Open Computational Creativity Problems in Computational Theory

Paul Bodily
Computer Science Department
Idaho State University
Pocatello, ID 83209 USA
bodipaul@isu.edu

Dan Ventura
Computer Science Department
Brigham Young University
Provo, UT 84602 USA
ventura@cs.byu.edu

Abstract

Despite clear benefits that would derive from their development, applications of computational creativity (CC) in math, science, and logic are heavily underrepresented in comparison with more artistic domains. In this paper, we examine the application of CC in the domain of computational complexity theory and identify several problems in the domain to which CC might be applied. In particular, we propose and define the task of creating reductions between NP-complete problems, the (sub)task of creating gadgets for use in constructing such reductions, and the task of gamification of reductions and argue that each of these may be addressed as interesting, fruitful CC challenge problems.

Introduction

Arguably the greatest achievements in human creativity have been in the fields of science, mathematics, and technology. And yet a 2017 review of application domains considered in computational creativity (CC) found that only 3% of 353 papers published over the preceding 12 years fell under the category of "Math, Science, and Logic" (Loughran and O'Neill 2017). This gap has been frequently mentioned in CC literature, and efforts have repeatedly been made to highlight the importance of applying CC in scientific and mathematical domains (Pease et al. 2019).

Computational complexity theory (CCT) represents a subfield of theoretical computer science that focuses on the classification of problems based on resource usage (e.g., time and memory) as well as how problems within and between complexity classes relate to one another. The classification of problems according to their complexity has profound real-world implications for the types of solutions that should be pursued for a particular problem and whether such solutions can be expected to be optimal. Besides providing mechanisms for proving the complexity of a particular problem, CCT also provides tools that can facilitate the reuse of existing algorithms to solve new problems.

In this paper we focus on the particular CCT subtopic of NP-completeness. Contributions in this subdomain tend to be impactful because such problems are ubiquitous in the real world and lie just beyond the grasp of modern computers when it comes to finding optimal solutions. NP-complete problems tend to take the form of optimization or decision problems with even minor improvements in algorithmic performance leading to significant cost savings in terms of time, energy, money, accuracy, or other value metrics. For this reason NP-complete problems have been studied in areas as diverse as advanced manufacturing (e.g., optimization of production lines, route inspection); computing/data/visualization (e.g., modularity maximization for graph visualization); homeland and cybersecurity (e.g., assembling an optimal Bitcoin block, cryptography); energy policy (e.g., the graph bandwidth problem in electronic design automation); energy-water (e.g., optimizing power/water flow across a network); innovative energy systems (e.g., the formulated energy and content aware vessel throughput maximization problem); and nuclear energy (e.g., the berth allocation problem as applied to reloading nuclear core fuel assemblies). The list of NP-complete problems grows ever longer.

We consider the main goal in this paper to be the framing and formal articulation of four important open problems in computational theory as CC problems. These problems are defined by characteristics that are typical of CC problems: each requires generation of a creative solution in a context with relatively well-established definitions of typicality, novelty, intention, and value. Similar to creating mathematical proofs, these problems are generally difficult even for trained humans. However, just like mathematical proofs, there are strategies that humans use that can aid in articulating a structured, generative process. Prerequisite to making substantive progress attempting solutions to these problems, the CC field needs a precise definition of these problems together with a clear understanding of the evaluative criteria associated with each. In essence, we aim to open a new potential subdomain of computational creativity to the CC field—the domain of NP-completeness in CCT—or, in other words, to bring awareness of the potential impact that computational creativity could have in a domain that has hitherto not been considered in the field of CC. We aim not merely to introduce the subdomain, but to articulate problems within this domain well enough that CC researchers will immediately be able to begin to innovate and implement CC solutions to these problems. Though the domain deals with theory, the practical implications are immediate and significant, and we will seek to highlight these as well.

Computational Complexity Theory

While there are many approaches to treating computation, we will find it convenient to consider computation from the perspective of determining set membership as, for example, is done in (Sipser 2013). Given a set of symbols (alphabet) Σ, we can define the set of all strings over that alphabet as Σ^*. We can then define a *language* $A \subseteq \Sigma^*$ as a set of strings. A language A is *decidable* if there exists a computable function $f_A : \Sigma^* \to \{0, 1\}$ such that[1]

$$f_A(w) = \begin{cases} 0 & \forall w \notin A \\ 1 & \forall w \in A \end{cases}$$

and we say that the language of f_A is A, $L(f_A) = A$. We can define computation as the problem of determining whether some string $w \in \Sigma^*$ is a member of a particular language A. For this reason, we use the terms *language*, *decision problem* (or simply *problem*), and *set* interchangeably. When speaking of decision problems, a string w being considered for membership in a set A is called an *instance* of problem A. As a running example, we will look at two decision problems in particular: 3SAT and CLIQUE.

3SAT In logic and computer science, a *Boolean literal* is either a variable, called a *positive literal*, or the negation of a variable, called a *negative literal*. A *clause* is a disjunction of literals (or a single literal). A *Boolean formula* is in *conjunctive normal form* (CNF) if it is a conjunction of clauses (or a single clause). A formula ϕ is 3CNF if the formula is in CNF and each clause in ϕ contains exactly 3 literals. Given a 3CNF Boolean formula ϕ, the 3SAT problem is to determine whether ϕ is satisfiable, i.e., whether or not there exists an assignment to each variable in ϕ such that ϕ evaluates to true. Using $\langle\phi\rangle$ to denote the string representation of ϕ, 3SAT is defined as the following decision problem:

$$\text{3SAT} = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3CNF formula}\} \quad (1)$$

A specific example of an instance of 3SAT is shown in Figure 2a. Many real-world problems in domains such as artificial intelligence, circuit design, and automatic theorem proving are representable as 3SAT instances.

