When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent*

Jérémie du Boisberranger† Danièle Gardy‡ Xavier Lorca§ Charlotte Truchet¶

Abstract

This article presents new work on analyzing the behaviour of a constraint solver, with a view towards optimization. In Constraint Programming, the propagation mechanism is one of the key tools for solving hard combinatorial problems. It is based on specific algorithms: propagators, that are called a large number of times during the resolution process. But in practice, these algorithms may often do nothing: their output is equal to their input. It is thus highly desirable to be able to recognize such situations, so as to avoid useless calls. We propose to quantify this phenomenon in the particular case of the AllDifferent constraint (bound consistency propagator). Our first contribution is the definition of a probabilistic model for the constraint and the variables it is working on. This model then allows us to compute the probability that a call to the propagation algorithm for AllDifferent does modify its input. We give an asymptotic approximation of this probability, depending on some macroscopic quantities related to the variables and the domains, that can be computed in constant time. This reveals two very different behaviors depending of the sharpness of the constraint. First experiments show that the approximation allows us to improve constraint propagation behaviour.

1 Constraint Solvers and Complexity Analysis.

1.1 Constraint Programming. This article explores a mathematical way to predict the behavior of some algorithms, called *propagators*, used in *Constraint Programming* [9].

Constraint Programming (CP) aims at solving hard combinatorial problems expressed as *Constraint Satisfaction Problems* (CSP). A CSP is made of three parts: the first one is a set of *variables* that represents the unknowns of the problem; the second part is a set of finite *domains* describing the possible (usually integer) values of each variable; the third part contains the *constraints* which express a combinatorial relation between the variables. A constraint can be built on classical logical predicates, for instance $x = y + 2$, or on specific relations called *global constraints* [2]. Some global constraints extend the language expressivity by adding predicates that could not be formulated with basic predicates. Others could be reformulated as classical constraints, but they encapsulate a global relationship that can be used to improve the algorithms implementing the constraints and the resolution process. For instance, the AllDifferent constraint forces a list of variables to take different values; it is semantically identical to $n(n-1)/2$ constraints of the type $x_i \neq x_j$ on all the possible couples of the variables. A *solution* for a CSP is an assignment of each variable to a value in its domain such that all the constraints are simultaneously satisfied.

CP operational nature is based on the *propagation-search* paradigm. The *propagation* mechanism detects and suppresses inconsistent parts of the domains, *i.e.* values that cannot appear in a solution. For instance, let us consider the constraint $x = y + 2$: assuming that the domain for x is $D_x = [1...20]$ and the domain for y is $D_y = [3...30]$, it is obvious that only the values $[5...20]$ for x and $[3...18]$ for y must be considered; the other values in D_x and D_y cannot appear in a solution, and are thus inconsistent. [1] For each particular constraint, a specific algorithm, called *propagator*, removes inconsistent values from the domains: it takes as an input the domains of the variables involved in the constraint, and outputs the corresponding consistent domains, which are called *globally arc-consistent* domains (GAC). In practice and depending on the constraint, a propagator does not always remove all the inconsistent values. When reaching GAC is too costly, propagators may only consider the bounds of the domains, and only

*Supported by the ANR-BOOLE project, France.

†Université de Versailles, PRiSM UMR 8144, FR-78035 Versailles, France, Jeremie.Du-Boisberranger@prism.uvsq.fr

‡Université de Versailles, PRiSM UMR 8144, FR-78035 Versailles, France, Daniele.Gardy@prism.uvsq.fr

§École des Mines de Nantes, LINA UMR CNRS 6241, INRIA, FR-44307 Nantes Cedex 3, France, Xavier.Lorca@mines-nantes.fr

¶Université de Nantes, LINA UMR CNRS 6241, INRIA, FR-44322 Nantes, France, Charlotte.Truchet@univ-nantes.fr

[1] A problem may be consistent without having a solution. Consider for example the constraints $x \neq y$, $y \neq z$ and $z \neq x$ with the domains $D_x = D_y = D_z = \{1, 2\}$: we cannot suppress values in the domains; yet no solution exists.

those bounds are ensured to be consistent. This weaker property is called *bound-consistency* (BC). More generally, the efficiency of propagators is decomposed in different classes [5]. Because there are several constraints in the problem, all the constraints are iteratively propagated until a *fixed point* is reached, *i.e.* all the domains are stable for all the propagators; see for example [1] for an overview of constraint propagation and questions relative to confluence.

It is obvious that, most of the time, propagation is not sufficient to solve the CSP. The domains can be stable, but still contain more than one value. A *search engine* then iteratively instantiates values to the variables, until either a solution or a failure (constraint always false, empty domains) is found. In case of a failure, a special mechanism, called *backtracking*, allows to return to the last choice point and try another value for the variable under consideration. Every time a choice is made (e.g. a particular value has been instantiated to a variable) propagation is run in order to remove the inconsistent values of the domains as soon as possible and to avoid useless computation. Due to the combinatorial nature of the problems, this makes the worst-case number of calls to the propagators exponential in the number of variables.

1.2 Motivating Example. We illustrate the way a constraint solver works on the following toy scheduling problem. Consider a construction site where six different tasks, each one lasting one day, have to be scheduled (eventually in parallel) within a given duration of 5 days. They are represented by their starting time, which is an integer value between 1 and 5. We have six variables $T_1 ... T_6$, with domains $D_1 ... D_6$ initially equal to [1...5]. The construction starts at time $S = 0$ and ends at $E = 6$.

Some tasks need to be finished before some other tasks begin (*e.g.* the walls need to be built before the roof). This is modeled by precedence constraints, which are inequalities between the two involved tasks. In addition, some tasks may be done in parallel, but others require the same equipments, so they cannot be executed at the same time (*e.g.* there is only one concrete mixer, required to build both the walls and the floor). This is modeled by AllDifferent constraints on the involved tasks.

This leads to a so-called scheduling problem with precedence constraints. We will detail the example shown on Figure 1 where the variable are the nodes, the precedence constraints the edges in red (C_1 to C_{10}) and the AllDifferent constraint the green ellipse around the variables it involves (C_{11}).

