Propagation via Kernelization: the Vertex Cover Constraint

Clément Carbonnel and Emmanuel Hebrard

LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France

Abstract. The technique of kernelization consists in extracting, from an instance of a problem, an essentially equivalent instance whose size is bounded in a parameter k. Besides being the basis for efficient parameterized algorithms, this method also provides a wealth of information to reason about in the context of constraint programming. We study the use of kernelization for designing propagators through the example of the Vertex Cover constraint. Since the classic kernelization rules often correspond to dominance rather than consistency, we introduce the notion of "loss-less" kernel. While our preliminary experimental results show the potential of the approach, they also show some of its limits. In particular, this method is more effective for vertex covers of large and sparse graphs, as they tend to have, relatively, smaller kernels.

1 Introduction

The fact that there is virtually no restriction on the algorithms used to reason about each constraint was critical to the success of constraint programming. For instance, efficient algorithms from matching and flow theory [2, 14] were adapted as *propagation* algorithms [16, 18] and subsequently lead to a number of successful applications. NP-hard constraints, however, are often simply decomposed. Doing so may significantly hinder the reasoning made possible by the knowledge on the structure of the problem. For instance, finding a support for the NVALUE constraint is NP-hard, yet enforcing some incomplete propagation rules for this constraint has been shown to be an effective approach [5, 10], compared to decomposing it, or enforcing bound consistency [3].

The concept of parameterized complexity is very promising in the context of propagating NP-hard constraints. A study of the parameterized complexity of global constraints [4], and of their pertinent parameters, showed that they were a fertile ground for this technique. For instance, a kernelization of the NVALUE constraint was introduced in [12], yielding an FPT consistency algorithm. A kernel is an equivalent instance of a problem whose size is bounded in a parameter k. If a problem has a polynomial-time computable kernel, then it is FPT since brute-force search on the kernel can be done in time $O^*(f(k))$ for some computable function f. Moreover, kernelization techniques can provide useful information about suboptimal and/or compulsory choices, which can be used to propagate. In this paper we consider the example of the *vertex cover* problem, where we want to find a set of at most k vertices S of a graph $G = (V, E)$ such that every edge of G is incident to at least one vertex in S. This problem is a long-time

favourite of the parameterized complexity community and a number of different kernelization rules have been proposed, along with very efficient FPT algorithms (the most recent being the $O(1.2738^k + k|V|)$ algorithm by Chen, Kanj and Xia [7]).

Since the complement of a minimum vertex cover is a maximum independent set, a VERTEXCOVER constraint can also be used to model variants of the maximum independent set and maximum clique problems with side constraints modulo straightforward modeling tweaks. Among these three equivalent problems, vertex cover offers the greatest variety of pruning techniques and is therefore the most natural choice for the definition of a global constraint. Through this example, we highlight the "triple" value of kernelization in the context of constraint programming:

First, some kernelization rules are, or can be generalized to, filtering rules. Since the strongest kernelization techniques rely on dominance they cannot be used directly for filtering. Therefore, we introduce the notion of *loss-less* kernelization which preserves all solutions and can thus be used in the context of constraint propagation. Moreover, we show that we can use a more powerful form of kernel, the so-called *rigid crowns* to effectively filter the constraint when the lower bound on the size of the vertex cover is tight. We discuss the various kernelization techniques for this problem in Section 3.

Second, even when it cannot be used to filter the domain, a kernel can be sufficiently small to speed up lower bound computation, or to find a "witness solution" and sometimes an exact lower bound. We also show that such a support can be used to obtain stronger filtering. We introduce a propagation algorithm based on these observations in Section 4. Along this line, the kernel could also be used to guide search, either using the witness solution or the dominance relations on variable assignments.

Third, because a kernel garantees a size at most $f(k)$ for a parameter k, one can efficiently estimate the likelihood that these rules will indeed reduce the instance. We report experimental results on a variant of the vertex cover problem in Section 5. These experiments show that, as expected, kernelization techniques perform better when the parameter is small. However, we observe that the overhead is manageable, even in unfavorable cases. Moreover, one could dynamically choose whether costly methods should be applied by comparing the value of the parameter k (in our case, the upper bound of the variable standing for the size of the cover) to the input size.

