The Balance Constraint Family
Christian Bessière, Emmanuel Hébrard, George Katsirelos, Zeynep Kiziltan,
Emilie Picard-Cantin, Claude-Guy Quimper, Toby Walsh

To cite this version:

HAL Id: lirmm-01067459
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01067459
Submitted on 15 Oct 2018

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.
The Balance Constraint Family

Christian Bessiere\(^1\), Emmanuel Hebrard\(^2\), George Katsirelos\(^3\), Zeynep Kiziltan\(^4\), Émilie Picard-Cantin\(^5\), Claude-Guy Quimper\(^5\), and Toby Walsh\(^6\)

\(^1\) CNRS, University of Montpellier, France, email: bessiere@lirmm.fr
\(^2\) LAAS-CNRS, Toulouse, France, email: hebrard@laas.fr
\(^3\) INRA, Toulouse, France, email: george.katsirelos@toulouse.inra.fr
\(^4\) DISI, University of Bologna, email: zeynep@cs.unibo.it
\(^5\) Université Laval, Canada, email: epicardcantin@petalmd.com, claude-guy.quimper@ift.ulaval.ca
\(^6\) NICTA and University of New South Wales, email: toby.walsh@nicta.com.au

Abstract. The Balance constraint introduced by Beldiceau ensures solutions are balanced. This is useful when, for example, there is a requirement for solutions to be fair. Balance bounds the difference \(B\) between the minimum and maximum number of occurrences of the values assigned to the variables. We show that achieving domain consistency on Balance is NP-hard. We therefore introduce a variant, Balance\(^\ast\) with a similar semantics that is only polynomial to propagate. We consider various forms of Balance\(^\ast\) and focus on AtMostBalance\(^\ast\) which achieves what is usually the main goal, namely constraining the upper bound on \(B\). We provide a specialized propagation algorithm, and a powerful decomposition both of which run in low polynomial time. Experimental results demonstrate the promise of these new filtering methods.

1 Introduction

In many scheduling, rostering and related problems, we want to share tasks out as equally as possible. For example, in the nurse rostering problems in [14, 4], we wish for all nurses to have a similar workload. As a second example, in the Balanced Academic Curriculum Problem (prob030 in CSPLib.org), we want to assign time periods to courses in a way which balances the academic load across periods. As a third example, when scheduling viewing times on a satellite, we might want agents to be assigned a similar number of observations. The Balance constraint introduced by Beldiceau in the Global Constraint Catalog\(^7\) [2] can be used to model situations like this where we need to minimize the difference in the number of times different values, which typically represent different resources, are used.

Beldiceau proposes an automaton based filtering algorithm for the Balance constraint that uses a counter for each value. This requires exponential space and time to work. Alternatively, Beldiceau proposes a decomposition that reorders the variables and then computes the difference between the longest and the smallest sequences of consecutive values. As we show, such a decomposition can hinder propagation. We therefore revisit this global constraint. We prove that propagating the Balance

\(^7\) http://www.emn.fr/z-info/sdemasse/gccat
constraint completely is intractable in general. We then introduce BALANCE* with a similar semantics that is only polynomial to propagate. We consider various forms of BALANCE*, focusing on ATMOSTBALANCE* which constrains the upper bound of $B$. This can be used when we desire solutions to be as balanced as possible and thus want to minimize $B$. We present a flow-based algorithm to maintain domain consistency on ATMOSTBALANCE* and compare it empirically to a decomposition augmented with implied constraints. The results show that the implied constraints significantly improve the performance of the decomposition, whilst the filtering algorithm in turn further improves performance.

The problem of ensuring a certain balance in the assignments of $[X_1, \ldots, X_n]$ has been previously studied with the SPREAD [7, 12, 11] and DEVIATION [13, 11] constraints. Both constraints look at the deviation from the mean $m = \frac{1}{n} \sum_{i=1}^{n} X_i$ with balancing criteria $D = \sum_{i=1}^{n} (X_i - m)^2$ and $D = \sum_{i=1}^{n} |X_i - m|$, respectively. The constraints in the BALANCE family are, however, different and cannot be expressed using SPREAD and DEVIATION. In particular, SPREAD and DEVIATION consider the values taken by $X_i$s, as opposed to the number of occurrences of each value. For instance, for SPREAD and DEVIATION, an assignment $[1, 2, 2, 2, 2, 2, 2, 2, 2]$ is better than $[1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3]$ as the deviation from the mean is lower in the first. For the constraints in the BALANCE family, however, the second assignment is better because the number of occurrences of 1, 2, and 3 is perfectly balanced (3 occurrences each, $B = 0$) whereas it is unbalanced in the first ($B = 6$). This criterion is very important in applications where we want to balance the occurrence of values where each occurrence of a value represents the use of a resource such as an employee or machine.

2 Background

A Constraint Network consists of a set of variables $\mathcal{X}$, a domain $D$ mapping each variable $X \in \mathcal{X}$ to a finite set of values $D(X)$, and a set of constraints $C$. An assignment $\sigma$ is a mapping from variables in $\mathcal{X}$ to values in their domains: for all $X_i \in \mathcal{X}$ we have $\sigma(X_i) \in D(X_i)$. We denote $\max(\text{D}(X))$ by $\max(X)$ and $\min(\text{D}(X))$ by $\min(X)$. When $\sigma$ is implied from the context, we write $X_i = v$ instead of $\sigma(X_i) = v$ and $X_i$ instead of $\sigma(X_i)$. A constraint $C$ is a relation on a set of variables. An assignment satisfies $C$ iff it is a tuple of this relation. We use capitals for variables and lower case for values. Constraint solvers typically use backtracking search to explore the space of partial assignments. At each assignment, propagation algorithms prune the search space by enforcing local consistency properties like domain consistency. A constraint $C$ on $\mathcal{X}$ is domain consistent (DC) if and only if, for every $X_i \in \mathcal{X}$ and for every $v \in D(X_i)$, there is an assignment $\sigma$ satisfying $C$ such that $\sigma(X_i) = v$. Such an assignment is a support. A CSP is DC iff all its constraints are DC. A constraint is disentailed iff there is no possible support.

