



**HAL**  
open science

# Best of Both Worlds: Solving the Cyclic Bandwidth Problem by Combining Pre-existing Knowledge and Constraint Programming Techniques

Guillaume Fertin, Eric Monfroy, Claudia Vasconcellos-Gaete

► **To cite this version:**

Guillaume Fertin, Eric Monfroy, Claudia Vasconcellos-Gaete. Best of Both Worlds: Solving the Cyclic Bandwidth Problem by Combining Pre-existing Knowledge and Constraint Programming Techniques. International Conference on Computational Science - ICCS 24, Jul 2024, Hanoi, Vietnam. pp.197-211, 10.1007/978-3-031-63775-9\_14 . hal-04646116

**HAL Id: hal-04646116**

**<https://univ-angers.hal.science/hal-04646116v1>**

Submitted on 12 Jul 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.




L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License



# Best of Both Worlds: Solving the Cyclic Bandwidth Problem by Combining Pre-existing Knowledge and Constraint Programming Techniques

Guillaume Fertin<sup>1</sup> , Eric Monfroy<sup>2</sup> , and Claudia Vasconcellos-Gaete<sup>2</sup> 

<sup>1</sup> Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004,  
44000 Nantes, France

[guillaume.fertin@univ-nantes.fr](mailto:guillaume.fertin@univ-nantes.fr)

<sup>2</sup> LERIA, Université d'Angers, Angers, France

[{eric.monfroy,cclaudia.vasconcellos}@univ-angers.fr](mailto:{eric.monfroy,cclaudia.vasconcellos}@univ-angers.fr)

**Abstract.** Given an optimization problem, combining knowledge from both (i) structural or algorithmic known results and (ii) new solving techniques, helps gain insight and knowledge on the aforementioned problem by tightening the gap between lower and upper bounds on the sought optimal value. Additionally, this gain may be further improved by iterating (i) and (ii) until a fixed point is reached.

In this paper, we illustrate the above through the classical CYCLIC BANDWIDTH problem, an optimization problem which takes as input an undirected graph  $G = (V, E)$  with  $|V| = n$ , and asks for a labeling  $\varphi$  of  $V$  in which every vertex  $v$  takes a unique value  $\varphi(v) \in [1; n]$ , in such a way that  $B_c(G, \varphi) = \max\{\min_{uv \in E(G)}\{|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|\}\}$ , called the *cyclic bandwidth of  $G$* , is minimized.

Using the classic benchmark from the Harwell-Boeing sparse matrix collection introduced in [16], we show how to combine (i) previous results from the CYCLIC BANDWIDTH literature, and (ii) new solving techniques, which we first present, and then implement, starting from the best results obtained in step (i). We show that this process allows us to determine the optimal cyclic bandwidth value for half of the instances of our benchmark, and improves the best known bounds for a large number of the remaining instances.

**Keywords:** Graph Labeling · Cyclic Bandwidth · Lower Bounds · Upper Bounds · Constraint Programming · Solver

## 1 Introduction

The classical CYCLIC BANDWIDTH problem is an optimization problem that takes as input an undirected graph  $G = (V, E)$  with  $|V| = n$  and asks for a labeling  $\varphi$  of  $V$  in which every vertex  $v$  takes a unique value  $\varphi(v) \in [1; n]$ , in such a way that  $B_c(G, \varphi) = \max\{\min_{uv \in E(G)}\{|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|\}\}$  (i.e., the *cyclic bandwidth of  $G$* ) is minimized.

The CYCLIC BANDWIDTH problem was first presented in [11] within the framework of creating a ring interconnection network for a group of computers. This problem can be regarded as a modification of the widely recognized BANDWIDTH MINIMIZATION problem, originally proposed by Harper [6] in 1964. For a comprehensive historical overview, readers can refer to the survey by Chinn et al. [2]. The BANDWIDTH MINIMIZATION problem also asks for a labeling  $\varphi$ , i.e., a bijection from  $V$  to  $[1; n]$ , using a computed value of  $B(G, \varphi) = \max_{uv \in E} \{|\varphi(u) - \varphi(v)|\}$ . The BANDWIDTH MINIMIZATION problem asks for a labeling  $\varphi^*$  such that  $B(G, \varphi^*)$  is minimized. It can be seen that both problems are related in the sense for any graph  $G$ , we have  $B_c(G) \leq B(G)$ .

The CYCLIC BANDWIDTH has been extensively studied. Its complexity has been established as NP-hard, even in the scenario of trees with a maximum degree of 3 [13]. Furthermore, the specific value of  $B_c(G)$  has been ascertained for graphs within distinct categories, including paths, cycles, Cartesian products of paths (or cycles, or a combination of both), full  $k$ -ary trees, complete graphs, complete bipartite graphs, and unit interval graphs [3, 8, 12, 13].

More recent papers are concerned with designing efficient heuristics for CYCLIC BANDWIDTH (see, e.g., [19, 20, 22]) or BANDWIDTH MINIMIZATION (see, e.g., [15, 17, 18]). For both problems, execution time, upper bounds, or lower bounds are considered, examined, and tested on a subset of the classical Harwell-Boeing sparse matrix collection<sup>1</sup>.

Other investigations studied the correlation between  $B_c(G)$  and  $B(G)$ , with a specific focus on identifying conditions that guarantee the equality  $B(G) = B_c(G)$  [4, 9, 14]. Another set of results focuses on establishing bounds for  $B_c(G)$ , particularly lower bounds, in the context of general graphs [4, 25]. Initially, it is evident that for any graph  $G$ ,  $B_c(G) \geq \frac{\Delta(G)}{2}$ , where  $\Delta(G)$  represents the maximum degree of  $G$ . Furthermore, across all graphs  $G$ , the relationship  $\frac{B(G)}{2} \leq B_c(G) \leq B(G)$  holds, with the leftmost bound stemming from [14]. Various other lower bounds have been derived in the literature, many of which are grounded in density (“propagation”) considerations, or a relevant cycle basis of the examined graph (refer to, e.g., [4, 25]).