2 Background and Notations

An *undirected graph* is an ordered pair $G = (V, E)$ where V is a set of vertices and E is a set of edges, that is, pairs in V. We denote the *neighborhood* $N(v) = \{u \mid \{v, u\} \in E\}$ of a vertex v, its *closed neighborhood* $N^+(v) = N(v) \cup \{v\}$ and $N(W) = \bigcup_{v \in W} N(v)$. The *subgraph* of $G = (V, E)$ induced by a subset of vertices W is denoted $G[W] = (W, 2^W \cap E)$. An *independent set* is a set $I \subseteq V$ such that no pair of elements in I is in E. A *clique* is a set $C \subseteq V$ such that every pair of elements in C is in E. A *clique cover* T of a graph $G = (V, E)$ is a collection of disjoint cliques such that $\bigcup_{C \in T} C = V$. A *matching* is a subset of pairwise disjoint edges. A *vertex cover* of G is a set $S \subseteq V$ such that every edge $e \in E$ is incident to at least one vertex in S, i.e., $S \cap e \neq \emptyset$. The *minimum vertex cover problem* consists in finding a vertex cover of minimum size. Its decision version is NP-complete [11].

The standard algorithm for solving this problem is a simple branch and bound procedure. There are several bounds that one can use, in this paper we consider the minimum clique cover of the graph (or, equivalently, a coloring of its complement). Given a clique cover T of a graph $G = (V, E)$, we know that all but one vertices in each clique of T must be in any vertex cover of G. Therefore, $|V| - |T|$ is a lower bound of the size of the minimum vertex cover of G. The algorithm branches by adding a vertex to the cover (left branch) or adding its neighborhood to the cover (right branch).

A *constraint* is a predicate over one or several variables. In this paper we consider the vertex cover problem as a constraint over two variables: an *integer* variable K to represent the bound on the size of the vertex cover, and a *set variable* S to represent the cover itself. The former takes integer values in a *domain* $\mathcal{D}(K)$ which minimum and maximum values are denoted $\underline{K}$ and $\overline{K}$, respectively. The latter takes its values in the sets that are supersets of a *lower bound* $\underline{S}$ and subsets of an *upper bound* $\overline{S}$. Moreover, the domain of a set variable is also often constrained by its cardinality given by an integer variable $|S|$. We consider a constraint on these two variables and whose predicate is the vertex cover problem on the graph $G = (V, E)$ given as a parameter:

Definition 1 (VERTEXCOVER **constraint).**
$\text{VERTEXCOVER}[G](K, S) \iff |S| \leq K \;\&\; \forall \{v, u\} \in E,\ v \in S \vee u \in S$

A *bound support* for this constraint is a solution of the VERTEXCOVER problem. Since enforcing *bound consistency* would entail proving the existence of two bound supports for each element in $\overline{S} \setminus \underline{S}$ and one for the lower bound of K, there is no polynomial algorithm unless P=NP. In this paper we consider pruning rules that are not complete with respect to usual notions of consistencies.

3 Kernelization as a Propagation Technique

3.1 Standard Kernelization

A problem is *parameterized* if each instance x is paired with a nonnegative integer k, and a parameterized problem is *fixed-parameter tractable* (FPT) if it can be solved in time $O(|x|^{O(1)} f(k))$ for some function f. A *kernelization algorithm* takes as input a parameterized instance (x, k) and creates in polynomial time a parameterized instance (x', k') of the same problem, called the *kernel*, such that

(i) (x', k') is satisfiable if and only if (x, k) is satisfiable;
(ii) $|x'| \leq g(k)$ for some computable function g, and
(iii) $k' \leq h(k)$ for some computable function h.

While this formal definition does not guarantee that the kernel is a subinstance of (x, k), in graph theory kernelization algorithms often operate by applying a succession of dominance rules to eliminate vertices or edges from the graph. In the case of vertex cover, the simplest dominance rule is the *Buss rule*: if a vertex v has at least $k+1$ neighbors, then v belongs to every vertex cover of size at most k; we can therefore remove v from the graph and reduce k by one. Applying this rule until a fixed point yields an elementary kernel that contains at most k^2 edges and $2k^2$ non isolated vertices [6]. A

more refined kernelization algorithm works using structures called crowns. A *crown* of a graph $G = (V, E)$ is a partition (H, W, I) of V such that

(i) I is an independent set;
(ii) There is no edge between I and H, and
(iii) There is a matching M between W and I of size $|W|$.

