The Community Structure of Constraint Satisfaction Problems and Its Correlation with Search Time

Medema, Michel; Lazovik, Alexander

Published in:
2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)

DOI:
10.1109/ICTAI50040.2020.00034

The Community Structure of Constraint Satisfaction Problems and Its Correlation with Search Time

1st Michel Medema
Bernoulli Institute
University of Groningen
Groningen, The Netherlands
m.medema@rug.nl

2nd Alexander Lazovik
Bernoulli Institute
University of Groningen
Groningen, The Netherlands
a.lazovik@rug.nl

***Abstract*—Constraint satisfaction problems are, in general, NP-complete problems, meaning that the computational complexity increases exponentially with the size of the problem in the worst case, under the assumption that P does not equal NP. The structure of a problem heavily influences its computational complexity, however, and problems with a restricted structure constitute one of the general classes of tractable problems. This paper explores the community structure of constraint satisfaction problems, a type of structure already found to be important for SAT problems that is inherent to certain real-world domains. The community structure of the instances of the MiniZinc Challenge of 2019 was identified, and its correlation with the search times of four state-of-the-art solvers as well as with the tree-width of the instances was analysed. The results reveal the strong community structure of many of the instances, although the strength of the community structure seems to only marginally affect the search times. On the other hand, a strong correlation between the community structure and the tree-width is observed, where stronger community structure suggests better decomposability. Taking community structure into account more explicitly during the search process may, therefore, allow constraints solvers to solve problems with strong community structure more efficiently.**

***Index Terms*—Constraint satisfaction problems, constraint optimisation problems, community structure, modularity**

I. Introduction

Constraint Satisfaction Problems (CSPs) are a generic type of search problem that can model numerous types of problems from a myriad of different domains [1]. A CSP consists of a set of variables, with corresponding domains, and a set of constraints that impose restrictions on the values that can be assigned to the variables. The goal is to find assignments to the variables from their respective domains such that all the constraints are satisfied or to find that no such solution exists.

Finding a solution to a CSP is, in general, an NP-complete problem, meaning that, under the assumption that P $\neq$ NP, the time required to find a solution to a problem grows exponentially with the size of the problem in the worst case [2]. The structure of a problem, formed by the interactions between the constraint scopes, has a strong influence on its complexity, however. Problems with a restricted structure comprise one of the general classes of tractable problems [3], and polynomial-time algorithms, for example, exist for CSPs that are tree-structured [2]. Decomposition methods generalise the notion of tree-structure, resulting in another class of tractable problems consisting of problems with low tree-width [4]. More generally, a particular structure oftentimes allows a solution to be found considerably faster than the worst-case exponential complexity, as is frequently the case for real-world problems.

Community structure, which is an important property that often appears in real-world networks, may have similar potential, as has already been demonstrated for SAT problems [5]. A CSP with a strong community structure consists of small groups of strongly interconnected variables, where, except for a small number of connections, these communities could be largely considered as independent problems. For example, for building automation systems or smart energy systems, the communities may correspond to the physical structure of the domain, such as the rooms of a building, where the communities provide a natural boundary for the constraints.

In this paper, the importance of community structure is established by analysing the community structure of a set of CSP instances. In addition to that, it investigates what effect the community structure has on the performance of several state-of-the-art constraint solvers, and whether the strength of the community structure, possibly together with other features of a particular problem, serves in any way as a meaningful indicator of the time it takes a constraint solver to find a solution to a problem. The community structure of a problem instance is also compared to its tree-width to determine to what extent these two measures coincide.

The paper is structured as follows. Section II discusses related work that has investigated community structure in other contexts. Section III provides details regarding community detection for CSPs, and the set of problems that is analysed is presented in section IV. Sections V, VI and VII present the modularity results, the correlation with the search time and the correlation with tree-width, respectively. Finally, the paper is concluded, and directions for future work are provided.

II. Related Work

Problems with strong community structure frequently appear across multiple domains [6], [7]. Ansótegui et al. computed the modularity of the SAT instances of the SAT Race Finals of 2010, a set of problems with industrial relevance, revealing the strong community structure of many of these instances [6]. For several classes of random instances of Model

2375-0197/20/$31.00 ©2020 IEEE
DOI 10.1109/ICTAI50040.2020.00034

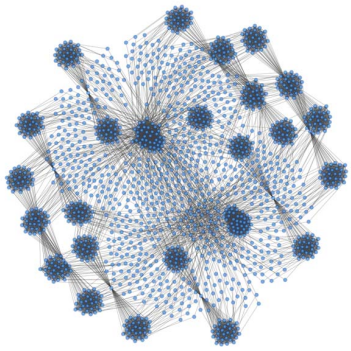

Fig. 1. An example of a graph with community structure.

RB, Li et al. have shown that the instances exhibit fairly strong community structure, although the modularity decreases as the number of constraints, the arity of the constraints and, to some extent, the number of variables increases [7]. Many real-world CSP instances are not random problems, however, and, like SAT instances where the community structure of random instances is considerably lower, these results may not directly apply to other classes of problems [6]. This work considers a broad set of CSP instances, including problems with industrial relevance. Besides identifying the community structure of these problems, it also investigates the correlation between the community structure and the duration of the search process.

Besides the existence of community structure, Newsham et al. also established a correlation between the community structure and the search time of conflict-driven clause-learning SAT solvers [8]. The authors show that the Literal Block Distance, a measure that indicates the importance of a clause learnt during the search process, strongly correlates with the number of communities for many instances. Furthermore, through the use of linear regression, they showed that the modularity is an important feature in the regression model for predicting the search time of SAT solvers. The number of communities alone is also highly influential, more so than the number of variables and clauses.

Based on the concept of community structure, Ganian and Szeider defined the notion of h-modularity, a measure that is similar to tree-width, and introduced an algorithm for which SAT problems are fixed-parameter tractable when parameterised by the h-modularity [5]. This algorithm makes explicit use of the existence of community structure in SAT formulas, making it possible to solve SAT instances with small h-modularity even if the values of other parameters would normally render the instance unsolvable.