In this paper, we propose the Recycling algorithm, orchestrating some existing results, both experimental and theoretical, based on bandwidth and cyclic bandwidth. For the 113 Harwell-Boeing graph instances proposed in [16], the Recycling algorithm is either able to tighten bounds, prove optimization, and find the optimum of the bandwidth or cyclic bandwidth.

Constraint programming (CP) [23] is a problem-solving paradigm for solving combinatorial problems using techniques issued from artificial intelligence, computer science, and operations research. In CP, rather than describing how to solve the problem, users formulate the problem by defining decision variables interconnected by constraints. As said by E. Freuder, “Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

<sup>1</sup> see, e.g., <https://math.nist.gov/MatrixMarket/collections/hb.html> and <https://sparse.tamu.edu/HB>

The constraints encompass various types of variables (such as Boolean, bounded integers, etc.) and constraint types (including linear, non-linear arithmetic, symbolics, etc.). In CP, a model represents a problem, and an instance is given by a model and some data. In our case, the problem consists of generating a graph labeling minimizing the cyclic bandwidth of graphs, and an instance is the instantiation of the model with a given graph. The direct modeling in CP of the cyclic bandwidth problem is very close to its mathematical formulation. However, this model is not well-suited for CP, and not efficiently solved.

Here, we propose a more original model, taking advantage of the strengths of CP. This model is based on constraints in extension [1], i.e., table constraints [10] considering the candidate labelings of two vertices linked by an edge to respect a given cyclic bandwidth value. This model is thus a satisfiability model for computing a labeling value less or equal to a given  $k$ . This  $k$  can easily be minimized by an efficient dichotomy algorithm considering a property of the problem: if there is a cyclic labeling of value  $k$ , there is also one of size  $k + 1$ , and if there is no labeling of value  $k$ , none of value  $k - 1$  exists. This model is solved more efficiently than the direct model and may improve the results of the Recycling algorithm, which in turn offers the possibility to better solve the CP model. A fixed-point application of the sequence Recycling algorithm, CP model solving, is thus beneficial.

Experimentally, on the 113 instances selected in [16], our method proves highly powerful, as it optimally solves half of the 113 instances. More precisely, the fixed point of our Recycling algorithm and CP model solving proves the optimality of 63 instances, determines the optimal value of 56 instances, and improves the bounds of one instance.

This paper is organized as follows: Sect. 2 presents the CYCLIC BANDWIDTH problem along with some existing results for it, or for its related problem, BANDWIDTH MINIMIZATION. Section 3 presents the Recycling algorithm and how it orchestrates the existing results from literature. Section 4 presents an optimization function for the Recycling Algorithm, using constraint programming. Finally, Sect. 5 presents the results obtained using the Recycling algorithm and its optimization.

## 2 The Cyclic Bandwidth Problem and Existing Results

### 2.1 The Problem

The CYCLIC BANDWIDTH problem is a graph labeling problem that can be formulated as follows. Let  $G(V, E)$  be a finite undirected graph (called the *guest graph*) of order  $n$ , and  $C_n(V', E')$  be a cycle graph (called the *host graph*) with  $|V'| = n$ . An embedding of  $G$  in  $C_n$  is an injection  $\phi : V \rightarrow V'$ . The *cyclic distance*  $d_c$  between two vertices  $u, v \in V$  linked by an edge of  $E$  is defined by:

$$d_c(u, v) = \min\{|\phi(u) - \phi(v)|, n - |\phi(u) - \phi(v)|\} \quad (1)$$

The cyclic bandwidth of an embedding  $\phi : V \rightarrow V'$  is the maximum distance between two vertices:

$$B_c(G, \phi) = \max_{u,v \in E} \{d_c(u, v)\} \quad (2)$$

The cyclic bandwidth problem consists in finding an embedding  $\phi^*$  among the set  $\mathcal{E}$  of embeddings from  $G$  to  $C_n$  such that  $B_c(G, \phi^*)$  is minimum:

$$B_C^*(G) = \min_{\phi \in \mathcal{E}} \{B_c(G, \phi)\} \quad (3)$$

## 2.2 Metaheuristics Results

CYCLIC BANDWIDTH and BANDWIDTH MINIMIZATION are both NP-complete problems. Solving any of these two problems requires exploring very large search spaces, hence it is not surprising that most of the methods developed to solve them are metaheuristics. In this section, we present some relevant algorithms and results that lately will become inputs for our Recycling algorithm. The first three metaheuristics solve BANDWIDTH MINIMIZATION (also known as MATRIX BANDWIDTH MINIMIZATION, or MBMP), while the last one is a recent algorithm developed for CYCLIC BANDWIDTH.

*Martí's enhanced Branch-and-Bound* [15]. This is one of the few exact methods proposed to solve MBMP. It takes advantage of a solution provided by a GRASP method to reduce the size of the tree to explore, focusing only in the branch for  $b_t = b_{up} - 1$ , with  $b_{up}$  being the solution provided by GRASP. It is also the first exact method to provide upper and lower bounds for some large instances ( $|V| > 500$ ).

*Mladenovic's VNS*. In [17] the authors propose a local search algorithm known as Variable Neighborhood Search (VNS) that combines reduced neighborhoods, fast local search procedures, and specific neighborhood structures (from [21]). Additionally, they use the *number of critical vertices* as a secondary objective function, to deal with several neighborhoods having the same bandwidth. Their results outperform several other heuristics in terms of solution quality and computing time required, and they improve the best-known solutions for 42 (out of the 113) Harwell-Boeing instances.

