 2006; Wing, 2008; Wolfram, 2016]. The usefulness of the computational modeling toolkit to disciplines outside of computer science has not gone un-noticed. Wing [2006; 2008] posited 'computational thinking' as a kind of analytical thinking that allows people working in any discipline to structure their problem solving processes more productively by deploying useful abstractions and methods used by computer scientists and engineers to solve large scale problems that usually exceed the computational powers of individual humans. She argues that problem solving routines that lie in the algorithm designer's toolbox (such as recursion) are useful to problem solvers in all domains – and not just to programmers coding solutions to algorithmic problems: it is a 'conceptualization skill' and not just 'a rote mechanical skill'. Both Wing and Wolfram [2016] focus on 'abstraction' and the use of 'multiple abstraction layers' as critical to the discipline of computational thinking, which, once mastered, can be deployed across domains as varied as linguistics, genomics, economics, cognitive science and sociology. *What about strategic management*?The task we set for ourselves in this paper is to come up with a *computational abstraction layer* for business problems that helps both strategic managers and researchers 'cut across domains of practice' and on one hand generate a set of algorithmic micro-foundations for strategy problem solving and on the other offer insights and guidelines to practitioners engaging with live, raw, ill-defined, ill-structured, potentially intractable problems.

'Intelligent Artificiality': Learning from Machines to Solve Problems Machines Won't Solve. To help shape perceptions about the project of building computational microfoundations for strategic problem solving, we reverse the phrase 'AI' and posit 'Intelligent Artificiality' (IA)

as the practice that human problem solvers in general and business problem solvers in particular to appropriate and deploy algorithmic and computational methods to problems in their own domains. The project of intelligent artificiality is summarized in Figure 1. The loosely-worded, 'fuzzy' problems that strategic managers often use to encode their predicaments (top layer) are defined, calibrated and classified to yield 'stylized business problems' (second layer from top), which are then mapped into a set of canonical problems (third layer from top) framed in a language amenable to algorithmic treatment. They inform the search for solution-generating processes – algorithms, heuristics – tailored to the structure of the problem to be solved. The analysis protocol can also be run backwards, to generate coherent and precise problem statements for business strategists using templates and models drawn from the canonical repertoire of problems in computer science.

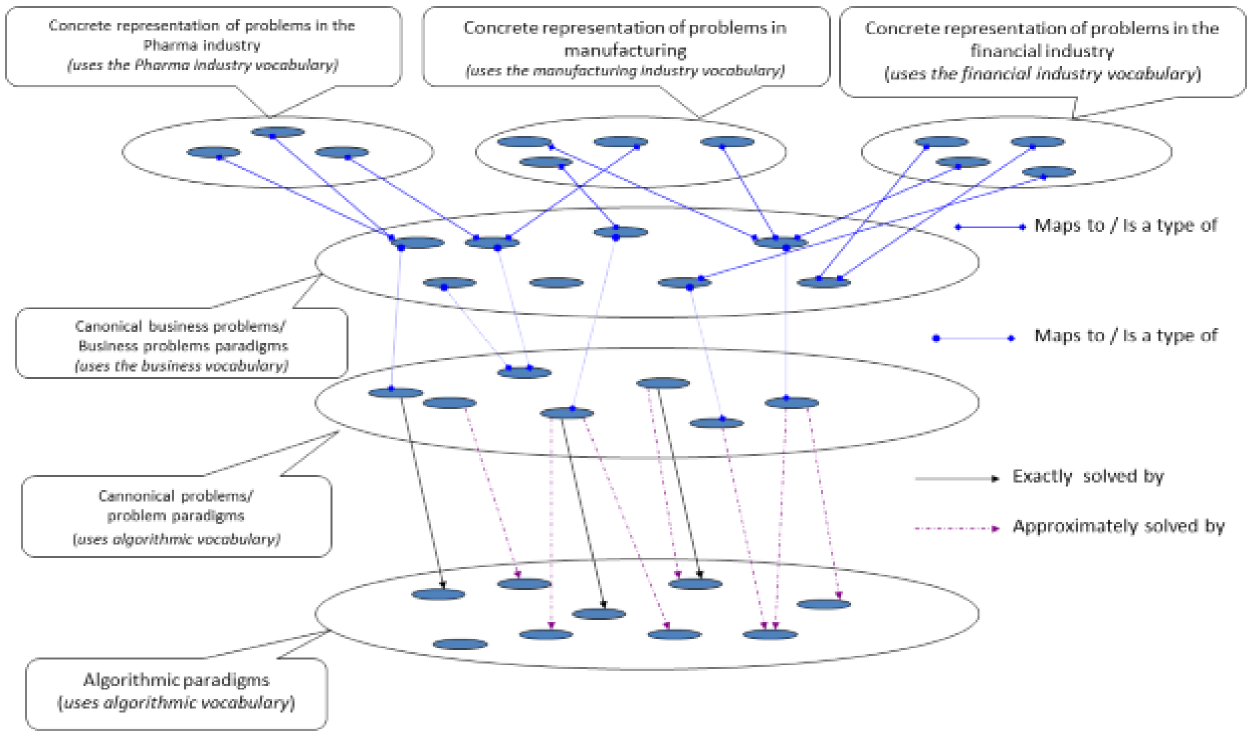

Concrete representation of problems in the Pharma industry (uses the Pharma industry vocabulary)
Concrete representation of problems in manufacturing (uses the manufacturing industry vocabulary)
Concrete representation of problems in the financial industry (uses the financial industry vocabulary)
Maps to / Is a type of
Canonical business problems/ Business problems paradigms (uses the business vocabulary)
Maps to / Is a type of
Cannonical problems/ problem paradigms (uses algorithmic vocabulary)
Exactly solved by
Approximately solved by
Algorithmic paradigms (uses algorithmic vocabulary)

Figure 1. A synoptic view of the process by which 'raw' problems in business are defined and encoded as canonical business problems and subsequently as canonical algorithmic problems for which highly efficient solution search procedures can be found.

For example, the loosely worded challenge ('We need to resolve a major personal accountability problem at the level of the top management team') can be defined and structured more precisely and actionably as ('Re-allocate decision rights to members of the top management team so as to increase a composite metric of accountability (comprising both results and self-reports) by 50% over the next two quarters, maintaining current levels of incentives and without firing anyone'), which then maps into a canonical problem (MAXSAT: allocate variables to clauses (decision rights to agents) so as to maximize a global objective function) for which highly efficient search procedures from the field of algorithm design (divide and conquer, local neighborhood search) can be applied to solve the problem far more efficiently than would be possible via experimentation or even 'offline' simulation.

We proceed by introducing computational micro-foundations for defining and structuring strategic problems and show how this representation allows us to guide the structuration of strategic problems and resgister departures from optimal or non-dominated structuration strategies. We introduce problem calibration as the process by which strategists specify what constitutes a 'good enough' solution, and introduce a typology of (well-defined problems) that allows strategists to productively parse and optimize the solution process. We show how *sizing up* a problem by measuring the best-case, average-case and worst-case complexity of its solution procedure can help strategists estimate – in advance of solving a problem or just 'plunging in' – the time and resources a solution will likely require. We discuss solution search processes for problems of different complexity – and provide a map by which strategic problem solvers can choose solution search procedures best suited to the structure and complexity class of their problems. Finally, we work out the implications of the algorithmic micro-foundations for strategic problem solving to the design and management of *problem solving teams and groups*.

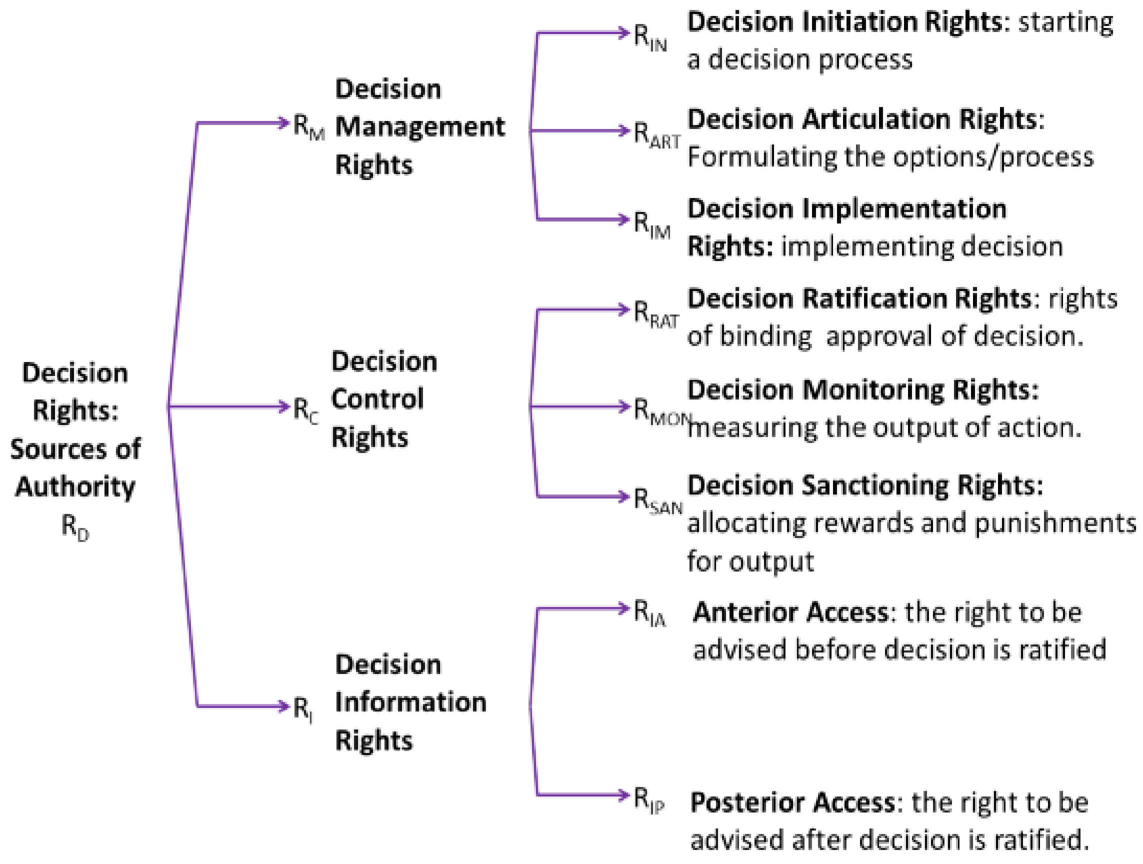
2. ***What*** **is the Problem and** ***What Kind*** **of Problem is It? Problem Definition, Structuration and Calibration.**

A significant gap in the literature on problemistic search that originated in the work of Cyert and March [1963] is a focus on problem definition and formulation as precursors to the adaptive selection of behavioral solutions to problems arising from mismatches between aspirations and results. As a recent review [Posen, Keil, Kim and Meissner, 2018] points out, attempts to address this gap must grapple with constraints of resources and rationality – and that is what computational modeling techniques do, by design: algorithm designers must deal, methodically and in advance of 'solving the problem' with the memory, computational and interface (I/O) constraints of the device they are planning to run the executable version of the code that embodies the solution search procedure chosen.

Defining Problems. Much of the literature on strategic problem solving starts by assuming that problems are 'given' and problem statements are 'self-evident' – but this is one of the assumptions that causes the 'problem formulation gap' in the problemistic search field – and which can cause strategists to simply 'plunge in' to solving a problem before carefully defining the problem statement. By contrast, the algorithmics literature focuses narrowly and on *defining* problems in ways that admit of systematic searches for solutions under time and resource constraints, and using as much as prudent but no more than available resources for storing information and performing computational operations [Simon, 1973; Cormen, Leiserson and Rivest, 2011; Hromkovic, 2003].

A well-defined problem [Moldoveanu and Leclerc, 2015] minimally involves the specification of a mismatch between current and desired conditions and the time required to move from the current to the desired conditions. This requires a specification of (observable and measurable) current and desired conditions – along with their methods of measurement. For example, *defining* the 'fuzzy problem': *'We have a major accountability problem within the top management team'* can be accomplished by first specifying the metrics for current and desired conditions ('accountability' as measured by average response time to mission critical emails to other management team members; percentage of individual commitments to the rest of the team fulfilled as measured by minutes to management team meetings; responses to survey of management team members and their direct reports on the degree to which a 'commitment culture' that promotes making sharp promises and keeping them or breaking them with advance notice and with good reason exists) and then specifying the percentage improvement in the overall metric sought (eg 50%), the time frame in which the improvement must be made (eg 6 months) and sharp constraints on changes that must be heeded (no change to membership of the team; no change to incentive structure; no change to schedule of top management team meetings). The loosely worded problem can be phrased as a well-defined problem statement, eg: 'We are seeking to change a composite accountability metric involving response times, percentage of individuals commitments to the rest of the team fulfilled and the mean of subjective reports on the degree to which the organization has a culture of commitment by 50% over the next 6 months, subject to maintaining current team membership and compensation structures and the current schedule of executive team meetings.'

While discussions of problem formulation and problem framing in the management field [Mintzberg, Raisinghani and Theoret, 1976; Lyles and Mitroff, 1980 – and the literature they spawned] stop at the specification of desired conditions, the algorithmic problem solving field additionally requires us to use a precise *language* system for stating the problem – one that allows algorithmic search procedures to be deployed on the solution search space generated by the language system in question. Because in computer science problems the language system is pre-determined (e.g. by a programming language)– much of the literature on problem formulation either does not focus on the language system or proceeds directly to the enumeration of a solution search space [Simon, 1973]. If we expect a well-defined problem to generate a solution search space, however [Moldoveanu and Leclerc, 2015], then we must choose a language system that will enable choices ('possible solutions') over variables that are observable and controllable. Given the constraints of the problem in this case, we can focus on the *decision rights* of members of the top management team (See Figure 2 below, based on the partitioning of decision rights in [Moldoveanu and Leclerc, 2015, which builds augments the partitioning of decision rights introduced in [Jensen and Meckling, 1998(1995)]) and trying to *re-allocate* decision rights over key decision classes (hiring, sales, budgeting, business development) so they encourage consistent and timely *reporting* and *information sharing* between management team members.



Decision Rights: Sources of Authority R_D
R_M Decision Management Rights
R_C Decision Control Rights
R_I Decision Information Rights
R_{IN} Decision Initiation Rights: starting a decision process
R_{ART} Decision Articulation Rights: Formulating the options/process
R_{IM} Decision Implementation Rights: implementing decision
R_{RAT} Decision Ratification Rights: rights of binding approval of decision.
R_{MON} Decision Monitoring Rights: measuring the output of action.
R_{SAN} Decision Sanctioning Rights: allocating rewards and punishments for output
R_{IA} Anterior Access: the right to be advised before decision is ratified
R_{IP} Posterior Access: the right to be advised after decision is ratified.

Figure 2. Eight distinct kinds of decision rights that one might allocate to members of a management team.

Moreover, given this representation of the problem, we can specify and *enumerate* a space of possible solutions: given *N* members of the team, each of whom can have (code as a '1') or not have (code as a '0') a specific kind of decision right over decisions falling into one of *D* classes, we can generate a solution search space of (2^{8N}-1) (number of distinct allocations of 8 kinds of decision rights to each of N managers, excluding (-1) the 'no decision right to anyone' allocation)) x *D* (kinds of decisions), and either exhaustively or selectively *evaluate* possible allocations with respect to their likely consequences to the accountability of the group as a whole.

