Kernelization, Generation of Bounds, and the Scope of Incremental Computation for Weighted Constraint Satisfaction Problems

T. K. Satish Kumar[*]
Computer Science Department
University of Southern California
tkskwork@gmail.com

Abstract

In this paper, we present an algorithmic framework for kernelization of combinatorial problems posed as weighted constraint satisfaction problems (WCSPs). Our kernelization technique employs a polynomial-time maxflow-based algorithm to fix the optimal values of a subset of the variables in a preprocessing phase. It thereby reduces the set of variables for which exhaustive search is eventually required to all but a small kernel. We present some implications of this algorithmic framework and show that it can be used to generate intelligent lower and upper bounds for the costs of optimal solutions to WCSPs. We also show that it can be used to study the scope of incremental computation for WCSPs.

Introduction

A weighted constraint satisfaction problem (WCSP) is a generalized optimization version of a constraint satisfaction problem (CSP) in which the constraints are no longer "hard" but are extended by associating non-negative *costs* to the tuples (Bistarelli et al. 1996). The goal is to find an assignment of values to all the variables from their respective domains such that the total cost is minimized.

More formally, a WCSP is defined by a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X} = \{X_1, X_2 \ldots X_N\}$ is a set of *variables* and $\mathcal{C} = \{C_1, C_2 \ldots C_M\}$ is a set of *weighted constraints* on subsets of the variables. Each variable X_i is associated with a discrete-valued *domain* $D_i \in \mathcal{D}$; and each constraint C_i is defined on a certain subset $S_i \subseteq \mathcal{X}$ of the variables. S_i is referred to as the *scope* of C_i; and C_i specifies a non-negative *cost* for every possible combination of values to the variables in S_i. An *optimal* solution is an assignment of values to all the variables from their respective domains so that the *sum* of the costs—as specified locally by each weighted constraint—is *minimized.* In a Boolean WCSP, the size of any variable's domain is equal to 2—i.e., $|D_i| = 2$ for all $i \in \{1, 2 \ldots N\}$. Boolean WCSPs are representationally as powerful as WCSPs; and it is well known that optimally solving WCSPs or Boolean WCSPs is NP-hard in general.

The WCSP framework is representationally very powerful and can be used to model a multitude of real-world combinatorial tasks. In fact, a wide range of combinatorial problems studied across different research communities fall under the broad umbrella of WCSPs. In probabilistic reasoning, for example, a Bayesian network can model the world using random variables and conditional probability tables associated with each of them (Russell and Norvig 2003). Given observations on some of the variables, the commonplace task of finding the most probable assignment of values to the remaining variables is equivalent to solving a WCSP by treating the negative logarithms of the probabilities in the conditional probability tables as costs.

In computer vision, tasks such as image smoothing can be posed as WCSPs by associating a variable with each pixel (Kolmogorov and Zabih 2002; Rother, Kolmogorov, and Blake 2004; Boykov and Funka-Lea 2006). Unary weighted constraints over individual variables represent the cost of deviation from the observed pixel values. Binary weighted constraints over variables corresponding to adjacent pixels represent smoothness requirements. In statistical physics, ground states of spin glasses can be computed by solving WCSPs. In the Edwards-Anderson model (Edwards and Anderson 1975), for example, a variable is associated with each particle in a d-dimensional lattice. Each particle can have a positive or negative Ising spin. While the sum of the unary spins represents the external magnetic field, the interactions between neighboring particles can be ferromagnetic or anti-ferromagnetic. Even classical combinatorial problems in graph theory and discrete mathematics can be modeled using the WCSP framework (Kumar 2008a; 2008b). For example, among many other combinatorial problems in graph theory, the tractable *min-st-cut* problem as well as the NP-hard *max-cut* problem can both be modeled using Boolean WCSPs with only binary constraints.

In this paper, we present an algorithmic framework for kernelization of Boolean WCSPs. Our kernelization technique employs a polynomial-time maxflow-based algorithm to fix the optimal values of a subset of the variables in a preprocessing phase. It thereby reduces the set of variables for which exhaustive search is eventually required to all but a small kernel. As mentioned above, Boolean WCSPs are representationally as powerful as WCSPs and are able to model diverse combinatorial tasks studied across different research communities. A kernelization technique for Boolean WCSPs therefore has direct and important implications on our ability to kernelize these real-world problem instances. For a

[*]T. K. Satish Kumar, alias Satish Kumar Thittamaranahalli, is a Research Scientist at the University of Southern California.

Boolean WCSP with N variables, a kernelization procedure that reduces the number of variables to $N/2$ in the kernel has huge benefits since the size of the search space reduces to $\sqrt{2^N}$ from the original 2^N. Of course, kernelization can work in conjunction with any efficient solver for the remaining kernel of variables.