*Pop's genetic algorithm* [18]. Using a list of interchange of rows or columns as individuals and a problem-specific genetic operator called pruning, this algorithm was successfully tested in the 113 Harwell-Boeing instances. Their results improve the best-known bandwidth values for almost a third of the graphs, and outperforming methods like the already described VNS [17] and Martí's branch-and-bound [15].

*Ren's New Iterated Local Search (NILS)*. This is a metaheuristic proposed in [20] to solve the CYCLIC BANDWIDTH problem. Starting from a random initial solution, it iterates over a local optimization (a dedicated Tabu search, DTS) and adds two perturbation strategies, to escape local optimum traps and explore

unvisited areas. When the DTS stagnates, a Direct perturbation is triggered to modify the current solution, using a randomized shift-insert operator. If this perturbation also stagnates, then a Strong perturbation is triggered to apply a destruction-reconstruction heuristic, moving some uncritical vertices closer to the critical ones.

The NILS algorithm was applied to 85 standard graphs (paths, cycles, caterpillars, etc.) and 28 Harwell-Boeing instances; from these 28 graphs, NILS improved 4 existing results (compared against [19,22]) and matched all other best results obtained.

### 2.3 Theoretical Structural Results

We summarize here three structural properties about  $B_c^*(G)$  (and possibly  $B^*(G)$ ) that apply to any graph  $G$ , and that rely on polynomial time computations [4]. They allow to: (i) obtain lower bounds for  $B_c^*(G)$ , (ii) determine conditions under which  $B_c^*(G) = B^*(G)$  and, in certain conditions, (iii) given a labeling for CYCLIC BANDWIDTH, provide a labeling for BANDWIDTH MINIMIZATION, of same value. All these properties will be used in our Recycling algorithm (see Sect. 3).

*Extended density.* Theorem 1 in [4] gives a lower bound for  $B_c^*(G)$  based on the neighborhood “up to distance  $i$ ” for any vertex  $u \in V(G)$  and any (relevant) value of  $i$ . It can be seen as a generalization of the obvious  $\frac{\Delta(G)}{2}$  lower bound. Theorem 3 in [4] is in the same spirit, but considers the neighborhood (up to distance  $i$ ) of any any pair of vertices connected by an edge in  $G$ .

*Cycle basis considerations.* Lower bounds on  $B_c^*(G)$  can also be obtained by computing the length  $\ell$  of the longest cycle in a cycle basis of  $G$  (the notion of cycle basis being a classical graph-theoretical notion, see e.g. [5]). This is the purpose of Theorem 9 in [4], which actually contains two results: a lower bound for  $B_c^*(G)$  based on  $\ell$ , and a condition under which  $B_c^*(G) = B^*(G)$ .

*Relabeling.* Another interesting result from [4] is its Algorithm 1, which provides a labeling  $\phi'$  for the BANDWIDTH MINIMIZATION problem, given a labeling  $\phi$  for the CYCLIC BANDWIDTH problem, in such a way that  $B(G, \phi') \leq B_c(G, \phi)$ . Note that this result is only guaranteed under some conditions described in Lemma 8 in [4]. This result may be useful as it may decrease upper bounds on  $B^*(G)$ , which in turn, may lead us to conclude that  $B^*(G) = B_c^*(G)$  (see Algorithms 2 and 4 in Sect. 3 for more details).

## 3 Recycling Algorithm: Orchestration of Existing Results

The Recycling algorithm we propose here is based on results of previous works, and part of its structure is directed by some theorems of [4] (see Sect. 2.3). For sake of readability, from now on (and if clear from the context), we will denote

by  $B_c^*$  (resp.  $B^*$ ) the optimal value  $B_c^*(G)$  (resp.  $B^*(G)$ ). The inputs for the Recycling algorithm are described in Table 1, whereas the Recycling algorithm is described in Algorithms 1 to 4.

**Table 1.** Inputs for the Recycling algorithm

	input	description
Graph	$G$	A graph
	$n$	Order of $G$
	$\ell$	Length of the longest cycle in a cycle basis of $G$
Bandwidth	$lb_{B\_Martí}$	Bandwidth lower bound from Martí et al. [15]
	$ub_{B\_Martí}$	Bandwidth upper bound from Martí et al. [15]
	$ub_{B\_Pop}$	Bandwidth upper bound from Pop et al. [18]
	$ub_{B\_Mladenovic}$	Bandwidth upper bound from Mladenovic et al. [17]
Cyclic Bandwidth	$ub_{C\_NILS}$	Cyclic Bandwidth upper bound from Ren et al. [20]
	$lb\_density$	Cyclic Bandwidth lower bound from the extended density notion [4]

## 4 Cyclic Bandwidth as an Optimization Constrained Problem

Constraint programming [23] (CP) is a paradigm for solving combinatorial problems using a wide range of methods issued from artificial intelligence, computer science and operations research. In CP, users focus on the “what”, not on the “how”: this means that users declaratively state the problem, not how to solve it. Hence, a problem is described as a constraint satisfaction problem (CSP) or constrained optimization problem (COP): a CSP is defined by some decision variables, each one with its domain (its candidate values) and constraints (relations) linking these variables; a COP is given by a CSP and an objective function to be optimized.