III. Community Detection for CSPs

A network or graph has community structure if the set of vertices can be partitioned such that the vertices within each partition are densely connected and the vertices of different partitions sparsely connected [9]. An example of a graph with community structure is shown in Figure 1.

The modularity is a frequently used measure that expresses the quality of a partitioning of the vertices of a graph, thereby providing information regarding the existence of community structure [10], [11]. For a given partition, it measures the fraction of edges within each community compared to the fraction of edges between the communities [10]. The value of the modularity ranges between -1.0 and 1.0, where a value closer to 1.0 signifies a better partitioning for which the fraction of edges within the communities is higher than between the communities. With this measure, the problem of identifying the communities reduces to finding the partitioning that maximises the modularity.

Detecting the community structure of a CSP is done through a graph representation of the problem. A CSP $\mathcal{P} = \langle X, D, C \rangle$ consists of a set of variables $X = \{x_1, \ldots, x_n\}$, the domains of the variables $D = \{D_1, \ldots, D_n\}$ and a set of constraints $C = \{c_1, \ldots, c_m\}$, where a constraint $c_i = \langle r_i, S_i \rangle$ is a tuple with r_i the constraint relation, which defines which combinations of values are allowed, and $S_i \subseteq X$ the scope of the constraint. Several graphical models exist, each capturing different aspects of the problem. To determine to what extent the representation influences the community structure, three commonly used graphical models are considered in this work: the incidence graph, the primal constraint graph and the dual constraint graph. Including multiple graphical models also benefits the analysis, because certain constraint solving techniques inherently use a particular representation, as is, for example, the case for Tree-Decomposition [12].

The incidence graph is a bipartite graph where the sets of vertices represent the variables and constraints, respectively. An edge connects a variable to a constraint if the scope of the constraint includes that variable. This representation captures the full structure of the problem, similar to the constraint hypergraph.

Definition 1 (Incidence Graph Representation [13])**.** The incidence graph of a CSP is a bipartite graph $G = \langle U, V, E \rangle$ where the sets of vertices U and V correspond to the variables X and the constraints C, respectively, and the edges $E = \{(x_i, c_j) \mid x_i \in X, c_j \in C \wedge x_i \in S_j\}$ connect a variable and a constraint if the variable is a member of the scope of the constraint.

The primal constraint graph includes a vertex for each of the variables of the problem, and an edge connects two variables if there exists a constraint scope of which both variables are a member. This representation does not retain all information about the problem, as a clique in the graph can, for example, correspond to both a single constraint scope or multiple overlapping constraint scopes.

Definition 2 (Primal Constraint Graph [14])**.** The primal constraint graph of a CSP is an undirected graph $G = \langle V, E \rangle$ with the set of variables X as its vertices and the set of edges $E = \{(x_i, x_j) \mid x_i, x_j \in X \wedge i < j \wedge \exists c_k \in C : x_i \in S_k \wedge x_j \in S_k\}$ connecting two variables if both appear in the scope of the same constraint.

A similar graphical model that focuses on the constraints instead of the variables is the dual constraint graph, where the vertices represent the constraint scopes, and an edge exists between two constraint scopes if they have at least one variable in common.

Definition 3 (Dual Constraint Graph [3])**.** The dual graph of a CSP is an undirected graph $G = \langle V, E \rangle$ where the set of vertices V corresponds to the constraint scopes and the set of edges $E = \{(S_i, S_j) \mid c_i, c_j \in C \wedge S_i \cap S_j \neq \emptyset\}$ contains an edge between each pair of constraint scopes S_i, S_j that share at least one variable.

Finding the partitioning of the vertices of a graph that maximises the modularity is, unfortunately, an NP-hard problem, making it unsuitable for all but the smallest graphs [9]. Heuristic-based algorithms generally manage to find good partitions, however, and also work for larger graphs. In this work, the Louvain algorithm, which finds good partitions and has a relatively small computational complexity, is used to analyse the community structure of the primal and dual constraint graphs [10]. For the incidence graph, the BiLouvain algorithm is used, because the Louvain algorithm only works for unipartite graphs [15], and while it is possible to represent the incidence graph as a unipartite graph, this conversion could result in a loss of information.

IV. Analysing MiniZinc Instances

The analysis focuses on the community structure of the CSP instances of the MiniZinc Challenge of 2019 [16]. This set comprises twenty classes of problems, each containing five instances, with both academic and industrial relevance, which should provide a comprehensive evaluation of the community structure of CSPs.

All of the problem instances are MiniZinc specifications that must first be compiled to FlatZinc before it is possible to obtain their graph representations [17]. The result of the compilation step is not necessarily unique, however, as it strongly depends on which constraints are made available to the compiler. The FlatZinc representation often also includes variables for which a constraint uniquely defines its value after reducing the remaining variables in its scope to a single value.

During the search process, a solver may safely ignore these defined variables since the constraints ultimately define their values based on the values of the other variables. The community structure of a graph representation will generally be vastly different depending on whether the defined variables are included, however, and while it is also important to analyse the complete version of the problem to grasp its overall structure, a version that does not include these defined variables may align more closely with the implicit representation that constraint solvers use during the search.

Annotations that the compiler adds to the FlatZinc representation make it possible to distinguish between the different types of primitive variables and identify which variables a constraint defines. Adding this information to the constraints extends the definition of a constraint to $c_i \in C = (r_i, S_i, F_i)$, where $F_i \subseteq X$ is the set of variables that constraint c_i defines (normally at most one). The set of variables can be subdivided into $X = X^S \cup X^D \cup X^R$, with X^S the set of search variables, $X^D = \bigcup_{c_i \in C} F_i$ the set of defined variables and $X^R = X \backslash X^D \backslash X^S$ the remaining variables, whose values are also defined but not necessarily by a single constraint.

