Applied Mathematics, 2014, 5, 3489-3510
Published Online December 2014 in SciRes. http://www.scirp.org/journal/am
http://dx.doi.org/10.4236/am.2014.521327

Scientific Research

Causal Groupoid Symmetries and Big Data

Sergio Pissanetzky

School of Science and Computer Engineering, University of Houston, Clear Lake, Texas, USA
Email: Sergio@SciControls.com

Received 25 September 2014; revised 20 October 2014; accepted 12 November 2014

Abstract

The big problem of Big Data is the lack of a machine learning process that scales and finds meaningful features. Humans fill in for the insufficient automation, but the complexity of the tasks outpaces the human mind's capacity to comprehend the data. Heuristic partition methods may help but still need humans to adjust the parameters. The same problems exist in many other disciplines and technologies that depend on Big Data or Machine Learning. Proposed here is a fractal groupoid-theoretical method that recursively partitions the problem and requires no heuristics or human intervention. It takes two steps. First, make explicit the fundamental causal nature of information in the physical world by encoding it as a causal set. Second, construct a functor F: $C \Rightarrow C'$ on the category of causal sets that morphs causal set C into smaller causal set C' by partitioning C into a set of invariant groupoid-theoretical blocks. Repeating the construction, there arises a sequence of progressively smaller causal sets C, C', C'', ⋯ The sequence defines a fractal hierarchy of features, with the features being invariant and hence endowed with a physical meaning, and the hierarchy being scale-free and hence ensuring proper scaling at all granularities. Fractals exist in nature nearly everywhere and at all physical scales, and invariants have long been known to be meaningful to us. The theory is also of interest for NP-hard combinatorial problems that can be expressed as a causal set, such as the Traveling Salesman problem. The recursive groupoid partition promoted by functor F works against their combinatorial complexity and appears to allow a low-order polynomial solution. A true test of this property requires special hardware, not yet available. However, as a proof of concept, a suite of sequential, non-heuristic algorithms were developed and used to solve a real-world 120-city problem of TSP on a personal computer. The results are reported.

Keywords

Big Data, Combinatorial Algebra, Groupoids, Machine Learning, Scaling, Traveling Salesman

1. Introduction

This paper is part and extension of the new causal theory, which was previously introduced (see for example [1]

and references thereof). The causal theory is a theory of Physics based on the fundamental principle of causality, that effects follow their causes. The working hypothesis in the theory is that information is the fundamental entity, instead of the more traditional space, time, and matter, which are considered as perceptions and therefore explainable by Physics and subject to the laws of Physics. Consequently, the theory also proposes that all information in the physical world is causal and should be encoded as a collection *w* of ordered cause—effect pairs over a set *S* of symbols, and represented by the causal set $C=(S,w)$, rather than the more traditional encoding as a string of symbols. Actually, information is always encoded as a causal set, except for the addition of a function *M* discussed below. These ideas are further expanded in Section 2.

The main goal of the causal theory is to derive structure from symmetries in the causal set. Every causal set has a permutation groupoid. The groupoid induces a block system *B* on *S*, consisting of a set S' of invariant blocks, and the partial order *w* induces a partial order w' on S'. Then $C'=(S',w')$ is a causal set, smaller than *C*, and a functor *F* can be defined as $F:C\to C'$. The same arguments can be applied to *C'*, and recursively repeated, until a trivial block system is encountered. The result is a hierarchy of invariant blocks, with progressively fewer and fewer blocks at each level, a mathematical fractal with properties such as scalability and compression of the information, described here as natural, mathematical properties of information itself. In the Traveling Salesman problem, this leads to an exponential decrease in tour complexity that may compensate for the NP combinatorial complexity of the problem. This is the algebraic core of the causal theory, that recursion, partition, scalability, and compression are already fundamental properties of causal information, and that exponential compression can compensate for combinatorial complexity.

The original motivation for the development of this theory was the author's interest in the binding problem, and more specifically the automation of the refactoring of computer software. A good introduction to the binding problem can be found in Sections 1 - 3 of [2]. Refactoring [3] is a behavior-preserving transformation of computer code, frequently code that is being developed, and makes the code better organized, more hierarchical, and more "understandable" for humans, hence easier to develop. In Big Data, the problem is that only humans can refactor, computers can't.

It turns out that refactoring is a problem of discrete optimization. The functional to be optimized is the intrinsic metric of causal sets. The functional is positive definite, homogeneous, and linear, and satisfies the property of locality. Because of locality, the process of optimization is fully parallelizable. It is a sum of positive terms independent of each other (some restrictions apply), and each term can be optimized by a separate processor, hence solving the problem in (nearly) constant time. The properties of recursion, scalability, compression, locality and parallelism are today the most coveted and sought-after in many disciplines such as Computer Science, Big Data, Supercomputing, Artificial Intelligence, and others placed at the very forefront of technology. But Turing computers lack these properties, because function *M* separates *S* from *w*, leaving *S* in the computer memory and *w* in the hands of human developers and inaccessible to the processor. Then, as the volume of data grows and exceeds the capacity of the human mind to comprehend them, development slows and may soon come to a standstill. These ideas are further expanded in Section 2.

Self-programming, defined as the automatic conversion of causal information acquired by sensors into refactored, object-oriented, human readable code, is also a problem of discrete optimization, and falls in the same category as refactoring [4].

The deep connection between the causal theory and the collection of natural, observable, and measurable phenomena we know as emergence, self-organization, adaptation, and intelligence cannot be overlooked [1]. It turns out that these phenomena are also problems of discrete optimization and the causal theory is needed to find explanations. In 2011, this author proposed a simple model where the symbols in *S* represent neurons and the (cause, effect) pairs in *w* represent dendrites connecting the neurons (see Section 4 in [4]). Based on this model, and on the working assumption that functor *F* describes brain function, he predicted that dendritic trees would have to be optimally short, and that they were optimally short for the purpose of saving biological resources and not that of creating intelligence. Neuroscientific insight at that time was that dendritic trees obeyed a 4/3 power law, which is not optimal. However, in 2012, a team of neuroscientists working independently found the 4/3 power law to be incorrect, and proposed instead a 2/3 power law that is optimally short, based on extensive experimental evidence from humans and other species [5]. This confirmed the prediction. Based on the new evidence, we proposed a "black-box" experimental program where a human and a causal machine were presented with the same set of observed or measured causal data, and their respective solutions were compared [6] [7]. A few black-box experiments were performed, such as: derivation of Newton second law, derivation of Euler Equ-

ations for a rotating body, refactoring and object-oriented analysis for a piece of "spaghetti code" in *C*, and the present application to NP-hard problems. In all cases, results confirmed the theory.

This is, no doubt, a vast panorama. It has to be addressed one piece at a time. In the present paper, the theory is discussed first, and then focus is sharply shifted to the Traveling Salesman problem (TSP) only. However, even with such a narrow focus, the human connection is still present. It turns out that humans are at least as good as computers to solve some TSP problems, and in some cases may even outperform algorithms by an order of magnitude, in spite of the fact that hardware runs at least 1,000,000 times faster than wetware. The prospect is thus raised that humans may be using better algorithms than those used today in computers [8]. Some animals such as ravens, bees, and ants solve TSP problems with great efficiency when deciding their foraging routes. Ants use information stored in their pheromone trails [9], but quantitative data are still scarce. Chimpanzees can outplay humans at strategic games [10]. And all of this is a problem for Big Data.

It is not possible to reach the theoretical limit for performance using sequential algorithms. Supercomputers would not help, either. The limit stems from the properties of locality and parallelism that causal information naturally possesses, but that supercomputers do not utilize. The functional to be optimized is a sum of positives, each one of which must be separately minimized, as discussed in more detail in Section 3.4 and Equation (8) below. However, and in order to provide a proof of concept, a suite of sequential algorithms was developed. The algorithms were designed for research, not production. The implementation emphasizes compliance with the theory, maximum flexibility, and no heuristics or adjustable parameters. They are applied here to the solution of TSP problems, but they can be easily extended to the general case of causal information, and not just for TSP. The algorithms run in the debugger, and contain numerous internal tests for early bug detection, needed to support constant changes as it always happens with research work. Benchmarks were not taken because performance is not a consideration.

Unlike other approaches such as quantum computation, the theoretical limit is easy to achieve in the causal theory. All it requires is a multicore processor, such as a GPU or a neuromorphic processor [11]. A concept for the design was published, but no action has been taken yet in that direction (see Section 4 in [4]). Section 7 below expands some more on this subject. The theoretical limit should be close to $O\left(N^0\right)$, which means constant execution time independent of the problem size, but this has not been demonstrated yet. It is interesting to note that humans can interpret an image in about 0.5 sec independent of size.

This paper presents the symmetric TSP problem as an application of the causal theory. TSP is one example of the *P* = *NP* problem, the famous problem of theoretical computer science asking whether for any problem for which we can verify a solution in polynomial runtime we can also find a solution in polynomial runtime. It has been selected as one of the Clay Millennium problems, and is considered by many as the most important unsolved problem in Mathematics. The expectation is that, by systematically partitioning *S* into a progressively smaller number of more and more comprehensive blocks, it should be possible to compensate the NP-hardness of TSP and solve it deterministically in low-order polynomial time and without the use of heuristics. Presented here is a case study of gr120, a symmetric TSP problem with 120 cities in Germany, with integer distances, solved in polynomial time by causal techniques.

The paper begins with an overview of the causal theory in Section 2. Matters such as the formulation of TSP as a causal set, block systems, permutation groupoids, the functor *F*, macrostates and their properties, and the grounding of the causal theory as a fundamental theory of Physics are discussed there.

2. The Causal Theory

The scope of this paper is limited to the solution of TSP problems. However, the causal theory needs to be introduced because it is what makes the solution possible. The causal theory proposes information as a fundamental entity with properties of its own. It also proposes that information should be encoded as a collection *w* of (cause, effect) pairs over a set *S* of symbols, instead of the more traditional encoding as a string of symbols. Such an encoding is known as a causal set $C=\left(S,w\right)$, and information encoded this way is called causal information, or simply information.

Actually information is always encoded as causal information, except for the addition of a function $M:S\to V$, where *V* is the set of symbols representing variables or structures that have a meaning for the analysis at hand, such as the state variables of some system of interest. For example, in a Turing machine, *S* is the tape, *w* is the transition function or program, and *M* is the tape layout. In Shannon's theory of information, *S* is the stream of

bits, *w* is the decoder, for example the controller in a TV set or cell phone, and *M* is defined as the map from *S* to the total order of the stream. The same triality happened even in Physics, where *S* consists of events, or interactions, *w* consists of experimentally observed (cause, effect) pairs, and spacetime and matter define function *M* by providing a layout structure where the events can be mapped. In all three cases, dealing with *w* is left for humans to do.

