WP. 5

ENGLISH ONLY

UNITED NATIONS STATISTICAL COMMISSION and
ECONOMIC COMMISSION FOR EUROPE
CONFERENCE OF EUROPEAN STATISTICIANS

Work Session on Statistical Data Editing
(Madrid, Spain, 20-22 October 2003)

Topic (ii): Developments related to new methods and techniques

NEW ALGORITHMS FOR THE EDITING AND IMPUTATION PROBLEM

Invited Paper

Submitted by DEIOC, Universidad de La Laguna, Tenerife, Spain[1]

Abstract

Data collected by statistical agencies may contain inconsistences caused by mistakes made during the acquiring, transcription and coding process. The optimization problem arising when an statistical agency must modify a microdata to guarantee that the records satisfy a set of rules (known as edits) is approached in this article. Indeed, before using a collection of data records to infer statistical properties of some groups of responders, the agencies must check and possibly correct the consistence of the collected data. To this end, the edits must be tested on each record and whenever a record does not satisfied all the edits the agency must determine the fields in the record to be modified, as well as impute the new values. Among all the possible solutions, the statistical agency is interested in finding one concerning with the minimum number of fields to be modified, thus leading a combinatorial optimization

[1]Prepared by Jorge Riera-Ledesma (jriera@ull.es) and by Juan-José Salazar-González (jjsalaza@ull.es), and supported by the Spanish project TIC2002-00895 ("Ministerio de Ciencia y Tecnología").

problem known as *Editing-and-Imputation Problem*. An Integer Linear Programming model for the particular case in which the edits are linear constraints is proposed, and solved through cutting-plane approaches. The new proposals are compared to other previously published in the literature and tested on benchmark instances. The overall performances of the new algorithms succeeded in solving difficult instances with up to 100 variables and 50 edits to optimality in about one minute of a personal computer.

Keywords: Editing and Imputation, Integer Linear Programming.

1 Introduction

Data collected by statistical agencies may contain errors because questions have been misunderstood by the respondents or mistakes have been made during the transcription and coding process. Therefore, since it may be impossible to get back to the original source, detection and correction of such errors becomes a necessary task before start data processing, in order to improve the integrity and quality of decisions made on the basis of this information. The task of identifying records containing errors and the specific fields causing these errors is known as the *data editing* problem, and the task of changing these fields in order to correct the errors is known as *imputation*. Both are typically carried out by experts in statistical agencies, thus consuming a large amount of their resources. Therefore, producing automatic techniques which help these experts with such a complex work becomes an interesting goal.

More precisely, the microdata collected by the agencies consists of a set of records, each one containing the answers of a respondent to a set of queries. Every value in a record is known as *field*, and it contains either a discrete or a continuous value. Discrete values correspond to *categorical queries* (e.g., marital status), while continuous numbers correspond to *quantitative queries* (e.g., weight). A microdata may contain both data types.

To introduce the problem let us suppose to have n queries, indexed by a finite set $I := \{1, \ldots, n\}$. Each record a is a $n \times 1$ vector, say $a = [a_i : i \in I]$, whose component a_i is an entry in field i provided by a respondent. The correctness of a record is given by a set E of consistence rules known as *edits*. This set is associated to a set $\mathcal{P}_E$ of all potential *valid records*. Given a set E of m edits, indexed by $J := \{1, \ldots, m\}$, a record a is said to be *valid* (or *consistent*) if $a \in \mathcal{P}_E$. For example, when the fields are continuous numbers and the edits in E are m linear equations then $\mathcal{P}_E$ is a polyhedron. If a record a of a microdata is not valid according to the edit set, then the aim of the *Editing and Imputation Problem* (EIP) is to modify the fewest possible items of data in order to obtain a valid record by keeping the non-modified field values [4]. The aim of this objective function is to preserve as much of the original information as possible. However, the goal of the EIP is typically extended by considering the minimization of the weighted number of fields to be changed to satisfy the set of edits. In other words, a weight w_i ($i \in I$) represents the confidence in the value of field i, and it can be thought as a surrogate, under certain assumptions, for the objective of maximizing the product of the probabilities that a changed field is in error (see Liepins [9]).

This definition of the EIP is also named the *Minimum Weighted Fields to Impute problem* and it can be formulated using a 0-1 variable x_i and a variable y_i for all $i \in I$. The variable x_i assumes value 1 if and only if the field i has to be modified, and the variable y_i represents the value of field i in the new record. Then a formulation is the following:

$$\min \sum_{i \in I} w_i x_i$$

subject to

$$y \in \mathcal{P}_E$$

$$x_i := \begin{cases} 1 & \text{if } a_i \neq y_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } i \in I. \tag{1}$$

Lieping, Garfinkel and Kunnathur show by a reduction to the satisfiability problem that EIP is a NP-hard problem in the strong sense. Many articles in literature concern algorithms to automatically solve this optimization problem. See, e.g., Kovar and Winkler [8] for a review on the literature.