Defining the problem in this way gives a further refinement to the problem statement we have synthesized, namely: 'We are seeking to change a composite accountability metric involving response times, percentage of individuals commitments to the rest of the team fulfilled and the mean of subjective reports on the degree to which the organization has a culture of commitment by 50% over the next 6 months, subject to maintaining current team membership and compensation structures and the current schedule of executive team meetings *by re-allocating the decision rights of top management team members over 2 different decision classes (strategic, operational) to promote (a) denser information sharing on critical decisions and (b) greater alignment between the specific knowledge of each team member and her level of authority over decisions requiring the use of that knowledge .'*

Structuring problems. As Simon [1973] points out, problems need not only be well-defined, but also well-structured (i.e. not ill-structured) for them to be reliably solvable by an algorithmic procedure. An ill-structured problem is one whose solution search space changes as a function of the specific steps we take to solve the problem. Strategic problems have a much greater risk of being ill-structured on account of the Heisenbergian uncertainty [Majorana, 2006 (1937); Moldoveanu and Reeves, 2017] implicit in social and human phenomena: The act of measuring a particular variable relating to a social group ('commitment', 'cohesion', 'motivation) can influence (increase or decrease) the values of the very variables that we are trying to measure. In the context of solving a problem like *P:* 'We are seeking to change a composite accountability metric involving response times, percentage of individuals commitments to the rest of the team fulfilled and the mean of subjective reports on the degree to which the organization has a culture of commitment by 50% over the next 6 months, subject to maintaining current team membership and compensation structures and the current schedule of executive team meetings by re-allocating the decision rights of top management team members over 2 different decision classes (strategic, operational) to promote (a) denser information sharing on critical decisions and (b) greater alignment between the specific knowledge of each team member and her level of authority over decisions requiring the use of that knowledge ' – the very act of jointly measuring both current allocations of decision rights among members of the top management team *and* the degree to which a commitment culture exists in the organization (via self-reports and surveys, for instance) can influence the estimate of both sets of variables. Members of the top management team may *under-report or over-report* their own decision rights, depending on the degree to which they believe they will lose them by any proposed solution; or strategically distort their responses to surveys meant to supply information to be used in an 'organizational accountability' metric in order to mitigate the degree to which a radical restructuration of their authority will be brought about.

The Heisenbergian uncertainty that accrues to the definition of strategic problems and requires us to *structure* them is pervasive in strategic problem solving scenarios. For instance, the problem: "*How do we increase cumulative research and development productivity by 10% over the next 24 months subject to not increasing R&D spend*?" can induce distortion of information required to estimate *current* R&D productivity. The problem: "*How do we decrease sales cycle duration by a third over the next quarter subject to maintaining the membership of the current sales force?"* can induce distortions in the reporting of information used to estimate both *current* sales cycle ("what constitutes a 'closed' sale?") and realistically achievable *target* sales cycle duration. *Structuring* strategic problems, therefore, is essential to *decoupling* the process by which the *problem is defined and a solution search space is generated* from the problem by which *initial conditions are measured and desired conditions are specified and measured*. Strategic managers can attempt to *structure* problems by at least three different classes of moves:

'Always-on measurement': In the age of 'big data' and ubiquitous observation of behaviors and even micro-behaviors, they can deploy a fabric of organizational measurement systems and platforms that constantly track large sets of variables that are useful to the solution of *broad* classes of problems, which gives the top management team discretion as to *which* problem to attempt to solve at any one point in time. In our examples, variables relating to levels and degrees of authority of top management team members, to sales cycle duration and current R&D productivity can be constantly maintained in regularly updated databases which can be mined for composite measures aimed at defining organizational change problems.

'Unobtrusive measures'. Strategic managers can also deploy unobtrusive and indirect measures of the values of the variables that are directly relevant to a strategic problem. They can use accounting data and online behavior to monitor sales cycle dynamics; or, email spectroscopy and mining of exchanged messages to figure out average and median response times to sensitive communications. Unobtrusive measures may allow strategic managers to address the Heisenbergian uncertainty of targeted, transparent measurement by cutting off the inferential chain which motivates those supplying the data to distort it in order to influence the solution that will be adopted.

'*Outsourcing of strategic problem solving'*. Large strategy consultancies can function as effective 'insulators' that help strategic management teams solve problems. Their engagements are often phrased in broad terms (see Table 1 in the Appendix), which allows them considerable latitude in probing, inquiring and measuring variables of interest in advance of announcing the problem to be solved – thus effectively decoupling the process of measurement from that of problem definition and solution search space enumeration. This suggests that the value of strategic consultancies as Heisenbergian uncertainty minimizers is independent of their value as framers, shapers and solvers of strategic problems.

Calibrating Problems: What is a Good Enough Solution? Core to the problemistic search literature is the notion of *bounded rationality* of problem solvers [Simon and March, 1958]. The search for solutions is conceived as narrow, local and sparse [Nelson and Winter, 1982]. This rudimentary conception of organizational problem solving closely tracks that of the mainstream economic analysis of optimization [Dixit, 1990], wherein value maximizers search – and evaluate the gradient of the payoff curve – in the close proximity of their current conditions. This approach significantly misses significant sources of heterogeneity in the deployment of problem solving prowess, and fails to distinguish between and among different *kinds* of bounds to rationality.

By contrast, computational science makes sharp distinctions among informational bounds (memory: how much can you store?), computational bounds (how quickly can you calculate?) and communication bounds (how quickly can you communicate and coordinate with others in a multiprocessing environment?) which are just as important in organizational settings in which useful data is scarce and expensive to produce, the time and resources available for systematically eliminating dominated solutions is scarce, and problem solving is increasingly performed in groups and collectives that require constant communication to 'stay on the same page'. A computational abstraction layer for strategic problem solving allows us to study the challenge of *selectively and purposefully bounding and deploying rationality*, summarized by the question: *When is it worth trying to solve this problem to this level of accuracy?* Computational science teaches us that *much depends on having the right algorithm* - or, problem solving procedure, at hand.

To take a simple example, suppose a strategic manager faces a decision between two lotteries. The first pays *$1MM* no matter what (once one chooses it). The second costs *$1MM* to participate in, but pays *$10MM* if the 7th digit of the decimal representation of the square root of *2* is the number *7*, and *$0* otherwise (leading to a *$1MM loss*). Without (any digital computer or) knowledge of a method for computing the decimal square root of 2, the expected value of the second lottery is *$0*: a one in ten chance (*there are 10 digits*) of guessing at the number correctly and realizing the $10MM gain, minus the cost outlay of $1MM for participating in the lottery. However, if the decision maker knows 'Newton's Method' (actually known to the Babylonians much earlier) for calculating an arbitrarily large number of *digits* in the square root of 2, then she can do much better than guessing. The method uses the first terms of a Taylor series expansion of $f(x) = x^2-2$ to generate a *recursive* set of approximations to $\sqrt{2}$: Starting with an initial guess x_0, successive iterations are generated via: $x_{n+1} = x_n - f(x_n)/f'(x_n)$, which, for $f(x)= x^2-2$, gives: $x_{n+1} = x_n-(x_n^2-2)/2x_n$.. The method not only allows one to exactly calculate the tenth digit of the decimal representation of the square root of 2 (and therefore the decision maker to realize a net gain of *$9MM* over the expected value of a decision maker who simply guesses), but it also allows her to estimate the expected informational value of each additional calculation that generates a more accurate answer, i.e. $x_{n+1} = x_n - f(x_n)/f'(x_n)$ (which in this case is 2 bits/iteration).

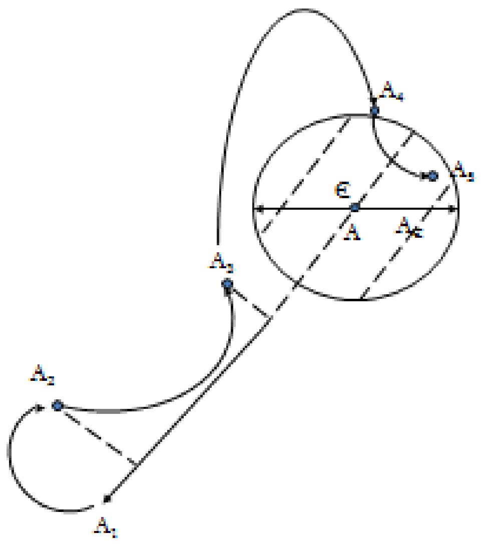
1

1.5

1.416666666666666666666666666666666666666666666666666666666666675
1.4142156862745098039215686274509803921568627450980392156862745
1.4142135623746899106262955788901349101165596221157440445849057
1.4142135623730950488016896235025302436149819257761974284982890
1.4142135623730950488016887242096980785696718753772340015610125
1.4142135623730950488016887242096980785696718753769480731766796

Figure 3. The first eight iterations in the recursive application of Newton's Method to the calculation of the square root of 2 to an initial guess of '1' as the answer.

Recursion is useful not only because it can be used to replace a difficult and cumbersome calculation with a set of easier calculations [Wing, 2006], but also because, importantly, it allows the problem solver to estimate the expected value of turning the recursion crank one step further in advance of doing so (Figure 4).



"A" represents the exact answer, ϵ is the diameter of an acceptable "error region" around A. $\{A_k\}$ are successive approximations of "A", outputs of successive iterations of the solution algorithm.

Figure 4. Graphical representation of the information dynamics of a recursive estimation procedure.

Strategic Equilibria. Recursive estimation can play a useful role in interactive reasoning of the type that allows two oligopolists selling un-differentiated products with uniform marginal cost of *c* into a price-taking market with downward sloping demand curve parametrized by *a-c* (where *a* is a constant), by allowing each supplier to reason its way recursively down a hierarchy of iterative 'best responses', whose generating function is [Saloner, 1991]: $q_0=0; q_{n+1}=(a-c)/2 - q_n/2$, that generates the series shown in Figure 5. Importantly, the strategic managers in each firm can calculate not only their best response to the product quantities the competitor chooses to produce, but also the *loss or gain* they stand to make by thinking to the next level of recursion (or, by thinking their way to the Nash Equilibrium).

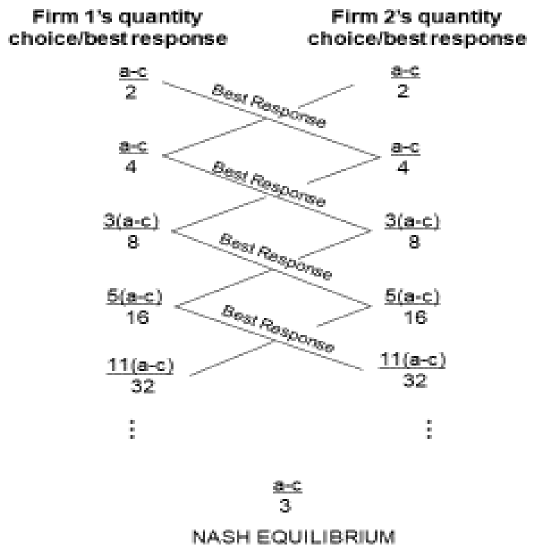


Figure 5. Iterative 'best responses' of two duopolists selling an undifferentiated product costing c *into a market with a downward sloping demand curve parametrized by* a-c.

3. **How *hard* is the problem we are about to try solving? *Prior Analytics for Problem Complexity.*** An important contribution that a computational abstraction layer for strategic problems contributes to the way in which bounded rationality is treated in the strategic problem formulation and problemistic search literature is a way of *sizing the problem* – figuring how hard it will be and what resources will be required to solve it, before trying to solve it. 'Plunging in' and just trying to solve a business problem without a sense of its definition or structure is often counterproductive, on account of 'dead ends', coordination failures among multiple problem solvers, or simply running out of time and resources while still trying to solve the problem but before even an approximate solution or solution concept has been derived [Moldoveanu, 2011]. Postulating 'bounded rationality' and 'muddling through' problems as catch-all explanations for sub-optimal problem solving behaviors misses the cases in which strategic managers optimize and ration their own strategic thinking according to the expected value of a good-enough solution [Moldoveanu, 2009] or devise "strategies for coming up with a strategy" [Reeves, Haanaes and Sinha 2015], deploying computational prowess and 'optimization cycles' to solve the problems most worth solving,

to the level of accuracy required for the situation at hand. Prescriptively, positing a *uniformly* bounded rationality – as a constitutive condition for a problem solver misses out on opportunities for optimizing the *ways* in which computational prowess and intelligence may be deployed by a strategic manager to achieve better solutions to problems deemed 'too difficult' or 'intractable'.

By contrast, algorithm designers often *start* their problem solving work by forming estimates of the *difficulty* of a problem and the degree to which it can be solved using the computational and memory resources available to them [Cormen, Leiserson and Rivest, 2011]. They think through the worst-case, best case and average case complexity of searching a database for a name *before* actually beginning to code the algorithms or designing interfaces to the database, in order to determine whether or not the search procedure will generate an answer in the appropriate time for the end user. To make sure that their solution is scalable to databases or various sizes, they generate estimates of the *running time* $(T(n))$ of an algorithm as a function of the number of entries in the database n; and they derive upper and lower bounds on the run time of the algorithm as a function of the number of independent variables n by examining the behavior of $T(n)$ as n increases.

Suppose the problem we are trying to solve is that of *ordering* a list of numbers that initially show up in random order (initial conditions) so as to produce a list in which the same numbers appear in *ascending order*. An intuitive but systematic method for doing so is to read each number (left to right), compare it with each of the number(s) to the left of it, and then *insert* it *to the left* of the smallest number it has been compared (Figure 6).

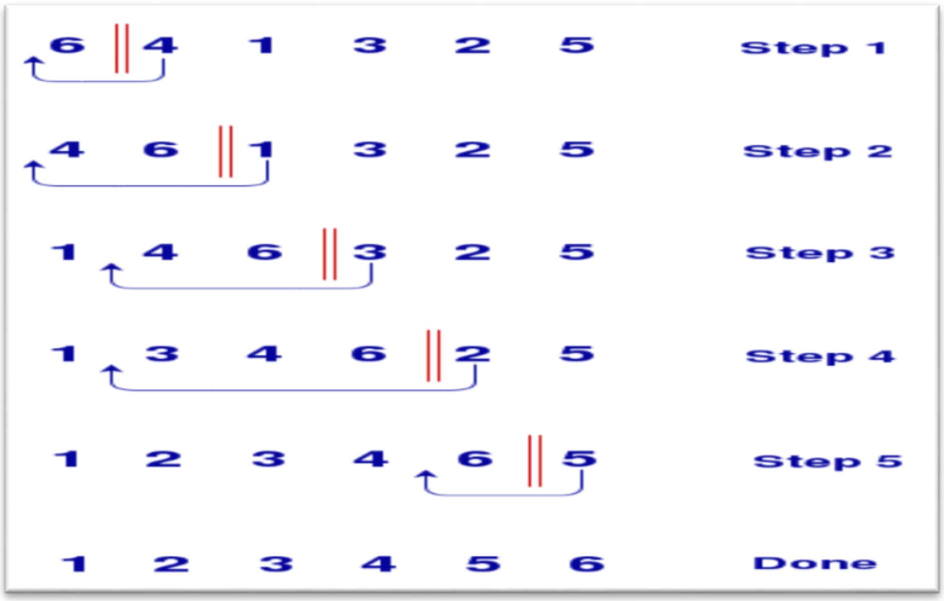


Figure 6. Elementary steps for sorting a list of numbers (low to high) using the INSERTION SORT procedure.