Let us work out in detail the initial propagation.

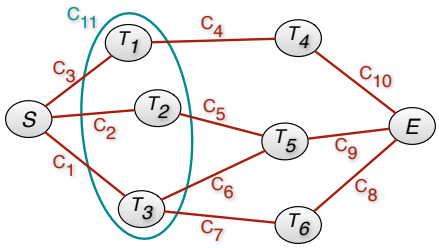

Figure 1: Example of a scheduling problem with precedence constraints and an AllDifferentconstraint.

First, D_4, D_5 and D_6 are reduced to [2...5] by propagation of C_4, C_5, C_6 and C_7. Conversely, propagating these precedence constraints from the end yields $D_1 = D_2 = D_3 = [1...4]$. At this stage, the AllDifferent constraint C_{11} should be propagated as well but it cannot reduce the domains in practice.

Then the search begins, and a variable, say T_4, is assigned a value, say 3. The precedence constraint C_4 is propagated to reduce D_1 to [1...2], but no propagation of C_{11} can be done: it can be checked that the remaining problem is still consistent. Thus a second choice is made, say T_5 is assigned to 4. Again, the remaining problem is already consistent and C_{11} is not propagated – although propagation of C_5 and C_6 reduces both D_2 and D_3 to [1...3]. So a third choice is made, say T_6 is assigned to 3. Now C_7 can be propagated and $D_3 = [1...2]$. No other propagation of precedence constraints can be done. Now the C_{11} constraint is not bound-consistent: [1...2] is equal to the union of the two domains D_1 and D_3, and we have two values to allocate to two variables T_1 and T_3; this leads to reducing D_2 to [4], hence assigning T_2.

This very simple example shows that the propagation of AllDifferent constraints may have an effect on the variables' domains only quite late in the search (here, after the third assignment on a problem with six variables), and is unlikely to have any effect on the domains when they are large enough, as is the case at the beginning of the search.

1.3 Cost of Propagation and Complexity Trade-offs. The propagators have an algorithmic cost which, most of the time, is a sizeable part of the propagation engine computing time. Practical experiments highlight that the algorithmic effect (*i.e.*, reduction of the variables' domains) of the propagators is not uniformly distributed during the resolution process. In other words, these algorithms are frequently called when solving a hard combinatorial problem, but often do

nothing. This phenomenon is rarely explored by the CP community, where most of the research efforts on propagation algorithms are focused on their worst-case time complexity; we refer the reader to [9] for a global reference and further studies.

In [7], Katriel identifies one of the major issues in studying propagation: the pursuit of a fair balance between efficiency (time complexity) and effective performance (number of inconsistent values detected by the propagators). The author proposes a particular mechanism to decrease the number of calls to the propagators in the case of the global cardinality constraint during the search process. She shows that only subsets of the values in the variable domains are important for the propagation engine, and proposes to delay the propagation algorithm calls until a certain number of values are removed from the domains. The main limit of this work remains the algorithmic cost related to the detection of these important values, which is never amortized during the search process.

Obviously, observing the worst-case complexity of a propagator does not give enough information on its usefulness. Average time complexity is certainly very difficult to obtain and has never been studied. As shown by Katriel, the problem of freezing calls to useless propagators (a weak, yet interesting way of measuring the algorithm efficiency) is tricky for two reasons. Firstly, missing an inconsistent value leads to important needless computations afterwards. Secondly, in order to know whether a propagator is useful or not, we need to have an indicator that can be computed much faster than the evaluation of the propagator itself.

We propose here a theoretical study of the behavior for a specific propagator, the one associated to the AllDifferent constraint. Given a set of variable domains, we provide a probabilistic indicator that allows us to predict if the AllDifferent propagator for bound-consistency will detect and remove some inconsistent part of these variable domains. We then show that such a prediction can be asymptotically estimated in constant time, depending on some macroscopic quantities related to the variables and the domains. Experiments show that the precision is good enough for a practical use in constraint programming. Compared to [7], we tackle on the same question but provide, within another model, a computable approximation for the effective performance of the algorithm, measured as the probability it does remove at least one value.

In the next Section we first give a formal definition of the bound consistency for the constraint AllDifferent, then consider how we can characterize situations where the constraint AllDifferent will not restrict any of the variables' domains and propagation should be avoided, and finally present a probabilistic model for variables and their domains. Section 3 presents exact and asymptotic formulae for the probability that the propagation of the constraint AllDifferent will have no effect on the domains, and deals with some issues related to the computation of this probability. Finally, Section 4 considers applying our results to an actual solver, what we can hope to gain, and the problems that we face. Sketches of proofs can be found in the Appendix.

2 A Probabilistic Model for AllDifferent.

The AllDifferent constraint is a well-known global constraint for which many consistency algorithms have been proposed. The reader can refer to the surveys of Van Hoeve [10] or Gent et al. [6] for a state of the art.

The property of being bound consistent (BC) applies to all global constraints. Intuitively, *a constraint on n variables is bound consistent if, assuming the domains of the n variables to be intervals, whenever we choose to affect any variable to either its minimal or maximal value, it is possible to find a global affectation of the remaining $n-1$ variables that satisfies the global constraint.* This intuition can be formalized, and we give below a mathematical characterization of bound consistency for AllDifferent. We next introduce a probabilistic model for Bound Consistency of AllDifferent and consider how we can check whether an AllDifferent constraint, initially BC, remains so after an instantiation.

2.1 Definitions and Notations. Consider an AllDifferent constraint on n variables $V_1...V_n$, with respective domains $D_1...D_n$ of sizes d_i, $1 \leq i \leq n$. We assume the size of each domain to be greater than or equal to 2, otherwise the corresponding variable is already instanciated. We focus here on bound consistency, as proposed by [8], and we assume that *all the domains D_i are integer intervals.*

We now introduce some notations.