Function *M* has obvious advantages in engineering. It is what makes computation and communication possible. But *M* also has a serious disadvantage. For the purposes of the causal theory, function *M* is not needed. The theory treats information as an entity, and focuses on its properties, specifically on the properties of the mathematical object *C* that represents information and irrespective of any map to any particular system. This gives the causal theory a degree of autonomy and generality that the traditional theories of information and computation lack. And this lack is the serious disadvantage of function *M*.

The limitations of *M* are hardly new. A solid awareness of these limitations exists today, and a huge effort is being conducted in disciplines such as Artificial Intelligence, Computational Intelligence, Supercomputation, Internet of Things, Big Data, and others in an attempt to overcome the limitations by eliminating *M* altogether. In Big Data, the volume of data has grown beyond the limitations of the human mind to comprehend them [6] [12] [13]. Approaches such as Deep Learning [14] [15] and Neuromorphic Integrated Circuits [11] propose different types of non-Turing machines, with results that are promising but so far insufficient, and with limitations such as the usual lack of scalability or the emergence of non-physical features that are very difficult to overcome. What is lacking, is a theory fundamental enough and general enough with support for unified information. That theory is the causal theory.

The abstraction from *M* in the causal theory has another, much more fundamental effect: it makes the theory independent from space, time, and matter (substrate), and as such more fundamental than other theories of Physics and a causal predecessor to them. In the causal theory, information is the fundamental entity, and space, time, and matter are perceptions.

This section presents an introduction to the causal theory. Additional details and references are available in other publications [1] [16].

2.1. The Origin of Causal Sets

Where do causal sets come from? This question is addressed in Section 6 of [1]. A summary follows. In brief, causal sets are information, and they come from nearly everywhere. As anticipated in the beginning of the Introduction, and further discussed in Section 2, information has always been encoded as a causal set and presented in a variety of notations. Any algorithm is a causal set. Any computer program is a causal set. Both are just human-readable notations for causal sets, and transformations are possible both ways. The output from any sensor or sense is described by a causal set, and any signal sent to an actuator or muscle is described as a causal set. Causal sets describe interactions in nature: every interaction has causes and effects. Sometimes we don't know the causes, or the effects, and this is why we have science, research, forensics, and why a child asks "why" all the time and why we argue and observe all the time. Causal sets are vastly present in every moment of our everyday lives.

Sometimes we know the causes and want to find their effects. That's prediction. Sometimes we know the effects, and want to find their causes. That's research, investigation, measurement, observation. Sometimes we know causes and effects at a coarse scale and need more detail. Then we use instruments, telescopes, MRI and fMRI machines, medical tests, surveillance, to find the details we need. We are always after causal sets. Causality is the root of prediction, and our ability to predict empowers us. The Traveling Salesman problem is easily expressible as a causal set, see Section 3.1 below.

2.2. Symmetry, Groups, Groupoids, and Causal Sets

When a geometric object such as a chair has automorphisms, we say it has a symmetry. When a mathematical object such as a set has automorphisms, we also say it has a symmetry. In both cases, the automorphisms map "parts" of the object to similar parts, and we call these parts structure. Groups and groupoids are tools used to study the symmetries and the structures.

Let *S* be a finite set with *N* elements. If *S* is represented as a sequence (string), then the automorphisms of *S* are the permutations $S \to S$. The automorphisms of a mathematical object depend on the representation. Let *T*

be the set of all the $N!$ permutations of *S*. Then the mathematical object $G=(S,T)$ is a permutation group, known as the symmetric group of *S*. Let also $T^* \subset T$ be a proper subset of *T* that contains some permutations and their inverses. Then the object $G^*=(S,T^*)$ is a permutation groupoid. Groupoids are more general than groups. In addition, causal groupoids, defined below, have the property of recursion that groups do not have. For all our purposes in this paper, we will use only finite sets and causal groupoids.

To represent the permutations, consider two-line notation. Arbitrarily select some reference sequence for the elements of *S* from T^*, say t_r. Then any permutation can be represented as (t_r/t), and T^* will appear as a table of ordered sequences to which we can refer either as sequences, permutations of *S*, automorphisms of *S*, or a representation of the action of the groupoid on *S*. Sometimes we will refer to these sequences as tours, or trajectories, depending on context. In fact, in the Traveling Salesman Problem, table T^* is the set of all tours that start and end at the warehouse city. Note that the identity element (t_r/t_r) will automatically be included in T^*. Let now

$$C=(S,w) \tag{1}$$

be a causal set. A causal set is a partially ordered set where *S* is a finite set and partial order *w* is irreflexive $(\neg(a \prec a))$, acyclic (if $a \prec b$ then $\neg(b \prec a)$), and transitive (if $a \prec b$ and $b \prec c$, then $a \prec c$), where $a, b, c \in S$ and $\prec$ means "precedes". A partial order can also be seen as a rule for non-commutativity, which provides an important connection between causality and non-commutative geometries [17].

With *C* given, let T^* be the table of all legal sequences of *S*. A sequence is legal when it does not violate partial order *w*. As before, table T^* also represents all legal permutations of *S* in the form (t_r/t), $t, t_r \in T^*$, where t_r is a sequence arbitrarily selected from T^*. Then, $G=(S,T^*)$ is the permutation groupoid corresponding to causal set *C*. A permutation groupoid derived from a causal set is known as a causal groupoid.

2.3. Block Systems, Structure, Invariance, and Information Compression

Let now *B* be a partition of *S* that is compact over the reference sequence t_r. A partition is a collection of disjoint subsets of *S* that together cover *S* completely, say $B=\{b_1,\cdots,b_{N'}\}$. For example, in TSP, any subset of 2 or more cities that appear together in t_r are a compact subset.

In general, the property of compactness of *B* is not preserved for other sequences in T^*. However, if a partition of *S* can be found such that the action of T^* does preserve compactness, then the following important properties follow:

- a symmetry has been found and structure has arisen; the "parts" that map to similar parts are the compact subsets of the partition;
- *B* is called a block system, the b_i's are called blocks, and block system *B* is said to be invariant under the action of *G*;
- compression of the information takes place, because the block system represents the same information as the causal set, but with a coarser granularity, and the blocks contain references to the finer granularity; and
- it becomes possible to separate the permutations in T^* into a set of permutations $B \to B$ and a collection of N' subsets of permutations $b_i \to b_i$, one for each of the N' blocks in set *B*.

The block system of T^* is a property of T^* alone, hence of the partial order *w* in the causal set *C* in Equation (1). Note that blocks are sets, hence elements of a block can appear in any order inside the block and the block is still compact. Note also that the properties above are always satisfied by the two trivial partitions, the one with *N* subsets of size 1, and the one with one subset of size *N*, except that the symmetry, structure, and partition are all trivial. It is only in the case of a non-trivial partition, one where $1<N'<N$ that nontrivial symmetry, structure and invariance arise, compression takes place, and function T^* is separable. The process is analogous to that of the separation of variables used in differential equations, except that here the separation happens at a fundamental and universal level, that of causality, and not just as an algebraic method used to solve differential equations.

Algorithms for finding block systems use the property of invariance. They start with a seed of 2 elements of t_r and try to expand the seed by testing invariance for the other sequences in T^*. See for example pages 40 - 41 in [18].

Causal sets have many more natural properties that are important and actively sought after in a variety of disciplines, including Big Data, supercomputation, and discrete optimization, and can be quantitatively calculated

directly from the causal set by Mathematics. A survey of the properties would be difficult to prepare and somewhat out of topic in this paper, but a few more have been discussed in [4] [7]. They include meaning, mathematical logic, semantics (the expression "physical meaning" has been in Physics for a long time, and this is the source of semantics), self-programming, binding, association, and associative memories. A few more are discussed in the next Section, in particular the supreme property of causal sets: their property of recursion.

2.4. Functor, Recursion, Hierarchies, Fractals, Granularity, Logic, and Semantics

As just explained in Section 2.3, when causal set $C=(S,w)$ of Equation (1) has a non-trivial block system consisting of a set of blocks $S'=\{b_1,\cdots,b_{N'}\}$, among which a partial order w' has been induced by partial order w in C, then

$$C'=(S',w') \tag{2}$$

is a causal set smaller than the original C. As a causal set, C' has its own block system with an induced partial order, which, if non-trivial, again forms another, even smaller causal set. This process is one of recursion. Recursion is a property that causal sets naturally possess. If recursion keeps generating non-trivial partitions, then a hierarchy of progressively smaller and smaller invariant block systems will result. If plotted on paper with the elements of each causal set listed horizontally in small boxes, this hierarchy has a pyramidal shape. A small example of such a plot is in figure 3 of [16]. In a more general case, the original causal set C can be very large and very complex. It may in general be disconnected and contain many connected components. Each connected component may give rise to a network of hierarchies, which may generally overlap at the bottom but become separated and individualized at the top, a jungle. Jungles are strongly reminiscent of the networks of hierarchies proposed by Fuster [19] [20] in the brain and mind. In the brain, the invariant blocks correspond to neural cliques, which are known to form hierarchies. In the mind, Fuster calls them cognits, which are also known to form hierarchies. The tops of the hierarchies are invariant and distinct, hence endowed with a physical meaning and capable of separating and identifying individual objects and images, even in a constantly changing environment.

A functor F in the category of causal sets can be defined based on the construction of C' from C discussed in the previous Section:

$$F:C\to C' \tag{3}$$

Functor F is an endofunctor that morphs C into C' and has the property of being conditionally associative under the operation of set union, provided the union of causal sets is properly defined. Let $C_1=(S_1,w_1)$ and $C_2=(S_2,w_2)$ be two causal sets. Then their union is defined as follows:

$$C_1\cup C_2=(S_1\cup S_2,w_1\cup w_2) \tag{4}$$

The condition for F to be associative under union is that C_1 and C_2 be disjoint:

$$F(C_1\cup C_2)=F(C_1)\cup F(C_2),\quad C_1\cap C_2=\varnothing \tag{5}$$

In the context of the jungle of hierarchies discussed in the preceding paragraph, it follows that the causal sets in the overlapping parts of the hierarchies are not disjoint and hence not associative under F, whereas the causal sets in the distinct tops are disjoint and hence associative. The property of associativity supports the task of comprehending a large body of data, *i.e.* finding meaningful features and grasping their significance, which still stands as THE most critical unsolved problem in machine learning and Big Data. The link between functor associativity and comprehension is too important to discuss here and will be further discussed in a future paper.

