Detecting Highly Overlapping Parts in Constraint Satisfaction Problems For Speed-Up

Sven Löffler, Ke Liu, and Petra Hofstedt

Brandenburg University of Technology Cottbus-Senftenberg, Germany
Sven.Loeffler@b-tu.de

Abstract. This paper describes a new approach on splitting constraint satisfaction problems (CSPs) by means of an auxiliary constraint satisfaction optimization problem (CSOP) that detects sub-CSPs with a potentially high number of conflicts. The purpose of this approach is to find sub-CSPs maximal in size which can be substituted by constraints with a higher consistency level or to be solved in parallel. This, eventually, often allows to significantly speed-up the solution process of a CSP.

Keywords: Constraint Programming · CSP · Refinement · Optimizations

1 Introduction

Constraint programming (CP) is a powerful method to model and solve NP-complete problems in a declarative way. Typical research problems in CP are rostering, graph coloring, optimization and satisfiability (SAT) problems [6].

Since the search space of CSPs is very big and the solution process often needs an extremely high amount of time we are always interested in improvement and optimization of the solution process. In general, a CSP is solved by interleaved backtrack search (often depth-first search) and propagation. If there is a fail in the search tree then a backtrack returns to the last step where another value assignment was possible. This means unnecessary work was done and all value assignments and propagations must be withdrawn by backtracking. With our approach we try to find constraints which are the origin of such fails so that we can remove or separate them (with other algorithms).

In this paper, we propose a way to model an auxiliary constraint satisfaction optimization problem (CSOP) to detect areas of a given CSP with a large number of overlapping constraints that are likely to cause fails during backtrack search. We see this approach as a preparation step for the substitution of sub-CSPs by new CSPs for speeding-up the solution process as described in [4] or for problem splitting (section 2.3.4 in [5]) in parallel constraint solving.

2 Preliminaries

In this section we introduce necessary definitions, methods and theoretical considerations which are the basic of our approach. We consider constraint satis-

faction problems (CSPs), constraint satisfaction optimization problems (CSOP) and sub-CSPs which are defined as follows.

CSP [3] A constraint satisfaction problem (CSP) is defined as a 3-tuple $P = (X, D, C)$ with $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite domains where D_i is the domain of x_i and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of primitive or global constraints covering between one and all variables in X.

CSOP [9, 10] A constraint satisfaction optimization problem (CSOP) $P_{opt} = (X, D, C, f)$ is defined as a CSP with an optimization function f that maps each solution to a numerical value. The task in a CSOP is to find the solution tuple with the optimal value with regard to the optimization function f.

Sub-CSP Let $P = (X, D, C)$ be a CSP. For $C' \subseteq C$ we define $P_{sub} = (X', D', C')$ such that $X' = \bigcup_{c \in C'} vars(c)$ with corresponding domains $D' = \{D_i \mid x_i \in X'\} \subseteq D$.

2.1 Definitions and Methods

For the further concepts and methods we assume that a CSP $P = (X, D, C)$ is given.

In this paper, it is important to distinguish between local [2] and global consistency [3]. Local consistency guarantees that each value of a variable in the scope of a certain constraint is at least part of one of its solutions. A CSP is locally consistent if all of its constraints are locally consistent. In contrast, global consistency implies that each value of the variable of the CSP can be extended to at least one solution of the entire CSP. Therefore, global consistency is a much stronger enforcement of consistency. In particular, search interleaved with global consistency is backtrack free.

In our description we use the two functions $vars(c)$ and $cons(x)$, where the method $vars$ has a constraint $c \in C$ as input and returns all variables $X' \subseteq X$ which are covered by this constraint. Analogously, the method $cons$ has a variable $x \in X$ as input and returns all constraints $C' \subseteq C$ which cover this variable.

We say two constraints $c_i, c_j \in C, i \neq j$ *overlap* if the intersection of variables $vars(c_i)$ and $vars(c_j)$ is not empty.

We define the maximal size $size(P)$ of a CSP $P = (X, D, C)$ as the product of all cardinalities of the domains of the CSP P.

$$size(P) = \prod_{i=1}^{|X|} |D_i| \tag{1}$$

The size of a variable and of a constraint can be defined similar, i.e. $size(x_i) = |D_i|$ and $size(c) = \prod_{x_i \in vars(c)} |D_i|$.