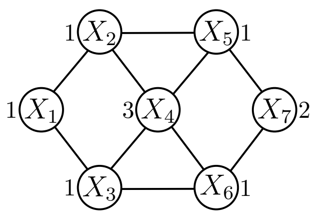
Despite its conceptual appeal, kernelization has not received much attention in the AI community. In the context of CSPs, arc-consistency (Lecoutre 2009)—and perhaps other forms of local consistency like singleton arc-consistency (Bessiere 2008)—can be viewed as kernelization techniques. But even these are not very powerful pruning techniques when generalized to WCSPs (Cooper and Schiex 2004; Larrosa 2004; Zytnicki 2009). In the complexity theory community, kernelization is studied for very specific problems—like the minimum vertex cover problem (Abu-Khzam et al. 2004)—often in conjunction with fixed-parameter tractability (Niedermeier 2006).

In this paper, we present a viable technique that uses maxflow computations for kernelization of any combinatorial problem posed as a Boolean WCSP. We study some implications of this algorithmic framework and show that it can also be used to generate intelligent lower and upper bounds on the costs of optimal solutions to WCSPs. The upper bound is generated constructively along with a candidate solution whose cost matches this bound. On further examination, we find that the same framework can also be used to understand the scope of incremental computation for combinatorial problems posed as Boolean WCSPs.

High-Level Ideas

In this section, we briefly outline the main ideas in the paper. Formal descriptions appear in later sections. The first idea is to exploit the mathematical framework of the *constraint composite graph* (CCG) introduced in (Kumar 2008a; 2008b). The CCG is a graphical representation of a given WCSP that is built using a simple composition of the individual lifted graphical representations for each weighted constraint. By design, it captures the "numerical" structure of the weighted constraints which the constraint network otherwise ignores. In addition, the treewidth of the CCG is no more than the treewidth of the constraint network (Kumar 2008a). This makes the CCG a unifying framework for simultaneously exploiting the numerical structure of the weighted constraints as well as the graphical structure of the variable-interactions. Classes of WCSPs that are tractable by virtue of the structure in their weighted constraints often map to tractable minimum weighted vertex cover (MWVC) problems over bipartite CCGs. Classes of WCSPs that have a low treewidth map to MWVC problems over CCGs that also have the same low treewidth (Kumar 2008a). Furthermore, any WCSP has a tripartite CCG associated with it and is equivalent to solving the MWVC problem over this CCG (Kumar 2008a; 2008b).

The second idea is to make use of the kernelization properties of the MWVC problem. In particular, a systematic study of kernelization techniques for the MWVC problem based on *crown reductions* is presented in (Chlebik and



$X_1 \backslash X_4$	0	1
0	4	7
1	5	6

Figure 1: The table on the right-hand side represents the projection of the MWVC problem onto the IS $\{X_1, X_4\}$ of the node-weighted undirected graph on the left-hand side. (The weights on X_4 and X_7 are set to 3 and 2, respectively, while all other nodes have unit weights.) The entry '7' in the cell $(X_1 = 0, X_4 = 1)$, for example, indicates that when X_1 is prohibited from being in the MWVC but X_4 is necessarily included in it, then the weight of the MWVC—$\{X_2, X_3, X_4, X_7\}$ or $\{X_2, X_3, X_4, X_5, X_6\}$—is 7.

Chlebikova 2007). In concluding remarks of this paper, the authors say "The technique developed in this paper may be a powerful tool in kernelization for other optimization problems . . .". The idea of the CCG confirms this conjecture. A systematic way to construct the CCG for any given Boolean WCSP is encapsulated as a simple polynomial-time procedure and is presented in (Kumar 2008a). Since the Boolean WCSP is equivalent to solving the MWVC problem on its associated CCG, kernelization techniques for the MWVC problem carry over to the original Boolean WCSP.

The third idea is to make use of the fact that the MWVC problem is half-integral (Nemhauser and Trotter 1974). This means that kernelization for the MWVC problem can be done using a staged maxflow computation on a bipartite graph (Chlebik and Chlebikova 2007). This yields a polynomial-time maxflow-based kernelization procedure for WCSPs using their CCGs. The overall procedure for solving a given WCSP would therefore be to: (a) construct its CCG; (b) kernelize the MWVC problem on its CCG using a maxflow computation; and (c) solve the kernel using search-based methods. While (a) and (b) are polynomial-time phases, (c) is not. Furthermore, since (b) is based on maxflow, it can be made incremental as described in (Kumar 2003; Kumar and Gupta 2003). This gives us an understanding of the scope of incremental computation for WCSPs. Because WCSPs are NP-hard in general, they are not easily amenable to incremental computation.[1] Nonetheless, the idea of a maxflow-based kernelization facilitates incremental computation to a certain extent. In particular, (b) can be made incremental and (c) poses a much smaller search space than for the original WCSP without kernelization.

