Constraint Solving by Composition

Student: Zhijun Zhang
Supervisor: Susan L. Epstein

The Graduate Center of the City University of New York,
Computer Science Department
365 Fifth Avenue, New York, NY 10016-4309, USA
zzhang@gc.cuny.edu, susan.epstein@hunter.cuny.edu

Abstract. A structure-based approach to the solution of constraint problems (e.g., tree-based decomposition) typically assumes that knowledge about the problem remains static, and demands extensive resources to pursue the solution. This may explain why such an approach is sometimes infeasible on large-scale, real-world problems. This paper proposes a new approach, called *composition* that partitions a problem into a disjoint set of star-like subproblems. By design, search for solution on each subproblem is tractable. Composition identifies, solves, and then propagates the result one subproblem at a time. This reformulation makes the original problem model both denser and tighter, and often facilitates search. Initial tests have proved promising.

1 Introduction and related work

A structure-based constraint solver usually decomposes a problem into possibly overlapping subproblems with an acyclic structure [1-7]. The premise is that the subproblems will be easier to solve individually than the original problem, and that the acyclic structure assembled from the solved subproblems admits a tractable solution to the entire problem. In contrast, the thesis of this work is that a large-scale, real-world problem may be better addressed by *composing* an equivalent problem from a set of disjoint, readily solvable subproblems that present a denser and tighter structure.

Formally, a CSP P is represented by a triple $<X, D, C>$, where X is a finite set of *variables* $\{X_1, X_2, \ldots, X_n\}$, D is a set of *domains* $\{D_1, D_2, \ldots, D_n\}$ in which D_i is a set of values for variable $X_i \in X$, and C is a set of *constraints* $\{C_1, C_2, \ldots, C_m\}$. A constraint C_j is defined by a scope $scope(C_j) \subseteq X$ and a relation $relation(C_j)$ that defines all consistent instantiations of variables in the scope. The problem model is currently limited to *binary* constraint problems, which have scopes of size at most 2. A *subproblem* $P_S = < X_S, D_S, C_S>$ of P is formed by a subset X_S of X, where $D_S = \{D_i \in D \mid X_i \in S\}$ and $C_S = \{C_i \in C \mid scope(C_i) \subseteq X_S\}$. The *meta-constraint* C_{TU} between subproblems $P_T =< X_T, D_T, C_T>$ and $P_U = < X_U, D_U, C_U>$ of P is formed from the constraints in P: $C_{TU} = \{C' \in C \mid \text{for } X_i, X_j \in X, X_i \in X_T, X_j \in X_U, scope(C') = [X_i, X_j]\}$.

The constraint graph formed with nodes representing subproblems and edges representing meta-constraints is called a *meta-structure*.

Tree-based decomposition is an example of structure-based constraint solving. In decomposition, a pair of adjacent meta-variables may share a set of original variables, which is called a *separator*. An acyclic meta-structure guarantees a tractable final solution [6, 7]. Thus, the solution complexity of the entire problem is dominated by the solution complexity of all meta-variables. For large-scale, real-world problems, however, tree-based decomposition may not be a good choice [1, 8] for several reasons. First, detecting special structures (e.g., a maximal independent variable set [3]) to construct an acyclic meta-structure can be costly, yet an approximation may not have the same impact on performance. Second, space complexity is potentially exponential in the maximum size of its separators [1]. Finally, addressing decomposition meta-variables first may mislead search. For example, if the cause of over-constrainedness is the meta-constraints, rather than the individual constraints, solving every meta-variable first is not productive. If a problem has few solutions, each meta-variable may still have many solutions, which ultimately prove invalid under the meta-constraints. In this case, the meta-constraints should be addressed early, because solving all the meta-variables may be NP-hard. We believe the last two of these conditions make some real world problems (e.g., RLFAP-11) particularly difficult to solve.

A novel approach to hard problems in combinatorial design studied the patterns embedded in the solutions of low-order problems, and abstracted the patterns into a *streamlined* constraint that partitioned the original problem P into P_1 and P_2 [11]. P_1 is small and polynomially solvable; it holds the variables subject to the streamlined constraints, P_2, which does not cover patterns, is larger and possibly NP-hard. The method solves P_1 first and then recursively partitions P_2 with streamlined constraints until it constructively solves the entire problem. Although this method solves higher order problems, there is no obvious way to apply it to real world problems because it relies on small-sized sample problems to estimate patterns first. Nonetheless, this work confirms that the properties of a problem's solutions provide the key to solving it. We conjecture, however, that comparatively easy subproblems play an important role in finding solutions to large scale, real world problems.