2.2 Theoretical Consideration

For a given CSP P, we can separate the propagators of the constraints into two sets: the one set ensure local consistency for the constraints and the other one not (e.g. bound consistency). Examples for both categories based on their implementation in Choco Solver [7] are for locally consistent constraints: *arithm*, *count* or *regular* and for not locally consistent constraints: *cumulative* or *sum*.

Consider a CSP P with only one constraint c_1, which has a propagator that ensures local consistency, this implies that P must be backtracking free. The question is what leads to a not backtracking free CSP? If we add another constraint c_2, which has a propagator that ensures local consistency, to P then there are two possibilities.

Case 1: The two constraints c_1 and c_2 do not overlap. It is clear that this cannot lead to a fail because we can decompose such a CSP into to two separate CSPs P_1 and P_2, solving them individually and merge the results together. Hence, each solution of P_1 contains no value assignment for a variable of P_2 and vice versa, every element of the cross product of the solutions of P_1 and P_2 is a solution of P.

Case 2: The two constraints c_1 and c_2 overlap. Depending on different things such as the types of constraints c_1 and c_2, search strategy, propagation order, variable order etc. the CSP has a fail or not.

Thus, the origin for fails in a CSP P with only locally consistent constraints are overlapping constraints. This leads to the consideration that areas of the CSP P with a high number of overlapping constraints have the potential to be responsible for a large number of fails.

3 The Detection of Highly Overlapping Sub-CSPs

In this section, we present an approach to detect highly overlapping areas in a CSP P using a CSOP P_{opt}.

3.1 A CSOP to Find Maximal Overlapping Sub-CSPs

The idea behind this approach is to find sub-CSPs in P which are responsible for a large number of fails inside the backtrack search and small enough a) to solve them in parallel or b) to replace them by another CSP with less conflicts inside. We assume that a sub-CSP with a large number of overlapping constraints is likely to incur fails during backtrack search.

Solving the following CSOP P_{opt} leads to a sub-CSP P_{sub} of $P = (X, D, C)$ with $size(P_{sub})$ smaller or equal to a given value $subCspSize$ and a maximum number of overlapping variables.

Let $P_{opt} = (X', D', C', f)$ be a CSOP with:

$$X' = \{x_{ds,i}, x_{nc,i}, x_{ac,j} \mid \forall i \in \{1, ..., |X|\}, j \in \{1, ..., |C|\}\},$$
$$D' = \{D_{ds,i}, D_{nc,i}, D_{ac,j} \mid \forall i \in \{1, ..., |X|\}, j \in \{1, ..., |C|\},$$
$$D_{ds,i} = \{1, |D_i|\},\ D_{nc,i} = \{0, ..., |cons(x_i)|\}, D_{ac,j} = \{0, 1\}\}$$

$C' = C_1 \cup C_2 \cup C_3 \cup C_4$ where
$C_1 = \{(x_{ds,i} = 1 \iff x_{nc,i} = 0) \mid \forall i \in \{1, ..., |X|\}\}$
$C_2 = \{(x_{nc,i} = \sum_{\forall j \in \{1,...,|C|\}, c_j \in cons(x_i)} x_{ac,j}) \mid \forall i \in \{1, ..., |X|\}\}$
$C_3 = \{(x_{ac,j} = 1 \iff x_{ds,i} = |D_i|) \mid \forall i \in \{1, ..., |X|\}, j \in \{1, ..., |C|\}$ where $x_i \in vars(c_j)\}$
$C_4 = \{(\prod_{i=1}^{|X|} x_{ds,i} \leq subCspSize)\}$
$\max f = (\sum_{i=1}^{|X|} x_{nc,i}) - |\{x \mid x \in \{x_{nc,1}, ..., x_{nc,|x|}\}, x \neq 0\}|$

There are three types of variables in P_{opt}. For each variable $x_i \in X$, two variables $x_{ds,i}$ and $x_{nc,i}$ are created and for each constraint $c_j \in C$ a binary variable $x_{ac,j}$ is created. The abbreviations ds, nc and ac are only names and stay for domain size, number of constraints and active constraint.