Every vertex cover of $G[W \cup I]$ has to be of size at least $|W|$ because of the matching M. Since I is an independent set, taking the vertices of W over those of I into the vertex cover is always a sound choice: they would cover all the edges between W and I at minimum cost and as many edges in $G[H \cup W]$ as possible. A simple polynomial-time algorithm that finds a crown greedily from a maximal matching already leaves an instance $G[H]$ with at most $3k$ vertices [1]. A stronger method using linear programming yields a (presumably optimal) kernel of size $2k$ [15].

3.2 Loss-less Kernelization

The strongest kernelization rules correspond to dominance relations rather than inconsistencies. However, the Buss rule actually detects inconsistencies, that is, vertices that must be in the cover. We call this type of rules *loss-less* as they do not remove solutions. We can extend this line of reasoning by considering rules that do not remove solutions close to the optimum: for the VERTEXCOVER constraint, the variable K is likely to be minimized and the situation where all solutions are close to the optimum will inevitably arise. This can be formalized in the context of *subset minimization problems*, which ask for a subset S with some property π of a given universe U such that $|S| \leq k$. In the next definition we denote by $\texttt{opt}$ the cardinality of a minimum-size solution.

Definition 2. *Given an integer z and a subset minimization problem parameterized by solution size k, a z-loss-less kernel is a partition (H, F, R, I) of the universe U where*

- *F is a set of* forced *items, included in every solution of size at most $\texttt{opt}+z$;*
- *R is a set of* restricted *items, intersecting with no solution of size at most $\texttt{opt}+z$;*
- *H is a residual problem, whose size is bounded by a function in k and*
- *I is a set of* indifferent *elements, i.e., if $i \in I$, then ϕ is a solution of size at most $k-1$ if and only if $\phi \cup i$ is a solution.*

An ∞-loss-less kernel is simply said to be *loss-less*. The Buss kernel is a loss-less kernel for vertex cover that never puts any vertices in R (F contains vertices of degree strictly greater than k, and I contains isolated vertices). In the case of vertex cover, the set R is always empty unless $z = 0$. Note that *loss-free* kernels introduced in the context of backdoors [17] are different since they only preserve *minimal* solutions; for subset minimization problems those kernels are called *full kernels* [9].

A kernel for vertex cover that preserves all minimum-size solutions has been introduced in [8]. In our terminology, this corresponds to a 0-loss-less kernel. Interestingly, this kernelization is based on a special type of crown reduction but yields a kernel of size $2k$ (matching the best known bound for standard kernelization). The idea is to consider only crowns (H, W, I) such that W is the *only* minimum-size vertex cover of

$G[W \cup I]$, as for this kind of crown vertices of W are always a *strictly* better choice that those of I. Those crowns are said *rigid*. The authors present a polynomial-time algorithm that finds the (unique) rigid crown (H, W, I) such that H is rigid crown free and has size at most $2k$. Their algorithm works as follows. First, build from $G = (V, E)$ the graph B_G with two vertices v_l, v_r for every $v \in V$ and two edges $\{v_l, u_r\}, \{u_l, v_r\}$ for every edge $\{v, u\} \in E$. Compute a maximum matching M of B_G (which can be done in polynomial time via the Hopcroft-Karp algorithm [14]). Then, if D is the set of all vertices in B_G that are reachable from unmatched vertices via M-alternating paths of even length, a vertex v in G belongs to the independent set I of the rigid crown if and only if v_l and v_r belong to D. This algorithm is well suited to constraint propagation as bipartite matching algorithms based on augmenting paths are efficient and incremental.

3.3 Witness Pruning

Last, even if the standard kernel uses dominance relations, it can indirectly be used for pruning. By reducing the size of the problem it often makes it possible to find an optimal vertex cover relatively efficiently. This vertex cover gives a valid (and maximal) lower bound. Moreover, given an optimal cover S we can find inconsistent values by asserting that some vertices must be in any cover of a given size.

Theorem 1. *if S is an optimal vertex cover of $G = (V, E)$ such that there exists $v \in S, J \subseteq N(v) \setminus S$ with $N(J) \subseteq N^+(v)$ then any vertex cover of G either contains v or at least $|S| + |J| - 1$ vertices.*

Proof. Let k be an upper bound on the size of the vertex cover, $v \in S$ be a vertex in an optimal vertex cover S. Consider $J \subseteq N(v) \setminus S$ such that $N(J) \subseteq N^+(v)$. Suppose there exists a vertex cover S' such that $|S'| < |S| + |J| - 1$ and $v \notin S'$. S' must contain every node in $N(v)$ and hence in J. However, we can build a vertex cover of size at most $|S| - 1$ by replacing J by v, since $V \setminus S$ and thus J are independent sets. □

If we can manage to find a minimum vertex cover S, for instance when the kernel is small enough so that it can be explored exhaustively, Theorem 1 entails a pruning rule. If we find a vertex $v \in S$ and a set $J \in N(v) \setminus S$ with $N(J) \subseteq N^+(v)$ and $|J| > k - |S|$ then we know that v must be in all vertex covers of size $\leq k$.

