ISSN 1870-4069

On Generating Random Graphs Based on Fuzzy Proximity Relationships between Vertices

Mohamed Amine Omrani, Wady Naanaa

Faculty of Sciences of Tunis,
Computer Science Department,
LIMTIC Laboratory-ISI, Ariana, Tunisia
mohamedamine.omrani@fst.utm.tn, wady.naanaa@fsm.rnu.tn

Abstract. Uncertain graphs are getting more and more important. They allow to tackle fuzzy situations in numerous frameworks. This paper investigates the issue of generating random graphs based on uncertain proximity relationships between vertices and the goal is to construct the most likely graph. The Constraint Programming paradigm was used to provide a systematic way to release uncertain graphs while maximizing and minimizing the paths that separate certain vertex pairs. The proposed approach allowed to generate uncertain graphs at a reasonable time. This is confirmed by experimental results obtained from a series of tests on several instances. Our solution lays the basis for further real world applications in different fields, such as analytical chemistry, telecommunication networks, and civil engineering.

Keywords: random graphs, fuzzy proximity, constrain programming.

1 Introduction

In mathematical context, a graph is a non-linear data structure consisting of a set of vertices and edges that link vertices together. One of the most attractive features of graphs is their adequacy to model pairwise relationship between objects. Indeed, such structural information would be useful to represent and solve relevant computational problems by providing a fairly accurate description of the problem at hand. Thereby, a random graph is a graph that is generated by a random process. Historically, the first model of random graphs was defined by *Paul Erdõs* [3] to give a probabilistic construction of a graph with large girth and large chromatic number. Later, there has been a shift of emphasis toward generating random graphs with prescribed feature constraints such as vertex degree [16, 15], imposed or forbidden induced sub-graphs [18], proximity relationships between vertices [17]. Graph theory is a vast subject in which the goals are relying on various graph properties.

The issue of random graph generation is not recent, but is still important because its applications are renewable in real-life situations. The growing interest in this fascinating issue can be attributed to several factors. One factor is the realization that networks are everywhere. From social networks such as

Facebook, Twitter, Instagram or Linkedin, the World Wide Web and the Internet to the complex interactions between isomers in the molecular structures, we face the challenge of elucidating their structure. By and large natural networks grow in an unpredictable manner and this is often modeled by a random construction. Another factor is the realization by Computer Scientists that NP-hard problems are often easier to solve than their worst-case suggests and that an analysis of running times on random instances can be informative.

However, generating graphs is commonly a complicated task since it amounts to the assembly of a set of vertices in all possible ways to ensure exhaustivity while being consistent with a set of structural constraints such as the number of vertices, edges, and connections between them. Figure 1 illustrates with an example case the task of generating constrained random graphs under constraints. Let G be a random simple graph to generate defined by a given degree sequence $\Delta = \langle 2, 2, 2, 3, 3, 4, 4 \rangle$, a set of 7 vertices, a set of 10 edges and other hard constraint such as imposing a distance of length two separating vertices 1 and 4. The goal is to generate all graphs that satisfy the given input data set. This configuration gives 249 simple undirected graphs of which 19 graphs are connected graphs with no symmetry. Of these, only two graphs respect the proximity relationship constraint between vertices 1 and 4.

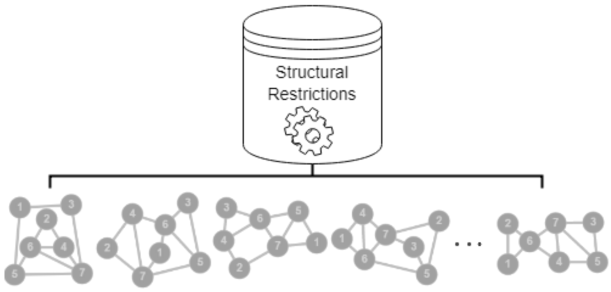


Fig. 1. Generating constrained graphs based on degree sequence $\Delta = \langle 2, 2, 2, 3, 3, 4, 4 \rangle$.

Often, the conditions under which the data are obtained are less favorable in reality. We have, therefore, to deal with data tainted by uncertainties due to a misinterpretation of real situation. Generally, a large set of graphs are consistent with a same degree sequence. In such a situation, determining the graph that models the true real-world situation is not always obvious because of the huge set of candidate graphs. Therefore, it appears that an intelligent assembly process is necessary to deal with the combinatorial nature of the problem. The second challenge comes from the use of uncertain data in order to narrow the set of candidate graphs. It is possible that after having listed the desired structural constraints, there is no way to satisfy them all simultaneously. In this case, the

problem is said to be over-constrained. A way to get around such a situation is to choose which constraints to ignore. On the other hand, when we get a huge set of graphs that simultaneously satisfy all the constraints, such graphs appear equally good, and another challenge consists in benefiting from the available uncertain information to select the more likely graph among the numerous candidate graphs.

Although several studies have been dedicated to random graphs in recent years, some focusing on graph properties [10], others on generation methods [18, 7]. They are often highly dedicated to specific situations and encounter difficulties in taking into account the uncertainty nature of some information. In addition, the assumed constraints are unlikely to occur in real-world situations. On the other hand, to our knowledge almost half of the structural problems contain uncertain information as input data. Therefore, an approach for dealing with uncertain graphs will be of great interest.