Directly removing the defined variables X^D and X^R would alter the dependency relations between the variables and the constraints. Instead, a defined variable is replaced with its defining variables to preserve any transitive dependencies. The defining variables of a variable $x_i \in X$ are the variables that, when reduced to a single value, uniquely define the value of x_i and are given by

$$\Delta(x_i, \hat{C}) = \begin{cases} \bigcup_{c_j \in \phi(x_i, \hat{C})} S_j \setminus \{x_i\} & \text{if } x_i \in X^D \\ \bigcup_{c_j \in \kappa(x_i, \hat{C})} S_j \setminus \{x_i\} & \text{if } x_i \in X^R \\ \{x_i\} & \text{otherwise} \end{cases}$$

with $\phi(x_i, C) = \{c_j \mid c_j \in C \wedge x_i \in F_j\}$ the constraint that defines variable x_i and $\kappa(x_i, C) = \{c_j \mid c_j \in C \wedge x_i \in S_j\}$ the set of constraints that include variable x_i in their scope (these sets of defining variables are not necessarily minimal).

Replacing a defined variable in the scope of a constraint with its defining variables is done using

$$\Psi(x, S, \hat{C}) = \begin{cases} S \setminus \{x\} \cup \Delta(x, \hat{C}) & \text{if } x \in S \\ S & \text{otherwise} \end{cases}$$

The full reduction of a FlatZinc instance is the problem where all the defined variables have been replaced.

Definition 4 (Full Reduction)**.** The full reduction of a CSP is the problem $\mathcal{P}' = \langle X', D', C' \rangle$, with $X' = X^S \setminus X^D$ the set of non-defined search variables, D' the domains of these variables and $C' = \Theta(x_1, \Theta(x_2, ...\Theta(x_n, C)...))$. The function $\Theta(x_i, \hat{C}) = \{(r_j, \Psi(x_i, S_j, \hat{C})) \mid c_j \in \hat{C}\})$ replaces a variable x_i with its defining variables in the scopes of all the constraints in $\hat{C}$.

After obtaining the full reduction, the objective function is also removed. The generally strong dependencies that this function introduces between the variables are not strict: for all the instances considered here, the objective function is a simple linear function that can be decomposed into several smaller objective function.

V. Community Structure of CSPs

In order to identify the community structure of the set of problems of the MiniZinc Challenge of 2019, the MiniZinc models are compiled, using version 2.4.3 of the MiniZinc IDE, for *Choco 4.0.4* [18], *OR-Tools 7.6* [19], *Gecode 6.2.0* [20] and *Chuffed 0.10.4* [21], four state-of-the-art constraint solvers that each support a different set of constraints, making it possible to assess the influence that custom constraints have on the

TABLE I

THE AVERAGE MODULARITY, TOGETHER WITH THE STANDARD DEVIATION IN PARENTHESES, FOR EACH CLASS OF PROBLEM INSTANCES OF THE MINIZINC CHALLENGE OF 2019. THE SECOND COLUMN INDICATES THE TYPE OF GRAPHICAL MODEL, WHERE THE INCIDENCE GRAPH, PRIMAL GRAPH AND DUAL GRAPH HAS BEEN ENCODED AS I, P AND D, RESPECTIVELY. INSTANCES WITH A MODULARITY VALUE OF 0.3 OR HIGHER HAVE BEEN HIGHLIGHTED.