4 A Propagation Algorithm for VERTEXCOVER

In this section we give the skeleton of a propagation algorithm for the VERTEXCOVER constraint based on the techniques discussed in Section 3.

Algorithm 1 takes as input the set variable S standing for the vertex cover, an integer variable K standing for the cardinality of the vertex cover, and three parameters: the graph $G = (V, E)$, an integer λ, and a "witness" vertex cover ω initialised to V.

The pruning in Line 1 is a straightforward application of the definition: the neighborhood of vertices not in the cover must be in the cover. Then, in Line 2, we apply the ∞-loss-less kernelization (Buss rule) described in Section 3.2 yielding a pair with a residual graph H^r and a set of nodes F^r that must be in the cover.

Algorithm 1: `PropagateVertexCover`$(S, K, G = (V, E), \lambda, \omega)$

1 $\underline{S} \leftarrow \underline{S} \cup N(V \setminus \overline{S})$;
2 $H^r, F^r \leftarrow$ `BussKernel`$(G[\overline{S} \setminus \underline{S}])$;
3 **if** $\omega \not\subseteq \overline{S} \ \vee \ |\omega \cup \underline{S}| \geq \overline{K}$ **then**
4 $\quad H^k, W^k \leftarrow$ `Kernel`(H^r);
5 $\quad$ **if** $\lambda > 0$ **then** $\omega \leftarrow F^r \cup W^k \cup$ `VertexCover`(H^k, λ);
6 $\quad$ **if** ω *is optimal* **then** $\underline{K} \leftarrow |\omega|$;
7 $\quad$ **else** $\underline{K} \leftarrow \max(\underline{K}, |F^r| + |F^k| +$ `LowerBound`$(H^k))$;
8 **if** $\underline{K} = \overline{K}$ **then**
9 $\quad H^r, F^r, R^r \leftarrow$ `RigidKernel`$(G[\overline{S} \setminus \underline{S}])$;
10 $\quad \overline{S} \leftarrow \overline{S} \setminus R^r$;
11 **else if** ω *is optimal* & $\overline{K} - \underline{K} \leq 2$ **then** $\underline{S}, \overline{S} \leftarrow$ `WitnessPruning`(G, ω);
12 $\underline{S} \leftarrow \underline{S} \cup F^r$;

Next, if Condition 3 fails, there exists a vertex cover ($\omega \cup \underline{S}$) of size strictly less than $\overline{K}$. As a result, the pruning from rigid crowns cannot apply. When the cover witness is not valid, we compute, in Line 4, a standard kernel with the procedure `Kernel`(G) using crowns, as explained in Section 3.1. We then use this kernel to compute, in Line 5, a new witness using the procedure `VertexCover`(G, λ) which is the standard brute-force algorithm described in Section 2. We stop the procedure when we find a vertex cover whose size is stricly smaller than the current upper bound, or when the search limit of λ, in number of nodes explored by the branch & bound procedure, is reached. In the first case, we know that the lower bound cannot be tight hence the constraint cannot fail nor prune further than the loss-less kernel. The second stopping condition is simply used to control the amount of time spent within the brute-force procedure.

If the call to the brute-force procedure was complete, we can conclude that the witness cover is optimal and therefore a valid lower bound (Line 6). Otherwise, we simply use the lower bound computed at the root node by `VertexCover`, denoted `LowerBound` in Line 7. If the lower bound is tight, then we can apply the pruning from rigid crowns as described in Section 3.2. Algorithm `RigidKernel` returns a triple H^r, F^r, R^r of residual, forced and restricted vertices, respectively. Finally we apply a restriction to pairs of the pruning corresponding to Theorem 1 in Line 11, and apply the pruning on the lower bound of S corresponding to the forced nodes computed by `BussKernel` and/or `RigidKernel`.