- The union of all the domains is $E = \bigcup_{1 \leq i \leq n} D_i$. Notice that, up to a relabelling of the values of the D_i, their union E can also be assumed to be an integer *interval* without loss of generality.
- For a set I, we write $I \subset E$ as a shortcut for : $I \subset E$ and I is an integer interval.
- For an interval $I \subset E$, we write $\underline{I}$ for its minimum bound and $\overline{I}$ for its maximum bound; hence $I = [\underline{I}...\overline{I}]$.

With these notations, we can now give the classical definition of bound-consistency for AllDifferent.

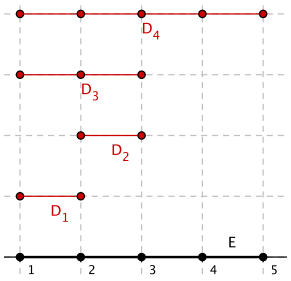

Figure 2: Unconsistent domain configuration for an AllDifferent constraint.

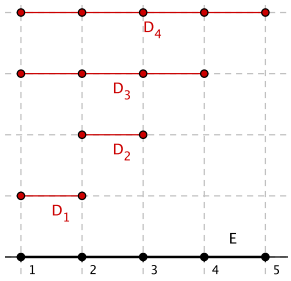

Figure 3: Consistent domain configuration for an AllDifferent constraint.

Definition 2.1. *Let* AllDifferent$(V_1...V_n)$ *be a constraint on n variables. It is* bound-consistent *iff for all $i \neq j$, $1 \leq i, j \leq n$, the two following statements hold:*

- $\exists v_j \in [\underline{D_j}...\overline{D_j}]$ *s.t.* AllDifferent$(v_1...v_{i-1}, \overline{D_i}, v_{i+1}...v_n)$;
- $\exists v'_j \in [\underline{D_j}...\overline{D_j}]$ *s.t.* AllDifferent$(v'_1...v'_{i-1}, \underline{D_i}, v'_{i+1}...v'_n)$.

An example of non bound-consistent domains for an AllDifferent constraint with four variables is shown on Figure 2: the domains for $V_1...V_4$ are respectively $[1...2], [2...3], [1...3]$ and $[1...5]$, and it can be checked that the value 1, the lowest bound of D_4, cannot appear in a solution.

For the example of Figure 3, the domains $[1...2], [2...3], [1...4]$ and $[1...5]$ are bound-consistent, since all the extremal values of the domains can be extended to a solution.

Definition 2.2. *Let $I \subset E$. We define K_I as the set of variables for which the domains are subintervals of I: $K_I = \{i \text{ such that } D_i \subset I\}$.*

For instance, on Figure 2 we have $K_{[1...3]} = \{1, 2, 3\}$, $K_{[3...5]} = \emptyset$ and $K_{[1...2]} = \{1\}$.

Some subintervals of E play a special rôle: they contain just enough values to ensure that every variable of K_I can be assigned a value. [2] Consequently the variables that do not belong to K_I cannot take their values in I. This leads to the following proposition, from [10].

[2] The subintervals I of E s.t. $|K_I| = |I|$ are called *Hall intervals;* they frequently appear in characterizations of bound cosnsistency for AllDifferent.

Proposition 2.1. *An* AllDifferent *constraint on a set of variables $V_1...V_n$ with a set of domains $D_1...D_n$ is* bound-consistent *if and only if the two following conditions are true:*

1. *for all $I \subset E$, $|K_I| \leq |I|$,*
2. *and for all $I \subset E$, $|K_I| = |I|$ implies $\forall i \notin K_I, I \cap \{\underline{D_i}, \overline{D_i}\} = \emptyset$.*

For example, on Figure 2 the domain $D_3 = [1...3]$ is of size 3, and so is $K_{[1...3]}$; hence this interval satisfies the first condition of Prop. 2.1. But it does not satisfies the second condition, since the lowest bound of D_4, which is 1, has a non-empty intersection with $[1...3]$. On the contrary, on Figure 3 all the subintervals of E satisfy the proposition, since every $I \subset E$ is strictly bigger than the associated set K_I.

For technical reasons, we reformulate Proposition 2.1 into the equivalent

Proposition 2.2. *An* AllDifferent *constraint on a set of variables $V_1...V_n$ with a set of domains $D_1...D_n$ is* bound-consistent *if and only if for all $I \subset E$, one of the following condition is true:*

1. *$|K_I| < |I|$,*
2. *$|K_I| = |I|$ and $\forall i \notin K_I, I \cap \{\underline{D_i}, \overline{D_i}\} = \emptyset$.*

This property is useful to determine BC of an AllDifferent constraint as a whole, at the beginning of the resolution for instance. In practice, the problem (or rather the data) is constantly modified during the solving process. Thus we are also interested in answering the following question:

> *Knowing that an* AllDifferent *constraint is initially BC, under which conditions does it remain BC after the instanciation of a variable?*

2.2 Bound Consistency After an Instanciation. We consider here the effect of an instantiation on the domains and on bound consistency. Up to a renaming of the variables, we can assume w.l.o.g. that the instantiation is done for the variable V_n, which is assigned a value $x \in D_n$. We also assume that the binary constraints $\neq$ have been propagated (that is, the x value has been removed from the other domains when applicable).

After the instanciation, the situation is thus the following: D_n has disappeared (or is reduced to $\{x\}$), and for $i \neq n$, two cases can occur. If $x \notin D_i$, then the domain remains unchanged, and if $x \in D_i$, the domain D_i is now the union of the two disjoint intervals $[\underline{D_i}...x-1]$ and $[x+1...\overline{D_i}]$. The question of bound consistency thus seems no longer relevant, because the domains are no longer intervals. However, we can define new domains $D_i^{'}=D_i \setminus \{x\}$, which are not subintervals of E, but of

$$E^{'} := [\underline{E}...x-1] \cup [x+1...\overline{E}]$$

because all the values of $E^{'}$ between $\underline{D_i}$ and $\overline{D_i}$ are in $D_i^{'}$. This leads to the following

Definition 2.3. *The* AllDifferent *constraint* remains BC *after the instanciation of* V_n *iff* AllDifferent *is BC with respect to the new domains* $D_1^{'}...D_{n-1}^{'}$.

