Implementing Brazing Alloy Design with Man-Machine Interaction

Find out how machine learning can be used to compute alloy compositions

BY LEO A. SHAPIRO AND ALEXANDER E. SHAPIRO

A human expert can use a machine assistant that combines neural network and logical artificial intelligence methods to design new alloys more quickly by making better use of existing data. This is easiest to do if the machine assistant is an advice-taker program.
Suppose you want to make a system that sufficiently often correctly predicts the composition of a filler metal for use in joining some given base materials according to some given constraints on the properties of the resulting joints, the filler metal, and the joining process (Ref. 1). For example, you expect the resulting joints to have shear strengths according to a given probability distribution or better, corrosion resistances greater than a given value, and the liquidus of the filler metal to be in a given range. Suppose that one or more human experts can, sufficiently infrequently, give this system advice or clues, but otherwise it works autonomously and comprehends, explicates, and uses such advice (Ref. 2).

Computing Compositions

Machine intelligence would be more useful in practice, and specifically in the industries that manufacture physical goods, if much less data were required for a machine intelligence to propose compositions that satisfy all the “unique-in-that-production-process-as-done-by-that-firm” constraints that occur in real industry. Furthermore, it would be ideal if it could take just a little advice from a human expert to supplement a preexisting set of data to transform almost-but-not-quite satisfactory compositions (ANQCs) into completely satisfactory compositions, because such ANQCs cannot be manufactured or used by a customer. That is, completely new conjectures by the machine might be satisfactory and yet lack certain very desirable aspects of the ANQCs now already known.

Unpredictability is a cost. Constructing a much larger set of training data and hoping that will improve the performance of the machine specifically as desired is too expensive. Not only is acquiring and organizing more data costly, but there is also no guarantee that the data added to the revised set will be relevant to train the machine to correct those near misses that interest the human user, and not even more data will be required and so on. Most possible compositions are useless, and the space of all possible compositions is too large to randomly consider or try one composition after another. There is a budget to follow, but every decision, especially if it involves an experiment, incurs a nonnegligible cost (Ref. 3). A human expert therefore designs a brazing filler metal by using his or her knowledge of the outcomes of experiments and the properties of materials together with some heuristics (Refs. 1, 3–8).

The expert might use, for example, Hume-Rothery rules 1, 2, and 3 (Ref. 9) and search for more easily predictable or otherwise produced-by-a-rule proxies or signs of alloys with greater strengths, such as combinations of elements having greater mixing entropies (Ref. 10). The expert might use a Darken-Gurry map, wherein each element corresponds to an atomic radius and electronegativity pair. In this space, each such pair is the center of an ellipse with leftmost and rightmost points at minus and plus 15% of the atomic radius of that element. All the pairs inside the region with this ellipse boundary correspond to elements that are most likely soluble solutes if the element that corresponds to the pair in the center is considered as a main solvent in a brazing alloy composition according to Hume-Rothery rule 1 (Ref. 11). Then, rather than use Hume-Rothery rule 4, he or she might instead use the Hildebrand solubility parameter as a better abstraction that has only a single value for each metal at its melting point (Ref. 12). The mutual solubility of the elements of a filler metal composition is significant for obtaining solid solutions as the main phases in most resulting brazed joints.

The expert might vary his or her representations of information by replacing electronegativity with Hildebrand solubility to get a Darken-Gurry map (Fig. 1) and use this with a Darken-Gurry map, varying the radii of the ellipses such that the viable region of one diagram is inside the viable region of the other diagram.

All such factors may be assigned initial weight coefficients of relative importance, and these might be randomly distributed and therefore like initial tentative probabilities that are revised after more information comes in and can always be assigned to events. This is normally part of what a human expert does mentally, and now it can be done using a machine and combined with neural network methods (Refs. 7, 13–17). Much smaller initial coefficients might be given to factors other than those that are the main heuristics. Data might be compositions and various actual measured properties when used to join multiple base metals. If there is enough data such that comparisons between predictions and known results happen sufficiently often, a network of heuristics can be used to generate compositions and predictions. Deviation from actual measured results can be used to revise the weight coefficients attached to different heuristics and their various parameterizations.

Challenges Encountered

Many such methods for determining the alloy filler metal composition existed in the distant past. What did not exist back then were sufficiently powerful computing machines (Refs. 16, 17). A challenge when people use such methods is they often require sets of training data that must be so large that the cost of constructing them, and especially the time it takes to do so, severely restricts the feasible uses of such methods. Another obstacle is the ability of such systems to find solutions of different types is unstable over data or too sensitively depend on the data sets. Our civilization needs to overcome such limits because our real issue can be called the long tail problem of materials (Ref. 18).