CLIQUE In graph theory, a *graph* $G = (V, E)$ consists of a set $V = \{v_0, \ldots, v_n\}$ of *nodes* or *vertices* and a set of edges E. For *directed graphs* an edge $e = (v_i, v_j)$ is an ordered pair where order indicates the direction of the edge; for *undirected graphs* an edge $e = \{v_i, v_j\}$ is an unordered pair. A *clique* in an undirected graph is defined as a subset of nodes $V' \subseteq V$ for which $\forall v_i, v_j \in V', \{v_i, v_j\} \in E$. Given a graph G and an integer k, the CLIQUE problem is that of determining whether or not there exists a clique in G of size $\geq k$. Using $\langle G, k\rangle$ to denote the string representation of a G, k pair, CLIQUE is defined as the following decision problem:

$$\text{CLIQUE} = \{\langle G, k\rangle \mid G \text{ contains a clique of size } \geq k\} \quad (2)$$

[1] e.g., a Turing machine that halts with 0 on its tape $\forall w \notin A$ and halts with 1 on its tape $\forall w \in A$.

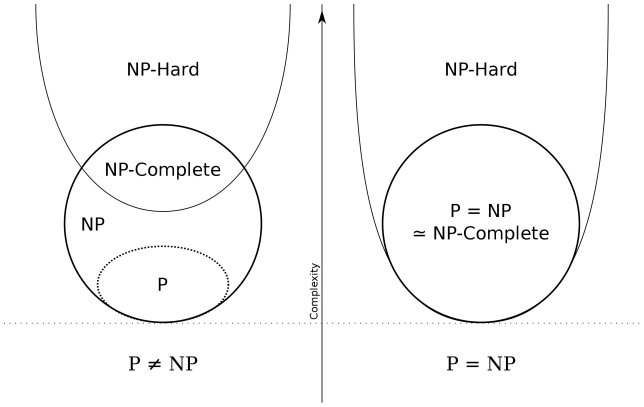


Figure 1: *Does P = NP?*. Two different views of some important computational complexity classes. It is unknown which view is correct, though most researchers believe P $\neq$ NP. If this is the case, several important theoretical questions about the class NP-complete, with significant, practical implications, provide interesting potential for CC research.

An instance of the CLIQUE problem is shown in Figure 2b. The CLIQUE problem has been used to represent instances of many real-world problems in domains such as social networks, bioinformatics, and computational chemistry.

The Theory of NP-completeness

The theory of NP-completeness offers a way to classify decision problems according to their inherent complexity. A problem is said to be in the class P if it can be decided (i.e., solved) by a polynomial-time algorithm.[2] A problem is said to be in the class NP if a solution to the problem (sometimes called a *certificate*) can be verified to be correct by a polynomial-time algorithm. Clearly problems that can be solved in polynomial time can also be verified in polynomial time, and therefore P $\subseteq$ NP. It is an open question of broad interest whether P = NP or P $\neq$ NP (see Figure 1).

The fascination with these two particular complexity classes stems from the fact that only polynomial-time algorithms can be effectively computed in reasonable time by classical computers for non-trivially-sized inputs. For all practical purposes, most computer scientists assume P $\subset$ NP, and this belief is largely perpetuated by the existence of a third class, NPC, of problems called *NP-complete* problems. This is a unique class of NP problems that are stubbornly resistant to being solvable by polynomial-time algorithms, and yet no one has been able to prove this barrier actually exists. NP-complete problems are considered the hardest problems in the class NP. But what makes them most fascinating is that every NP-complete problem is a *gateway problem*: the existence of a polynomial algorithm for deciding any one of them would mean that the entire class of languages is decidable in polynomial time. To be more specific, every NP-complete problem A can be reduced to every other NP-complete problem B (written $A \leq_{\text{P}} B$)

[2] A polynomial-time algorithm is an algorithm whose run time can be bounded with a polynomial function of the size of the input.

via some polynomial-time reduction algorithm. As a consequence, if one NP-complete problem B is one day discovered to have a polynomial-time solution algorithm, then by transitivity every other NP-complete problem A can be solved in polynomial-time by first reducing it in polynomial-time to B and then using the polynomial-time solver of B to find a solution to A. This is the basis for proofs of NP-completeness—a language B is NP-complete if it can be shown that:

1. $B \in \text{NP}$
2. $\forall A \in \text{NP}, A \leq_{\text{P}} B$

NP-complete problems are ubiquitous in the real-world and play prominent roles across nearly every field,[3] and it is common for those tasked with solving such problems to use this approach to prove NP-completeness in order to justify the use of heuristic or approximation algorithms when solving such problems. Requirement 1, membership in NP, will not figure prominently into our arguments here. Requirement 2 is traditionally proven via transitivity: if there exists a polynomial time reduction to B from some language A already known to be in NPC, then because (by definition) all problems in NP reduce in polynomial time to A, all problems in NP reduce in polynomial time to B. In other words, another way of proving requirement 2 is to prove $\exists A \in \text{NPC}, A \leq_{\text{P}} B$.[4] This idea of a *reduction function* (or simply *reduction*) is formalized as

$$\exists f : \Sigma^* \to \Sigma^*, w \in A \iff f(w) \in B \tag{3}$$

Because NPC is concerned with time-complexity, there is an additional requirement that the function f is computable in polynomial time. If this reduction exists, then, because NPC is an equivalence class (with respect to reciprocal polynomial reducibility), there will also exist a second (distinct) polynomial-time reduction g:

$$\exists g : \Sigma^* \to \Sigma^*, w \in B \iff g(w) \in A \tag{4}$$

Both reductions play important roles for different reasons. Given a language B suspected to be NP-complete, a reduction f from a known NP-complete language A is important in proving B is NP-complete. But it is the reduction g that allows existing approximation and solution algorithms for deciding A to be used to decide B.