Moreover, for every subinterval $I^{'} \subset E^{'}$, we define an associated interval $I \subset E$ as

$$I = \begin{cases} I^{'} \cup \{x\} \text{ if it is a subinterval of } E; \\ I \text{ otherwise.} \end{cases}$$

The following Proposition details in which cases the constraint, being BC on $V_1...V_n$, *remains* BC (as defined above) after the instantiation of V_n. This result, although not usually explicitly stated as such, belongs to the folklore of CP; we present it for the sake of completeness.

Proposition 2.3. *With the above notations, the* AllDifferent *constraint remains BC after the instanciation of* V_n *to a value* x *iff for all* $I^{'} \subset E^{'}$, *such that* $I = I^{'} \cup \{x\}$ *and* $D_n \not\subset I$, *none of the two following statement holds:*

(i) $|K_I| = |I|$,

(ii) $|K_I| = |I| - 1$ *and there exists* $i \notin K_I$ *such that* $\underline{D_i} \in I$ *or* $\overline{D_i} \in I$.

2.3 A Probabilistic Model for Variables and Domains. The key ingredients that determine the consistency of an AllDifferent constraint are the domain sizes and their relative positions. But we do not always need to know precisely the domains, to decide whether the constraint is consistent or not. For instance, an AllDifferent constraint on three variables with domains of size 3 is always BC, whatever the exact positions of the domains. If the domains are of size 2, then the constraint may be BC or not, depending on the relative positions of the domains. But if their union E is also of size 2, the constraint is always inconsistent.

Such basic remarks show that a partial knowledge on the domains sometimes suffices to determine consistency properties. This is the basis of our probabilistic model: we assume that we know the union E of the domains (which can indeed be observed in usual cases), but the domains themselves become discrete random variables with a uniform distribution on the set $\mathcal{I}(E)$ of subintervals of E, of size ≥ 2: they are not fully determined. Their exact sizes and positions are unknown; only some macroscopic quantities are known.

We consider first the union $E = \bigcup_{i=1}^{n} D_i$ of the domains, which is assumed to satisfy the following assumption:

> **A1.** *E is an integer interval of known size m; w.l.o.g. we take $E = [1...m]$.*

Assume from now on that the domains $D_1, ..., D_n$ are replaced by random variables $\mathcal{D}_1, ..., \mathcal{D}_n$, as follows.

> **A2.** *The variables $\mathcal{D}_i$ are independent and uniformly distributed on $\mathcal{I}(E) = \{[a...b], 1 \leq a < b \leq m\}$.*

As a consequence of Assumption **A2**, the sample space $\mathcal{I}(E)$ has size $m(m-1)/2$ (we recall that we forbid domains of size 1). For $J \subset E$ and $1 \leq i \leq n$, we have $P[D_i = J] = 2/m(m-1)$. Indeed, there are $m-1$ possible subintervals of size 2, $m-2$ of size 3, ..., 1 of size m.

3 The Probability of Remaining BC.

This Section details the evaluation of the probability that an AllDifferent constraint remains BC after an instantiation, under the assumptions **A1** and **A2** of Section 2.3. We establish a general formula for this probability, then compute its asymptotic value in the case where the observable variables are large; we also show that this asymptotic approximation can be computed in constant time.

3.1 Exact Results. We first consider some intermediate probabilities that we shall use in order to write down the probability of remaining BC.

Proposition 3.1. *For a given interval $I \subset E$ and a domain $\mathcal{D}$ drawn with a uniform distribution on $\mathcal{I}(E)$,*

with $m = |E|$ and $l = |I|$, let p_l and q_l be the respective probabilities that $\mathcal{D} \subset I$, and that either $\mathcal{D} \cap I = \emptyset$, or $\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}$. Then

$$p_l = \frac{l(l-1)}{m(m-1)}; \qquad q_l = \frac{(m-l)(m-l-1)}{m(m-1)}.$$

In order to compute the probability that the constraint remains BC, we now inject into Proposition 2.3 the quantities we have just obtained, which leads to the following result.

THEOREM 3.1. *Consider an* ALLDIFFERENT *constraint on variables $V_1, ..., V_n$, initially BC. Let E and the domains $\mathcal{D}_i$, $1 \leq i < n$, satisfy the assumptions* **A1** *and* **A2**. *Furthermore, assume that we know the domain $D_n = [a...b]$. Then the probability $P_{m,n,x,a,b}$ that the constraint remains BC after the instanciation of the variable V_n to a value x is*

$$P_{m,n,x,a,b} = \prod_{l=1}^{n-2} (1 - P^{(1)}_{m,n,l} - P^{(2)}_{m,n,l})^{\Phi(m,l,x,a,b)}$$

where the function $\Phi(m, l, x, a, b)$ is defined as

$$\min(x, m-l) - \max(1, x-l) + 1$$

if $l < b - a$ and as

$$\min(x, m-l) - \max(1, x-l) - \min(a, m-l) + \max(1, b-l)$$

otherwise, and where

$$P^{(1)}_{m,n,l} = \binom{n-1}{l+1} p_{l+1}^{l+1} (1 - p_{l+1})^{n-l-2},$$

$$P^{(2)}_{m,n,l} = \binom{n-1}{l} p_{l+1}^{l} \left((1 - p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1} \right),$$

with p_l and q_l given by Proposition 3.1.

3.2 Asymptotical Approximation. We recall that n is the number of variables and that the union of their domains has size m. From the expression of $P_{m,n,x,a,b}$ given in Theorem 3.1, we can compute the probability $P_{m,n,x,a,b}$ in time $O(n)$. However, if we are to use a probabilistic indicator for the bound consistency as part of a solver, this indicator will be computed repeatedly, and we must be able to do this in a reasonably short time even when n and m are both large. Thus the formula of Theorem 3.1 cannot be used as such, and we need an approximation of it, both precise and that can be computed "quickly enough". The following proposition gives such an asymptotic approximation in a scale of powers of $1/m$. Two different behaviors arise, depending on how n compares to m. In the first case n is proportional to m, which corresponds to an ALLDIFFERENT constraint with many values and few variables. In the second case $m - n = o(m)$, which corresponds to a sharp ALLDIFFERENT constraint where there are nearly as many values as variables.

