Constraint Composite Graph-Based Weighted CSP Solvers: An Empirical Study

Orazio Rillo and **T. K. Satish Kumar**[1,2,3,4,5]
[1]Department of Computer Science
[2]Department of Physics and Astronomy
[3]Department of Industrial and Systems Engineering
[4]Information Sciences Institute
[5]University of Southern California
oraziorillo@gmail.com, tkskwork@gmail.com

Abstract

The Weighted Constraint Satisfaction Problem (WCSP) is a very expressive framework for optimization problems. The Constraint Composite Graph (CCG) is a graphical representation of a given (Boolean) WCSP that facilitates its reduction to a Minimum Weighted Vertex Cover (MWVC) problem by introducing intelligently chosen auxiliary variables. It also enables kernelization: a maxflow procedure used to fix the optimal values of a subset of the variables before initiating search. In this paper, we present some CCG-based WCSP solvers and compare their performance against toulbar2, a state-of-the-art WCSP solver, on a variety of benchmark instances. We also study the effectiveness of kernelization.

Introduction

A Weighted Constraint Satisfaction Problem (WCSP) is defined as a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, X_2 \dots X_N\}$ is a set of *variables*, $\mathcal{D} = \{D_1, D_2 \dots D_N\}$ is the set of their respective discrete-valued *domains*, and $\mathcal{C} = \{C_1, C_2 \dots C_M\}$ is a set of *weighted constraints* on subsets of the variables. Each constraint C_j is defined on a certain subset $S_j \subseteq \mathcal{X}$, where $|S_j|$ is referred to as the *arity* of C_j. C_j specifies a non-negative *weight* for every possible combination of values of the variables in S_j. An *optimal assignment*, also called an *optimal solution*, is an assignment of values to all the variables from their respective domains such that the total weight, i.e., the sum of the weights locally induced by each weighted constraint, is minimized.

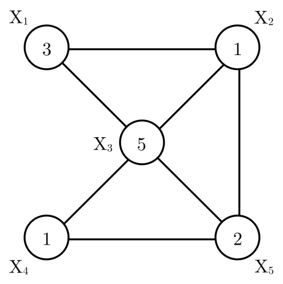
Boolean WCSPs are WCSPs in which all the variables are Boolean, i.e., $|D_i| = 2$ for all $i \in \{1, 2 \dots N\}$. A binary weighted constraint is a weighted constraint that involves only two variables. Boolean WCSPs have the same representational power as general WCSPs when non-binary weighted constraints are allowed (Xu et al. 2020).

While WCSPs and Boolean WCSPs are NP-hard to solve optimally, WCSP solvers can benefit from recognizing and exploiting "structure" at an instance level. WCSP instances can exhibit two kinds of structure: the *graphical structure* and the *numerical structure*. The former represents which variables interact with each other; the latter represents how they interact with each other. The Constraint Composite Graph (CCG), first introduced in (Kumar 2008), enables the exploitation of both kinds of structure simultaneously. The CCG is an automatically constructed graphical representation of a Boolean WCSP with vertices corresponding to each of the original variables and some intelligently chosen auxiliary variables. Solving the Minimum Weighted Vertex Cover (MWVC) problem on the CCG of a Boolean WCSP is equivalent to solving the original WCSP (Kumar 2008).

Although the CCG introduces auxiliary variables, it enables the Nemhauser-Trotter (NT) reduction, a maxflow-based kernelization procedure that reduces the number of variables in the substrate MWVC problem (Kumar 2016). The CCG also enables a new generation of WCSP solvers that can take advantage of the developments in MWVC solvers. In this paper, we present two such CCG-based solvers. The first utilizes Gurobi (Gurobi 2022), a state-of-the-art Integer Linear Programming (ILP) solver, to solve the substrate MWVC problem. The second utilizes Fast-WVC (Cai et al. 2019), a state-of-the-art MWVC solver based on local search. Both these solvers can invoke the kernelization procedure in a preprocessing phase.