The current research was carried out to shed light on the problem of generating uncertain graphs, where the uncertainty nature comes from Uncertain Proximity Relationships (UPRs) between vertices. A UPR is a proximity relationship between two vertices for which the topological distance is not known but bounded (above or below) by a given constant. What is meant by topological distance is the number of edges that composes the shortest path which connects the concerned vertex pair. UPRs are of two types: UPRs to maximize (max-UPRs) and UPRs to minimize (min-UPRs). The min-UPRs type includes uncertain proximity favoring short topological distances separating vertices. Structural preferences here state that the shorter the overall min-UPRs distances is, the more likely the graph is to correspond to the true graph. The second type of uncertain proximity, that is max-UPR, is basically the first type reversed, favoring a long topological distance between each vertex pair involved in a max-UPR. As mentioned above, an UPR is specified by an upper or a lower bound on the topological distance separating a given vertex pair.

This paper proposes a weighted constraint-based approach providing a systematic way to generate uncertain graphs, while enforcing uncertain proximity relationship preferences of certain prescribed vertex pairs. More concretely, we explore here the use of the Constraint Programming (CP) paradigm to generate graphs on the basis of a given degree sequence, while maximizing the sum of all topological distances of max-UPRs preferences and minimizing those of min-UPRs.

Our interest in UPRs is motivated by their importance as basic data structures for many real-life applications. For example, in analytical chemistry, the molecular structure can be considered as a graph and the problem of generating molecular structure consists in finding all molecular graphs that are consistent with a dataset derived from different kinds of spectra. Often, this dataset imposes a set of uncertain inter-atoms proximity relationships for which the topological distance between interacting atoms cannot be accurately known but rather bounded by given constants [2]. This issue, besides being interesting, is useful in a variety of situations and is of great interest not only for chemists, but also to

biologists, computer scientists, economists, electrical engineers, mathematicians, neuroscientists and sociologist [23, 2]. From the theoretical point of view, this problem is NP-hard on general graphs. And to the best of our knowledge, research on this topic are scarce. A key part of this contribution is to rely on uncertain proximity relationships to guide the construction process to the most likely graphs.

The contribution of the paper is three folds. First, we propose a new formulation, precisely the one resorting to Constraint Programming (CP), a paradigm that has proved to be an adequate framework for dealing with various combinatorial optimization problems. So, generating uncertain graphs is encoded as a Weighted Constraint Satisfaction Problem (WCSP) in an easy and declarative way that enables an efficient integration of various pieces of structural restrictions. Second, we show how this encoding can be employed to get the best we can from min-UPR restrictions, as well as, from max-UPR restrictions during the graph generation process. Third, we have succeeded in defining a single objective function, developed as a soft constraint of the weighted constraint-based formulation, that combines the two contradictory preferences, max-UPR and min-UPR to finally get the most likely uncertain graphs.

The rest of the paper is organized as follows: Section 2 and 3, are respectively devoted to giving preliminary notions of graph theory and constraint programming. Section 4 illustrates our weighted constraint-based approach for generating uncertain graphs which optimizes the inter-vertices distances revealed by UPRs. In Section 5, we report a series of experimental results that show the validity and the efficiency of our approach on several uncertain graph generation instances.

2 Basic Graph Notions

To begin with, we shall expose the theoretical definitions and terminologies needed to cope with the graph theory framework. In what follows, $G{=}(V, E)$ will designate the graph comprising a non-empty finite set V of vertices related together with a collection, $E{\subseteq}\binom{V}{2}$ of edges, each of which is an unordered pair of vertices (*undirected*). If $\{i, j\} \in E$, then i and j are *adjacent*, and the edge $\{i, j\}$ is *incident* to both i and j. For conciseness, $\{i, j\}$ will be denoted by i-j. The *neighborhood* of vertex i is $N(i) = \{j | i\text{-}j \in E\}$. We set $n = |V|$ and $m = |E|$. The degree of a vertex $i \in V$, denoted by $deg(i)$, is the number of edges incident to i.

The integer sequence $\Delta = \langle\ deg(1), \ldots, deg(n)\ \rangle$ is known as the *degree sequence* of G. Giving a real-world example, in a computer network, a pair of computers is in E, if and only if, they are connected by some medium. If E contains multiple copies of an edge, that is, E is a multi-set, then G is a multi-graph.

A *chain* of G is a sequence of vertices $(1, \ldots, \ell)$ such that, i-$i{+}1 \in E, 1 \leqslant i < \ell$, where ℓ is the length of the chain. A *path* or in G is a chain $(1, \ldots, \ell)$ in which $i \neq j$, for all $1 \leqslant i, j < \ell$. A path in G of length ℓ is called an ℓ-path of G. Note that the number of vertices in a ℓ-path is equal to the number of edges.

G is said to be *connected* if, for every $i, j \in V$, with $i \neq j$, there is an path in G from i to j. The *topological distance* between vertices i and j, denoted $d(i, j)$, is the length of the shortest path separating i and j in G.

An *uncertain proximity relationship* of length ℓ between i, j of type $s \in \{0, 1\}$ will be denoted by $\text{upr}^s_{i,j,\ell}$. If $(s = 1)$ then the UPR is of type max-UPR, otherwise $(s = 0)$ the UPR is of type min-UPR. Therefore, $\text{upr}^s_{i,j,\ell}$ suggests that the distance separating i and j in G verifies $d(i, j) \leqslant \ell$, if $s = 0$ and $d(i, j) \geqslant \ell$, otherwise.

We can now pose the graph generation problem as follows: Given a set V of n vertices, a corresponding degree sequence Δ of n integers and a set of uncertain proximity relationships (UPRs), is there a graph G comprised of the vertices in V that realizes Δ and satisfies the UPRs? In other words, can we construct an edge set E such that Δ is the degree sequence of $G = (V, E)$ and the set of UPRs is satisfied?