For the purposes of illustration, let us imagine that we have not yet determined whether or not the CLIQUE problem is NP-complete and that we want to prove that it is. Let us assume we have shown CLIQUE∈NP (satisfying requirement 1). All that remains is to show a valid reduction from an existing NP-complete problem, e.g., 3SAT. As any computational theorist will attest, this is a scenario in which a fair amount of creativity (in the CC sense of the word) must be employed: finding a valid reduction from one NP-complete problem to another. Algorithm 1 from (Sipser 2013) gives pseudocode for a reduction 3SAT $\leq_{\text{P}}$ CLIQUE, and Figure 2 shows the output (2b) of Algorithm 1 for the input (2a). In this example, the string $w = \langle\phi\rangle$ for ϕ shown in Figure 2a, and, because there is a satisfying truth assignment for ϕ (i.e., $x =$ FALSE, $y =$ TRUE), $\langle\phi\rangle \in$ 3SAT. $f(w) = \langle G, k\rangle$ for (G, k) shown in Figure 2b, and because there is a k-clique in G [i.e., $(y_3, \overline{x}_4, \overline{x}_7)$], $\langle G, k\rangle \in$ CLIQUE, as required.

It is worth pausing to note some details about this reduction. Both the 3SAT instance (2a) and the equivalent CLIQUE instance (2b) have modular elements that are parallel between them. For each clause in the 3SAT instance there is a corresponding subgrouping of 3 nodes in the CLIQUE instance (as reflected by the colored highlights). For each Boolean literal in the 3SAT instance there is a corresponding node in the CLIQUE instance. These modular elements and groupings are found in every NP-complete problem and are commonly referred to as *gadgets*. Identifying both the quantity and nature of gadgets in an NP-complete problem is an important first step towards finding a valid reduction because ultimately a reduction is simply a matter of mapping the right gadgets (or some creative combination of gadgets) to one another. In this sense, one can think of NP-complete reductions as a form of analogical reasoning. Here then is a second scenario in which creativity must be employed: creating gadgets for NP-complete problems for use in reductions.

In addition to proving CLIQUE NP-complete, a reduction from 3SAT to CLIQUE also has a practical usage: it allows instances of 3SAT to be solved by existing solution algorithms for CLIQUE.[5] This is a remarkable and useful property of NP-complete problems that is surprisingly underappreciated. In short, rather than having to design, implement, and compare new solutions every time a new NP-complete problem B is discovered, one need simply reduce B to an existing NP-complete problem A and then apply and compare any number of existing solutions to A (or via transitivity to any other NP-complete problem for which reductions from A are available). This application of NP-complete reductions for leveraging existing solutions to NP-complete problems is of significant interest and is a topic we return to below.

Note finally that the reduction shown in Algorithm 1 is only one of an infinite number of valid reduction functions from 3SAT to CLIQUE. In addition, the reduction function itself is incomplete without an accompanying proof that the reduction is in fact a valid mapping reduction and that it is a polynomial-time function.

[3] In fact, it has been suggested that the problem of (computational) creativity itself is at least NP-hard, and may very likely be undecidable (Ventura 2014).

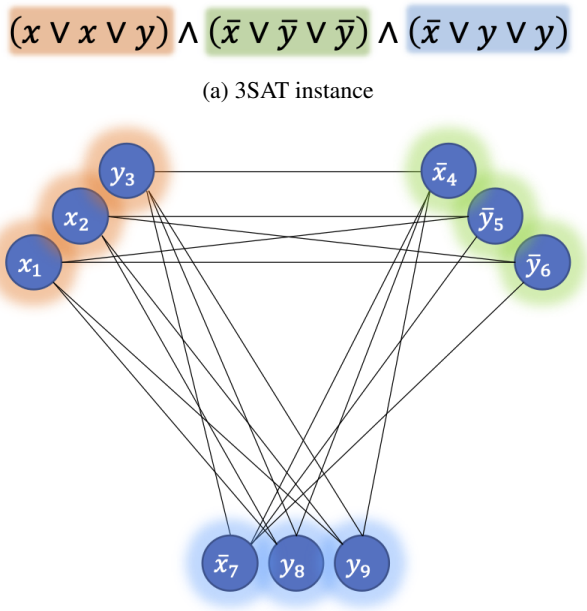
[4] This formulation presents a chicken-and-egg conundrum—from where do we get the first NPC problem? The conundrum is resolved with the the Cook-Levin Theorem (Cook 1971), which established Boolean satisfiability (SAT) as that problem by giving an elegant proof that $\forall A \in \text{NP}, A \leq_{\text{P}}$ SAT.

[5] Technically *solving* an NP-complete problem implies finding an optimal solution, but where such is impractical for NP-complete problems, the term *solve* usually refers to the use of heuristic or approximation algorithms to find good, but nonetheless suboptimal, solutions in a more tractable time frame.

Algorithm 1 Reduction from 3SAT to CLIQUE

Require: A 3CNF Boolean expression ϕ

procedure REDUCE(ϕ)
  $N \leftarrow \{\lambda_i | \lambda_i$ is the ith instance of literal λ in $\phi\}$
  $V \leftarrow \{(\lambda_i, \nu_j) | \lambda_i, \nu_j \in N$
    and λ_i, ν_i are not in the same clause in ϕ
    and $\lambda \neq \overline{\nu}\}$
  $G \leftarrow (N, V)$
  $k \leftarrow$ the number of clauses in ϕ
  return G, k



(a) 3SAT instance

(b) CLIQUE instance ($k = 3$)

Figure 2: *3SAT to CLIQUE reduction*. (a) an instance of the 3SAT problem and (b) equivalent CLIQUE instance to which the 3SAT instance reduces. Matching clause gadgets are highlighted with colors. Both the function (Algorithm 1) that maps the 3SAT instance to the CLIQUE instance as well as the individual gadgets in the generated CLIQUE instance represent artifacts resulting from creative processes.