The hierarchy of blocks has, from the start, several mathematical properties that are crucial for the causal theory and in discrete optimization and computation in general. Each level of the hierarchy represents the same information, but at a coarser granularity than the preceding level, and includes references to the finer granularity of the preceding level, hence compression of the information takes place. The hierarchy is a mathematical fractal, because every level of the hierarchy has the same mathematical properties: its blocks are made of blocks of the lower level, and serve as components for blocks of the next higher level. And the blocks in each level have a partial order inherited from the partial order in the lower level, and which is inherited by the blocks of the next higher level. The levels of the hierarchy become progressively smaller at higher levels. At the top of the hierarchy there is a single block, or a set of trivial blocks that cannot be compressed any further. As examples, con-

sider Maxwell's equations, which are a set of 4 equations, and the equation of General Relativity, which is the following single equation: $G_{\mu\nu} = 8\pi\kappa T_{\mu\nu}$.

All of which means that information compression takes place. It has been conjectured, but not yet proved, that the compression reaches the Kolmogorov limit. Being a fractal, the hierarchy is scalable and supports power laws. Fractals and power laws are well known to exist nearly everywhere in nature. And here is the core property of causal sets:

Causal sets are naturally recursive, and give rise to scale-free hierarchies, mathematical fractals, power laws, and information compression.

In a more general case, a large causal set can generate not just one hierarchy but a large collection of inter-connected hierarchies, an entire *network* of invariant blocks. An example of one such network in nature is the network of cognits that exists in the brain, where blocks are referred to as "cognits" and proposed to exist not just in the brain but also in the mind [19] [20]. In Physics, the notion that invariants are observable and have a physical meaning has been present for a very long time. In image recognition, "features" and hierarchies of features that our minds make to recognize objects from vision are networks of hierarchies of invariant blocks. In both cases, they originate from the recursive property of causal information.

Logic, meaning, and semantics are clearly associated with the hierarchical structures of invariants that causal recursion generates. The hierarchies are "new facts," derived from previously existing facts, the previously existing facts being the causal pairs in w. This perfectly fits the definition of mathematical logic, and this author has called it Causal Mathematical Logic (CML) [6]. All of which are natural properties of information itself, and there is no need for engineering methods for compression and various forms of logic or semantics. They are already there.

2.5. Physics, Macrostates, Action, Population, and Entropy

This section introduces the important notion of macrostate, and the action, population and entropy of macrostates. The causal theory is a theory of Physics. Some terms and concepts used in the theory that come from Physics and Mathematics are briefly introduced here in that context. The causal set is a mathematical model of a physical system. The symbols in S correspond to variables in the phase space of the system, and partial order w is the dynamic law. The dynamics of the system is a search of the directed graph of the causal set. A state is the collection of variables already visited by the search at some point. A transition corresponds to the search following an edge of the graph from a visited vertex and visiting an unvisited one. A trajectory is a path of causal edges in the graph that starts from some initial state and ends at some final state, and can potentially be followed by the search. A macrostate is the set of all paths that start from a given initial state and end at a given final state. The population of a macrostate is the number of trajectories in that macrostate. In TSP, a state is the set of visited cities, a path in the graph is a tour, and a macrostate is the set of all tours of the same length. In the causal theory, there is no time. Time is causality itself.

The theory applies to information because information is about physical systems. It proposes that information is fundamental, and that spacetime and mass are perceptions. Information has observable properties of its own, such as symmetry and structure, and entropy, all of which follow directly from the fundamental principle by way of Mathematics. Information also has energy [21]. The macrostates have populations, with thermodynamic, hence time- and space-independent properties such as entropy, uncertainty, and action. Action is an important quantity in Physics, with the dimensions of energy times time. And the populations are meaningful and can better our understanding of problems of Physics such as TSP. In TSP, action is proportional to the length of a tour.

Because the theory is scale free, it applies to physical systems at all scales and can represent them at any desired granularity. It can also "learn" more detail and smoothly transition to a finer granularity when more detail becomes available from observation or experiment. When it does, then the hierarchies of invariants change and new behaviors emerge.

The importance and physical meaning of macrostates stem from the following property: the causal theory predicts a general relation between the action and the population of a macrostate (see for example figure 1 in [1]). It looks a lot like a Gaussian distribution curve, except that it is deterministic and not probabilistic. If entropy is defined in the usual way, as proportional to the logarithm of the population, then the relation becomes one between the action and the entropy of the macrostate. This function was first observed in a study of small

causal sets, and later confirmed by comparison with human electro-encephalograms.

The action vs. entropy function is again confirmed in this paper from observed TSP results. It keeps re-appearing in completely unrelated circumstances. Macrostates containing long tours have very large populations, while macrostates with short tours have very small populations. Randomly generated tours are almost always very long, around 50,000 km for gr120, simply because there are so many of long tours than there are short tours. During optimization, as tour lengths approach the optimum, the populations per macrostate dwindle to 0 or 1, or 2 in some cases. Population also drops dramatically towards the other end, the end with most action and very long tours, where the populations and entropy again become small. The "knee" on the left of the population curve separates the large populations regime from the small populations regime, and may play a useful role for monitoring the approach to the optimum. Here are two TSP examples, where the notation $n(L1-L2)$ means a set of *n* tours with lengths in the range *L*1 to *L*2:

- 1265 (7175 - 7359). Starts from 1 tour per macrostate, begins to grow near 7254, peaks to 25 - 29 in range 7279 - 7289, then drops to 1 - 3 to the right of 7350. Roughly places (left knee - peak - right knee) as (7254 - 7285 - 7350) for this set.
- 2908 (7049 - 7223). Starts from 1 tour per macrostate, begins to grow near 7063, peaks at 26 - 33 in range 7088 - 7156, then drops to 1 - 4 near 7216. Roughly places (left knee - peak - right knee) as (7063 - 7110 - 7216) for this set.

It would seem that the causal Gauss-like population curve is quite general, suggesting that the Gauss distribution itself may actually follow from the causal theory. The Gauss distribution is known to be ubiquitous in nature and the natural sciences. But the causal theory is even more ubiquitous. These properties have not been used for the present studies, but they may find an application to predict which search techniques are better in each regime.

The notions discussed in this section may also one day help to answer the Kolmogorov limit for the compression of information, causal information in this case. I have not studied this issue, but a recursive implementation of a "decompressor" for a hierarchy of blocks should be quite simple, perhaps provably simple, and it does not even have to involve the causal set or groupoid. For now, I can only conjecture about it.

3. Causal Optimization

The direct application of groupoid-theoretical techniques to discrete optimization problems, within the context of the causal theory, is discussed in this section.

3.1. The Traveling Salesman Problem as a Causal Set

In this section, an expression for the causal set for the symmetric closed-tour TSP is introduced. In TSP, the elements of *S* are the cities, and the cause-effect pairs in *w* correspond to the problem statement, that the warehouse city—the city where the salesman's journey begins—is first in the tour, and the destination city, in this case the same warehouse city, is last in the tour. One peculiarity of causal modeling is that causality replaces time. There is no time or space in a causal set, only causality. Time, if needed, is treated as a perception. The immediate consequence is that the destination city, which is the "same" warehouse city that the salesman departs from, is not really "the same city" but only "the same city at a later time" —a different city in causal lingo—and needs to be explicitly included as another city in the causal model, which is the causal successor of all other cities including the initial warehouse city. Hence, for a TSP with *N* cities:

$$\begin{aligned} &S = \{s_1, s_2, s_3, \cdots, s_N, s_{N+1}\}, \\ &w = \{s_1 \prec s_2, s_1 \prec s_3, \cdots, s_1 \prec s_N, s_1 \prec s_{N+1}, s_2 \prec s_{N+1}, s_3 \prec s_{N+1}, \cdots, s_N \prec s_{N+1}\}, \\ &C = (S, w), \end{aligned} \tag{6}$$

where *S* is the set of symbols representing cities, set *w* defines a *partial order* on *S*, s_1 represents the departure city, s_{N+1} represents the destination city, the tour is now open, and the symbol $\prec$ means "precedes". Causal set *C* represents all the causal information related to the problem at hand. In addition, metric information must also be given, to be associated as weight to the causal relations in *w*. In this case, the information is the (symmetric) matrix of distances between cities, including the destination city counted with the same distances as the departure city. Even though the original TSP defines a closed tour, its causal version is open—there is no path

from S_{N+1} back to S_1. Causal sets are acyclic.

The TSP problem is a particular case of a much more general class of problems that can be solved with the group-theoretical method known as Causal Mathematical Logic (CML) [6]. The motivation of the present work is to address the *P* = *NP* problem, but all the methods and ideas discussed are of a general nature and can easily be extended to the CML case.

3.2. Optimization and Symmetry Breaking Generate Structure

A set, such as the set *S* of cities in the causal set of Equation (6) by itself, has symmetry. The symmetry is represented by the collection of allowed tours, which is simply "all tours" as far as set *S* is concerned. If *S* has *N* cities, then *S* has exactly *N*! tours, and each one of them represents exactly the same set *S* because the elements in a set have no order. By definition, a mathematical object such as a causal set has symmetry when it can be represented in more than one way. This set of *N*! permutations is called the *symmetric group* of *S*, and also happens to be the groupoid of *S*. However, there is no non-trivial structure associated with this symmetry: the groupoid only induces a trivial block system onto *S*, and no non-trivial hierarchy of blocks is generated. The symmetry is just too strong for it to be relevant.

However, when partial order *w* is added and causal set $C=(S,w)$ is considered, then some tours are no longer allowed and the symmetry changes. A tour is legal only if $s_1 \in S$ is the first city and $s_{N+1} \in S$ is the last. Tours that do not satisfy this condition are not allowed. We say that the partial order places a constraint on *S* and breaks its symmetry. This new reduced symmetry may well, and usually does have a meaningful structure.

The process of optimization itself also breaks the symmetry of *S* even further because it progressively eliminates longer tours. As optimization proceeds, shorter and shorter tours are found and longer tours are discarded. Optimization works as an additional constraint placed on *S*, and it too breaks the symmetry and generates additional structure. The more we optimize, the more non-trivial structure is generated, and the structures are at their maximum when approaching the optimum.