Background on CCGs

In an undirected graph $G = \langle V, E \rangle$, $U = \{u_1, u_2 \dots u_k\}$ is said to be an *independent set* (IS) of G if and only if no two nodes in U are connected by an edge in E. A *vertex cover* (VC) is a set of nodes $S \subseteq V$ such that every edge has at least one end-point in S. A *minimum* VC is a VC of minimum size. When non-negative weights are associated

[1]If they were, one could start with a trivial case, add the variables of a given WCSP one at a time, and build the optimal solution incrementally and efficiently.

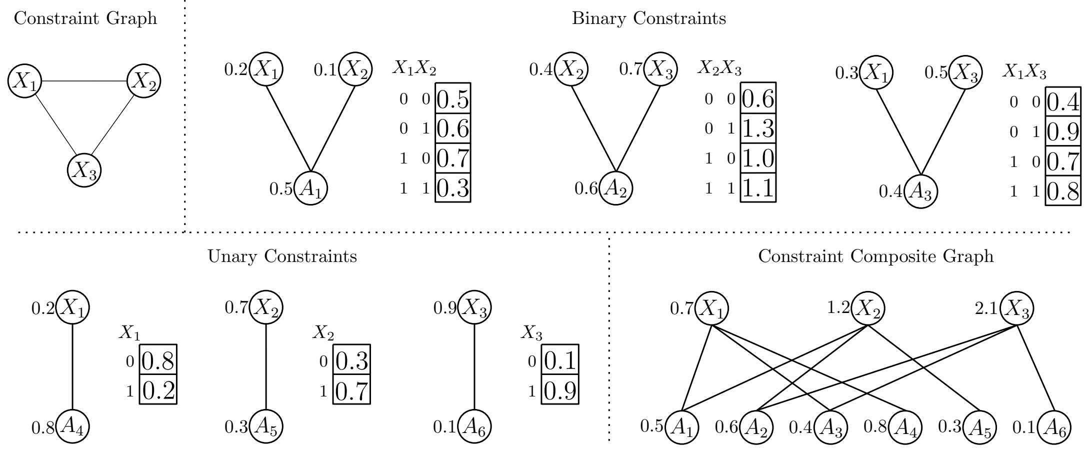


Figure 2: Shows a WCSP over 3 Boolean variables. The constraint network is shown in the top-left cell, and the 6 binary and unary weighted constraints are shown along with their lifted graphical representations in the 1^{st} and 2^{nd} rows. The CCG is shown in the bottom-right cell.

with nodes, the MWVC is defined to be a VC of minimum total weight on its nodes.

The concept of the MWVC on a given undirected graph $G = \langle V, E \rangle$ can be extended to the notion of projecting MWVCs onto a given IS $U \subseteq V$. The input to such a projection is the graph G as well as an identified IS $U = \{u_1, u_2 \dots u_k\}$. The output is a *table* of 2^k numbers. Each entry in this table corresponds to a k-bit vector. We say that a k-bit vector t imposes the following restrictions: (a) if the i^{th} bit t_i is 0, the node u_i is necessarily excluded from the MWVC; and (b) if the i^{th} bit t_i is 1, the node u_i is necessarily included in the MWVC. The projection of the MWVC problem onto the IS U is then defined to be a table with entries corresponding to each of the 2^k possible k-bit vectors $t^{(1)}, t^{(2)} \dots t^{(2^k)}$. The value of the entry corresponding to $t^{(j)}$ is equal to the weight of the MWVC *conditioned* on the restrictions imposed by $t^{(j)}$. Figure 1 presents a simple example to illustrate the notion of projecting MWVC problems onto an IS in a node-weighted undirected graph.

The table of numbers produced above can be viewed as a weighted constraint over $|U|$ Boolean variables. Conversely, given a (Boolean) weighted constraint, we can think about designing a "lifted" representation for it so as to be able to view it as the projection of an MWVC problem in some intelligently constructed node-weighted undirected graph. This idea was first discussed in (Kumar 2008a; 2008b). The benefit of constructing these graphical representations for individual constraints lies in the fact that the lifted graphical representation for the entire WCSP can be obtained by simply "merging" them. This merged graph is referred to as the CCG associated with the WCSP.

Figure 2 shows an example WCSP over 3 Boolean variables to illustrate the construction of the CCG. Here, there are 3 unary weighted constraints and 3 binary weighted constraints; and their lifted representations (as projections of MWVC problems) are shown next to each of them. The figure also illustrates how the CCG is obtained from the individual graphs representing each of the weighted constraints. In the CCG, nodes that represent the same variable are simply "merged"—along with their edges—and every "composite" node is given a weight equal to the sum of the individual weights on the merged nodes. Computing the MWVC for the CCG yields an optimal solution for the WCSP; namely, if X_i is in the MWVC, then it is assigned the value 1 in the WCSP, else it is assigned the value 0 in the WCSP.