3 Constraint Programming Background

Constraint Programming (CP) is a convenient paradigm to model and solve highly combinatorial problems, that draws on a wide range of techniques from Artificial Intelligence, Operations Research, Algorithms, and elsewhere. A constraint is a restriction, so, the basic idea in Constraint Programming (CP) is that the user states the constraints and the decision variables, and a constraint solver is used to solve them [5]. The best CP definition that can be given here is the one described by *Eugene Freuder*: *Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem; the computer solves it.* The following equation describes the principle of the CP paradigm, consisting of two mainly interconnected components, *Modelling* and *Solving*:

$$CP = Modelling + Solving. \tag{1}$$

Modelling is the mental process of mapping an informal description of a problem into a formal description in a particular formal system [9]. After this mapping, the role of constraint solvers begins. They search the solution space systematically, as with backtracking or branch and bound algorithms, or use forms of local search which may be incomplete.

There are many real-life problems that require decision-making in the presence of constraints. Such problems can be modeled as Constraint Satisfaction Problems (CSPs). More formally, as described by Tsang [22], a CSP is a problem that can be defined by a triple (X, D, C), where X is a finite set of variables, each of which is associated with a finite domain D, and a set of constraints C that restricts the values that variables can simultaneously take.

A general-purpose constraint solver is used to solve the CSP by finding an assignment of values to all the decision variables which satisfy the constraints, while reducing the CSP problem to another one that is equivalent but more simpler by means of inference. There is a wide choice of solvers available in the

literature, which include Choco [14], Gecode [6], ECLiPSe [1], ILOG CP [11], Minion [8]. The role of constraint solvers is to assign a value to each variable satisfying all the constraints. Constraint solvers explore the solution space either systematically, as with backtracking or branch and bound algorithms, or use forms of local search [12]. In systematic search, the constraints are not merely treated as tests, but play an active role by helping to discover the inconsistencies early on, via the so-called constraint propagation. This latter process allows constraint solver to eliminate parts of the search space and then makes the search shorter.

The so-called ILOG CP solver, developed by IBM, is the one we use in this work. It represents a main component of the integrated development environment (IDE) CPLEX Optimization Studio. CPLEX Optimization Studio consists of the OPL modelling and scripting language for developing optimization models, and the IDE for running and testing optimization models. It also includes the optimization engines CPLEX Optimizer and CP Optimizer for solving Mathematical Programming (MP) and Constraint Programming (CP) models. In particular, the modelling language OPL allows to state Constraint Programming (CP), Logic Programming (LP) and Mixed Integer Programming (MIP), as well as combinations of all of them. A strong side of OPL is that it allows a mathematical encoding of the problem that is separate from the data, while offers several basic branching strategies such as depth-first search which is the default search strategy, and a number of other strategies including a best-first strategy.

3.1 Valued CSP formalism (VCSP)

Many real-life combinatorial problems can be naturally modeled and often efficiently solved using constraint programming. However, in some real situations, the classical CSP framework does not help. Indeed, it may occur that after having listed the desired constraints against the decision variables, there is no way to satisfy them all simultaneously. In this case, the instance is said to be over-constrained. In contrast, in other situations, all the constraints can be easily satisfied, which results in several solutions. Such solutions appear equally good, and there is no way to select among them. These scenarios often occur when constraints are used to formalize desired properties rather than requirements that cannot be violated. Such desired properties are not faithfully represented by classical constraints but should rather be considered as preferences whose violation should be avoided as far as possible. To cope with similar situations, classical constraints have been generalized to soft constraints, providing one way to allow constraint relaxation. Therefore, a general formalism, called Valued Constraint Satisfaction Problem (VCSP), has extended the CSPs to handle soft constraints.

The Valued Constraint Satisfaction Problem (VCSP) formalism can be used to solve optimization problems as well as over-constrained problems as they allow constraint relaxation. A VCSP is simply obtained by annotating the constraint of classical CSP with valuations denoting the costs of violations. More precisely, VCSP extends the CSP framework (X, D, C) by associating weights (or simply

costs) to tuples (combinations of values that can simultaneously be taken by variables). Generally, costs are specified by means of valuation structure defined as a triple $S = (E, \oplus, \preceq)$, where E is the set of cost values totally ordered by $\preceq$. The maximum and minimum costs are denoted by $\top$ and $\bot$, respectively. $\oplus$ is a commutative, associative and monotonic operation on E used to aggregate costs. The commutativity and associativity of $\oplus$ ensure that the evaluation of an instantiation does not depend of the order in which the evaluations are done. Monotony ensures that valuations cannot decrease when constraint violations become more important.

More formally, a VCSP is defined by a quadruplet (X, D, C, S) as follows [13]:

- $X = \{x_1, \ldots, x_n\}$ is a finite set of variables;
- $D = \{D_1, \ldots, D_n\}$ is a finite set of value domains so that D_k is the domain of x_k;
- $C = \{C_1, \ldots, C_m\}$ is a set of valued constraints. A constraint is a pair (X_k, θ_k), where $X_k = \langle x_{k_1}, \ldots, x_{k_r} \rangle \subseteq X$ is the constraint scope and $\theta_k : \prod_{x \in X_k} D_x \rightarrow E$ is a cost function;
- $S = (E, \oplus, \preceq)$ is a valuation structure.