A decomposition of a constraint $C$ is a reformulation of $C$ into a conjunction of constraints that is logically equivalent to $C$, potentially including extra variables. A decomposition $N_1$ is stronger than $N_2$ if and only if propagation on $N_1$ detects a superset of the inconsistent values detected by $N_2$ [3].
The domain of a variable $X_i$ is an interval iff $|D(X_i)| = \max(X_i) - \min(X_i) + 1$. Let $\{X_1, \ldots, X_n\}$ be a set of variables. We call $\text{occ}(v) = |\{i \mid X_i = v\}|$ the number of occurrences of the value $v$ in this set. The constraint GCC is defined over the variables $[X_1, \ldots, X_n]$ and is parameterized by two sets of integers $\{l_1, \ldots, l_m\}$ and $\{u_1, \ldots, u_m\}$. It ensures that for all $j \in [1, \ldots, m]$ we have $l_j \leq \text{occ}(v_j) \leq u_j$. Achieving DC on GCC is polynomial [10]. If the lower and upper bounds on the occurrences are given by variables $[O_1, \ldots, O_m]$, DC on the variables $X_i$ can be achieved with the same computational complexity provided that the domains of the occurrences are intervals.

3 The Balance Constraint Family

BALANCE bounds the difference in the number of occurrences of values assigned to variables.

**Definition 1 (BALANCE).**

$$\text{BALANCE}([X_1, \ldots, X_n], B) \iff B = \max_{v \in \{X_1, \ldots, X_n\}} \text{occ}(v) - \min_{v \in \{X_1, \ldots, X_n\}} \text{occ}(v)$$

Notice that only values occurring at least once are considered. Depending on the application, this may or may not be desirable. For instance, if we want to select a subset of resources and distribute tasks among them in a balanced way, then BALANCE is suited. However, if resources are already selected, then such a solution might be imbalanced as some resources may receive no tasks. BALANCE also can do limited inference. It is hard to know if a value will be used for sure until all variables are set. As a consequence, filtering is weak. We therefore consider a variant, BALANCE* in which all values in a set $V$ are considered (without loss of generality, we shall assume that $V = \{1, \ldots, m\}$). We shall see that achieving DC on BALANCE is NP-hard, while it is polynomial for BALANCE*.

**Definition 2 (BALANCE*).**

$$\text{BALANCE}^{*}(V, [X_1, \ldots, X_n], B) \iff B = \max_{v \in V} \text{occ}(v) - \min_{v \in V} \text{occ}(v) \land \forall_i X_i \in V$$

We also consider the variants of BALANCE and BALANCE* where $B$ is only a lower or an upper bound. By replacing “=” in Definition 1 and 2 by “≥” and “≤”, we define the constraints ATMOSTBALANCE, ATLEASTBALANCE, ATMOSTBALANCE* and ATLEASTBALANCE*. While ATMOSTBALANCE and ATMOSTBALANCE* ensure that a solution is “balanced enough”, ATLEASTBALANCE and ATLEASTBALANCE* ensure that the solution is “somewhat unbalanced”. The first two constraints are useful when we seek balanced solutions and want to minimize $B$, whilst the last two are useful when we cannot make $B$ lower than a certain value or desire some level of imbalance.

**Theorem 1.** Enforcing DC on BALANCE takes polynomial time if the number of values $m$ is bounded.
Proof. We construct a REGULAR constraint \([6]\) with states containing counters for every value, i.e. a state is labeled with a tuple \(\langle c_{v_1}, \ldots, c_{v_m} \rangle\) where \(c_{v_i}\) is the number of times the value \(v_i \in V\) was encountered. The unfolded automaton therefore has \(O(n^m)\) states. Enforcing DC then takes \(O(n^m + 1)\) time. One can reduce the number of counters by 1. Choose a value and delete its counter. After parsing the string, the missing counter should have value \(n - \text{sum of the other counters}\). Total complexity is then \(O(n^m)\).

\[\boxdot\]

Theorem 2. Enforcing DC on BALANCE is NP-hard.

Proof. Reduction from 3-SAT to the problem of finding a support of BALANCE. Given a formula \(\varphi\) with \(n\) atoms \(1, \ldots, n\) and \(m\) clauses, we build a BALANCE constraint finding the balance \(B\) over a set of \((n + 1)(m + 1)\) variables. Let \(l_j^k\) be the \(k\)-th literal of the \(j\)-th clause of \(\varphi\). We define the variables:

\[
\begin{align*}
B &= 0 \\
D(X_{1,j}) &= \{0\} & \forall j \in 1..m + 1 \\
D(X_{2,i}) &= \{\hat{i}, i\} & \forall i \in 1..n \\
D(X_{3,j}) &= \{l_j^1, l_j^2, l_j^3\} & \forall j \in 1..m \\
D(X_{4,l}) &= \{\hat{n}, \ldots, \hat{1}, 1, \ldots, n\} & \forall l \in 1..(n - 1)m
\end{align*}
\]

The domains of variables (3.1) and (3.2) force every value to occur 0 or \(m + 1\) times. The existence of a model of \(\varphi\) implies that there is a solution of BALANCE. Indeed, atom and clause variables (3.3,3.4) can be assigned using only the \(n\) literals appearing in the model. The total number of occurrences of these \(n\) values on the variables \(X_{2,i}\) and \(X_{3,j}\) is \(n + m\). Thanks to the \((n - 1)m\) filler variables (3.5), we can ensure that each value of these \(n\) values occurs exactly \(m + 1\) times. Thus, 0 for \(B\) is consistent.

Now consider a solution of BALANCE. Since \(B = 0\), every value occurs \(m + 1\) times or never. Since each \(X_{2,i}\) must take a value, either the value \(i\) or \(\hat{i}\) must occur at least once, hence \(m + 1\) times. There are \((n + 1)(m + 1)\) variables in total and we identified \(n + 1\) values (counting the value 0) that must occur \(m + 1\) times each. Therefore, the other values must not occur at all. The model which contains the literal \(i\) iff the value \(i\) occurs \(m + 1\) times, and \(\hat{i}\) otherwise, is a model of \(\varphi\). Indeed, for every clause \(c_j \in \varphi\), since \(X_{3,j}\) (3.4) must be assigned a value, it follows that the model above has at least one literal from \(c_j\).

\[\boxdot\]

This proof also shows that AtMostBalance is NP-hard to propagate as it only requires an upper bound on \(B\). By comparison, it is easier to reason with Balance*, AtLeastBalance*, AtLeastBalance, and AtMostBalance*. For the first three constraints, we give complexity results in the form of an algorithm intended only to prove polynomiality. For AtMostBalance*, we will present a practical filtering algorithm.

Theorem 3. Enforcing DC on Balance*, AtLeastBalance* and AtLeastBalance takes polynomial time.
Proof. Consider a restricted case of BALANCE* where the least (most) occurring value is required to be \( v_{\text{least}} \) (\( v_{\text{most}} \)) and must occur exactly \( e \) times \((c + b) \) times). DC can be enforced using GCC with \( D(O_{v_{\text{least}}}) = \{c\} \), \( D(O_{v_{\text{most}}}) = \{c + b\} \) and \( D(O_v) = \{c, c + b\} \) for all values \( v \notin \{v_{\text{least}}, v_{\text{most}}\} \). To filter BALANCE*, one can test whether a value \( v \in D(X_i) \) has a support in one of the restricted cases where \( b \in D(B) \), \( c \in [0, n - b] \), \( v_{\text{least}} \in [1, m] \), and \( v_{\text{most}} \in [1, m] \). Since there are \( O(|D(B)|nm^2) \) such cases, the filtering can be done in polynomial time.