Any given weighted constraint on Boolean variables (that is, a Boolean weighted constraint) can be represented graphically using a *tripartite* graph that can be constructed in polynomial time (Kumar 2008a). In many cases, the lifted graphical representations turn out to be only *bipartite*. Since the resulting CCG is also bipartite if each of the individual graphical representations is bipartite, the tractability of the language $\mathcal{L}^{Boolean}_{bipartite}$—the language of all Boolean weighted constraints with a bipartite graphical representation—is readily established. This is because solving MWVC problems on bipartite graphs is reducible to maxflow problems and can therefore be done efficiently in polynomial time (Goldberg and Tarjan 1988).

Kernelization of Boolean WCSPs

Given a Boolean WCSP $\mathcal{B}$, let $\mathcal{C}(\mathcal{B}) = \langle V_{\mathcal{C}(\mathcal{B})}, E_{\mathcal{C}(\mathcal{B})} \rangle$ be its associated CCG. As mentioned before, $\mathcal{C}(\mathcal{B})$ can be built using a simple polynomial-time algorithm presented in (Kumar 2008a). The algorithm first converts each weighted constraint to a multi-variate polynomial and then composes the gadgets built for each of the individual terms in this polynomial. While linear and negative nonlinear terms have bipartite representations, positive nonlinear terms have tripartite representations. In general, $\mathcal{C}(\mathcal{B})$ is a tripartite graph over the original variables in $\mathcal{B}$—i.e., $\mathcal{X}(\mathcal{B}) = \{X_1, X_2 \dots X_N\}$—as well as some auxiliary variables $\mathcal{A}(\mathcal{B}) = \{A_1, A_2 \dots A_{N'}\}$. In other words, we have $V_{\mathcal{C}(\mathcal{B})} = \mathcal{X}(\mathcal{B}) \cup \mathcal{A}(\mathcal{B})$.

Finding the optimal solution for $\mathcal{B}$ is equivalent to solving the MWVC problem on $\mathcal{C}(\mathcal{B})$. Consider the Integer Linear Programming (ILP) formulation of this MWVC problem

Algorithm 1: KERNELIZE-BOOLEAN-WCSP
Shows the general maxflow-based kernelization procedure for Boolean WCSPs. The procedure: (a) constructs the CCG; (b) kernelizes its MWVC; and (c) projects it back to the Boolean WCSP.

```
Input: a Boolean WCSP $\mathcal{B}$ with variables $\mathcal{X}(\mathcal{B}) = \{X_1, X_2 \dots X_N\}$
Output: a subset of variables $\mathcal{S} \subseteq \mathcal{X}(\mathcal{B})$ with their optimal Boolean assignments; $\mathcal{X}(\mathcal{B}) \setminus \mathcal{S}$ is the kernel
(1) Using the polynomial-time procedure presented in (Kumar 2008a), construct $\mathcal{C}(\mathcal{B}) = \langle V_{\mathcal{C}(\mathcal{B})}, E_{\mathcal{C}(\mathcal{B})} \rangle$, the CCG of the Boolean WCSP $\mathcal{B}$.
(2) Build a bipartite graph $\mathcal{D}(\mathcal{B}) = \langle V_L^d, V_R^d, E^d \rangle$ as follows:
    (a) For each node $v \in V_{\mathcal{C}(\mathcal{B})}$, create two corresponding nodes of the same weight in $\mathcal{D}(\mathcal{B})$: $v_L^d \in V_L^d$ and $v_R^d \in V_R^d$.
    (b) For each edge $(u, v) \in E_{\mathcal{C}(\mathcal{B})}$, create two corresponding edges in $\mathcal{D}(\mathcal{B})$: $(u_L^d, v_R^d) \in E^d$ and $(v_L^d, u_R^d) \in E^d$.
(3) Using the polynomial-time maxflow-based algorithm of (Goldberg and Tarjan 1988), solve for $VC^d$, the MWVC of the bipartite graph $\mathcal{D}(\mathcal{B})$.
(4) Define a half-integral vertex cover $HI$ for $\mathcal{C}(\mathcal{B})$ as follows:
    (a) For $v \in V_{\mathcal{C}(\mathcal{B})}$, set $Z_v = 1$ iff both copies $v_L^d \in V_L^d$ and $v_R^d \in V_R^d$ belong to $VC^d$.
    (b) For $v \in V_{\mathcal{C}(\mathcal{B})}$, set $Z_v = 0$ iff neither copy $v_L^d \in V_L^d$ or $v_R^d \in V_R^d$ belongs to $VC^d$.
    (c) For $v \in V_{\mathcal{C}(\mathcal{B})}$, set $Z_v = \frac{1}{2}$ iff exactly one of the two copies $v_L^d \in V_L^d$ or $v_R^d \in V_R^d$ belongs to $VC^d$.
(5) For each $i \in \{0, \frac{1}{2}, 1\}$, construct $VC_i = \{v \in V_{\mathcal{C}(\mathcal{B})} : Z_v = i\}$.
(6) For $v \in V_{\mathcal{C}(\mathcal{B})}$ that corresponds to a variable $X$ in $\mathcal{X}(\mathcal{B})$:
    (a) If $v \in VC_0$, add $X$ to $\mathcal{S}$ and give it the Boolean assignment 0.
    (b) If $v \in VC_1$, add $X$ to $\mathcal{S}$ and give it the Boolean assignment 1.
(7) Return the set $\mathcal{S}$.
```