To figure out *how hard* the sorting problem is to solve using INSERTION SORT, we can evaluate how *costly* the algorithm is to run, by listing its elementary operations (left column, below) assigning a cost to each operation (middle column) and counting the *number of times* that each operation has to be performed (right column).

```
i ← 1
while i < n                                  c1      n-1 times
   j ← I                                     c2      n-1 times
    while j > 0 and A[j-1] > A[j]            c3      n-1 times
        swap A[j] and A[j-1]                 c4      n(n-1)/2 times
        j ← j – 1                            c5      n(n-1)/2 times
    i ← i + 1                                c6`     n-1 times
```

Figure 7. 'Sizing up' the costs of running the sorting procedure INSERTION SORT on a list that is n *numbers long by counting the total number of operations and the unit cost per operation. The unit cost is usually in units of TIME that the hardware on which the code embodying the algorithm runs normally takes to complete an operation of the kind specified in the left-most column*

With these estimates, an algorithm designer can figure out if the algorithm works *quickly enough* randomly ordered lists of desired length to produce ordered lists in the time that a user of the code would deem acceptable (simply by taking the cost-weighted sum of all of the operations). To form more *robust* expectations of the cost of the algorithm when the size of the input list is unknown, the designer might perform an *asymptotic analysis* of the rate at which the cost of running the code grows as a function of the number of entries n in the list. By inspection of Figure 7, we see that the n^2 terms in the cost function, in the limit as n becomes large will grow far more quickly than the terms that vary with n alone (eg. $n-1$). Asymptotically, an upper bound on the growth of $T(n)$ as n grows will be n^2, which can be represented by saying that $T(n)=O(n^2)$ ('is of the order of n^2).

Applying the *sizing* of problems to the space of strategic challenges requires we understand the basic operations we take to be standard in algorithm design (listing, comparison, selection, addition, etc.) will be *different* and have *different marginal cost structures* than those performed by humans. A strategic management meeting aimed at creating a *short and ordered list* of strategic investment projects may proceed by first pre-ranking the projects based on electronically polling the entire management team, distributing the preliminary ranking to the management team members, securing agreement on the process by which elimination of projects will proceed, and then (pending approval of this particular process) identifying a 'minimum score' that a viable project must have, identifying the projects that clear that hurdle, and (if they are too many or their costs exceed the budget available) discussing the costs and benefits of each of the n lowest ranked projects before re-ranking them and eliminating the $n-k$ lowest. Although the specification of this *solution procedure* falls short of specifying operations at the same level of precision as that of INSERTION SORT, it nevertheless allows managers to (a) evaluate the likely costs (time, probability of conflict, number of online and offline meetings) required for each step, (b) evaluate the costs of implementing alternative deliberation and selection/ranking procedures and (c) examine the behavior of the implementation costs of any chosen procedure as a function of the number of projects, the number of decision makers, and the distribution of scores of the projects after an initial round of evaluations by decision makers.

Just as with the implementation of an algorithm on a computer the costs of *enumerating* possible solutions are different from the costs of *evaluating* and *ordering* solutions in terms of desirability, so in strategic problem solving we can have different kinds of costs for each elementary operation, depending on the kind of evaluation we would want to perform on candidate solutions. In the case of the problem maximizing joint commitment and collaboration by the re-allocation of *8* kinds of decision rights to *N* management team members over *3* different classes of decisions, we might see out a solution by the following procedure:

ENUMERATION: specify the full set of decision right allocations to each of the members of the management team. [This will require $3 \times 2^{8N}-1$ low cost operations.]

COARSE GRAIN ELIMINATION: Eliminate the solutions that are obviously non-functional (like: giving no ratification rights to any decision maker (a factor of 2 reduction in the space of solutions), giving no initiation rights to anyone (another factor of 2), giving no monitoring rights to anyone (factor of 2), giving no sanctioning rights to anyone (factor of 2) and giving no prior and posterior information rights to anyone (factor of 4). [This entails the again very low cost of listing the 2^{6} dominated solutions and crossing them off the list].

COARSE GRAIN EVALUATION: Break up the resulting solutions into a number (ranging from 1 $3 \times 2^{8N-6}-1$) of *classes* of viable solutions that can be ordered in terms of their attractiveness (eg: not everyone should have ratification rights on every decision in this class; at most 1 executive should have sanctioning rights, etc.) – which will require at least 1 and at most $3 \times 2^{8N-6}-1$ different *evaluations* – which can be low cost (performed by the CEO and requiring mental simulation work, or high cost (performed by gathering industry data that supports a causal inference from decision right re-allocation to enhancements in accountability);

FINE GRAIN EVALUATION. Choose up to *K* different solutions from the classes of viable solutions using a decision rule (eg 1 per each different class; the most promising solution in each class; the most promising *K* solutions, cutting across classes, etc) and carefully evaluate their benefits and costs *vis a vis* the overall objective function. This will require at least 1 and at most *K* different operations, which may be lower (if performed by the CEO) or higher (if performed by the CEO in collaboration with the top management team, which will entail coordination and deliberation costs).

FINAL SELECTION: Evaluate each of the *K* solutions deemed most promising, rank them according to desirability, select a final decision right allocation to be implemented for the top management team. This will require at least 1 and at most *K* different operations, which may be lower (if performed by the CEO) or higher (if performed by the CEO in collaboration with the top management team, which will entail coordination, deliberation and 'debacle' costs.

To make the use of the computational abstraction layer to such problems more transparent, it is important to notice that while the basic *operations* of an algorithm like INSERTION SORT have been defined and fixed (by programming languages, machine languages and hardware architecture), the basic operations involved in solving a problem like REALLOCATE STRATEGIC DECISION RIGHTS can be defined in ways that seem most natural to the problem solver. *Enumerating* solutions can be done in different ways (mechanically, lexicographically) by different people, as can *simulating* the likely effects of a particular re-allocation and *evaluating* the costs and benefits of that re-allocation. The point of making the *procedure* by which the problem is solved explicit is to allow the strategic an executive to *visualize* the sequence of operations required, to *measure* alternative ways of solving the problem (by changing operations or sequences of operations) and to *estimate* the time and resources required to achieve a comprehensive solution to the problem.

As Porter and Nohria [2018] show, the ways in which CEO's in large, complex organizations allot and apportion their time may exhibit significant levels of *X-inefficiency* [Leibenstein, 1966: inefficiency arising from sub-optimal cross allocation of an important resource (time in this case) to tasks having different expected values] arising from uncertainty about how much time to allocate to different activities, or, alternatively, to how long each of a set of activities required to bring about a particular outcome will take. Their research produced an exceptionally valuable transversal picture of time allocation by CEO's to different classes of activities (Figure 8). However, what is needed in order to effectively re-structure time allocations is a *longitudinal picture* of time allocation that lays bare the sequences of activities that are linked together to produce an outcome (eg: a series of emails set up to motivate a meeting at which everyone shows up mentally present and fully prepared to make a decision by a deliberation process all agree with).

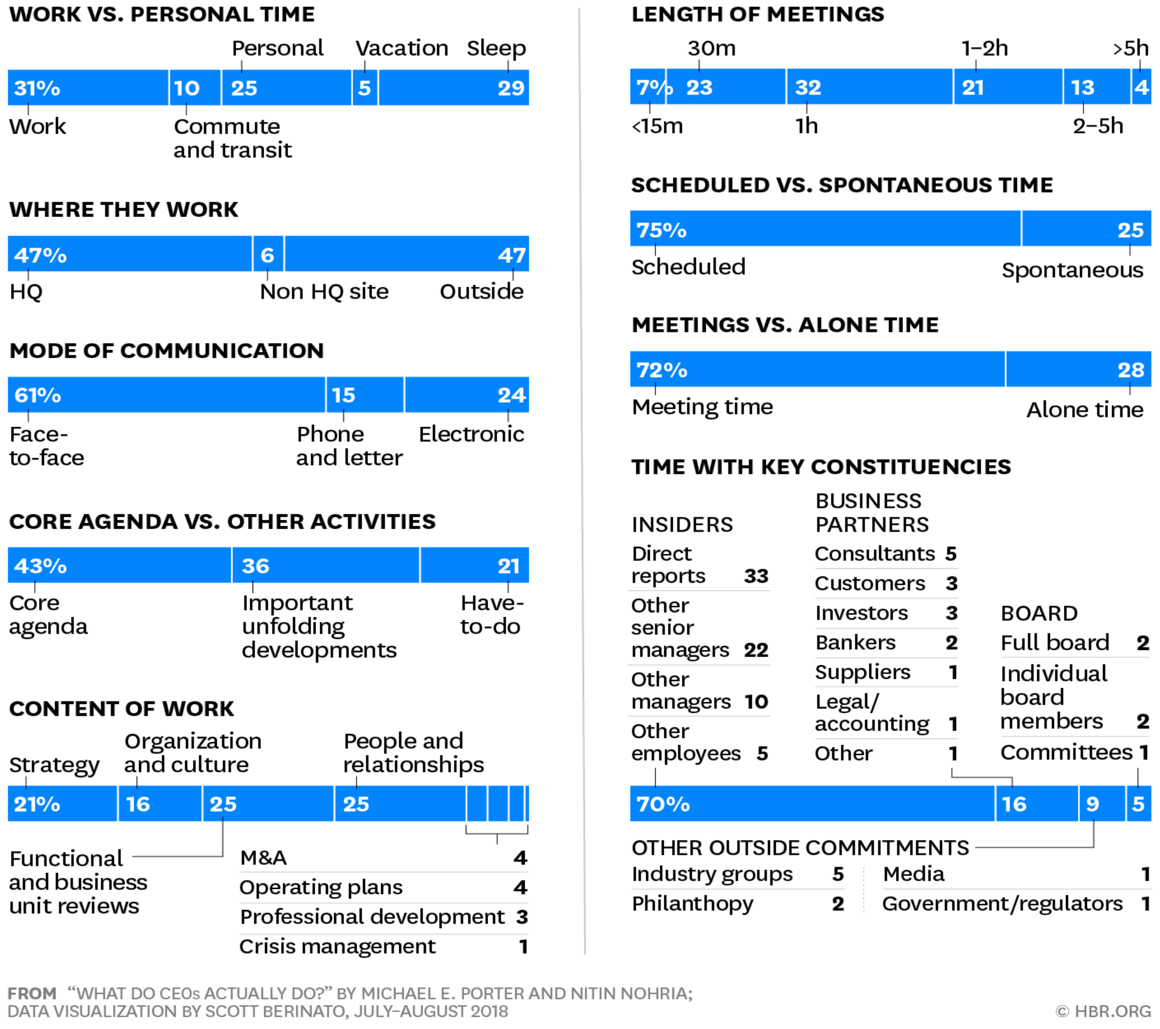


Figure 8. X-inefficient? Tabulation of relevant dimensions of the evidence on the allocation of time by CEO's (from Porter and Nohria, 2018).

A computational abstraction layer can be used to specify both the basic set of *organizational* operations (meetings, preparation, scheduling) required to carry out a task (a strategic re-structuration of levels of authority) and to create a bottom-up estimate of the costs of these operations (in time) – which in turn allows the strategic planner to choose activity sequences on the basis of an understanding of their *real cost structure*, and to optimize the cost of each operation. An operational cost approach to planning a strategic re-allocation of decision rights might proceed by creating a set of bottom-up estimates of the basic operations involved in carrying out the organizational task, and then (a) to plan the sequence of operations on the basis of a tally of the current estimate of the time-costs of each operation and (b) to invest in *reducing* the costs of each operation and the degree to which these costs scale with the number of whose agreement is required for their successful completion.

Operational Unit	***Cost of operation***	***Estimate of How Cost Scales with N (Multiplicative factor)***
Securing agreement for a meeting from N *participants*	2 days	N^2
Scheduling the meeting	2 days	N^2
Preparing agenda and pre-work materials	3 hours	1
Answering pre-meeting questions and socializing answers	2 hours	N
Deliberating on Alternatives	1 hour	*0.1 x N*
Securing Principled Agreement of preferred option	0.5 hour	*N*

Table 1. Sample operation-wise estimation of the time complexity of a requisite sequence of tasks (operations) required to carry out a strategic re-organization of decision rights

The computational analysis of business problems makes clear why and how *modularization* helps make problem solving more efficient and organizations more proficient at solving certain kinds of problems. In the example of INSERTION SORT, 'costing out' each operation – highlights the benefits of creating algorithmic *modules* that perform these operations very quickly, thus decreasing the marginal cost of the loop that uses them, and thus the overall cost of ordering an entire list. In the case of a strategic problem like DECISION RIGHT REALLOCATION, it may be that certain kinds of operations (eg.: 'getting together a meeting of people who will all have read the pre-work materials and will show in time minded to address the task at hand', or 'evaluating the coordination costs to a team of a re-allocation of authority') can become 'multiple re-use' modules whose marginal costs can be decreased through learning based on repetition and the incorporation of feedback, leading to a much faster implementation of strategic transformation.

Designing Google. Sizing a problem can be critical to the design of a *strategic solution* to a market problem – as the case of the design of Google centrality indicates. The 'ill-defined problem' of 'ordering the Web' just before the founding of Google was to produce a rank-ordering or semi-ordering (in terms of relevance, salience, footprint, impact, known-ness, etc) of the ~ 10^9 www pages on the Internet. To turn it into a well-defined problem [Moldoveanu and Leclerc, 2015] represent the Internet as a (directed, not fully connected) network in which web pages are nodes (n in number) and the links between them are the links (m in number) from one page to another that a surfer might follow in 'surfing the Web'.

Now the problem can be *defined* as that of ordering all web pages according to measures of their centrality (like between-ness, Bonacich, closeness) in the graph that represents the network of the Web, within a time that makes this measure relevant (24 hours) and using the available computational resources of the company at the time (defined in GBytes of RAM and GFLOPS (billions of floating point operations per second) of computational speed. The problem was also well-structured, as unobtrusive crawlers jumping from one site to another were unlikely to affect the behavior of web site owners and developers in ways that changed the problem on the time scales envisioned.

The challenge to solving the problem can only be revealed through *sizing the problem* by figuring out *how many operations are required to compute the centrality measure* of different nodes such that this measure will order the *World Wide Web*. To do so, we need O(*n, m)* estimates of the rate at which the complexity of the problem grows with the number of web pages *(n)* and links between them *(m)*. When we do so, however (Figure 8) we find that the number of operations required to compute betweenness, closeness and eigenvector (Bonacich) centrality measures for a large network grows *very quickly* as a function of the number of nodes and edges in the network (web pages and hyperlinks between them), and that the computational complexity of making these calculations exceeds that which can be accommodated on available (1998) resources during short 'refresh' cycles (a few hours*)*.

Motivated in part by addressing the computational difficulty of synthesizing a centrality-based relevance or importance measure for all available Web pages using available computational resources at the time to produce an estimate that refreshes in a time frame that makes the measure of immediate interest, Brin and Page [1998] introduced *Google centrality* as a proxy for the important or relevance of a Web page that can be computed (a) with substantially fewer operations than available centrality measures and (b) whose estimates can be *recursively* improved – taking advantage of increases in computational capacity of hardware processors. They proceed by starting from the adjacency matrix that describes the network of web pages A_{ij} with entries $a_{ij}= 1$ *(node connected to node j); 0 (otherwise)* and normalizing the non-zero entries (so they sum up to *1)* and replacing the *0* entries with *1/N*, where N is the total number of nodes to create the Markov transition matrix S_{ij}, from which they calculate the *Google Matrix* $G_{ij}= aS_{ij} + (1-a)/n$, where *a* is a parameter representing a 'damping factor' *((1-a)* being the probability that a Web surfer suddenly signs off during the process of skipping among web pages passed on the hyperlinks embedded in them [Ermann, Frahm and Shepelyansky, 2015]. (The Google search engine uses an *a* value of 0.85.) For values of $0<a<1$, *G* belongs to the class or Perron-Frobenius operators and its right eigenvector *L(i/G)* has real, non-negative elements corresponding to the probabilities that a random surfer can be found on Web page *i,* and is its PageRank. This construction enables a super-fast computation method for *PageRank(i)* based on *recursion*: Start with an initial guess

PageRank₀ and generate the next iteration by multiplying the previous guess by the Google matrix G_{ij}: PageRank$_{l+1}$= $G_{ij}PageRank_l$. – which only takes $O(n+m)$ operations to compute (rather than $O(nm)$ as in the other centrality measures' cases). Fast convergence of the algorithm (after up to 100 iterations) is guaranteed by selecting $(1-a)$ to be suitably small (a to be suitably large, respectively).

