Constrained global optimization for wine blending

Philippe Vismara, Remi Coletta, Gilles Trombettoni

constraints manuscript No.
(will be inserted by the editor)

Constrained Global Optimization for Wine Blending

Philippe Vismara · **Remi Coletta** ·
Gilles Trombettoni

Received: date / Accepted: date

Abstract Assemblage consists in blending base wines in order to create target wines. Recent developments in aroma analysis allow us to measure chemical compounds impacting the taste of wines. This chemical analysis makes it possible to design a decision tool for the following problem: given a set of target wines, determine which volumes must be extracted from each base wine to produce wines that satisfy constraints on aroma concentration, volumes, alcohol contents and price. This paper describes the modeling of wine assemblage as a mixed constrained optimization problem, where the main goal is to minimize the gap to the desired concentrations for every aromatic criterion. The deterministic branch and bound solvers `Couenne` and `IbexOpt` behave well on the wine blending problem thanks to their interval constraint propagation/programming and polyhedral relaxation methods. We also study the performance of other optimization goals that could be embedded in a configuration tool, where the different possible interactions amount to solving the same constraints with different objective functions. We finally show on a recent generic wine blending instance that the proposed optimization process scales up well with the number of base wines.

Keywords Wine · Wine blending · Assemblage · Interval constraint programming · Global optimization · Mixed numerical CSP · Mixed integer nonlinear programming

1 Introduction

Assemblage consists in blending wines from different vineyard plots and/or different grape varieties, each contributing its own special flavor.

Wine blending is generally carried out by oenologists working for wineries. Oenologists can obtain wine blendings of the highest quality, but taste saturation entails a strong limit in the number of daily wine tasting sessions. Therefore the Nyseos company (`www.nyseos.fr`),

P. Vismara, R. Coletta, G. Trombettoni
LIRMM, UMR5506 Université de Montpellier - CNRS, Montpellier, France
Tel.: +33-4-67418541
Fax: +33-4-67418500
E-mail: {Philippe.Vismara, Remi.Coletta, Gilles.Trombettoni}@lirmm.fr

P. Vismara
MISTEA, UMR0729 Montpellier SupAgro - INRA, Montpellier, France

which submitted the blending problem to us, provides chemical analysis tools to avoid a number of tasting sessions. These tools can analyze wine aromas by measuring a set of chemical compounds that impact wine taste [11]. These tools make it possible to develop a decision-support software for the following problem: given a set of target wines to be produced, which volumes must be taken from each base wine in order to make wines satisfying constraints on aroma concentrations, volumes, alcohol content, price, etc.

Moore and Griffin have shown that aroma concentrations of a wine blending satisfy linear constraints [22]. However, several other requirements lead to nonlinear constraints. For instance, the Nyseos company works on a model able to predict the color of a wine. The model will not be linear and the complexity of color modeling is confirmed by other researches [13]. Another critical point is that no less than a given amount of wine can be transferred from a tank to a target because of the loss of liquid in the pipes and the manipulation cost. As we will see in this article, this requirement leads to a disjunctive constraint that can be modeled by Boolean variables. Finally, trying to minimize the gap to the desired concentrations for every aromatic criterion and/or minimize the volume gap between base and target wines leads to nonlinear constraints.

Research papers about alcohol blending generally describe the chemical process and protocol used for a given blend. For instance, [16] describes how to obtain a specific type of Muscat. They use a manual optimization process to select the best blend among a few trials.

An interesting algorithmic research on wine blending was presented in [13]. An artificial neural network approach was used to select the wine quantities extracted from each base in order to elaborate a wine matching predefined aromatic criteria. In this work, aromatic criteria were not chemically analyzed. Instead, a panel of students carried out tasting to quantify predefined criteria. The neural network performed multicriteria optimization for adjusting each aroma. The comparison with our approach is difficult in terms of quality since we resort to monocriterion optimization. In addition, no performance (CPU time) results were shown in [13]. In another research [12], the main objective was to find the best matching between chromatograms of base and target wines. This problem was modeled by a *non constrained* nonlinear optimization solved by a local (Nelder-Mead) optimization method.