Analogical reasoning

The concept of using a reduction f to compare an instance of problem A to an equivalent instance of problem B is, in some sense, a formalization of analogical reasoning: a is to A as b is to B. Finding f is essentially finding the relationship that makes the analogy valid. While this form of analogy has not yet been addressed in the CC literature, there has been work on other forms, including lexical analogy using WordNet (Hayes, Veale, and Seco 2004); bilingual lexical analogy using HowNet, an ontology for English and Chinese (Veale 2006); cross-domain analogical reasoning for improved text generation (Hervás et al. 2006); analogy emerging as a consequence of concept space exploration (Thornton 2008); constructing visual analogies of the kind found on intelligence tests (McGreggor, Kunda, and Goel 2010); analogical reasoning for mathematical creativity (Pease, Guhe, and Smaill 2010); using analogy for story generation (Zhu and Nón 2010); an autonomous system for generating analogical comparisons (O'Donoghue and Keane 2012); analogy to facilitate concept-blending (Besold and Plaza 2015); and transforming song lyrics using vector-based analogy in word embeddings (Oliveira 2020).

Four CC Problems in NP-completeness Theory

Having outlined the basic concepts relevant to NP-completeness, we can now identify four open questions/problems in this area that are ideally suited for being addressed by CC systems:

1. Given NP-complete problems A and B, can we create a valid polynomial-time reduction from A to B?
2. Given an NP-complete problem A, can we define meaningful gadgets for A that would be helpful in creating a valid polynomial-time reduction to/from A?
3. There are many examples of games/puzzles that are NP-complete. Given an NP-complete problem A and an NP-complete game/puzzle G, can we either create a new reduction or modify an existing reduction from A to G such that the reduced game/puzzle instances of G are fun/engaging?
4. Given an NP-complete problem A, can we create an efficient, effective polynomial-time heuristic or approximation algorithm to solve A?

Note that only the last of these proposed artifacts represents an actual (approximate) solution to an NP-complete problem. While creating a system that produces algorithmic (approximate) solutions to arbitrary NP-complete problems has not yet been addressed directly in the CC literature, there has been some work on CC systems/approaches for producing computer programs to solve arbitrary problems (Cook, Colton, and Gow 2013; Charnley et al. 2016; Znidarsic et al. 2016; Colton, Powley, and Cook 2018; Colton et al. 2019), and, to our knowledge, this is the only one of the four questions that has been given previous consideration in the CC literature.[6] So, although we include it as an example of a CC problem from the domain of CCT, we recognize that CC for computer programming is of much broader interest and in some sense its own subdomain of CC. And as well it should be; while the first three problems are well-defined in terms of the typicality constraints they must satisfy, the intention that they must meet, and the value they should provide, the creation of arbitrary computer programs for solving arbitrary problems is much less well-defined. For these reasons, we will focus the following discussion on the first three problems and direct the reader to extant literature that addresses the fourth.

Here we consider each of the first three questions/problems in more detail. We define each as a decision

[6]Recently, some work has also been done on the reverse problem—applying software engineering principles to the problem of designing CC systems (Glines, Griffith, and Bodily 2021).

problem and discuss notions of typicality, novelty, intentionality, and value in each of the three domains.

Artifact 1: NP-Complete Reduction Algorithms

Given NP-complete problems A and B, can we create a valid polynomial-time reduction from A to B? This question represents an open and challenging problem in CCT that essentially presents the set of all valid polynomial-time reductions as (potentially) creative artifacts. Represented as a decision problem, this set could be written as

$$\begin{aligned}\text{REDUCTIONS} = \{\langle A, B, f\rangle \mid A, B \in \text{NPC and} \\ f \text{ is a polynomial-time} \\ \text{reduction from } A \text{ to } B\}\end{aligned} \tag{5}$$

In order to meet standards of typicality for artifacts in this domain, a reduction f must meet at least two basic criteria: first, f must be a valid reduction from A to B—that is, it must be proven that $w \in A \iff f(w) \in B$; second, a reduction must operate in polynomial-time. Besides being necessary for typicality, a well-formed proof demonstrates intentionality in the reduction artifact.

In the authors' personal experience, the creation of a reduction function is an iterative process that starts simply with finding functions that translate a well-formed instance of problem A into a well-formed instance of problem B followed by experimentation and (if experimentation is successful) a formal proof. If experimentation is unsuccessful, the function is revamped. In light of this, even a system that is capable of translating a well-formed instance of problem A into a well-formed instance of problem B would possess significant value as a co-creative agent.

Novelty in this domain is primarily a function of whether *any* reduction from A to B has been previously documented. CCT guarantees that there exists a valid polynomial-time reduction between every pair of NP-complete problems. However, compared to the vast number of reductions that we know exist, relatively few reductions have been published. Several efforts have been undertaken to catalog what reductions have been published. The Redux platform discussed below represents an effort currently underway to make such reductions more accessible via Web APIs and a pedagogical visualization tool. Where a reduction has been published for a particular A, B pair, novelty can still be measured by comparing the similarities/differences between the several reductions that have been presented.

In assessing the value of a particular reduction, there are a few characteristics worth considering. First, valued reductions tend to be those which reduce instances of A to simpler (i.e., smaller) instances of B. For example, if one 3SAT-CLIQUE reduction algorithm reduces a 3CNF Boolean formula ϕ with k clauses to a graph with $3k$ nodes and a second reduction reduces ϕ to a graph with $4k$ nodes, we would value the simpler graph (all else held equal). Second, valued reductions are explainable reductions. Explainability is a metric that has previously been suggested for assessing value (Bodily and Ventura 2018).