Since the above model is not suitable to be solvable a mathematical programming approach, several articles in literature [4, 5, 6, 10, 11] have considered the following 0-1 integer Linear formulation:

$$\min \sum_{i \in I} w_i x_i \tag{2}$$

subject to

$$\sum_{i \in I_k} x_i \geq 1 \qquad \text{for all } k \in K \tag{3}$$

$$x_i \in \{0, 1\} \qquad \text{for all } i \in I. \tag{4}$$

In this formulation K is a subset of a collection $E^* := E \cup E'$, where E' is an additional subset of edits that will be described latter. The edits in E are called *explicit edits* and the edits in E' are called *implicit edits*. The set $I_k \subseteq I$ associated to edit $k \in K$ represents the candidate set of fields to be modified in order to satisfy this edit. Hence the constraint (3) imposes that at least one of those fields must be modified if the current record a does not satisfy edit k.

Clearly K must contain the explicit edits failed by the record a and, in fact, each optimal solution for EIP satisfies constraints (3) when K is only this subset of constraints because they state that at least one of its fields must be changed for every failed edit. However, Fellegi and Holt [4] observed that not all solutions of model (2)–(4) are feasible solutions for EIP if K only

contains the explicit edits failed by a (see [4, 11] for examples). Another important contribution of Fellegi and Holt [4] is to give a full description of the implicit edit set E', thus determining the *complete* edit set E^*, and to develop a constructive algorithm to obtain such a set. Another important result of this work is the definition of the *essentially new* edit set, which is a minimal subset of implicit edits to guarantee that a solution of (2)–(4) is a EIP solution. Nevertheless, a remarkable drawback of this method is that the generation of the implicit edit set can grow exponentially with respect to the cardinality of the original edit set.

Garfinkel, Kunnathur and Liepins [5] deal with the important combinatorial problem of generating the implied edits for MWFI with categorical data in a more effective way, proposing a tree-traversal procedure. Since the set of generated edits can still be too large, they also propose a cutting-plane algorithm which at each iteration solves model (2)–(4) with some additional constraints and generates a missing implied edit (if any exists). Computational experiments using both algorithms are shown on instances with $|I| = 27$ and $|J| = 21$. The above cutting-plane algorithm is extended by the same authors to continuous data in Garfinkel, Kunnathur and Liepins [6]. An important assumption in this work is that the edits are linear constraints, thus the implied edits turn to be surrogate constraints from the explicit edits. As in [5], the iterative algorithm in [6] solves a set-covering problem at each iteration and possibly a new implied edit is generated to cut-off the current integer solution of the set-covering problem. The algorithm for continuous data is computational tested on instances with $|I| = 18$ and $|J| = 12$. On the same idea, Ragsdale and McKeown [11] propose some variants of this iterative mechanism by considering two 0-1 variables x_i^+, x_i^- and the additional constraint $x_i^+ + x_i^- \leq 1$ for each field i, solving randomly generated instances with $|I| = 50$ and $|J| = 20$.

In all these proposals, an integer program must be solved to optimality at each iteration and the overall iterative procedure tends to be quite consuming time in practice.

Our contribution in this article is to describe new close-related approaches with better performances in practice. Section 2 presents a simple algorithm for categorical and continuous data. This algorithm follows a scheme similar to the cutting-plane algorithm proposed in [5], and it differs on the set of generated cuts. Another algorithm, closely related to the one proposed in [6] for continuous data and linear edits, is also presented in this article. The new algorithm has the advantage that the cuts can be generated also from non-integer solution of the set-covering problem, thus only a linear program must be solved at each iteration. However, if the final solution satisfying all the (explicit and implied) edits is non-integer, the approach follows a branch-and-bound scheme in order to achieve integrability of the variables, and the cutting-plane generation is applied at each node of the branch-decision tree. The whole procedure is a so-called *branch-and-cut algorithm*. The underlying mathematical model is presented in Section 3 and the branch-and-cut algorithm is described in Section 4. The paper finishes with an extensive computational analysis in Section 5 in which the different approaches are compared on randomly generated instances from the literature.