To model the CYCLIC BANDWIDTH problem, we consider finite domain decision variables, i.e., bounded integer variables, and arithmetic constraints. A label corresponds to a finite domain variable ranging from 1 to  $n$ , i.e.,  $\mathcal{N} = \{1, \dots, n\}$  is a set of  $n$  labels, and for each  $v \in V$ , the variable  $\phi_v$  represents the label of  $v$ :

$$\forall v \in V, \phi_v \in \mathcal{N} \quad (4)$$

**Algorithm 1.** The Recycling Algorithm

---

▷ Collect all knowledge about lower and upper bounds for  $B^*$  and  $B_c^*$

$$lb_C \leftarrow \max\{lb\_density, \min\{lb_{B\_Marti}, \lceil \frac{n}{\ell} \rceil\}\}$$

$$ub_C \leftarrow ub_{C\_NILS}$$

$$lb_B \leftarrow lb_{B\_Marti}$$

$$ub_B \leftarrow \min\{ub_{B\_Marti}, ub_{B\_Pop}, ub_{B\_Mladenovic}\}$$

▷ Compare the lower (resp. upper) bounds and adjust

**if**  $lb_B < lb_C$  **then**  
    $lb_B \leftarrow lb_C$   
**end if**  
**if**  $ub_B < ub_C$  **then**  
    $ub_C \leftarrow ub_B$   
**end if**

▷ If  $ub_C < lb_B$  then  $B_c^* \neq B^*$ , otherwise check whether equality holds

**if**  $ub_C \geq lb_B$  **then**  
   eq\_or\_unknown()  
**end if**

---

**Algorithm 2.** eq\_or\_unknown()

---

**if**  $ub_B \leq \lceil \frac{n}{\ell} \rceil$  **then**  
   update\_bounds()      ▷ In that case,  $B_c^* = B^*$  and bounds should be updated  
**else**  
   try\_relabeling()      ▷  $B_c^* = ? B^*$  Relabeling is then tested.  
**end if**

---

**4.1 Arithmetic (or Direct) Model**

This model is a direct translation of the mathematical definition of the problem. The constraints of the arithmetic model are:

- All the labels must be unique:

$$\text{AllDifferent}\{\phi_v | v \in V\} \quad (5)$$

with **AllDifferent** [7] being the standard global constraint<sup>1</sup> which states that all variables in this constraint must be pairwise different.

- Cyclic bandwidth of the current labeling  $\phi$ :

$$B_c(G, \phi) = \max_{(u,v) \in E} \{d_c(\phi_u, \phi_v)\} \quad (6)$$

where  $B_c(G, \phi)$  is a finite domain variable ranging from 1 to  $n - 1$ . Note that  $d_c$  has been defined in Sect. 2.1.

<sup>1</sup> A global constraint provides some abstractions to improve expressiveness, but also are treated more efficiently by the solver using some dedicated algorithms.



**Algorithm 3.** `update_bounds()`


---

▷ Note: this algorithm is invoked only if  $B^* = B_c^*$

▷ Note: if  $ub_C$  improved  $ub_B$ , relabeling gives a labeling of value  $ub_B$  for bandwidth

$ub_B, ub_C \leftarrow \min\{ub_B, ub_C\}$

$lb_B, lb_C \leftarrow \max\{lb_B, lb_C\}$

**if**  $ub_B = lb_B$  **then** ▷ We have reached the optimum for both problems

$B^*, B_c^* \leftarrow lb_C$

**end if**

---

**Algorithm 4.** `try_relabeling()`


---

**if**  $ub_B > ub_C$  **then** ▷ Relabeling useful only in this case

$ub'_B \leftarrow \text{apply\_relabeling}()$  ▷ Algorithm 1 from [4] – see Section 2.3

**if**  $ub'_B < ub_B$  **then** ▷ Relabeling has improved bandwidth upper bound

$ub_B \leftarrow ub'_B$

`eq_or_unknown()` ▷ We can test again whether  $B^* = B_c^*$

**end if**

**end if**

---

- Optimization to find the labeling  $\phi^*$  that minimizes the cyclic bandwidth:

$$B_C^*(G) = \text{minimize } B_C(G, \phi) \quad (7)$$

where  $B_C(G)$  is a finite domain variable ranging from 1 to  $n - 1$ .

The arithmetic model  $\mathcal{M}_A^*$  is thus:

$$\mathcal{M}_A^* = (4) \wedge (5) \wedge (6) \wedge (7)$$

However, in terms of efficiency, a constraint solver uses dynamically and intensively the constraints to prune/reduce the search space, and it cannot take full advantage of this formulation since most of the problem is in the objective function.

## 4.2 Finite Domain Extensional Constraint Model for Satisfiability

We now consider finite domain extensional constraints (see, e.g., [1]), also known as table constraints [10]: a constraint is defined by enumerating the allowed (resp. forbidden) tuples of constants satisfying (resp. violating) it. Then, we enforce a tuple of variables to be an element of this table (using the *in* keyword). There are several types of table constraints and we use the classic one.

We have first to change the problem into a satisfiability problem: given an integer  $k$ , find a labeling  $\phi$  such that  $B_C(G, \phi) \leq k$ . Now, let us consider  $\mathcal{L}(k)$ , the set of possible pairs of labels for pairs of vertices linked by an edge:

$$\mathcal{L}(k) = \{(\ell, \ell') \mid \ell, \ell' \in \mathcal{N}^2, \ell \neq \ell', \min\{|\ell - \ell'|, n - |\ell - \ell'|\} \leq k\}$$

The finite domain variables we need are the same as for the  $\mathcal{M}_A$  model. The constraints are the following:

- All labels must be different. For this, we use the `AllDifferent` constraints as above (see (5)).
- Labels must respect the cyclic bandwidth of cost  $k$  w.r.t. distance  $d_c$ , i.e.

$$\forall(u, v) \in E, (\phi_u, \phi_v) \text{ in } \mathcal{L}(k) \quad (8)$$

Thus, the finite domain extensional constraint satisfiability model is:

$$\mathcal{M}_E = (4) \wedge (5) \wedge (8)$$

### 4.3 Possible Improvements

To break symmetries, i.e., to remove some symmetric solutions that could be derived from remaining solutions, we can add the constraint:

$$\phi_{u_\Delta} = 1 \quad (9)$$

where  $u_\Delta$  is the vertex of highest degree. This constraint removes  $n - 1$  cyclic permutation solutions that can be recovered later on (if necessary) by rotation of the remaining solution. This constraint thus reduces the search space. Note that we could assign 1 to the label of any vertex. However, intuitively, we feel that fixing the vertex of highest degree is more efficient.