Artifact 2: NP-Complete Reduction Gadgets

Given an NP-complete problem A, can we define meaningful gadgets for A that would be helpful in creating a valid polynomial-time reduction to/from A? This question essentially presents the set of all possible gadgets for a problem as (potentially) creative artifacts. The notion of what defines a gadget is inherently ambiguous as it varies from one problem to another and depends on whether the problem is on the input or output end of the reduction. This is a domain that will be easier to define as more and more examples of gadgets are cataloged. In general, we can think of a gadget $t(w)$ as a function that, given an instance w of an NP-complete problem, returns some collection of subunits of w. Represented as a decision problem, we could write this set as

$$\begin{aligned}\text{GADGETS} = \{\langle A, t\rangle \mid A \in \text{NPC and } \forall w \in A, t(w) \\ \text{generates a collection of subunits of } w\}\end{aligned} \tag{6}$$

We have seen how gadgets are valuable for the role they play in reductions. However, it is likely that gadgets could have value in other contexts, as well. For example, consider the goal of designing an algorithm that, given an NP-complete problem A, generates a greedy heuristic algorithm to solve A. Many such algorithms consist of little more than a few nested while loops iterating over what essentially amount to gadgets (e.g., a greedy heuristic algorithm for 3SAT would likely involve some sort of loop over clauses with an inner loop over variables within the clause). In general, we consider that defining the concept of a gadget for a particular problem has the potential of being a valuable creative artifact independent from whatever context in which it might be used.

With this in mind, intentionality in the definition of gadgets could be fixed on their intended use. When gadgets are intended for use in designing reduction functions, their value would depend on whether or not they contribute to a valid reduction. Again, simple gadgets are (all else held equal) valued over more complex gadgets. Whereas explainability serves as a meaningful value metric for reductions, it is sometimes more difficult to make an argument for this metric with respect to gadget artifacts, though certainly, gadgets that are intuitive or that do elucidate the construction or interpretation of a reduction will be of high value.

The novelty of a gadget not only depends on the definition of the gadget itself but also on the context in which it is used. For a given problem A, different gadgets for A become useful in reductions to/from different problems so that the presentation of a particular gadget when constructing a new reduction to/from a problem B could be considered a form of novelty.

In general, a typical gadget is some atomic unit of a problem instance and typically gadgets are exact subsequence/subset/subgraph units of an instance.

Artifact 3: NP-Complete Game Instances

Given an NP-complete problem A and an NP-complete game/puzzle G, can we either create a new reduction or modify an existing reduction from A to G such that the reduced game/puzzle instances of G are fun/engaging? We

could, of course, consider instead the problem of simply trying to make game/puzzle instances of an NP-complete problem A more creative. But again, this is not particularly unique to CCT (many researchers have considered the challenge of making games more creative). Far more interesting and specific to CCT is consideration of how to amplify creativity in games/puzzles *that are formed as reductions from other NP-complete problems*. Represented as a decision problem, we could write this set as

$$\begin{aligned}\text{GAME} = \{\langle A, G, f\rangle \mid\ & A \in \text{NPC and} \\ & G \in \text{NPC is a game or puzzle} \\ & \text{and } f \text{ is a polynomial-time} \\ & \text{reduction from } A \text{ to } G\}\end{aligned} \tag{7}$$

This problem is what initially piqued our interest in applying CC to CCT: could we take an arbitrary NP-complete problem from the real world and turn it into a game or puzzle that people would find engaging? Human intuition is remarkably adept at finding good solutions to NP-complete problems. Unfortunately, people do not typically enjoy solving Boolean satisfiability or graph theory problems. But they do like games. If we can render arbitrary NP-complete problems as fun and engaging games or puzzles then we can leverage the power of crowd-sourcing to find solutions that may be better than any computer could come up with (Cusack et al. 2010).

As an example, consider the protein folding game FoldIt (Cooper et al. 2010). According to its website,

> *Knowing the structure of a protein is key to understanding how it works and to targeting it with drugs. The number of different ways even a small protein can fold is astronomical because there are so many degrees of freedom. Figuring out which of the many, many possible structures is the best one is NP-complete and is regarded as one of the hardest problems in biology today. Current methods take a lot of money and time, even for computers. Foldit attempts to predict the structure of a protein by taking advantage of humans' puzzle-solving intuitions and having people play competitively to fold the best proteins*

See Figure 3 for an example screenshot of the game.

Our initial foray into this problem was an attempt to reduce the well-known NP-complete travelling salesperson problem to a popular NP-complete flood fill game called KAMI. We know from CCT that a reduction exists. However, despite months of trying, we have yet to devise a valid solution (much of this time was spent creating and combining different gadget artifacts from these two problems). We are aware of a reduction from the shortest common supersequence problem to flood fill puzzles which provided some ideas on how gadgets could be created for the KAMI problem (see Figure 4) (Marchetti and Bodily 2022; Clifford et al. 2012).

Many games and puzzles have been shown to be NP-complete including well-known examples such as *Battleship* (Sevenster 2004), *FreeCell* (Helmert 2003), *Instant Insanity* (Garey and Johnson 1979), *LaserTank* (Alexandersson and Restadh 2020), *Pandemic* (Nakai and Takenaga 2012), *Rubik's Cube* (Demaine, Eisenstat, and Rudoy 2018) and *Sudoku* (Yato and Seta 2003). Because these games are well-known and fun, they are excellent candidates for reduction targets from other NP-complete problems.