For ATLEASTBALANCE, we set the domain \( D(O_{v_{\text{least}}}) = [1, c] \) to ensure that \( v_{\text{least}} \) occurs at least once. We set \( D(O_{v_{\text{most}}}) = [c + \min(B), n] \) so that the balance is at least \( \min(B) \) and we set the domains of the other occurrence variables to \( O_i \in [0, n] \) for \( i \notin \{v_{\text{least}}, v_{\text{most}}\} \). There are \( O(nm^2) \) restricted cases to test since \( v_{\text{least}} \in [1, m] \), \( v_{\text{most}} \in [1, m] \), and \( c \in [0, n - \min(B)] \). The restricted cases apply for ATLEASTBALANCE* except that we set the domain \( D(O_{v_{\text{least}}}) = [0, c] \) to allow the value \( v_{\text{least}} \) to occur 0 times. \( \square \)

4 Decompositions

We focus mainly on BALANCE and BALANCE*, as their decompositions can be used also for the others by suitably constraining only the lower or upper bound of \( B \). The Global Constraints Catalog [2] proposes a decomposition of BALANCE that uses the constraint SORTEDNESS([\( X_1, \ldots, X_n \)], [\( Y_1, \ldots, Y_n \)]) to count the minimum and maximum length of stretches of equal value in the sequence [\( Y_1, \ldots, Y_n \)]. Then, it makes sure that the difference between the length of the maximum and minimum stretch is equal to \( B \). We propose another decomposition of BALANCE using GCC. Let \( \bigcup_{i=1}^{n} D(X_i) = \{1, \ldots, m\} \):

\[
\begin{align*}
\text{GCC}([X_1, \ldots, X_n]; [O_1, \ldots, O_m]) & \land \\
P & = \max\{\{O_1, \ldots, O_m\}\} & \\
Q & = \min\{\{O_1, \ldots, O_m\} \setminus \{0\}\} & \\
B & = P - Q
\end{align*}
\]

As DC on BALANCE is NP-hard, it is no surprise that neither decomposition enforces DC. However, Example 1 shows that, even if we assume perfect communication between the variables [\( Y_1, \ldots, Y_n \)] and \( B \) in the decomposition using SORTEDNESS\(^8\), then this decomposition is not stronger than the new decomposition using the GCC constraint.

Example 1. \( m = 6 \), \( X_1 \in \{1, 6\} \), \( X_2 \in \{2, 5\} \), \( X_3 \in \{3, 4\} \), \( Y_1 \in \{1, 2, 3, 4\} \), \( Y_2 \in \{2, 3, 4, 5\} \), \( Y_3 \in \{3, 4, 5, 6\} \), \( B = 1 \). The domains of occurrences variables \( O_v \) are set to \( \{0, 1\} \) by GCC, hence the variables \( P \) and \( Q \) are both set to 1, and thus the constraint is found inconsistent. However, the SORTEDNESS decomposition allows stretches greater than 1. It is therefore consistent, irrespective of the reasoning used on \( Y_i \) and \( B \).

\(^8\) Such filtering can be obtained, for instance, through a REGULAR constraint.
Similar to BALANCE, the BALANCE* constraint can also be decomposed using the GCC constraint. Let \( V = \{1, \ldots, m\} \).

\[
\forall i \in \{1, \ldots, n\}, \ X_i \in V \\
\text{GCC}([X_1, \ldots, X_n], [O_1, \ldots, O_m]) \& \\
P = \max\{O_1, \ldots, O_m\} \& \\
Q = \min\{O_1, \ldots, O_m\} \& \\
B = P - Q \\
D(P) = \left\lceil \frac{n}{m} \right\rceil, n \\
D(Q) = [0, \left\lfloor \frac{n}{m} \right\rfloor]
\]

The domains of the variables \( P \) and \( Q \) are based on the observation that the average of the occurrence variables is exactly \( \frac{n}{m} \). Consequently, the greatest occurrence should be no smaller than the average and the smallest occurrence should be no greater than the average. Example 2 shows that this decomposition does not maintain DC, even on ATMOSTBALANCE*.

**Example 2.** \( m = 4 \), \( B \in [0, 2] \), \( X_1 = X_2 = 1 \), \( X_3 \in \{1, 2, 3\} \), \( X_4 \in \{1, 3, 4\} \), \( X_5 \in \{1, 3, 4\} \). After propagation, the domains of these variables remain the same and we get:

\[
O_1 \in [2, 3], \quad O_2 \in [0, 1], \quad O_3 \in [0, 3], \quad O_4 \in [0, 2], \quad P \in [2, 3], \quad Q \in [0, 1]
\]

However, the only way for the occurrence variables to sum to 5 and have a balance of at most 2 is to take their values in the multiset \( \{2, 2, 1, 0\} \) or \( \{2, 1, 1, 1\} \). In other words, a value cannot occur three times and the value 1 should be removed from the domains of \( X_3 \), \( X_4 \), and \( X_5 \).

In order to investigate the limits of a decomposition of BALANCE* based on the GCC constraint, we consider another decomposition using the constraints together with an automaton defined on the occurrence variables. Observe that we have perfect communication from the \( X_i \)’s domains to the \( O_j \)’s bounds (through GCC) and perfect communication between the \( O_j \)’s and \( B \) (through REGULAR). However, we shall see that this is still not sufficient to achieve DC on BALANCE*.

\[
\text{GCC}([X_1, \ldots, X_n], [O_1, \ldots, O_m]) \& \\
\text{REGULAR}([O_1, \ldots, O_m, B], \mathcal{A})
\]

The automaton \( \mathcal{A} \) has \( O(n^3) \) states. Non-final states are tuples \( \langle S, q, p \rangle \) which respectively encode the current sum, the minimum value encountered, and the maximum value encountered. The final state is denoted \( f \). The starting state is \( \langle 0, n, 0 \rangle \). The transition function is:

\[
\delta(\langle S, q, p \rangle, x) =
\begin{cases} 
\langle S + x, \min(q, x), \max(p, x) \rangle & \text{if } S + x \leq n \\
\langle f \rangle & \text{if } S = n \text{ and } x = p - q
\end{cases}
\]

The total complexity to propagate the REGULAR constraint is \( O(mn^4) \). This decomposition is costly yet is still insufficient to maintain DC. Consider the following example.
Example 3. $m = 4$, $B \in [0, 2]$, $X_1, X_2 = 1$, $X_3 \in \{1, 2, 3\}$, and $X_4, X_5, X_6 \in \{1, 3, 4\}$. After filtering the \textsc{Regular} constraint, we obtain the domains $O_1 \in [2, 3]$, $O_2 \in [0, 1]$, $O_3 \in [1, 2]$, and $O_4 \in [1, 2]$ that do not allow the GCC to filter 1 from the domain of $X_3$.