A variable $x_i \in X$ must be assigned a value from its domain D_i. If a valued constraint is defined by a cost function whose domain is limited to $\{\top, \bot\}$ then it is a hard constraint, otherwise it is a soft constraint. The arity of a valued constraint is the size of its scope. The arity of a problem is the maximum arity over all its constraints. The valuation of an assignment t to a subset of variables $V \subseteq X$ is given by:

$$\Theta_P(t) = \bigoplus_{(X_t, \theta) \in C, X_k \subseteq V} \theta(t{\downarrow}X_k), \tag{2}$$

where $\downarrow$ denotes the projection of t on the variables of X_k. Therefore, an optimal global solution for a VCSP on n variables is an n-tuplet t, such that $\Theta_P(t)$ is minimal over all possible tuples. A VCSP has many variations which mainly differ mainly by the valuation structure. The choice of the most appropriate valuation structure depends on the characteristics of the problem to be formulated in terms of VCSPs. Accordingly, a classical CSP can be seen as a VCSP variation with a valuation structure $S = (E, \oplus, \preceq)$, where $E = \{0, +\infty\}$, with standard integer ordering $\preceq$ and $\oplus$ being the classical sum. The maximum and minimum costs are respectively 0 and $+\infty$. In addition to classical CSPs, other VCSPs variations have been proposed in the literature. Among them, Fuzzy CSPs (FCSP) [20], Probabilistic CSPs (Prob-CSP) [4], Possibilist CSPs (Pos-CSP) [21] and Weighted CSPs (WCSP) which is the VCSP variant chosen here to solve the problem of uncertain graph generation.

3.2 The Weighted CSP variant (WCSP)

In Weighted Constraint Satisfaction Problems (WCSPs), tuples come with an associated cost. This allows modeling optimization problems where the goal is

to minimize the total cost of the proposed solution. More formally, a Weighted CSP, is a Valued CSP where the valuation structure $S = (E, \oplus, \preceq)$ such that E contains a set of integers $\{0, \ldots, k\}$ with standard integer ordering $\leqslant$. The aggregation operation $\oplus$ is defined by $a \oplus b = min(k, a+b)$. Moreover, for any integer pair a and b such that $b \leqslant a$, substraction $\ominus$ is defined as follows:

$$a \ominus b \quad \begin{cases} a \text{ - } b, & \text{if } (a \neq k), \\ k, & \text{otherwise.} \end{cases} \tag{3}$$

Without loss of generality, we assume that there exits a zero-ary valued constraint $C_\varnothing = (\varnothing, \theta_\varnothing)$ dedicated to maintaining a lower bound for the minimum cost of the problem.

4 Uncertain Graph Generation as WCSP

As stated in the introduction, the goal of this work is to generate the most likely graph by assembling a set of vertices in all possible ways to ensure exhaustivity. The assembled graphs must satisfy hard structural constraints coming from the degree sequence, Δ, while optimizing a set of prescribed soft restrictions derived from uncertain proximity relationships data (UPRs). In what follows, we encode the likely graph to be generated as a simple adjacency matrix, and then we attempt to post hard and soft constraints ensuring Δ and UPRs.

Representing a graph with a given degree sequence is straightforward. We opted for the adjacency matrix and we tried to post constraints in order to enforce the available uncertain proximity relationships between certain vertex pairs. To reach this goal based on uncertain connectivity information, we used the Weighted CSP framework. A judicious choice that will allow us to express soft restrictions. Before we proceed, we notice that the n vertices of the degree sequence Δ will be indexed by the integers from 1 to n.

4.1 The Basic Model

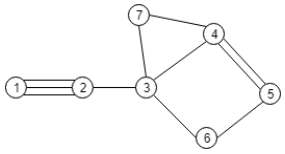
As stated above, a Weighted CSP is defined by a quadruplet (X, D, C, S). For the purpose of encoding constrained uncertain graphs, the variable set, X, will be composed of two subsets, which are the *Adjacency variables* and the *proximity variables*, let us refer to them as A and Λ; we therefore, have $X = A \cup \Lambda$.

The first variable subset, that is A, contains the variables that handle the adjacency of vertices in the graph. Hence, as shown in Figure 2, for a undirected multi-graph G with n vertices, we create $n \times n$ variables denoted by $a_{i,j}$, a variable for each ordered pair of vertices (i, j), which will be referred to as adjacency variables. Setting $a_{i,j}$ to a non-zero value means that there is one or more edges between vertex pair i and j; whereas, setting $a_{i,j}$ to 0 means that there is no edge connecting i and j. The value domain of each variable $a_{i,j}$ depends of the degrees, $deg(i)$ and $deg(j)$, relative to i and j.

The second subset of X, that is Λ, contains the decision variables that will encode UPRs. Recall that a UPR is a set of uncertain proximity relationship

$$\begin{pmatrix} 0&3&0&0&0&0&0\\ 3&0&1&0&0&0&0\\ 0&1&0&1&0&1&1\\ 0&0&1&0&2&0&1\\ 0&0&0&2&0&1&0\\ 0&0&1&0&1&0&0\\ 0&0&1&1&0&0&0 \end{pmatrix}$$

A



G

Fig. 2. An adjacency matrix A (left) giving the values of the adjacency variables corresponding to the degree sequence $\Delta = \langle 3, 4, 4, 4, 3, 2, 2\rangle$. The resulting target graph is G (right).

on vertex pairs for which the topological distance is not known but bounded by a given constant. More precisely, an element of UPR can be a max-UPR or a min-UPR. For any element of UPR (a min-UPR or a max-UPR), involving vertices i and j with bounding constant ℓ, we introduce a set $\Lambda_{i,j}$ of ℓ variables. The domain of each of these variables is set to the list of vertices that may correspond to the intermediate vertices on the shortest chain connecting vertex i to vertex j. Therefore, we have $\Lambda_{i,j} = \{\lambda_{i,j,1}, \ldots, \lambda_{i,j,\ell}\}$. The domain of each variable $\lambda_{i,j,k}$ could contain any vertex except i because the target graphs are loopless.