As the labeling is cyclic, this means that we can turn one way (counterclockwise for example) or the other (clockwise). This symmetry can be broken to enforce one direction by ordering any two labels:

$$\phi_u \geq \phi_v \quad (10)$$

Note that  $u$  and  $v$  can be any label, but fixing  $u = u_\Delta$  and  $v$  the vertex having the second highest degree seems a good intuition. Constraints (9) and (10) can be added to models  $\mathcal{M}_E$  and  $\mathcal{M}_A^*$ .

Some redundant constraints can also be added. For example, consider all the cycles of size 3 in  $G$ . Then, we can build a table of 3-uples representing the “legal” labeling of 3 vertices for a cyclic bandwidth of cost  $k$ . Although too expensive in the general case, this kind of redundant constraint can be added beneficially only around the vertex of the highest degree for example.

### 4.4 From Satisfiability to Optimization Models

It is obvious that if there is a cyclic labeling of cost  $k$ , there is also one of cost  $k + 1$  and, using the contraposition, if there is no labeling of cost  $k$ , then none of cost  $k - 1$  exists. We propose to use the satisfiability model  $\mathcal{M}_E$  inside a dichotomy algorithm (see Algorithm 5) benefiting from the above property.

This `optimization` function (as described in Algorithm 5) can be called with the lower bound  $lb = \lceil \Delta/2 \rceil$  (where  $\Delta$  is the maximum degree of  $G$ ) and the upper bound  $ub = \lceil n/2 \rceil$ . When knowing better bounds, such as the ones returned by our Recycling Algorithm, this function will be more efficient and

**Algorithm 5.** optimization()

---

```

 $k_{best} \leftarrow ub$ 
while  $lb < ub$  do
   $k \leftarrow (ub + lb) \text{ div } 2$ 
   $\mathcal{L} \leftarrow \{(\ell, \ell') \in \mathcal{N}^2 \mid \ell \neq \ell', \min\{|\ell - \ell'|, n - |\ell - \ell'|\} \leq k\}$ 
  if solve( $\mathcal{M}_E, \mathcal{L}$ ) is SAT then
     $ub \leftarrow k$ 
     $k_{best} \leftarrow ub$ 
  else
     $lb \leftarrow k + 1$ 
  end if
end while
return  $k$ 

```

---

will succeed more often in a reasonable time. The `solve` function creates the required model and solves it with an appropriate solver, i.e., a finite domain (FD) solver for  $\mathcal{M}_E$ .

*Iterating Recycling and Optimization algorithms.* Both the Recycling and Optimization algorithms aim at reducing the search interval, whose extremities are respectively the lower and upper bounds for  $B_c^*(G)$ , for any instance  $G$ . Obviously, if this interval is reduced to one value, then we have determined the optimal value  $B_c^*(G)$ . If not, then it is possible to iteratively apply Algorithms 1 and 5, aiming at further reducing the search interval, until a fixed point is reached.

## 5 Results

The benchmark we use here to evaluate performances of both our Recycling algorithm (Algorithm 1) and our Optimization algorithm (Algorithm 5) on the CYCLIC BANDWIDTH problem, is the classic 113 Harwell-Boeing benchmark [16], extracted from the Harwell-Boeing sparse matrix collection.

The results are shown Table 2 (33 medium-sized instances of our benchmark), along with Tables 3 and 4 (remaining 80 instances). In these three tables, we describe, in the three first columns, the name of each instance, its number of vertices and edges. The four subsequent columns, collectively called *inputs*, respectively provide the best lower and upper bounds for the BANDWIDTH MINIMIZATION and the CYCLIC BANDWIDTH problems. These four values correspond to the ones obtained by applying lines 1–4 of Algorithm 1. The two following columns, collectively called *Recycling algorithm*, respectively indicate whether  $B_c^* = B^*$ , and the search interval (lower bound/upper bound) for  $B_c^*$  obtained after applying Algorithm 1. Finally, the rightmost column *value* indicates either the optimal value  $B_c^*$  computed by Algorithm 5, or a lower/upper bound interval for  $B_c^*$ . In this column, ‘-’ indicates that the Optimization algorithm timed out.