A computational abstraction layer contributes to the strategist not just a set of heuristics for *sizing* problems and solving them in situations in which the complexity of the solution process matters, but also a set of models and templates for *defining* problems – in this case as a set of *network position problems* (Table 2).

Loosely worded problem ('challenge')	**Tighter formulation**	**Definition as a Network Search Problem**
Identify the influencers of pharma product buyer decisions in a large health care provider network	Find the most respected physicians and basic researchers in the relevant fields.	*Find the nodes with the greatest degree and Bonacic centrality in the graph G of publications (nodes) and citations and co-citations (edges) of published findings, within a week.*
Change the perception of the market regarding the 'monolithic culture' of the firm.	Increase the diversity of the sources of outbound information about the company in Web materials and social media.	*Find and order the degree centrality of nodes in the graph G of communicators (nodes) disseminating messages (links) through various outlets (nodes) within a week and change the spread of these nodes by a factor of 2 with a quarter.*
Increase the efficiency of coordination of the organization around rapid-turnaround design tasks.	Increase the spread and distribution of project-relevant information so that more people are in the know more of the time.	*Find the information flow centrality of all decision makers in graph G of project-relevant communications (email, text) and increase the average centrality of the lowest 30% of the nodes by a factor of 3 within a month.*

Table 3. Using network position searches as problem definition and shaping tools for common business problems.

CENTRALITY MEASURE TO BE COMPUTED	***Computational complexity as a function of nodes and edges in the network***	**Number of operations required to order a billion (n) web pages connected with 500 billion (m) ties…**	**…connected with 50 billion ties…**	**… with 5 billion ties…**	**… with 0.5 billion ties…**
Betweenness: The probability that any network shortest path (geodesic) connecting two nodes J,K pass through node i	***T(n,m)=O(nm) WORST CASE: O(n^3)***	***5x10^{20}***	***5x10^{19}***	***5x10^{18}***	***5x10^{17}***
Closeness: The inverse of the average network distance between node I and any other node in the network	***T(n,m)=O(nm) WORST CASE: O(n^3)***	***5x10^{20}***	***5x10^{19}***	***5x10^{18}***	***5x10^{17}***
Bonacich: the connectedness of a node i to other well-connected nodes in the network	***T(n,m)=O(nm) WORST CASE: O(n^3)***	***5x10^{20}***	***5x10^{19}***	***5x10^{18}***	***5x10^{17}***
Google PageRank: 'the probability that a random surfer can be found on page i	***T(n,m)=O(n+m)***	***5x10^{11} (5x10^{13}) assuming 100 iterations***	***5x10^{10} (5x10^{12}) assuming 100 iterations***	***5x10^{9} (5x10^{11}) assuming 100 iterations***	***5x10^{8} (5x10^{13}) assuming 100 iterations***

Table 2. Order of magnitude (O(n,m) Estimates of the Rate of growth of Time complexity (T(n,m)) as a function of the number n of nodes and number m of links in a network.

Creating, Sanitizing and Camouflaging Toxic Debt. The *Great Recession* of 2008 (aka 'financial crisis') has been conceptualized as a classic 'market for lemons' problem [Akerlof, 1981] compounded by the computational complexity of detecting defective products in large batches [Arora et al, 2010]. DeMarzo [2005] shows how a collection of mortgages held by a bank with borrowers whose risk of default is broadly and independently distributed across borrowers and falling into different risk classes can be tranched and re-assembled into a collection of securities that lower risk for their buyer by spreading it across the risk classes into which mortgages had been previously assembled by the bank (who is the lender for the mortgage and also the seller of the CDO security).

Because the bank sells *many* CDO's (*N*), each of which is based on tranches from many mortgages (*M*) drawn from many different risk classes (L), it can use its superior knowledge of risk class profiles and the added degree of freedom afforded by the mix-and-assemble process by which it creates CDO's to over-represent mortgages drawn from certain (higher) risk classes (usually the ones it would like to convert into cash as soon as possible – the 'toxic' ones) and realize a gain (over the fair market value of the CDO) by selling all of the CDO's at the same price. This gain can be understood as a 'computational' one: a buyer whom is aware of the seller's lemon camouflage strategy would have to inspect the graph (Figure 9) of CDO's and mortgages (grouped into risk classes) and identify and examine its *densest subgraphs* (collections of nodes that are linked the maximal number of links) in order to determine the degree to which over-representation of mortgages from any one of *K* risk classes can skew the distribution of returns in any one of *N* CDO's.

Sizing the problem of finding the densest sub-graph(s) of a network is the key to realizing the 'computation gain' of camouflaging lemons. Goldberg's algorithm [Goldberg, 1984] does so at a computational cost of $O(nm)$ – multiplicative in the size (nodes n and edges m) of the entire network, and therefore not easily computable for large networks (hundreds of thousands or millions of nodes). If a buyer is (a) aware of the possibility of the 'complexity camouflage' of lemons and (b) in the possession of Goldberg's algorithm for detecting dense subgraphs linking mortgages to CDO's based on them, then she may still be computationally challenged (or unable) to detect them. If the seller *knows* this, then he might be able to get away with skewing the distribution of returns on some CDO's without affecting the market price of the CDO bundle.

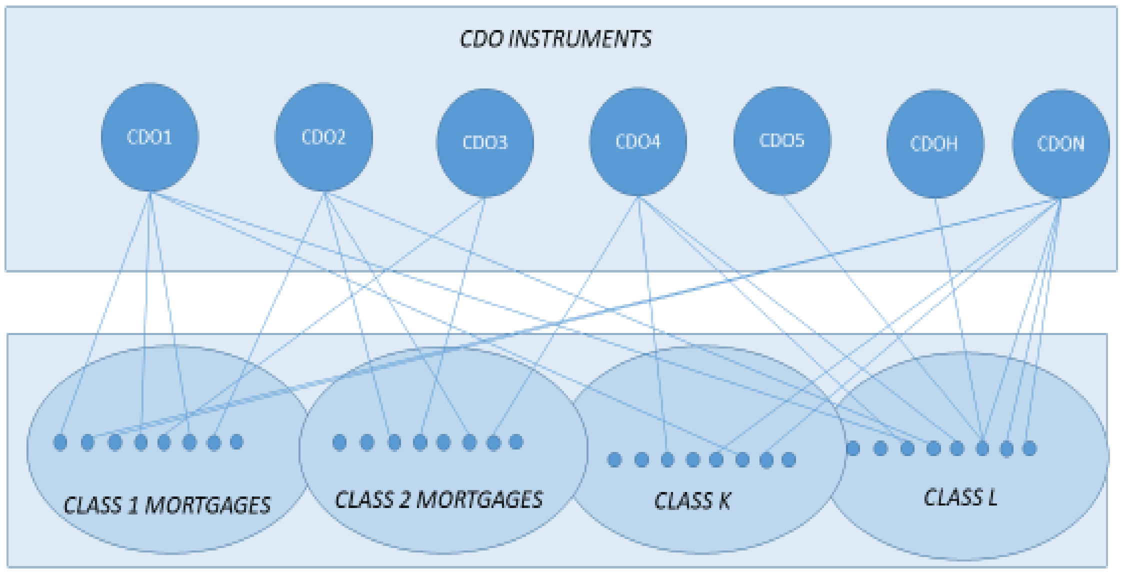


Figure 9. Illustrating the graph structure of constructing N *CDO securities from a collection of* M *mortgages that have been grouped into* L *risk classes.*

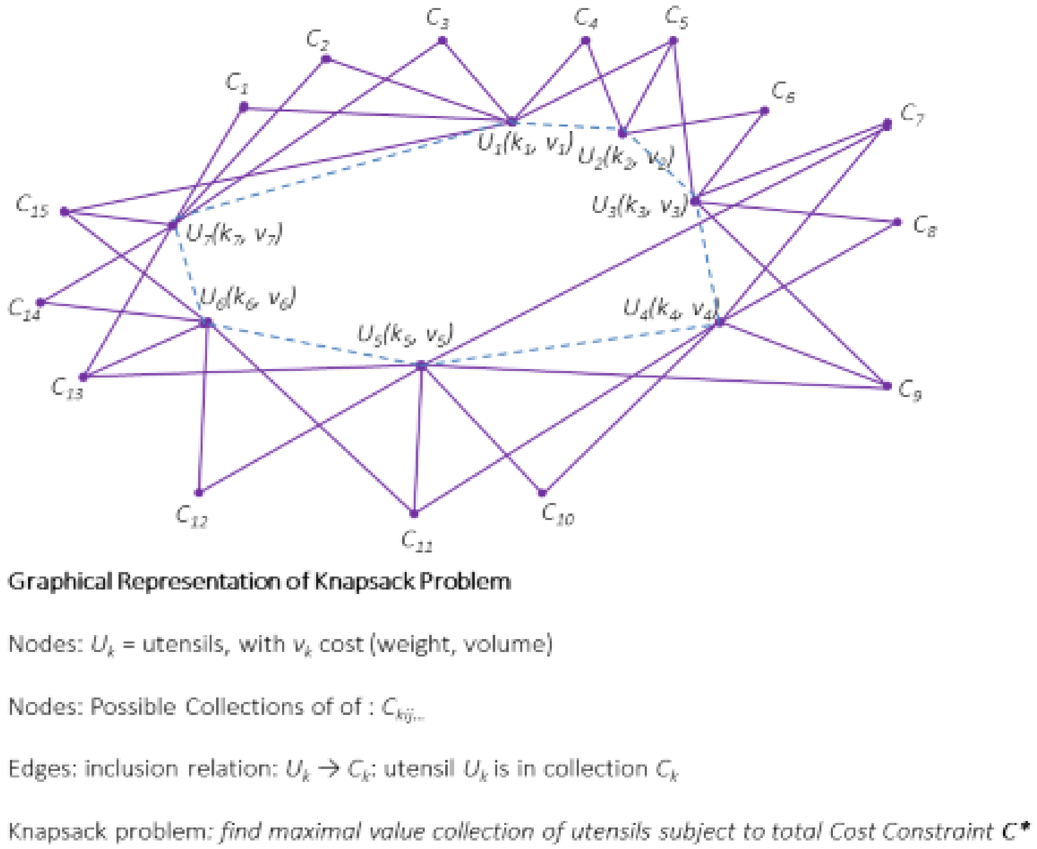
However, a buyer that is equipped with a *more efficient algorithm* for detecting the densest subgraph(s) of a network (such as the greedy algorithm-based approach of [Charikar, 2000], which has a complexity of $O(n+m)$ for producing a reliable estimate) then she may be able to check on the degree to which the seller has planted lemons and camouflaged them using his knowledge of the computational complexity of detecting them. Finally, if a buyer *knows* that the seller is using a *specific* approximation scheme for detecting dense subgraphs of mortgages and CDO's, then she might look for lemon planting strategies (kinds of subgraphs) that the approximation scheme specifically used by the buyer's lemon sniffer may not be able to detect with the requisite reliability and in the time normally allotted.

Once again, a computational abstraction layer contributes a set of *problem definition tools* – aimed at formulating problems relating to the detection, measurement and optimization of highly important *network structures* (such as sub-graphs and cliques: Table 4) – similar to that for defining and sharpening problems of detecting, measuring and purposefully changing *network position*. – in a way that highlights the versatility of *graph searches* more generally in the definition of strategy problems.

Loosely worded problem (‘challenge’)	**Tighter formulation**	**Definition as a Network Search Problem**
How do we increase our presence and influence using Web communities that discuss our products?	Identify the HUBS of users that participate in discussion forums about the company’s products in the Web and seek to influence these users by shaping their dialogue and letting them persuade one another.	*Find the DENSEST SUBNETWORKS of users and discussants in Web forums and discussion circles within 2 weeks (nodes are individuals, edges are co-presence on discussion foruym), communications targeted to their current linguistic practices within 2 additional weeks.*
How do we increase the collaborative capability of our R&D groups?	Identify the size and distribution of the most densely knit social circles among researchers and increase the size of these circles.	*Find the distribution of K-CLIQUES* (fully connected subnetworks of the communication network (nodes are researchers, directed edges are pairwise communications, edge weights are communication frequency) of researchers) within 1 week, re-design workspaces to increase the maximum and average clique size by a factor of 1.5 over a month.
How do we decrease the cost structure of our value-linked activity chain?	Identify the relevant structures in the network of value-linked activities and seek lower cost replacements for the most prominent activity units.	*Find the MINIMUM VERTEX COVER* of the network of value-linked activities (nodes are individual activities, directed edges are value links that capture the degree to which value created by activity J is dependent on value created by activity K) within a month, identify lower-cost substitutes for these activities within another month.

Table 4. Table 3. Using network structure and topology searches as problem definition and shaping tools for common business problems.

Designing Complex Products and Services. ‘Graph searches’ constitute suggestive but tight generative models for business problems requiring the fast search for nodes and structures of potentially very large networks, which are increasingly prevalent in ‘Big Data’ environments [Moldoveanu, 2015]. However, using *network searches* as abstraction layers is also useful to the process of problem definition in situations that are not themselves immediately suggestive of a network representation. For example, the problem of designing or modifying a strategically important new product or service can be represented as a problem well known to computer scientists – the Knapsack Problem (Figure 10): a hiker going on a long and arduous trek has more tools, utensils and other items than she can pack in the only knapsack she knows she can take along. The knapsack has a finite volume – as does each candidate item. Additionally, each item has a certain value to the hiker, which she knows or can estimate. Her challenge is to find the combination of items that will give her the maximum value, subject to fitting inside the volume of the knapsack – hence the name of the problem.



Graphical Representation of Knapsack Problem

Nodes: U_k = utensils, with v_k cost (weight, volume)

Nodes: Possible Collections of of : $C_{kij...}$

Edges: inclusion relation: $U_k \rightarrow C_k$: utensil U_k is in collection C_k

Knapsack problem: *find maximal value collection of utensils subject to total Cost Constraint* C^*

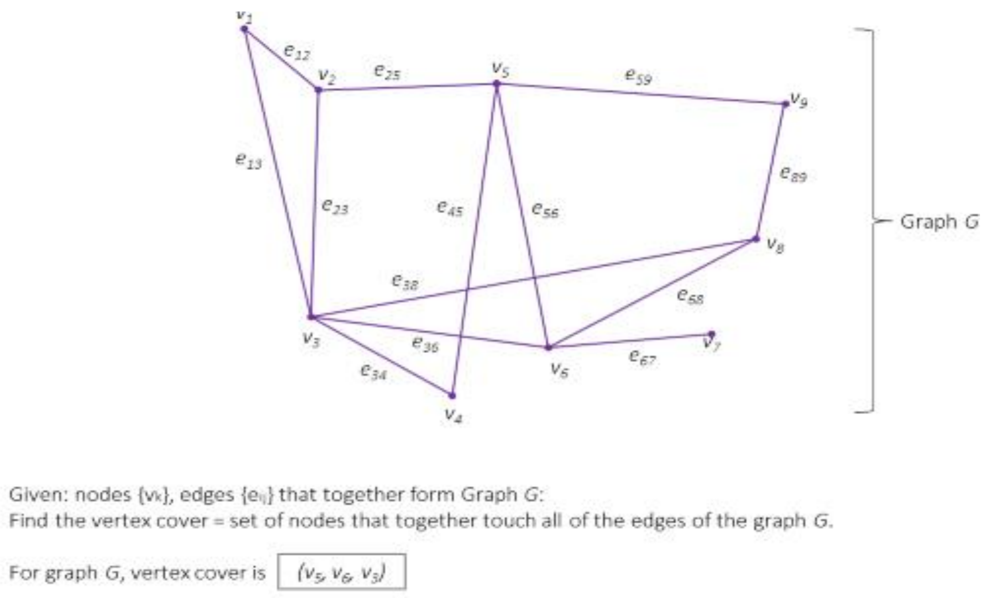
Figure 10. A Graphical Representation of the Knapsack Problem, wherein the volume of each item is represented as a unit cost c*, and the total volume of the knapsack is represented as a total cost constraint C*.*

The solution search space for the problem comprises all of the possible combinations of items that could 'pack' the full knapsack. Solving the problem by exhaustive enumeration and evaluation of the possible solutions entails packing the knapsack with all possible combinations of items (of which there are 2^N-1 if there are *N* items), evaluating the total estimated value of each combination, ranking all feasible (volume-fitting) combinations according to their total values, and selecting the combination of items that has the maximum total value.