CCG-based WCSP solvers have the potential to simultaneously exploit both the graphical and the numerical structures of a WCSP instance. Some evidence of this is presented in (Xu, Koenig, and Kumar 2017), which shows that the CCG-based ILP encoding of Boolean WCSPs is more congenial to ILP solvers (such as Gurobi) compared to their direct ILP encoding. However, a more thorough investigation of the advantages of CCG-based WCSP solvers over other state-of-the-art WCSP solvers is required. In particular, toulbar2, an open-source WCSP solver, is the culmination of many other algorithmic techniques and, therefore, a good competitor. In fact, toulbar2 is the winner of several medals in recent competitions such as UAI 2022, XCSP3 2022, UAI 2014, UAI 2010, CPAI08, and UAI 2008.

In this paper, we present experimental results, rendered as graphical plots, comparing our CCG-based solvers against toulbar2. The plots portray the performance of the solvers in finding optimal and bounded suboptimal solutions for the given problem instances. We present an analysis of the results and a discussion about the strengths and weaknesses of the various solvers.



X_2 \ X_4	0	1
0	10	11
1	8	7

Figure 1: An example of the projection of an MWVC problem onto an independent set. The left panel shows a vertex-weighted undirected graph. The right panel shows the table that represents the projection of the MWVC problem onto the independent set $\{X_2, X_4\}$.

Background on Graph Theory

In this section, we review some preparatory background material from graph theory.

We denote an undirected graph by $G = \langle V, E\rangle$, where V is the set of its vertices and E is the set of its edges. A *vertex-weighted* undirected graph is an undirected graph with a non-negative real *weight* associated with each vertex. We denote such a graph by $G = \langle V, E, w\rangle$, where V and E follow the definition above, and w is a function that maps each vertex v to its weight $w_v = w(v)$. The weight of a set of vertices $S \subseteq V$ is defined as the sum of the weights of the individual vertices in S.

A set of vertices $U \subseteq V$ constitutes an independent set of an undirected graph G if and only if no two vertices in U are connected by an edge. A vertex cover is a subset of vertices $S \subseteq V$ such that every edge in E has at least one of its end points in S. A Minimum Vertex Cover (MVC) is a vertex cover of minimum cardinality. The MWVC is defined to be a vertex cover of minimum total weight on a vertex-weighted undirected graph. The problem of finding an MVC or an MWVC, referred to as the MVC problem or the MWVC problem, respectively, is generally NP-hard (Papadimitriou 1994). However, both problems can be solved in polynomial time on bipartite graphs. A bipartite graph is a graph whose vertices can be divided into two disjoint independent sets, referred to as its partitions.

The Constraint Composite Graph

In this section, we show how to construct the CCG corresponding to a given Boolean WCSP instance. This construction is presented formally in (Kumar 2008). First, we define the concept of projecting MWVC problems onto independent sets of a graph. Later, we use this idea to construct the "lifted" graphical representations of individual weighted constraints. Finally, we show how to build the lifted graphical representation of an entire WCSP, i.e., its CCG.

Projections of MWVC Problems onto Independent Sets

Given a vertex-weighted undirected graph $G = \langle V, E, w\rangle$, let $U = \{u_1, u_2 \ldots u_k\} \subseteq V$ be an independent set of G. Let a k-bit vector t be such that $t_i = 0$ imposes that u_i is necessarily excluded from the MWVC, and $t_i = 1$ imposes that u_i is necessarily included in the MWVC. The *projection* of the MWVC problem onto the independent set U is a table with 2^k entries, each corresponding to one of the possible 2^k k-bit vectors $t^{(1)}, t^{(2)} \ldots t^{(2^k)}$. The entry corresponding to $t^{(j)}$ is set to be the weight of the MWVC conditioned on the restrictions imposed by $t^{(j)}$.

Figure 1 illustrates this concept on an example of a graph. Here, we project the MWVC problem onto the independent set $\{X_2, X_4\}$; this yields a table of four entries, also shown in the figure. The value of each entry is computed according to the restrictions imposed by its indices. For instance, the indices $(X_2 = 1, X_4 = 0)$ impose the inclusion of X_2 in the MWVC and the exclusion of X_4 from the MWVC. With these restrictions, the MWVC is $\{X_2, X_3, X_5\}$. Its total weight is 8, which determines the value of the entry.