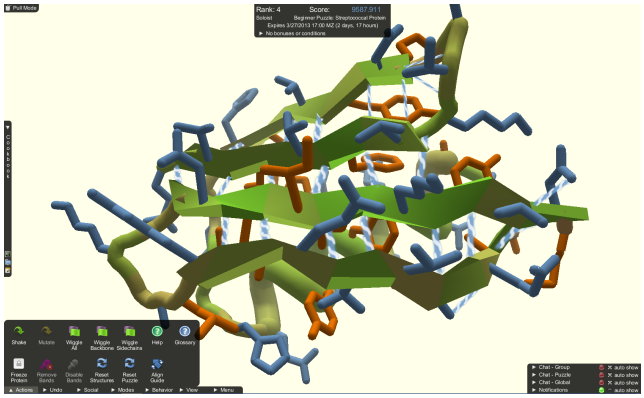


Figure 3: *FoldIt*. FoldIt is a crowd-sourced game for solving difficult instances of the protein folding problem, an NP-complete problem from biology, the solutions to which have implications in understanding protein function (and thus also for designing medical or other interventions). Screen shot from the game taken from `https://fold.it`.

As far as what defines creativity in this domain, since art and gaming already enjoy broad treatment in the CC field, more traditional definitions of novelty, value, typicality and intentionality can be invoked directly to assess the quality of this artifact type. Although the definitions of GAME and REDUCTION are very similars, in the case of REDUCTION the focus is on the behavior of creating novel, valid reductions *ab initio*. In the case of GAME, we take a valid reduction for granted and focus on the creativity of the artifacts generated *from* the reduction. One possible approach to attacking this problem may be related to procedural puzzle generation (De Kegel and Haahr 2020).

There are two concerns that should be mentioned with regard to reducing NP-complete problems to CC-enhanced games. First, most NP-complete games are played with instances of very limited input sizes. An instance of the game Battleship, for example, is defined in terms of the size of the board (typically 10×10) and the number of ships (typically 5). One can easily imagine reductions from arbitrary NP-complete problem instances that could result in very large game instances (imagine playing Battleship on a $10,000 \times 10,000$ board with 5,000 ships), much larger than human players are used to playing and larger perhaps than would appeal to many players. This diminishing value with increasing input size is certainly relevant to considerations on how CC might be used to attempt to create valuable artifacts in this space. It is worth noting that the FoldIt game (Figure 3) is at least one example of an NP-complete game with non-trivially-sized instances that has seen success.

Second, reduction algorithms tend to be highly prescriptive which could severely limit the variability with which game instances could be rendered. For example, KAMI has

(a) (b)

Figure 4: *Limitations on creativity in derived KAMI puzzles*. (a) KAMI is an NP-complete flood fill game whose puzzles typically allow a wide range of aesthetic expression. (b) Due to the highly prescriptive nature of reduction algorithms, a KAMI puzzle that is derived via reduction from an instance of the NP-complete shortest common supersequence problem will always necessarily be composed of diamond gadgets, like those shown, which significantly constrains the ways in which CC could be applied to enhance the creativity of the puzzle without invalidating the reduction.

come to be known for its highly aesthetic and creative puzzles (e.g., see Figure 4a). However, when we take a different NP-complete problem, e.g., the shortest common supersequence (SCS) problem, and reduce it to KAMI, the result is always a puzzle consisting of several diamonds (the diamond is a gadget), each consisting of a pattern of concentric multicolored rings (color is another gadget), all on a common background canvas color (see Figure 4b). The number, size, color scheme, and even shape (to some extent) of the diamonds could change without invalidating the reduction, but otherwise, all KAMI puzzles generated from a particular SCS reduction will follow a similar pattern (Clifford et al. 2012). It is possible that other reductions could potentially produce other patterns. The point being made here is that the nature of reductions is highly prescriptive and consequently places some limits on *how* CC would need to be applied to enhance the creativity of puzzles derived from NP-complete reductions in order not to invalidate the reductions.

An Ontology of NP-completeness

CC systems across a spectrum of application domains rely on knowledge bases of existing artifacts from which they extract patterns for the creation of new artifacts (Ventura 2017). Though some efforts have been made to create a knowledge base of NP-complete problems, there does not exist a well-established resource cataloging NP-complete problems, reductions, and/or solutions. To this end we have undertaken to create Redux, an ontological knowledge base of NP-complete problems, reductions, solutions, and verifiers accessible via a web API[7]. In addition to the knowledge base, we also aim to build a pedagogical visualization front end to the knowledge base. A mockup of the system can be seen in Figure 5.

[7]http://redux.aws.cose.isu.edu/

We envision the Redux knowledge base allowing researchers to perform meta-analyses over various aspects of NP-complete problems in order to gain insights on such questions as:

- What gadgets have been identified/created before when reducing to/from a particular NP-complete problem?
- What patterns exist in how gadgets for one problem map to gadgets for another problem?
- What in general is the relationship between previously identified gadgets for a particular NP-complete problem and the formulation of, say, a greedy heuristic solution algorithm for the problem?
- What additional power or knowledge can be leveraged via transitivity of the reductions in the knowledge base?

In addition, researchers will be able to directly access NP-complete problem definitions, example instances, and formatting; call reduction algorithms between particular pairs of NP-complete problems; run solution algorithms for particular NP-complete problems; and verify proposed solutions for particular NP-complete problem instances. Our hope is that this knowledge base will spur innovative ideas and solutions in and around the domain of NP-completeness.

Conclusion

The CC research community has long been interested in balancing its focus on applications in artistic domains with more CC applications in the fields of science, mathematics, and logic. Our purpose in this paper has been to suggest that computational complexity theory, and NP-completeness in particular, is a ripe candidate for contributing to this balance. We have attempted to highlight and provide some definition to four open CC problems in this domain. We have argued that progress towards addressing these problems promises to make significant impacts in the field of CCT, which in turn promises to make significant impacts in many real-world domains.