Seen as the problem of packing a knapsack, the problem seems unnecessarily formalistic and complicated (people pack knapsacks all of the time without worrying about enumerating a solution search space that increases exponentially in the number of items). However, seen as a component of a computational abstraction layer for business problem solving, the problem is a *model* for a number of problems that show up in business– such as that of optimizing the set of features F of a new product, where each feature has a cost and is expected to add a particular value to the product, subject to an overall cost constraint on the new product, with the aim of maximizing the value of overall product (which may be additive, sub-additive, or super-additive in the values of the individual features). If the number of features is large, their costs and marginal values added are not easy to calculate and the overall value of the product can only be estimated once a full set of features and the relationships between them has been specified, then the growth in the complexity of the product design process as a function of the number of features becomes highly relevant.

Versatility and *portability* to different scenarios are highly useful components of a good abstraction layer. Computer scientists routinely use *one problem as a model for a different problem*, which allows them to export from the solution of one problem to the solution of a different problem [Sedgewick and Wayne, 2014; Knuth, 2011]. One way to export insights from one problem to another is to consider changes of variables that will map problems one into the other – and together onto the *canonical problem* one has already dealt with. In the case of the Knapsack Problem, re-labeling the network so the nodes become value-added activities in an industry that can be constituted into more or less advantageous value-linked activity networks allows us to frame the problem of the *de novo* design of platform-based businesses (Uber, AirBnB, Dropbox, Salesforce.com) as problems of finding the optimal set of value-linked activities that should be integrated within the same organization [Novak and Wernerfeldt, 2002]. In the case of Uber, they include writing off depreciation of personal vehicles and additional disposable income (drivers), predictable scheduling and billing, ease of access and ubiquitous accessibility and secure, auditable, trackable payment (riders), traffic decongestion and alleviation of parking lot overload (municipalities), scalability and marginal profitability on a per transaction basis (shareholders) – each with an associated cost of provisioning (designing, supporting, securing the platform; recruiting and certifying drivers, etc) and an expected total value. *Designing Uber* can be modeled as solving a Knapsack Problem where the "knapsack" is the value-linked activity network of the business and the individual components are valuable activity sets that can be integrated on the same platform. Seen in this form, the abstract-form Knapsack problem can be 'exported' to the design of new businesses eg: (a secure, Blockchain based platform for tracking educational degrees, courses, certifications, student loans, employer recommendations, job performance, skills; or, a secure enterprise-level recruitment platform that allows seamless interaction between employers and candidates on the basis of written documents (CV's), candidate introduction videos ('video CV's), interview transcripts, credentials and certificates, etc.)

Using canonical problems to encode frequently recurring business problems offers up possibilities for defining and shaping predicaments messy and peppered with idiosyncratic details down to searchable solution spaces and systematic search procedures optimally suited to their enumeration and evaluation. The well-known VERTEX COVER problem, for instance, can be used to sharply and precisely model:



Given: nodes $\{v_k\}$, edges $\{e_{ij}\}$ that together form Graph G:
Find the vertex cover = set of nodes that together touch all of the edges of the graph G.

For graph G, vertex cover is $\boxed{(v_5, v_6, v_3)}$

Figure 11. Schematic description of the VERTEX COVER PROBLEM, enjoining the problem solver to find the set of nodes that collectively touch ALL of the edges of a graph.

- The problem of identifying the 'key information agents' in a large organizations – those whose communications reach all members (nodes are people, edges are communications in a particular medium);
- The problem of identifying the 'critical set' of components of a product – those whose functionality affects the functionality of all of the other components (nodes are components, edges are causal contributions of the functionality of one component to that of another);
- The problem of identifying the 'minimal trust core' of a large group of users of a product or service – the group of people whose communications are followed and valued by other users (nodes are Web users, edges are 'followed and liked by' links whose followership and 'like' scores are above a certain threshold.

"The Intractables". Several of the 'canonical' problems we have seen – such as the Knapsack and Vertex Cover problems – exhibit a level of complexity that grows very quickly with the size of the problem. Computational science made a massive stride in its practical and

theoretical reach with the introduction of a sharp definition of 'very quickly' that allows computational scientists to sort problems in advance of solving them. This distinction is based on the difference in the growth of polynomial functions and super-polynomial functions (for instance, exponential functions). The set of problems whose solution complexity *T(n)* grows more quickly than any polynomial function *T(n)>P(n),* where P is any polynomial function of *n* – has a structure to it [Cook, 1960]: many of the problems that fall in that class are solvable in only probabilistically in polynomial time (or, they can be solved in *non-deterministic* polynomial time) and any candidate solution to them can be verified within *polynomial time*.

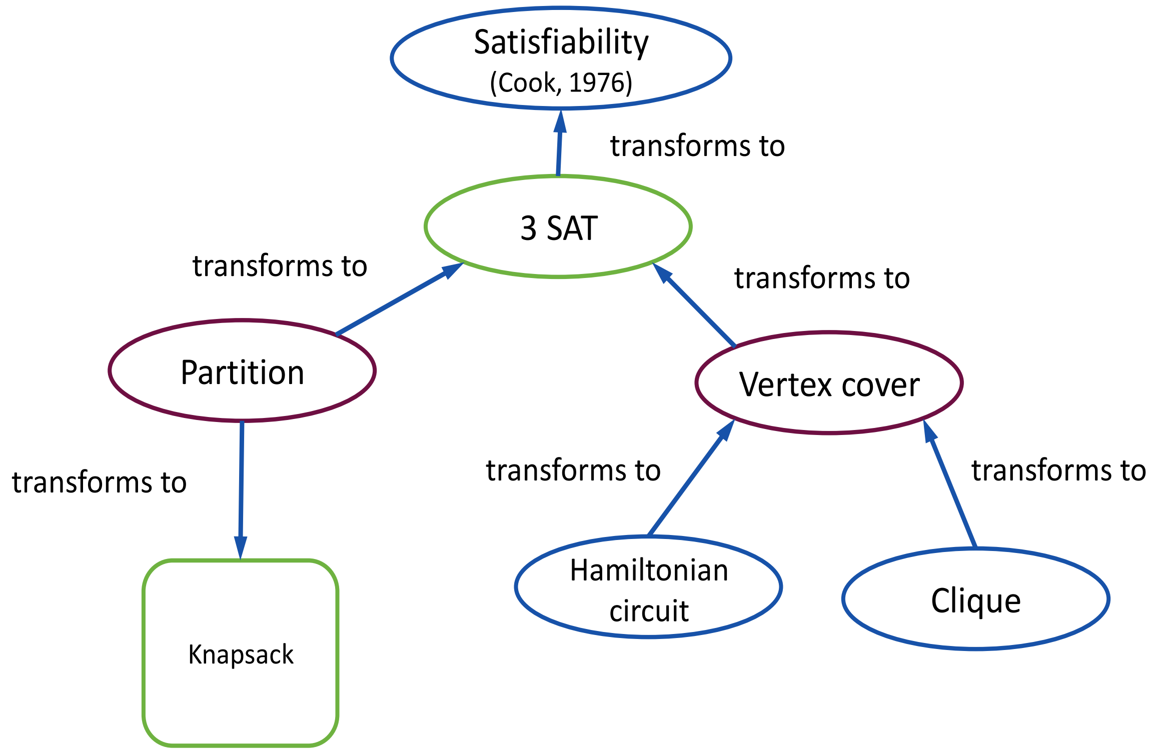


Figure 12 : The Class of Intractable (NP-hard) problems, showing analytic reductions among different common problems that are irreducibly complicated to solve.

The rapid growth of computational science across fields as disparate as molecular genetics, linguistics, legal analysis, actuarial science and high frequency trading can be traced to the construction of a *family* of *NP*-hard problems that are structurally equivalent to one another [Karp, 1972] (see Figure 12). This network of *canonical problems that are known to be* *intractable* by analytical reduction one to the other allow one to size up the complexity of a problem before attempting to solve it (simply by showing it to be analytically reducible to another *NP* hard problem) and to device strategies for simplifying the problem by trading off either the probability of finding its optimal solution or the accuracy of the solution itself. Several of the problems we have considered – optimally re-allocating decision rights (SAT), constructing a new product with an optimal set of features, drawing the boundaries of the firm around the maximum 'core' of value linked activities (KNAPSACK), and identifying the most significant networks of informers and contributors in an organization or market (VERTEX COVER) - fall into the *NP* family, which allows strategists to see how their difficulty scales as the number of their variables (features, activities, informers) grows.

As 'strategy problems' are increasingly 'big data' problems comprising very large numbers of variables and data points, it becomes critically important to examine the size of the problems that 'big data' generates [Moldoveanu, 2015]. It is not just the number of variables, the heterogeneity of the databases and formats in which the values of the variables are stored and the ways in which data can be combined and re-combined to generate predictions that matters, but also the rate at which the difficulty of classification and prediction problems based on the data grows with the size of the data set. A precise formulation of 'learning algorithms' ('probably approximately correct' learning [Valiant, 1984]) stresses the requirement that the difficulty of any learning routine or algorithm *not* exhibit an exponential blow-up as a function of the number of instances required for learning the concept. A computational abstraction layer that allows for explicit problem *sizing* is equally important to those who are trying to solve both *optimization* problems (building the optimal product or architecting the optimal team authority structure) and *prediction* problems (building risk classes for mortgage defaults on the basis of millions of data points relating to the borrowing behavior of users in different ethnic, cultural and demographic classes). (The distinction is somewhat contrived: Prediction problems in the machine learning literature are formulated in terms of the minimization of a 'cost function' (or the extremization of an objective function) that represents the distance between observed, recorded and registered data and data that is generated by a predictive model of the process that generates it, whose parameters evolve as a function of the success of previous predictions.)

Kinds of Problems: A Tree Structured Picture of a Computational Abstraction Layer. It may be useful to summarize work we have done so far on generating a computation abstraction layer that supplies microfoundations for business problem solving in the form of a classification tree (Figure 13 below) that helps strategists distinguish between:

- well-defined and ill-defined problems (presence of measurable current and desired conditions, an objective function and a variable space that can be used to synthesize a space of possible solutions);
- well-structured and ill structured problems (independence of the implementation of a solution search procedure from changes in the solution search space itself);
- tractable versus intractable problems (super-polynomial (eg: exponential blow-up) increase in the number of operations estimated to be required to solve the problem as a function of the number of variables in the problem);
- difficult and simple problems (the degree to which the difficulty of the problem stresses and stretches the resources of the organization).

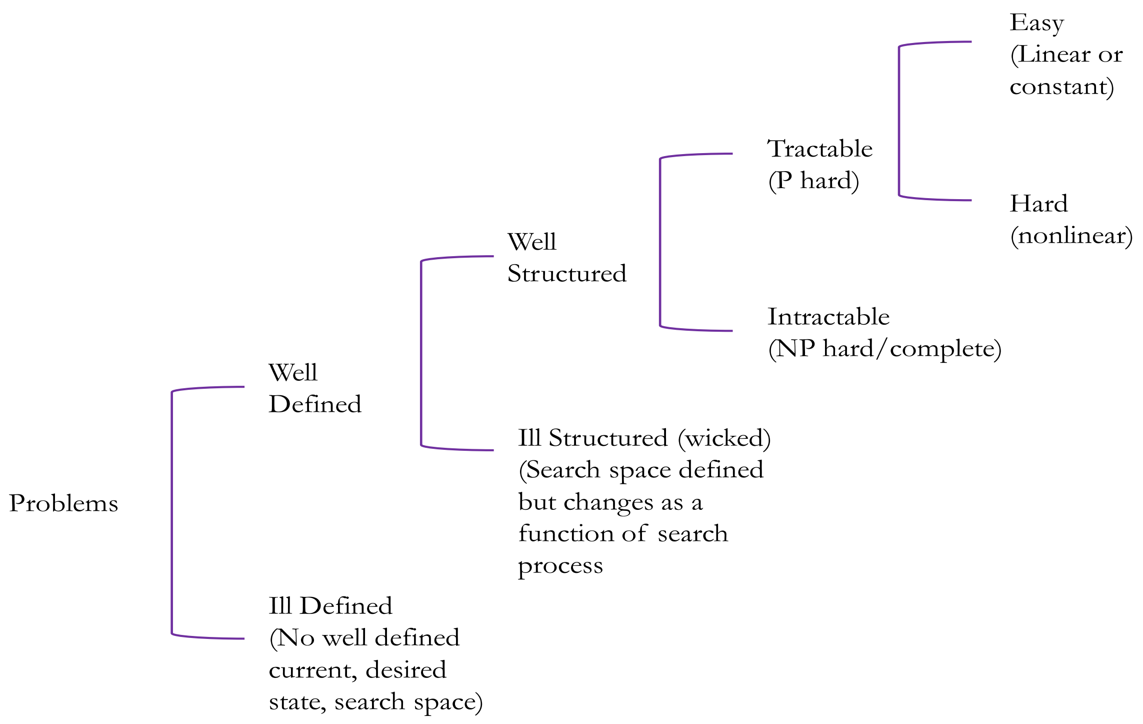

Problems
Well Defined
Ill Defined (No well defined current, desired state, search space)
Well Structured
Ill Structured (wicked) (Search space defined but changes as a function of search process
Tractable (P hard)
Intractable (NP hard/complete)
Easy (Linear or constant)
Hard (nonlinear)

Figure 13. A 'Classifier' for Strategic Problems, showing the basic distinctions that we have drawn in the paper thus far.

4. What is the *best way* to solve *this* problem? Algorithmic micro-foundations for problem solving processes.

A computational abstraction layer for business problem supplies the problem solver not only with ways of defining, structuring and sizing problems, but also with a *repertoire of canonical problem solving techniques* in the form of procedures for searching solution spaces that are either provably optimal for certain kinds of problems and solution spaces, or can be adaptively optimized to the problem the strategist is trying to solve. Making some further relevant distinctions is what allows us to cut business problem solving at the joints using the right concepts from computational science, as follows:

Types of Search: Deterministic and Probabilistic. Computer scientists distinguish between deterministic and probabilistic searches of solution spaces. If we consider the problem of evaluating the set (of all subsets) of *100* different possible value-linked features, each of which can be specified into a final product, we see that searching *exhaustively*, in a deterministic fashion (each step determined by the preceding one), will often be infeasible under time and resource constraints. Many algorithms computer scientists use when faced with such problems use the planned introduction of *randomness* ('flip a coin or a sequence of *l* coins after move *k* to decide what move *k+1* should be) that can help to make the search process faster at the cost of sacrificing *certainty* about achieving the optimal solution. Instead of obsessing over each possible combination, one can start with random subsets of features, and, after verifying they satisfy a total cost constraint, evaluate the total value of the product that comprises them; then seek alternatives by making small substitutions (take out one feature, add a lower cost feature from the set of features not yet included, and re-evaluate the value function of the entire product.)