		Choco		OR-Tools		Gecode		Chuffed		Reference	
Class	**G**	**Complete**	**Reduced**	**Complete**	**Reduced**	**Complete**	**Reduced**	**Complete**	**Reduced**	**Complete**	**Reduced**
accap	I	**.34** (.01)	**.36** (.01)	**.34** (.01)	**.36** (.01)	**.34** (.01)	**.36** (.01)	**.39** (.02)	**.44** (.03)	**.39** (.02)	**.44** (.03)
	P	.02 (.02)	.01 (.01)	.02 (.02)	.01 (.01)	.02 (.01)	.01 (.01)	**.60** (.15)	**.35** (.19)	**.59** (.15)	**.35** (.19)
	D	**.60** (.19)	**.62** (.19)	**.60** (.19)	**.61** (.19)	**.60** (.19)	**.62** (.19)	**.51** (.15)	**.51** (.14)	**.51** (.14)	**.51** (.14)
amaze	I	**.59** (.00)	**.53** (.01)	**.59** (.00)	**.51** (.00)	**.59** (.00)	**.53** (.01)	**.58** (.00)	**.53** (.01)	**.59** (.00)	**.51** (.00)
	P	**.79** (.02)	**.38** (.04)	**.82** (.01)	**.39** (.04)	**.79** (.02)	**.39** (.04)	**.82** (.02)	**.38** (.04)	**.82** (.01)	**.39** (.04)
	D	**.72** (.01)	**.49** (.02)	**.79** (.01)	**.57** (.02)	**.72** (.02)	**.49** (.01)	**.79** (.01)	**.57** (.02)	**.79** (.01)	**.57** (.02)
fox-geese-corn	I	**.48** (.01)	.01 (.01)	**.47** (.00)	.01 (.01)	**.46** (.00)	.01 (.01)	**.47** (.00)	.01 (.01)	**.48** (.01)	.01 (.01)
	P	**.72** (.07)	.00 (.00)	**.73** (.07)	.00 (.00)	**.73** (.07)	.00 (.00)	**.72** (.07)	.00 (.00)	**.73** (.06)	.00 (.00)
	D	**.73** (.09)	.27 (.05)	**.73** (.10)	.27 (.05)	**.73** (.09)	.27 (.05)	**.73** (.10)	.27 (.05)	**.73** (.10)	.27 (.05)
groupsplitter	I	**.52** (.06)	.29 (.36)	**.54** (.04)	.29 (.35)	**.52** (.06)	.29 (.36)	**.51** (.02)	.29 (.36)	**.53** (.03)	.01 (.01)
	P	**.47** (.06)	.20 (.25)	**.52** (.11)	.20 (.25)	**.47** (.06)	.20 (.25)	**.51** (.10)	.20 (.25)	**.57** (.11)	.20 (.14)
	D	**.64** (.08)	**.31** (.06)	**.64** (.08)	**.34** (.11)	**.64** (.08)	**.31** (.06)	**.62** (.07)	**.34** (.11)	**.74** (.10)	.15 (.16)
hrc	I	**.52** (.12)	.03 (.01)	**.50** (.12)	.03 (.01)	**.49** (.13)	.08 (.01)	**.52** (.11)	.03 (.01)	**.53** (.11)	.03 (.01)
	P	**.44** (.07)	.00 (.00)	**.44** (.07)	.00 (.00)	**.44** (.07)	.00 (.00)	**.44** (.07)	.00 (.00)	**.44** (.07)	.00 (.00)
	D	**.87** (.04)	.03 (.01)	**.87** (.04)	.03 (.01)	**.86** (.04)	.03 (.01)	**.88** (.04)	.03 (.01)	**.87** (.04)	.03 (.01)
kidney-exchange	I	.18 (.01)	.01 (.00)	**.51** (.00)	.00 (.00)	.22 (.01)	.01 (.00)	**.51** (.00)	.00 (.00)	**.51** (.00)	.01 (.00)
	P	**.33** (.00)	.00 (.00)	**.87** (.01)	.00 (.00)	.29 (.00)	.00 (.00)	**.87** (.01)	.00 (.00)	**.87** (.01)	.00 (.00)
	D	.12 (.01)	.00 (.00)	**.32** (.00)	.00 (.00)	.11 (.01)	.00 (.00)	**.32** (.00)	.00 (.00)	**.41** (.00)	.29 (.01)
liner-sf-repositioning	I	**.49** (.02)	**.38** (.07)	**.48** (.04)	**.41** (.03)	**.50** (.03)	**.40** (.03)	**.47** (.03)	**.38** (.05)	**.46** (.02)	**.39** (.04)
	P	**.48** (.08)	.18 (.04)	**.47** (.10)	.18 (.04)	**.46** (.09)	.18 (.04)	**.49** (.09)	.18 (.04)	**.49** (.09)	.18 (.04)
	D	**.45** (.20)	.16 (.07)	**.43** (.28)	.16 (.12)	**.41** (.26)	.14 (.10)	**.47** (.23)	.18 (.09)	**.50** (.20)	.20 (.07)
lot-sizing	I	**.37** (.01)	.01 (.00)	**.56** (.03)	.04 (.01)	**.37** (.01)	.01 (.00)	**.50** (.00)	.03 (.01)	**.54** (.00)	.04 (.01)
	P	**.57** (.01)	.09 (.04)	**.86** (.02)	.09 (.04)	**.57** (.01)	.09 (.04)	**.85** (.02)	.09 (.04)	**.86** (.02)	.09 (.04)
	D	.24 (.05)	.00 (.00)	**.51** (.01)	.01 (.02)	.26 (.04)	.00 (.00)	**.47** (.01)	.01 (.02)	**.56** (.01)	.01 (.02)
median-string	I	**.49** (.01)	.00 (.00)	**.43** (.00)	.00 (.00)	**.43** (.00)	.00 (.00)	**.43** (.00)	.00 (.00)	**.49** (.01)	.00 (.00)
	P	**.74** (.03)	.00 (.00)	**.71** (.06)	.00 (.00)	**.71** (.06)	.00 (.00)	**.71** (.06)	.00 (.00)	**.74** (.03)	.00 (.00)
	D	**.92** (.04)	.00 (.00)	**.91** (.04)	.00 (.00)	**.91** (.04)	.00 (.00)	**.91** (.04)	.00 (.00)	**.92** (.04)	.00 (.00)
multi-knapsack	I	.01 (.01)	.00 (.00)	.05 (.05)	.05 (.08)	.05 (.05)	.05 (.08)	.05 (.05)	.05 (.08)	.05 (.05)	.05 (.08)
	P	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)
	D	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)	.00 (.00)
rotating-workforce	I	**.50** (.00)	.01 (.00)	**.56** (.01)	.07 (.07)	**.50** (.00)	.19 (.00)	**.56** (.01)	.16 (.03)	**.58** (.01)	.18 (.00)
	P	.17 (.07)	.00 (.00)	**.57** (.03)	.00 (.00)	.10 (.03)	.00 (.00)	**.60** (.03)	.00 (.00)	**.84** (.03)	.00 (.00)
	D	**.56** (.07)	.21 (.04)	**.69** (.01)	.18 (.01)	**.56** (.08)	.18 (.00)	**.68** (.01)	.19 (.01)	**.77** (.02)	.10 (.00)
stack-cuttingstock	I	**.52** (.00)	**.40** (.04)	**.45** (.01)	**.45** (.01)	**.45** (.01)	**.45** (.01)	**.53** (.00)	**.39** (.03)	**.55** (.00)	**.67** (.04)
	P	**.76** (.05)	.01 (.01)	**.74** (.07)	.01 (.01)	**.75** (.05)	.01 (.01)	**.76** (.05)	.01 (.01)	**.81** (.04)	.01 (.01)
	D	**.76** (.04)	**.35** (.03)	**.75** (.04)	**.36** (.03)	**.74** (.04)	**.36** (.03)	**.76** (.04)	**.35** (.03)	**.85** (.04)	**.34** (.03)
steelmillslab	I	**.56** (.00)	.24 (.00)	**.52** (.00)	.24 (.00)	**.56** (.00)	.24 (.00)	**.52** (.00)	.24 (.00)	**.52** (.00)	.24 (.00)
	P	**.96** (.00)	.00 (.00)	**.97** (.00)	.00 (.00)	**.96** (.00)	.00 (.00)	**.97** (.00)	.00 (.00)	**.97** (.00)	.00 (.00)
	D	**.95** (.00)	**.42** (.00)	**.96** (.00)	**.42** (.00)	**.95** (.00)	**.42** (.00)	**.96** (.00)	**.42** (.00)	**.96** (.00)	**.42** (.00)
stochastic-vrp	I	**.51** (.09)	**.59** (.04)	**.52** (.09)	**.56** (.04)	**.52** (.09)	**.56** (.04)	**.49** (.03)	**.52** (.08)	**.47** (.03)	**.48** (.10)
	P	**.55** (.11)	**.45** (.11)	**.53** (.10)	**.45** (.11)	**.53** (.10)	**.45** (.11)	**.65** (.08)	**.45** (.11)	**.64** (.09)	**.45** (.11)
	D	**.53** (.08)	**.48** (.03)	**.53** (.09)	**.35** (.04)	**.52** (.09)	**.35** (.04)	**.68** (.04)	**.48** (.02)	**.78** (.03)	**.51** (.06)
triangular	I	**.32** (.01)	**.33** (.00)	**.32** (.01)	**.33** (.00)	**.32** (.01)	**.33** (.00)	**.32** (.01)	**.33** (.00)	**.32** (.01)	**.33** (.00)
	P	.00 (.00)	.08 (.01)	.00 (.00)	.08 (.00)	.00 (.00)	.08 (.00)	.00 (.00)	.08 (.00)	.00 (.00)	.08 (.00)
	D	.23 (.01)	.23 (.02)	.23 (.01)	.23 (.02)	.23 (.02)	.24 (.01)	.23 (.02)	.23 (.01)	.23 (.02)	.23 (.01)
zephyrus	I	**.49** (.00)	**.41** (.01)	**.46** (.00)	**.38** (.01)	**.49** (.00)	**.41** (.01)	**.46** (.00)	**.38** (.01)	**.45** (.00)	**.38** (.01)
	P	**.46** (.03)	.00 (.00)	**.67** (.00)	.00 (.00)	**.47** (.01)	.00 (.00)	**.67** (.00)	.00 (.00)	**.67** (.00)	.00 (.00)
	D	**.69** (.01)	**.53** (.02)	**.78** (.01)	**.66** (.01)	**.69** (.01)	**.53** (.02)	**.78** (.00)	**.66** (.01)	**.77** (.01)	**.66** (.01)

