Strong Partial Clones and the Complexity of Constraint Satisfaction Problems

Who? Victor Lagerkvist

From? TU Dresden, Institut für Algebra

When? September 8

Motivation

- The *constraint satisfaction problem* is a widely studied computational problem.

Motivation

- The *constraint satisfaction problem* is a widely studied computational problem.
- The *algebraic approach* offers a systematic approach for studying its compexity.

Motivation

- The *constraint satisfaction problem* is a widely studied computational problem.
- The *algebraic approach* offers a systematic approach for studying its compexity.
- Most research is devoted to separating *tractable* from *intractable* problems.

Motivation

- The *constraint satisfaction problem* is a widely studied computational problem.
- The *algebraic approach* offers a systematic approach for studying its compexity.
- Most research is devoted to separating *tractable* from *intractable* problems.
- In this talk we will look at generalizations allowing a more fine-grained complexity analysis.

Outline of the Presentation

1. The constraint satisfaction problem.
2. The algebraic approach.
3. A more refined approach.
4. Two non-trivial applications.

The Constraint Satisfaction Problem

Assume that we are given a map of Australia and want to colour its states with three colours, in such a way that two adjacent states are not assigned the same colour.

The Constraint Satisfaction Problem

This kind of problem is an example of a *constraint satisfaction problem*.

The Constraint Satisfaction Problem

This kind of problem is an example of a *constraint satisfaction problem*.

- We have some objects that we want to assign values to.

The Constraint Satisfaction Problem

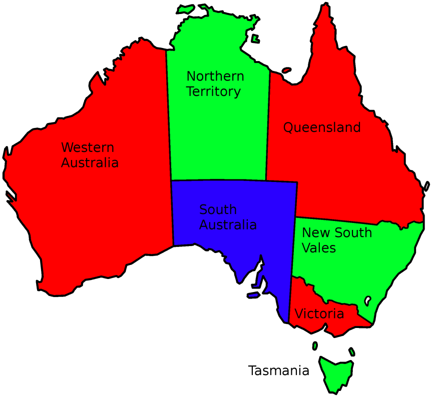
This kind of problem is an example of a *constraint satisfaction problem*.

- We have some objects that we want to assign values to.
- But when assigning values we have to do it in such a way that all *constraints* are satisfied.

The Constraint Satisfaction Problem

This kind of problem is an example of a *constraint satisfaction problem*.

- We have some objects that we want to assign values to.
- But when assigning values we have to do it in such a way that all *constraints* are satisfied.



The Constraint Satisfaction Problem

Definition

Let D be a finite set of values. A k-ary *relation* over D is a subset of the k-ary Cartesian product of D.
A set of relations S is called a *constraint language*.

The Constraint Satisfaction Problem

Definition

Let D be a finite set of values. A k-ary *relation* over D is a subset of the k-ary Cartesian product of D.
A set of relations S is called a *constraint language*. The *constraint satisfaction problem* over S (CSP(S)) is defined as follows.
Instance: A tuple (V, C) where V is a set of variables and C a set of constraints over V and S.
Question: Does there exist a function $f : V \to D$ such that $(f(x_1, \ldots, x_k)) \in R$ for every constraint $R(x_1, \ldots, x_k)$ in C?

The Constraint Satisfaction Problem

If S is Boolean the CSP(S) problem is sometimes denoted by SAT(S), the so-called *generalised satisfiability problem*.

The Constraint Satisfaction Problem

If S is Boolean the CSP(S) problem is sometimes denoted by SAT(S), the so-called *generalised satisfiability problem*.

Example

Let $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$ and let $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$. Then the two well-known NP-complete problems monotone 1-in-3-SAT and NOT-ALL-EQUAL-3-SAT can be formulated as SAT($\{R_{1/3}\}$) and SAT($\{R_{\mathrm{NAE}}\}$).

The Algebraic Approach

- Given a constraint language S, is CSP(S) tractable or intractable?

The Algebraic Approach

- Given a constraint language S, is CSP(S) tractable or intractable?
- The most successful approach to study this question is based on relating constraint languages with *algebras*.

The Algebraic Approach

Definition

Let R be a relation. An n-ary function f is a *polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ (applied componentwise).

The Algebraic Approach

Definition

Let R be a relation. An n-ary function f is a *polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ (applied componentwise).
Similarly f is a polymorphism of a constraint language S if it is a polymorphism of every relation in S. We also say that S is *invariant* under f or that f *preserves* S.