5 Experimental Evaluation

We experimentally evaluated our propagation algorithm on the "balanced vertex cover problem". We want to find a minimum vertex cover which is balanced according to a partition of the vertices. For instance, the vertex cover may represent a set of machines to shut down in a network so that all communications are interrupted. In this case, one might want to avoid shutting down too many machines of the same type, or same

client, or in charge of the same service, etc. By varying the degree of balance we can control the similarity of the problem to pure minimum vertex cover. We used a range of graphs from the `dimacs` and `snap` repositories. For each graph $G = (V, E)$, we post a VERTEXCOVER constraint on the set variable $\emptyset \subseteq S \subseteq V$.

Table 1: Comparison of approaches on the "Balanced Vertex Cover" problem.

		Decomposition				Clique Cover				Kernel Pruning				Kernel & witness				VERTEXCOVER			
		#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd
balancing constraint: tight																					
3	kel	2	2.00	9.7	0.4M	2	2.00	10.6	0.2M	2	2.00	9.1	0.2M	2	2.00	26.6	0.1M	2	2.00	41.0	0.1M
15	p_h	12	5.73	8.6	0.5M	10	5.20	15.6	1.1M	11	5.20	11.2	0.6M	11	4.67	27.7	0.4M	11	4.67	28.8	0.4M
12	bro	9	3.67	0.1	11K	9	3.67	0.1	4K	9	3.67	0.1	3K	9	3.67	0.1	2K	9	3.67	0.2	2K
4	joh	1	0.00	0.1	10K	1	0.00	0.0	1K	1	0.00	0.0	1K	1	0.00	0.0	971	1	0.00	0.0	937
15	san	15	10.87	12.2	1.8M	11	9.80	13.3	1.9M	11	9.80	13.7	1.1M	11	9.80	10.8	0.6M	11	9.80	12.4	0.6M
7	c-f	3	10.29	0.2	9K	3	10.29	0.2	18K	3	10.29	0.1	7K	3	10.29	0.1	7K	3	10.29	0.1	7K
6	ham	4	9.00	26.3	2.2M	3	9.00	3.1	0.3M	3	9.00	3.4	0.2M	3	9.00	5.1	0.2M	3	9.00	5.1	0.2M
32	gra	29	40.47	24.6	2.5M	28	40.47	19.8	3.5M	28	39.22	17.6	2.0M	28	40.47	18.9	1.5M	28	41.22	9.8	0.5M
4	man	3	91.00	1.1	33K	3	91.00	1.1	51K	3	91.00	0.9	31K	3	91.00	1.4	31K	3	91.00	1.5	30K
5	mul	5	8.40	7.2	1.8M	4	7.60	41.4	6.3M	3	7.60	24.6	2.1M	3	7.60	25.1	2.1M	3	7.60	19.2	1.7M
3	fps	3	105.00	0.1	7K	3	103.67	40.5	4.2M	3	103.67	56.0	3.2M	3	103.67	61.0	3.2M	3	103.67	14.9	0.8M
3	zer	3	44.67	11.9	2.6M	3	44.67	11.4	1.6M	3	44.67	14.7	1.4M	3	44.67	13.9	1.4M	3	44.67	8.2	0.9M
3	ini	3	191.33	57.5	6.1M	3	191.33	72.7	6.1M	3	191.33	82.3	6.1M	3	191.33	82.6	6.1M	3	191.33	82.6	6.1M
5	p2p	5	38.60	1.0	8K	5	22.80	36.1	23K	2	11.80	2.8	11K	2	11.80	3.1	11K	2	11.80	3.4	11K
5	ca-	5	14.40	31.6	0.2M	4	9.00	35.6	0.2M	4	1.80	99.3	0.2M	3	2.60	102.3	0.2M	3	1.80	96.1	0.2M
balancing constraint: medium																					
3	kel	2	1.67	24.1	1.2M	2	0.67	35.9	1.0M	2	0.67	54.7	1.0M	2	0.00	32.8	2K	2	0.00	32.1	2K
15	p_h	12	3.07	21.5	1.2M	10	1.27	24.3	0.7M	11	1.27	34.4	0.6M	10	0.87	18.6	60K	10	0.87	18.8	59K
12	bro	9	0.83	15.6	1.9M	8	0.17	17.8	1.0M	8	0.17	25.4	1.0M	8	0.17	23.9	451	8	0.17	22.2	450