community structure. For the same purpose, a reference set is created by compiling all the MiniZinc models using only the default set of constraints included in the standard MiniZinc library. The full reduction is computed for each of the FlatZinc instances produced by the compiler, and for both the original FlatZinc instance and its reduction, the graph representations are obtained, where the community structure of the primal constraint graph and dual constraint graph is analysed using

the Louvain algorithm, and the community structure of the incidence graph is analysed using the BiLouvain algorithm. Four classes have been excluded from the analysis because too much time was required to either compile the model or analyse its graph representation. Specifically, the instances of the ”code-generator”, ”nside”, ”ptv” and ”rcpsp-wet-diverse” classes were excluded, as well as instance ”n37” of the ”triangular” class.

Table I reports the average modularity, together with the standard deviation, per class of problems for each collection of instances, showing both the community structure of that particular class and how much it differs among the different instances of that class. The second column indicates the type of graphical model, where the incidence graphs, primal graphs and dual graphs have been encoded as I, P, and D, respectively. A modularity value of 0.3 or higher signifies a strong community structure; the grey-coloured cells highlight these instances. Moreover, by design, a partitioning where all the vertices reside in their own community or where all vertices are part of the same community have a modularity value of 0.0.

For most classes, at least one of the graphical models exhibits a strong community structure, and often the modularity is high for all three types of graphs; the highest modularity is generally observed for the dual constraint graph. There are also cases where one of the graphical models does not seem to have any community structure, while the community structure of the remaining two graphs is fairly strong. For such occurrences, the community structure of the primal constraint graph is principally nonexistent, which could suggest the inability of this graph representation to capture the structure of the problem in certain cases (using a weighted version of the graph seemed to make little difference). Other classes, such as *multi-knapsack*, do not seem to have any community structure, regardless of the graph representation. This result is expected, as the constraints that are involved in the knapsack problem include all the variables, making a single community, which has a modularity value of 0.0, the only reasonable partitioning.

The modularity of the reduction is almost always lower than the full version. This version only includes the search variables and replaces all the defined variables in the scopes of the constraints, which often results in more densely connected graphs as the scopes include many of the same variables. Occasionally, the modularity of the reduction is slightly higher, something that may happen when there are no defined variables but the removal of the objective function eliminates several connections between variables and constraints. The community structure of this version may be more important for certain techniques, however, as it may be more representative of how those techniques perceive a particular problem.

The standard deviation shows that the modularity is fairly consistent within a particular class of problems, meaning the community structure is either strong or weak for all of the instances. An example of an exception to this is *liner-sf-repositioning*, for which the standard deviation is 0.20 or higher for some of the graph representations. Possibly, this difference can be attributed to the types of constraints that are used in those instances.

Some important differences between the modularity of the different solvers are apparent. Usually, the difference in modularity is negligible; it is either the same or differs by less than 0.05. For example, for *liner-sf-repositioning* the modularity is nearly identical, both between the solvers and between the solvers and the reference set. In other cases, these differences are more substantial, such as for *zephyrus*, where two distinct groups are recognisable. The modularity for *Choco* and *Gecode* are almost identical, as are the modularity values of the remaining solvers and the reference set. The first two seemingly support a constraint, or lack this support, leading to a different representation with, in this particular instance, weaker community structure. The set of constraints supported by a solver conceivably affects the community structure, although the majority of the classes seem to include constraints that are commonly supported by all solvers.

VI. Correlating Community Structure and Search Time

To uncover a potential correlation between the search time and the community structure, the search times of the four solvers are recorded for all of the CSP instances. Each solver solves the solver-specific version as well as the reference version of an instance, both following the search specification and using ”free search”, which allows a solver to disregard the search specification. A time limit of 30 minutes is imposed, and the search time of any instance that was not solved within this limit is set to 30 minutes. All experiments were performed on the HPC cluster of the University of Groningen, where each search process was granted a single core of one of the two Intel Xeon E5 2680v3 CPUs and 3GB of memory.