Weighted Constraints as Projections of MWVC Problems

The projection of an MWVC problem onto an independent set $U \subseteq V$ of a given graph $G = \langle V, E, w\rangle$ produces a table of size $2^{|U|}$. This can be seen as a weighted constraint over $|U|$ Boolean variables. Conversely, we might view a weighted constraint as the projection of an MWVC problem onto an independent set.

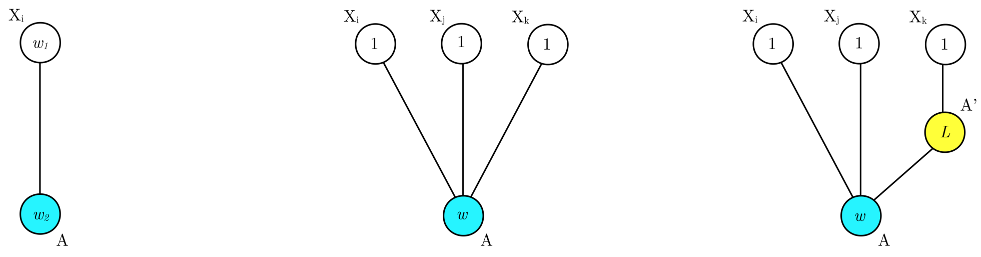
The procedure for constructing the lifted graphical representation of any weighted constraint over Boolean variables goes through the unique multivariate polynomial representation of a weighted constraint. It is an efficient procedure that outputs a graph containing vertices that represent the original variables as well as intelligently chosen auxiliary variables. The size of the graph is no more than the size of the tabular representation of the weighted constraint. This procedure uses the following two steps (Kumar 2008).

First, we convert each weighted constraint into a multivariate polynomial. This is done using a standard *Gaussian elimination* procedure for solving systems of linear equations. For each possible combination of values assigned to the participating variables, a linear equation is enforced between the weight of this combination of values and the evaluation of the multivariate polynomial at the same combination of values. This system has as many linear equations as the number of entries in the tabular representation of the weighted constraint. Furthermore, it has the same number of unknowns, i.e., the to-be-determined coefficients of the multivariate polynomial. For example, when applied to the generic ternary weighted constraint with scope $\{X_i, X_j, X_k\}$, we obtain a multivariate polynomial $P(X_i, X_j, X_k)$ of degree 3, as shown in Figure 2. Since the variables are Boolean, the multivariate polynomial has a total of eight coefficients to be determined: $a_{000}, a_{001} \ldots a_{111}$. We determine them by solving a system of eight linear equations, one for each combination of values of the variables in the scope. For each combination, the evaluation of $P(X_i, X_j, X_k)$ is equated to the weight of the corresponding combination in the weighted constraint. The system of linear equations has a unique solution that can be computed

X_i \ X_jX_k	00	01	10	11
0	c_{000}	c_{001}	c_{010}	c_{011}
1	c_{100}	c_{101}	c_{110}	c_{111}

$$P(X_i, X_j, X_k) = a_{000} + a_{100}X_i + a_{010}X_j + a_{001}X_k + a_{110}X_iX_j + a_{101}X_iX_k + a_{011}X_jX_k + a_{111}X_iX_jX_k$$
$$P(0,0,0) = c_{000} \quad P(0,0,1) = c_{001} \quad P(0,1,0) = c_{010} \quad P(0,1,1) = c_{011}$$
$$P(1,0,0) = c_{100} \quad P(1,0,1) = c_{101} \quad P(1,1,0) = c_{110} \quad P(1,1,1) = c_{111}$$

Figure 2: The multivariate polynomial representation of a generic ternary weighted constraint on the Boolean variables $\{X_i, X_j, X_k\}$. The top panel shows the tabular representation of the weighted constraint. The bottom panel shows the multivariate polynomial with the to-be-determined coefficients and the system of linear equations that yields them.