4	joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15	san	15	8.33	35.6	5.0M	7	2.67	30.4	2.4M	7	2.73	33.5	1.6M	7	1.53	42.8	0.4M	7	1.53	43.1	0.4M
7	c-f	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40
6	ham	4	4.67	0.2	53K	2	4.67	0.0	1K	2	4.67	0.0	360	2	4.67	0.0	359	2	4.67	0.0	359
32	gra	26	29.28	32.8	2.7M	22	26.50	21.9	2.2M	22	24.44	23.8	1.4M	22	24.25	21.5	0.9M	22	24.25	23.0	0.9M
4	man	3	89.00	29.3	1.3M	3	88.75	44.6	1.6M	3	88.75	21.1	0.6M	2	88.50	29.6	0.6M	2	88.50	33.4	0.6M
5	mul	5	1.20	0.3	61K	1	1.20	0.0	1K	1	1.20	0.0	682	1	1.20	0.0	682	1	1.20	0.0	560
3	fps	3	103.00	0.0	250	1	102.67	0.0	429	1	102.67	0.0	404	1	102.67	0.0	404	1	102.67	0.0	261
3	zer	3	3.33	37.4	8.2M	1	3.00	11.6	1.5M	1	3.00	25.4	1.5M	1	3.00	14.5	1.5M	1	3.00	14.5	1.5M
3	ini	3	189.00	0.6	65K	1	189.00	0.0	4K	1	189.00	0.0	3K	1	189.00	0.0	3K	1	189.00	0.0	3K
5	p2p	5	35.40	1.0	8K	5	15.80	38.3	26K	1	4.40	3.3	11K	1	4.40	3.5	11K	1	4.40	3.8	11K
5	ca-	5	14.40	0.7	5K	4	8.60	2.3	8K	3	0.40	72.6	18K	2	1.20	74.1	16K	2	0.40	64.0	15K
balancing constraint: loose																					
3	kel	2	1.67	43.3	1.8M	2	0.67	20.1	0.6M	2	0.67	30.4	0.6M	2	0.00	27.7	447	2	0.00	28.0	419
15	p_h	12	2.40	20.6	1.2M	10	0.73	32.1	1.0M	11	0.73	47.1	1.0M	9	0.27	18.0	3K	9	0.27	18.0	3K
12	bro	9	0.67	16.2	1.9M	8	0.00	10.6	0.7M	8	0.00	15.8	0.7M	8	0.00	13.6	264	8	0.00	13.6	264
4	joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15	san	15	8.20	28.1	4.0M	7	2.13	40.9	2.3M	7	2.27	29.6	1.4M	5	0.27	36.8	3K	5	0.27	36.8	3K
7	c-f	2	0.71	0.0	1K	0	0.00	0.6	98K	0	0.00	0.1	7K	0	0.00	0.2	7K	0	0.00	0.2	7K
6	ham	4	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118
32	gra	23	18.97	29.4	1.9M	17	12.84	12.9	0.8M	17	12.06	15.6	0.5M	17	11.56	14.5	66K	17	11.50	17.3	0.1M
4	man	3	88.50	38.1	0.8M	3	88.50	8.5	0.3M	3	87.75	30.6	0.8M	2	87.00	43.5	0.8M	2	86.50	52.4	0.8M
5	mul	5	0.00	0.0	93	0	0.00	0.0	92	0	0.00	0.0	91	0	0.00	0.0	91	0	0.00	0.0	91
3	fps	3	1.67	1.1	0.1M	1	1.00	12.6	1.0M	1	1.00	13.4	0.5M	1	1.00	13.9	0.5M	1	0.67	53.5	2.1M
3	zer	3	2.00	0.2	48K	0	1.00	2.6	0.4M	0	1.00	2.7	0.2M	0	1.00	2.6	0.2M	0	1.00	0.6	58K
3	ini	3	0.67	6.9	0.5M	0	0.00	20.9	1.3M	0	0.00	28.9	1.0M	0	0.00	31.9	1.0M	0	0.00	11.6	0.5M
5	p2p	5	33.00	1.0	8K	5	13.40	38.3	26K	1	2.00	3.4	11K	1	2.00	3.5	11K	1	2.00	3.9	11K
5	ca-	5	14.40	0.7	5K	4	8.60	2.5	8K	3	0.40	74.0	18K	2	1.20	73.9	16K	2	0.20	69.2	16K

Then, we compute (uniformly at random) a balanced 4-partition $\{s_1, s_2, s_3, s_4\}$ of the vertices and we post the following constraints: $\max(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) - \min(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) \leq b$. For each graph instance, we generated 3 instances for $b \in \{0, 4, 8\}$ denoted "tight", "medium" and "loose" respectively. However, the classes `p2p` and `ca-` are much too large for these values to make sense. In this case we used three ratios $0.007, 0.008$ and 0.009 of the number of nodes instead.