The remainder of this paper explains how composition detects structure inexpensively to form the meta-structure, avoids expensive separators, can update the problem model dynamically if necessary, and pursues an effective distribution of search and propagation within a cyclic meta-structure.

2 Composition, a new constraint solver

A CSP $P = \langle X, D, C \rangle$ can be represented by a *constraint graph* $G = \langle V, E \rangle$, where each V_i in V corresponds to a unique X_i in X, and each edge in E joins two vertices in V if and only if their corresponding variables have a common constraint in C. A *tractable meta-variable* of a CSP P is represented by an induced graph $G[V_i, E_i]$ where $V_i \subseteq V$ and $E_i = \{E_j \in E \mid scope(E_j) \subseteq V_i\}$ that can be solved in polynomial time. Given a CSP $P = \langle X, D, C \rangle$, *composition* transforms P into a CSP $P' = \langle X', D', C' \rangle$

```
1.   Composition (P = <X, D, C>, G = <V, E>)
2.     Until G′ is sufficiently dense
3.       i = 0
4.       Until G is empty
5.         i ← i + 1
6.         Construct a component G[V_i], where V_i ⊆ V
7.         [Optional] Search on G[V_i] to some degree
8.         [Optional] Propagate the (partial) solution of G[V_i]
9.         Remove G[V_i] from G
10.      Compose all components to construct a new P′
11.    sol_P ← Solve P′
12.    Return sol_P
```

Fig. 1. Pseudocode for composition constructs a partition on P and composes a new P',

without loss of any solution. Pseudocode for composition appears in Figure 1, where problem P has constraint graph G. The *partition loop* in lines 4 – 9 repeatedly identifies and removes a tractable meta-variable from P. Each time composition identifies a new meta-variable $G[V_i]$, the solver may search and/or perform inference on it to some degree. Once the partition is completed, composition (line 10) combines all the meta-variables into a new model P' with constraint graph G'. If a problem changes gradually over time, line 10 could incrementally adapt meta-variables and meta-constraints and/or create new ones. Composition (unlike decomposition) forms a true partition that makes update much easier because there is no need to synchronize on shared variables in the separators. The conjecture is that P' will be easier to solve than P, because P' is much denser and

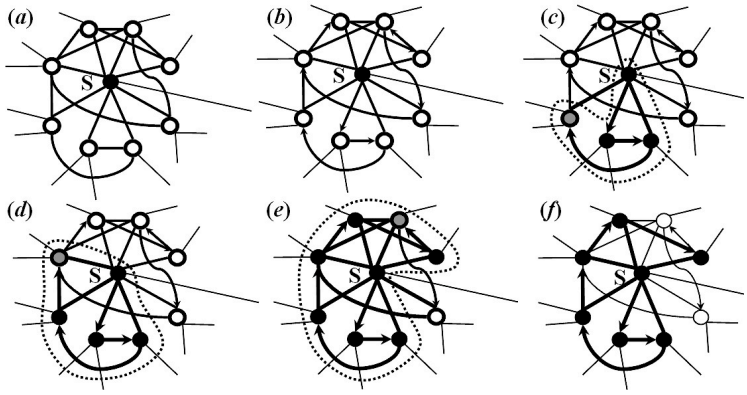

Fig. 2. One implementation of a neighbor rule: (a) Select center S and identify $G[S]$ with the algorithm of Figure 2. (b) Calculate spanning tree ST over $G[S]$ depth-first, selecting vertices in ascending order of their degree in $G[S]$. (c, d, e) Starting from the center, incorporate one neighbor at a time along the longest path in ST. The *neighbor rule* shown here specifies that any vertex whose degree in the corresponding induced graph is no larger than 3 can be included in the meta-variable. (e) Construction stops at the first vertex that violates the neighbor rule in the longest path in ST. (f) All the selected vertices constitute now the tractable meta-variable. After that, any dangling tree incident on the meta-variable can be included as a part of the meta-variable since the center removal still makes the remainder acyclic.