2 General algorithm

We discuss here an approach that can be applied to mixed data (i.e., with categorical and continuous fields) and general edits. It is similar to the cutting-plane algorithms described in Garfinkel, Kunnathur and Liepins [5, 6] and summarized as follows:

Step 0: Let $K \subseteq E$ be the set of explicit edits not satisfied by the record a;

Step 1: Let x^* be the optimal integer solution of the set-covering model (2)–(4);

Step 2: Check whether there exists a record y satisfying the edits with $y_i = a_i$ when $x_i^* = 0$ for $i \in I$. If such record does exist, then stop the procedure: x^* is the optimal solution of the problem. Otherwise, add the constraint $\sum_{i:x_i^*=0} x_i \geq 1$ to K in the the set-covering problem (2)–(4), and go to Step 1.

This procedure iteratively strengthen the set-covering problem with additional constraints, one at each iteration. Each iteration consists of checking whether the current solution x^* guarantees the existence of a valid record y (i.e., $y \in \mathcal{P}_E$) such that $y_i = a_i$ when $x_i^* = 0$, generating a cutting-plane inequality when the non-existence is proved. Since the cutting-plane inequality is not satisfied by x^*, this procedure computes a different solution of the restricted set-covering problem in the next iteration. Each inequality imposes that a new field not currently present in the solution must be modified in the next iteration.

The above-described mechanism is based on two ingredients: a set-covering problem solver and a procedure to check whether a subset of $\mathcal{P}_E$ is empty or not. Even if the set-covering problem is a difficult problem, there are in the Operational Research literature many approaches to it. The second procedure is more undefined, as it strongly depends on the data and the edits. For continuous data and linear edits the check can be easily done by, e.g., solving a linear programming model in accordance with the Farkas' Lemma (see, e.g., Schrijver [13]). In case of categorical data, the feasibility test can be a $\mathcal{NP}$-hard problem (e.g., when the correct records are the integer solutions of the polyhedron defined by the edits), but still an enumerative search of all the possibilities could be feasible in practice.

Because the cutting-plane inequality generated at each iteration is not satisfied by the current solution x^*, this constraint (called *cut*) is new and must be added to the current set-covering problem. Nevertheless, previous introduced cuts can become not relevant, or even dominated by the new cut. This happens, for example, when the set $I_k \subset I_l$ for $k > l$. Then the cut introduced in the l-th iteration can be removed. Also other non-dominated constraints should be removed from the set-covering problem to keep a reasonable number of constraints in it, but probably these non-dominated constraints must be memorized in a cut-pool structure to be examined before entering in the feasibility phase. If a violated constraint in the cut-pool is found, then it should be introduced in the set covering problem and the time of the feasibility check is saved.

When compared with other similar algorithms, as done in Section 5, our proposal has the advantage of easily generating a violated cut for an incorrect pattern x^* without determining the implied edits. Still, it has the disadvantage of this type of algorithms, in which an integer model must be solved to optimality at each iteration, thus tending to generate a lot of equivalent incorrect solutions before finding a correct optimal one. For the particular case of continuous data and linear edits, this difficulty can be avoided by applying the cut generation to also non-integer solutions, as it is presented in the next sections.

3 Mathematical Model

The aim of this section is to present better models for EIP of microdata composed by continuous numbers and where the edits are described by linear inequalities. From now on, we shall assume that each field value a_i is a continuous number in a known interval $[lb_i, ub_i]$. Moreover, we shall also assume that the

explicit edit set E is given by a collection of linear constraints, and therefore it can be represented by a linear system $My \leq b$, where M is a $m \times n$ matrix and b is a $m \times 1$ vector. Whenever more detail is needed, the linear system will be represented by $\sum_{i \in I} m_{ij} y_i \leq b_j$ for all $j \in J$, where J is the index set of the explicit edits. Without lost of generality, we will assume that $lb_i \leq a_i \leq ub_i$ for all $i \in I$, and $Ma \not\leq b$. Under these assumptions, the set of valid records $\mathcal{P}_E$ is given by the points of the polyhedron

$$\mathcal{P}_E := \{y : My \leq b, lb \leq y \leq ub\}.$$

We now present a new mixed integer linear model for EIP which has the advantage of exploiting the bounds $[lb_i, ub_i]$ to link a 0-1 variable x_i and a continuous variable y_i for each field i to produce a set of constraints more compact than the SCP model. Variable x_i assumes value 1 if and only if the field must be modified, and y_i is the modified value. Then an integer linear programming model is:

$$\min \sum_{i \in I} w_i x_i, \tag{5}$$

subject to

$$\sum_{i \in I} m_{ij} y_i \leq b_j \qquad \text{for all } j \in J \tag{6}$$

$$a_i - (a_i - lb_i)x_i \leq y_i \leq a_i + (ub_i - a_i)x_i \qquad \text{for all } i \in I \tag{7}$$

$$x_i \in \{0, 1\} \qquad \text{for all } i \in I. \tag{8}$$

Constraints (6) ensure that the corrected record y is in the valid set $\mathcal{P}_E$, and constraints (7) guarantee that the field i is not modified unless $x_i = 1$.