3.3. Locality Is the Cornerstone of Scalability and Parallelism

Locality, the property that an object or structure is influenced only by its immediate environment, is the cornerstone of scalability and parallelism. Consider the minimization of a function *F* defined as the sum of *N* non negatively-defined functions, where *N* can be very large:

$$F = f_1 + f_2 + \cdots + f_N, \quad f_i \geq 0, \quad i = 1, N \tag{7}$$

where the functions f_i are non-negative and independent of each other. Then, the only way to minimize *F* is to minimize each f_i locally by itself. There follow three conclusions. First, the processes that minimize the f_i's can proceed without any need for global information such as the global effects of the minimizations of *F* or the purpose, goal, or stability of the result. No processor needs to know that there is such a thing as "the rest of the tour". This satisfies the property of locality. Second, the processes can proceed simultaneously and without synchronization, on separate processors, and the total execution time will not exceed the execution time of the longest process. This satisfies the property of parallelism. And third, the first and the second conclusions are independent of *N*, which implies the theoretical limit of $O(N^0)$ for the groupoid-theoretical optimization. Of course, these conclusions require that the number of processors be *N* or more. The initial global problem of size *N* has been effectively divided into *N* local independent problems of size 1, each of which can be solved by itself on a separate processor.

For the hardware architecture of Section 7, the functions f_i correspond to the dendrites, not the neurons. The neurons need to compete for the best locations in the tour, making the f_i's not completely independent. For the proposed linear architecture, this constraint is in general strong, and the device will only approximate the theoretical limit. However, in TSP in particular, the constraint is much weaker because any city other than city 1 or city $N+1$ can be in any location in the tour. There is another important option that has not yet been exploited: develop a three-dimensional architecture, where the number of choices for a neuron to go will be much larger and the constraint much weaker.

The functional of TSP, the total length of the tour, is positive-definite and satisfies the conditions for *F* provided the transformations to which *F* is subjected are local. Transformations such as swapping cities or shifting or flipping blocks, are local, and the synchronization among them—blocking neighboring cities from participat-

ing in more than one transformation at a time—is local too. The rest of the tour remains unaffected and capable of engaging in numerous simultaneous and asynchronous transformations. Under such conditions, it is technically possible to assign one core to each f_i and require it to minimize this particular f_i while blocking neighboring participating cores from initiating their own minimizations. This way, the problem of minimizing *F* can be solved in parallel, quasi-independently from the value of *N*, with only local synchronization and without any global information flowing along the network. If, in addition, all f_i's are similar, as they are proposed to be in TSP, then the network will consist of *N* identical processors.

The total execution time for such a network will depend on the average total *shift* of a city or block subject to repeated swapping. If the shift depends on *N*, for example if it grows linearly with *N*, then the execution time will be of $O(N)$. If, on the other hand, a limit on the maximum average city shift can be found—as is the case in many CML problems, but less frequently in TSP problems—then the total execution time will be constant and independent of *N*, perhaps a few microseconds with good hardware if judging by the simplicity of the operations involved in each city swap. Humans can interpret images in about 0.5 sec irrespective of the degree of complexity of the image. However, tasks such as scientific reasoning or software refactoring may have a higher complexity.

The average city shift tends to be shorter when the tour is good and some blocks have formed, because cities tend to stay within their blocks, which provide a natural upper limit to the shifts. Hence it may help to start from a good seed, perhaps a good Dijkstra tour. The blocks themselves can shift, but as the optimization progresses a hierarchy is built and blocks become parts of bigger blocks, and so on, which become less and less likely to shift. Actually, the complexity of group-theoretical swapping algorithms *improves* closer to the optimum, rather than deteriorating.

One more consideration is needed: pre-calculated inter-city distances are a global parameter and would violate the requirements for locality. For an implementation of a small problem, it may be possible to store one entire row of the matrix of distances in each core's memory, but probably not for larger problems. However, larger problems remain strictly local for an Euclidean TSP, where relative cartesian coordinates for each city are stored locally in that city's core and used to calculate distances, or some other way to calculate them. It is known that humans and some animals can estimate distances from visual perception and use them to solve TSP problems more efficiently by using relative positions.

3.4. Groupoid-Theoretical Optimization

Block system formation in the course of an optimization process are essential for the process to succeed. Optimization can be viewed as a process that starts from an initial seed tour and repeatedly transforms it in various ways so that it becomes progressively shorter and shorter. The transformations are chosen in such a way that they favor the formation of shorter tours. However, due to irregularities in the inter-city distances and the complexity of the combinations it generates, this process is not uniform over the entire tour. It tends to create clusters of strongly bound cities, interspersed among weakly bound or not yet bound cities. City and block swapping causes the cities to compete with each other until they find stronger bonds and stabilize and form first clusters. The clusters keep competing among themselves and with the remaining cities, until they further stabilize and form stable blocks that remain invariant under the transformations, just because their bonds are strong enough. Similarly, as the process continues, blocks form blocks of blocks, and an entire hierarchy arises.

The causal theory proposes a functional to be minimized, defined by the basic mathematical metric for sets. In TSP, the functional to be optimized is the length of the tour, which is the sum of the distances between pairs of adjacent cities in the tour. For a tour $t=(c_i, i=1, N+1)$, where the parenthesis indicate total order, the functional is:

$$L=\sum_{i=1}^{N} d(c_i, c_{i+1}) \tag{8}$$

where $d(c_i, c_{i+1})$ is the distance between cities c_i and c_{i+1} in tour *t*. This functional is positive-definite. It also satisfies the property of locality, as discussed in Section 3.3, hence it automatically gives rise to the properties of scalability and full parallelism for the system. Two adjacent cities or blocks of cities can be swapped locally if they know their coordinates, and many such swaps can be taking place simultaneously in different places in the tour. In addition, since the groupoid-theoretical blocks are structures that "remain", or are invariant under

the transformations, they are also observable and have a physical meaning for us, that we assign and can use for prediction and survival.

3.5. Machine Learning

Machine learning is a vast field of research, and must be mentioned here because it is a very important and direct application of the causal theory. The intent in this section is not to review the field but only to place that research in the context of the causal theory. Machine learning has been evolving slowly but steadily in the course of decades, and this evolution has brought it from simple approaches that rely heavily on hand engineering and computer simulation to more sophisticated approaches that come closer and closer to the causal theory.

At the forefront of those developments is Deep Learning (see for example this video [15]). Based on current neuroscientific knowledge that most perception in the brain is due to one single learning algorithm, Andrew Ng proposes to stop investing so much effort in hand-engineering the feature representations and instead search for a "crude approximation" to that algorithm. However, Ng's very next step is to engineer a feature representation for the algorithm. The video also shows a plot of performance vs. resources, which suggests poor scaling, as the curves shown appear to be approaching a plateau. Scaling is critically important for the Internet Of Things, particularly for the case of untagged data, where the data sets will be very large.

Ng's working hypothesis about the single learning algorithm is correct. A decomposition in terms of a heuristic orthogonal base is not. The base is a feature and pre-specifying it would be feature engineering and would result in scaling problems an unphysical results. What is needed is a fractal base, one that is orthogonal across fractal levels. The causal pairs and their block system hierarchies play this role. Orthogonality alone isn't enough.

The single learning algorithm is already known, and it is Causal Mathematical Logic (CML) [1]. There is no need to find an approximation, much less a crude one. CML makes its own features and represents them as a fractal structure of invariants. Scaling problems never arise because fractals are scale-free, and there are no heuristics, only rigorous Mathematics. The features are determined by the available information and will change if the information changes. Machine learning in the causal theory is the causal set of Equation (1) that quickly changes as a result of a constant stream of untagged causal information coming from sensors in a robot or from senses in an animal. The new information brings new elements for set S and new pairs for set w. These new elements are simply appended to the sets, while at the same time the optimization process of Equation (7) works in the background and keeps the structures updated. That's all. In the causal theory, learning with a supervisor (tagged information) or without one (untagged information) make no difference. Tags are features, and CML makes them. The net result is a stream of information coming in and a fractal network of invariant features coming out. Many features remain invariant, new features appear, some change, others may become unstable and disappear completely. Features that stabilize can be stored and used for prediction, but they too are subject to change.

4. Mathematical Proof

To prove that $P = NP$ for TSP, one would have to prove that all cases of TSP, or more in general for CML, can be solved quickly. There is one case per causal set, and the number of causal sets of all sizes is countable infinite, so that a proof by induction seems to be a natural choice.

Chaos theory pioneer Edward N. Lorenz has defined chaos as a case where "the present determines the future, but the approximate present does not approximately determine the future". If a given causal set is considered as the past and the corresponding hierarchies as the future, then any process that changes the causal set, such as machine learning, will be chaotic. The resulting hierarchies are very sensitive to details in the causal set (butterfly effect). This effect was first observed in a study of small causal sets (about 12 elements or less) [16]. Minor changes such as the addition or removal of one single element or causal relation, or a simple substitution of a causal pair for another, may in some cases result in completely different structures. Unfortunately, a proof by induction with, say, a proved base case with N cities, would entail the addition of one city when trying to prove the case with $N + 1$ cities, and this would result in chaos.

A proof by exhaustion would work for problems of a given subset, for example all problems of a prescribed size or range of sizes, but it would fall short of proving the statement. A direct proof is highly unlikely to exist, because, as argued in [6], CML is logic. Other types of proof such as contradiction that imply logic would not

work for the same reason. A proof of existence is provided in this paper for the example case gr120, and no counter-examples are known, at least for now. This author believes that a proof of $P = NP$ in the mathematical sense is unlikely to exist, and that the best one can do is to be satisfied with computer experiments for particular cases. Which would be necessary anyway if one wanted to search for counter-examples.

On the other hand, the physical world does provide numerous examples of CML in action. The existence of fractals and power laws, the property of scaling in nature, the ability of humans to solve TSP, all the observed phenomena of emergence and self-organization, are examples of CML in action.

5. The Sequential Algorithms

This section is addressed to the person who is interested in programming details and practical hints. It reports explicit results obtained from computation for TSP gr120. Many theoretical details of practical importance are also included. All results were obtained by application of the tunneling algorithms described below, and all computations were performed in debug mode on a personal desktop computer using an Intel Pentium 2.70 GHz 2 cores processor with 4 GB of memory and a 64-bit operating system.

City tunneling (*ct*) is the workhorse for the computation. The code is very light. A tour with 120 cities can be fully surveyed for all 120 cities and the shortest tour found in about 0.042 sec. This time interval grows linearly with the number of cities, so it is about 350 μsec per city. The code is experimental and has not been optimized for production. It is designed for ease of change and experimentation. It has not yet reached full automation, and there still remain some operations that I do manually, such as copy the results from a *ct* run to a file that's used as input for the run that follows. There is no user interface.