where Z_i is a Boolean variable that represents whether or not v_i with weight w_i is chosen in the MWVC.

$$\begin{aligned} &\text{Minimize} \sum_{i=1}^{|V_{\mathcal{C}(\mathcal{B})}|} w_i Z_i \\ \forall\, v_i \in V_{\mathcal{C}(\mathcal{B})} :\ & Z_i \in \{0, 1\} \\ \forall\, (v_i, v_j) \in E_{\mathcal{C}(\mathcal{B})} :\ & Z_i + Z_j \geq 1 \end{aligned} \tag{1}$$

Now consider the Linear Programming (LP) relaxation where $Z_i \in [0, 1]$ is used instead of $Z_i \in \{0, 1\}$. It is well known that in any optimal solution to the LP relaxation, each Z_i will be such that $Z_i \in \{0, \frac{1}{2}, 1\}$. This special property of the MWVC problem makes it a *half-integral* problem (Nemhauser and Trotter 1974). Furthermore, because of the half-integral property, the LP itself can be solved as a maxflow problem as described below.

We build a bipartite graph $\mathcal{D}(\mathcal{B}) = \langle V_L^d, V_R^d, E^d \rangle$ as follows. For each $v \in V_{\mathcal{C}(\mathcal{B})}$, we create two corresponding nodes of the same weight in $\mathcal{D}(\mathcal{B})$: $v_L^d \in V_L^d$ and $v_R^d \in V_R^d$. For each edge $(u, v) \in E_{\mathcal{C}(\mathcal{B})}$, we create two corresponding edges in $\mathcal{D}(\mathcal{B})$: $(u_L^d, v_R^d) \in E^d$ and $(v_L^d, u_R^d) \in E^d$.

Since $\mathcal{D}(\mathcal{B})$ is a bipartite graph, the MWVC problem on it can be solved using the polynomial-time algorithm of (Goldberg and Tarjan 1988) that runs in time $O(|V_L^d \cup V_R^d||E^d|\log(|V_L^d \cup V_R^d|^2/|E^d|))$. Consider the MWVC VC^d for $\mathcal{D}(\mathcal{B})$. It defines a half-integral vertex cover for $\mathcal{C}(\mathcal{B})$ as follows. For a vertex $v \in V_{\mathcal{C}(\mathcal{B})}$, the corresponding variable Z_v is set to 1 if and only if both copies $v_L^d \in V_L^d$ and $v_R^d \in V_R^d$ are in VC^d. Z_v is set to 0 if and only if neither copy is in VC^d. Z_v is set to $\frac{1}{2}$ if and only if exactly one of the two copies is in VC^d.

Lemma 1 of (Chlebik and Chlebikova 2007) shows that such a half-integral vertex cover is indeed optimal for $\mathcal{C}(\mathcal{B})$ when it is derived from the MWVC of $\mathcal{D}(\mathcal{B})$. Furthermore, the Nemhauser-Trotter (NT) reduction (Nemhauser and Trotter 1974)[2] guarantees that if $VC_i = \{v \in V_{\mathcal{C}(\mathcal{B})} : Z_v = i\}$ for each $i \in \{0, \frac{1}{2}, 1\}$, then there exists an MWVC for $\mathcal{C}(\mathcal{B})$ that has all the vertices in VC_1 and none of the vertices in VC_0. As described in (Chlebik and Chlebikova 2007), this serves as a kernelization procedure for the MWVC problem. Any $Z_v = 0$ or 1 from the optimal half-integral solution can be committed to for the optimal integral solution as well. The kernel variables correspond to all those nodes for which $Z_v = \frac{1}{2}$—i.e., the set $VC_{\frac{1}{2}}$.

Algorithm 1 shows how to adapt this MWVC kernelization procedure to general Boolean WCSPs. The adaptation carefully heeds to the consequences of using auxiliary variables in the construction of the CCG for a given Boolean WCSP. The following theorem proves its correctness.