The model (5)–(8) is a simple way of writing the first non-linear model described in Section 1 and pointed in, e.g., Garfinkel Kunnathur and Liepins [6]. The key point for the new model is the assumption of having the external

bounds lb_i and ub_i defining the interval $[lb_i, ub_i]$ of potential values y_i for each field $i \in I$, which is not a rare assumption. Indeed, the new model remains valid even when lb_i is very small and ub_i is very big. What is more, as also pointed and used in [6], typically one knows that y_i is a non-negative number, thus $lb_i = 0$ is a realistic assignment. The advantage of model (5)–(8) is to fit into the Mixed Integer Linear Programming (MILP), and the today state-of-the-art of Mathematical Programming provides powerful tools to solve this type of models. The next section illustrates this claim describing a branch-and-cut approach.

This problem was shown to be an $\mathcal{NP}$-hard problem in the strong sense in Garfinkel, Kunnathur and Liepins [6], but using the new model (5)–(8) we can alternatively find a more simple proof. Indeed, any instance of the *set-covering problem* can be easily formulated as a particular model (5)–(8). As a result, one cannot expect to find an efficient (i.e., running in polynomial time in the size of the record) approach, but still the problem must be solved and the next section describes a way of addressing the problem. Section 5 discusses the effectiveness of the approach on solving randomly generated instances.

An important feature of model (5)–(8) is that it can be easily extended to work also on some discrete data. Indeed, if a_i is an integer number in a known interval $[lb_i, ub_i]$ (which is the case when i represents, e.g., the sex of a person, thus $a_i \in \{0, 1\}$) and if the edits can be written as linear constraints, then the model (5)–(8) with the additional constraint "y_i integer" is still a MILP model valid for the EIP on these data and edits.

A relevant disadvantage of model (5)–(8) is that in many practical situations the values lb_i and ub_i are big numbers, thus creating ill-conditioned problems. An immediate procedure to solve it is to apply a general-purpose software for Mixed Integer Linear Programming models. Nevertheless, the existence

of the external bounds lb_i and ub_i is a bad ingredient for solvers based on continuous linear programming relaxations (this is empirically confirmed by our experiments) and therefore one has to develop tricks to skip from these numerical difficulties. An example of these tricks is the branching rule. Indeed, in the problem the classical binary scheme of choosing one field i and creating to subproblems, one by fixing $x_i = 0$ and another by fixing $x_i = 1$, could create wrong solutions due to the tolerances. A better branching rule is to choose a field i and create to subproblems, one by fixing $y_i = a_i$ and another by fixing $x_i = 1$. In theory the two rules are the same but in practice, due to the finite precision of computers, they are different. Still, a more-elaborated technique is necessary, and this is covered in the next section.

4 Branch-and-Cut algorithm

Since the model (5)–(8) is a Mixed Integer Linear Programming model, one can alternatively apply Benders' Decomposition (see, e.g, [13]) to solve it. Briefly, this method consists of iteratively solving a master problem defined only by the binary variables and whose set of constraints is enlarged at each iteration by solving a subproblem defined by the continuous variables. See, e.g., Shrijver [13] for details. At first glance, there is not advantage on using this alternative approach since it requires solving a sequence of smaller problems, possibly in a large number. Nevertheless, the good news is that the constraints generated by solving the subproblems can be strengthened for this problem, and therefore the new approach can provide better linear relaxation than using directly the MILP model. Moreover, another advantage is that this decomposition technique does not deal with large bounds and decision variables in the same problem, which typically creates ill-conditioned problems in numerical analysis. This issue will be latter discussed. Let us now describe the procedure in detail.

Suppose we are given with a vector $x^* = [x_i : i \in I]$. For simplicity, we will assume that $x_i^* \in \{0, 1\}$, even if we will observe that this integrality requirement can be relaxed and the procedure be also applied with a minor modification to work on non-integer vectors. We are interested in checking if the polyhedron

$$\mathcal{P}_E(x^*) := \left\{ y : \sum_{i \in I} m_{ij} y_i \leq b_j, j \in J \; ; \; a_i - (a_i - lb_i)x_i^* \leq y_i \leq a_i + (ub_i - a_i)x_i^*, i \in I \right\}$$

is empty or not. If yes, then the pattern x^* is a feasible solution for the EIP. Otherwise, there is something wrong with x^* and we are interested in deriving a linear inequality cutting off this *infeasible* solution but not any feasible solution for the EIP. If we have a procedure to generated such inequality from a vector x^* when it exists, then it means that we already have a cutting-plane scheme to find an optimal solution x^* for EIP by simply calling the procedure iteratively. The kernel problem of finding this inequality violated by a given x^* or prove that such inequality exists is called *separation problem*, while the problem of solving the problem of finding the new vector x^* with the generated inequalities is called *master problem*. Because the master problem looks for integer solutions then, typically, a branch-decision scheme is required and the whole procedure to solved is called *branch-and-cut algorithm*. This is a modern technique for solving hard combinatorial optimization problems that have been successfully used in many real-world applications. See, e.g., Caprara and Fischetti [3].