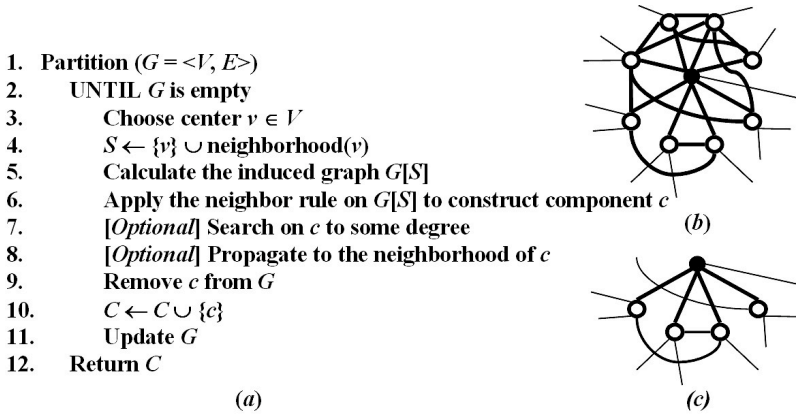

Fig. 3. (a) The pseudocode for one simple partition and the tractable meta-variable; (b) The meta-variable $G[S]$. (c) The tractable meta-variable after applying the neighbor rule at line 6 to $G[S]$. Note that the center removal from the meta-variable makes the remainder acyclic, which guarantees that the meta-variable has a tractable solution [6, 7].

tighter, due both to the nature of the partition and to any search and propagation executed in lines 7 and 8.

Unlike decomposition, which permits overlap among meta-variables, a *partition* of a CSP forms a set of meta-variables that are mutually exclusive and collectively exhaustive. I have explored several partition methods for the loop from line 4 to 9 in Figure 1. During partition, one important question is how to construct a meta-variable that has a tractable solution. It is guaranteed by the *neighbor rule*. Figure 2 gives an example of a neighbor rule that constructs a tractable meta-variable. Figure 3 is an example of an implementation for the partition loop in Figure 1.

3 Implementation and initial testing

A first version of composition, *static composition*, is now implemented, with pseudocode in Figure 4. During partition, static composition enforces singleton arc consistency [10] over each meta-variable for every possible instantiation of its center only, and stores the resulting domains for the neighbors in a dedicated data structure. Thus, line 10 becomes a simple lookup that assigns values to the centers and corresponding domains to neighbors in a meta-variable. Line 11 solves the remainder of the problem by backtrack search. Currently, the algorithm makes a single partition cycle (i.e., $i = 1$) because the constraint graph of P' is usually almost complete after one cycle.

Static composition has been tested on RLFAP scenes 1, 2, 3, 10, and 11, and compared to one traditional solver that uses MAC and another that uses FC. Each solver used a variable-ordering heuristic that minimized dynamic domain size divided by dynamic degree and selected values lexically. On scenes 1, 2, and 3, the static-composition algorithm required search time comparable to both the other solvers, but

```
1.   Static-Composition (P = <X, D, C>, G = <V, E>)
2.       Until G′ is sufficiently dense
3.           i = 0
4.           Until G is empty
5.               i ← i + 1
6.               Construct a star-based component G[V_i], where V_i ⊆ V
7.               Enforce SAC over G[V_i]
8.               Remove G[V_i] from G
9.           Compose new P′
10.      sol_P = Solve centers in P′
11.      sol_P = sol_P + Solve neighbors in P′
12.      Return sol_P
```

Fig. 4. Static-composition, one possible implementation of composition in Figure 1. Centers and their neighbors in the tractable meta-variable are predefined by the partition. The method in Figure 2 is applied at line 6. Singleton arc consistency is enforced over $G[V_i]$ at line 7. Lines 10 and 11 define a static variable ordering for the solution on the new model P'.

devoted extra time to partitioning. On scene 10, however, static composition is significantly slower than both of the others. Scene 11 is well-known to be the difficult one in this suite. Here, despite a somewhat inefficient current implementation, static composition solved the problem in about 10% of the time required by the other solvers, including its preparation time to partition the variables.

4 Discussion and current work

To understand composition, let $P^{(m,d)}$ be the meta-graph generated by a composition algorithm on CSP P, where m is the maximum meta-variable size and d is the maximum degree of any meta-variable in the resulting meta-graph. Thus, $P^{(n,0)}$ treats the original problem as a single meta-variable with degree zero. $P^{(1,d)}$ is equivalent to the original constraint graph on n nodes with maximum degree d. Effectively, $P^{(1,\ d)}$ treats every variable as a single meta-variable and every constraint as a meta-constraint.