I will now report numerical results obtained in the course of the study in a more or less logical order, not necessarily chronological, to make it easier for the reader to follow the argument. Here I report only summary tables, the tours themselves are found in the Supplementary Material [22]. A seed tour is needed to start the algorithms. The first seed can be a Dijkstra tour, or a randomly generated tour. Random tours for gr120 are usually very long, with a length in the order of 50,000. However, a tour of that length can be easily reduced by *ct* alone to a length close to 8000. Hence, the computational advantage of using shorter Dijkstra tours over much longer random tours is not very significant. In addition, the assortment of random tours is much larger. For the purpose of this work, I used only random tours as first seeds.

In this section, I introduce the suite of sequential algorithms based on the causal theory and specifically developed for the TSP problem. The algorithms were developed only for proof-of-concept and a guide for future developments, and do not represent the long-term goal, which is to develop a hardware prototype that can solve large problems, including TSP, hopefully in microseconds using a specially designed GPU-style chip. I call them *tunneling* algorithms because they are designed to tunnel through the walls of local minima basins. The purpose of tunneling is to evaluate a given seed tour in regard with its capacity to generate shorter tours, and to do so by using only local information.

Many depth-first algorithms create a collection of local minima different from the global minima that exist in the search space. The local minima are determined by the search algorithm, and not by the search space. To every local minimum, there corresponds a catchment basin, also determined by the algorithm, from where a simple descent search cannot break out. The algorithms are designed to "tunnel" through or over the walls of the basin and search for tours outside it that could allow the depth-first process to continue. The term is taken from Quantum Mechanics, where a charged particle such as an electron trapped in a potential well can free itself by tunneling through the walls of the well (thus making the existence of transistors possible). The combination of locality, efficiency, and the ability to tunnel make tunneling algorithms a very powerful tool.

These algorithms bear some resemblance with the classical simulated annealing approach, particularly for planning-type problems, but make their optimization decisions based on group theory and not on heuristics. A comparison with simulated annealing is in Section 5.3. The most basic tunneling algorithms are *city tunneling* and *block tunneling*, but there are several others. The algorithms start from a given seed tour and find only one tour, the shortest that algorithm can find for the given seed. City tunneling is discussed first.

5.1. City Tunneling

The most basic tunneling algorithm is *city tunneling*, or *ct*. It starts from any given tour, such as a random tour or a Dijkstra tour, or any other tour obtained in the course of optimization. Cities are selected from set *S* in a

random order. Each selected city is (virtually) removed from its current location in the tour, inserted back into the tour at the leftmost location, and then repeatedly swaped with the neighboring city to the right until the end of the tour. After each swap, the (signed) increment of tour length is locally calculated. After completion of the tunnel the city is re-inserted to the best location, which could be its original position, and left there. The next city in the random order is selected, and the process continues until all cities have been used. *ct* is very light code, and is the workhorse among the tunneling algorithms. Let:

$$\cdots \quad A \quad [B \quad C] \quad D \quad \cdots \tag{9}$$

be the part of a tour where a swap is about to happen between adjacent cities *B* and *C*. After the swap, this piece of tour becomes:

$$\cdots \quad A \quad [C \quad B] \quad D \quad \cdots \tag{10}$$

and the resulting (signed) increment of length Δ caused by the swap is:

$$\Delta = \left(d_{AC} + d_{CB} + d_{BD}\right) - \left(d_{AB} + d_{BC} + d_{CD}\right) \tag{11}$$

where d_{IJ} is the distance between cities *I* and *J* and the rest of the tour is not affected. This analysis is for symmetric TSP problems, hence $d_{CB} = d_{BC}$ and the two terms cancel:

$$\Delta = d_{AC} + d_{BD} - d_{AB} - d_{CD} \tag{12}$$

The calculation of Δ for each swap requires only 4 memory accesses and 3 additions, all of them locally performed near city *B* in this case. Since cities are selected for tunneling in a random order, it may be convenient to try a few more iterations. Additional iterations will select cities in a different random order and generate additional tours that were not tested in the first iteration. My practical experience, limited to the case study gr120, indicates that convergence seldom takes more than 3 iterations, and frequently only 1. However, careful attention must be paid to the meaning of the term "convergence": *ct* iterations converge only when no city, if tunneled to any location in the current *ct*-optimized tour, is capable of achieving any improvement at all. A tour that satisfies this condition is called a *city-stabilized* tour. Since there are $O(N)$ cities to tunnel, and $O(N)$ slots for each city to tunnel into, and assuming the number of iterations is small compared with *N*, the algorithmic complexity for obtaining a city-stabilized tour from an arbitrary given tour is $O(N^2)$.

As explained in Section 3, group-theoretical blocks of cities form when two or more cities bind together. The process of optimization forces the cities to compete for places in the tour and to bind together and create a block when the bond is sufficiently strong to resist further transformations imposed by the optimization. As optimization proceeds, more and more blocks are formed, and fewer and fewer free cities are left. A city-stabilized tour bears blocks, and may still have free cities, and any further improvement in tour length can be achieved only by tunneling blocks, not cities. The algorithm can be described in the following steps:

Step 1. Arbitrarily select a city *c* that is not the warehouse from the seed tour.

Step 2. Using successive transpositions of adjacent cities, and advancing in one direction, say left and right, move the city to each one of the slots in the tour. For each slot, calculate the increment of tour length using Equation (12).

Step 3. Leave the city in the slot where the increment is the most negative.

Step 4. Arbitrarily select another city and repeat until all cities are exhausted.

The (signed) increment Δ in tour length due to a transposition can be calculated locally by a simple, fixed number of operations by Equation (12). On a regular computer the increments are accumulated, but on a hardware implementation there is no need to keep track and the calculation remains local. The input to the algorithm is a seed tour, and the output is another tour, hopefully shorter than the seed. The condition where the output is the seed can arise only when each city is in its best position in the tour, so that any attempt at tunneling a city can only result in a non-negative increment. The condition implies that group-theoretical blocks have formed and are strongly bound, strongly enough that they can remain bound under the tunneling of any city.

5.2. Block Tunneling

Sets T and $T^* \subset T$ are defined in Section 2.2. The block system of T^* is likely to be trivial if T^* is too

large, but it is more likely to be non-trivial if T^* is small. For a T^* of size two, containing just 2 permutations, the block system is almost certainly non-trivial (a theorem is needed). Given a tour $t \in T^*$ that is non-optimal, then there may exist a t' that is shorter than *t* and groupoid $\{t,t'\}$ will have non-trivial blocks. But t' is unknown, and the problem is to find t' when *t* is given. *t* and t' share at least one non-trivial block, but this block is unknown. The problem of finding t' is thus reduced to the problem of finding a block in *t* and a suitable operation on the block, that could be tunnel, block inversion, block rotation, in-place block permutation, etc, such that the operation results in a shorter tour t'. Finding that block and that operation is, in many cases, a low-order polynomial problem. The worst-case scenario would be the case where the operation is a full-block permutation and the block is too large.

But when only the city-stabilized tour is available, there is no block and no transformation. The blocks are not visible, and the transformation is not defined. With only one tour, the algorithm in pages 40 - 41 in [18] for finding block systems cannot be applied. So we are going to consider all possible blocks in the given tour, and all possible transformations for that block, and select the block-transformation combination that yields the shortest tour. The algorithm is as follows:

1) Select a block size in the range 2 to $N/2$, and repeat what follows for each size.

2) Select a block of that size from the tour and repeat what follows for all such blocks.

3) Tunnel the block over the tour and leave it where the tour length is the shortest.

For each block, the shift of the block from its original position to its new position is the transformation, and the block is clearly invariant under this transformation because it exists in both *t* and t'. The number of block sizes for a TSP with *N* cities is $O(N)$, the number of blocks of each size is $O(N)$, and the number of slots into which the block can be tunneled is $O(N)$. Hence, the algorithmic complexity of the tunneling algorithm is $O(N^3)$.

There are some complications. One of them is that a block of cities is a set, not a sequence. The cities in the block can be permuted in any order, which may be computationally expensive. Another complication is that the action of shifting a block from one place in a tour to another upsets the city balance, resulting in a tour that may not be city-stabilized and could be improved by restoring the balance. To address the problem, several categories of permutations were defined and graded by cost, and a different algorithm was created for each category. This way, it becomes possible to adjust the power of the search at each step to the degree of difficulty of that step, and to do so automatically. This can be done by starting each step with a low-graded algorithm, and escalating only if necessary. The idea proved to be very effective in the case study, where several threads reached the optimum using only the least cost algorithms. The categories are:

1) *ct*—City tunneling. Scales as $O(N^2)$.

2) *bt*—Block tunneling as-is. Scales as $O(N^3)$. However, this figure is tempered by the fact that block tunneling is applied only to city-stabilized tours, which contain fewer than *N* blocks. Furthermore, as optimization progresses, the number of blocks being processed becomes small, and it becomes very small precisely at the point where it is most difficult to find shorter tours: near the optimum.

3) *bipi*—Block in-place inversion. A block is inverted in-place (flipped over) in a tour. The computational effort is tiny. Scales as $O(N^2)$, but again, the number of blocks is usually smaller than *N*. Block flipping has been used for spin models in Simulated Annealing. Scales as $O(N^3)$.

4) *bti*—Block tunneling with inversion. In each tunnel, the block is considered both as-is and inverted. Because the additional computational cost is so small, this category is regularly included with as-is.

5) *bct*—Block-city tunneling. Block tunneling with block as-is and block inversion followed with city tunneling to restore the city balance before a decision is made regarding which tours to keep and which to discard. Scales as $O(N^5)$.

6) *fbp*—In-place full block permutations.

7) *fbpt*—Full block permutations followed with block tunneling before a decision is made.

8) *fbpbct*—Full block permutations followed with block tunneling and city tunneling before a decision is made.

Other categories with various degrees of difficulty can be defined, for example block rotation, both in place or with tunneling, but have not been implemented. Also, the categories can be subdivided by block size, for example *bt*2 would be block tunneling as-is and inverted with blocks of size 2. Of course, with blocks of size 2, this is the same as full block permutation. The sequential algorithms have not been optimized or benchmarked, because they are intended only for proof-of-concept and not production. However, the hardware proposed in Section 7

should work within $O(N)$ and may have a theoretical limit of $O(1)$.