For the EIP it is quite immediate to solve the separation problem by applying Farkas' Lemma (see, e.g., [13]) on $\mathcal{P}_E(x^*)$. In fact, this polyhedron is non-empty if and only if

$$\sum_{j \in J} \alpha_j b_j + \sum_{i \in I} \beta_i (a_i + (ub_i - a_i)x_i^*) - \sum_{i \in I} \gamma_i (a_i - (a_i - lb_i)x_i^*) \geq 0 \qquad (9)$$

for all direction of the cone:

$$\mathcal{C}_E := \{(\alpha, \beta, \gamma) : M^T\alpha + \beta - \gamma = 0, \alpha \geq 0, \beta \geq 0, \gamma \geq 0\}.$$

By simple operations on (9) we get a valid inequality of a feasible pattern x as follows:

$$\sum_{i \in I}[\beta_i(ub_i - a_i) + \gamma_i(a_i - lb_i)]x_i \geq \alpha^T(Ma - b). \tag{10}$$

Because $\beta_i, \gamma_i, ub_i - a_i, a_i - lb_i$ are non-negative numbers and because x_i must be 0 or 1, then it is possible to strengthen these constraints by rounding down a left-hand side coefficient to the right-hand side whenever it is bigger. As reported in the next section, our computational experiments proved that this strengthening is very effective for the success of the approach.

In conclusion, the alternative model for EIP on continuous data and with linear edits is given by (5) subject to (10) for all (α, β, γ) of $\mathcal{C}_E$.

Observe that in the particular case of small values of lb_i and large values of ub_i, the strengthening operation produces the following set covering constraints:

$$\sum_{i \in I'} x_i \geq 1$$

where I' is the set of field indices with a dual variable β_i or γ_i at a positive value, which is equivalent to the field indices with non-zero value in the array $M^T\alpha$. This is a quite interesting observation since the obtained inequalities coincide with the inequalities generated by the method proposed by Garfinkel, Kunnathur and Liepins [6]. Still, an improvement of the above-proposed approach is that it can be also applied when x_i^* are non-integer solutions, which make the approach quite suitable to work in a branch-and-cut framework. Indeed, x^* is a parameter during the application of the Farkas' Lemma, and therefore it can also be applied when x^* is a non-integer vector. This implies that the branch-and-cut algorithm does not need to produce an integer solution of a relaxed

model before solving the separation procedure, since it can also be defined from fractional solution of linear programming relaxations. This observation allows cutting off infeasible solutions without going deep in branch-decision trees. Nevertheless, fractional solutions can define separation problems which do not generate cuts, and then the branching phase of the whole procedure must be applied.

We now address how to manage the large amount of constraints (10). Obviously, we only need to impose these constraints for the extreme directions of $\mathcal{C}_E$, but still they are in a so large amount that one cannot expect to solve a 0-1 linear program with all of them. The good news is that we can replace the resolution of such huge program by a sequence of medium-size programs, generating a violated one from the current solution x^* at each iteration. Indeed, this scheme coincides with the one used in Garfinkel, Kunnathur and Liepins [5, 6], but with the important advantage that now our approach does not need to solve an integer program at each iteration because a most-violated cutting-plane (if any) can be determined from a (possibly fractional) solution of the linear relaxation. The way to do it for a given solution x^* is by solving the linear program:

$$\min \left\{ \sum_{j\in J} \alpha_j b_j + \sum_{i\in I} \beta_i (a_i + (ub_i - a_i)x_i^*) - \sum_{i\in I} \gamma_i (a_i - (a_i - lb_i)x_i^*) \ : \ (\alpha,\beta,\gamma) \in \mathcal{C}_E \right\}$$

If the optimal objective value of this linear program is non-negative, then there is no violated inequality and therefore x^* guarantee a valid record (i.e., $\mathcal{P}_E(x^*) \neq \emptyset$). Otherwise, the problem is unbounded and a direction $(\alpha^*, \beta^*, \gamma^*)$ with negative objective function value defines a violated inequality (10) to be considered. The procedure finishes when no violated inequality is identified and the solution x^* is integer. Because the number of extreme directions in $\mathcal{C}_E$ is finite then the iterative procedure will stop (in theory) with an optimal solution.