In the rest of the paper, we will focus on the GCC decomposition of \textsc{Balance}*. In addition to being cheaper than the \textsc{Regular} decomposition, it can be strengthened by adding implied constraints, as we will show next.

4.1 Constraints Implied by \textsc{Balance}*

We can strengthen the GCC decomposition of \textsc{Balance}* thanks to the following inequality:

$$P + (m - 1)Q \leq n.$$  (4.1)

In other words, we have an upper bound $P \leq \left\lfloor \frac{n - B}{m} \right\rfloor + B$. Consider again Example 2 which shows that the GCC decomposition of \textsc{Balance} does not maintain DC. Due to (4.1), we discover that $P \leq \left\lfloor \frac{n + (m - 1)B}{m} \right\rfloor \leq \left\lfloor \frac{5 + 3 \times 2}{4} \right\rfloor = 2$ and the upper bound of $P$, $O_1$, and $O_3$ is reduced to 2. Therefore, the constraint GCC removes 1 from the domains of $X_3$, $X_4$, and $X_5$.

We can make a similar argument to obtain a lower bound on $Q$. We have: $Q + (m - 1)P \geq n$ which is equivalent to:

$$mQ + (m - 1)B \geq n$$  (4.2)

Again in Example 2, thanks to (4.2), we discover that $Q \geq \left\lceil \frac{n - (m - 1)B}{m} \right\rceil \geq \left\lceil \frac{5 - 3 \times 0}{4} \right\rceil = 1$ and the lower bound of $Q$, $O_2$, and $O_3$ are increased to 1, and the variable $X_3$ is set to 2.

It is possible to add implied constraints providing even stronger level of filtering. The following constraints are implied by the decomposition whilst being stronger than constraints (4.1) and (4.2).

$$\left( \sum_{j=1}^{m} \max(P - B, O_j) \right) \leq n \leq \left( \sum_{j=1}^{m} \min(P, O_j) \right)$$  (4.3)

$$\left( \sum_{j=1}^{m} \min(Q + B, O_j) \right) \geq n \geq \left( \sum_{j=1}^{m} \max(Q, O_j) \right)$$  (4.4)

Indeed, consider the following example.

Example 4. $m = 7$, $X_1, X_2 \in \{1\}$, $X_3, X_4 \in \{2\}$, $X_5, X_6 \in \{3\}$, $X_7, X_8, X_9 \in \{4, 5, 6, 7\}$, $B \in \{1, 2\}$. The domains of occurrences variables $O_1, O_2, O_3$ are set to 2
and \(O_4, O_5, O_6, O_7\) to \([0, 3]\) by GCC, hence the variables \(P\) and \(Q\) are set respectively to \([0, 1]\) and \([2, 3]\), and thus the GCC decomposition as well as constraints (4.1) and (4.2) are DC. However, \(P = 3\) is not consistent with the constraint (4.3) and \(Q = 1\) is not consistent with the constraint (4.4). Therefore we can deduce \(B = 2\).

Notice that these two extra constraints require a dedicated, albeit rather straightforward, filtering algorithm because using SUM and MIN/MAX constraints would hinder propagation. The algorithm proceeds by shaving the bounds of the variables \(P\) and \(Q\). For instance, after having temporarily fixed \(P\) to its upper bound, we find a support for the relation \(\sum_{j=1}^{m} \max(P - B, O_j) \leq n\) by using the maximum value for \(B\) and the minimum value for each \(O_j\). If this is not sufficient to keep the sum below \(n\), then we can deduce that \(P = \max(P)\) is inconsistent. In Example 4, assuming \(P = 3\), we have \(P - \max(B) = 1\), and \(\sum_{j=1}^{m} \max(P - B, O_j) = 2 + 2 + 2 + 1 + 1 + 1 + 1 + 1 > 9\).

Last, we can add another cheap implied constraint. If the number of values \(m\) does not divide the number of variables \(n\), then \(B\) cannot be equal to 0. Conversely, if \(n = mk\), then \(B\) cannot be equal to 1. Indeed, suppose that the balance is 1. Furthermore, suppose that a value occurs \(k - 1\) times or less. Then since \(n = mk\), at least one other value occurs \(k + 1\) times or more, hence the balance is greater or equal to 2. The same contradiction arises if we suppose that a value occurs \(k + 1\) times or more. Therefore, the value of \(B\) cannot be equal to 1. These two rules can be combined together as follows:

\[
1 + \left\lceil \frac{n}{m} \right\rceil - \left\lfloor \frac{n}{m} \right\rfloor \neq B
\]  (4.5)

As we will show later in the empirical results, the implied constraints presented in this section turn out to be very effective in propagating \(\text{BALANCE}^*\).

4.2 Special Cases of \(\text{BALANCE}^*\)

There exist some special cases of the \(\text{BALANCE}^*\) where we have a simple encoding that does not hurt DC propagation. For instance, if \(B = n\) then all variables must be equal. We can thus post: \(X_i = X_{i+1}\) for \(1 \leq i < n\). Another case is when \(m = 2\). In this case, the implied constraints (4.1) and (4.2) reveal that: \(P \leq \frac{n + B}{2}\), \(Q \geq \frac{n - B}{2}\). Since there are two values, one occurs \(P\) times, and the other \(Q\) times, and \(P + Q = n\). Therefore, we have \(P = \frac{n + B}{2}\) and \(Q = \frac{n - B}{2}\). It follows that the value of \(B\) must be even if and only if \(n\) is even. Moreover, we can safely assume that the two values are 0 and 1 since any binary domain can be mapped to these values. Therefore, the expression \(\sum_{i=1}^{n} X_i\) gives either \(P\) or \(Q\). Thus, we post:

\[
B \mod 2 = n \mod 2 \land \sum_{i=1}^{n} X_i \in \left\{ \frac{n - B}{2}, \frac{n + B}{2} \right\}
\]

There are other cases where the decomposition with the implied constraints is sufficient.