THEOREM 3.2. *Consider an* ALLDIFFERENT *constraint on n variables $V_1, ..., V_n$. Assume that the domains $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$ follow a uniform distribution on E. Let $D_n = [a...b]$. Define a function $\Psi(m, x, a, b)$, for $1 \leq a \leq x \leq b \leq m$ and $a \neq b$, by*

- $\Psi(m, x, a, a+1) = 1$, *except* $\Psi(m, 1, 1, 2) = \Psi(m, m, m-1, m) = 0$;
- *If $b > a + 1$ then $\Psi(m, x, a, b) = 2$, except $\Psi(m, 1, 1, b) = \Psi(m, m, a, m) = 1$.*

Then the probability $P_{m,n,x,a,b}$ that the constraint remains BC after the instanciation of V_n to the value x has asymptotic value

- *if $n = \rho m$, $\rho < 1$,*

$$1 - \Psi(m, x, a, b) \frac{2\rho(1 - e^{-4\rho})}{m} + O\left(\frac{1}{m^2}\right);$$

- *if $n = m - i$, $i = o(m)$,*

$$e^{C_i} \left(1 - \Psi(m, x, a, b) \frac{2(1 - e^{-4}) + D_i}{m} + O\left(\frac{1}{m^2}\right)\right).$$

When $n = m - i$ with $i = o(m)$, the constants C_i and D_i (which also depend on x and a) can be expressed as

$$\sum_{a-i \leq j < x-i} (j + i + 1 - a)\varepsilon_{i,j} + (x - a) \sum_{j \geq x-i} \varepsilon_{i,j},$$

with $\varepsilon_{i,j}$ equal to $\log(1 - f_{i,j})$ for C_i and to $g_{i,j}/(1 - f_{i,j})$ for D_i, where we set

$$f_{i,j} = \lambda_{i,j} \left(1 + \frac{j}{2(i+j)}\right)$$

and

$$g_{i,j} = \lambda_{i,j} \left(i(i+1) + \frac{j}{4}(3i+5) + \frac{j(i^2-1)}{4(i+j)}\right)$$

with

$$\lambda(i, j) = \frac{(i+j)^j 2^j e^{-2(i+j)}}{j!}.$$

Theorem 3.2 is important for two reasons. In the first place, it gives the quickly-computable probabilistic indicator that we expected (see the discussion in the

next Section). Then, it also exhibits two different behaviors for an AllDifferent constraint. It thus formalizes and gives a rigorous proof of what is folklore knowledge in CP: the sharpness of the AllDifferent constraint (the ratio of n over m) is a key ingredient for the efficiency of its propagation. It can be seen from the expression of Theorem 3.2 that the probability of remaining BC has an asymptotic limit equal to 1 in the first case, and to a constant strictly smaller than 1 in the second case. Thus, for large m, propagation of AllDifferent is almost surely useless unless $m - n = o(m)$, that is, unless m and n are very close.

3.3 Practical Computation of the Indicator. We have just seen that Theorem 3.2 gives an asymptotic approximation (when m becomes large) for the probability of remaining BC.

When $n = \rho\, m$ for fixed ρ, we have a closed-form expression for the approximation, that can be computed in constant (small) time. In the case $n = m - i$ the constants C_i and D_i, although not depending on m and n (they do depend on a and x), have no closed-form expressions but are given as limits of infinite sums. Nevertheless, a good approximation can be obtained with a finite number of terms. Indeed, the terms $f_{i,j}$ and $g_{i,j}$ are exponentially small for fixed i and $j \to +\infty$. E.g., the value $\log(1 - f_{i,j})$ is roughly of exponential order $(2/e)^j$, and so is the general term of the series: the convergence towards the limit is quick enough for fast computation.

Another practical question is: how do we choose between the two cases of the formula? that is, how do we decide when n is "close enough" to m? Theorem 3.2 is valid for $m \to \infty$, but splits into two cases according to whether n is such that m/n remains roughly constant, or $m - n = o(m)$.

A numerical example is shown on Fig. 4, where the theoretical probability (plain) and its approximation (dashed) are plotted for $m = 25$ and $m = 50$, and for a varying ratio n/m. (In both figures, the values x, a and b were arbitrarily fixed at 15, 3 and 9 respectively.) The dashed curve actually has a discontinuity at some point: we applied Case 1 for $n \leq m - 2\sqrt{m}$ and Case 2 for $n > m - 2\sqrt{m}$, which appears from our computations to be the best compromise. Even though $m = 50$ is not a large value, the approximation already fits closely the actual values.

In practice, numerical evaluations for varying values of m and n do indicate that the best compromise is indeed to set a threshold at $n = m - 2\sqrt{m}$ and to use it to distinguish between the two cases of Theorem 3.2. Notice that this threshold can be used as a quantitative definition for the sharpness of the constraint, which

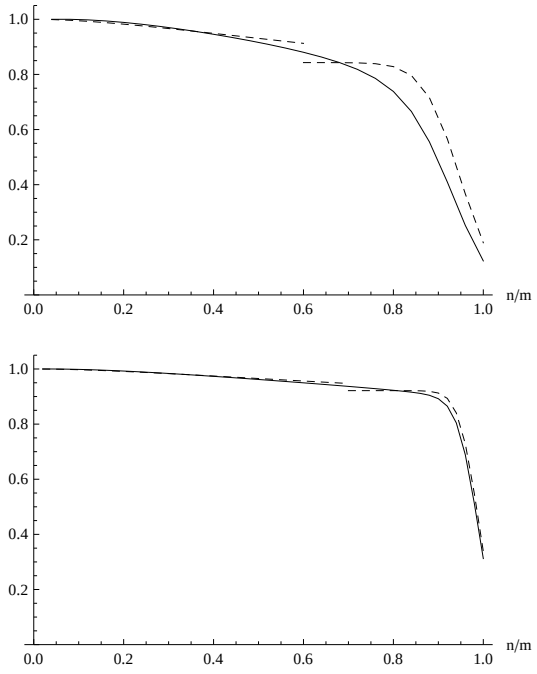

Figure 4: Numerical evaluation of the theoretical (plain) and approached (dashed) probability $P_{25,n,15,3,19}$ for $m = 25$ (top) or 50 (bottom), and for n varying from 1 to m.