*Results obtained by our Recycling algorithm.* Algorithm 1 (lines 1–4), by simply gathering the best knowledge from literature and (when applicable) by relying on the fact that  $B_c^* = B^*$ , can provide optimal values  $B_c^*$ , or drastically

**Table 2.** Results for 33 medium Harwell-Boeing instances. The results with an \* in the Recycling algorithm column indicates a discrepancy between the Recycling algorithm and the optimization function

Instance	vertices	edges	Inputs				Recycling algorithm		Opt. algorithm
			lb <sub>B</sub>	ub <sub>B</sub>	lb <sub>C</sub>	ub <sub>C</sub>	$B_c^* = B^*$ ?	value	value
pores_1	30	103	7	7	7	7	yes	7	7
ibm32	32	90	11	11	8	9	no	[8, 9]	9
bcspwr01	39	46	5	5	4	4	no	4	4
bcsstk01	48	176	16	16	12	12	no	12	12
bcspwr02	49	59	7	7	7	7	yes	7	7
curtis54	54	124	10	10	8	8	no	8	8
will157	57	127	6	6	6	6	yes	6	6
impcol_b	59	281	19	19	14	17	no	[14, 17]	17
steam3	80	424	7	7	7	7	yes	7	7
ash85	85	219	9	9	9	9	yes	9	9
nos4	100	247	10	10	10	10	yes	10	10
gent113	104	549	25	25	20	23	no	[20, 23]	23
bcsstk22	110	254	9	10	6	6	no	6	6
gre__115	115	267	20	22	20	23	yes*	[20, 23]	23
dwt__234	117	162	11	11	11	11	yes	11	11
bcspwr03	118	179	9	9	9	10	yes*	[9, 10]	10
lms__131	123	275	18	20	18	20	yes	[18, 20]	20
arc130	130	715	63	63	62	63	unknown	[62, 63]	63
bcsstk04	132	1758	36	37	33	37	unknown	[33, 37]	37
west0132	132	404	23	28	23	31	unknown	[23, 31]	31
impcol_c	137	352	23	27	23	24	unknown	[23, 24]	24
can__144	144	576	13	13	7	7	no	7	7
lund_a	147	1151	19	23	19	23	yes	[19, 23]	23
lund_b	147	1147	19	23	19	23	yes	[19, 23]	23
bcsstk05	153	1135	19	20	19	20	yes	[19, 20]	20
west0156	156	371	33	33	23	32	no	[23, 32]	–
nos1	158	312	3	3	3	3	yes	3	3
can__161	161	608	18	18	18	18	yes	18	18
west0167	167	489	31	34	28	34	unknown	[28, 34]	34
mcca	168	1662	32	32	32	37	yes	32	–
fs_183.1	183	701	52	58	52	58	unknown	[52, 58]	58
gre__185	185	650	17	19	17	21	yes*	[17, 21]	21
will199	199	660	55	65	34	50	no	[34, 50]	–

reduce the search interval. Indeed, we immediately conclude optimality for 28 of the 113 studied instances. Among the 85 remaining instances, the search interval is of length 2 (resp. 3) for 12 (resp. 9) instances.

**Table 3.** Results for 80 large Harwell-Boeing instances (*part 1*)

Instance	vertices	edges	Inputs				Recycling algorithm		Opt. algorithm value
			lb <sub>B</sub>	ub <sub>B</sub>	lb <sub>C</sub>	ub <sub>C</sub>	$B_c^* = B^{*?}$	value	
impcol_a	206	557	24	32	23	32	<i>unknown</i>	[23, 32]	–
dwt_209	209	767	20	23	20	23	<i>yes</i>	[20, 23]	23
gre_216a	216	660	17	21	17	21	<i>yes</i>	[17, 21]	21
dwt_221	221	704	11	13	11	13	<i>yes</i>	[11, 13]	13
impcol_e	225	1187	34	42	34	42	<i>unknown</i>	[34, 42]	–
saylr1	238	445	12	14	12	14	<i>yes</i>	[12, 14]	14
steam1	240	1761	32	44	32	44	<i>yes</i>	[32, 44]	–
dwt_245	245	608	21	21	21	21	<i>yes</i>	21	21
nnc261	261	794	22	24	22	24	<i>yes</i>	[22, 24]	24
bcspr04	274	669	23	24	23	24	<i>yes</i>	[23, 24]	–
ash292	292	958	16	19	16	19	<i>yes</i>	[16, 19]	–
can_292	292	1124	34	36	34	38	<i>yes</i>	[34, 36]	–
dwt_310	310	1069	11	12	11	12	<i>yes</i>	[11, 12]	12
gre_343	343	1092	23	28	23	28	<i>yes</i>	[23, 28]	–
dwt_361	361	1296	14	14	14	14	<i>yes</i>	14	14
plat362	362	2712	28	34	28	34	<i>yes</i>	[28, 34]	–
plskz362	362	880	14	18	14	18	<i>yes</i>	[14, 18]	–
str_0	363	2446	87	116	58	91	<i>unknown</i>	[58, 91]	–
str_200	363	3049	90	124	65	99	<i>unknown</i>	[65, 99]	–
str_600	363	3244	95	132	71	103	<i>unknown</i>	[71, 103]	–
west0381	381	2150	117	151	86	113	<i>no</i>	[86, 113]	–
dwt_419	419	1572	22	25	22	25	<i>yes</i>	[22, 25]	–
bcsstk06	420	3720	37	45	37	45	<i>yes</i>	[37, 45]	–
bcsstm07	420	3416	37	42	37	45	<i>yes</i>	[37, 42]	–
impcol_d	425	1267	36	40	24	35	<i>no</i>	[24, 35]	–
hor_131	434	2138	46	55	46	55	<i>yes</i>	[46, 55]	–
bcspr05	443	590	25	27	25	27	<i>yes</i>	[25, 27]	26
can_445	445	1682	45	52	45	46	<i>unknown</i>	[45, 46]	–
pores_3	456	1769	13	13	13	13	<i>yes</i>	13	13
bcsstk20	467	1295	8	13	8	13	<i>unknown</i>	[8, 13]	–
nos5	468	2352	52	63	52	63	<i>yes</i>	[52, 63]	–
west0479	479	1889	81	118	80	105	<i>unknown</i>	[80, 105]	–
mbeacxc	487	41686	246	260	243	243	<i>no</i>	243	–
mbeaflw	487	41686	246	261	243	243	<i>no</i>	243	–
mbeause	492	36209	249	254	245	246	<i>no</i>	[245, 246]	–
494_bus	494	586	25	29	25	28	<i>unknown</i>	[25, 28]	–
west0497	497	1715	69	85	69	81	<i>unknown</i>	[69, 81]	–
dwt_503	503	2762	29	40	29	41	<i>yes</i>	[29, 40]	–
lns_511	503	1425	33	44	33	44	<i>yes</i>	[33, 44]	–
gre_512	512	1680	30	36	30	36	<i>yes</i>	[30, 36]	–