Theorem 1. *Algorithm 1 returns a set $\mathcal{S}$ such that $\mathcal{X}(\mathcal{B}) \setminus \mathcal{S}$ is a valid kernel.*

Proof. From Lemma 4 of (Kumar 2008a), we know that the MWVC of $\mathcal{C}(\mathcal{B})$ corresponds to an optimal solution of the Boolean WCSP $\mathcal{B}$. That is, if v is a node in $V_{\mathcal{C}(\mathcal{B})}$ that corresponds to a variable X in $\mathcal{X}(\mathcal{B})$, then the optimal value of X is equal to 1 when v is in the MWVC of $\mathcal{C}(\mathcal{B})$ and 0 otherwise. From Lemma 1 of (Chlebik and Chlebikova 2007), Steps 2-4 compute an optimal half-integral vertex cover for $\mathcal{C}(\mathcal{B})$. Now, from the NT reduction (Nemhauser and Trotter 1974), we know that there exists an MWVC for $\mathcal{C}(\mathcal{B})$ that has all the vertices in VC_1 and none of the vertices in VC_0

[2]also see page 3 of (Chlebik and Chlebikova 2007)

where VC_i for each $i \in \{0, \frac{1}{2}, 1\}$ is as defined in Step 5. Step 6 counts only a subset of the nodes in VC_0 and VC_1—namely, $VC_0 \cap \mathcal{X}(\mathcal{B})$ and $VC_1 \cap \mathcal{X}(\mathcal{B})$. Put together with the NT reduction, there now exists an MWVC for $\mathcal{C}(\mathcal{B})$ that has all the vertices in $VC_1 \cap \mathcal{X}(\mathcal{B})$ and none of the vertices in $VC_0 \cap \mathcal{X}(\mathcal{B})$. But once again, since the MWVC of $\mathcal{C}(\mathcal{B})$ corresponds to an optimal solution of $\mathcal{B}$, this is equivalent to saying that there exists an optimal solution of $\mathcal{B}$ that extends the commitments made for set $\mathcal{S}$ in Step 6. This proves that $\mathcal{X}(\mathcal{B}) \setminus \mathcal{S}$ is a valid kernel. □

One of the hallmarks of a good kernelization procedure is to be able to recognize tractable classes of Boolean WCSPs. This is much like the hallmark of a good SAT solver that not only works on general SAT instances but also solves the special case of 2-SAT instances in polynomial time. In the following theorem, we prove that the kernelization procedure of Algorithm 1 exhibits this desirable property.

Theorem 2. *Algorithm 1 produces an empty kernel for the tractable language of weighted constraints* $\mathcal{L}^{Boolean}_{bipartite}$.

Proof. We first assume that there is a unique MWVC for $\mathcal{C}(\mathcal{B})$.[3] From Theorem 6 of (Kumar 2008a), we know that the language $\mathcal{L}^{Boolean}_{bipartite}$—that contains Boolean weighted constraints with lifted bipartite graphical representations—produces a bipartite CCG $\mathcal{C}(\mathcal{B})$. From Step 2, it is easy to see that for a bipartite $\mathcal{C}(\mathcal{B})$, the bipartite graph $\mathcal{D}(\mathcal{B})$ has two disconnected complementary copies of $\mathcal{C}(\mathcal{B})$. VC^d in Step 3 therefore has two complementary copies of the same unique MWVC for $\mathcal{C}(\mathcal{B})$. In other words, for any node $v \in V_{\mathcal{C}(\mathcal{B})}$, $v^d_L \in VC^d$ if and only if $v^d_R \in VC^d$. In turn, this means that $VC_{\frac{1}{2}}$ is empty making the kernel empty as well. □

The above proof assumes the existence of a unique MWVC for $\mathcal{C}(\mathcal{B})$. Although this may not be true in general, it is easy to "simulate" it by perturbing the weights by tiny random amounts. A formal proof for this uses the *Isolation Lemma* (Valiant and Vazirani 1986; Mulmuley, Vazirani, and Vazirani 1987) but is skipped here for the sake of simplicity.

Generation of Lower and Upper Bounds

In this section, we show how the polynomial-time kernelization procedure of Algorithm 1 can also be used to produce intelligent lower and upper bounds for optimal solutions to combinatorial problems posed as Boolean WCSPs. The idea is to leverage the polynomial-time factor-2 approximability of the MWVC problem. For example, Clarkson's primal-dual algorithm provides a factor-2 polynomial-time approximation for the MWVC problem (Vazirani 2001).[4]

We note that a factor-2 approximation for the MWVC can also be derived from the kernelization procedure of Algorithm 1. In particular, consider the optimal half-integral vertex cover generated for $\mathcal{C}(\mathcal{B})$ in Step 4. An integral vertex cover AC of at most twice the cost can be generated from this by simply rounding up $Z_v = \frac{1}{2}$ to $Z_v = 1$. That is, $AC = VC_{\frac{1}{2}} \cup VC_1$, where $VC_{\frac{1}{2}}$ and VC_1 are as generated in Step 5. The cost of AC is at most twice the cost of the optimal half-integral vertex cover and therefore at most twice the cost of the MWVC.