There is value in blocks. A groupoid-theoretical block is a strongly coupled collection of cities that is weakly coupled to the rest of the cities. A block is not necessarily stable all the way to the optimal tour, but chances are that it will remain stable at least within a certain range of tour lengths, hopefully sufficient for our purposes. Hence, we prefer to use groupoid blocks for the optimization whenever possible.

The transformation of *block inversion*, or inverting a block in place, simply consists of reversing the order of the cities in the block. It is advantageous to combine block tunneling with block inversion in a single algorithm. The block is tunneled into a slot, the increment is calculated, locally as usual, and then the block is inverted in place and the new increment is calculated, also locally, because in both cases the internal length of the block remains the same for a symmetric problem.

Heuristic versions of block tunneling have been proposed.

Depth-first search algorithms, even tunneling algorithms, can cease to be effective at a local minimum they can't tunnel through. At that point, it may be necessary to switch to a more powerful algorithm, even if it is less efficient. The tunneling arsenal includes several more types of tunneling, such as full permutation of the cities in a block, or permutations of the sub-blocks in a block. It is not known if these types are needed in general. The case study was solved without them.

A worst-case scenario for a problem with *N* cities would be one where full permutation of the cities in a large block turns out to be necessary. The largest possible block size is $N/2$. Even if blocks of this size were necessary to solve a very hard problem, then we would still have reduced the original TSP problem of size *N* to a TSP problem of size $N/2$. It is still worth the effort.

5.3. Comparison of City Tunneling with Simulated Annealing

There are obvious similarities between the city-swapping algorithm *ct* and the well known method of Simulated Annealing (SA). However, *ct* is actually used in a very different manner. In comparison with SA, *ct* considers group-theoretical blocks as valuable entities, with their own identity and a value that deserves to be preserved. A direct comparison at the level of benchmarks is not possible at this time because *ct*, and the other sequential algorithms, are intended for research, and have been optimized for ease of change and early bug detection and diagnostic, but not for performance. Production code has not been developed because it needs special hardware, which does not exist yet. The following lists some of the most important conceptual differences between *ct* and SA.

- SA does not use scale-free fractals, as *ct* does, and as a result scaling in SA is limited and places limitations on the size of the solvable problems. Further improvements of parameter adjustments may be possible, but are not likely to result in major improvements. With *ct*, it should not be too difficult to come close to the theoretical limit of *O*(1), which means constant execution time independent of *N*, but this has not been demonstrated yet.
- In SA, the assumed heuristic transition probability between states depends not only on the states but also on the temperature, which is a global parameter and compromises the locality of TSP.
- In SA, blocks of cities[1] are viewed as mere computational artifacts that can be swapped and benefit the annealing process by increasing the acceptance rate (see for example [23]). This view results in a need for heuristics, such as a range window for block migrations, and the implementation is not obvious. In *ct*, instead, group-theoretical blocks are considered as individuals endowed with the property of invariance under transformations and as stores of valuable binding information. The preceding sentence is more meaningful when read backwards: the transformations of optimization should leave the bound information and the blocks invariant. Binding has occurred and is preserved by those individuals, giving them value and meaning, and making them seeds for the growth of even bigger blocks where even more binding and more meaning are preserved. The *ct* algorithm capitalizes on these properties, SA does not because it makes decisions based on heuristics, while *ct* makes decisions based on group theory, and does not need heuristics.
- *ct* considers the problem as a physical system and takes advantage of the lessons learned from Physics. SA does not. Nature has solved the optimization problem, SA is not there yet.
- SA can guarantee an acceptable solution in a fixed amount of time, but not the optimum solution. *ct* guarantees either the optimum solution or a partition of the problem by half or more.

[1] not to be confused with city blocks.

- SA is a probabilistic heuristic process, *ct* is a deterministic group-theoretical process.
- SA works by finding best neighbors for a given seed state, by sometimes is forced to find worst neighbors if the search algorithm gets trapped in a local minimum of that algorithm. To do this, a heuristic probability of acceptance is defined that prefers best neighbors but also allows worst neighbors with a lower acceptance rate. This is where temperatures come into play. *ct* uses well-defined techniques to tunnel out of traps.
- SA cannot tell a local minimum from a global one. *ct* can.

6. The Case Study

A case study of the gr120 TSP example was undertaken in order to demonstrate the applicability of the causal methods to NP-hard problems and estimate performance and other features. The problem is defined in the TSP Library [24]. Data for gr120 such as inter-city distances in XML format are available from [25]. A published optimum tour corresponds to our tour 6942A, defined below, and is available from [26]. The length of the optimum tour is 6,942 km. Display coordinates and city distances are found in [27]. Plots on Goggle Maps are found here [28].

Some results of the study are summarized in **Figure 1** in the form of a directed graph. The vertices of the graph represent tours, each labeled with its length and an upper-case letter such as A or B. The edges of the graph represent search paths followed by the optimization algorithms, and are labeled with the identification code of the corresponding algorithm (see Section 5). All tours referenced in the graph are explicitly given in the attached computer-readable and human-readable file TheTours.txt, [22] including both global optimum tours.

Problem gr120 has 120 cities. This is a closed, symmetric problem, meaning that the distances between cities are independent of the direction of travel. The city where the tour starts and ends is called the warehouse city. In order to express the TSP problem in causal notation, and as discussed in Section 3.1, it is necessary to add an extra city, which is a replica of the warehouse city at a later time. The 121 cities are numbered 0 to 120, with 0 being the warehouse city and 120 being the causal replica. In the causal theory, Equation (6) requires for city 0 to be first, and city 120 to be last in any tour. All other cities remain mobile and are allowed to bind in any order. There are two optima, labeled 6942A and 6942B. The vertical scale in the plot represents tour length. The scale is not uniform, but marks are provided at levels 0.5%, 1%, and 2% above the optimum. Many interesting features can be observed in the optimization graph. They are discussed in the sections that follow.

6.1. The Optimization Graph

The graph of **Figure 1** contains a total of 8 different, partially overlapping full paths that start from a random seed tour and end at one of the two optima. Each path consists of several segments—the edges of the graph. At each step, the optimization algorithms of Section 5 are applied in the order of increasing difficulty and power, with *ct* used first to completion, followed with the tunneling algorithms.

Three major sections can be distinguished in the graph, and are labeled as (a), (b), and (c). Section (b) is where the algorithms found the optimum for the first time, and also where I invested most of my research work. I worked with Section (b) for a long time while designing the algorithms in terms of the causal theory and trying to reduce manual intervention and automate procedures to bring them closer to a production level, all of that while rigorously avoiding the introduction of any heuristics. The use of heuristics would have considerably reduced the value of the theory and also the scope of its application. In addition, I had to dispel the possibility that the initial success could have been due to pure luck, and not to the systematic effectiveness of the search algorithms. And this is why I invested so much research in Section (b), particularly below the 2% level. Section (b) alone has yielded half of the 8 paths to optimality, suggesting that the approach to the optimum is robust and the optimum should not be too hard to find because it can be reached in many different ways.

Sections (a) and (c) represent preliminary production work, intended to test the algorithms, and highlight several of their features, although I never intended to bring them to full production capability. This algorithms, as explained above, are only for proof of concept, and not for full production. Sections (a) and (c) also help to dispel the notion that random seed tour 56081A has some hidden property or is in some sense unique in its ability to generate a good path. In fact, it is remarkable that this single seed, tour 56081A, has generated two completely different paths, one leading to optimum 6942A, and the other leading to the other optimum 6942B. This feature attests to the power and generality of causal optimization. Sections (b) and (c) were also instrumental for

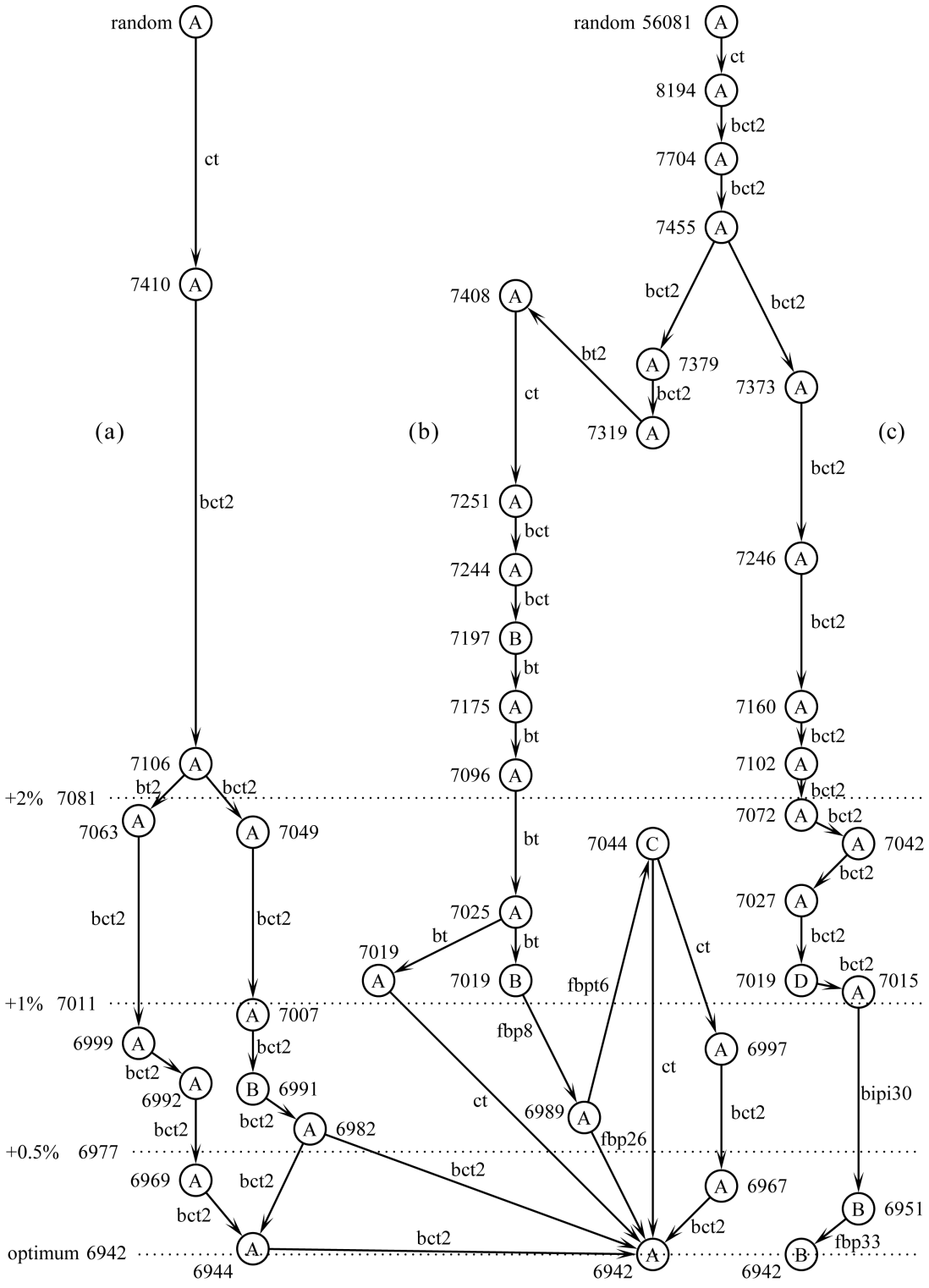

Figure 1. Group-theoretical solution of the NP-hard 120-cities gr120 TSP problem on a single-processor desktop computer in low-order polynomial time. Labels on the left identify macrostates (tour length). Node labels identify specific tours of that length. Arrow labels identify the specific algorithm that was successful in achieving that transition. The algorithms are described in detail in Section 5. The hardware proposal in Section 7 aims at developing multiprocessor hardware that uses local search and can solve NP-hard optimization problems including TSP in time comparable to the theoretical limit, hopefully a few microseconds.

obtaining an estimate of the execution time, which is close to 30 minutes. This estimate leaves plenty of room for improvement—perhaps by a factor of 10 or more—if the algorithms were to be brought to a full production level.