If the variable x_i is part of the sub-CSP then the size of the sub-CSP is multiplied with the domain size of x_i, otherwise the size of P_{sub} is unchanged. Thus, the domain $D_{ds,i}$ of variable $x_{ds,i}$ contains the values 1 and $|D_i|$ (1 as a neutral element of the multiplication). The product of all $x_{ds,i}, i \in \{1, ..., |X|\}$ is then the size of P_{sub}. The constraint in C_4 guarantees that the resulting sub-CSP P_{sub} has a size $size(P_{sub})$ smaller or equal to a given value $subCspSize$.

The variable $x_{nc,i}$ represents the number of constraints which are part of the sub-CSP P_{sub} and cover the corresponding variable x_i. Hence the possible values of $x_{nc,i}$ are between 0 and the number of constraints over x_i. Only variables $x \in X$ which are covered by a constraint of P_{sub} are part of the sub-CSP. The constraints in C_1 ensure this. If the number of constraints of a variable x_i in P_{sub} is null ($x_{nc,i} = 0$) then it has no influence on the size of P_{sub} (so $x_{ds,i}$ must be the neutral element of the multiplication) and vice versa.

Each variable $x_{ac,j}$ corresponds to a constraint $c_j \in C$ in a way, that if the constraint c_j is part of P_{sub} then the value of $x_{ac,j}$ is 1, else it is 0. From the descriptions of the variables $x_{nc,i}$ and $x_{ac,j}$ follows that the sum of all $x_{ac,j}$ where $x_{ac,j}$ corresponds to a constraint c_j which is part of P_{sub} and covers x_i must be equal to $x_{nc,i}$, which is realized by the constraints in C_2.

The constraints in C_3 guarantee that if a constraint $c_j \in C$ is part of P_{sub} (corresponding to $x_{ac,j} = 1$) then also all variables which are covered by this constraint ($x_i \in vars(c_j)$, corresponding to $x_{ds,i} = |D_i|$) must be part of the sub-CSP and vice versa.

The goal of P_{opt} is to find the sub-CSP with the most overlaps. This is expressed by the objective function f. The variable $x_{nc,i} \forall i \in \{1, ..., |x|\}$ represents the number of constraints of the variable $x_i \in X$ in P_{sub}. The number of overlapping constraints for x_i in P_{sub} is the value of $x_{nc,i}$ reduced by one if the value is greater than null or null otherwise. To count the number of overlaps correctly we must subtract one for each variable $x_{nc,i}$ which is not null. The objective function f is modeled exact like this.

A solution of P_{opt} can be transformed into a sub-CSP in the following way. All variables $x_i \in X$, where the corresponding variable $x_{ds,i}$ is not set to value 1 are element of the sub-CSP. Analogously all constraints $c_j \in C$ are part of the sub-CSP, where the corresponding variable $c_{ac,j}$ is set to 1.

3.2 Remarks on the CSOP

Using our approach to solve the CSP P, we do the following: 1) generating and solving CSOP P_{opt} to find highly overlapping sub-CSPs, 2) replacing there by equivalent sub-CSPs, which can be solved more efficiently (e.g. with less fails in the search), 3) solving the newly generated CSP P'. Thus, solving the COSP P_{opt} should not be too complex or time consuming computed to the original CSP P. Consider the following remarks:

1. The depth of the search tree can be much smaller then the number of variables of P_{opt}. The reason for this is that after setting all binary variables $c_{ac,j}|\forall j \in \{1, ..., |C|\}$ to a value (1 or 0) and propagating them, all other variables are also instantiated. Each variable $x_{nc,i}$ is instantiated by the constraints of C_2 and each variable $x_{ds,i}$ is then instantiated by the constraints in C_1. So it makes sense to use a search strategy which searches only variables in the set of $c_{ac,j}|\forall j \in \{1, ..., |C|\}$.
2. A first possible (not perfect) solution of P_{opt} can be given directly by setting all $c_{ac,j}|\forall j \in \{1, ..., |C|\}$ to null. While the size of the sub-CSP is smaller or equal $subCspSize$ one constraint after the other can be added to P_{sub}. This solution can be used as a starting point for local search.
3. Portfolios like explained in [1] can be used which often leads to good speed-ups in optimization problems.
4. It is not necessary to find a perfect solution. Often a very good solution can be found in a fraction of the time needed to find the best solution.
5. For solving sub-CSPs in parallel more than one sub-CSP is necessary. For this we can extend our CSOP in a way that we add for each sub-CSP we want to find again a set of variables $X'' = \{x_{ds,i}, x_{nc,i}, x_{ac,j} \mid \forall i \in \{1, ..., |X|\}, j \in \{1, ..., |C|\}\}$ with domains D'' and constraints C'' analogously to D' and C'. Then we only need to add *count* constraints over the variables $x_{ac,j}$ of all sub-CSPs which only allows one value one in all variables $x_{ac,j}$ with index j from the different sub-CSPs. Such constraints guarantee that each constraint of the given CSP P is only associated to one sub-CSP. The new objective function is then the sum of the objective functions of all sub-CSPs.

Currently, the approach is specialized to find sub-CSPs with a high density of overlapping constraints which are responsible for a high number of fails in the search tree. This is especially useful for the substitution of the constraints of a sub-CSP with only one constraint which has a propagator that ensure local consistency like the `table` or `regular` constraint. Doing this leads to a high reduction of fails in the search tree of the original CSP.

A second approach is to use the detected sub-CSPs as work load balancing decompositions for parallel consistency [8] techniques. Parallel consistency enforcement encloses the problem of combining partial solutions of sub-CSPs into one solution of the complete CSP. Using our approach to find sub-CSPs with a high number of overlapping constraints inside each sub-CSP but with a low number of constraints between different sub-CSPs can be a promising approach. There we could apply a different objective function, which searches for the lowest number of overlaps between different sub-CSPs instead.

4 Conclusion

Summary and Conclusion. We presented a new approach for the decomposition of CSPs into sub-CSPs using a CSOP. It was presented how a CSOP can be modeled to find decompositions with a high number of constraint overlaps inside. We explained two possibilities how the decomposed sub-CSPs can be used to improve the solving process of the original CSP. One posibility is the substitution of the constraints of a sub-CSP by a locally consistent constraint to reduce the number of fails in the original CSP significantly. The other posibility is using the detected sub-CSPs as work balancing decomposition for parallel consistency methods.

Future work. In the future we will do benchmark tests to verify the influence of our optimizations on real problems. Besides, we want to investigate the idea of transformations of sub-CSPs into locally consistent regular constraints as discussed in [4]. Finally, we want to find a criterion, when to apply the approach.

References

1. Amadini, R.: Portfolio approaches in constraint programming. Constraints **20**(4), 483 (2015). https://doi.org/10.1007/s10601-015-9215-9, https://doi.org/10.1007/s10601-015-9215-9
2. Apt, K.: Principles of Constraint Programming. Cambridge University Press, New York, NY, USA (2003)
3. Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)
4. Löffler, S., Liu, K., Hofstedt, P.: The power of regular constraints in csps. In: Eibl, M., Gaedke, M. (eds.) INFORMATIK 2017. pp. 603–614. Gesellschaft für Informatik, Bonn (2017)
5. Malapert, A., Régin, J., Rezgui, M.: Embarrassingly parallel search in constraint programming. J. Artif. Intell. Res. **57**, 421–464 (2016). https://doi.org/10.1613/jair.5247, https://doi.org/10.1613/jair.5247
6. Marriott, K.: Programming with Constraints - An Introduction. MIT Press, Cambridge (1998)
7. Prud'homme, C., Fages, J.G., Lorca, X.: Choco documentation (2018), http://www.choco-solver.org, last visited 2018-05-04
8. Rolf, C.C., Kuchcinski, K.: Parallel consistency in constraint programming. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2009, Las Vegas, Nevada, USA, July 13-17, 2009, 2 Volumes. pp. 638–644 (2009)
9. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming. Elsevier, Amsterdam, First edn. (2006)
10. Tsang, E.P.K.: Foundations of constraint satisfaction. Computation in cognitive science, Academic Press (1993)