We also note that while the MWVC of the CCG $\mathcal{C}(\mathcal{B})$ encodes an optimal solution to the Boolean WCSP $\mathcal{B}$, and although AC is a factor-2 approximation to this MWVC, the solution encoded by AC is not necessarily a factor-2 approximation to the original Boolean WCSP. This is because the machinery of the transformation—as explained in (Kumar 2008a)—introduces an offset $\kappa(\mathcal{B})$ such that the cost of the MWVC in $\mathcal{C}(\mathcal{B})$ is equal to the cost of the optimal solution in $\mathcal{B}$ plus $\kappa(\mathcal{B})$. The offset $\kappa(\mathcal{B})$ is computable in the transformation and depends on the particular instance $\mathcal{B}$.

Of course, $\kappa(\mathcal{B})$ cannot be 0 in all cases since that would contradict the inapproximability results of combinatorial problems—like the maximum independent set problem—that can be modeled as Boolean WCSPs. Nonetheless, useful bounds can be generated for the cost of the optimal solution in $\mathcal{B}$ using the vertex cover AC. Clearly, an upper bound is $cost(AC) - \kappa(\mathcal{B})$; and a lower bound is $\frac{cost(AC)}{2} - \kappa(\mathcal{B})$. In general, what we have, therefore, is a procedure that generates maxflow-based lower and upper bounds for the cost of the optimal solution to a combinatorial problem posed as a Boolean WCSP. Although this task is riddled with negative results in complexity theory, our procedure for generating bounds is useful as an instance-specific method that exploits the numerical structure of the given Boolean WCSP. Of course, the generation of the upper bound is constructive since AC encodes the corresponding approximate solution.

The Scope of Incremental Computation

In this section, we discuss the idea of incremental computation for solving Boolean WCSPs. Given two instances $\mathcal{B}_1$ and $\mathcal{B}_2$, the idea is to solve $\mathcal{B}_2$ optimally having solved $\mathcal{B}_1$ optimally. Of course, this raises the first question of how to characterize the "closeness" of $\mathcal{B}_1$ and $\mathcal{B}_2$ and then the second question of how to exploit it towards solving $\mathcal{B}_2$ more efficiently instead of solving it from scratch.

Several interesting procedures have been suggested for solving maxflow problems incrementally (Kumar 2003; Kumar and Gupta 2003). Intuitively, the idea is to exploit the fact that maxflow algorithms typically operate on a *residual graph* that enables easy backtracking. The residual graph that emerges as a byproduct of solving the first instance can be used as a starting point for the second instance (Kumar 2003). The "closeness" of the two instances can be captured in terms of how much flow remains to be pushed for the second instance starting from the residual graph of the first.

For tractable classes of Boolean WCSPs that have bipartite CCGs, the existing theory of incremental maxflow computation is very useful. This is because: (a) the MWVC computation in the CCGs corresponds to finding the optimal solution for the Boolean WCSPs; and (b) MWVC computation for bipartite graphs can be done using maxflow procedures (Goldberg and Tarjan 1988). Put together, it is easy to design incremental maxflow-based algorithms for Boolean

[3]Of course, this may not be true in general, but it is easy to obviate this assumption. See the running text just after the proof.

[4]A better approximation for the MWVC problem is unlikely since it is *Unique Games*-hard (Khot and Regev 2008).

WCSPs with bipartite CCGs. This establishes a framework where CCGs can not only be used to identify and solve tractable classes of Boolean WCSPs in polynomial time but can also be used to recompute their optimal solutions efficiently for incremental changes.

For general Boolean WCSPs, incremental computation is not readily applicable. This is because they are NP-hard, and if they were easily amenable to incremental computation, one could start with a trivial case, add the variables of a given WCSP one at a time, and build the optimal solution incrementally and efficiently. Nonetheless, the idea of a maxflow-based kernelization facilitates incremental computation to a certain extent. In particular, the kernelization itself can be made incremental since it is based on maxflow.

The overall procedure for solving a given Boolean WCSP is to: (a) construct its CCG; (b) kernelize the MWVC problem on its CCG using a maxflow computation; and (c) solve the kernel using search-based methods. While (a) and (b) are polynomial-time phases, (c) is not. But since (b) is based on maxflow, it can be made incremental as described above. This gives us an understanding of the scope of incremental computation in WCSPs for pruning large parts of the search space during kernelization. Furthermore, (c) poses a much smaller search space than for the original Boolean WCSP without kernelization.

