

Decomposing Constraint Satisfaction Problems by Means of Meta Constraint Satisfaction Optimization Problems

Sven Löffler, Ke Liu^a and Petra Hofstedt

Brandenburg University of Technology Cottbus-Senftenberg, Germany

Department of Mathematics and Computer Science, MINT, Germany

Programming Languages and Compiler Construction Group, Konrad-Wachsmann-Allee 5, 03044 Cottbus, Germany

Keywords: Constraint Programming, CSP, CSOP, Decomposition, Parallelization, Optimization, Parallel Constraint Solving, Problem Splitting.

Abstract: This paper describes a new approach to decompose constraint satisfaction problems (CSPs) using an auxiliary constraint satisfaction optimization problem (CSOP) that detects sub-CSPs which share only few common variables. The purpose of this approach is to find sub-CSPs which can be solved in parallel and combined to a complete solution of the original CSP. Therefore, our decomposition approach has two goals: 1. to evenly balance the workload distribution over all cores and solve the partial CSPs as fast as possible and 2. to minimize the number of shared variables to make the join process of the solutions as fast as possible.

1 INTRODUCTION

Constraint programming (CP) is a powerful method to model and solve (not only but especially) NP-complete problems in a declarative way. Typical research problems in CP are rostering, graph coloring, optimization, and satisfiability (SAT) problems (Marriott, 1998).

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. Parallel constraint solving is a promising way to enhance the performance of constraint programming. The four most common kinds of parallel constraint programming are parallel search (Hamadi, 2002; Nguyen and Deville, 1998), parallel consistency (Régin et al., 2013; Rolf and Kuchcinski, 2009), combining parallel search and parallel consistency (Rolf and Kuchcinski, 2009) and distributed CSPs (Faltings, 2006; Yokoo and Hirayama, 2000). Parallel search can furthermore be divided by the way of portfolios, problem splitting, and search space splitting (e. g. embarrassingly parallel search (Régin et al., 2013)).

The presented approach finds decompositions of CSPs in a way that each part has approximately the same size (number of variables) and as little as possi-


ble shared variables so that we can use it effectively for problem splitting. Because the algorithm not only distributes the variables but also the constraints of the CSP it also can be used for parallel consistency techniques.

2 PRELIMINARIES

In this section, we introduce the necessary definitions, methods and theoretical considerations which are the basis of our approach. We consider constraint satisfaction problems (CSPs), constraint satisfaction optimization problems (CSOP) and sub-CSPs which are defined as follows.

CSP (Dechter, 2003) A constraint satisfaction problem (CSP) is defined as a 3-tuple $P = (X, D, C)$ with $X = \{x_1, x_2, \dots, x_n\}$ is a set of variables, $D = \{D_1, D_2, \dots, D_n\}$ is a set of finite domains where D_i is the domain of x_i and $C = \{c_1, c_2, \dots, c_m\}$ is a set of primitive or global constraints containing between one and all variables in X .

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

^a  <https://orcid.org/0000-0002-5256-9253>

CSOP (Rossi et al., 2006; Tsang, 1993) 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.

2.1 Definitions and Methods

In this section we define methods required by our algorithm. Let a CSP $P = (X, D, C)$ be given.

scope(c) (Dechter, 2003) The method *scope* has a constraint $c \in C$ as input and returns all variables $X' \subseteq X$ which are covered by this constraint.

cons(x) The method *cons* has a variable $x \in X$ as input and returns the set of constraints $C' \subseteq C$ which cover variable x .

sharedVars(c₁, c₂) We define a variable $x \in X$ which is covered by two constraints c_1 and $c_2 \in C$ as shared variable of c_1 and c_2 . Based on this, the method *sharedVars* gets two constraints c_1 and c_2 as input and returns all shared variables of the constraints. Analogously if *sharedVars* gets two sub-CSPs P_1 and P_2 as input it returns the shared variables of the sub-CSPs.

2.2 Problem Splitting

The problem splitting process has three steps.