The results include only the search times of the solver-specific versions for which the solvers followed the search specification; all the solvers solved fewer instances of the reference set, although some instances required less time to solve, and even though the ”free search” resulted in a considerable increase in the number of solved instances for all but *Choco*, the correlations are remarkably similar. For the instances of the ”kidney-exchange” and ”lot-sizing” classes, *OR-Tools* and *Chuffed* raised an exception, caused by the lack of support for the ”indomain_median” heuristic that is included in the search specification. These instances have not been excluded from the experiments, however, because of the limited impact it had on the results.

Correlations are analysed between the community structure, consisting of the modularity, the number of communities and the maximum community size, and several other features of the problems, including the search time, the number of variables, the number of constraints, the domain size and the average and maximum degree (the average and maximum size of the constraint scopes, respectively). The Spearman’s rank correlation coefficient, which can detect more general relations besides linear ones, is used to compute the strength of the relationships between these features.

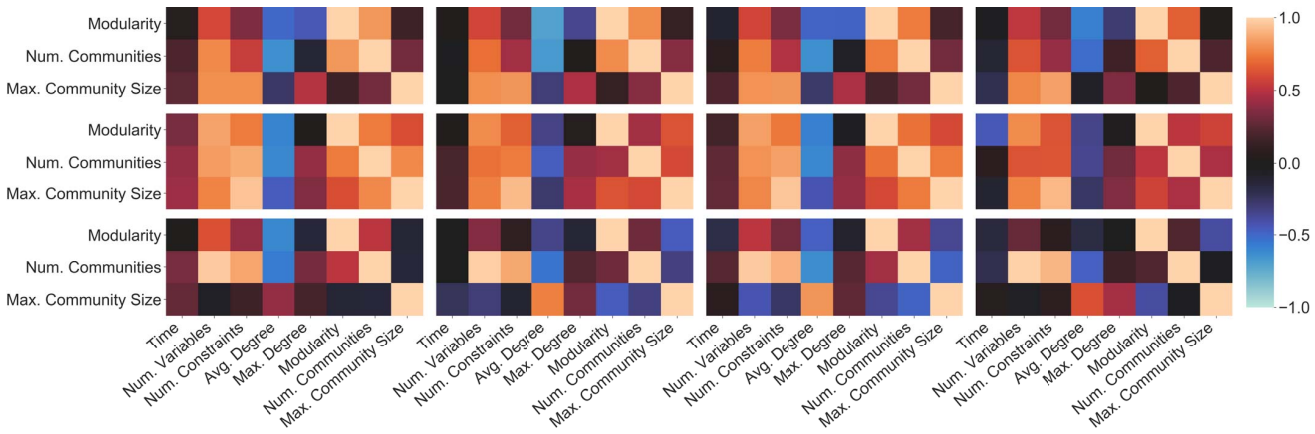

Fig. 2. The Spearman correlation between the community structure and other features of the complete versions of the problem instances. The columns represent *Choco*, *OR-Tools*, *Gecode* and *Chuffed*, respectively, and the rows correspond to the primal graph, dual graph and the incidence graph.

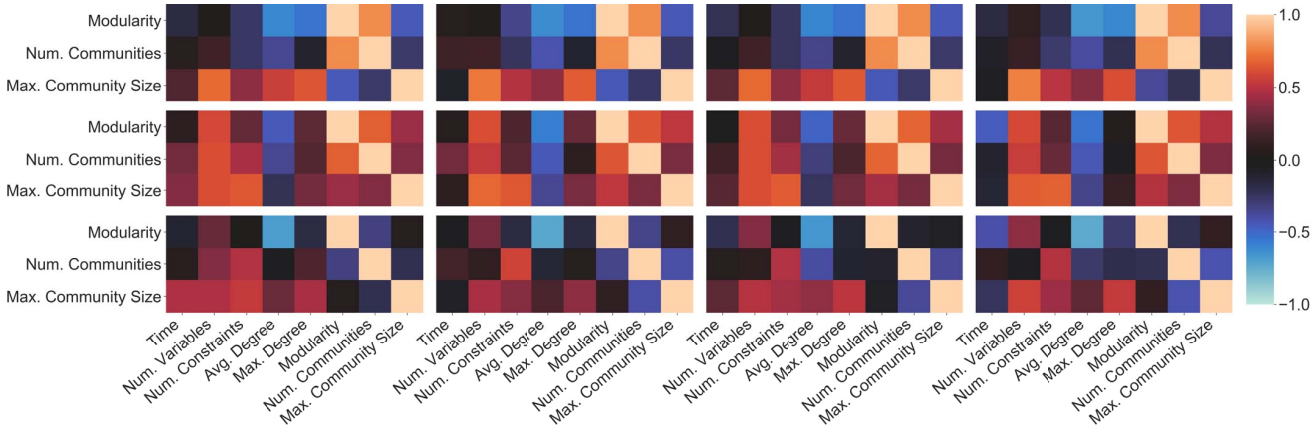

Fig. 3. The Spearman correlation between the community structure and other features of the reduced versions of the problem instances. The columns represent *Choco*, *OR-Tools*, *Gecode* and *Chuffed*, respectively, and the rows correspond to the primal graph, dual graph and the incidence graph.

Figures 2 and 3 show the correlation coefficients of the different combinations of features for the complete problem and the reduction, respectively. In general, the search times are lower for problem instances with higher modularity, whereas the maximum community size and, to some extent, the number of communities, suggest a higher search time; the correlations between these features are often fairly weak, however. Increasing the number of variables or constraints leads to higher modularity, more communities and larger communities, except for some of the reductions, where increasing the number of constraints has the opposite effect. A higher average degree, on the other hand, implies fewer communities that are more strongly connected. The features of the community structure also appear to be strongly correlated, but the actual correlation varies greatly from strongly positive to strongly negative. These observations are highly consistent between the different solvers, and even for the reductions the coefficients largely coincide, although those are closer to zero. Only between the various graphical models, larger differences are discernible, where the strongest correlations are typically reserved for the dual constraint graphs.