Conclusions and Future Work

In this paper, we presented an algorithmic framework for kernelization of combinatorial problems posed as Boolean WCSPs. Our kernelization technique is a polynomial-time maxflow-based algorithm that fixes the optimal values of a subset of the variables in a preprocessing phase. It thereby reduces the original set of variables for which exhaustive search is required to all but a small kernel of variables. We showed that for the language of all Boolean weighted constraints that have lifted bipartite graphical representations, not only is the CCG bipartite—thereby establishing tractability of this language—but the kernelization procedure also yields an empty kernel.

For general Boolean WCSPs, we showed that the kernelization procedure not only fixes the optimal values of a subset of the variables but also produces intelligent lower and upper bounds for the cost of the optimal solution. The upper bound is generated constructively along with a candidate solution whose cost matches this bound.

We also used our algorithmic framework to understand the scope of incremental computation for Boolean WCSPs. While the kernelization procedure itself can be made incremental since it is akin to well-studied incremental maxflow algorithms, solution procedures for the kernel are unlikely to be made incremental with significant benefits. Once again, for the special case of weighted constraints with lifted bipartite graphical representations, the entire algorithm can be made incremental because the kernel is provably empty.

Future work is mostly directed towards implementation of algorithms that are based on the theory in this paper. Generalization of this theory from Boolean WCSPs to WCSPs is relatively straightforward. Parts of this generalization already appear in (Kumar 2008b).

References

Abu-Khzam, F.; Collins, R.; Fellows, M.; Langston, M.; Suters, W.; and Symons, C. 2004. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. In *Technical Report, University of Tennessee*.

Bessiere, C. 2008. Theoretical Analysis of Singleton Arc Consistency. In *Artificial Intelligence Journal*.

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaillie, G. 1996. Semiring-Based CSPs and Valued CSPs: Basic Properties and Comparison. In *Proceedings of Over-Constrained Systems*.

Boykov, V., and Funka-Lea, G. 2006. Graph Cuts and Efficient N-D Image Segmentation. In *International Journal of Computer Vision*.

Chlebik, M., and Chlebikova, J. 2007. Crown Reductions for the Minimum Weighted Vertex Cover Problem. In *Discrete Applied Mathematics*.

Cooper, M., and Schiex, T. 2004. Arc Consistency for Soft Constraints. In *Artificial Intelligence Journal*.

Edwards, S., and Anderson, P. 1975. Theory of Spin Glasses. In *Journal of Physics F: Metal Physics*.

Goldberg, A., and Tarjan, R. 1988. A New Approach to the Maximum-Flow Problem. In *Journal of the ACM*.

Khot, S., and Regev, O. 2008. Vertex Cover Might be Hard to Approximate within 2 - ϵ. In *Journal of Computer and System Sciences*.

Kolmogorov, V., and Zabih, R. 2002. Multi-Camera Scene Reconstruction via Graph Cuts. In *Proceedings of the European Conference on Computer Vision*.

Kumar, S., and Gupta, P. 2003. An Incremental Algorithm for the Maximum Flow Problem. In *Journal of Mathematical Modelling and Algorithms*.

Kumar, T. K. S. 2003. Incremental Computation of Resource-Envelopes in Producer-Consumer Models. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*.

Kumar, T. K. S. 2008a. A Framework for Hybrid Tractability Results in Boolean Weighted Constraint Satisfaction Problems. In *Proceedings of the Fourteenth International Conference on Principles and Practice of Constraint Programming*.

Kumar, T. K. S. 2008b. Lifting Techniques for Weighted Constraint Satisfaction Problems. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics*.

Larrosa, J. 2004. Solving Weighted CSP by Maintaining Arc Consistency. In *Artificial Intelligence Journal*.

Lecoutre, C. 2009. Constraint Networks: Techniques and Algorithms. In *ISTE/Wiley Publishers*.

Mulmuley, K.; Vazirani, U.; and Vazirani, V. 1987. Matching is as Easy as Matrix Inversion. In *Combinatorica 7 (1): 105–113*.

Nemhauser, G., and Trotter, L. 1974. Properties of Vertex Packing and Independence System Polyhedra. In *Mathematical Programming*.

Niedermeier, R. 2006. Invitation to Fixed-Parameter Algorithms. In *Oxford University Press*.

Rother, C.; Kolmogorov, V.; and Blake, A. 2004. Grabcut: Interactive Foreground Extraction Using Iterated Graph Cuts. In *ACM Transactions on Graphics*.

Russell, S., and Norvig, P. 2003. Artificial Intelligence: A Modern Approach. In *Upper Saddle River, New Jersey: Prentice Hall*.

Valiant, L., and Vazirani, V. 1986. NP is as Easy as Detecting Unique Solutions. In *Theoretical Computer Science*.

Vazirani, V. 2001. Approximation Algorithms. In *Springer-Verlag Publishers*.

Zytnicki, M. 2009. Bounds Arc Consistency for Weighted CSPs. In *Journal of Artificial Intelligence Research*.