Too many remaining challenges are complex problems, and most of them can’t be solved if people can’t quickly create special purpose materials with unique properties, each of which solves only a few but very different subproblems. When there are too many of these and they have perhaps nothing in common, each subproblem is a bottleneck. That means people must create materials very inexpensively because special purpose materials have tiny markets and cannot be sold in large volumes. In such cases, people cannot recoup research costs unless those costs, too, are small enough.

Consider all the problems where the solutions are constructed or found by examining or testing different cases. Suppose that each case is a system of related and interacting factors together with the parameters of these relations; the relations of the factors can also be such systems. Too often a set of human (or machine) experts has enough usable (but mostly fragmentary) knowledge to significantly reduce the size of the search for the solutions but not enough knowledge to directly
solve the problems if there are too many combinations. In other words, there are more cases than what can be searched manually or even automatically if searching further costs something. But the number of these cases isn’t always increasing, that is, not infinitely; hence, most formal and mathematical methods also can’t be profitably used. We can only ask what tool or machine can enable those experts to use their fragmentary knowledge to have a significantly better chance to find a solution (Ref. 19).

The free design of a material, given some constraints or goals, is a typical problem of intermediate-scale combinatorics, and most problems in this class are solved only by somehow becoming able to use enormous quantities of too-loosely-structured and incomplete data. This is such that probabilistic or thermodynamic abstractions don’t work because too many actual or possible relations in that data must also be taken into account and can’t just be abstracted away (Refs. 20, 21).

**Language**

Language is an issue: With all too few exceptions, people can’t or won’t solve problems that they can’t or don’t discuss. But more often than not, they don’t talk about anything that intrinsically involves too much conditional-ity or complexity — people mostly discuss what they like. They don’t like to talk about anything they can’t describe concisely. What people can comprehend is that which can be sufficiently well compressed or discussed concisely (Refs. 22, 23). Most systems that have too many conditional behaviors and parts reoccur too infrequently in nature. Thus, people have created no short words that point to such things, and too many words must be defined in the places where they are used if people discuss systems that are too conditional. It is almost impossible to use natural or even mathematical languages if the discussion must also proceed concisely (Ref. 20). In other words, “more is different” (Ref. 24).

Yet over time, this one class of problems becomes more and more significant in practice; ultimately, most remaining industrial problems are in this class (Ref. 25). So simply leaving such problems unsolved, as if they are mere exercises, is not a feasible, real option, as stagnation is never a real option. Human conditions improve and human resources increase when people solve their problems, and that mostly happens when people use technologies. However, unsolved significant problems create accumulating costs and a corresponding decline in human conditions and resources. But every set of technologies itself creates other and more complex problems, especially interacting technologies. Furthermore, creating or using yet another set of technologies to do something about all these new problems too often requires some level of sufficient resources and good-enough conditions. Most problems cannot be solved only because there is not enough time, and we can observe that many problems in the past were solved either quickly or never. That is the role of time. So there is always either increase or decrease, never stagnation, and a time limit, hence an urgency (Ref. 25).

Every industry that grows necessarily develops methods that solve more difficult problems more quickly, but over time, more and more of the problems that remain involve the combinatorics of the intermediate scale, “where more is different,” specifically, very large and complex systems (Ref. 24).

Natural language describes things with a large amount of redundancy so that not too many misunderstandings occur in the communication of information if errors like omissions happen. So natural language describes well mostly unconditional circumstances concisely and reliably, thus providing reliable but inexpensive...
sharing of especially significant contextual information between people about the circumstances in which they most frequently found themselves and what they most wanted to discuss in the past (Ref. 26): usually the behavior of people, memories, and the appearances of things. Such language therefore also poorly describes any system that sensitively depends on too many significantly different conditions or parts that conditionally interact and have a highly intricate structure, like in computation, because its use here produces sentences that are too long (Ref. 20). Complex programs and historical novels both resist being well summarized, and both tend to be very long.

But this works in both directions — because these are the very same reasons that most computations do involve conditionally interrelated and yet fragmented data, so working profitably with conditionally interrelated and fragmentary data needs computation (Ref. 19). All of the solutions to complex problems can only be computational solutions. Meanwhile, the whole language issue is one of the major subproblems in man-machine communication and hence a critical subproblem in solving all other complex problems. Therefore, all solutions will have something to do with language. This is the sense in which computational solutions to problems have been compared with a notion of magic (Ref. 27), because where else does describing something using a slightly wrong word or model cause inefficiencies or real things to not work? In what other problems of production of physical things can people ever get more done or something made merely by talking differently about it?