(a) $w \cdot X_i$ (b) $-w \cdot (X_i \cdot X_j \cdot X_k)$ (c) $w \cdot (X_i \cdot X_j \cdot X_k)$

Figure 3: The three kinds of multivariate monomials and their lifted graphical representations. Panel (a) shows the lifted graphical representation of a linear term $w \cdot X_i$. Here, w_1 and w_2 are non-negative, while $w = w_1 - w_2$ can be either positive or negative. Panel (b) shows the lifted graphical representation of a negative nonlinear term $-w \cdot (X_i \cdot X_j \cdot X_k)$. Here, w is positive. Panel (c) shows the lifted graphical representation of a positive nonlinear term $w \cdot (X_i \cdot X_j \cdot X_k)$. Here, w is positive. Only in this case, we use a second auxiliary variable A', shown in yellow, with a large weight L.

using the Gaussian elimination procedure.

Second, we represent each term of the multivariate polynomial graphically. There are only three kinds of terms, discussed below, each of which can be represented using a "gadget", i.e., a template graphical representation of a multivariate monomial.

- A *linear* term $w \cdot X_i$ can be represented by a graph with two connected vertices, one being the variable X_i, with weight w_1, and the other being an auxiliary variable A, with weight w_2. As shown in Figure 3a, w_1 and w_2 are set such that $w_1 - w_2 = w$.
- A *negative nonlinear* term, such as $-w \cdot (X_i \cdot X_j \cdot X_k)$, can be represented by a "flower" structure that contains one vertex for each variable in the term and an auxiliary variable A that is connected to all the others. As shown in Figure 3b, the auxiliary variable A has a weight w, while all the other vertices have unit weights.
- A *positive nonlinear* term, such as $w \cdot (X_i \cdot X_j \cdot X_k)$, can be represented by a "flower" structure that contains one vertex for each variable in the term, an auxiliary variable A, and a second auxiliary variable A', called the "thorn", between A and one of the variables. As shown in Figure 3c, the auxiliary variable A has a weight w, the thorn A' has a large weight L, while all the other vertices have unit weights.

Constructing the CCG

Once we have built the lifted graphical representation of each weighted constraint independently, we can construct the lifted graphical representation of the entire Boolean WCSP in a straightforward manner. We simply "merge" the vertices—along with their edges—that refer to the same variables. Then, we assign to each vertex a weight equal to the sum of the weights of the merged vertices. We refer to the resulting graph as the CCG of the given Boolean WCSP instance. Solving the MWVC problem on the CCG is equivalent to solving the original Boolean WCSP instance (Kumar 2008). This procedure allows us to go from "local" to "global" reasoning elegantly.

As mentioned before, since the CCG aids the reduction of a Boolean WCSP instance to an MWVC problem instance, it is able to simultaneously exploit both the graphical and the numerical structure in the weighted constraints. Graphically, the MWVC problem instance has the same treewidth as the constraint network of the original Boolean WCSP instance (Kumar 2008). Numerically, when the individual weighted constraints have bipartite lifted representations, the CCG is also bipartite. In this case, the MWVC problem can be solved efficiently using maxflow algorithms (Kumar 2008).

The possibility of reducing a Boolean WCSP instance to an MWVC problem instance inspires a new generation of

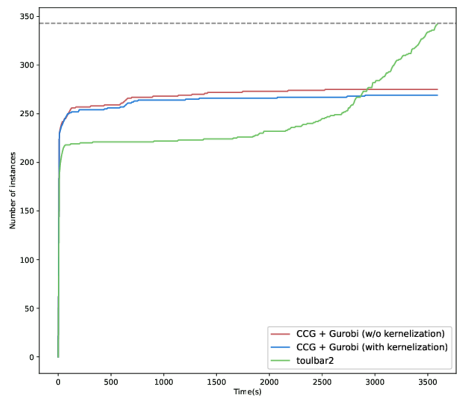


Figure 4: Total number of instances solved within a progressively increasing allotted time.