In addition we have presented Redux, a nascent ontological knowledge base of NP-complete problem definitions, reductions, and solution algorithms. Our intent is to augment the knowledge base through crowd-sourcing with the ultimate goal of providing a comprehensive and accessible resource on NP-completeness by which CC and other researchers can push forward the boundaries of applied computational complexity research. As a final note, it is worth mentioning that many of the significant open problems in applying CC are themselves likely NP-complete (or harder) problems. And though it is also likely that creativity itself lies beyond the realm of NP-completeness, advances in CCT are likely to translate directly into advances in the field of computational creativity.

Author Contributions

Both authors contributed to all aspects of the work, including ideation, narrative/position development and writing.

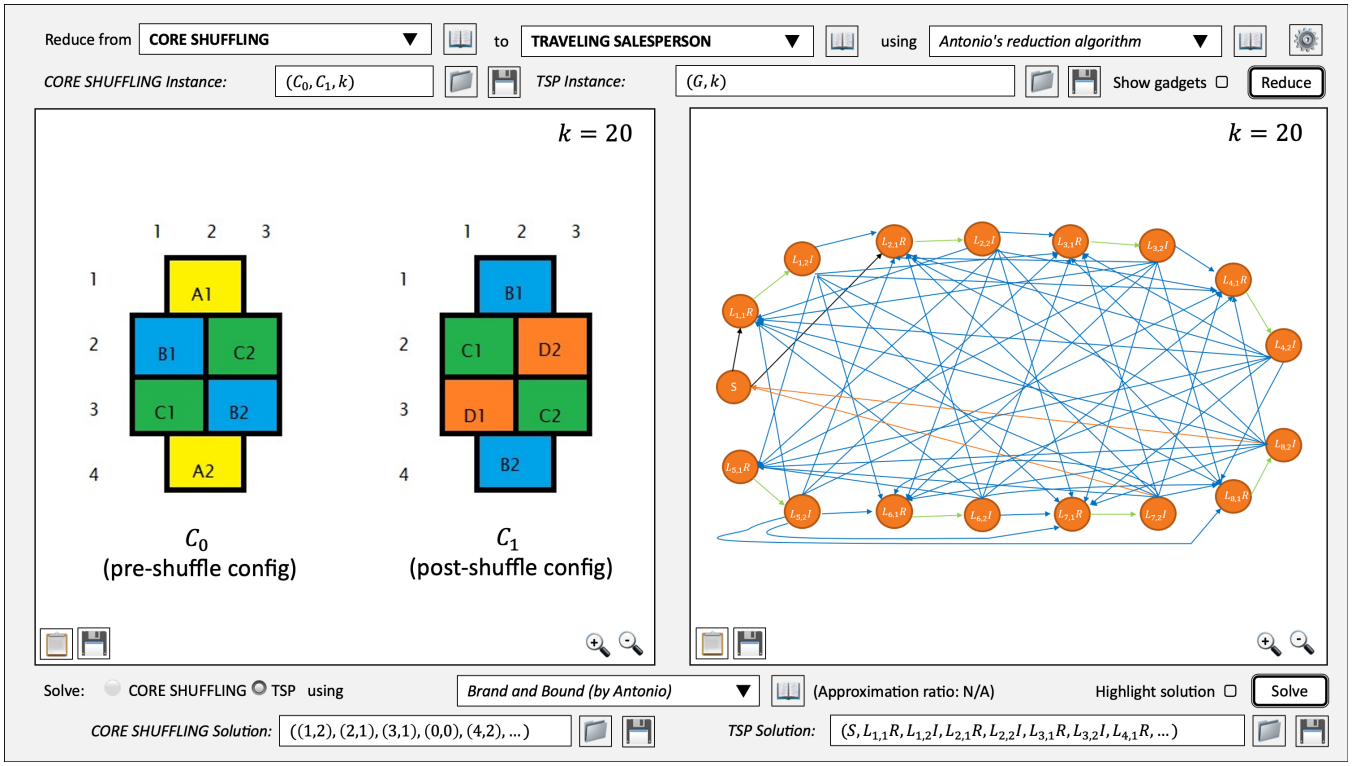


Figure 5: *The Redux application*. The tool shown serves as a graphical user interface providing access to a crowd-sourced knowledge base of NP-complete problems, reduction algorithms, and solution algorithms. Users can select (or add new) problems. A unique reduction algorithm is required for each unique (ordered) pair of selected problems. Users can contribute new reduction algorithms. Then for a given instance of the problem on the left [e.g., (nuclear) CORE SHUFFLING], the reduction algorithm is applied to generate an equivalent instance of the problem on the right (e.g., TRAVELING SALESPERSON). A solution to one problem instance can also be mapped to the equivalent solution for the equivalent problem instance. Visualization of instances with gadgets and/or solutions highlighted is included for pedagogical purposes. The tool highlights the power of reduction, allowing existing solutions to one problem to be reused to solve new problems. Possible applications of CC in this context include using CC to create novel reduction algorithms; using CC to propose gadgets for co-creative development of reduction algorithms; application of CC to aesthetically present reduced problem instances as engaging puzzles pursuant to crowd-sourcing solutions to NP-complete problems; and using CC to create novel solution algorithms.

Acknowledgments

The authors gratefully acknowledge Antonio Tahhan for contribution of the core shuffling diagrams and Mark Bodily for the contribution of the KAMI puzzle examples.

References

Alexandersson, P., and Restadh, P. 2020. Lasertank is NP-complete. In *Mathematical Aspects of Computer and Information Sciences*, LNCS 11989, 333–338.

Besold, T. R., and Plaza, E. 2015. Generalize and blend: Concept blending based on generalization, analogy, and amalgams. In *Proceedings of the Sixth International Conference on Computational Creativity*, 150–157.