Individually, the features that capture the community structure only seem to have a marginal influence on the search time. Given the complex nature of CSPs and the many factors that potentially influence the search time, combining multiple features may provide a better model of the search time. The relevance of these combinations of features has been tested using linear regression. Models are built separately for each of the graphical models of the solvers. The features have been scaled to reduce their skewness and to equalise their magnitude, and a logarithmic transformation has been applied to the search time. All combinations of features are considered, and the best ones, determined using the adjusted R^2 measure, are shown in Table II.

Most models do not accurately predict the search times. The best model, corresponding to the primal constraint graph of the reductions of *Choco*, has an R^2 value of only 0.5. At least one of the features related to the community structure is almost always included in the best models, however, which may be an indication of its importance. Other important features

TABLE II
THE INCLUDED FEATURES AND THE ADJUSTED R^2 VALUES FOR THE BEST LINEAR REGRESSION MODELS FOR EACH COMBINATION OF SOLVER AND GRAPH TYPE, FOR BOTH TYPES OF PROBLEM INSTANCES: THE COMPLETE AND REDUCED VERSION, ENCODED AS C AND R, RESPECTIVELY.

	Graph	Type	R^2	Modularity	Num. Communities	Max. Community Size	Num. Variables	Num. Constraints	Domain Size	Avg. Degree	Max. Degree
Choco	Primal	C	.25			x		x	x		x
		R	.50	x	x	x	x	x	x	x	x
	Dual	C	.25		x	x			x	x	
		R	.45			x		x	x	x	
	Incidence	C	.28	x		x		x	x	x	x
		R	.43			x		x		x	
OR-Tools	Primal	C	.15	x			x	x	x		
		R	.19	x		x	x	x	x	x	x
	Dual	C	.25		x		x	x	x	x	x
		R	.17	x	x			x	x	x	
	Incidence	C	.19	x		x	x	x	x	x	
		R	.21		x	x	x		x	x	x
Chuffed	Primal	C	.13	x	x				x		x
		R	.31	x			x	x		x	x
	Dual	C	.27	x	x		x		x	x	
		R	.45	x		x	x		x	x	x
	Incidence	C	.07						x	x	
		R	.42	x	x	x	x		x	x	x
Gecode	Primal	C	.08	x	x			x		x	
		R	.23	x			x	x		x	
	Dual	C	.08					x		x	x
		R	.22				x	x		x	
	Incidence	C	.10	x				x			x
		R	.23	x			x	x		x	

that are present in most models are the number of variables and constraints, the domain size and the average degree. The models corresponding to the reductions perform better in nearly all cases; a clear distinction between the graphical models is not obvious. The instances that could not be solved within the time limit of 30 minutes have a strong influence on the models, which can be seen in Figure 4 where these create the distinct pattern that is visible in the top right corner. Nonetheless, it appears that one or more important features are missing from these models.

VII. CORRELATION BETWEEN MODULARITY AND TREE-WIDTH

Decomposition techniques, such as Tree-Decomposition, convert a CSP into a tree-like structure where the subtrees represent independent subproblems [22]. The tree-width captures how closely the tree decomposition resembles a tree, thereby providing information about the decomposability of the problem. These decomposition techniques directly manipulate the structure of a problem, which is, at least in part, characterised by the community structure.

To determine the influence of the community structure on the decomposability of a CSP and its relation to the tree-width,

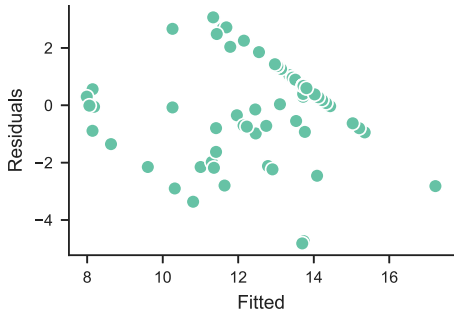

Fig. 4. The residuals versus fitted values for the best linear regression model of *Choco* corresponding to the reduced version of the primal constraint graph.

the instances of the reference set are decomposed using the *Heuristic Tree-Decomposition without Triangulation* algorithm [22]. The tree-width cannot easily be compared directly, however, as it is an absolute measure that partially depends on the size of the problem. Therefore, the ratio between the tree-width and the number of search variables is used, which gives a relative measure of the decomposability of a CSP. Only the full reductions are considered because the tree-decomposition of the full version of a FlatZinc instance virtually always has a tree-width that is higher than the number of search variables. Additionally, as the algorithm performs decomposition directly on the variables, which corresponds to analysing the structure of the primal constraint graph, the community structure of the primal constraint graphs is used (the community structure of the other graphs has been investigated but did not show any interesting relations).

Figure 5 shows the relation between the modularity of the primal constraint graph and the ratio between the tree-width and the number of search variables. The ratio appears to decrease for higher values of the modularity, an indication that a stronger community structure corresponds to a better decomposition, although this trend becomes less apparent for higher values of the modularity. The modularity unmistakably separates the instances into two groups, however, where the ratio is close to 1.0 for instances for which the modularity is close to 0.0, and where the ratio is at most equal to 0.8 otherwise. Community structure seems to have a positive influence on the tree-decomposition, and decomposition techniques may, perhaps implicitly, make use of its existence. Incorporating decomposition techniques into general constraint solvers may allow those solvers to recognise these types of structures, possibly leading to improved performance.

The tree-width also appears to be highly correlated with the maximum degree, the size of the largest constraint scope, as shown in Figure 6. For many instances, the maximum degree and the tree-width have equal values, and the tree-width of all other instances is higher than the maximum degree. The maximum degree seemingly serves as a good indicator of the tree-width, or at least as a lower bound on the tree-width. This correlation even persists after excluding the instances for which the modularity is close to zero or the ratio between the

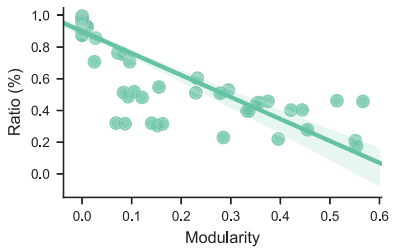

Fig. 5. The ratio between the tree-width and the number of search variables plotted versus the modularity.