1. Decompose the given CSP $P = (X, D, C)$ in k sub-CSPs $P_{sub} = \{P_1, \dots, P_k\}$ in a way that each sub-CSP $P_i = (X_i, D_i, C_i) \forall i \in 1, \dots, k$ has a subset of constraints $C_i \subseteq C$, where $C_i \cap C_j = \emptyset \forall i, j, i \neq j$ and $\bigcup_{i=1}^k C_i = C$. Note that the constraints of the CSPs P_i are disjoint but the variables can be shared.
2. Solve all sub-CSPs in P_{sub} .
3. Join the results of the sub-CSPs in P_{sub} to a solution of the original CSP P .

The time complexity of problem splitting depends essentially on two considerations. First, the time needed for solving the most complex sub-CSP P_i , because the algorithm can only continue if all sub-CSPs were solved. If D_{max} is the biggest domain of variables in X then the complexity for finding all solutions of sub-CSP P_i is in $O(|D_{max}|^{|X_i|})$.

The second influence on the overall effort for problem splitting comes from the time needed to join the solutions of the sub-CSPs P_i and P_j . Algorithms like sort-merge join can join two tables S and R in $O(N \log N)$, where N is the maximum of $|S|$ and $|R|$. In our case S and R are the partial solutions

of the shared variables of the sub-CSPs P_i and P_j . So let n_{SV} be the number of shared variables $n_{SV} = |sharedVars(P_i, P_j)|$. This means that the maximum number of different value assignments for the shared variables (and so the maximum size N of S and R) must be smaller or equal $|D_{max}|^{n_{SV}}$. That leads to a time complexity for merging two sub-CSPs P_i and P_j given by (1).

$$O(|D_{max}|^{n_{SV}} \log(|D_{max}|^{n_{SV}})) \quad (1)$$

So the overall complexity for both points is given by (2).

$$O(|D_{max}|^{\max(|X_i|, |X_j|)} + |D_{max}|^{n_{SV}} \log(|D_{max}|^{n_{SV}})) \quad (2)$$

If more than two sub-CSPs $P = \{P_1, \dots, P_k | k > 2\}$ are created then we can join them step after step like a binary tree. In each step we join two sub-CSPs. In the next step two of the before joined sub-CSPs will be joined and so on. Let n_{maxSV} be the maximum number of shared variables of all join processes. The complexity for the biggest join process is then in $O(|D_{max}|^{n_{maxSV}} \log(|D_{max}|^{n_{maxSV}}))$.

So the time complexity of the problem splitting process depends on the size of the maximal sub-CSP $P_i \in \{P_1, \dots, P_k\}$ and the maximum number of shared variables in all join processes. Thus, our decomposition CSOP will minimize one or both of these values.

3 THE META CSOP FOR THE DECOMPOSITION OF CSPs

In this section, we present an approach to decompose CSPs into sub-CSPs using a Meta CSOP.

3.1 A CSOP for Separating a CSP into Sub-CSPs

For detecting k sub-CSPs $\{P_1, \dots, P_k\}$ of a given CSP $P = (X, D, C)$ where $X = \{x_1, \dots, x_n\}$, $D = \{D_1, \dots, D_n\}$ and $C = \{C_1, \dots, C_m\}$ we define a CSOP $P_{opt} = (X', D', C', f)$ where $X' = \{x'_j | \forall j \in \{1, \dots, m\}\} \cup \{x_{opt}\}$, $D' = \{D'_j = \{1, \dots, k\} | \forall j \in \{1, \dots, m\}\} \cup \{D_{opt} = \{0, \dots, \infty\}\}$, $C' = \{cspSplit(X', M, k, x_{opt}, w) \cup \{count(l, X', \geq 1) | \forall l \in \{1, \dots, k\}\}$, and $f = minimize(x_{opt})$ which finds an optimized decomposition of P into k sub-CSPs P_1, \dots, P_k if they exist.