CCG-based WCSP solvers. This paper proposes the use of such solvers and empirically compares them to other state-of-the-art WCSP solvers. CCG-based solvers have the benefit of invoking specialized MWVC solvers: These exploit the fact that there are only two variables per constraint in the MWVC problem. Moreover, the MWVC problem is amenable to a kernelization procedure called NT reduction (Xu, Kumar, and Koenig 2017). In general, kernelization is a polynomial-time procedure that reduces the number of variables in a problem instance before starting search. Since NP-hard problems are typically characterized by an exponential search space, reducing the number of variables via kernelization has significant benefits.

Experiments

In this section, we present experimental results comparing our CCG-based solvers against toulbar2, the state-of-the-art WCSP solver. We first describe the experimental setup, including the benchmark instances, the various CCG-based solvers, the hardware, and other system configuration settings. Later, we show experimental results rendered as graphical plots. They portray the performance of the solvers in finding optimal and bounded suboptimal solutions for the given problem instances. Finally, we present an analysis of the results.

Benchmark Instances

We conducted our experiments on 343 Boolean WCSP instances. These come from the 'evalgm' repository[1] and include benchmark instances from:

- **CFN**. A collection of handcrafted, random, and real-world cost function networks (CFN 2010);
- **MRF**. A collection of problems from the Probabilistic Inference Challenge 2011 (PIC 2011), a probabilistic inference competition on graphical models, that includes segmentation problems and grid networks;
- **CVPR**. The Computer Vision and Pattern Recognition OpenGM2 Benchmark (CVPR 2015), a database of discrete energy minimization problems;
- **MaxCSP**. A collection of problems from the Third International CSP Solver Competition (CSP 2008), a competition based on solving a wide variety of CSPs, Max-CSPs, and WCSPs.

Methodology

Our framework creates a CCG starting from the input Boolean WCSP instance. Then, it adopts different strategies to solve the substrate MWVC problem. This leads to four different solvers.

1. **CCG + Gurobi (w/o kernelization)**. The MWVC problem on the CCG is reformulated as an ILP problem. In turn, this is solved using the Gurobi optimizer. Being an exact solver, Gurobi guarantees the optimality of the solution it finds, provided that it has sufficient time. Otherwise, Gurobi yields the best solution it is able to find within the given time. The solution of the ILP problem is then converted back to a solution of the original problem.
2. **CCG + Gurobi (with kernelization)**. The NT reduction kernelization procedure is applied to the CCG to reduce the size of the MWVC problem instance. Then, the same strategy described in (1) is used to solve the reduced MWVC problem instance.
3. **CCG + FastWVC (w/o kernelization)**. The MWVC problem on the CCG is solved using FastWVC. As it is based on local search, FastWVC does not necessarily produce optimal solutions. Even when it does, it is unable to prove the solution's optimality.
4. **CCG + FastWVC (with kernelization)**. The MWVC problem on the CCG is first kernelized via NT reduction. It is then solved using FastWVC.

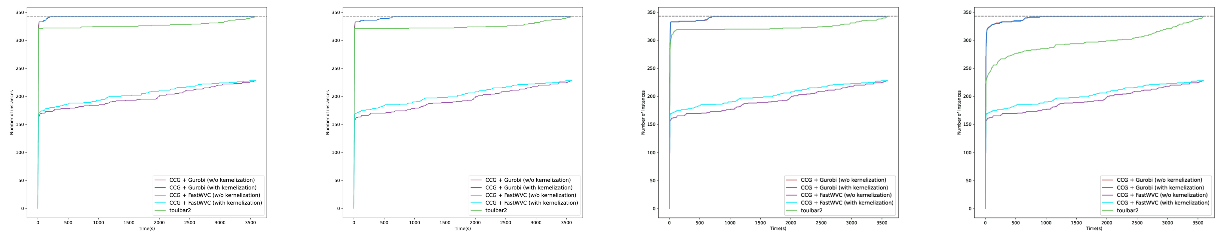
Hardware, Compilation, and Run Configuration

The experimental results were produced on a machine equipped with a 12th Gen Intel(R) Core(TM) i9-12900K processor and 128GB of RAM. We ran each experiment on a single performance core, with a frequency of up to 5.20GHz.