**Proposition 1.** The GCC decomposition with the implied constraints (4.1),(4.2) and (4.5) achieves DC on \(\text{BALANCE}^*\) if \(B \leq 1\).
Proof. When $B = 0$, the implied constraints (4.1) and (4.2) entail that: $P \leq \frac{n}{m}$, $Q \geq \frac{n}{m}$. This enforces the occurrence variables $O_v$ for all $v \in V$ of the GCC decomposition to be set to $\frac{n}{m}$. Since $P, Q$ and $B$ are fixed, the constraint is now equivalent to GCC. When $B = 1$, the implied bounds are:

$$P \leq \left\lceil \frac{n-1}{m} \right\rceil + 1 \leq \left\lceil \frac{n}{m} \right\rceil, \quad Q \geq \left\lfloor \frac{n+1}{m} \right\rfloor - 1 \geq \left\lceil \frac{n}{m} \right\rceil$$

This will enforce $D(O_v) = \left[\left\lfloor \frac{n}{m} \right\rfloor, \left\lceil \frac{n}{m} \right\rceil\right]$ for all $v \in V$ of the GCC decomposition. We know that either $m$ does not divide $n$ or $B = 1$ is inconsistent. Since the latter case as already been treated, we check the former. In this case, constraint (4.5) implies that a balance of 0 is not consistent, hence $D(B) = \{1\}$. However, any assignment consistent with GCC with the bounds given by $P$ and $Q$ will have a balance of 1. Therefore, in either case, we achieve DC on $\text{BALANCE}^*$.

Another special case is as follows.

**Proposition 2.** The GCC decomposition with the implied constraints (4.1),(4.2) and (4.5) achieves DC on $\text{ATMOSTBALANCE}^*$ if $m \leq 2$.

**Proof.** Let $b = \max(B)$. The implied constraints (4.1) and (4.2) put an upper bound on $P$ of $\left\lceil \frac{n-b}{m} \right\rceil + b$, and a lower bound on $Q$ of $\left\lfloor \frac{n+b}{m} \right\rfloor - b$. Hence:

$$\max(P) - \min(Q) \leq 2b - \left\lceil \frac{n+b}{m} \right\rceil + \left\lfloor \frac{n-b}{m} \right\rfloor \leq 2b - \frac{n+b}{m} + \frac{n-b}{m} = 2b - \frac{2b}{m}$$

Now, suppose $m \leq 2$. Then $\max(P) - \min(Q) \leq b$ and for any $1 \leq j \leq m$, $\max(O_j) - \min(O_j) \leq b$. Thus, any solution of the GCC also satisfies $\text{ATMOSTBALANCE}^*$.

To summarize the results of this section, we have shown that our decomposition with the introduced implied constraints achieves DC on $\text{BALANCE}^*$ if $B \leq 1$, and on $\text{ATMOSTBALANCE}^*$ if $m \leq 2$. Moreover, there exists another decomposition achieving DC on $\text{BALANCE}^*$ if $m \leq 2$. In the general case, however, the decomposition does not achieve DC on $\text{BALANCE}^*$ and $\text{ATMOSTBALANCE}^*$ given that even a perfect communication between the variables $O_j, P, Q$ and $B$ is not enough (see Example 3).

## 5 A Filtering Algorithm for $\text{ATMOSTBALANCE}^*$

We present now a filtering algorithm that achieves DC on $\text{ATMOSTBALANCE}^*$. The algorithm (see Algorithm 1) proceeds in two steps. First, it finds a support by iteratively reducing the balance of a support for GCC until it is minimal. Second, it computes the union of the supports over each possible window of width $\max(B)$ for the values’ occurrences. The resulting union can be computed efficiently and corresponds to the domain consistent values.
Algorithm 1: FilterAtMostBalance([V, [X_1, ..., X_n], B])

\[
\begin{align*}
& \bar{b} \leftarrow \max(B); \\
& D(X') \leftarrow D(X_i) \text{ for all } i = 1..n; \\
& 1 \text{ Find a support } \sigma \text{ for } \text{ATMOSTBALANCE}^* \text{ whose balance is minimal and let } q \text{ be the occurrence of the least occurring value;} \\
& \quad \text{Set } \min(B) \text{ to be the balance of the support } \sigma; \\
& \quad D(O_i) \leftarrow [q, q + b] \text{ for all } i = 1..m; \\
& 2 \text{ filter GCC}([D(X_1), ..., D(X_n)], [O_1, ..., O_m]); \\
& \quad \text{if no filtering occurred then return;} \\
& \quad \text{if filtering occurred because of a Hall set then } k \leftarrow 1; \\
& \quad \text{else } k \leftarrow -1; \\
& \quad D(O_i) \leftarrow [q + k, q + \bar{b} + k] \text{ for all } i = 1..m; \\
& \quad \text{filter GCC}([D(X'_1), ..., D(X'_n)], [O_1, ..., O_m]); \\
& \quad D(X_i) \leftarrow D(X_i) \cup D(X'_i) \text{ for all } i = 1..n;
\end{align*}
\]

5.1 Finding a Support

To find a support, the algorithm computes a flow in a graph similar to the one used for the GCC. There is one node \( X_i \) per variable, one node \( v \) per value, a source \( s \), and a sink \( t \). Each edge has a capacity \([a, b]\), i.e. a lower capacity \( a \) and an upper capacity \( b \). There is an edge of capacity \([0, 1]\) between \( s \) and each variable node \( X_i \). There is an edge of capacity \([0, 1]\) between each node \( X_i \) and value \( v \) for \( v \in D(X_i) \). Finally, there is an edge of capacity \([0, n]\) between each value \( v \) and \( t \). Let \( f(a, b) \) be the amount of flow that circulates from node \( a \) to \( b \). A maximum flow \([1]\) from \( s \) to \( t \) corresponds to an assignment of the variables, i.e., \( X_i = v \iff f(X_i, v) = 1 \). The value \( v \) occurs exactly \( f(v, t) \) times in the assignment. To modify the assignment so that it satisfies the \( \text{ATMOSTBALANCE}^* \) constraint, the algorithm finds a path in the residual graph from the most occurring (or least occurring) value \( v \) to any value \( v' \) such that \( f(v', t) \leq f(v, t) - 2 \) (or such that \( f(v', t) \geq f(v, t) + 2 \)). The algorithm pushes a unit of flow along this path to modify the assignment. The algorithm repeats this operation until no such path exists. If no such path exists and if the balance of the current assignment is strictly greater than \( \max(B) \), then no support exists. To prove correctness, we show that if there is a solution of \( \text{ATMOSTBALANCE}^* \), then there is a sequence of such paths leading to it from any maximum flow. In other words, if no such path exists, then the gap between the maximum and minimum flow going through an edge for a value node to the sink node is a lower bound of \( B \).