The Algebraic Approach

Definition Let R be a relation. An n-ary function f is a *polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ (applied componentwise).
Similarly f is a polymorphism of a constraint language S if it is a polymorphism of every relation in S. We also say that S is *invariant* under f or that f *preserves* S.

Example Let $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ and let $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$. Let neg be the unary function defined as $\mathrm{neg}(0) = 1$ and $\mathrm{neg}(1) = 0$.

The Algebraic Approach

Definition

Let R be a relation. An n-ary function f is a *polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ (applied componentwise).
Similarly f is a polymorphism of a constraint language S if it is a polymorphism of every relation in S. We also say that S is *invariant* under f or that f *preserves* S.

Example

Let $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ and let $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$. Let neg be the unary function defined as $\mathrm{neg}(0) = 1$ and $\mathrm{neg}(1) = 0$.
Then

- neg is a polymorphism of R_{NAE}, but

The Algebraic Approach

Definition Let R be a relation. An n-ary function f is a *polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ (applied componentwise).
Similarly f is a polymorphism of a constraint language S if it is a polymorphism of every relation in S. We also say that S is *invariant* under f or that f *preserves* S.

Example Let $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ and let $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$. Let neg be the unary function defined as $\mathrm{neg}(0) = 1$ and $\mathrm{neg}(1) = 0$. Then

- neg is a polymorphism of R_{NAE}, but
- neg is not a polymorphism of $R_{1/3}$ since $\mathrm{neg}((0,0,1)) = (1,1,0) \notin R_{1/3}$.

The Algebraic Approach

Definition If S is a constraint language then we let $\mathrm{Pol}(S)$ be the set of all polymorphisms of S.

The Algebraic Approach

Definition If S is a constraint language then we let $\mathrm{Pol}(S)$ be the set of all polymorphisms of S.

- Sets of the form $\mathrm{Pol}(S)$ are known as *clones*.

The Algebraic Approach

Definition If S is a constraint language then we let $\mathrm{Pol}(S)$ be the set of all polymorphisms of S.

- Sets of the form $\mathrm{Pol}(S)$ are known as *clones*.
- Clones are sets of functions closed under functional composition.

The Algebraic Approach

The polymorphisms of a constraint language determines the complexity of the CSP problem up to polynomial-time reductions.

Theorem (Jeavons et al.)

Let S and S' be two finite constraint languages. If $\mathrm{Pol}(S) \subseteq \mathrm{Pol}(S')$ then CSP(S') is polynomial-time many-one reducible to CSP(S).

The Algebraic Approach

The polymorphisms of a constraint language determines the complexity of the CSP problem up to polynomial-time reductions.

Theorem (Jeavons et al.)

Let S and S' be two finite constraint languages. If $\mathrm{Pol}(S) \subseteq \mathrm{Pol}(S')$ *then CSP*(S') *is polynomial-time many-one reducible to CSP*(S).

Very useful when separating tractable CSP problems from NP-complete CSP problems...

The Algebraic Approach

... But does not say anything about the relative worst-case time complexity for the NP-complete cases.

The Algebraic Approach

... But does not say anything about the relative worst-case time complexity for the NP-complete cases.

- 1-in-3-SAT is solvable in roughly $O(1.09^n)$ time.

The Algebraic Approach

... But does not say anything about the relative worst-case time complexity for the NP-complete cases.

- 1-in-3-SAT is solvable in roughly $O(1.09^n)$ time.
- 3-SAT is only known to be solvable in $O(1.308^n)$ time.

The Algebraic Approach

... But does not say anything about the relative worst-case time complexity for the NP-complete cases.

- 1-in-3-SAT is solvable in roughly $O(1.09^n)$ time.
- 3-SAT is only known to be solvable in $O(1.308^n)$ time.
- But both problems correspond to the same clone and are polynomial-time reducible to each other.

The Algebraic Approach

- Want something more fine-grained than polymorphisms.

The Algebraic Approach

- Want something more fine-grained than polymorphisms.
- One alternative is to consider *partial polymorphisms*.

A More Refined Approach

Definition

Let R be a relation. A partial function f is a *partial polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ such that $f(t_1, \ldots, t_n)$ is defined for each componentwise application.

A More Refined Approach

Definition Let R be a relation. A partial function f is a *partial polymorphism* of R if $f(t_1, \ldots, t_n) \in R$ for every $t_1, \ldots, t_n \in R$ such that $f(t_1, \ldots, t_n)$ is defined for each componentwise application.