Our solvers were implemented in C++ using the Boost graph library (Boost 2015) and compiled using GCC 7.5.0 with the "-O3" option.

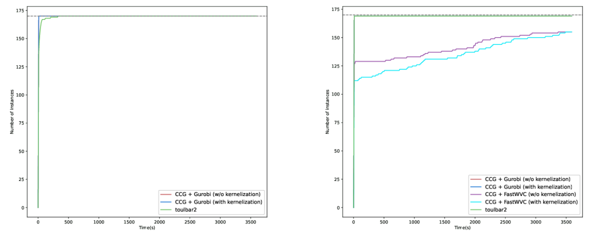
Each solver was given a time limit on each problem instance to find the best possible solution. In all the experiments, the time limit was set to one hour. The CCG-based solvers were run with default settings. toulbar2 was run using the parameters "-hbfs" (hybrid best-first search (Allouche et al. 2015)), "-dee" (restricted dead-end elimination (de Givry, Prestwich, and O'Sullivan 2013)), "-V" (VAC-based value ordering heuristic (Cooper et al. 2008)), and "-A" (enforcement of VAC at each search node with a search depth less than the default value 0).

[1]The WCSP instances in this repository can be freely downloaded in ".wcsp" (and many other) format(s) from `http://genoweb.toulouse.inra.fr/~degivry/evalgm`.



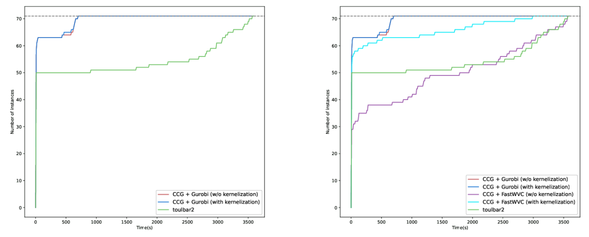
(a) $\beta = 0.01$ (b) $\beta = 0.001$ (c) $\beta = 0.0001$ (d) $\beta = 0.00001$

Figure 5: Total number of instances solved with a suboptimality factor β within a progressively increasing allotted time. In all the figures, the red curve is hidden behind the blue curve.



(a) exactly solved instances (b) $\beta = 0.00001$

Figure 6: A summary of the experiments performed on the CFN collection. (a) Total number of instances solved within a progressively increasing allotted time. (b) Total number of instances solved with a suboptimality factor $\beta = 0.00001$ within a progressively increasing allotted time. In (a), the red curve is hidden behind the blue curve. In (b), both the red and the blue curves are hidden behind the green curve.



(a) exactly solved instances (b) $\beta = 0.00001$

Figure 7: A summary of the experiments performed on the MRF collection. (a) Total number of instances solved within a progressively increasing allotted time. (b) Total number of instances solved with a suboptimality factor $\beta = 0.00001$ within a progressively increasing allotted time. In both panels (a) and (b), the red curve is mostly hidden behind the blue curve.

Results

We summarize our experimental results from two different perspectives. First, we show the ability of the solvers to produce optimal solutions within allotted times. Then, we show their ability to produce suboptimal solutions for varying suboptimality bounds within allotted times.

For the first set of results, we ran all the solvers on all the instances mentioned previously in this section. Then, we counted the total number of instances solved by each solver within an allotted time, which is the same for each instance. Figure 4 shows these results for a progressively increasing allotted time (expressed in seconds on the x-axis). For each value of the allotted time $t \in \{1, 2 \dots 3600\}$, the corresponding y-coordinate for each solver s is defined as:

$$y_s(t) = \text{number of instances solved by } s \text{ within } t \text{ seconds.}$$

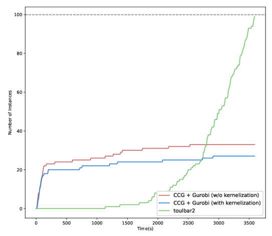
This plot does not include the results of CCG + FastWVC because FastWVC is not guaranteed to produce optimal solutions.