For each constraint $c_j \in C$ a variable x_j with domain $D'_j = \{1, \dots, k\}$ is created. The value assignment v for a variable x_j represents that the corresponding constraint c_j is part of sub-CSP P_v . The *count* constraint is defined like in (Prud'homme et al., 2017)

and ensures that no empty sub-CSP is created. More accurate, the $count(l, X', \geq 1)$ constraint guarantees that the value l occur at least one time (≥ 1) in the variable set X' . The *cspSplit* constraint is explained in detail in the next section. The variable x_{opt} is used to optimize the sizes of the sub-CSPs (P_1, \dots, P_k) and the shared variables between them, as these are the main factors which influence the problem splitting complexity (explained above, see 2.2). The objective function f minimizes the x_{opt} variable with domain $D_{opt} = \{1, \dots, \infty\}$ which is also explained in the following section.

3.2 The cspSplit Constraint

We developed a new constraint, *cspSplit*, which partitionate constraints of a given CSP P into a set of sub-CSPs $\{P_1, \dots, P_k\}$.

Doing this the *cspSplit* constraint takes a set of variables X' , a two dimensional array of integers M , the number of sub-CSPs k , the optimization variable x_{opt} and a two dimensional weight vector $w = (w_1, w_2)$ as input.

Each variable $x_j \in X'$ represents the corresponding constraint $c_j \in C$ of the given CSP P . If the variable x_j is set to a value v it means that the constraint c_j is part of the sub-CSP P_v , and therefore not included in other sub-CSPs.

The two dimensional array M indicates which variables are covered by which constraints. For each constraint $c_j \in C$ of CSP P exists one line which contains all variables X_c which are covered by the constraint ($X_c = \{x_i | x_i \in scope(c)\}$). So the entry $M_{j,l} = i$ follows from the fact that constraint c_j contains variable x_i at l -th position.

Algorithm 1 shows the propagation algorithm of the *cspSplit* constraint. In line 1, a binary vector of size $|X|$ (number of variables in the original CSP P) will be created for all k sub-CSPs which should be created. So b_1 represents the first sub-CSP and b_k the last. For each vector b_l the i -th entry ($\forall i \in \{1, \dots, |X|\}$) represents that the i -th variable $x_i \in X$ of CSP P is part of the sub-CSP P_l . Each vector is instantiated with values *null* so no variable is associated to a sub-CSP at the beginning.

In lines 2 to 6, the variables $x_i \in X'$ (representation of c_i in the original CSP P) are checked whether they are already instantiated. If a variable x_i is instantiated to value v then this means that constraint $c_i \in C$ of P is part of the sub-CSP P_v . If such assignments exist, then the algorithm sets all bits in b_v to true which represent variables that are covered by the constraint c_i (line 5 and 6).

Algorithm 1: The *cspSplit* algorithm.

Data: variables X' , $\text{int} \times \text{int}$ M , int k ,
variable X_{opt} , intVector $w = (w_1, w_2)$

```

1 Create  $k$  binary vectors  $b_1, \dots, b_k$  of size  $|X|$ 
2 forall  $x_i \in X'$  do
3   if (isInstantiated( $x_i$ )) then
4     int  $v = x_i.get\text{Value}()$ 
5     forall  $j \in \{1, \dots, |scope(c_i)|\}$  do
6        $b_v[M_{i,j}] = 1$ 
7 forall  $x_i \in X'$  do
8   if (isNotInstantiated( $x_i$ )) then
9     forall  $j \in \{1, \dots, k\}$  do
10      if ( $\forall b_j[M_{i,*}] == 1$ ) then
11        instantiate  $x_i$  to  $j$ 
12        continue with loop in line 7
13 int  $min\text{Value} = w_1 * \text{maximum}(\{\text{number of ones in } b_i | \forall i \in \{1, \dots, k\}\}) +$ 
     $w_2 * \text{maximum}(\text{number of shared variables})$ 
14 update lowerBound of  $x_{opt}$  to  $min\text{Value}$ 

```

In lines 7 to 11, constraints $c_i \in C$ of P (represented by $x_i \in X'$ of P_{opt}) are checked if they are assigned to a sub-CSP. If a constraint c_i has not yet been assigned to a sub-CSP, then we check whether there is a sub-CSP P_j which contains all variables which are covered by this constraint. If this is the case then the constraint will be assigned to that sub-CSP (x_i instantiate to j).