Today, powerful computing machines exist that cost sufficiently less compared to everything else, and they are common enough that mostly pragmatic methods are suddenly much more feasible than mostly realistic methods. In the past, the opposite was the case because there were no machines with all of these characteristics.

Method

Realistic methods of solving problems involve theories: Rules that make sufficiently few errors so that using very few such rules together is enough to find solutions to problems with acceptably few errors. In contrast, pragmatic methods instead involve heuristics: Rules that possibly make very many errors so that very many such rules together, along with switching and searching, are needed to find solutions to problems with acceptable few errors (Refs. 1–8, 17, 19, 28). We always mean possibly an empty set having nothing in it whenever we speak about a set of things. Now suppose there is an initial set of data and a record in which data can be written, and sometimes what is written is nothing and that overwrites something. Additionally, there are “operations,” each of which can take some subset of the data in the record and then write a set of pieces of data in the record. Operations themselves can also be data. We call a sequence of operations that which writes a solution to a problem in the record before or when it completes a productive sequence over the set of data. So it is a path in a graph (Refs. 3–7, 28–30).

One or more of the operations that is used in a pragmatic method is such that in too many cases the productive sequence does not solve the problem correctly, and there is no guarantee that any productive sequence will (Refs. 3–7, 28).

Furthermore, often a problem comprises many concurrent subproblems (Refs. 3–5, 28–30), and each subproblem can then be delegated to subsystems. This suggests many cooperating general problem-solving actors (Refs. 31–33).

Notice that all of this applies to how machines can cooperate with other machines, how people can cooperate with other people, and how people can cooperate with machines. We conjecture what is needed for the design of alloys in the sense of the long tail problem of future manufacturing of composite large systems, like future vehicles, to be some automation of human-like expertise. One or more human experts can advise one or more machine experts or many cooperating general problem-solving actors that are significantly more numerous than people. As a result, people, using few words, can delegate complex subproblems in significantly more, deeper, and longer chains of workload delegation (Refs. 2, 3, 31, 33).

Today, most computing systems already comprise many concurrent cooperating subsystems that either encapsulate many other concurrent cooperating processes or else are themselves sequential processes. Subsystem processes communicate by sending private messages directly whereby only one actor can read a message. Reading the inside of another actor is really sending it a message that requests a copy of its inside structure, which can then be sent and received as a message or by updating records or public messages, or it can involve a notion of location such that communication can have an area of effect (Refs. 8, 34–36). Messages can be pure data or actors that comprise data and computing capability or a mix thereof. Pure data cannot change itself unlike an actor in between its interactions with other actors. But a database with operations embedded in it can be a message and can change itself, according to the operations and control logic it contains, after being sent but before being read; for example, while its name is in a buffer mailbox of things to read in some order (Refs. 8, 37).

Human and Machine Experts

The practical emphasis in solving problems is on a human expert being able to advise a machine regarding complex problems very concisely despite the fact that complex systems in any natural or even mathematical language are difficult to discuss concisely; the machine experts need both common sense and network computation to succeed (Ref. 2). Although many human experts can be involved in such problem-solving processes, “more is different” (Ref. 24). What we are referring to here is hundreds of thousands or millions of machine experts working together with a set of human experts to solve most problems within a few seconds, minutes, or hours rather than how long it would actually take to organize an equal amount of human experts who should then work together.

We can consider some problems to be near others in a space of problems that is relevant to being able to define expertise in solving that type of problem and being able to point to an expert. An expert is not necessarily somebody who already knows the answer to enough problems in a space of problems. Rather, he or she is somebody who can sufficiently often predict what questions should be asked or what experiment should be performed to get the information needed to solve the problem. If he or she can do this
How do we select a framework and therefore the trade-offs we’ll live with? Why not consider the computing machines we’ll have to actually work with?

The most widely mass-produced and easily acquired computing systems today are all massively parallel, distributed, self-timed machines that also have unlimited memory and unlimited storage for most practical purposes. They are all reasonably affordable, and most people and organizations already have them. That was not the case in the past.

For example, a personal computer today can be equipped with more than a trillion bytes of memory and more than a hundred general-purpose processor cores, each of which performs several billion basic operations per second; so, in practice, in the programming languages we use, we need to be able to apply functions to data in a massively concurrent manner.

Consider a programming language in which there are mostly lists and these lists describe sets (order doesn’t matter) and where white space is not significant. So, for example, [[A, [B, X]], [C, Y]]], ..., is the same as [[A, [C, Y]], [B, X]]], ...].