In the second set of results, we highlight the efficiency and the effectiveness of our solvers at approaching the optimal solution. For this purpose, we counted the total number of instances solved by each solver with a suboptimality factor β within an allotted time, which is the same for each instance. We denote by $A_s(i,t)$ the cost of the best solution found by a solver s for the ith Boolean WCSP instance within t seconds; and by $A^*(i)$ the cost of the optimal solution. For each value of the allotted time $t \in \{1, 2 \dots 3600\}$, the corresponding y-coordinate for each solver s is defined as:

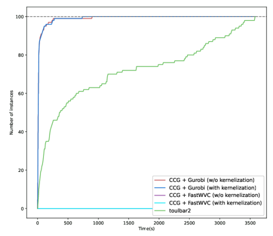
$$y_s(t) = \left|\left\{ i \;\middle|\; \frac{A_s(i,t) - A^*(i)}{A^*(i)} \leq \beta \right\}\right|.$$

Figure 5 shows these results for progressively decreasing values of β: 0.01 (Figure 5a), 0.001 (Figure 5b), 0.0001 (Figure 5c), and 0.00001 (Figure 5d).

In the third set of results, we categorize the plots according to the benchmark collections mentioned in this section: CFN, MRF, and CVPR. MaxCSP is excluded because there are only two Boolean WCSP instances in this collection. This number is insufficient to conclusively show any trends in the behaviors of the solvers. Figures 6, 7, and 8 show the categorized results for CFN, MRF, and CVPR, respectively. In each of these figures, we present two panels. The first shows the total number of instances solved by each solver



(a) exactly solved instances



(b) $\beta = 0.00001$

Figure 8: A summary of the experiments performed on the CVPR collection. (a) Total number of instances solved within a progressively increasing allotted time. (b) Total number of instances solved with a suboptimality factor $\beta = 0.00001$ within a progressively increasing allotted time. In (b), the purple curve is hidden behind the cyan curve and the red curve is mostly hidden behind the blue curve.

within an allotted time, which is the same for each instance. The second shows the total number of instances solved by each solver with a suboptimality factor $\beta = 0.00001$ within an allotted time, which is the same for each instance.

Analysis and Discussion

The experimental results highlight how CCG-based solvers compare to toulbar2 on various performance metrics.

On the CFN collection (Figure 6), our CCG + Gurobi solvers perform slightly better than toulbar2, as they are able to solve all the instances in this category in less than one second. In comparison, toulbar2 takes considerably more time to solve some of the instances. In contrast, our CCG + FastWVC solvers are less competitive. In fact, the amount of time that FastWVC takes to approach the optimum with a sufficiently good suboptimality factor is generally greater than the corresponding time taken by its competitors.

On the MRF collection (Figure 7), our CCG-based solvers are in a more favorable position compared to toulbar2. In Figure 7a, we observe that the CCG + Gurobi solvers solve all the instances in less than 1000 seconds, while toulbar2 takes up to one hour to solve some of the instances. Figure 7b shows that the CCG + FastWVC solvers are able to find solutions with a good suboptimality factor within a competitive amount of time. In fact, toulbar2 is only able to outperform our CCG + FastWVC (w/o kernelization) solver for small allotted times. In contrast, our CCG + FastWVC (with kernelization) solver delivers better performance at finding high-quality solutions for small allotted times.

On the CVPR collection (Figure 8), toulbar2 outperforms our CCG-based solvers for large allotted times. However, for allotted times smaller than 2500 seconds, our CCG + Gurobi solvers are able to solve more instances compared to toulbar2. In Figure 8b, we observe that our CCG + Gurobi solvers approach the optimum within a very small amount of time. In fact, they both find a solution with a suboptimality factor of 0.00001 for all the instances in less than 1000 seconds. However, on the instances in this collection, our solvers generally struggle to prove the optimality of an optimal solution after they have found it.