Kinds of Search: Global and Local. Similarly, in situations in which the sheer size of the solution search space daunts the resource base of the problem solving organization, one can restrict the search to lie in the neighborhood of a solution that is already known to work, but, which we suspect, can be improved. Instead of *searching globally*, then, computer scientists seek meaningful *local neighborhoods* within a solution search space, wherein to search for a good enough solution. (Superficially, this looks like Simon's first cut at articulating 'satisficing' [Simon, 1947], but it adds a useful refinement: the solution search space is restricted to a local neighborhood that can be chosen to maximize the chances a good enough solution will be found there.) A strategic problem solver seeking a reallocation of decision rights to her team may choose to search in the neighborhood of either the current allocation of decision rights (by adding or removing single decision rights from single agents and simulating or emulating or experimenting to find likely consequences), or may choose to start in the neighborhood of an allocation of decision rights found (by best practice mapping and transfer machines like strategic consultancies) to be successful in the industry in which the firm operates – or across other relevant industries.

Modes of Search: Serial and Parallel. Parallelizable hardware architectures have enabled computer scientists to make searching for solutions to difficult problems *faster* and thereby *cheaper* by parallelizing the search process. Searching for a name in a database can be made faster by splitting up the database into N non-overlapping sub-databases and having each of N processors search each of the smaller data bases. In general, problems with large solution search spaces are not immediately or self-evidently parallelizable: ordering a 10^{15}-long list of bit strings in order ascending in the numbers they specify cannot be done simply by splitting up the list into 10^5 lists of 10^{10}strings and ordering each – as the costs of then merging the lists in an order preserving way can cancel out the benefits of parallelizing the search. Business problems with large solution search spaces may be more easily parallelizable (splitting up decision right allocations according to the types of decisions that they relate to) or less so (splitting up a Web influence network into sub-networks in order to evaluate the hub of maximally influential agents).

What is particularly useful about these distinctions in business problem shaping and solving is the ability to combine and concatenate them into a set of *meta-algorithms* for solving difficult problems, as follows:

Meta-Algorithmics of Search: Strategic Ingenuity and Intelligent Adaptations to Complexity.

Divide and Conquer (*DC*). Algorithms designers often divide up larger problems into smaller sub-problems, whose individual solutions can then be aggregated into the solution to the entire problem (Figure 14). It involves (a) partitioning the problem search space into smaller search spaces that can be more easily searched, and (b) piecing together the separate solutions to the smaller problems to form (possibly sub-optimal, but still superior) solutions to the larger, intractable, problem. The set of all possible subsets of a set of features of a product can be divided up into subsets-of-subsets of features that can be searched independently by several different individuals or teams working in parallel, provided that the value of the product is strictly additive in the values of the features.

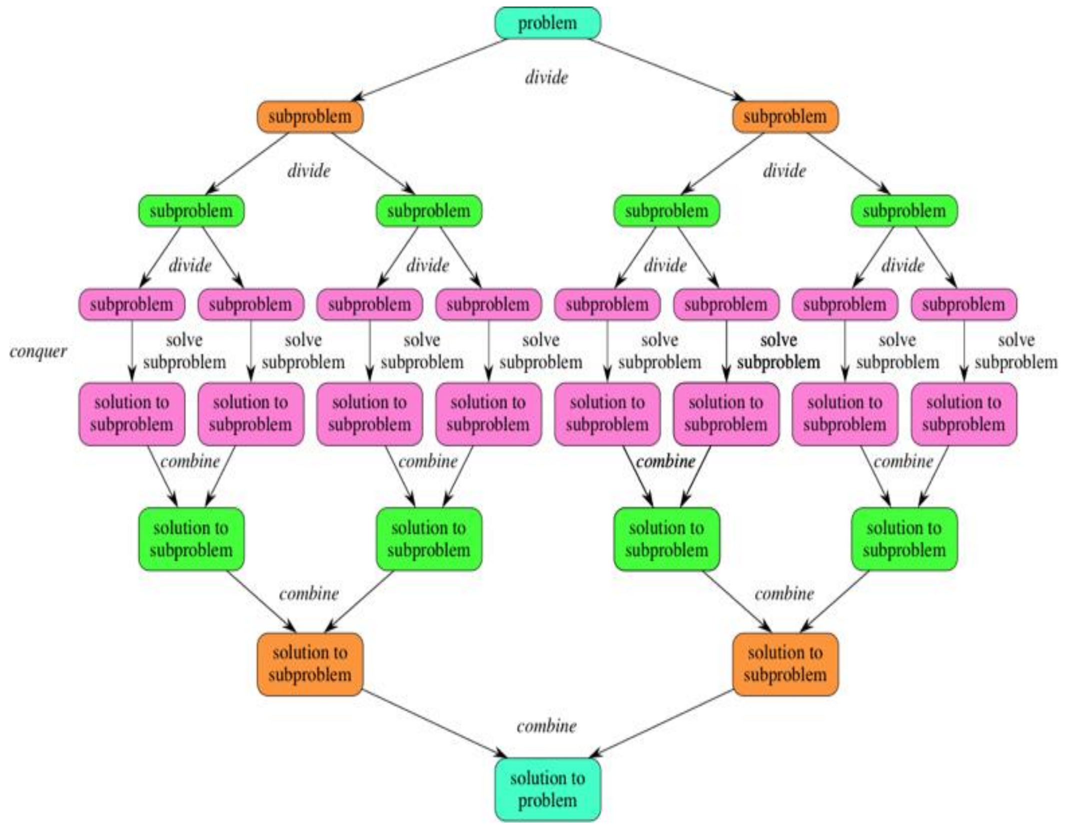


Figure 14. Illustrating the Basic Intuition Behind the 'Divide and Conquer' Heuristic.

There is no guarantee the concatenation or integration of solutions to smaller sub-problems will in general be an optimal solution to the bigger problem: Breaking up the problem of evaluating the set of all possible decision right allocations to each of *N* individuals on a team over decisions arising from *K* different classes of problems will not generate an efficiently *separable* solution process if the efficiency of allocating certain kinds of decision rights to some people (initiation rights) depends on the allocation of other kinds of decision rights to other people (ratification rights). However, one may still be able to divide up the problem of evaluating decision right allocations to certain classes of people (ratification rights to executives; implementation rights to employees) and more efficiently evaluate all allocations of *other* decision rights between executives and employees with respect to their efficiency.

The use of a divide and conquer heuristic is not limited to problems that can be tightly formulated in graph-theoretic language. One can divide up the problem of 'getting people together for a meeting' (desired conditions: they will all show up, at time *t* and location *S,* informed and ready to engage in task *T)* by dividing up the task of inviting them and motivating the meeting to *K* different convenors (who trust and copy one another on emails). Each operation in the overall task can be defined *ostensively* ('getting people of *this* hierarchical rank to answer *this* kind of email') – even if a precise algorithmic model for 'answering this type of email' is not available or not derivable – which allows us to quantify the marginal and average cost of such operations in different environments, as a function of the number of operations involved in the successful completion of a task. Similarly, the problem of *designing a strategy for the containment of a large information leak from the organization* can be divided up into a series of sub-problems (identifying the source of the leak, gathering evidence for the act of leaking, sequestering access to information from the source) which can be divided up among problem solvers specializing in the tasks relating to solving each sub-problem.

Local Neighborhood Search (LNS). Algorithm designers deploy local searches around best-guess solutions to solve problems that are clearly intractable if pursued on a brute force basis. The *Traveling Salesman Problem* (TSP) – finding the minimum distance path connecting *N* locations – has a solution search space (all *N!* fully connected circuits) that is super-exponential (and thus super-polynomial) in size. The dramatic reduction in the time complexity of TSP highlighted in Figure 12 above was accomplished by a procedure for searching the *N!*- size search space of the TSP using a local search meta-algorithm named after its inventors, Lin and Kernighan [Lin and Kernighan, 1973]. The procedure involves selecting an 'almost-complete' tour of the cities (a 'delta path') which includes all of the cities exactly once, except for the last one, measuring the total distance of the delta-path that had been generated, making switches among the edges included in the delta path and edges that are 'available' but not included, comparing the total distance of the modified circuit with its last version, and retaining the more efficient path. One key to the (exponential) speed-up achieved by this heuristic is the way in which edges are exchanged, which is (usually) *2* (eg: *1*-

3&3-2 replaced with 1-2&2-3) at a time - entailing a search space of *N(N-3)/2*. The algorithm allows for the exclusion 'by inspection' of many inferior paths: for instance, in the 4663 city TSP one can exclude combinations such as(*Toronto (Central Canada) Kelowna (Western Canada)-London (Central Canada*) without 'evaluation'.

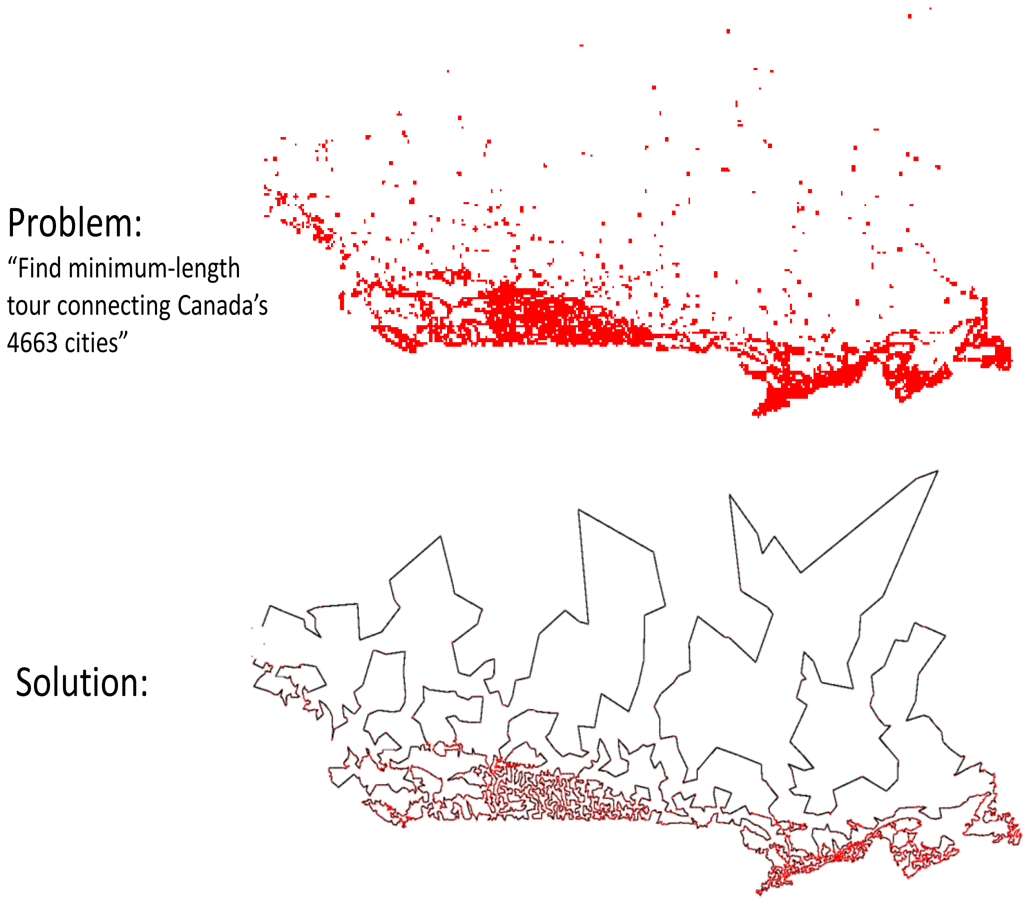


Figure 15. A TSP Problem Search Space for Canada's 4663 Cities and Solution to the Problem Using Lin Kernighan Local Search Heuristic.

While the TSP can itself serve as a canonical problem model for problems of strategic significance ('find the optimal *influence path* within a network of consumers') the uses of *local neighborhood searches* as a way of simplifying business problems is highly exportable. One can constrain the search for a new allocation of decision rights to members of a large team by searching in the neighborhoods of current allocations (making small changes an simulating or experimenting with their consequences) or in the neighborhood of allocations that have been shown to work in cross sectional studies supplying correlative evidence, or longitudinal studies supplying plausible causal evidence for their value.

Branch and Bound (BB) techniques partition the solution search space via a tree whose nodes represent decisions among different families of a solutions. Calculating bounds on the performance of a solution that arises from different branches of the tree, and deleting from the search space branches likely to result in a sub-optimal solution. The key feature of a good tree structure for *BB* methods is that it is 'quickly prunable': estimates of performance bounds for different branches are calculated in advance, to lower the chances that an optimum be 'missed' by the resulting search. For the TSP problem a *BB*-suitable tree search can be built on the basis of *whether or not a path contains a particular segment connecting two cities*. The first node of the tree creates two 'buckets' of possible routes: one containing routes containing *AB* and one containing routes that do not. Subsequent nodes of the tree (there will be *N(N-1)/2* nodes for an *N* city tree in total) provide finer-grained partitioning of the space of possible paths. The key to reducing the time complexity of the search is a tight characterization of the best/worst case performance that one can expect from *any given sub-tree*: Each fork of the tree cuts the number of search operations required by 50 per cent. BB methods can be used to quickly narrow the search space of 'big' problems of strategic choice. The computation and selection of the optimal Nash Equilibrium in a game – or of the Nash Equilibrium that has a minimum payoff to one player of at least P – is and intractable (NP-hard) problem [Austrin, Braverman and Chlamtak, 2011]. In a simultaneous move oligopolistic competition game with 4 competitors, each of whom has 6 strategies at her disposal, the search space for combinations of strategies has 1296 distinct outcomes (6^4). A BB method can quickly narrow this space by 50% by eliminating combinations of strategies that include a 'low cost' product or service offering on the basis of examining the worst case scenario (a price war) that is likely to be triggered by this choice. Each step of eliminating combinations of strategies that contain an undesired component will trigger a similar contraction of the search space. In the case of optimizing the design of a new product by choosing among features that have additive costs and values subject to a total cost constraint and with the objective of maximizing value, one can 'parse' the overall solution set by creating a tree whose branches explore the subsets of features containing *(x_i=1)* or not containing *(x_i=1)* any particular feature, and bounding – along the length of each subtree – the overall value of the resulting achievable subset.

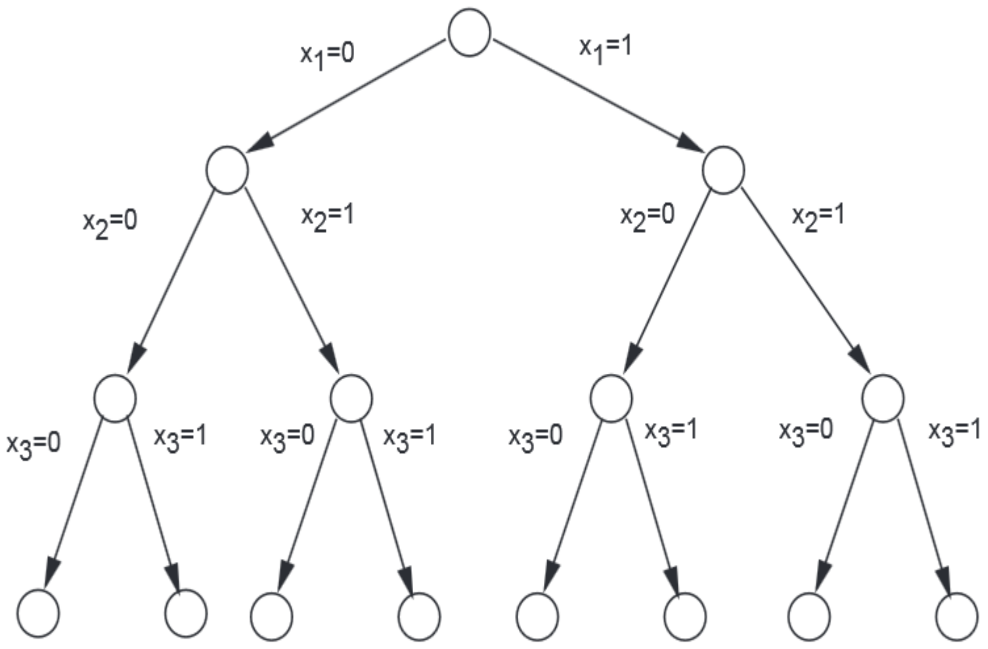


Figure 16. The basic 'decomposition of independent subspaces of solution search space' principle of the branch-and-bound approach.