In what follows, we detail the constraint set C, of the Weighted CSP quadruplet, which defines the constraints related to the simultaneous processing of all structural restrictions, (*hard*) and (*soft*), in the problem.

4.2 Hard Constraints

Since the target graph comprises n vertices, is undirected and loopless, then we post the following hard constraints:

1. Since our uncertain graphs are undirected, we post $n(n\text{-}1)/2$ binary constraint. For each pair $i < j$, stating that $a_{i,j}{=}a_{j,i}$.
2. To forbid self-loops, we add the constraint $a_{i,i}{=}\ 0$, for each $1{\leqslant}i{\leqslant}n$.
3. Given the degree $deg(i)$ of vertex i, we can post a constraint $\sum_{j=1}^{n} a_{i,j}{=}deg(i)$, for each vertex i to ensure that the vertices have the correct degree.
4. If a pair of vertices can be related by more than one edge, this will result in a multigraph and then we may have $a_{\mathrm{i,j}} \geqslant 2$, for some vertex pairs (i, j).
5. For each UPR of length ℓ and type s, denoted $\mathrm{upr}^{s}_{i,j,\ell}$ involving the variable set $\Lambda_{i,j} = \{\lambda_{i,j,1}, \ldots, \lambda_{i,j,\ell}\}$, there is a hard restriction portion on the distance separating i and j in the target graph, depending on parameter s. Hence, $d(i, j) \leqslant \ell$ if UPR is a min-UPR ($s = 0$) and $d(i, j) \geqslant \ell$ otherwise ($s = 1$). This constraint can be expressed by the following two logical formulas:

$$(s = 0) \implies \mathrm{card}(\lambda_{i,j,k} \neq j) \leqslant \ell, \qquad k : 1, \ldots, \ell, \tag{4}$$

$$(s = 1) \implies \text{card}(\lambda_{i,j,k} \neq j) \geqslant \ell, \qquad k : 1, \ldots, \ell. \tag{5}$$

4.3 Soft Constraints: UPRs

Recall that a Uncertain Proximity Relationship (UPR) can be min-UPR or max-UPR. Expressing a UPR is a delicate task on account of the desired soft preferences (maximizing and/or minimizing topological distances). In addition to the two hard constraints expressed by equations (4) and (5), the encoding of this task requires adding many variables and constraints according to the type of prescribed proximity relationship $s \in \{0, 1\}$. Each UPR, either be min-UPR or a max-UPR, that constraint a pair of vertices (i, j), favors respectively short or long topological distance that separates i from j in the target graph. The goal here is to minimize or maximize that distance as much as possible with respect to parameters ℓ and s.

In the Weighted CSP formulation proposed below, we provide a solution to handle the uncertain nature of these preferences.

According to the type of UPR, min-UPR or a max-UPR, the idea consists in placing j, respectively, as close, or as far, as possible to vertex i through an objective based on cost functions. The objective value will vary depending on the number of intermediate vertices between i and j. This can be achieved as follows: Every $\text{upr}^s_{i,j,\ell}$ is defined by a subset of ℓ variables $\Lambda_{i,j}$. The domain of each $\lambda_{i,j,k}$ is set to $[1..n]\backslash i$. In addition, we post, for each k: 1, ...,ℓ-1, a ℓ-unary constraint $(\lambda_{i,j,k}, \theta^s_{i,j})$ for which the cost functions that define the weighted constraint are set as follows:

$$\theta^s_{i,j}(v) \begin{cases} s, & \text{if } \ v = j, \\ \\ 1\text{-}s, & \text{otherwise.} \end{cases} \tag{6}$$

To connect the set of variables $\Lambda_{i,j}$ to the adjacency variables, we post for each variable $\lambda_{i,j,k}$, $1 \leqslant k \leqslant \ell$-1, the following ternary constraint:

$$(\ \lambda_{i,j,k}{=}u \quad \wedge \quad \lambda_{i,j,\text{k}+1}{=}v) \implies a_{u,v}{\neq}0. \tag{7}$$

Also, the beginning of the separating chain entails the following constraint:

$$\lambda_{i,j,1}{=}u \quad \Rightarrow \quad a_{i,u}{\neq}0. \tag{8}$$

We can notice that the sum of costs of every variables subset $\Lambda_{i,j} \subset \Lambda$ depends on the topological distance separating i and j at hand. In fact, the sum of the costs over all the $\lambda_{i,j,k}$ is equal to the distance between i and j minus one. This coding is extended to obtain the most probably uncertain graph, that is, the one that minimizes the overall costs Φ, imposed by UPR set. This global objective can be expressed as follows:

$$\Phi(I) = \sum_{(\lambda_{i,j},\theta^s_{i,j})} \sum_{k=1}^{\ell} \theta^s_{i,j}(I(\lambda_{i,j,k})). \tag{9}$$

Note that, if the pair of vertices i and j are directly related in the uncertain graph, the global objective value is equal to zero.