In our experiments, we also observe that kernelization is not significantly beneficial, despite its theoretical usefulness. In most cases, it either solves an instance entirely by fixing the optimal values of all the variables, or is not helpful at all. This all-or-none effect of kernelization is worth investigating in future work. We observe some benefits only in Figure 7, where kernelization is able to fix the optimal values of all the variables in some instances. In general, enabling kernelization does not deteriorate the performance of the solver either, since it runs in polynomial time. However, when the instances are very large, enabling kernelization can incur an overhead cost, particularly if it is ineffective in reducing the number of variables. For example, in Figure 8a, we observe that CCG + Gurobi (w/o kernelization) marginally outperforms CCG + Gurobi (with kernelization).

Conclusions and Future Work

In this paper, we presented some CCG-based WCSP solvers with the intent to simultaneously exploit the graphical and the numerical structures at an instance level. In the same framework, we are able to render any improvements to solvers for the specific MWVC problem as upgrades to solvers for the more general WCSP. CCG-based solvers also have the theoretical advantage of enabling kernelization. In general, we observed that CCG-based solvers outperform toulbar2 on certain types of WCSP instances, although they do not dominate it. We envision a next-generation WCSP solver that combines the complementary strengths of our CCG-based solvers and toulbar2.

Acknowledgements

This work at the University of Southern California is supported by NSF under grant number 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government.

References

Allouche, D.; de Givry, S.; Katsirelos, G.; Schiex, T.; and Zytnicki, M. 2015. Anytime hybrid best-first search with tree decomposition for weighted CSP. In *Proceedings of the Twenty-First International Conference on Principles and Practice of Constraint Programming*, 12–29.

Boost. 2015. Boost c++ libraries. `http://www.boost.org/`. last accessed 23 Nov 2022.

Cai, S.; Li, Y.; Hou, W.; and Wang, H. 2019. Towards faster local search for minimum weight vertex cover on massive graphs. *Information Sciences* 471:64–79.

CFN. 2010. CFLib, library of cost function networks. `http://costfunction.org/benchmark`. last accessed 23 Nov 2022.

Cooper, M.; de Givry, S.; Sanchez, M.; Schiex, T.; and Zytnicki, M. 2008. Virtual arc consistency for weighted CSP.

In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, 253–258.

CSP. 2008. Third international CSP solver competition (CSP, max-CSP and weighted-CSP competition). `http://www.cril.univ-artois.fr/CPAI08/`. last accessed 23 Nov 2022.

CVPR. 2015. OpenGM benchmark. `http://hciweb2.iwr.uni-heidelberg.de/opengm/index.php?l0=benchmark`. last accessed 23 Nov 2022.

de Givry, S.; Prestwich, S. D.; and O'Sullivan, B. 2013. Dead-end elimination for weighted CSP. In *Proceedings of the Nineteenth International Conference on Principles and Practice of Constraint Programming*, 263–272.

Gurobi. 2022. Gurobi optimizer reference manual. `https://www.gurobi.com`. last accessed 24 Nov 2022.

Kumar, T. K. S. 2008. 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*, 282–297.

Kumar, T. K. S. 2016. Kernelization, generation of bounds, and the scope of incremental computation for weighted constraint satisfaction problems. In *Proceedings of the Fourteenth International Symposium on Artificial Intelligence and Mathematics*.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

PIC. 2011. Probabilistic inference challenge 2011. `http://www.cs.huji.ac.il/project/PASCAL/realBoard.php`. last accessed 23 Nov 2022.

Xu, H.; Sun, K.; Koenig, S.; Hen, I.; and Kumar, T. K. S. 2020. Hybrid quantum-classical algorithms for solving the weighted CSP. In *Proceedings of the Sixteenth International Symposium on Artificial Intelligence and Mathematics*.

Xu, H.; Koenig, S.; and Kumar, T. K. S. 2017. A constraint composite graph-based ILP encoding of the boolean weighted CSP. In *Proceedings of the Twenty-Third International Conference on Principles and Practice of Constraint Programming*.

Xu, H.; Kumar, T. K. S.; and Koenig, S. 2017. The nemhauser-trotter reduction and lifted message passing for the weighted CSP. In *Proceedings of the Fourteenth International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*.