Our model can also benefic by strengthening the separated cutting-plane inequalitoes, and also from the addition of other well-known inequalities useful in this kind of models (e.g., cover inequalities, flow inequalities, Gomory cuts, etc.). Also, as it is usually done in branch-and-cut algorithms, the management of the cuts is very important in order to keep a reasonable linear relaxation to be solved at each iteration. This is possible since not all the so far generated cuts are useful at each iteration, and therefore they do not need to be in the current linear program. Nevertheless, a cut-pool structure is convenient in the algorithm to save previously generated cuts instead of simply deleting the cut from the linear program. Indeed, in forthcoming iterations the removed generated inequalities can turn to be violated (and therefore useful) again. We do not go here into these important tricks, which are standards of modern branch-and-cut algorithms. See, e.g., [13] for details.

Further improvements can be applied to strengthen some constraints (10). Let $\sum_{i \in I'} \delta_i x_i \geq 1$ be one of this inequalities. For the above considerations we can assume that $0 < \delta_i \leq 1$ for all $i \in I'$. If the sum of all the coefficients $\delta_i < 1$ is also smaller than 1 then we can improve the inequality by rounding down all the coefficient to an integer number, thus producing a set-covering inequality. This consideration appeared to be speed up the whole algorithm on our computational results.

Another classical way of speeding up the procedure is to generate more than one violated cut (if any) at each iteration, and this is easily performed by re-optimizing the subproblem on the cone $\mathcal{C}_E$ after increasing the cost of the non-basic variable with negative reduced cost that determined the unboundness of the subproblem. In this way, other different directions can be generated in the same iteration, and therefore the linear relaxation turns to be more complete (i.e., with more implied edits) with each iteration and, hopefully, reduce the

total number of necessary iterations.

An interesting remark is that all the separation problems share the same feasible region $\mathcal{C}_E$, and they only differ in the objective function. Because each separation problem is a linear program, by Duality Theory we can alternatively solve the dual version which is a more compact program since the upper and lower bound on the variables y_i. A dual solution α^* will automatically produce the vector $(\alpha^*, \beta^*, \gamma^*)$ because $\beta^* := \max\{M^T\alpha, 0\}$ and $\gamma := \max\{-M^T, 0\}$.

5 Preliminary Computational Experiments

To measure the effectiveness of our proposal compared with other previous works, we have implemented several algorithms, using the general software CPLEX 8.1 [7] as a framework for the mathematical models. In particular, we have considered the following algorithms:

Algorithm 1: It is a classical branch-and-bound approach for the mixed integer linear model (5)–(8), where the bound is computed by solving the linear relaxation. Special considerations are necessary to reduce the ill-conditioned numerical problems.

Algorithm 2: It is the cutting-plane algorithm described in Garfinkel, Kunnathur and Liepins [6], where an optimal integer solution of a set-covering problem is computed at each iteration.

Algorithm 3: It is the modified version of Algorithm 2 proposed by Ragsdale and McKeown [11] with double number of binary variables in the integer program of each iteration.

Algorithm 4: It is the branch-and-cut algorithm described in Section 4 when the separation problem is applied only on integer vectors x^*.

Algorithm 5: It is the branch-and-cut algorithm described in Section 4, which is an improved Benders' decomposition approach to solve model (5)–(8).

As described, Algorithm 5 exploits the fact that the separation problem described in Section 4 can be also applied to non-integer vectors, which is not considered in Algorithm 4. On the other hand, Algorithm 4 and Algorithm 2 share the fact that both solve the same separation problems, while they differ in the master problems: Algorithm 2 solves a set-covering problem and Algorithm 4 solves a more general integer program. Algorithm 3 is the variant of Algorithm 2 introduced in [11], so we can measure the impact of the differences, and Algorithm 1 is the compact mixed integer model equivalent to the Bender's decomposition approach addressed by Algorithm 5.

Since there is not a benchmark collection of test-bed instances in the literature, we have conducted our experiments on three classes of instances. The instances in Class I are randomly generated as described in Ragsdale and McKeown [11], which means:

- the number of fields is $|I| = n = 50$,
- the number of edits is $|J| = m = 20$,
- the weights are all identical ($w_i = 1$ for all $i \in I$),
- the record values are uniformly generated in the interval [-100,+100], thus $lb_i = -100$ and $ub_i = 100$ for all $i \in I$,
- the right-hand side b_j where generated in the interval [0,1000],
- the elements m_{ij} are zero with probability 0.2, and the non-zero values are generated in [1,20] with probability 0.3 and in [-20,-1] with probability 0.7.