Table 4. Results for 80 large Harwell-Boeing instances (*part 2*)

Instance	vertices	edges	Inputs				Recycling algorithm		Opt. algorithm
			lb <sub>B</sub>	ub <sub>B</sub>	lb <sub>C</sub>	ub <sub>C</sub>	$B_c^* = B^*$ ?	value	value
fs_541.1	541	2466	270	270	270	270	<i>unknown</i>	270	–
sherman4	546	1341	21	27	21	27	<i>yes</i>	[21, 27]	–
dwt_592	592	2256	22	28	22	29	<i>yes</i>	[22, 28]	–
steam2	600	6580	54	63	54	63	<i>yes</i>	[54, 63]	–
nos2	638	1272	3	3	3	3	<i>yes</i>	3	3
west0655	655	2841	109	160	94	149	<i>unknown</i>	[94, 149]	–
662_bus	662	906	36	39	36	38	<i>unknown</i>	[36, 38]	–
shl____0	663	1682	211	226	211	212	<i>unknown</i>	[211, 212]	–
shl_200	663	1720	220	231	220	220	<i>unknown</i>	220	–
shl_400	663	1709	213	230	213	215	<i>unknown</i>	[213, 215]	–
nnc666	666	2148	33	40	33	41	<i>yes</i>	[33, 40]	–
nos6	675	1290	15	16	15	16	<i>yes</i>	[15, 16]	16
fs_680.1	680	1464	17	17	17	17	<i>yes</i>	17	17
saylr3	681	1373	35	47	35	46	<i>yes</i>	[35, 46]	–
sherman1	681	1373	35	47	35	46	<i>yes</i>	[35, 46]	–
685_bus	685	1282	30	32	30	32	<i>yes</i>	[30, 32]	–
can_715	715	2975	54	72	52	60	<i>unknown</i>	[52, 60]	–
nos7	729	1944	43	65	43	65	<i>yes</i>	[43, 65]	–
mcfe	731	15086	112	126	112	126	<i>yes</i>	[112, 126]	[125, 126]
fs_760.1	760	3518	36	38	36	38	<i>yes</i>	[36, 38]	–
bcsstk19	817	3018	13	14	13	14	<i>yes</i>	[13, 14]	–
bp____0	822	3260	174	236	174	207	<i>unknown</i>	[174, 207]	–
bp__200	822	3788	186	258	186	218	<i>unknown</i>	[186, 218]	–
bp__400	822	4015	188	268	188	220	<i>unknown</i>	[188, 220]	220
bp__600	822	4157	190	272	189	229	<i>unknown</i>	[189, 229]	229
bp__800	822	4518	197	278	190	239	<i>unknown</i>	[190, 239]	239
bp__1000	822	4635	197	287	191	241	<i>unknown</i>	[191, 241]	241
bp__1200	822	4698	197	291	193	241	<i>unknown</i>	[193, 241]	241
bp__1400	822	4760	199	290	193	242	<i>unknown</i>	[193, 242]	242
bp__1600	822	4809	199	293	192	241	<i>unknown</i>	[192, 241]	241
can_838	838	4586	75	86	57	58	<i>no</i>	[57, 58]	58
dwt_878	878	3285	23	25	23	24	<i>yes</i>	[23, 24]	24
orsirr_2	886	2542	62	84	62	85	<i>yes</i>	[62, 84]	–
gr_30_30	900	3422	31	31	31	32	<i>yes</i>	31	31
dwt_918	918	3233	27	32	27	32	<i>yes</i>	[27, 32]	–
jagmesh1	936	2664	24	27	20	20	<i>no</i>	20	20
nos3	960	7442	43	43	43	43	<i>yes</i>	43	43
jpwh_991	983	2678	82	90	82	88	<i>yes</i>	[82, 88]	–
west0989	989	3500	123	210	123	217	<i>unknown</i>	[123, 217]	–
dwt_992	992	7876	35	35	35	35	<i>yes</i>	35	35

*Results obtained by our Recycling algorithm along with the Optimization function.* Tightening the search interval through Algorithm 1 is obviously advantageous for our `optimization()` function (Algorithm 5), as it is based on dichotomy. As a matter of fact, the results we obtain through “Recycling + optimization”, prove very efficient. Indeed, among the 113 initial instances, Algorithm 5 determines the optimal  $B_c^*$  value for half of them (56 cases), and the search interval is reduced in one case. Note also that, every time  $B_c^* = B^*$ , determining  $B_c^*$  or tightening its search interval also improves knowledge on  $B^*$ . All experiments are run on a computer equipped with an Intel Xeon ES 2630, 2.66 GHz processor, and coded in Python using the PyCSP<sup>3</sup> v2.2 library [24]. Each instance has up to 120 h to complete the `optimization()` function, which can lead to several calls to the CSP solver (same instance, different  $k$  values).