In line 13, the new lower bound of the optimization variable is calculated.

Finally, in line 14, the lower bound of the optimization variable is updated.

Remarks: The steps in line 7-12 associate constraints to a sub-CSP for which no variables must be added to this sub-CSP. This does not influence the correctness of the algorithm but it can increase the solution speed heuristically.

The *minValue* (in line 13) connects exactly the detected parameters (Section 2.2) which has impact on the solution speed of problem splitting multiplied with a weight vector. The number of ones in b_i is exactly the number of variables in the sub-CSP P_i . The number of shared variables can be calculated by creating the intersection of two vectors b_i and b_j . The weight vector allows the user to decide whether he/she wants to improve the solution process or the join process more.

Finding a perfect solution for the CSOP P_{opt} is also very time consuming and in this case may not be useful. It is likely to find a good solution in a short time.

3.3 The Complexity of the CspSplit Algorithm

In this section we consider the complexity of our csp-Split algorithm. Creating k binary vectors of size $|X| = n$ has complexity $O(k*n)$ (line 1).

The cost in lines 2 to 6 depends only from the two nested loops. The other cost are more or less constant. So the complexity for this part of the algorithm (lines 2 to 6) is in $O(m*n_{max})$, where m is the number of variables in X' of P_{opt} (equal to the number of constraints in C of P), and n_{max} is the maximum number of variables a constraint c_i in C of P has.

The cost in line 8, 11 and 12 are constant again so it is only necessary to multiply the number of repetitions of the two loops ($m*k$) with the complexity of the forall statement, which is in $O(n_{max})$. So the overall complexity of this part (lines 7 to 12) is in $O(m*k*n_{max})$.

The cost for the calculation of the *minValue* is in $O(n*k + (k-1)*n^2)$. The complexity of the calculation of the maximum number of shared variables results from the complexity of a join $O(n*n)$ multiplied with the number of joins $(k-1)$ ($k/2$ joins on the first level, then join the $k/4$ joined variable sets again and so on).

This leads to an overall complexity of $O(k*n) + (m*n_{max}) + (m*k*n_{max}) + (n*k + (k-1)*n^2)$. Because of $n_{max} \leq n$ it is in $O(m*k*n + k*n^2)$ where m is the number of constraints in P , n is the number of variables in P and k the number of sub-CSPs we want to create.

4 EXAMPLES AND EXPERIMENTAL RESULTS

In this section, we first present an example for the inputs and outputs of our Meta CSOP and then some of our experimental results. We show that our Meta CSOP can be used for the decomposition of big problems in an acceptable time. In the third part we give a concrete example which improves the solution speed of a CSP significantly by the use of the CSOP.

4.1 Example for the Meta CSOP

For the better understanding we show a small example CSP $P = (X, D, C)$, the corresponding Meta CSOP and a result of its decomposition.

Consider a CSP $P = (X, D, C)$ where $X = \{x_1, \dots, x_6\}$, $D = \{D_1 = \dots = D_6 = \{1, 2, 3, 4, 5\}\}$ and $C = \{c_1 = (x_1 > x_2), c_2 = (x_2 < x_3), c_3 = (x_4 >$

$x_5), c_4 = (x_6 > x_5), c_5 = (x_1 + x_2 = x_3), c_6 = (x_2 + x_4 = x_5 + x_6), c_7 = (x_2 \neq x_4)\}$.

We follow the steps in Section 3.1 and want to detect $k = 2$ sub-CSPs. This yields the Meta CSOP $P_{opt} = (X', D', C', f)$ where $X' = \{x'_1, \dots, x'_7, x_{opt}\}$, $D = \{D'_1, \dots, D'_7, D_{opt}\}$ with $D'_1 = \dots = D'_7 = \{1, 2\}$ and $D_{opt} = \{0, \dots, \infty\}$, $C' = \{cspSplit(\{x'_1, \dots, x'_7\}, M, 2, x_{opt}, w), count(1, \{x'_1, \dots, x'_7\}, \geq, 1), count(2, \{x'_1, \dots, x'_7\}, \geq, 1)\}$ and $f = minimize(x_{opt})$.