The generated records satisfying all the edits were removed, and the others were classified according to the number of failed edits into five groups: $[1, 4]$, $[5, 8]$, $[9, 12]$, $[13, 16]$ and $[17, 20]$. The random generator was executed until we get five instances in each group, as done in [11].

Because of the small size of the previously described instances, we could not obtain conclusive results, and therefore we have also generated a second class of instances. The instances of Class II were identically generated as before but with $|I| = 100$ and $|J| = 40$. Furthermore, we have generated three families inside Class II by varying the range of the variable's bounds in the intervals $[-10^3, 10^3]$, $[-10^4, 10^4]$ and $[-10^5, 10^5]$ respectively, to study the influence of these bounds on the algorithm performances.

Finally, the instances in Class III are artificial instances supplied by US Census consisting of 10,994 records with 17 fields and two set of edits. The first set contains 136 edits of type:

$$dlb_j \leq \frac{a_i}{a_k} \leq dub_j \qquad \text{for some } i, k \in I (i < k) \text{ and } j \in J,$$

in which $dub_j - dlb_j$ take values between 10^{-1} and 10^7. The second set of edits contains to two balancing edits of type:

$$a_i + a_k = a_l \qquad \text{for } i, k, l \in I.$$

All the algorithms were implemented by the same programmer in C++ programming language, and all the experiments were executed on a personal computer Pentium 1500 Mhz under Windows XP. Tables 1–3 shows the computational results on the instances of Classes I–III, respectively. Each line in the Tables 1 and 3 represents the average results over five random instances, while in Table 2 the average is given by the number of records given in column #. The first column in Tables 1 and 2 and the second column in Table 3 gives the range of failed explicit edits by the records. The first column in Table

3 represents the interval used to generate the bounds of each instance, thus providing the three record families. Column *Obj.* is the (average) number of records to be modified in an optimal solution. Column *Nodes* is the (average) number of nodes explored in the branch-search tree of the algorithm, and *Sec.* is the time in seconds of the personal computer consumed by the whole algorithm. Column (10) gives the number of inequalities (10), which is close to the number of separation problems solved, and Column *Cliq.* is the number of clique inequalities (...) generated. Column *Limit* gives the ratio of the number of instances solved to optimality within a time limit of 1 hour over all the trials.

Table 1 shows the computational results running Algorithms 1, 4 and 5 on the instances in Class I. Algorithms 1 and 5 were both very efficient to solve all the instances, while Algorithm 4 was not. Indeed, from our experiments we observed that algorithms where the master problem is an integer program tends to require a lot of iterations. This is because our instances have w_i and therefore there are a lot of infeasible solutions with the same objective function, and the algorithm seems to required a different cut for eliminating each one which, unfortunately, was obtained after solving a difficult (integer) master problem. This behaviour was not observed when the separation problem works on non-integer solutions, thus the (continuous) master problem can be solved faster and the cut can be inserted earlier.

Table 2 tries to compare Algorithms 1 and 5 on more difficult instances. In our experiments we get better performances by using Algorithm 5, specially on solving instances with larger bounds. Indeed, when lb_i and ub_i are generated in $[-10^4, 10^4]$, Algorithm 1 did not solve the five instances in all cases, while Algorithm 5 was always successful. The situation was worse for Algorithm 1 on the third family of records and a large number of failed records, while this type of instances seems even easier than the one in the previous class for Algorithm

5.

A similar conclusion can be derived from Table 3, which again resulted to be simple instances for our new proposals.

6 Conclusions

We have addressed the combinatorial problem of finding the fewest number of fields to modified in a record to produce another record that satisfies a set of edits. When the field values are continuous numbers and the edits are linear equations we presented two integer linear programming models that exploits the fact that imputation values are always inside known bounds. The first model is a mixed integer model while the second is based on the Benders' decomposition approach. We have described and tested approaches for solving both models, observing the better performances of the second proposal on difficult instances.

References

[1] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik* 4 (1962) 238–252.

[2] R. Bruni, A. Sassano, "Logic and optimization techniques for an error free data collecting", working paper, University of Roma, 2001.

[3] A. Caprara, M. Fischetti, "Branch and cut algorithms". In M. Dell'Amico, F. Maffioli, S. Martello (eds), *Annotated Bibliographies in Combinatorial Optimization*, Wiley, Chichester, 45–63, 1997.

[4] I.P. Fellegi, D. Holt, "A systematic approach to automatic edit and imputation", *Journal of the American Statistical Association* 71 (1976) 17–35.