Bodily, P., and Ventura, D. 2018. Explainability: An aesthetic for aesthetics in computational creative systems. In *Proceedings of the 9th International Conference on Computational Creativity*, 153–160.

Charnley, J.; Colton, S.; Llano, M. T.; and Corneli, J. 2016. The FloWr online platform: Automated programming and computational creativity as a service. In *Proceedings of the 7th International Conference on Computational Creativity*, 363–370.

Clifford, R.; Jalsenius, M.; Montanaro, A.; and Sach, B. 2012. The complexity of flood filling games. *Theory of Computing Systems* 50(1):72–92.

Colton, S.; Pease, A.; Cook, M.; and Chen, C. 2019. The HR3 system for automatic code generation in creative settings. In *Proceedings of the 10th International Conference on Computational Creativity*, 108–115.

Colton, S.; Powley, E. J.; and Cook, M. 2018. Investigating and automating the creative act of software engineering: A position paper. In *Proceedings of the 9th International Conference on Computational Creativity*, 224–231.

Cook, M.; Colton, S.; and Gow, J. 2013. Nobody's a critic: On the evaluation of creative code generators—a case study in video game design. In *Proceedings of the 4th Interna-*

tional Conference on Computational Creativity, 123–130.

Cook, S. 1971. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158.

Cooper, S.; Khatib, F.; Treuille, A.; Barbero, J.; Lee, J.; Beenen, M.; Leaver-Fay, A.; Baker, D.; Popović, Z.; et al. 2010. Predicting protein structures with a multiplayer online game. *Nature* 466(7307):756–760.

Cusack, C.; Largent, J.; Alfuth, R.; and Klask, K. 2010. Online games as social-computational systems for solving NP-complete problems. In *Meaningful Play*. Michigan State University East Lansing.

De Kegel, B., and Haahr, M. 2020. Procedural puzzle generation: A survey. *IEEE Transactions on Games* 12(1):21–40.

Demaine, E.; Eisenstat, S.; and Rudoy, M. 2018. Solving the Rubik's cube optimally is NP-complete. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science*, article 24.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman.

Glines, P.; Griffith, I.; and Bodily, P. 2021. Software design patterns of computational creativity: A systematic mapping study. In *Proceedings of the 12th International Conference on Computational Creativity*, 218–221.

Hayes, J.; Veale, T.; and Seco, N. 2004. The Bible is the Christian-Koran: Exploring lexical analogy via WordNet. In *Proceedings of the First Joint Workshop on Computational Creativity*, 59–64.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

Hervás, R.; Pereira, F. C.; Gervás, P.; and Cardoso, A. 2006. Cross-domain analogy in automated text generation. In *Proceedings of the Third Joint Workshop on Computational Creativity*.

Loughran, R., and O'Neill, M. 2017. Application domains considered in computational creativity. In *Proceedings of the 8th International Conference on Computational Creativity*, 197–204.

Marchetti, K., and Bodily, P. M. 2022. KAMI: Leveraging the power of crowd-sourcing to solve complex, real-world problems. In *Proceedings of the 2nd Intermountain Engineering, Technology, and Computing Conference*.

McGreggor, K.; Kunda, M.; and Goel, A. 2010. A fractal approach towards visual analogy. In *Proceedings of the First International Conference on Computational Creativity*, 65–74.

Nakai, K., and Takenaga, Y. 2012. NP-completeness of Pandemic. *Journal of Information Processing* 20(3):723–726.

Oliveira, H. G. 2020. Weirdanalogymatic: Experimenting with analogy for lyrics transformation. In *Proceedings of the Eleventh International Conference on Computational Creativity*, 228–235.

O'Donoghue, D., and Keane, M. T. 2012. A creative analogy machine: Results and challenges. In *Proceedings of the Third International Conference on Computational Creativity*, 17–24.

Pease, A.; Colton, S.; Warburton, C.; Nathanail, A.; Preda, I.; Arnold, D.; Winterstein, D.; and Cook, M. 2019. The importance of applying computational creativity to scientific and mathematical domains. In *Proceedings of the 10th International Conference on Computational Creativity*, 250–257. Association for Computational Creativity.

Pease, A.; Guhe, M.; and Smaill, A. 2010. Some aspects of analogical reasoning in mathematical creativity. In *Proceedings of the First International Conference on Computational Creativity*, 60–64.

Sevenster, M. 2004. Battleships as a decision problem. *ICGA Journal* 27(3):142–149.

Sipser, M. 2013. *Introduction to the Theory of Computation*. Boston: Cengage Learning.

Thornton, C. 2008. Analogy as exploration. In *Proceedings of the Fifth Joint Workshop on Computational Creativity*, 81–90.

Veale, T. 2006. Re-representation and creative analogy: A lexico-semantic perspective. *New Generation Computing* 24(3):223–240.

Ventura, D. 2014. Can a computer be lucky? and other ridiculous questions posed by computational creativity. In *Proceedings of the Seventh Conference on Artificial General Intelligence*, LNAI 8598, 208–217.

Ventura, D. 2017. How to build a CC system. In *Proceedings of the 8th International Conference on Computational Creativity*, 253–260.

Yato, T., and Seta, T. 2003. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E86-A(5):1052–1060.

Zhu, J., and Nón, S. O. 2010. Towards analogy-based story generation. In *Proceedings of the First International Conference on Computational Creativity*, 75–84.

Znidarsic, M.; Cardoso, A.; Gervás, P.; Martins, P.; Hervas, R.; Alves, A. O.; Oliveira, H.; Xiao, P.; Linkola, S.; Toivonen, H.; Kranjc, J.; and Lavrac, N. 2016. Computational creativity infrastructure for online software composition: A conceptual blending use case. In *Proceedings of the 7th International Conference on Computational Creativity*, 371–379.