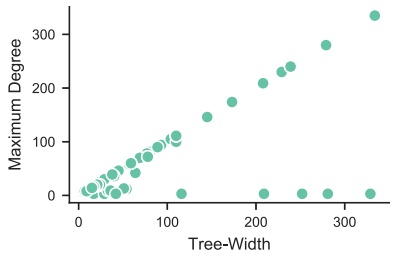

Fig. 6. A visualisation of the correlation between the tree-width and the maximum degree.

tree-width and the number of search variables is close to one. The maximum degree and the maximum community size, not included in this graph, also appear to be strongly correlated, which implies that the correlation between the tree-width and the maximum community size follows a similar trend.

VIII. Conclusion & Future Work

The structure of a CSP is an important characteristic with a potentially strong influence on the computational complexity of a problem. Community structure signifies an inherent partitioning of the variables and constraints, a type of structure that search algorithms may be able to exploit. An analysis of the community structure of the problem instances of the MiniZinc Challenge of 2019 shows that almost all instances exhibit strong community structure, albeit of varying degree, and that instances that belong to the same class have comparable community structure. As an independent feature, the community structure does not appear to be of great importance for explaining the differences in search time, but it has greater significance when combined into a linear model together with other features. For Tree-Decomposition, on the other hand, the modularity is a reasonable predictor of the decomposability of a problem, at least for the full reductions. Additionally, the tree-width shows a strong correlation with the maximum community size and the maximum degree, two features that are also mutually related.

For the solvers included in the experiments, the community structure only appears to have a marginal effect on the search time. Determining to what extent it is possible for these solvers to take advantage of the community structure requires a more comprehensive analysis of the community structure. The community structure could potentially be used to define a new class of tractable problems as well, similar to the one defined for SAT problems. The relation between the tree-width and the community structure should also be investigated further to uncover possible cases where these two measures do not coincide. In all of these cases, being able to generate problem instances with a known community structure may be valuable, as it allows for a more controlled way to examine the behaviour of different algorithms.

References

[1] V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey," *AI Magazine*, vol. 13, p. 32, Mar. 1992.

[2] C. Carbonnel and M. C. Cooper, "Tractability in constraint satisfaction problems: a survey," *Constraints*, vol. 21, pp. 115–144, Apr. 2016.

[3] G. Gottlob, N. Leone, and F. Scarcello, "A comparison of structural CSP decomposition methods," *Artificial Intelligence*, vol. 124, pp. 243–282, Dec. 2000.

[4] P. Jegou, S. Ndiaye, and C. Terrioux, "Computing and exploiting tree-decomposition for (Max-)CSP," Jan. 2005.

[5] R. Ganian and S. Szeider, "Community Structure Inspired Algorithms for SAT and #SAT," in *Theory and Applications of Satisfiability Testing – SAT 2015*, vol. 9340, pp. 223–237, Springer International Publishing, 2015.

[6] C. Ansótegui, J. Giráldez-Cru, and J. Levy, "The Community Structure of SAT Formulas," in *Theory and Applications of Satisfiability Testing – SAT 2012*, vol. 7317, pp. 410–423, Springer Berlin Heidelberg, 2012.

[7] R. Li, S. hu, and J. Wang, "The Community Structure of the Constraint Satisfaction Problem Instances of Model RB," *Journal of Computational and Theoretical Nanoscience*, vol. 12, pp. 6088–6093, Dec. 2015.

[8] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon, "Impact of Community Structure on SAT Solver Performance," in *Theory and Applications of Satisfiability Testing – SAT 2014*, vol. 8561, pp. 252–268, Springer International Publishing, 2014.

[9] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, pp. 75–174, Feb. 2010.

[10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, vol. 2008, p. P10008, Oct. 2008.

[11] M. J. Barber, "Modularity and community detection in bipartite networks," *Phys. Rev. E*, vol. 76, p. 066102, Dec. 2007.

[12] P. Jégou and C. Terrioux, "Hybrid backtracking bounded by tree-decomposition of constraint networks," *Artificial Intelligence*, vol. 146, pp. 43–75, May 2003.

[13] M. C. Cooper and S. Zivny, "Hybrid Tractable Classes of Constraint Problems," in *The Constraint Satisfaction Problem: Complexity and Approximability*, vol. 7 of *Dagstuhl Follow-Ups*, pp. 113–135, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[14] D. Marx, "Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries," *J. ACM*, vol. 60, pp. 42:1–42:51, Nov. 2013.

[15] C. Zhou, L. Feng, and Q. Zhao, "A novel community detection method in bipartite networks," *Physica A: Statistical Mechanics and its Applications*, vol. 492, pp. 1679–1693, Feb. 2018.

[16] P. J. Stuckey, T. Feydy, A. Schutt, G. Tack, and J. Fischer, "The MiniZinc Challenge 2008–2013," *1*, vol. 35, pp. 55–60, June 2014.

[17] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "MiniZinc: Towards a Standard CP Modelling Language," in *Principles and Practice of Constraint Programming – CP 2007*, vol. 4741, pp. 529–543, Springer Berlin Heidelberg, 2007.

[18] N. Jussien, G. Rochart, and X. Lorca, "Choco: an Open Source Java Constraint Programming Library," in *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pp. 1–10, 2008.

[19] "OR-Tools | Google Developers." https://developers.google.com/.

[20] "GECODE - An open, free, efficient constraint solving toolkit."

[21] C. Geoffrey, P. J. Stuckey, A. Schutt, T. Ehlers, G. Gange, and K. Francis, "Chuffed," Apr. 2020. https://github.com/chuffed/chuffed.

[22] P. Jégou, H. Kanso, and C. Terrioux, "An Algorithmic Framework for Decomposing Constraint Networks," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1–8, 2015.