Every first object in a pair labels an attribute in a frame (here it is a set) whose value is the second object in that pair; furthermore, every frame can be a possible value of an attribute. Nesting objects inside objects with arbitrary depth is possible in this language. Each recipient of a piece of data decides how to read an ambiguous list. But most often a name of an attribute simply isn’t further divisible, that is, not a list.

An action or function is an operation that takes a thing and transforms it or something else and does this conditionally or possibly unconditionally upon what it first takes is written as an outermost pair of objects, which is constructed using the semicolon separator instead of the comma separator. Order here is not significant and some data and none or one or more operations can be present in any given order. For example, [[F1; X], [[A, M], [B, N]], [P2; Y], [F3; Z]], ...].

Function application in such a massively distributed language is preference.

Creating a new actor is a possible action that can be done, and ceasing to exist is another operation that can be done. But the operations that change an actor must be done by that actor itself: Other actors can only request an actor to do something and any actor can always refuse a request, preventing deadlocks, for example, in the case of some error.

A system that is an advice taker must also be able to directly infer many implications of any data or advice and treat these implications too as data. In other words, it must have common sense, and this combined with the use of heuristics means it makes defeasible conjectures and interpretations based on the data or exactly how human science proceeds (Refs. 2, 38). However, networks of (simple) neurons that are not general problem-solving actors don’t do this (Ref. 33). Suppose that our man-machine system comprises one or several independent problem-solving machine processes such that each one begins searching somewhere in a space, and they can move from place to place in that space by taking an action and will mostly work with notes, data, and heuristics until they converge to compositions where they stop and output their conjectures.

They should be able to ask questions, and a human expert should be able to concisely answer these questions in a programming language but with a natural or mathematical language flavor. This language should be its own metalanguage, which means it will be somewhat ambiguous or inconsistent, but that has no other consequences because these systems already use heuristics to reason, parse, and conjecture. Problem-solving actors are concurrent and self-timed. They could be embodied one per processor if there were enough processors and each actor could create another actor at any time and delegate it some workload by a message or by creating it with a workload in it, and each message could be an an actor (Refs. 31, 34, 35, 39). Creation of an actor by an actor makes it possible for actors to use genetic and selection algorithms (Refs. 40, 41) inside coalitions of networked general problem solvers as if they were complex neurons and arbitrarily mix pragmatic heuristics and stricter rules, some of which could be realistic such as formal theories over data.

Each problem requires using some operations more than others to solve it, but each definite manner of working with data involves a trade-off regarding which operations on data are both possible and feasible and which ones are only merely possible (Ref. 37). Like a large ship is also slow because it is so large, and a small ship is fast because it is so small, everything is good for the same reason that it’s bad; there’s always a trade-off.

How do we select a framework and therefore the trade-offs we’ll live with? Why not consider the computing machines we’ll have to actually work with?

Coming Up with a Framework

The most widely mass-produced
A piece of data is a tacit request to do work on it, and each actor can use any of its operations on its workload if no exceptions are triggered. Exceptions are operations that are done first if they are present, and they can temporarily conditionally change the probabilities of other operations that an actor has equipped and can temporarily equip or disable operations. If the workload contains as part of the data a preference that is a name of an operation (and that operation is available in that actor or that actor can request it from its network of notes or actors, then it can negotiate successfully to request and receive a copy of it or they from their notes, and so on), then that operation will be considered to be tried first on this workload. If more than one preference is present, like in the example above, and they have weights X, Y, Z then the actor in this example will first try function F2 with a random chance of Y/(X+Y+Z) if it is available. If the result is invalid or judged later to be useless, the actor can backtrack and try other operations it has, after trying all three. So if F2 was tried and it failed or was not actually applicable to that load, then it is pruned leaving F1 to be tried before others with a probability frequency X/(X+Z) and F3 to be tried before others with a probability frequency Z/(X+Z) of being chosen next. Pruning can be done by simply deleting from memory because actors only perform operations after making a copy of themselves together with that single workload after selecting randomly one workload to process next. This copy is called an instance-like actor, and the original actor that selects the load to be worked on next is called its class-like actor. The one simply deletes the load that it delegated to the instance it created previously while the other simply deletes the operations it tried that failed, and instance-like actors terminate themselves (along with any data) when they have an empty set of operations remaining to try on their piece of data. The system does not get stuck, but it can delete information. This is important during explicit abstraction, and further operations can be added at any time with no risk of coding errors or harmful operations from breaking or too severely disabling the advice taker.