Example Recall that the function $\mathrm{neg}(x) = 1 - x$ was not a polymorphism of $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$. Define the partial unary function neg' as $\mathrm{neg}'(0) = 1$ and let it be undefined for 1. Then neg' is a partial polymorphism of $R_{1/3}$ since it will always be undefined on any application of a tuple from $R_{1/3}$.

A More Refined Approach

- Let $\mathrm{pPol}(S)$ be the set of all partial polymorphisms of a constraint language S.

A More Refined Approach

- Let $\mathrm{pPol}(S)$ be the set of all partial polymorphisms of a constraint language S.
- Sets of the form $\mathrm{pPol}(S)$ are known as *strong partial clones*.

A More Refined Approach

- Let $\mathrm{pPol}(S)$ be the set of all partial polymorphisms of a constraint language S.
- Sets of the form $\mathrm{pPol}(S)$ are known as *strong partial clones*.
- Strong partial clones are sets of partial functions closed under functional composition and containing all subfunctions.

A More Refined Approach

The partial polymorphisms determines the complexity of CSP problems up to $O(c^n)$ time complexity.

Theorem (Jonsson et al.)

Let S and S' be two finite constraint languages. If $\mathrm{pPol}(S) \subseteq \mathrm{pPol}(S')$ *and CSP*(S) *is solvable in* $O(c^n)$ *time, then CSP*(S') *is also solvable in* $O(c^n)$ *time.*

A More Refined Approach

The lattice of Boolean strong partial clones is uncountably infinite. Even worse:

Theorem (Schölzel) *Assume P $\neq$ NP. Then the set* $\{\mathrm{pPol}(S) \mid \mathsf{SAT}(S)$ *is NP-complete*$\}$ *is uncountably infinite.*

A More Refined Approach

The lattice of Boolean strong partial clones is uncountably infinite. Even worse:

Theorem (Schölzel) *Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathsf{SAT}(S)$ is NP-complete$\}$ is uncountably infinite.*

Theorem (Lagerkvist & Roy) *Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathrm{pPol}(S) \supset \mathrm{pPol}(\{R_{1/3}\})\}$ is (at least) countably infinite.*

A More Refined Approach

The lattice of Boolean strong partial clones is uncountably infinite. Even worse:

Theorem (Schölzel) *Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathsf{SAT}(S)$ is NP-complete$\}$ is uncountably infinite.*

Theorem (Lagerkvist & Roy) *Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathrm{pPol}(S) \supset \mathrm{pPol}(\{R_{1/3}\})\}$ is (at least) countably infinite.*

Theorem (Lagerkvist & Wahlström) *Let $\mathrm{Pol}(S)$ be an essentially unary clone over a finite domain. If S is finite then $\mathrm{pPol}(S)$ does not have a finite base.*

A More Refined Approach

The lattice of Boolean strong partial clones is uncountably infinite. Even worse:

Theorem (Schölzel)

Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathrm{SAT}(S)$ is NP-complete$\}$ is uncountably infinite.

Theorem (Lagerkvist & Roy)

Assume $P \neq NP$. Then the set $\{\mathrm{pPol}(S) \mid \mathrm{pPol}(S) \supset \mathrm{pPol}(\{R_{1/3}\})\}$ is (at least) countably infinite.

Theorem (Lagerkvist & Wahlström)

Let $\mathrm{Pol}(S)$ be an essentially unary clone over a finite domain. If S is finite then $\mathrm{pPol}(S)$ does not have a finite base.

Implies that reasoning with partial polymorphisms is almost always difficult.

Two Non-Trivial Applications

The "easiest NP-complete SAT(S) problem".

Theorem (Jonsson et al.)

Assume $P \neq NP$. Then there exists a relation R such that SAT($\{R\}$) *is NP-complete but not strictly harder than any other NP-complete* SAT(S) *problem.*

Two Non-Trivial Applications

The "easiest NP-complete SAT(S) problem".

Theorem (Jonsson et al.) *Assume $P \neq NP$. Then there exists a relation R such that* SAT($\{R\}$) *is NP-complete but not strictly harder than any other NP-complete* SAT(S) *problem.*

Proof. Proof sketch:

If $\mathrm{pPol}(S) \subseteq \mathrm{pPol}(S')$ then SAT(S') is not computationally harder than SAT(S).

■

Two Non-Trivial Applications

The "easiest NP-complete SAT(S) problem".