Example 1. Let's take the problem of constructing G, a uncertain graph composed of eight vertices $\{1, \ldots, 8\}$ and defined by $\Delta = \langle 2, 2, 2, 2, 2, 2, 2, 2 \rangle$ as degree sequence. Let's focus on the following four min-UPR restrictions which state the following uncertain proximity relationship to minimize: $\{\text{upr}^0_{8,3,3}, \text{upr}^0_{7,4,4}, \text{upr}^0_{1,2,3}, \text{upr}^0_{1,6,4}\}$.

This problem can be outlined by the basic model as described in Subsection 4.1 with $\Lambda = \{\ \Lambda_{8,3},\ \Lambda_{7,4},\ \Lambda_{1,2},\ \Lambda_{1,6}\}$ as proximity relationship variables to minimize. The applicability of the proposed WCSP encoding on this graph instance, using the CP Optimizer library developed by IBM (https://www.ibm.com) which provides a constraint programming engine, performs well, giving good results. Figure 3 shows the obtained uncertain graph with its corresponding statistics. Three seconds as CPU time and 0.79 MB of ROM memory are required to generate G having a global objective $\Phi = 4$, knowing that 1025 graph candidates are obtained before the integration of min-UPRs restrictions.

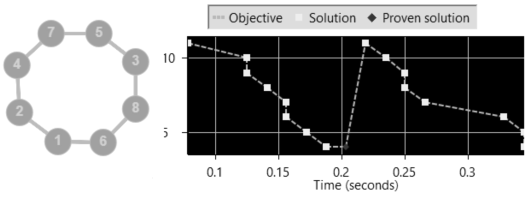


Fig. 3. min-UPR results (first Attempt).

Example 2. Let's again take the problem of constructing the most likely graph, with the same input data given in Example 1, which are the set of 8 vertices $\{1, \ldots, 8\}$ and the degree sequence $\Delta = \langle 2, 2, 2, 2, 2, 2, 2, 2 \rangle$. This time, our soft restrictions concern max-UPR and composed by four prescribed max-UPR, which are $\text{upr}^1_{8,4,5},\ \text{upr}^1_{7,3,5}$, $\text{upr}^1_{6,2,5},\ \text{upr}^1_{6,4,5}$.

As seen in Figure 4, by encoding the max-UPRs constraints described below based on set $\Lambda = \{\Lambda_{8,4},\ \Lambda_{7,3},\ \Lambda_{6,2},\ \Lambda_{6,4}\}$, we have gotten the most likely graph G among 1025 candidates. The experiment statistical results indicate that it takes 0.28 seconds of CPU running and 1.05 MB of Memory to generate G. The global objective in output,Φ, is equal to 12.

Although widely accepted, one limitation of our previous WCSP encoding to maximize UPRs is found. For every max-$\text{upr}^s_{i,j,\ell}$ with $s = 0$, it must be imposed that the path, taken into account by each subset of variables $\Lambda_{i,j}$, represents the shortest path connecting i to j in the target uncetain graph G. Therefore, for

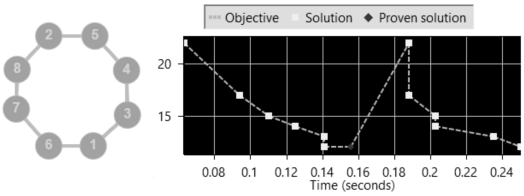


Fig. 4. max-UPR results (second Attempt).

each $\lambda_{i,j,k}$ the following hard constraint must be added to the set of constraints:

$$(\lambda_{i,j,k} = u \wedge u \neq j) \Rightarrow (\bigvee_{k'=1}^{k\text{-}1} a_{\Lambda_{i,j,k'},\Lambda_{i,j,k'+1}} = 0) \vee (a_{i,\lambda_{i,j,1}} = 0) \vee (a_{u,j} = 0). \tag{10}$$

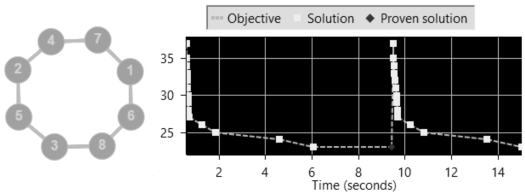


Fig. 5. max-UPR results (third attempt).

The results illustrated in Figure 5 confirm that it is necessary to add equation 10 below. Experiment statistics indicate that our solution for max-UPR takes 18.3 seconds of CPU time and 2.76 MB of ROM to effectively deal with the problem cited in the example 2 resulting graph G with a global objective $\Phi = 17$. To this end, we give a last example which takes into account uncertain proximity relationships information in a general way to ensure the generation of the most likely uncertain graph. However, valid results are achieved when applying our constraint-based encoding.

Example 3. Let's one more time take the same problem proposed in Example 1 and refined in Example 2. We focus here on both min-UPR restrictions of Example 1 and simultaneously on max-UPR restrictions defined as input data in Example 2. Hence, we have $\{\text{upr}^0_{8,3,3}, \text{upr}^0_{7,4,4}, \text{upr}^0_{1,2,3}, \text{upr}^0_{1,6,4},\}$ to minimize and $\{\text{upr}^1_{8,4,5}, \text{upr}^1_{7,3,5}, \text{upr}^1_{6,2,5}, \text{upr}^1_{6,4,5},\}$ to maximize.

This problem can be outlined by the basic model as described in Subsection 4.1 that is made of two subsets which are the *Adjacency variables* (A) and the

proximity variables (Λ). Thus, $\Lambda = \{\Lambda_{8,3},\ \Lambda_{7,4},\ \Lambda_{1,2},\ \Lambda_{1,6},\ \Lambda_{8,4},\ \Lambda_{7,3},\ \Lambda_{6,2},\ \Lambda_{6,4}\}$.