**Lemma 1.** If \( v \) is the most occurring value and there is no value \( v' \) such that \( f(v', t) \leq f(v, t) - 2 \) and that \( v \) can reach \( v' \) in the residual graph and if \( w \) is the least occurring value and there is no value \( w' \) such that \( f(w', t) \geq f(w, t) + 2 \) and that \( w \) can reach \( w' \) in the residual graph, then the balance of the current assignment is minimal.

**Proof.** We prove the contraposition. Suppose there is a flow \( f^* \) whose corresponding assignment has a smaller balance than the assignment given by \( f \). Let the most occur-
ring and least occurring values in each flow be:

\[ v_{\text{most}} = \arg\max_{v \in V} f(v, t), \quad v_{\text{least}} = \arg\min_{v \in V} f(v, t), \]

\[ v^*_{\text{most}} = \arg\max_{v \in V} f^*(v, t), \quad v^*_{\text{least}} = \arg\min_{v \in V} f^*(v, t). \]

Necessarily, we have \( f(v_{\text{most}}, t) > f^*(v^*_{\text{most}}, t) \lor f(v_{\text{least}}, t) < f^*(v^*_{\text{least}}, t) \). Suppose that \( f(v_{\text{most}}, t) > f^*(v^*_{\text{most}}, t) \), we have \( f(v_{\text{most}}, t) > f^*(v^*_{\text{most}}, t) \geq f^*(v_{\text{most}}, t) \). Since both flows have the same flow value, the difference of the vectors \( f^* - f \) describes a circulation, i.e. a collection of cycles on which the flow circulates. Since \( f^*(v_{\text{most}}, t) - f(v_{\text{most}}, t) < 0 \), the flow circulates from \( t \) to \( v_{\text{most}} \) in the circulation which means that there is a value \( v' \) for which the flow circulates from \( v' \) to \( t \) which implies \( f^*(v', t) - f(v', t) > 0 \). We conclude that \( f(v_{\text{most}}, t) > f^*(X^*_{\text{most}}, t) \geq f^*(v', t) \) and \( f(v_{\text{most}}, t) \geq f(v', t) + 2 \). Finally, the edges \( (v', t) \) and \( (t, v) \) lie on the same cycle in the circulation. Hence there is a path that connects \( v \) to \( v' \) in the residual graph. A symmetric reasoning applies if we suppose that \( f(v_{\text{least}}, t) < f^*(v^*_{\text{least}}, t) \).

5.2 Filtering the Domains

First, we filter the lower bound of \( B \) to the balance value of the support found in the first phase. The balance of this solution is, by Lemma 1, the maximum lower bound on \( B \). Next, we set \( q = \min_v f(v, t) \) to be the frequency of the least occurring value. Let \( \tilde{b} = \max(B) \). We then run the filtering algorithm of the GCC with the domains of the occurrence variables set to \([q, q + \tilde{b}]\). If this does no filtering, then each value in the domains belongs to a support where the occurrences of the values lie between \( q \) and \( q + \tilde{b} \). All these supports satisfy the AtMostBalance* and we are done. However, if the filtering algorithm detects the assignment \( X_i = v \) is inconsistent for the GCC, it is not necessarily inconsistent for AtMostBalance*. The assignment \( X_i = v \) can occur in a support where the maximum and minimum number of occurrences do not belong to \([q, q + \tilde{b}]\). Therefore, we need to test for a support with different domains such as \([q - 1, q + \tilde{b} - 1]\) and \([q + 1, q + \tilde{b} + 1]\). Fortunately, we do not need to test all possible intervals of size \( \tilde{b} \). A Hall set is a set of values \( H \) for which exactly \( (q + \tilde{b}) \times |H| \) variable domains are subset of \( H \), since \( q + \tilde{b} \) is the maximum allowed occurrences for any value in \( H \). Conversely, an unstable set is a set of values \( U \) for which exactly \( q \times |U| \) variable domains intersect \( U \). From [8], an assignment \( X_i = v \) is filtered either because \( v \) belongs to a Hall set or the domain of \( X_i \) intersects with an unstable set. The following lemmas restrict search to two windows.

**Lemma 2.** If \( H \) is a Hall set for a GCC with occurrences bounded by \( q \) and \( q + \tilde{b} \) and \( k \) a positive integer, then the bounds \( q - k \) and \( q + \tilde{b} - k \) are inconsistent.

**Proof.** Since \( H \) is a Hall set when the upper bound is equal to \( q + \tilde{b} \), then there are \((q + \tilde{b}) \times |H|\) variables whose domains are included in \( H \). Therefore the total number of values in \( H \) is at least \((q + \tilde{b}) \times |H|\). Therefore at least one value must occur at least \( q + \tilde{b} \) times. This is a contradiction with the upper bound \( q + \tilde{b} - k \).
The dual result for unstable sets can be obtained in a similar way (proof omitted):

**Lemma 3.** If \( U \) is an unstable set for a GCC with occurrences bounded by \( q \) and \( q + \bar{b} \) and \( k \) a positive integer, then the bounds \( q + k \) and \( q + 1 + k \) are inconsistent.

Using these two lemmas, we can show that we only need to check the window \([q + 1, q + \bar{b} + 1]\) if the pruning was due to Hall sets only, the window \([q - 1, q + \bar{b} - 1]\) if it was due to unstable sets only, and no other window otherwise. The following theorem follows.

**Theorem 4.** Enforcing DC on AtMostBalance* takes \( O(n^2 m) \) time.

**Proof.** First, the pruning on \( B \) is correct by Lemma 1. Since \( B \) is only an upper bound, its own upper bound is never pruned. It follows that the pruning on \( B \) is complete and a support is found if and only if the constraint is not disentailed. Now, consider Algorithm 1. Let \( \bar{b} = \max(B) \) and \( q \) be the minimum occurrence of any value found in the support (Line 1). We compute all consistent values for a GCC constraint where occurrence variables are bounded by \([q, q + \bar{b}]\). Suppose first that no pruning occurs. Then every value is consistent for GCC. Each support is such that the difference between maximum and minimum occurrence is less than or equal to \( \bar{b} \). Hence it is a support for AtMostBalance*.