Theorem (Jonsson et al.) *Assume $P \neq NP$. Then there exists a relation R such that* SAT($\{R\}$) *is NP-complete but not strictly harder than any other NP-complete* SAT(S) *problem.*

Proof. Proof sketch:

- If $\mathrm{pPol}(S) \subseteq \mathrm{pPol}(S')$ then SAT(S') is not computationally harder than SAT(S).
- It is possible to find a relation R such that $\mathrm{pPol}(S) \subseteq \mathrm{pPol}(\{R\})$ for any S such that SAT(S) is NP-complete.

□

Two Non-Trivial Applications

- A related problem to studying worst-case time complexity is *kernelization*.

Two Non-Trivial Applications

- A related problem to studying worst-case time complexity is *kernelization*.
- It can be seen as a preprocessing technique for reducing a problem to a smaller version of the problem, a *kernel*.

Two Non-Trivial Applications

- A related problem to studying worst-case time complexity is *kernelization*.
- It can be seen as a preprocessing technique for reducing a problem to a smaller version of the problem, a *kernel*.
- For SAT(S) we measure the size of the kernel with respect to the number of constraints.

Two Non-Trivial Applications

- A related problem to studying worst-case time complexity is *kernelization*.
- It can be seen as a preprocessing technique for reducing a problem to a smaller version of the problem, a *kernel*.
- For SAT(S) we measure the size of the kernel with respect to the number of constraints.
- Polymorphisms doesn't work for studying kernelizability of SAT(S) problems.

Two Non-Trivial Applications

Theorem (Lagerkvist & Wahlström)

SAT(S) *has a kernel with* $O(n)$ *constraints if* S *is "embeddable" into a language* $\hat{S}$ *preserved by a Maltsev polymorphism.*

Two Non-Trivial Applications

Theorem (Lagerkvist & Wahlström)

SAT(S) *has a kernel with* $O(n)$ *constraints if* S *is "embeddable" into a language* $\hat{S}$ *preserved by a Maltsev polymorphism.*

Proof.

- Translate instance I of SAT(S) to instance of SAT($\hat{S}$).

Two Non-Trivial Applications

Theorem (Lagerkvist & Wahlström)

SAT(S) *has a kernel with* $O(n)$ *constraints if* S *is "embeddable" into a language* $\hat{S}$ *preserved by a Maltsev polymorphism.*

Proof.

- Translate instance I of SAT(S) to instance of SAT($\hat{S}$).
- Use a variation of the simple algorithm for Maltsev constraints to remove redundant constraints.

Two Non-Trivial Applications

Theorem (Lagerkvist & Wahlström) SAT(S) *has a kernel with* $O(n)$ *constraints if* S *is "embeddable" into a language* $\hat{S}$ *preserved by a Maltsev polymorphism.*

Proof.

- Translate instance I of SAT(S) to instance of SAT($\hat{S}$).
- Use a variation of the simple algorithm for Maltsev constraints to remove redundant constraints.
- Reduce back to SAT(S).

□

Two Non-Trivial Applications

If S is not "embeddable" into a language preserved by a Maltsev polymorphism then this can be witnessed by certain Boolean *partial Maltsev polymorphisms*.

Theorem (Lagerkvist & Wahlström) *If S is* not *preserved by a partial Maltsev operation then* SAT(S) *does not have a kernel with $O(n^{2-\varepsilon})$ constraints for any $\varepsilon > 0$.*

Two Non-Trivial Applications

If S is not "embeddable" into a language preserved by a Maltsev polymorphism then this can be witnessed by certain Boolean *partial Maltsev polymorphisms*.

Theorem (Lagerkvist & Wahlström)

If S is not *preserved by a partial Maltsev operation then* SAT(S) *does not have a kernel with $O(n^{2-\varepsilon})$ constraints for any $\varepsilon > 0$.*

Hence, the absence of partial polymorphisms provides a lot of structural information for a SAT problem.

Concluding Remarks

- To study the worst-case time complexity of CSP problems we used partial polymorphisms instead of total polymorphisms.

Concluding Remarks

- To study the worst-case time complexity of CSP problems we used partial polymorphisms instead of total polymorphisms.
- The resulting theory is much more complicated.

Concluding Remarks

- To study the worst-case time complexity of CSP problems we used partial polymorphisms instead of total polymorphisms.
- The resulting theory is much more complicated.
- But non-trivial results can still be obtained.