The applicability of the proposed WCSP encoding on this graph generation instance, using the CP Optimizer library developed by IBM (https://www.ibm.com) which provides a constraint programming engine, performs well, giving good results.

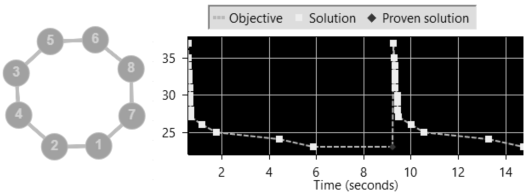


Fig. 6. UPR final results (fourth Attempt).

Figure 6 shows the obtained uncertain graph with its corresponding statistics. Three seconds as CPU time and 2.8 MB of ROM memory are required to generate G with a global objective $\Phi = 36$. Know that 1025 graph candidates are obtained before the integration of UPRs restrictions. The results of the experiment found clear support for the soft constraint-based uncertain graph generation.

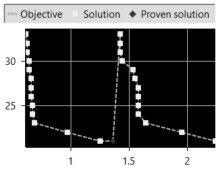
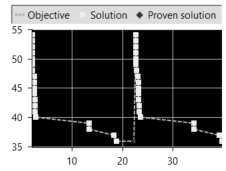
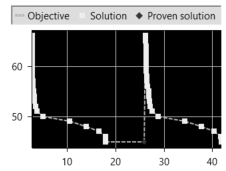
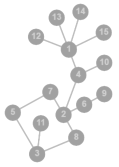
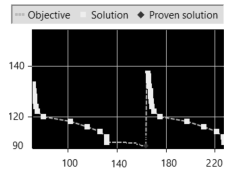
5 Experimental Results

To evaluate the effectiveness of our approach, we have tested it on several simple graph instances. All tests have been done on a computer (Intel Core i5-8250U CPU 1.60GHz 1.80GHz, 8 GB of RAM) running windows 10 (64 bit). During graph generation, we used the ILOG CP Optimizer solver developed by IBM (https://www.ibm.com), which provides a constraint programming engine.

Table 1 summarizes the experimental results obtained. For each instance of degree sequence Δ, presented in a simplified version such as $\langle 4_2, 3_3, 2_2, 1_1\rangle$ is equivalent to $\langle 4, 4, 3, 3, 3, 2, 2, 1\rangle$, we first reported the corresponding number of solutions ($\sharp$). We also included the uncertain proximity input data ($\text{upr}^s_{i,j,\ell}$) with their results in terms of memory usage (ROM) in MB and CPU time in seconds (CPU).

In addition, the column entitled (Cost Φ), is related to the cost function (global objective) of the Weighted-CSP formulation proposed in this paper. The last two remaining columns show the results, obtained by the program after considering UPRs, in terms of the most likely graph solution and respectively its experimental statistic. What is worth noting in our program is that, when we consider the uncertain proximity relationships in graph instances, we got the most likely graph admitting in its structure the most plausible topological distances over all pairs of vertices in interaction.

Table 1. Experimental results obtained from our Weighted CSP formulation.

Degree Sequence	$\mathfrak{u}$	$\mathbf{upr}^{s}_{i,j,\ell}$	**ROM**	**Cost** Φ	**CPU**	**Graph Solution**	**Statistics**
$\langle 4_2, 3_2, 2_2, 1_2\rangle$	2480	$\Lambda^0_{8,1,3}$, $\Lambda^0_{6,1,3}$ $\Lambda^0_{6,5,4}$, $\Lambda^0_{2,7,4}$ $\Lambda^1_{7,8,5}$, $\Lambda^1_{6,8,5}$ $\Lambda^1_{6,4,5}$, $\Lambda^1_{3,5,5}$	27.68	21	2.30		
$\langle 5_1, 4_1, 3_3, 2_2, 1_2\rangle$	11960	$\Lambda^0_{1,9,3}$, $\Lambda^0_{2,3,3}$ $\Lambda^0_{8,1,3}$, $\Lambda^0_{6,1,3}$ $\Lambda^0_{6,5,4}$, $\Lambda^0_{2,7,4}$ $\Lambda^1_{1,8,5}$, $\Lambda^1_{2,4,5}$ $\Lambda^1_{7,8,5}$, $\Lambda^1_{6,8,5}$ $\Lambda^1_{6,4,5}$, $\Lambda^1_{3,5,5}$	71.37	36	42.81		
$\langle 5_1, 4_1, 3_2, 2_3, 1_3\rangle$	21080	$\Lambda^0_{1,10,3}$, $\Lambda^0_{2,4,3}$ $\Lambda^0_{8,2,3}$, $\Lambda^0_{6,2,3}$ $\Lambda^0_{6,5,4}$, $\Lambda^0_{2,7,4}$ $\Lambda^1_{1,7,5}$, $\Lambda^1_{2,3,5}$ $\Lambda^1_{7,8,5}$, $\Lambda^1_{6,4,5}$ $\Lambda^1_{6,3,5}$, $\Lambda^1_{3,4,5}$	102.12	45	50.09		
$\langle 5_1, 4_1, 3_2, 2_4, 1_7\rangle$	62917	$\Lambda^0_{1,10,3}$, $\Lambda^0_{2,4,3}$ $\Lambda^0_{8,2,3}$, $\Lambda^0_{6,2,3}$ $\Lambda^0_{6,5,4}$, $\Lambda^0_{2,7,4}$ $\Lambda^0_{10,12,3}$, $\Lambda^0_{10,13,3}$ $\Lambda^0_{10,15,4}$, $\Lambda^0_{11,2,4}$ $\Lambda^1_{1,7,5}$, $\Lambda^1_{2,3,5}$ $\Lambda^1_{7,8,5}$, $\Lambda^1_{6,4,5}$ $\Lambda^1_{6,3,5}$, $\Lambda^1_{3,4,5}$	252.12	90	229.12		

The results obtained from a series of tests on several instances can only validate the proposed approach, allowing to generate uncertain graphs at a reasonable time. Recall that our purpose is not to generate all graphs as quickly as possible, but to encode a soft constraint that can be used with an external search procedure on problems with side constraints, to consider proximity relationships preferences.