For the reason that the original CSP P has seven constraints ($|C| = 7$), the Meta CSOP P_{opt} has seven variables ($|X'| = 7$) with domains $\{1, 2\}$ (because there will be two sub-CSPs, $k = 2$) and one optimization variable x_{opt} with domain $\{0, \dots, \infty\}$. The *cspSplit* constraint gets the variables x'_1, \dots, x'_7 , the matrix M (see in (3)) the number of sub-CSPs ($k = 2$), the optimization variable x_{opt} and a weight vector w as input.

The matrix M indicates in each line i and row j if variable $x_j \in X$ is covered by constraint $c_i \in C$. The weight vector $w = (w_1, w_2)$ can be chosen freely, depending on whether the user wants to prioritize a minimum number of variables per sub-CSP (increase w_1) or a minimum number of shared variables between both sub-CSPs (increase w_2).

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \quad (3)$$

For weight vector $w = (1, 1)$ the algorithm generates the following perfect solution $x'_1 = 1, x'_2 = 1, x'_3 = 2, x'_4 = 2, x'_5 = 1, x'_6 = 2, x'_7 = 2, x_{opt} = 5$. The value 5 for x_{opt} results from the maximum number of variables in each sub-CSP (in this case the first sub-CSP has three variables and the second four) plus the number of shared variables between both sub-CSPs (in this case one because we have only the shared variable x_2).

The solution of the Meta CSOP P_{opt} leads to the following sub-CSPs P_1 and P_2 of the CSP P :

$P_1 = (X^1, D^1, C^1)$ where $X^1 = \{x_1, x_2, x_3\}$, $D^1 = \{D_1, D_2, D_3\}$ and $C^1 = \{c_1 = (x_1 > x_2), c_2 = (x_2 < x_3), c_5 = (x_1 + x_2 = x_3)\}$.

$P_2 = (X^2, D^2, C^2)$ where $X^2 = \{x_2, x_4, x_5, x_6\}$, $D^2 = \{D_2, D_4, D_5, D_6\}$ and $C^2 = \{c_3 = (x_4 > x_5), c_4 = (x_6 > x_5), c_6 = (x_2 + x_4 = x_5 + x_6), c_7 = (x_2 \neq x_4)\}$.

Table 1: Experimental results for the Meta CSOP of the benchmark suite from (Gottlob and Samer, 2009). (*Values after 1s, 10s and 10min).