Suppose now that there is at least one Hall set. Then by Lemma 2, we know that a GCC on the same variables but with all occurrence variables bounded by \([q - k, q + \bar{b} - k]\) would be inconsistent. In other words, these values have no support for AtMostBalance* on lower windows. By Lemma 2, since the GCC is consistent for the window \([q, q + \bar{b}]\), there will not be any Hall set on higher windows. It follows that values inconsistent for GCC on the window \([q, q + \bar{b}]\) are inconsistent for AtMostBalance* only if they are pruned because of an unstable set on \([q + 1, q + \bar{b} + 1]\) and all higher windows. However, by Lemma 3, if there is an unstable set on the window \([q + 1, q + \bar{b} + 1]\), then all higher windows will be inconsistent. It follows that values pruned by GCC on the windows \([q, q + \bar{b}]\) are inconsistent if and only if they are also pruned on the window \([q + 1, q + \bar{b} + 1]\). The second case is when there is at least one unstable set when setting the occurrence variables to the window \([q, q + \bar{b}]\). Symmetrically, values are inconsistent if and only if they would be pruned also for the window \([q - 1, q + \bar{b} - 1]\). Finally, if the window \([q, q + \bar{b}]\) has both a Hall and an unstable set, then all other windows are inconsistent.

The running time is bounded by the time to find a support. This requires finding \( O(n - \max(B)) \) augmenting paths, each with a depth-first search (DFS) in \( O(nm) \) time. The two calls to the filtering algorithm of GCC take \( O(n^{3/2}m) \) time. Finding what caused the filtering uses a DFS in the transposed residual graph that marks nodes that can reach the sink \( t \). If the value \( v \) was filtered out of the domain of \( X_i \) and that \( v \) cannot reach the sink, then the filtering occurred because of a Hall set. Otherwise, it occurred because of an unstable set. The total complexity is thus \( O(n^2 m) \). □

In practice, the complexity can be considerably reduced. For instance, the support \( \sigma \) can remain valid for multiple consecutive calls to the filtering algorithm. In such a situation, the running time is equivalent to executing the filtering algorithm of GCC twice. If the support is no longer valid, it can usually be updated rather than computed from scratch. This involves finding much fewer than \( n - \max(B) \) augmenting paths.
6 Experimental Results

We evaluate our DC algorithm for ATMostBALANCE∗ and its decompositions on the Balanced Academic Curriculum Problem (BACP) and a shift scheduling problem. All experiments are carried out using Choco (version 2.1.5) running under Linux on a 3Ghz processor with 12 GB of ram.

6.1 Balanced Academic Curriculum Problem

In BACP (prob030 in CSPLib.org), a set of $n$ courses must be assigned to $m$ time periods, such that (1) a lower and an upper bound on the number of courses per period must be respected; (2) some courses are prerequisite for others; (3) the load (i.e., the sum of the credits of the assigned courses) of each period should be balanced.

The “standard” model [5] has a variable $X_i$ for each course $i$, whose value is the period allocated to this course. A variable $O_j$ for each period $j$ gives the load on this period. In order to channel these two sets, $\{0, 1\}$ variables $Y_{i,j}$ are introduced and constrained such that $X_i = j \iff Y_{i,j} = 1$ and $O_j = \sum_{i=1}^{n} c(i) \cdot Y_{i,j}$ where $c(i)$ stands for the number of credits of a course $i$. Constraint (1) is modeled by a GCC constraint on $\{X_1, \ldots, X_n\}$, constraints (2) are simple precedences between the corresponding $X_i$ and $X_j$. For the objective (3), a criterion is minimized representing how balanced the set $\{O_1, \ldots, O_m\}$ is. In [5], the criterion is the maximum load. We here consider the gap between the minimum and maximum load, denoted $L(\infty)$ in [13]. We thus have three variables $P$, $Q$ and $B$ with the constraints $P = \max(\{O_1, \ldots, O_m\})$, $Q = \min(\{O_1, \ldots, O_m\})$ and $B = P - Q$, and we minimize $B$.

We propose an alternative model which respects $L(\infty)$ using our BALANCE∗ constraint. We have the same variables $\{X_1, \ldots, X_n\}$ with the same constraints for (1) and (2). However, we do not need $Y_{i,j}$ and $O_j$. We can directly post BALANCE∗ using $V = \{1, \ldots, m\}$ and $B$ on the multiset of variables containing $c(i)$ times each $X_i$.

We report the results obtained by running 7 models on three real instances involving 8, 10 and 12 periods. Standard is the first model described above. The following four models correspond to the alternative model using BALANCE∗. The first uses the basic GCC decomposition for BALANCE∗ (Decomp.), the second uses the decomposition with the implied constraints (4.1) and (4.2) (IMPLIED), the third uses the implied constraints (4.3) and (4.4) (IMPLIED∗), and the fourth uses the DC algorithm (Balance∗). Finally, in the two last models we balance the load using the DEVIATION constraint [13] on the load variables $O_j$. The way that DEVIATION is used differs in two models. In the former model (Deviat.), following [13], $O_j$ variables are channelled to the original $X_i$ variables via the $Y_{i,j}$ variables as done in the standard model. In the latter model (Gcc+Deviat.), the $O_j$ variables are channelled to the duplicated $X_i$ variables using a GCC. That is, we use a GCC and a DEVIATION constraint together to balance the load. We then report the $L(\infty)$ value of the best solutions provided by these two models. Notice that (almost) perfectly balanced solutions exist for the instances we used, thus the $L(\infty)$ and deviation criteria are very close to each other.

As branching strategy, we use Impact Based Search [9] in all cases. Each instance is run 20 times with a 900s cutoff by each method after randomly shuffling the variables so that initial ties are broken randomly. We report in Table 1 the average observed balance
Table 1: Balanced Academic Curriculum Problem

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>#</td>
<td>Time</td>
<td>B</td>
<td>#</td>
<td>Time</td>
<td>B</td>
</tr>
<tr>
<td>08</td>
<td>16.3</td>
<td>-</td>
<td>1.020</td>
<td>116</td>
<td>1.020</td>
<td>112</td>
</tr>
<tr>
<td>16</td>
<td>15.2</td>
<td>-</td>
<td>1.020</td>
<td>2626</td>
<td>1.020</td>
<td>2665</td>
</tr>
<tr>
<td>12</td>
<td>31.6</td>
<td>-</td>
<td>2.1</td>
<td>19104</td>
<td>2.015</td>
<td>21984</td>
</tr>
</tbody>
</table>

$(B = L(\infty))$ over the 20 runs, and the number of runs where optimality was proven (#). Moreover, when possible, we report the average run time in milliseconds (Time) over all completed runs (i.e., in which optimality was proven). In other words, CPU times can be compared only when the same set of instances have been solved by all methods. As also shown in [5], the standard model rarely solves the instances to optimality which renders difficult the computation of averages for which optimality is proven by all methods. We use **bold** to highlight the best results. When multiple methods solve the same instances, we also highlight the best average CPU time.