$P^{(1,\ d)}$ shows that any constraint graph can be partitioned into meta-variables that can be solved tractably and composed into a meta-structure equivalent to the original graph. We conjecture that the parameters m and d dominate the space and time complexity of composition, and that composition can strike a balance in performance by finding an appropriate $P^{(x,\ y)}$ between $P^{(1,\ d)}$ and $P^{(n,\ 0)}$. This is analogous to the parameterized decomposition algorithm achieved in [1], which finds the optimal separator size for the tradeoff between time and space complexity of decomposition.

Current work focuses on a dynamic version of Figure 1, a composition algorithm that solves the entire problem one meta-variable at a time during partition. Under dynamic composition, center selection plays a critical role, since each meta-variable is constructed based on the experience and consequences of solving the previous meta-variables. Center selection could be governed by new variable ordering heuristics. We also intend to explore the role of value ordering in center instantiation, meta-variable size control, and level of propagation, and how to propagate meta-variable solutions,

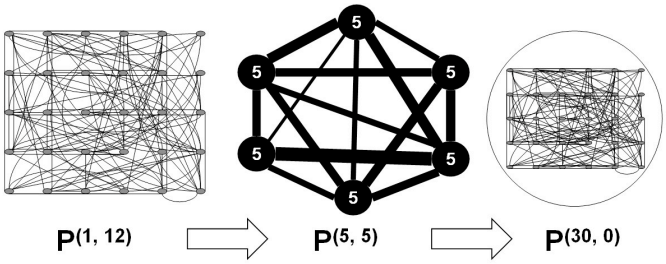

Fig. 5. Understanding composition with a CSP on 30 variables whose maximum degree is 12. The CSP on the left is a meta-graph identical to the original constraint graph; it consists of 30 "meta-variables" of size 1 with maximum degree 12. The CSP on the right re-represents the problem with a single meta-variable of degree 0 that contains all 30 of the original variables. The CSP in the center re-represents the problem with 6 tractable meta-variables of size 5, whose maximum degree is the meta-graph is 5. Composition constructs and works on a representation of the problem like that in the center, which offers an efficient solution under composition. Different thicknesses of the meta-edges represent the number of original constraints imposed by the problem.

backjumping from one meta-variable to another during dynamic composition. We will also explore hybrid approaches that combine local search with systematic search.

References

1. Rina Dechter and Yousri El Fattah, Topological Parameters for time-space tradeoff. *Artificial Intelligence*, 2001. 125: p. 93-118.
2. Rina Dechter and Judea Pearl, Tree Clustering for constraint net-works. *Artificial Intelligence*, 1989. 38: p. 353-366.
3. Joel Gompert and Berthe Y. Choueiry, A decomposition techniques for CSPs using maximal independent sets and its integration with local search. in *Proceedings of the International FLAIRS Conference* (FLAIRS'05). 2005.
4. Rainer Weigel and Boi Faltings, Compiling Constraint Satisfaction Problems. *Artificial Intelligence*, 1999. 115: p. 257-287.
5. Yaling Zheng and Berthe Y. Choueiry, New structural decomposition techniques for constraint satisfaction problems. *Lecture Notes in Artificial Intelligence*, 2005. 3419: p. 113-127.
6. Eugene C. Freuder, A sufficient condition for backtrack-free search. *JACM*, 1982. 29(1): p. 24-32.
7. Eugene C. Freuder, A sufficient condition for backtrack-bounded search. *JACM*, 1985. 32(4): p. 755-761.
8. Philippe Jegou and Cyril Terrioux, Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 2003. 146: p. 43-75.
9. Georg Gottlob, Nicola Leone, and Francesco, Scarcello, Hypertree decompositions: a survey. *Lecture Notes in Computer Science*, 2001. 2136: p. 37-57.
10. Romual Debruyne and Christian Bessière, Some practicable filtering techniques for the constraint satisfaction problem. in *Proceedings of IJCAI'97*. 1997. Nagoya, Japan.
11. Carla Gomes and Meinolf Sellmann, Streamlined Constraint Reasoning. in *Principles and Practice of Constraint Programming. 2004*. Toronto, Canada: Springer.