Name (Cons, Vars)	4 sub-CSPs		8 sub-CSPs		16 sub-CSPs	
	Max Vars*	Shared Vars*	Max Vars*	Shared Vars*	Max Vars*	Shared Vars*
<i>adder</i> _99(496, 694)	279, 266, 239	259, 233, 188	160, 159, 141	312, 302, 220	85, 80, 77	347, 290, 275
<i>NewSystem4</i> (418, 718)	273, 260, 245	226, 214, 172	144, 142, 132	224, 222, 183	78, 77, 72	233, 246, 219
<i>bridge</i> _50(452, 452)	184, 178, 156	171, 180, 120	107, 105, 94	214, 206, 174	68, 61, 56	277, 263, 238
<i>bridge</i> _75(677, 677)	301, 264, 252	327, 241, 224	182, 151, 145	348, 300, 260	107, 106, 85	432, 417, 335
<i>bridge</i> _99(893, 893)	420, 349, 322	441, 343, 262	256, 210, 194	501, 418, 365	141, 123, 107	576, 492, 404
<i>grid2D</i> _30(450, 450)	212, 206, 167	235, 233, 143	136, 133, 115	295, 293, 230	91, 85, 72	348, 338, 286
<i>grid2D</i> _35(612, 613)	359, 271, 256	460, 268, 248	216, 180, 167	456, 388, 329	128, 115, 99	492, 473, 402
<i>grid2D</i> _40(800, 800)	462, 345, 315	430, 320, 278	286, 236, 216	533, 496, 473	169, 152, 127	679, 636, 550
<i>grid2D</i> _45(1013, 1013)	590, 456, 387	601, 336, 330	375, 305, 266	792, 622, 549	214, 200, 166	837, 791, 714
<i>grid2D</i> _50(1250, 1250)	735, 627, 446	817, 503, 364	468, 418, 320	1036, 729, 663	266, 258, 210	1056, 978, 875
<i>grid2D</i> _60(1800, 1800)	1047, 1013, 624	1314, 712, 447	718, 651, 475	1437, 1159, 963	419, 386, 301	1532, 1483, 1220
<i>grid2D</i> _70(2450, 2450)	1461, 1427, 876	1911, 1311, 626	954, 907, 673	2001, 1823, 1427	593, 526, 424	2066, 2059, 1719
<i>grid2D</i> _75(2812, 2813)	1667, 1655, 991	2100, 1563, 736	1092, 1058, 728	2297, 2215, 1521	696, 600, 491	2364, 2332, 1984
<i>s</i> 713(412, 447)	182, 170, 159	165, 139, 132	101, 94, 91	187, 164, 145	57, 56, 51	203, 210, 178
<i>s</i> 838(422, 457)	190, 183, 168	189, 171, 138	111, 106, 101	200, 198, 188	64, 63, 58	247, 227, 213
<i>s</i> 953(424, 440)	193, 188, 178	213, 190, 169	115, 112, 106	224, 225, 202	66, 65, 61	248, 264, 240
<i>s</i> 1196(547, 561)	239, 234, 223	239, 228, 205	145, 142, 136	279, 276, 267	84, 81, 79	312, 312, 300
<i>s</i> 1238(526, 540)	236, 231, 221	237, 227, 204	143, 139, 134	269, 255, 265	84, 80, 78	308, 282, 280
<i>s</i> 1423(731, 748)	316, 286, 274	290, 249, 210	185, 175, 165	356, 320, 290	108, 100, 97	426, 361, 353
<i>s</i> 1488(659, 667)	264, 254, 247	211, 212, 185	158, 152, 149	272, 245, 239	96, 91, 89	329, 329, 315
<i>s</i> 1494(653, 661)	267, 254, 242	244, 218, 185	159, 154, 146	279, 267, 220	95, 90, 88	319, 297, 277
<i>s</i> 5378(2958, 2993)	1249, 1338, 1134	1484, 1388, 952	803, 772, 684	1579, 1566, 1253	490, 421, 392	1594, 1597, 1482

4.2 Experimental Results: Decomposition

We present our experimental results of our Meta CSOP when applying the benchmark suite provided by Gottlob et al. used in (Gottlob and Samer, 2009). We focused on the bigger problems with more than 400 constraints and more than 400 variables. In contrast to Gottlob's det- k -decomp (Gottlob et al., 2000) or to our det- k -CP algorithm (Liu et al., 2018) we try to find a decomposition of a given CSP P in a way that it is optimized for problem splitting. Thus, a comparison with the aforementioned algorithms is not suitable.

We used the weight vector $w = \{1, 0\}$ and printed the best decomposition results found after 1 second, 10 seconds and 10 minutes. The weight vector was chosen in this way because we realized that the number of shared variables is also decreasing during the experiment. Therefore, we only focus on minimizing the maximum number of variables per sub-CSP. In future it must be investigated if there is a connection between both goals or not.

Table 1 shows the experimental results for our Meta CSOP of the benchmark suite from (Gottlob and Samer, 2009). The first column shows the name of the CSP and in parentheses the number of constraints and variables inside the CSP. For each problem and the different number of sub-CSPs ($k = 4, 8$ and 16) a

decomposition could be found in less than 1 second. The values in the columns with name "Max Vars" are the maximum number of variables in each sub-CSP after running the CSOP 1 second, 10 seconds and 10 minutes.