In particular, we have not considered connectivity or symmetry, and there are many symmetries in these problems. We view symmetry as a separate feature that can be maintained independently. Paper [19] describes some explorations of constraint programming in graph generation, concentrating on forcing the graphs to be connected.

6 Conclusions

In this paper, we used a variant of the Valued CSP approach to address the graph generation problem under uncertainty. In particular, we have presented a Weighted CSP formulation that allowed the integration of hard and soft numerous structural constraints in a single framework. This study has shown the advantage of adding fuzzy information or preferences, found in uncertain proximity relationships between pairs of vertices, to select the most probably graphs. From a practical point of view, we used the programming library named ILOG CP. The experimental study showed that our constraint-based solution is successful in coping with the hard problem of uncertain graph generation. As a future work, we plan to take account of more complex uncertain data like fuzzy degrees or uncertain subgraphs. Such data are frequently encountered in real-world situations and may be very helpful in further limiting the set of candidate solutions. To this end, once enhanced with the capability of handling more uncertain data, our solution could be more useful in several real-world applications.

References

1. Apt, K.: Constraint logic programming using ECLiPSe. Cambridge University Press, Cambridge (2007)
2. Elyashberg, M.: Computer-based structure elucidation from spectral data : the art of solving problems. Springer, Heidelberg (2015)
3. Erdös, P.: Graph theory and probability. Canadian Journal of Mathematics 11, 34–38 (1959)
4. Fargier, H., Lang, J.: Uncertainty in constraint satisfaction problems: A probabilistic approach. pp. 97–104 (2005)
5. Freuder, E.C., O'Sullivan, B.: Grand challenges for constraint programming. Constraints 19(2), 150–162 (Apr 2014)
6. G Tack, M.L., Schulte, C.: Gecode: generic constraint development environment. http://www.gecode.org/ (2009)
7. Gent, I.P., Macintyre, E., Prosser, P., Smith, B.M., Walsh, T.: Random constraint satisfaction: Flaws and structure. Constraints 6(4), 345–372 (Oct 2001)
8. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast, scalable, constraint solver. In: ECAI 2006: 17th European Conference on Artificial Intelligence. pp. 98–102. Frontiers in Artificial Intelligence and Applications, IOS Press (2006), eCAI 2006: 17th European Conference on Artificial Intelligence, Riva del Garda
9. Hnich, B., Hnich, B.: Function variables for constraint programming. Tech. rep. (2003)
10. van der Hofstad, R.: Random Graphs and Complex Networks. Cambridge University Press (2016)
11. IBM: Ibm ilog solver v6.7. https://www.ibm.com/analytics/cplex-cp-optimizer, accessed: 2019-05-02
12. Jussien, N., Debruyne, R., Boizumault, P.: Maintaining arc-consistency within dynamic backtracking. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming. pp. 249–261. CP '02, Springer-Verlag (2000)

13. Kolmogorov, V., Krokhin, A.A., Rolinek, M.: The complexity of general-valued csps. CoRR abs/1502.07327 (2015)
14. Laburthe, F.: Choco: Implementing a cp kernel. CP00 Post Conference Workshop on Techniques for Implementing Constraint Programming Systems (TRICS) (01 2000)
15. Mihail, M., Vishnoi, N.K.: On generating graphs with prescribed vertex degrees for complex network modeling. In: 3rd Workshop on Approximation and Randomization Algorithms in Communication NEtworks (2003)
16. Milo, R., Kashtan, N., Itzkovitz, S., Newman, M.E., Alon, U.: On the uniform generation of random graphs with prescribed degree sequences. arXiv preprint cond-mat/0312028 (2003)
17. Omrani, M.A., Naanaa, W.: A constrained molecular graph generation with imposed and forbidden fragments. In: Proceedings of the 9th Hellenic Conference on Artificial Intelligence - SETN16. ACM Press (2016)
18. Omrani, M.A., Naanaa, W.: Graph generation with imposed and forbidden patterns. In: IADIS International Conference Applied Computing - AC'17. pp. 179–186. IADIS Digital library (2017)
19. Prosser, P., Unsworth, C.: A connectivity constraint using bridges. In: Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy. pp. 707–708. IOS Press, Amsterdam, The Netherlands, The Netherlands (2006), http://dl.acm.org/citation.cfm?id=1567016.1567174
20. Ruttkay, Z.: Fuzzy constraint satisfaction. In: Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference. IEEE
21. Schiex, T.: Possibilistic constraint satisfaction problems or "how to handle soft constraints?". CoRR abs/1303.5427 (2013)
22. Tsang, E.: Foundations of Constraint Satisfaction. Books On Demand (2014)
23. Yu, K., Wang, X., Li, Q., Zhang, X., Li, X., Li, S.: Individual morphological brain network construction based on multivariate euclidean distances between brain regions. Frontiers in Human Neuroscience 12 (may 2018), https://doi.org/10.3389/fnhum.2018.00204