Table 1: Average results on five instances from Class I

		Algorithm 1		*Algorithm* 5				*Algorithm* 4	
	Obj.	*# Nodes*	*Sec.*	(10)	*Cliq.*	*# Nodes*	*Sec.*	*Sec.*	*Limit*
$[1, 5]$	3.4	2.0	0.047	15.6	0.0	3.2	0.043	10.953	3/5
$[6, 10]$	5.0	8.4	0.066	24.4	0.0	12.4	0.078	-	0/5
$[11, 15]$	5.6	30.2	0.116	65.4	0.0	33.8	0.206	1.256	1/5
$[16, 20]$	6.8	149.6	0.418	91.2	0.0	34.0	0.287	-	0/5
$[21, 25]$	7.4	289.2	0.631	385.6	0.0	223.4	1.259	-	0/5

Table 2: Average results on five instances from Class II

			Algorithm 1			*Algorithm* 5		
		Obj.	*# Nodes*	*Sec.*	*Limit*	(10)	*# Nodes*	*Sec.*
	$[1, 10]$	6.4	926.4	6.2	5/5	1189.4	631.2	6.7
	$[11, 20]$	9.2	6297.4	27.7	5/5	1641.4	661.6	9.4
$[-10^3, 10^3]$	$[21, 30]$	10.0	9822.2	47.1	5/5	17816.0	4438.0	115.6
	$[31, 40]$	12.0	8752.8	53.1	5/5	14348.6	4170.2	88.1
	$[41, 50]$	12.4	6336.0	44.6	5/5	17275.0	3511.2	120.9
	$[1, 10]$	3.6	1389.2	17.7	5/5	138.8	122.6	1.0
	$[11, 20]$	5.0	25644.8	187.4	5/5	2254.6	1159.6	13.8
$[-10^4, 10^4]$	$[21, 30]$	5.6	149298.5	1595.9	4/5	13319.8	3199.4	92.0
	$[31, 40]$	5.8	93463.0	1011.3	4/5	19524.0	4104.2	127.2
	$[41, 50]$	6.8	232025.0	3063.8	4/5	87244.0	15334.6	1452.4
	$[1, 10]$	3.6	6501.8	54.0	5/5	100.4	67.8	0.6
	$[11, 20]$	4.8	96563.5	1366.0	4/5	1741.4	1930.4	14.4
$[-10^5, 10^5]$	$[21, 30]$	6.4	-	-	0/5	2210.2	2536.2	17.0
	$[31, 40]$	6.6	-	-	0/5	3868.6	4063.8	31.0
	$[41, 50]$	7.6	-	-	0/5	7054.0	7621.0	57.2

Table 3: Average results on instances from Class III

			Algorithm 1		*Algorithm* 5			
	#	*Obj.*	# *Nodes*	*Sec.*	(10)	*Cliq.*	# *Nodes*	*Sec.*
$[1, 15]$	223	1.75	0.229	0.008	0.135	0.000	0.000	0.004
$[16, 30]$	5384	2.91	0.673	0.016	0.297	0.004	0.001	0.005
$[31, 45]$	4281	4.10	1.342	0.026	0.308	0.008	0.000	0.005
$[46, 60]$	1018	5.28	3.949	0.037	0.315	0.006	0.000	0.006
$[61, 75]$	87	6.20	8.034	0.046	0.287	0.126	0.000	0.005

[5] R.S. Garfinkel, A.S. Kunnathur, G.E. Liepins, "Optimal imputation of errorneous data: continuous data, linear constraints", *Operations Research* 34 (1986) 744–751.

[6] R.S. Garfinkel, A.S. Kunnathur, G.E. Liepins, "Error location for errorneous data: continuous data, linear constraints", *SIAM Journal on Scientific and Statistical Computing* 9 (1988) 922–931.

[7] ILOG, "User's Guide & References", Version 7, 2000.

[8] J. Kovar, W.E. Winkler, "Editing economic data", working paper, 2000.

[9] G. E. Liepins, "A rigorous, systematic approach to automatic data editing and its statistical basis" ORNL/TM -7126, 1980.

[10] P.G. McKeown, "A mathematical programming approach to editing of continuous survey data", *SIAM Journal on Scientific and Statistical Computing* 5 (1984) 785–797.

[11] C.T. Ragsdale, P.G. McKeown, "On solving the continuous data editing problem", *Computers & Operations Research* 23 (1996) 263–273.

[12] J. Schaffer, "Procedure for solving the data-editing problem with both continuous and discrete data types", *Naval Research Logistics* 34 (1987) 879–890.

[13] Schrijver, A. *Theory of linear and integer programming*, John Wiley & Sons (1986).