We compared 5 methods, all implemented in Mistral [13] and ran on CORE I7 processors with a time limit of 5 minutes:

Decomposition is a simple decomposition in 2-clauses and a cardinality constraint. **Clique Cover** uses only Buss kernelization and the clique cover lower bound. It corresponds to non-colored lines in Algorithm 1. The witness is initialised to V and never changes, and Line 4 is replaced by a simple identity $H^k \leftarrow H^r$. **Kernel Pruning** uses kernelization, but no witness cover. It corresponds to Algorithm 1 minus the instruction line 11, with λ set to 0. **Kernel & witness** uses kernelization, and the witness cover for the lower bound $\underline{K}$. It corresponds to Algorithm 1 minus the instruction line 11, with λ set to 5000. VERTEXCOVER is Algorithm 1 with λ set to 5000.

The results of these experiments are reported in Table 1. Instances are clustered by classes whose cardinality is given in the first column. These classes are ordered from top to bottom by decreasing ratio of minimum vertex cover size over number of nodes. We report four values for each class and each method: '#s' is the number of instances of the class that were not solved to optimality, 'gap' is the average gap w.r.t. the smallest vertex cover found, 'cpu' and '#nd' are mean CPU time in seconds and number of nodes visited, respectively, until finding the best solution. Notice that CPU times and number of nodes are then only comparable when the objective values (gaps) are equal. We color the tuples $\langle$#s, gap, cpu, #nd$\rangle$ that are lexicographically minimum for each class[1].

Instances with same value of b are grouped in the same sub-table. The "shift" of colored cells from left to right when going from top to bottom in each subtable was to be expected since the kernelization is more effective on instances with small vertex cover. It should be noted that many instances from the `dimacs` repository are extremely adverse to our method as they tend to have very large vertex covers. On the other hand, kernelization is very effective on large graphs from `snap`.

We can also observe another shift of colored cells from left to right when moving to a subtable to the next. This was also an expected outcome since the pruning on this constraint becomes more prevalent when the problem is closer to pure vertex cover.

Last, we can observe that every reasoning step (0-loss-less kernels, lower bound from the witness and pruning from the witness) improves the overall results.

6 Conclusion

We have shown that the kernelization techniques can be an effective way to reason about NP-hard constraints that are fixed parameter tractable. In order to design a propagation algorithm we introduced the notion of loss-less kernel and outlined several ways to benefit from a small kernel. Our experimental evaluation on the VERTEXCOVER constraint shows the promise of this approach.

[1] With a "tolerance" of 1s and 1% nodes.

References

1. Faisal N Abu-Khzam, Michael R Fellows, Michael A Langston, and W Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.
2. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.
3. Nicolas Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *CP*, pages 211–224, 2001.
4. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh. The Parameterized Complexity of Global Constraints. In *AAAI*, pages 235–240, 2008.
5. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4):271–293, 2006.
6. Jonathan F Buss and Judy Goldsmith. Nondeterminism within pˆ*. *SIAM Journal on Computing*, 22(3):560–572, 1993.
7. J. Chen, I. A. Kanj, and G. Xia. Improved Parameterized Upper Bounds for Vertex Cover. In *MFCS*, pages 238–249, 2006.
8. Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.
9. Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.
10. J.-G. Fages and T. Lapègue. Filtering AtMostNValue with Difference Constraints: Application to the Shift Minimisation Personnel Task Scheduling Problem. In *CP*, pages 63–79, 2013.
11. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
12. S. Gaspers and S. Szeider. Kernels for Global Constraints. In *IJCAI*, pages 540–545, 2011.
13. E. Hebrard. Mistral, a Constraint Satisfaction Library. In *The Third International CSP Solver Competition*, pages 31–40, 2008.
14. J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.
15. George L Nemhauser and Leslie E Trotter Jr. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.
16. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *AAAI*, pages 362–367, 1994.
17. Marko Samer and Stefan Szeider. Backdoor trees. In *AAAI*, volume 8, pages 13–17, 2008.
18. W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming: Flow-Based Soft Global Constraints. *Journal of Heuristics*, 12(4-5):347–373, 2006.