Randomization need not be *blind*. The general set of approaches that pass under the name of *intelligent randomization* provides a way of probing a large solution search space and intensifying the search in the neighborhood of promising (but randomly generated) solutions. A problem-generating model (*a generative model*) that yields to intelligently randomized solution search procedures in the well-known *NK* model of (co)-evolutionary dynamics introduced in [Kauffman and Levin, 1987]. *N* genes – whose phenotypic expression is dependent on afferent contributions from at least *K* other genes – can generate highly complex (many local optima) fitness landscapes (payoffs to the phenotype) as a function of the specific informational content of the genes. The resulting *NK* model has been used [Levinthal and Warglien, 1997; Ghemawat and Levinthal, 2008] to examine the dependence of a firm's performance on the structure and topology of its activity sets – or *K*-wise coupled and interdependent interdependent 'policies'. To a firm that aims to strategically tailor the set of activities it pursues, the problem of choosing the set that optimizes its performance ('fitness') was shown [Kauffman and Weinberger, 1989; Weinberger, 1991] to be computationally equivalent to the well-known intractable (*NP*-hard) *k-SAT* problem for $k>1$.[1] . The *NK* strategic decision problem ('Is there a fitness function of the *N* activities, each mutually coupled to *k* others with value greater than V?') maps into the *kSAT* problem ('Is there a set of variables whose aggregate satisfiability score is at least *V* when plugged into a set of *M k*-variable formulas?') trivially for $M=N$, and with padding of the search space for $M>N$ and $M<N$ [Weinberger, 1996]. Rivkin [2000] argued the intractability of the *NK* problem (derived from the intractability of the *kSAT* problem for $k>1$) can make complex strategies (characterized by the design of value chains comprising many linked activities) difficult to imitate. But the complexity of solving the *kSAT* problem yields to searches of the solution space based on randomly permuting both the sets of initial variables and the assignment of truth values to variables within a formula [Schoening, 2002; Brueggermann and Kern, 2004; Ghosh and Misra, 2009]. These approaches achieve a worst-case complexity of solving the 3SAT problem of $(1.d)^N$ (where *d* is a natural number following the decimal point) instead of 2^N, which, even for very large values of *N* can produce a massive decrease in the complexity of the *3SAT* problem (Table 3): a factor of 10^{20} reduction in the complexity of solving the 3SAT problem for $N=100$ is achieved by a random walk based algorithm.

1 The problem takes as an input a set of *N* variables and a set of *M* Boolean expressions containing up to *k* variables AND, OR, and NOT, and asks for an assignment of the *N* variables to the *M* expressions that will make these expressions true (i.e. will 'satisfy' them, hence the name of the problem)

N	Exhaustive Search Complexity 2^N, Total Number of Operations.	Random Walk Based Search Complexity 1.334^N, Total Number of Operations.
1	*2*	*1.3334*
2	*4*	*1.7796*
3	*8*	*2.3707*
4	*16*	*3.1611*
10	*1,048*	*17.7666*
20	*1,048,576*	*315.6523*
100	1.27×10^{30}	3.16×10^{9}

Table 3: Comparison of Computational Complexity of Solving 3SAT Problem Using Deterministic Exhaustive Search (Column 2) Versus a Set of Rapid Random Walks (Column 3) As a Function of the Number of Variables (Column 1).

Stochastic Hill Climbing. Randomized algorithms can be both *local* and *adaptive* [Hromkovic, 2003]. The difficulty of searching the solution spaces of most hard problems arises from the vast number of 'local minima' that some procedures ('local hill climbing based on gradient ascent') can get trapped into. Knowing this, algorithm designers have sought ways of 'getting out' of local optima [Sedgewick, 2011]. One way to do so is to randomly 'jump around' a solution search space and vary the rate and size of the jumps, using simple decision rules such as: 'search deterministically and locally in gradient-maximizing small jumps when hitting a promising solution' and 'jump far away when the gradient levels off'.

Such stochastic hill climbing methods (*SHC*) [Michalewicz and Fogel, 2004] ease limited search processes from the constraints of local optima by probabilistically causing the searcher to 'jump' to different regions of the search space and thus to perform large numbers of bounded local searches. SHC approaches to the problem of the optimal allocations of *N* decision rights to *M* agents (generate the entire search space, encode neighboring solutions (small differences in decision right allocations among people) in a consistent fashion, execute a probabilistic version of local neighborhood search in the vicinity of the allocation of decision rights the next jump lands on), of finding the core (or, vertex cover) of an influence network (generate all possible subsets of nodes, explore their connectedness in the network) can be used to significantly reduce the time required to find optimal solutions. The success of the approach depends on the distribution of optima in the solution search space: we can expect a 'rugged landscape' of many sharp peaks separated by large troughs in the first case (changing one decision right given to one person can make a very large difference) and a smoother landscape in the second.

Physically inspired algorithms [Xing and Gao, 2014] can take the form of intelligent randomization, in a form intuitive for non-computational scientists. *Simulated annealing algorithms* are based on the cycle of liquefying and cooling a metal or plastic in various forms, allowing the heating process ('going to a high entropy state') to generate the 'random search step' required to explore a potentially very large solution search space (all possible forms the cooled metal can take). Simulated annealing algorithms specify various *temperature gradients* of the search process: temperature denotes 'mean kinetic energy' of a state of particles assembled in a macro-state (e.g. liquid), and the *temperature of a search process* increases with the probability that the process will jump away from the local search it is currently performing within a certain time window.

The method can be applied by strategists without a specific model of the computational problem being solved. As one CEO anecdotally shared, he regularly runs meetings aimed at deliberating on and selecting organization-wide policies by first allowing the meeting to proceed without guidance or intervention from him, observing the process and guessing at the likely solution the group will converge, and then either encouraging the group to converge on the solution that seems to organically emerge or 'blowing up' the process - by making comments that undermine the credibility of the process itself, introducing information he knows not be known to others – depending on how globally attractive he believes the solution the group is about to converge on to be. This is a case of algorithmic insight being applied to a problem solving process without a specific model of the process itself: many algorithmic processes can yield to 'simulated annealing' approaches.

Biologically inspired algorithms can be used to synthesize useful heuristics and problem space search strategies in business, again, intuitively and often without the need for a precise computational model that encodes the problem at hand. Genetic *(or, evolutionary) algorithms* (GA) [Goldberg, 1989] combine the insight that randomization can produce (probabilistic) speedup of search with a structured approach to the solution generation process inspired from evolutionary biology [Baeck, 1996]. Using the basic operators of variation, selection, recombination and retention applied to populations of 'parent' solutions or components of solutions that can be concatenated or otherwise recombined, genetic programmers create algorithmic frameworks for solving complicated problems by the intelligent recombination and selection of randomly produced guesses at pieces of a solution. Primitive, random candidate solutions or 'partial solutions' (eg.: delta paths in a Lin Kernighan representation of *TSP*) are perturbed ('mutation') and combined ('sexual reproduction') to produce new candidate solutions that are then selected on the basis of the quality of the solution they encode [Fogel, 1995]. Mutation rate, selection pressure ('temperature' of the process) and recombination intensity (binary, *N*-ary) are parameters under the control of the problem solver. Exponential speedup of convergence to the shortest route was reported for the TSP [Wang, Zhang and Li, 2007] based on the parallelization of the randomization operator across members of a population of candidate solutions ('mutation', 'recombination') and the iterative application of the selection operator at the level of the entire population.

As we saw with simulated annealing, evolutionary algorithms offer the strategist a way of conceiving and structuring exploratory activity even when a precise encoding of the solution search space onto the space of 'genetic material' on which evolutionary operators do their work is not specified. For instance, organizational decision processes can be viewed as *selection mechanisms*, carried out over alternative policies, investments, technologies, or humans (hiring/firing/promotion). An analysis of selection mechanism described by evolutionary programmers [eg Baeck, 1996] – such as tournament selection, Boltzmann selection, roulette wheel selection, ranking based selection that is either 'soft' or 'hard' depending on the reversibility and corrigibility of the decision made– can be used as templates for alternative selection processes embodied in the organization, and as templates to experiment with alternative selection mechanisms that maximize organizational objective functions. Alternative or auxiliary goals and objectives can also be specified and embedded in the design of the selection (or recombination) procedure. They can include the 'diversity loss' in the offspring population associated with certain kinds of selection rules, the convergence rate of any selection scheme to the optimal solution allowed by the information embodied in the parent populations, and the 'sheer novelty' generated by a suitably designed mutation and selection process [Lehman and Stanley, 2010].

5. What is the best organizational *architecture* for solving the problem? Teams as multiprocessor units.

Software 'runs' on hardware. Computational designers regularly take advantage of hardware configurations to improve the efficiency with which algorithms solve the problems they were designed to solve. For shapers and solvers of business problems, studying and abstracting from the partitioning and porting of 'soft' algorithms onto 'hard' processors can offer a set of distinctions and choice points – as well as a set of normative schemata for allocating tasks to people from which deviations observed in practice can be systematically measured (which is the way what passes for purely descriptive theory is formulated in the 'special sciences'). A computational abstraction layer provides a unitary language system in which problem solving procedures are apportioned among processors and humans, enabling both strategic managers to speak more transparently and competently to developers and architects of algorithms and software, and those who primarily talk to machines to explain to strategic managers what they are doing and why they are doing it. The distinctions arising from understanding collective problem solving as a problem that partly relates to that of porting software on hardware architectures highlight the dependence on the optimal partitioning of tasks to problem solvers on the structure of the underlying problem and choice of problem solution procedure, as follows:

Topology: Centralized and De-Centralized. Just like an algorithm developer must consider the degree to which a central *controller* is used to schedule, allocate, distribute, and evaluate the output of different tasks (solution to sub-problems or intermediate stages of a problem solving process), business problem solvers must consider the degree to which decision rights over various components of a problem solving task are centralized in one central person or distributed across different problem solvers. Just as a *RISC* controller *manages* the inputs and outputs of tasks across other processors, for instance, a chief architect or chief developer must manage the inputs and outputs to different developers, which will enable them to do their design jobs most efficiently. The decomposability of a problem matters to the optimal structure of the processor units that solve it. Attempting to solve the *Traveling Salesman Problem* by breaking up the N locations into two groups of $N/2$ locations, solving each problem independently and then concatenating the solutions to generate a candidate solution is not a good idea, as the combination of two optimal circuits joining $N/2$ locations will not be equivalent to the optimal TSP circuit for the N locations. If humans were carrying out the tasks of enumerating and evaluating the paths for k collections of N/k sub-circuits, then significant coordination costs would result from having to piece together the optimal path joining N cities, and a central processing unit would have to expend significant effort to mitigate these coordination costs. On the other hand, a large strategy consultancy could break up the process of evaluating the allocation of M decision rights over decisions in K independent domains to a team of N people, there is a natural partitioning rule $N=K$ that allows for partitioning the problem to K different consultants (the processors), achieving an order of e^K reduction in the overall computational complexity of the task. And, finally, solving the TSP by exhaustive search may be effectively separated by decoupling the core tasks of *enumerating* all possible paths from that of *evaluating* the length of each path, from that of *finding the shortest path* on the list. This approach is clearly not scalable to large numbers of variables, but it offers the significant advantage that *enumerators, evaluators and selectors* can specialize, and lower the marginal cost of each operation, which is useful in situations where exhaustive search is feasible.

Less obviously, the choice of a solution method for a particular problem can affect the degree to which task centralization is beneficial. An evolutionary algorithm based approach to the solution of the TSP will be far more amenable to a decentralized, peer to peer task environment in which candidate circuits are quickly generated, and combined (decentralized) and then evaluated using an agreed upon selection procedure (centralized). Using a branch and bound approach to solve the Knapsack Problem for the set of optimal features of a product subject to total cost constraints can similarly be decomposed into parallelizable subtasks of evaluating the best and worst case scenarios that can result when traveling down each of the branches that include/do not include a particular branch. Solving for optimum strategies using backward induction similarly can proceed by de-centralizing the processes by which particularly interesting and important 'sub-games of the Nash tree' are evaluated.

Coherence: Synchronous and Asynchronous. As every real-time software developer knows, *the clock* – both its speed and the degree to which a multiprocessor architecture is synchronized to it - is a critically important component to implementing an algorithm in silicon in a way that allows it to process information of certain average and worst case complexity under time constraints of a maximum duration. The importance of the clock is easy to understand: the advantage of decentralization of task performance is often the parallelization and multiplexing of problem solving tasks. However, for a parallelized task to be significantly more efficient than its serial counterpart, the processors must be *synched up* in order to produce outputs that are jointly required at the next stage of the problem solving process. Some tasks are more sensitive to differences in processing speed than others: a genetic search among possible combinations of knapsack components will require tights synchronization of the transitions between *variation, recombination and selection* steps because delays compound over time, whereas a branch-wise decomposition of the same problem will be far more tolerant to slippage of the local clocks relative to the central clock.

Functionality: Homogeneous and Inhomogeneous. Programmers and the hardware designers that seek to offer them the structures on which their code will run know hardware can be optimally configured to perform certain kinds of operations – such as complex multiplications for Fast Fourier Transform Computation, add-select-compare processors for Trellis decoders – and so forth. When developers must commit to a hardware architecture *before* knowing in advance the kinds of operations their code will require, they opt for architectures that have *heterogeneous* subunits specialized for different kinds of operations. By analogy, business problem solvers called upon to solve wide ranges of problems across an industry or several industries (a large strategy consultancy) can usefully heed this 'diversity heuristic' in their own problem solving team constitution practice, and even optimize the 'diversity gain' they achieve for certain kinds of problems to exceed the sometimes higher costs of coordinating problem solvers trained in different disciplines [Moldoveanu and Leclerc, 2015]. A computational abstraction layer allows strategic problem solvers to quantify this diversity gain: Hong and Page [Hong and Page, 2011] show that a group of heterogeneous non-expert problem solvers outperform a group of experts on the task of finding a solution when the problem is complex enough that it can only be solved by generating and applying a set of heuristics whose value increases with its diversity.

7.Discussion. Intelligent Artificiality as a Computational Abstraction Layer for Strategic Problem Solving.