The columns "Shared Vars" show the maximum number of shared variables when joining the sub-CSP pairwise until only one is left. For example, for $k = 4$, sub-CSP P_1 and P_2 have approximately 100 shared variables and sub-CSP P_3 and P_4 may have 93 shared variables and the join of the join of P_1 and P_2 and P_3 and P_4 has 156 shared variables, then the maximum number of shared variables is 156.

In summary, we now conclude that our Meta CSOP can find a balanced decomposition of a given CSP within a reasonable execution time, also for large instances (as far as the number of variables and constraints are concerned). If more time is invested for solving the Meta CSOP a better solution could be found. Depending on the time the user has, he/she can decide if he/she needs a fast decomposition (for example in less than one second) or a very good decomposition. For big problems it may be advantages to invest 10 minutes or more to find decompositions if the problem size is reduced significantly.

4.3 Experimental Results: Influence on the Solution Speed

Next, we want to show that the decomposition of a CSP can influence significantly its solution speed.

We consider the Kakuro shown in figure 1 which was published under the name "Kakuro 1537 medium" in "The Guardian" on February, 3rd, 2017.

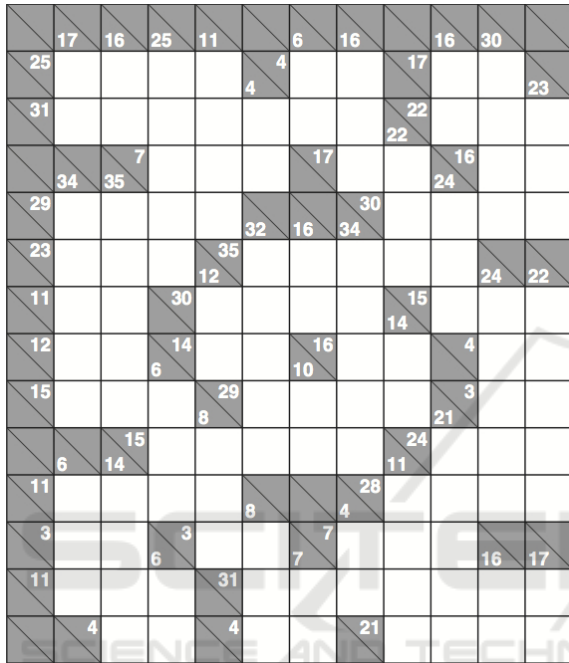


Figure 1: Kakuro 1537.

The rules for Kakuro are "Fill the grid so that each run of squares adds up to the total in the box above or to the left. Use only numbers 1-9, and never use a number more than once per run (a number may reoccur in the same row, in a separate run)".

We modelled the problem with *sum* and *allDifferent* constraints and could solve the problem in less than a half second. To increase the size of the problem we removed the numbers in the cells (sums) and substituted them by variables. So we have not anymore a Kakuro solver, we have a Kakuro generator.

This offers us a problem with a high number of solutions. We solved the problem with and without use of the Meta CSOP approach.

We used a weight vector $w = (1, 5)$ to prioritize the minimization of the shared variables. In both cases (using the Meta CSOP or not) we limited the total time by 5 minutes. So the CSP without decomposition had 5 minutes to solve the problem, and the CSP with decomposition approach had 5 minutes in total (for the decomposition, solving of both sub-CSPs and

joining of the sub-CSPs). We used a decomposition into two sub-CSPs. Important is that we do not do anything in parallel. So we solved the two sub-CSPs sequentially one after the other. This means that we can use other parallelizations like search space splitting too, to increase the solving speed even more. Of course we also can solve the sub-CSPs in parallel to save a little bit less as the half total time.