leads us to propose the following

Definition 3.1. *An* AllDifferent *constraint on n variables and m values is* sharp *iff $m - n < 2\sqrt{m}$.*

4 Conclusion and Further Work.

We have presented here a probabilistic framework for modeling the AllDifferent constraint and for checking whether it is worthwhile to propagate it. We have obtained the probability that the constraint does reduce the sizes of domains, and given an asymptotical formula that can be computed in constant time.

Like all models, ours relies on a simplified view of reality, and this simplification is expressed by our mathematical hypotheses **A1** and **A2**. A fundamental point is whether these hypotheses (union of domains an a specific interval, independance of the domains, identically distributed and uniform distribution) are valid in real-life situations, and what is the robustness of our results if not.

The first assumption presents no difficulty: it can always be satisfied by a renaming of the values of the domains.

As long as we limit ourselves to the single constraint AllDifferent, the assumption that the domains can

be modelized as independent variables seems reasonable (the interaction of the variables is restricted to this constraint). Of course such independence may no longer hold, should we wish to take into account several constraints: extending our model would require us to consider precisely how different constraints might interact.

The next point we wish to discuss: the identical, uniform distribution for the domains, is due to the modelization choices we have made. Our approach can be extended to other kinds of distributions; indeed we considered in the technical report [3] the situation where domains have a common, fixed size. Such a case happens quite often, for instance in problems coming from Artificial Intelligence (a classical example is the n-queens puzzle). To our knowledge, these two cases should be sufficient to model a lot of real-life scenarios. The extension of our results to situations where domains follow different probability distributions alos seems possible.

We are currently working on implementing our results into the *Choco* constraint solver [4]. This integration raises several practical questions. Theorem 3.2, which gives a quick approximation of the probability of remaining BC, is valid in the limit for $m \to \infty$. In practice, this means that m is required to be large, *i.e.* our result holds at the beginning of the search process. The next issue is the characterization of a threshold value for the probability of remaining BC: the AllDifferent constraint would be propagated when the probability falls under this value, and not propagated when above. Another important point is, when should the evaluation of the probability be undertaken by the solver? If this is done every time a variable appearing in the AllDifferent constraint is modified, this quickly leads to an important number of evaluations (typically, 10^5 to 10^6 calls for a standard problem). Indeed, most of the time the modification that can arise on a variable domain is a value removal, without direct impact on the probability that the constraint remains consistent. Thus, the evaluation of the probability should take place only when a variable is instantiated to a value. Finally we observe that, even if computing the probability is theoretically done is constant time, the practical computation cost still has to be amortized.

First results seem to indicate that, from an operational point of view, some problems with an AllDifferent constraint are more efficiently solved when this constraint is replaced by the equivalent clique of difference constraints. This means that we have to tackle problems for which the AllDifferent propagation algorithm is not required all the time. We hope to be able to answer all these questions in a further paper.

References

[1] K. R. Apt, *The Essence of Constraint Programmation*, Theoretical Computer Science, 221, 1-2, (1999), pp. 179-210.

[2] C. Bessière and P. van Hentenryck, *To Be or Not to Be... a Global Constraint*, in Principles and Practice of Constraint Programming (2003), pp. 789-794.

[3] J. du Boisberranger, D. Gardy, X. Lorca, and C. Truchet, *A Probabilistic Study of Bound Consistency for the Alldifferent Constraint*, http://hal.archives-ouvertes.fr/hal-00588888/en/.

[4] Choco Team, *Choco: an Open Source Java Constraint Programming Library*, Ecole des Mines de Nantes, RR 10-02-INFO (2010).

[5] R. Debruyne and C. Bessière, *Domain Filtering Consistencies*, J. Artificial Intelligence Research, 14 (2001), pp. 205-230.

[6] I. P. Gent and I. Miguel and P. Nightingale, *Generalised Arc Consistency for the* AllDifferent *Constraint: An Empirical Survey*, Artif. Intell., 172/18 (2008), pp. 1973-2000.

[7] I. Katriel, *Expected-Case Analysis for Delayed Filtering*, in CPAIOR (2006), Springer Lecture Notes in Computer Science, vol. 3990, pp. 119-125.

[8] J.-F. Puget, *A Fast Algorithm for the Bound Consistency of* AllDifferent *Constraints*, AAAI/IAAI (1998), pp. 359-366.

[9] F. Rossi and P. van Beek and T. Walsh, Handbook of Constraint Programming (Foundations of Artificial Intelligence) (2006), Elsevier Science Inc.

[10] W.J. van Hoeve, *The* AllDifferent *Constraint: A Survey*, CoRR, cs.PL/0105015 (2001).

Appendix.

4.1 Proposition 2.3. We present in Figure 5 a summary of the different cases that must be checked, organized into a tree. Let $I' \subset E'$.

First of all, let us remark that $I = I'$ implies $|K_{I'}| = |K_I|$. By consistency, we have either $|K_I| < |I|$, or $|K_I| = |I|$ and the domains not in K_I have no bound inside. If $|K_I| < |I|$, then $|K_{I'}| < |I'|$ and I' satisfies Proposition 2.2. If $|K_I| = |I|$, all the domains not in K_I have no bound in I, and this also holds for I'. Thus, I' satisfies again Proposition 2.2.