We are now in possession of a *computational abstraction layer* for business problem solving. Its expressive power is at least equivalent to that supplied by microeconomic theory - which has been the predominant net exporter of conceptual frameworks for business problem solving for the past fifty years. The language of marginal, average and sunk cost analysis, marginal value comparisons and marginal rates of substitution, strategic complements and substitutes, 'comparative' and 'competitive' advantage of various kinds, equilibria of both the Arrow-Debreu and Nash variety, temporal discounting and intertemporal substitutions, and so forth – can now be complemented, and, where inferior as a prescriptive and ascriptive tool, replaced, by elements of a computational abstraction layer comprising the following operators:

DEFINITION – the definition of problems via the specification of a set of current and desired conditions, time and resource constraints, and a space of independent, dependent and control variables whose cross or tensor products can be used to specify *solution search spaces;*

STRUCTURATION – the shaping, filtering or transformation of a problem such that the process of defining the problem, communicating the definition of the problem to others who need to be involved in solving it, observing and measuring current conditions, and designing methods of measurement for the desired conditions does not change or modify the values or nature of the variables in question and the structure of the solution search space;

SIZING- the process of evaluating the time and resources required to solve a problem before attempting to solve it;

ENUMERATION – the process of specifying, individually or synthetically, the set of solutions that satisfy the *constraints* of the problem;

EVALUATION – the process of evaluating candidate solutions with respect to their desirability;

RECURSION – the process of iteratively narrowing the solution search space through procedures aimed at synthesizing intermediate or along the way solutions, whose performance vis a vis desired conditions increases with each iteration, and which allow the problem solver access to a better approximation to a solution with each iteration;

ELIMINATION – the elimination of dominated or inferior candidate solutions with a view to narrowing the solution search space of the problem;

RANDOMIZATION – the implementation of operations in a solution search procedure that are probabilistically and not deterministically linked to the current state of the problem-solving process;

LOCALIZATION – the purposive narrowing of the search to a sub-space of the solution search space, with a view to simplifying the search;

PARALLELIZATION – the process of allocating different tasks in a problem solving process to different problem solvers, who can work in parallel and thereby reduce the total time required to attain an acceptable solution;

CENTRALIZATION – the process of vesting decision rights over the allocation of tasks and subtasks to different problem solvers as a part of the problem solving process;

SYNCHRONIZATION – the process of aligning and coordinating the 'clocks' of different problem solvers working on the same problem to as to minimize the losses resulting from coordination failures;

DIVERSIFICATION – the process of purposefully introducing heterogeneity in the core skills and capabilities of problem solvers, to take advantage of the *diversity gain* accruing to the application of a variety of different heuristics to solving a very complicated problem, or for the purpose of maximizing the 'insurance value' of the problem solving team when it is designed to confront a wide variety of heterogeneous problems.

We conclude with a discussion of the measurement of problem solving performance. Unlike economists, who focus primarily on allocational efficiency as a measure of process performance and focus their efforts on observable *outcomes*, usually compared cross-sectionally at the industry level, when they derive measures of comparative or competitive advantage, computational scientists focus on the *trade-offs* between and among the *accuracy, reliability and speed that a particular kind of processor implementing a certain kind of algorithm operating over and information set of a given size and dimensionality* that a given problem solving process (comprising silicon CPU's and memory, algorithms, and data sources) can achieve in the best, worst and average case scenarios expected.

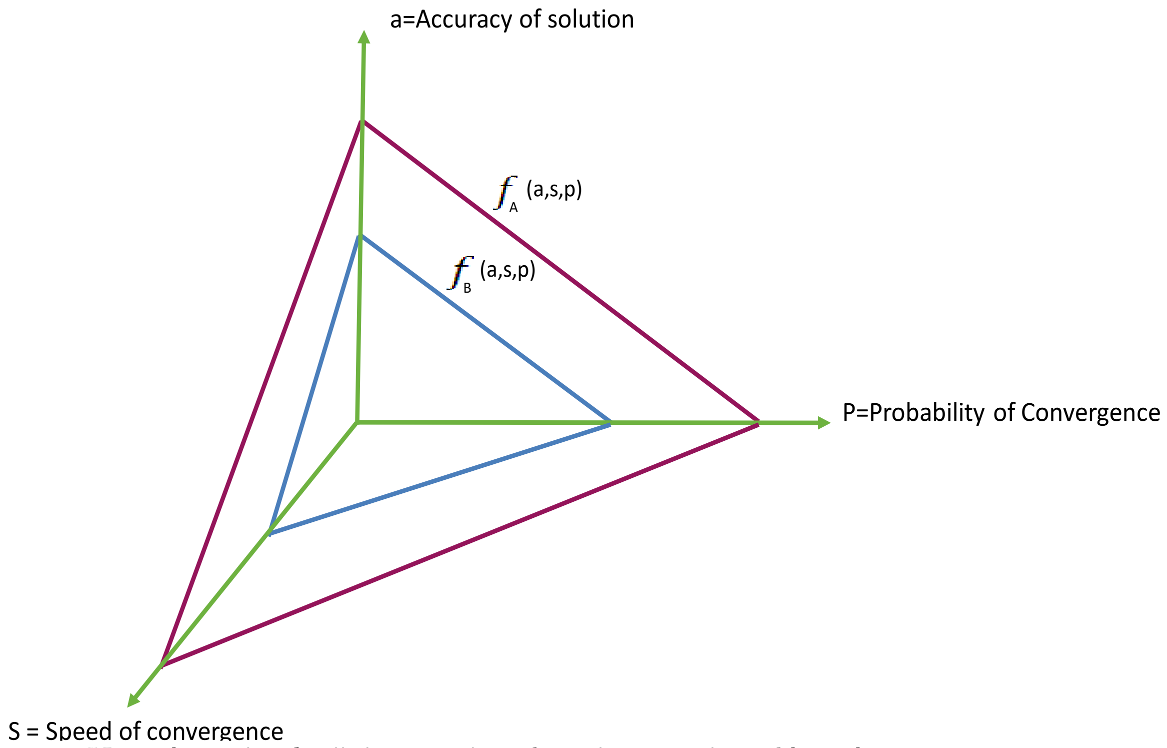


Figure 17. Hyperplanes of trade-offs for quantifying the performance of a problem solving process, comprising the accuracy of the solution generated by the process, the probability of converging to a solution of that accuracy, and the speed (inverse of implementation time) with which a solution of a particular accuracy can be reached with a particular reliability.

This three-dimensional measure of problem solving performance allows for the *contextualization* of the measurement of the prowess of a team, group, organization or value linked activity chain that transcends organizational boundaries to solve the specific problems it confronts by being specific to the structure and definition of the underlying problem. But it allows for the *transfer* of learning about problem solving across different problems of the same structure, and therefore within firms, across firms and across industries. It unpacks the black box of 'process' that economic models focused on measuring the efficiency of processes by reference to inputs and outputs alone leaves opaque by being specific about the individual *operators* and *operations* of problem solving, and about the link between the problem structure, problem solving policy and resulting performance.

The three-dimensional approach to the quantification and measurement of problem solving performance contributes an alternative definition of comparative advantage of a firm over another, and, where the firms are operating in a contested market domain – of their respective competitive advantages. In Figure 17, firm A dominates Firm B along all dimensions of problem solving prowess – speed, accuracy and reliability – in solving problems of a particular structure. This *unpacks* comparative advantage in ways that illuminate and point to improvement paths: by focusing on reducing the costs of the basic organizational operations that comprise 'everyday problem solving', or by altering the topology and coordination mechanisms of parallel problem solving groups; which is what an abstraction layer that does not merely 'represent' or 'describe' experience and practice - but also reveals guideposts *to* and levers *for* changing them - should do.

References

Akerlof, G. A. 1970. The Market for "Lemons": Quality Uncertainty and the Market mechanism. *Quarterly Journal of Economics*, 84(3): 488-500.

Arora, S., Barak, B., Brunnermeier, B. and Ge, R. 2010. Computational Complexity and Information Asymmetry in Financial Products . *Innovations in Computer Science (ICS) Conference.*

Austrin. P., M. Braverman and E. Chlamtac. 2011. *Inapproximability of NP Complete Variants of Nash Equilibrium*, arXiv:1104.3760v1.

Back, T., 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press.

Baer, M., Dirks, K.T. and Nickerson, J.A., 2013. Microfoundations of strategic problem formulation. *Strategic Management Journal*, *34*(2), pp.197-214.

Bhardwaj, G., A. Crocker, J. Sims and R. Wang. 2018. Alleviating the Plunging In Bias, Elevating Strategic problem Solving. *Academy of Management Learning and Education, 17.3.*

Brin, S., and L. Page. 1998. Bringing Order to the Web: A Hypertext Search Engine. Mimeo, Stanford University.

Charikat, M. 2000. Greedy Approximation Algorithms for Finding Dense Components in a Graph. *Proceedings of APPROX:* 84-95.

Chen, X., X. Deng and S.-H. Teng. 2009. Settling the Complexity of computing Two-Player Nash Equilibria. *FOCS. J. Association for Computing Machinery,* 56:3.

Christian, B. and Griffiths, T., 2016. *Algorithms to live by: The computer science of human decisions.* Macmillan.

Cook, S. 1971. The Complexity of Theorem Proving Procedures, *Proceedings of the Third Annual ACM Symposium on the Theory of Computing.*

Cyert, R.M. and J.G. March. 1963. *A Behavioral Theory of the Firm*. New Jersey: Prentice-Hall.

DeMarzo, P.M., 2004. The pooling and tranching of securities: A model of informed intermediation. *The Review of Financial Studies*, *18*(1), pp.1-35.

Dixit, A., 1990. *Optimization in Economic Theory*. New York: Oxford University Press.

Ermann, L., Frahm, K.M. and Shepelyansky, D.L., 2015. Google matrix analysis of directed networks. *Reviews of Modern Physics*, *87*(4), p.1261.

Fogel, D.B. 1995. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway.

Fortnow, L. 2009. The Status of P versus NP Problem. *Communications of the ACM*. 52(9). 78-86.

Garey, M.R., and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP Completeness*. San Franscisco, Freeman.

Ghemawat, P. and Levinthal, D., 2008. Choice interactions and business strategy. *Management Science*, *54*(9), pp.1638-1651.

Ghosh, S.K., and Misra, J. 2009. *A Randomized Algorithm for 3-SAT*. Honeywell Technology Solutions Laboratory.

Goldberg, D.E. 2008. *Genetic Algorithms*. Perason Education.

Holland, J. H. 1962. Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM*. 9(3): 297-314.

Hong, L. and S.E. page. 2004. Groups of Diverse problem Solvers can outperform groups of high ability problem solvers. *Proceedings of the National Academy of Sciences*. 101(46), 16385-16389.

Hromkovic, J. 2003. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation and Heuristics*. 2nd edition, Springer, Heidelberg.

Jensen, M.C. and W.H. Meckling. 1998. Specific and general Knowledge and organizational Structure. In Jensen, M.C. , *Houndations of Organizational Strategy*, Boston: Harvard Business School Press.

Karp, R.M. 1972. Reducibility Among Combinatorial Problems. In Miller, R.E., and J.W. Thatcher, eds., *Complexity of Computer Computations*. Plenum, New York.

Kauffman, S. and Levin, S., 1987. Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, *128*(1), pp.11-45.

Knuth, D.E. 2011. *The Art of Computer programming*. New York: Addison Wesley.

Lehman, J. and Stanley, K.O., 2010, July. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (pp. 837-844). ACM.

Leibenstein, Harvey (1966), "Allocative Efficiency vs. X-Efficiency", *American Economic Review* **56** (3): 392–415

Levinthal, D., and P. Ghemawat. 1999. Choice Structures, Business Strategy and Performance: An NK Simulation Approach. Working Paper 00-05, Wharton School.

Levinthal, D.A., and Warglien, M. 1997. Landscape Design: Designing for Local Action in Complex Worlds, *Organization Science*, 10(3); 342-57.

Lin, S., and B.W. Kernighan. 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*. **21**. 498–516.

Lyles, M. and I. Mitroff. Organizational problem Formulation: An Empirical Study. *Administrative Science Quarterly*. 25:1. 102-119.

Majorana, E.2006. The Value of Statistical laws in the Physical and Social Sciences. in Bassani, G.F. *scientific Papers of Ettore Majorana*. New York: Springer.

March, J. G.1991. Exploration and Exploitation in Organizational Learning. *Organizational Science*. 2: 71-87.

March, J. G., Simon H. A. 1958. *Organizations*. John Wiley, New York

Martello, S., and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, New York.

Michalewicz, Z., and D. Fogel. 2004. *How to Solve It: Modern Heuristics*. Springer, Heidelberg.

Mintzberg, H., Raisinghani, D. and Theoret, A., 1976. The structure of" unstructured" decision processes. *Administrative science quarterly*, pp.246-275.

Moldoveanu, M. and Reeves M., 2017. 'Artificial intelligence: The gap between promise and practice.' *Scientific American Online*. November.

Moldoveanu, M. and Leclerc, O., 2015. *The Design of Insight: How to Solve Any Business Problem*. Stanford University Press.

Moldoveanu, M.C. 2011. *Inside Man*: *The Discipline of Modeling Human Ways of Being.* Stanford Business Books Stanford.

Moldoveanu, M.C. 2009. Thinking Strategically About Thinking Strategically: The Computational Structure and Dynamics of Managerial Problem Selection and Formulation, *Strategic Management Journal*. 30: 737-763.

Moldoveanu, M.C. and R.L. Martin. 2009. *Diaminds: Decoding the Mental Habits of Successful Thinkers*. Toronto: University of Toronto Press.

Nelson, R. and Winter, S., 1982. An evolutionary theory of the firm. *Belknap, Harvard*, *41*.

Novak, S. and Wernerfelt, B. (2012), On the Grouping of Tasks into Firms: Make-or-Buy with Interdependent Parts. *Journal of Economics and Management Strategy*. 21: 53–77

Page, L. 2001. Methods for Node Ranking in a Linked Database. *United States Patent*.

Porter, M.E. and Nohria, N., 2018.How CEO's Manage Their Time. *HARVARD BUSINESS REVIEW*, *96*(4), pp.41-51.

Posen, H.E., Keil, T., Kim, S. and Meissner, F.D., 2018. Renewing Research on Problemistic Search—A Review and Research Agenda. *Academy of Management Annals*, *12*(1), pp.208-251.

Reeves, M, Haanaes K., and Sinha J., 2015. *Your Strategy Needs a Strategy*. Boston, Harvard Business School Press.

Rivkin, J. 2000. Imitation of Complex Strategies. *Management Science*. 46: 824-844.

Schoening, U. 2002. A Probabilistic Algorithm for k-SAT Based on Limited Local Search and Restart, *Algorithmica*, 32: 615-623.

Sedgewick, R. and Wayne, K., 2011. *Algorithms*. Addison-Wesley Professional.

Siggelkow, N., and Levinthal, D. 2005. Escaping Real (Non-Benign) Competency Traps: Linking the Dynamics of Organizational Structure to the Dynamics of Search. *Strategic Organization*. 3(1): 85-115

Siggelkow, N., and Levinthal, D. 2003. Temporarily Divide to Conquer: Centralized, Descentralized, and Reintegrated Organizational Approaches to Exploration and Adaptation. *Organizational Science*. 14: 650-669.

Simon, H.A., 1973. The structure of ill structured problems. *Artificial intelligence*, *4*(3-4), pp.181-201.

Valiant, L. 2006. *Probably Approximately Correct*. Cambridge: Harvard University Press.

Valiant, L.G., 1984. A theory of the learnable. *Communications of the ACM*, *27*(11), pp.1134-1142.

Wang, L., J. Zhang, and H. Li. 2007. An Improved Genetic Algorithm for TSP, *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics*, Hong Kong.

Weinberger, E.D. 1996.NP Completeness of Kauffman's *N-k* Model: A Tunably Rugged Energy Landscape. *Santa Fe Institute Working Paper 96-02-003*.

Wing, J.M., 2006. Computational thinking. *Communications of the ACM*, *49*(3), pp.33-35.Wing, 2008

Wing, J.M., 2008. Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, *366*(1881), pp.3717-3725.

Wolfram, S. 2016. 'How to teach computational thinking.' *Stephen Wolfram Blog*.

Wright, A. H., Thompson, R. K., and Zhang, J. 2000. The Computational Complexity of N-K Fitness Functions. *IEEE Transactions on Evolutionary Computation*. 44(4): 373-379.

Wolfram, S, 2016. 'How to teach computational thinking.' *Stephen Wolfram Blog*.

Xing, B. and Gao, W-J.. 2014. *Innovative Computational Intelligence: 134 Clever Algorithms*. New York: Springer.