However, we might have a phrase or a frame as the probability weight and that refers to a model or to weights encapsulated in an actor (experience). An actor might have exceptions that say how to process a model corresponding to an abstract phrase to derive a definite probability weight there. These correspond to objective vs. subjective notions of probability. The model itself would be provided in some network of other linked actors if that model happens to be one that can autonomously change over time or is directly stored in the memory of the actor if mostly static over time. The model could be explicitly in the workload after the semicolon, for example, \( [P;M] \), and if we replace M with a semantic network representation of a model, for example, \( [\text{Causes}(A,B), \text{Causes}(C,D), \text{OnTopOf}(A,C)] \).

Probabilities are always conditional on the circumstances; furthermore, probability theory is about correctly calculating other probabilities from known probabilities and not about the source of the initial probabilities or their type. They can be frequencies or the initial probabilities can even be assigned randomly and subjectively and updated like the weight coefficients in a neural network. Actors can assign a probability tentatively to any event provided they can truly improve this estimate when they operate with data and iterate the process of assignment, and the set of current probabilities can be considered knowledge (Refs. 13, 15, 42). However, general problem solvers also contain knowledge in terms of the set of operations they equipped and after discarding some operations, and operations that can be passed around like data between actors when they work — copied, equipped, or unequipped. This can be called conmessaging and an affine productive sequence of operations where the ordered weight coefficients are the abstract critical parameter in a sequence wherein no conmessaging occurs. Coaffine arcs comprise entirely conmessaging events, and they prepare other actors to converge to affine behavior. Actors can improve their chance of finding satisfactory compositions by revising the probability with which they use different operations on data or else by pruning or equipping different operations that they can affinely use later.

Probability is actually not only a frequency, number, weight, or function but is really a functor (Ref. 43) that takes a network and produces a simpler network with fewer vertices or edges. This is possibly at the cost of decorating edges with constructed data that makes the selection of the next operation uncertain if sequences of operations are paths, all edges are operations that write data in a record, vertices are all pieces of data, and the assignment of probability must reproduce enough of the data in the larger original graph if run through repeatedly. There are connections to the ensembles of Boltzmann and Gibbs and classical thermodynamics as always used as a typical probability functor, and affine paths can be viewed as straight lines. Pattern recognition and solving problems is really compression of fragmentary information that actors hold knowledge of in the form of probability (Ref. 44).

We can identify where the system lacks enough knowledge and what the advice specifically should be about so that it would improve the reasoning of the machine dramatically. The lengths of sequences of operations also encode probabilities and thus pragmatic knowledge if two random walks (sequences) through a search space, such as two sequences of operations that some actors perform while solving problems, might be two paths having very different shapes and lengths, but all the same they might have almost the same probability of occurring in that space if the shortest distance in the space between where each path begins and ends is the same for both random walks and if those random walks that end less far from where they start along the shortest possible path in the space have a greater conditional probability than those that terminate farther along the shortest path there. The conditions or contexts of these sequences are more comparable if they are contained in closer local regions of the same search space. They are maximally random or purely trial and error guesses where the system apparently has no knowledge that is relevant to solving the problem if, conversely, two sufficiently comparable paths such as those in the same region have very similar probabilities but the shortest path between the endpoints of one random walk is arbitrarily longer than the shortest path between the end points of the other random walk, that is, percolation occurs (Ref. 45).

An important aspect of nature can also be represented beside human experience in the sense that systems do not require precise specification, communication, or measurement of all microvariables or details but only a few out of any such set and only with a
limited degree of precision, and nothing is used that is not communicated. The result should usually be stable and the same or almost the same because local units simply don’t use or need detailed information about their neighbors to produce stable and predictable systems in the large — Born’s principle. In other words, while we focus on making pragmatic modeling easy, we also want to make realistic modeling possible and feasible simply because it just so happens that with this kind of system we can.

The human expert can express his or her practical experience and rules of thumb. A pragmatic program is nature-like and scales for use by a team if the machine is already computing a solution to, requires much less computation than we understand something and it and still know what it is or what it do it more easily if we can omit part of those the machine is already computing.

Conclusion

Human experts can concisely communicate extremely conditional statements about complex systems, such as materials that must be joined under some constraints, to machine advice takers if these comprise networks of cooperating general-problem-solver actors that pass around framed and organized knowledge that also conveys preference information to these actors. People can reduce expert knowledge to such statements and that allows human experts to express their intentions to machines directly and specifically regarding just those points where the machine made an error or proposed something that is not useful in the opinion of those human experts, whereas most of what the machine is thinking is already satisfactory. Being able to inexpensively improve upon ANQCs with little additional information significantly increases the probability that machine intelligence can actually help people solve the long tail problem in the materials industry and other practical challenges that are like it. [W]

References