We observe that the standard CP model has extremely poor performance, the solutions found are all suboptimal. Notice that the criterion optimized by the Deviation models is different. Several symmetric solutions for the $L(\infty)$ criterion have a different deviation. Indeed, we can see that the Deviat. model, which is the same as the standard model where the simple objective function is replaced by Deviation, greatly outperforms the standard model. However, neither of the Deviation models is able to find an optimal solution and prove it in every case. The models using the decompositions of the Balance* constraint is very efficient on instances 08 and 10, however, only the filtering algorithm is able to find an optimal solution and prove it within the cutoff time in all cases.

### 6.2 Shift Scheduling

In order to better assess the advantages of the propagator over the different decompositions with or without implied constraints, we ran another series of tests (done in the same conditions).

We consider a task assignment problem. We have $m$ tasks per day. Each task requires a separate worker so we have $m$ workers. Over the $n$ days of the schedule, we want each worker to receive an assignment as balanced as possible. We have one variable $X_{i,j}$ per worker $i$ and per day $j$. We make sure that on any day $j$, all tasks are performed by distinct workers through an All-Different constraint over the variables $[X_{1,j}, \ldots, X_{m,j}]$. We bound the balance of the tasks assigned to each worker $i$ by a shared variable $B$ which we minimize with AtMostBalance* over $[X_{i,1}, \ldots, X_{i,n}]$.

To make problems hard, we ensure that not all workers are available for every task every day. Given a ratio $0 \leq \alpha < 1$, we randomly forbid $\lceil an^2m \rceil$ triples $(i, j, k)$ for which we remove the value $k$ from the variable $X_{i,j}$ (so that worker $i$ cannot do task $k$ on day $j$). We make sure that $i \in X_{i,j}$ for all $i$ and $j$, to ensure a feasible solution exists. We randomly generated instances for 6 to 8 workers/tasks ($m$) and 16 to 20 days ($n$). For each pair $(m, n)$, we generated 25 instances with a ratio of unavailability $\alpha$ ranging from 0.1 to 0.58 by increments of 0.02.
We compare the basic GCC decomposition \((\text{Decomp.})\), the decompositions with implied constraints \((\text{Implied})\) and \((\text{Implied}^+)\), and the DC algorithm \((\text{Balance}^*)\). We report the same statistics as for BACP, however, the averages are obtained over the values of \(\alpha\) instead of over random runs. A static variable and value ordering was used so that the decrease in number of backtracks is only due to stronger propagation. Consequently we also report the average number of backtracks, again only computed on instances solved to optimality within the cutoff.

We can clearly see that the implied constraints have a huge impact for a very low overhead. On the smaller instances, while the filtering algorithm saves backtracks, it is almost twice as slow (in terms of backtracks per second) as either decomposition with implied constraints. It is nevertheless almost always faster, but only by a small margin. As the instances get larger, the benefits of the algorithm over the decompositions, and of the stronger decompositions over the weaker ones, become more evident. Indeed, the algorithm allows to prove optimality in 84% of the cases for \(m = 8\), whereas the decompositions \((\text{Decomp.}, \text{Implied}, \text{Implied}^+)\) can only do it in 13%, 74% and 76% of the cases, respectively. Moreover, still for \(m = 8\) the objective value is decreased 48%, 16% and 12% in average with respect to these three decompositions.

### 7 Conclusions

We have studied constraints for ensuring solutions are balanced. We first proved that enforcing domain consistency on the AMOSTBALANCE and therefore on the BALANCE constraint is NP-hard. This is due to the disjunctive choice in the semantics of BALANCE that ignores a value which does not occur. We therefore introduced a variant, BALANCE* with a similar semantics in which all values are considered. We provided a specialized propagation algorithm, and a powerful decomposition both of which work in low polynomial time. Experimental results demonstrated the promise of these new filtering methods.

| \(m\) | \(n\) | # | \(B\) | Time | Bkt | # | \(B\) | Time | Bkt | # | \(B\) | Time | Bkt | # | \(B\) | Time | Bkt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 16 | 8 | 1.92 | 66 | 65 | 7 | 1.88 | 37 | 74 | 7 | 1.88 | 35 | 42 | 7 | 1.88 | 33 | 60 |
| 6 | 17 | 11 | 2.16 | 107 | 76 | 7 | 1.88 | 63 | 87 | 25 | 2.16 | 36 | 249 | 25 | 2.16 | 36 | 249 |
| 6 | 18 | 8 | 3.2 | 166 | 114 | 7 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 |
| 6 | 19 | 8 | 3.2 | 106 | 693 | 7 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 |
| 6 | 20 | 7 | 3.04 | 260 | 25 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 | 25 | 2.64 | 504 | 692 |
| 7 | 16 | 6 | 4.44 | 325 | 4764 | 7 | 4.14 | 2112 | 238 | 7 | 4.14 | 2112 | 238 | 7 | 4.14 | 2112 | 238 |
| 7 | 17 | 7 | 1.96 | 84 | 9080 | 7 | 1.96 | 6697 | 7236 | 25 | 1.96 | 1576 | 9378 | 25 | 1.96 | 1576 | 9378 |
| 7 | 18 | 9 | 1.76 | 2366 | 71 | 7 | 1.76 | 14300 | 183069 | 24 | 1.68 | 13920 | 90685 | 24 | 1.68 | 13920 | 90685 |
| 7 | 19 | 4 | 1.88 | 2198 | 13840 | 24 | 1.76 | 6378 | 36822 | 22 | 1.76 | 13920 | 90685 | 22 | 1.76 | 13920 | 90685 |
| 7 | 20 | 2 | 1.56 | 46267 | 600434 | 23 | 1.76 | 6378 | 36822 | 22 | 1.76 | 13920 | 90685 | 22 | 1.76 | 13920 | 90685 |
| 8 | 16 | 8 | 1.2.12 | 1286 | 2109236 | 8 | 1.02 | 17420 | 231594 | 25 | 0.72 | 34216 | 462257 | 25 | 0.44 | 3797 | 14999 |
| 8 | 17 | 7 | 1.88 | 1579 | 715789 | 8 | 1.88 | 14300 | 183069 | 24 | 1.68 | 13920 | 90685 | 24 | 1.68 | 13920 | 90685 |
| 8 | 18 | 1 | 1.76 | 56785 | 542326 | 16 | 1.56 | 5264 | 15636 | 16 | 1.56 | 5264 | 15636 | 16 | 1.56 | 5264 | 15636 |
| 8 | 19 | 2 | 1.40 | 64074 | 462257 | 15 | 1.40 | 51899 | 68059 | 15 | 1.40 | 51899 | 68059 | 15 | 1.40 | 51899 | 68059 |
| 8 | 20 | 2 | 1.40 | 64074 | 462257 | 15 | 1.40 | 51899 | 68059 | 15 | 1.40 | 51899 | 68059 | 15 | 1.40 | 51899 | 68059 |
References