From now on, let us suppose that $I = I' \cup \{x\}$. We have $|I'| = |I| - 1$. If $D_n \subset I$, then $|K_{I'}| = |K_I| - 1$. Again, $|K_{I'}| \leq |I'|$ and all the domains have no bound in $K_{I'}$, so I' satisfies Proposition 2.2. Thus inconsistency is only possible if $I = I' \cup \{x\}$ and $D_n \not\subset I$.

Let us suppose furthermore that $D_n \not\subset I$. If $|K_I| < |I| - 1$, no inconsistency is possible. If $|K_I| = |I|$, we have $|K_{I'}| = |K_I| = |I| = |I'| + 1$ and I' does not satisfy Proposition 2.2. Finally, if $|K_I| = |I| - 1$, we have $|K_{I'}| = |K_I| = |I| - 1 = |I'|$, and I' satisfies

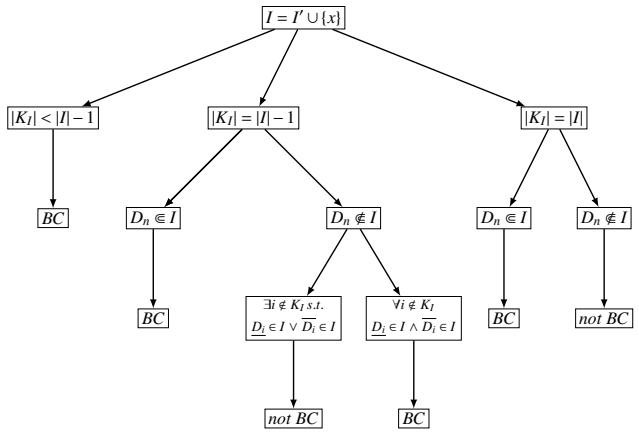

Figure 5: The different subcases of Proposition 2.3.

Proposition 2.2 iff a domain not belonging to K_I has no bound in I (the condition is the same for I and $I^{'}$).

4.2 Proposition 3.1. The probability that a domain $\mathcal{D}$, following a uniform distribution over $\mathcal{I}(E)$, has size $d \geq 2$, is

$$P[|\mathcal{D}| = d] = \frac{2(m-d+1)}{m(m-1)}.$$

The probability that a domain of size d is a subinterval of I is the number of subintervals of E of size d which are also subintervals of I, divided by the total number of subintervals of E of size d. Thus

$$P[\mathcal{D} \subset I \mid |\mathcal{D}| = d] = \max\left(\frac{l-d+1}{m-d+1}, 0\right).$$

Then

$$\begin{aligned} p_l &= \sum_{d=2}^{m} P[\mathcal{D} \subset I \mid |\mathcal{D}| = d] * P[|\mathcal{D}| = d] \\ &= \sum_{d=2}^{l} \frac{2(l-d+1)}{m(m-1)} = \frac{l(l-1)}{m(m-1)}. \end{aligned}$$

Besides, we have $q_l = P[\mathcal{D} \cap I = \emptyset] + P[\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}]$ since both events are incompatible. The first term $P[\mathcal{D} \cap I = \emptyset]$ is the probability that $\mathcal{D}$ is a subinterval of $[\underline{E}...\underline{I}[$ or a subinterval of $]\overline{I}...\overline{E}]$:

$$P[\mathcal{D} \cap I = \emptyset] = \frac{\underline{I}(\underline{I}-1) + (m-l-\underline{I})(m-l-\underline{I}-1)}{m(m-1)}.$$

For the second term, the number of domains which satisfy $\underline{\mathcal{D}} < \underline{I}$ and $\overline{\mathcal{D}} > \overline{I}$ is the number of choices for the lower bound in $[\underline{E}...\underline{I}[$, times the number of choices for the upper bound in $]\overline{I}...\overline{E}]$, i.e. $\underline{I}.(m-l-\underline{I})$, which gives

$$P[\underline{\mathcal{D}} < \underline{I} \wedge \overline{\mathcal{D}} > \overline{I}] = \frac{2\underline{I}(m-l-\underline{I})}{m(m-1)}.$$

The formula for q_l comes easily by summing the two contributions.

4.3 Theorem 3.1. Following Proposition 2.3, we define

$$\begin{aligned} P^{(1)}[I] &= P(|K_I| = |I|); \\ P^{(2)}[I] &= P(|K_I| = |I| - 1; \exists i \notin K_I : \underline{D_i} \in I \vee \overline{D_i} \in I). \end{aligned}$$

We have that

$$P_{m,n,x,a,b} = \prod_I \left(1 - P^{(1)}[I] - P^{(2)}[I]\right),$$

where the product is on the subintervals I of E, such that $x \in I$ and $D_n \not\subset I$. Let $I^{'} = I \setminus \{x\}$; we have $I^{'} \subset E^{'} = E \setminus \{x\}$. Define $l = |I^{'}|$ and α as the position in E of $\underline{I^{'}}$: as $x \in I$, we have $|I| = l + 1$.
Now $P^{(1)}[I]$ is the probability that $|I|$ domains are included in I, that is $l+1$ domains among the domains $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$ are included in I. In the same vein, $P^{(2)}[I]$ is the probability that $|I|-1$ domains are included in I and at least one of the other domains (not included in I) has a bound in I, which is equivalent to saying that l domains among $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$ are included in I and all the others are not included in I, but at least one of them has a bound inside I. Hence

$$\begin{aligned} P^{(1)}(I) &= \binom{n-1}{l+1} p_{l+1}^{l+1} \ (1-p_{l+1})^{n-l-2}; \\ P^{(2)}(I) &= \binom{n-1}{l} p_{l+1}^{l} \ \left((1-p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1}\right). \end{aligned}$$

The quantities $P^{(1)}[I]$ and $P^{(2)}[I]$ depend only on m, n and l, hence the probability we seek will be of the type

$$P_{m,n,x,a,b} = \prod_l (1 - P^{(1)}[I] - P^{(2)}[I])^{\Phi(l)}$$

where the index l is the length of $I^{'}$, and the exponent $\Phi(l)$ may also depend on the parameters m, x, a, and b. (We use here the notation $\Phi(l)$ for the sake of conciseness, whereas we write $\Phi(m, l, x, a, b)$ in the statement of Theorem 3.1.) Before computing $\Phi(l)$, we first point out that the product on l ranges from 1 to $n-2$: both $P^{(1)}[I]$ and $P^{(2)}[I]$ are null for $l \geq n-1$.