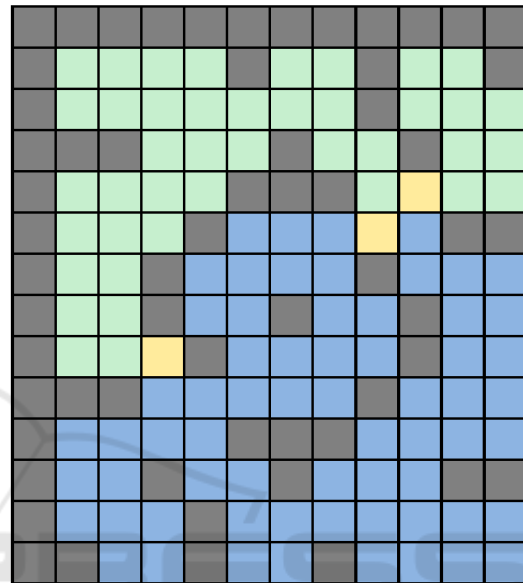


Figure 2: Decomposition of the Kakuro.

Figure 2 shows the decomposition of the Kakuro into two sub CSPs. The variables which are represented by the green cells are only in sub-CSP P_1 , the blue cells only in sub-CSP P_2 . The yellow cells are represented by shared variables which are in both sub-CSPs. A decomposition into two sub-CSPs with only three shared variables is an optimal solution for this Kakuro.

The decomposition approach could find 14.91 times more solutions than the other, in the same time.

Remark: It is clear that not every CSP can be improved with the Meta CSOP, but there are usecases, like the Kakuro example shows. Future work should investigate for which kind/ structure of CSPs this new approach is especially useful.

5 CONCLUSION AND FUTURE WORK

We have presented a new way to decompose a CSP into sub-CSPs by the use of a Meta CSOP. The decomposition is specialized to find distributed sub-CSPs for parallel problem splitting with a minimum

number of variables in each sub-CSP and a low number of shared variables between the sub-CSPs. We evaluated our approach by a benchmark suite of Gottlob (Gottlob and Samer, 2009). Our results have shown that our approach is appropriate for partitioning a constraint network distribution on a given number of parallel constraint solving cores in short time. In contrast to (Liu et al., 2018), the Meta CSOP works also for big CSPs with more than 2000 variables and constraints.

Future work will include a comparison with other decomposition methods like (Karypis and Kumar, 1998) and their developments, and more practical tests which will show the influence of the decomposition on the solving speed of the original CSP. Furthermore we will also focus on searching decomposition methods which minimize the size of the shared domain sizes (i. e. $\prod_{x_i \in \{\text{shared variable}\}} |D_i|$) and not only the shared variables.

REFERENCES

- Dechter, R. (2003). *Constraint processing*. Elsevier Morgan Kaufmann.
- Faltings, B. (2006). Distributed constraint programming. In *Handbook of Constraint Programming*, pages 699–729.
- Gottlob, G., Leone, N., and Scarcello, F. (2000). A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2):243–282.
- Gottlob, G. and Samer, M. (2009). A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13:1:1.1–1:1.19.
- Hamadi, Y. (2002). Optimal distributed arc-consistency. *Constraints*, 7(3-4):367–385.
- Karypis, G. and Kumar, V. (1998). Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1998, November 7-13, 1998, Orlando, FL, USA*, page 28. IEEE Computer Society.
- Liu, K., Löffler, S., and Hofstedt, P. (2018). Hypertree decomposition: The first step towards parallel constraint solving. In Seipel, D., Hanus, M., and Abreu, S., editors, *Declare 2017 – Conference on Declarative Programming*, pages 91 – 103.
- Marriott, K. (1998). *Programming with Constraints - An Introduction*. MIT Press, Cambridge.
- Nguyen, T. and Deville, Y. (1998). A distributed arc-consistency algorithm. *Science of Computer Programming*, 30(1-2):227–250.
- Prud’homme, C., Fages, J.-G., and Lorca, X. (2017). Choco documentation.
- Régin, J., Rezgui, M., and Malapert, A. (2013). Embarrassingly parallel search. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 596–610.
- Rolf, C. C. and Kuchcinski, K. (2009). 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*, pages 638–644.
- Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier, Amsterdam, First edition.
- Tsang, E. P. K. (1993). *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press.
- Yokoo, M. and Hirayama, K. (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207.