## 6 Conclusions and Future Work

This paper shows how the combination of pre-existing knowledge and some solving techniques can help to improve results for some hard combinatorial problems, such as the CYCLIC BANDWIDTH problem. In our case, we leverage the close relation between the BANDWIDTH MINIMIZATION and the CYCLIC BANDWIDTH problems to propose the Recycling algorithm. Our algorithm takes advantage of existing knowledge and uses it in a way that no solver is required to tighten bounds, prove optimization, or find the optimum at almost zero computational cost. For instances that require long computational runtimes, the bound tightening is a direct gain in the quest to find (or validate) optimums. In a second step, we propose an optimization function, based on constraint programming, to test the remaining values in the lower and upper bounds interval obtained in the first step. The experimental results validated the proposed approach by obtaining optimums for 56 (out of the 113) Harwell-Boeing instances and tightening bounds for another instance. We also found three instances presenting discrepancies between the Recycling algorithm and the optimization function; upon reviewing the data and running a CP-based bandwidth minimization function, it seems there may be errors in the reported values from Pop’s article. Our next steps will involve: 1) a revision of bandwidth minimization values and, 2) the study of graph properties or different modeling approaches in CP to overcome the specificities of certain instances for which the solver could not find a solution.

## References

1. Apt, K.R., Monfroy, É.: Constraint programming viewed as rule-based programming. *Theory Pract. Log. Program.* **1**(6), 713–750 (2001)
2. Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices - a survey. *J. Graph Theor.* **6**(3), 223–254 (1982)
3. Chung, F.R.: Labelings of graphs. *Sel. Top. Graph Theor.* **3**, 151–168 (1988)
4. Déprés, H., Fertin, G., Monfroy, E.: Improved lower bounds for the cyclic bandwidth problem. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds.) *ICCS 2021*. LNCS, vol. 12742, pp. 555–569. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-77961-0\\_45](https://doi.org/10.1007/978-3-030-77961-0_45)

5. Harary, F., Manvel, B.: On the number of cycles in a graph. *Matematický časopis* **21**(1), 55–63 (1971)
6. Harper, L.H.: Optimal assignments of numbers to vertices. *J. Soc. Ind. Appl. Math.* **12**(1), 131–135 (1964)
7. van Hoeve, W.J.: The all different constraint: a survey. CoRR **cs.PL/0105015** (2001). <https://arxiv.org/abs/cs/0105015>
8. Hromkovič, J., Müller, V., Sýkora, O., Vrto, I.: On embedding interconnection networks into rings of processors. In: Etiemble, D., Syre, J.-C. (eds.) *PARLE 1992*. LNCS, vol. 605, pp. 51–62. Springer, Heidelberg (1992). [https://doi.org/10.1007/3-540-55599-4\\_80](https://doi.org/10.1007/3-540-55599-4_80)
9. Lam, P.C.B., Shiu, W.C., Chan, W.H.: Characterization of graphs with equal bandwidth and cyclic bandwidth. *Discret. Math.* **242**(1–3), 283–289 (2002)
10. Lecoutre, C.: Optimization of simple tabular reduction for table constraints. In: Stuckey, P.J. (ed.) *CP 2008*. LNCS, vol. 5202, pp. 128–143. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85958-1\\_9](https://doi.org/10.1007/978-3-540-85958-1_9)
11. Leung, J.Y., Vornberger, O., Witthoff, J.D.: On some variants of the bandwidth minimization problem. *SIAM J. Comput.* **13**(3), 650–667 (1984)
12. Lin, Y.: A level structure approach on the bandwidth problem for special graphs. *Ann. N. Y. Acad. Sci.* **576**(1), 344–357 (1989)
13. Lin, Y.: The cyclic bandwidth problem. *Syst. Sci. Math. Sci.* **7** (1994)
14. Lin, Y.: Minimum bandwidth problem for embedding graphs in cycles. *Networks* **29**(3), 135–140 (1997)
15. Martí, R., Campos, V., Piñana, E.: A branch and bound algorithm for the matrix bandwidth minimization. *Eur. J. Oper. Res.* **186**, 513–528 (2008)
16. Martí, R., Laguna, M., Glover, F., Campos, V.: Reducing the bandwidth of a sparse matrix with tabu search. *Eur. J. Oper. Res.* **135**, 450–459 (2001)
17. Mladenovic, N., Urosevic, D., Pérez-Brito, D., García-González, C.: Variable neighbourhood search for bandwidth reduction. *Eur. J. Oper. Res.* **200**(1), 14–27 (2010)
18. Pop, P., Matei, O., Comes, C.A.: Reducing the bandwidth of a sparse matrix with a genetic algorithm. *Optimization* **63**(12), 1851–1876 (2014)
19. Ren, J., Hao, J., Rodriguez-Tello, E.: An iterated three-phase search approach for solving the cyclic bandwidth problem. *IEEE Access* **7**, 98436–98452 (2019)
20. Ren, J., Hao, J.K., Rodriguez-Tello, E., Li, L., He, K.: A new iterated local search algorithm for the cyclic bandwidth problem. *Knowl.-Based Syst.* **203**, 106136 (2020)
21. Rodriguez-Tello, E., Hao, J.K., Torres-Jimenez, J.: An improved simulated annealing algorithm for bandwidth minimization. *Eur. J. Oper. Res.* **185**(3), 1319–1335 (2008)
22. Rodriguez-Tello, E., Romero-Monsivais, H., Ramírez-Torres, G., Lardeux, F.: Tabu search for the cyclic bandwidth problem. *Comput. Oper. Res.* **57**, 17–32 (2015)
23. Rossi, F., van Beek, P., Walsh, T. (eds.): *Handbook of Constraint Programming, Foundations of Artificial Intelligence*, vol. 2. Elsevier (2006)
24. XCSP3 Team: *PyCSP<sup>3</sup> v2.2* (2023). <https://www.pycsp.org/>
25. Zhou, S.: Bounding the bandwidths for graphs. *Theoret. Comput. Sci.* **249**(2), 357–368 (2000)