The set $I^{'}$ is characterized by α and l[3]. What are the constraints on these values? Obviously, since $|E|^{'} = m-1$ and $I^{'} \subset E^{'}$, we must have $\alpha > 0$

[3]We remind the reader that $I^{'}$ is a subinterval of $E^{'}$ but (generally) not of E.

and $\alpha + l \leq m$. The condition $x \in I$ corresponds to $\alpha \leq x \leq l + \alpha$. Both conditions together are equivalent to

$$\max(x - l, 1) \leq \alpha \leq \min(x, m - l). \tag{4.1}$$

Now if $l < b - a$ there is no possibility that $D_n \subset I$; hence the number of intervals I of length $l + 1$ is equal to the number of suitable α, *i.e.*

$$\Phi(l) = \min(x, m - l) - \max(x - l, 1) + 1.$$

The situation is slightly more intricate if $l \geq b - a$: I has length greater than or equal to D_n, and we must take care that $D_n \not\subset I$. This last condition corresponds to $a < \alpha$ or $b > \alpha + l$, i.e. $b - l \leq \alpha \leq a$ is forbidden. As $1 \leq \alpha \leq m - l$, this means that we forbid the values α such that

$$\max(b - l, 1) \leq \alpha \leq \min(a, m - l). \tag{4.2}$$

If we now substract from the possible number of values for α, given by Equation (4.1), the number of values that satisfy Equation (4.2), i.e. $\min(a, m-l) - \max(b-l, 1) + 1$, to , we obtain the expression for $\Phi(l)$.

4.4 Simplifying Φ. We now give explicit values for the function $\Phi(l)$. We have just proved that $\Phi(m, l, x, a, b)$ is equal to $\min(x, m-l) - \max(1, x-l) + 1$ if $l < b - a$, and to $\min(x, m - l) - \max(1, x - l) - \min(a, m - l) + \max(1, b - l)$ otherwise. This can be further simplified, as follows.

4.4.1 Case $l < b - a$.

	$l < m - x$	$l \geq m - x$
$l < x$	$l + 1$	$m - x + 1$
$l \geq x$	x	$m - l$

4.4.2 Case $l \geq b - a$.

To compute the explicit value of Φ, we have to compare l to $m - x$, $m - a$, x and b. This should give 16 cases, but the condition $x \in I$ forbids some of them:

- $x \leq b$, hence $b \leq l < x$ is forbidden.
- $a \leq x$ which gives $m - x \leq m - a$, hence $m - a \leq l < m - x$ is forbidden.

The value of $\min(a, m-l) - \max(1, b-l) + 1$ is given by the following table.

	$l < m - a$	$l \geq m - a$
$l < b$	$a - b + l + 1$	$m - b + 1$
$l \geq b$	a	$m - l$

This gives the following tables (NA stands for those cases that cannot happen), where we can easily check that all values are ≥ 0.

	$l < m - x$		
		$l < m - a$	$l \geq m - a$
$l < x$	$l < b$	$b - a$	NA
	$l \geq b$	NA	NA
$l \geq x$	$l < b$	$x + b - a - l - 1$	NA
	$l \geq b$	$x - a$	NA

	$l \geq m - x$		
		$l < m - a$	$l \geq m - a$
$l < x$	$l < b$	$m + b - a - x - l$	$b - x$
	$l \geq b$	NA	NA
$l \geq x$	$l < b$	$m + b - 2l - a - 1$	$b - l - 1$
	$l \geq b$	$m - l - a$	0

4.5 Theorem 3.2. Simple asymptotics give the approximate expression for $P_{m,n,x,a,b}$ with $\Psi(m, x, a, b) = \Phi(m, 1, x, a, b)$, which can easily be simplified according to the tables of the preceding Section.

When $n = m - i$ with $i = o(m)$, we have that

$$C_i = \lim_{t\to\infty} \sum_{j=1}^{\sqrt{t}} \Phi(t, t - i - j - 1, x, a, b) \log(1 - f_{i,j});$$

$$D_i = \lim_{t\to\infty} \sum_{j=1}^{\sqrt{t}} \Phi(t, t - i - j - 1, x, a, b) \frac{g_{i,j}}{1 - f_{i,j}}.$$

The constants C_i and D_i are both of the type

$$\lim_{t} \sum_{j=1}^{\sqrt{t}} \Phi(t, t - i - j - 1, x, a, b)\ \varepsilon_{i,j},$$

with $\varepsilon_{i,j}$ equal to $\log(1 - f_{i,j})$ for C_i and to $g_{i,j}/(1 - f_{i,j})$ for D_i.

We again use the expression of Φ given in Section 4.4 to simplify these constants as follows. We compute them for fixed x, a, b and i, for large t, and for j ranging from 1 to $\sqrt{t}$. In this range, we can assume that t is large enough for $t - j - (i + 1)$ to be always larger that (x and) b. Then

- for $j \leq a - i - 1$, we have that $\Phi = 0$;
- for $a - i - 1 \leq j \leq x - i - 1$, we have that $\Phi = j + i + 1 - a$;
- for $j \geq x - i - 1$, we have that $\Phi = x - a$.

This gives

$$\sum_{j=1}^{\sqrt{t}} \Phi(t, t - i - j - 1, x, a, b)\ \varepsilon_{i,j}$$

$$= \sum_{j=a-i}^{x-i-1} (j+i+1-a)\,\varepsilon_{i,j} + (x-a) \sum_{j=x-1}^{\sqrt{t}} \varepsilon_{i,j}.$$

In both cases, $\varepsilon_{i,j}$ is exponentially small for fixed i and $j \to +\infty$ and the sum from $x-1$ to $\sqrt{t}$ converges towards a finite limit; hence the result.