6.2. Steepest Descent

Nearly all segments of search in the graph are either steady descent or steepest descent. The only exceptions are in Section (b), and are the result of my intervention as part of the research. However, segments (7319A, 7408A, 7251A) and (6989A, 7044C, 6942A), are not an exception: both are steepest descent. Segment (7319A, 7408A, 7251A) is actually a single *bct*2 step, directly from 7319A to 7251A, which consisted of an initial *bt*2 step that generated 7408A, followed by the *ct* step to 7251A. The second segment, (6989A, 7044C, 6942A), is an *fbpt*6 to 7044C, followed by a *ct* to 6942A, as it should, except that I had not yet implemented what should have been the combination *fbpct*6 and had no choice but to perform both steps manually and plot the intermediate results. In other words, a *fbpct*6 step would be a maximum descent directly from 6989A to 6942A. The *ct* step from 7044C also happened to generate tour 6997A, which leads to 6942A by way of two light *bct*2 steps. I plotted this path as well.

The only two ascending arrows in the graph that I have chosen to plot teach us another important lesson. The first arrow ascends from 7319A to 7408A, for a total ascent of 89 km. The second arrow ascends from 6989A to 7044A, for an ascent of 55 km. The two ascents are different, and both are automatically determined by the algorithm and not by an arbitrary heuristic. No heuristics and no probabilities are necessary, as they are for example in SA.

6.3. Difficult Steps

There are two segments in the graph, *fbp*26 in Section (b), and *fbp*33 in Section (c), that need further discussion. It is not possible to calculate all permutations for blocks of these sizes using a desktop computer. But the initial 120-city TSP has been reduced to a 26-city TSP in the first case, or to a 33-city TSP in the second case, and the problem can be further pursued if needed. Alternatively, finding close-to-optimal solutions when computer power or time are limited is also of major importance. Tour 6989A is 0.6% above the optimum, and tour 6951B is only 0.13% above optimum, if an estimate of the position of the optimum is available, or they may just be short enough for the purpose.

However, in the first case, three different bypasses have been found. The first one is a single *bct* step directly from 7025A to the optimum 6942A, shown as an initial *bt* step to 7019A followed with a *ct* step to 6942A. The second bypass starts with a *fbpct*6 step passing through 7044C and then *ct* to optimum, and the third bypass is an *fbpt*6 step to 7044C followed with 3 light steps. The lesson we learn is that, whenever an excessive degree of difficulty is encountered, it may be worth to try alternatives.

In the last case, the *fbp*33 segment at the end of Section (3) in the graph, it may be possible to find a bypass, but I haven't tried because the intention was to present a clean production thread that aims directly to the optimum. However, this paper is for research, and all questions that help understanding are legitimate. But then, the question remains, how do we know that the final step requires *fbp*33 if we can't even get there? In this case, and for research purposes only, I compared tour 6951B with the optimum tour 6942B. Of course this answer is of no help for a search, but it is a valid action if the purpose is to gain understanding.

The comparison between 6951B and 6942B is shown at the bottom of TheTours.txt. It starts with a block of size 19 made of 19 trivial sub-blocks of size 1, identical in both tours. Then there follows a block of size 25 with the 4 cycles (5 47 72 61 83 101 56 98), (35 100 82 103 109 113 36 34 111 105 66 9), (79 38 26 62), and (4), of sizes 8, 12, 4, and 1, respectively. After this, there is a block of size 8 with the cycle (76 52 108 96 94 63 87 11), and finally a block of size 69 with 69 trivial sub-blocks, each of size 1, identical in both tours. The 4 cycles of the block of size 25 are inter-tangled, and not bound. The block of size 8 has only 1 cycle, hence it is bound, however *bipi*8 does not yield any improvement. The level of difficulty here can be as high as *fbp*33, as no kind of block tunneling can untangle the 5 cycles together. In this case, the initial TSP of 120 cities has been reduced to one of size not higher than 33 cities.

By contrast, all *bipi* steps, as the one near the end of Section (c), are very light regardless of the size of the block.

6.4. The Power of *bct*2

Section (a) of the optimization graph, with the only exceptions of the initial *ct* and the *bt*2 segment from 7106A to 7063A, is entirely made of *bct*2 segments. Section (c), again with the exception of initial *ct* and the 2 last segments *bipi*30 and *fpb*33, is also made of *bct*2. If counting segment (6944A, 6942A) twice, because it is part of two different optimization paths, then of the 7 segments that reach the global optimum 6942A, 4 are *bct*2. What gives *bct*2 such power? And why does it work so well with blocks of size 2, but not with other sizes?

I believe *bct*2 is so powerful because it allows the search process not only to tunnel out of a local minimum basin through its walls, as *bt* does, but also to overcome the basin entirely by jumping over its edge. After cities were stabilized by *ct* to form initial blocks and *bt* was invoked for every block size but failed to find a tunnel, we know that both *ct* and *bt* are trapped in a basin. This is the precise point where the steepest descent prescription must be abandoned, and the next logical step is to try to go over the edge of the basin. *bct*2 tunnels the same as *bt*2 does, except that, after each move, and even if the resulting tour is longer, it uses *ct* to stabilize cities and only then decides on whether to accept or reject the resulting tour. If the resulting tour is shorter that the seed, it means that the resulting tour is outside the *ct*, *bt* basin.

Going over the edge of a basin without knowing where the edge is located, is a new type of search. The bct algorithm effectively tries to jump over the edge or find a place where the wall is thinner and tunnel through it. In the process, it senses the height and shape of the wall as it crawls its way upwards. The indicator of escape is a tour shorter than the current seed, which is outside the basin. There are many ways to implement this algorithm and make it more efficient, but I implemented only the most basic form at this time.

7. Hardware Proposal

A preliminary concept for a hardware architecture that would approximate the theoretical limit for group-theoretical optimization methods was presented in Section 4 of [4]. It consists of a linear array of processors representing neurons, and an associative memory that consists of connections between the neurons representing dendrites. The number of processors must equal or exceed the number of elements in set *S*. The array itself represents the causal set *C* of Equation (1). The dendrites can be virtual, not necessarily physical. The neurons correspond to the elements of set *S*, and the dendrites to the set *w* of (cause, effect) pairs. Each core manages only its own dendrites, and tries to locally optimize their length. Minimum local synchronization is required at the level of the dendrites to avoid the simultaneous processing of the same dendrite by two different neurons. The sequential algorithms are not needed, the causal machine is not a parallel implementation of the sequential algorithms. The proposed architecture is one-dimensional, but two- and three-dimensional architectures should also be considered. GPU multicore units or experience gained with neuromorphic chips [11] may help, but this is a different design. There are no plans to build a first prototype, and hence no test results to be reported.

As explained in the Introduction, the prediction based on the causal theory, later confirmed by neuroscientific research, that dendritic trees must be optimally short, is an early success of the causal theory. The proposed architecture attempts to do what the brain does: save biological resources by making short dendrites, and in the process optimize the functional of Equation (8) and generate the patterns or hierarchies of information discussed in Section 2.4.

The crucial question that needs to be resolved is that dendrites in either the brain or the causal machine must satisfy two completely different conditions. They must be (or become) optimally short, and at the same time they must correspond to the (cause, effect) pairs in the partial order *w*. In humans, there are 86 billion neurons and about 10,000 dendrites per neuron. Hebbian learning explains how dendrites are formed, but it is not clear why there are so many of them or how they are made (or become) optimally short. A possible explanation might be the following. Neurons in the brain do not have access to each other except by way of their dendrites. There is no bus. Hence, dendrites play the role of sensors: they sense the environment around a neuron, and provide the neuron with the information it needs to select the best connection, one that corresponds to a (cause, effect) pair and at the same time is optimally short. This hypothesis appears to be confirmed by the pyramidal cells in layer 6 of the human neocortex, which may have as many as 200,000 dendrites each.

It may appear from the discussion that, once the “best” connections have been found for a neuron, only a few of the dendrites would remain in the circuit and the rest would be disconnected. This may be true for a static environment, such as the one we are discussing here. In a real-world scenario, the process of learning by way of a stream of causal information constantly coming into the brain from the senses, is anything but static. Set *S* and

partial order *w* constantly change as the result of new information, and a dendrite that is optimally short at one point may not be at another. To support the changes, the structure of dendrites must remain flexible, and this requires many dendrites per neuron. At least for short-term and mid-term memory.

In the case of long-term memory, the very slow process of methylation "freezes" the functioning dendrites (and may eliminate the rest, but this is not confirmed). This is how people keep childhood memories for their entire lives. The same mechanisms are proposed for the causal machine.

The cores in the causal machine are of course stationary in the architecture. However, it is easier to explain how the hardware works if one assumes that the neurons actually move, or virtually move along the linear array, constantly swapping their locations between adjacent neurons, and physically rearranging their locations in such a way that dendrites become optimally short. They know how to do this by counting the number of active dendrites that "pull" from them in each direction. If the differential pull acting on a neuron is positive (to the right), but negative for the neuron at its right, then the two neurons will swap their positions. All neurons constantly do this, hence they work in parallel, and their collective action quickly optimizes the overall length of the dendrites. In the brain, neurons do this to save biological resources, and not to optimize anything to create the structures of thought. Intelligence is a side effect of resource optimization, and was not intended by evolution. Note that this process is driven by resource preservation, and acts in the direction of minimizing the entropy and action in the system, towards point λ (see figure 1 in [1]). It opposes the causal entropic forces [29], which try to drive the system to point *S* in the plot, where entropy and disorder are maximum.

As the neurons adjust their locations in the array, they form clusters. These clusters are group-theoretical blocks. Why? They are clusters because they "resist", *i.e.* are invariant under the transformations of the swapping process, the internal binding in a cluster is stronger than external bindings that try to pull the cluster apart, and this is precisely the group-theoretical requirement for blocks to form.

Once the first clusters have formed, they tend to travel together. They do this simply because they are stronger than the rest of the neurons. Hence, now we have blocks of neurons swapping their locations with single neurons or other blocks of neurons. This swapping corresponds to the processing of the second-level causal set C' of Equation (2). The process continues recursively, forming blocks of blocks, and so on, until the hierarchy of invariants has formed and no more binding is possible. The process is not uniform along the array. There are parts of the array connected to input, where organization is only incipient, and other parts that grade towards optimum. In the brain, this transition is seen in the hippocampus, whereas organization is complete in the neocortex.

8. Conclusions

There is a number of mathematical properties that are being feverishly sought after in front line disciplines and technologies such as Big Data, Supercomputation, Discrete Optimization, Neuroscience, Complexity, Internet of Things, Machine Learning, Robotics, Surveillance, Computer Science. The properties include scalability, partition, recursion, structure, compression, granularity, determinism, certainty, prediction, fractals, power laws, meaning, semantics, and others. It turns out that these properties are precisely the mathematical properties of causal sets, and the reason they are lacking is that the mathematics of causal sets—the causal theory—has not yet been applied.

Causal sets are information in its most crude form. They come from all sources that information comes from. The properties are theoretical, hence they can be directly and quantitatively applied in engineering and development. There is no need to approximate them with heuristics or engineer them into the technologies, they are already there.

Many of the ideas described here are not just new but also unique in the literature. All the properties emerge from three factors: the universality of the principle of causality, the unification of information as a representation of the principle, and the power and generality of causal groupoids and their symmetries over more traditional groups. The methods are here described in the context of TSP, but they are general and can be extended to CML in general.

This paper has focused on explaining the theory and the properties of interest, and provided supporting evidence for the working hypothesis that a unified encoding of information as a causal set and the recursive groupoid-theoretical partition of NP-hard problems can compensate for their combinatorial complexity and lead to a deterministic solution in low-order polynomial time. To illustrate the concept, sequential, but non-heuristic algorithms were developed and applied for a case study of the Traveling Salesman problem, presented in this pa-

per. All experiments were successful and none failed. The theoretical limit for the algorithmic complexity may be as low as $O(1)$, but it is not clear how close to this limit a practical implementation can be. However, evidence exists that humans and some animals come close to the limit, at least in some cases. The option of using two-dimensional and three-dimensional hardware architectures has not yet been explored and remains wide open.

Evidence from computations can verify a theory and make it more believable and more reliable. I hope this goal is achieved and will help to open new fields of research and new possibilities in technology.

Acknowledgements

Felix Lanzalaco called my attention to literature about TSP in humans published in journals of Cognitive Psychology that I don't normally read. He also connected TSP with the phase offset in the human hippocampus [7]. Discussions with Tyler Bryson, Eitan Chatav, Neil Ghani, Tom LaGatta, and others on topics of Category theory and Robert Rosen's "Complexity and Life" are greatly appreciated. I thank everybody for their enthusiasm and encouragement.

References

[1] Pissanetzky, S. (2014) Causal Groupoid Symmetries. *Applied Mathematics*, **5**, 628-641. www.scirp.org/Journal/Home.aspx?IssueID=4511 http://dx.doi.org/10.4236/am.2014.54059

[2] Kauffman, S. (2011) Answering Descartes: Beyond Turing. *Proceedings of European Conference on Artificial Life* (*ECAL* 2011), Paris, 8-12 August 2011, 11-22. http://mitpress.mit.edu/sites/default/files/titles/alife/0262297140chap4.pdf

[3] Opdyke, W.F. (1992) Refactoring Object-Oriented Frameworks. Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana Champaign, Illinois. http://dl.acm.org/citation.cfm?id=169783

[4] Pissanetzky, S. (2012) Reasoning with Computer Code: A New Mathematical Logic. *Journal of Artificial General Intelligence*, **3**, 11-42. www.degruyter.com/view/j/jagi.2012.3.issue-3/issue-files/jagi.2012.3.issue-3.xml

[5] Cuntz, H., Mathy, A. and Hausser, M. (2012) A Scaling Law Derived from Optimal Dendritic Wiring. *Proceedings of the National Academy of Sciences of the United States of America*, **109**, 11014-11018. www.pnas.org/content/109/27/11014.abstract http://dx.doi.org/10.1073/pnas.1200430109

[6] Pissanetzky, S. and Lanzalaco, F. (2013) Black-Box Brain Experiments, Causal Mathematical Logic, and the Thermodynamics of Intelligence. *Journal of Artificial General Intelligence*, **4**, 10-43. www.degruyter.com/view/j/jagi.2013.4.issue-3/jagi-2013-0005/jagi-2013-0005.xml

[7] Lanzalaco, F. and Pissanetzky, S. (2013) Causal Mathematical Logic as a Guiding Framework for the Prediction of "Intelligence Signals" in Brain Simulations. *Journal of Artificial General Intelligence*, **4**, 44-88. www.degruyter.com/view/j/jagi.2013.4.issue-3/jagi-2013-0006/jagi-2013-0006.xml

[8] MacGregor, J.N. and Chu, Y. (2011) Human Performance on the Traveling Salesman and Related Problems: A Review. *The Journal of Problem Solving*, **3**, 1-29. http://docs.lib.purdue.edu/jps/vol3/iss2/2/ http://dx.doi.org/10.7771/1932-6246.1090

[9] Dorigo, M. and Gambardella, L.M. (1997) Ant Colonies for the Travelling Salesman Problem. *Biosystems*, **43**, 73-81. www.sciencedirect.com/science/article/pii/S0303264797017085 http://dx.doi.org/10.1016/S0303-2647(97)01708-5

[10] Martin, C.F., Bhui, R., Bossaerts, P., Matsuzawa, T. and Camerer, C. (2014) Chimpanzee Choice Rates in Competitive Games Match Equilibrium Game Theory Predictions. *Scientific Reports*, **4**, Article No. 5182. www.nature.com/srep/2014/140605/srep05182/full/srep05182.html http://dx.doi.org/10.1038/srep05182

[11] Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., *et al*. (2014) A Million Spiking-Neuron Integrated Circuit with a Scalable Communication Network and Interface. *Science*, **345**, 668-673. www.sciencemag.org/content/345/6197/668.abstract http://dx.doi.org/10.1126/science.1254642

[12] Zhai, Y., Ong, Y.S. and Tsang, I.W. (2014) The Emerging "Big Dimensionality". *IEEE Computational Intelligence Magazine*, **9**, 14-26. www.IEEE-CIS.org

[13] Huijse, P., Estevez, P.A., Protopapas, P., Principe, J.C. and Zegers, P. (2014) Computational Intelligence Challenges and Applications on Large-Scale Astronomical Time Series Databases. *IEEE Computational Intelligence Magazine*, **9**, 27-39. www.IEEE-CIS.org

[14] Zaremba, W., Szegedy, C., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R. (2014) Intriguing Properties of Neural Networks. Computer Vision and Pattern Recognition, arxiv.org/abs/1312.6199.

[15] Ng, A. (2014) RSS2014: 07/16 09:00-10:00 Invited Talk: Andrew Ng (Stanford University): Deep Learning. www.youtube.com/watch?v=W15K9PegQt0

[16] Pissanetzky, S. (2011) Emergence and Self-Organization in Partially Ordered Sets. *Complexity*, **17**, 19-38. http://dx.doi.org/10.1002/cplx.20389

[17] Connes, A. (1994) Noncommutative Geometry. Academic Press, San Diego. http://www.alainconnes.org/docs/book94bigpdf.pdf

[18] Hulpke, A. (2010) Notes on Computational Group Theory. www.math.colostate.edu/~hulpke/CGT/cgtnotes.pdf

[19] Fuster, J.M. (2005) Cortex and Mind. Oxford University Press, New York. http://ukcatalogue.oup.com/product/9780195300840.do

[20] Fuster, J.M. (2009) Cortex and Memory: Emergence of a New Paradigm. *Journal of Cognitive Neuroscience*, **21**, 2047-2072. http://cogsci.fmph.uniba.sk/~farkas/courses/Neurocomp/References/fuster.memory.jocn09.pdf http://dx.doi.org/10.1162/jocn.2009.21280

[21] Berut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R. and Lutz, E. (2012) Experimental Verification of Landauer's Principle Linking Information and Thermodynamics. *Nature*, **483**, 187-189. www.nature.com/nature/journal/v483/n7388/full/nature10872.html http://dx.doi.org/10.1038/nature10872

[22] Pissanetzky, S. (2014) Tours for the Traveling Salesman Problem gr120. www.scicontrols.com/Publications/TheTours.txt

[23] Eguro, K., Hauck, S. and Sharma, A. (2005) Architecture Adaptive Range Limit Windowing for Simulated Annealing FPGA Placement. Microsoft Research, *Design Automation Conference*, San Francisco, 14-17 June 2005, 439-444. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10020

[24] Reinelt, G. (1995) Discrete and Combinatorial Optimization. tsplib. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[25] Index of /groups/comopt/software/tsplib95/xmltsplib/instances. 1995. www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/XML-TSPLIB/instances/

[26] Assembla Oocplex. www.assembla.com/code/oocplex/subversion/nodes/3/objectOrientedIntegerProgramming/sampleData/TSPLIB/gr120.opt.tour

[27] Assembla Oocplex. www.assembla.com/code/oocplex/subversion/nodes/3/objectOrientedIntegerProgramming/sampleData/TSPLIB/gr120.tsp

[28] The Traveling Salesman Problem. www.math.uwaterloo.ca/tsp/

[29] Wissner-Gross, A.D. and Freer, C.E. (2013) Causal Entropic Forces. *Physical Review Letters*, **110**, Article ID: 168702. www.alexwg.org/publications/PhysRevLett_110-168702.pdf