In this article, we present a mathematical modeling of the wine assemblage problem defined in Section 2. This allows us to design simultaneously several target wines. We build a constrained optimization model that minimizes in each target wine the gap between desired aromatic concentrations and obtained concentrations, while taking into account the minimal transfer disjunctive constraint. The problem is modeled by a mixed (discrete and continuous) constrained optimization problem (COP), from which absolute value and max operators have been removed (see Section 3).

We use `IbexOpt` [10, 9] and `Couenne` [4] solvers to solve the implemented model. Both apply an interval Branch & Bound using mathematical programming and interval constraint programming methods briefly described in Section 4. `Couenne` is endowed with integer variables and can thus directly handle our mixed COP. In Section 5, we validate the model on two real instances provided by the Nyseos company and on an artificial instance mixing both instances. We also study the performance of other optimization goals that could be embedded in a configuration tool, where the different possible interactions amount to solving the same constraints with different objective functions.

In Section 5.4, we finally test our constrained optimization tool on the latest wine blending problem provided by Nyseos. This corresponds to a generic wine blending instance where the number of wine bases can be significantly increased up to 20. Also, the number of aromas is even greater (Nyseos is able to chemically analyze more and more aromas over time). The experiment shows how `Couenne` performs when the problem scales up.

2 The wine assemblage problem

Figure 1 illustrates the definition of wine assemblage.

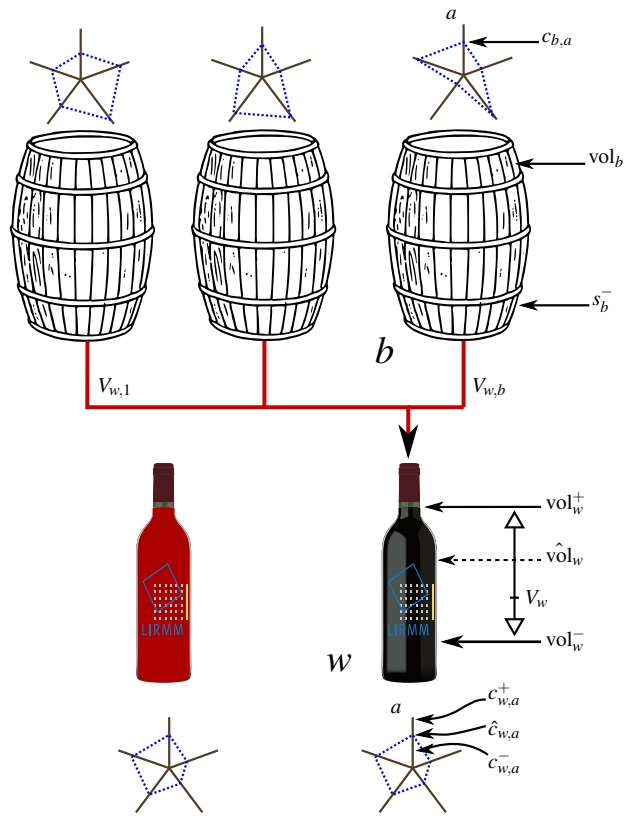


Fig. 1 Wine assemblage

We consider a set of base wines numbered from 1 to $\mathscr{B}$. We denote by vol_b the volume of the base $b \in 1..\mathscr{B}$.

For different reasons, it is sometimes impossible to completely empty a base tank. Let s_b^- be the minimum volume that must remain in tank b. (We have: $0 \leq s_b^- \leq \mathrm{vol}_b$.) All base wines are analyzed in order to measure the concentration of selected key aroma compounds. These compounds are numbered from 1 to $\mathscr{A}$. We denote by $c_{b,a}$ the concentration of aroma a in base b.

A wine assemblage support tool should help to simultaneously build *several* target wines from a given set of bases. This is one of the major benefits of our approach compared to the traditional approach of oenologists. Hence, we consider a set of target wines, numbered from 1 to $\mathscr{W}$. For each wine w, we aim at producing an optimal volume $\hat{\mathrm{vol}}_w$. The final volume V_w of wine w should be as close as possible to $\hat{\mathrm{vol}}_w$ and must remain greater (resp. smaller)

than a given lower bound vol_w^- (resp. an upper bound vol_w^+), i.e.:

$$\forall w \in 1..\mathscr{W},\ \ \text{vol}_w^- \leq V_w \leq \text{vol}_w^+ \tag{1}$$

These bounds are used to fulfill an order of a specific volume or to avoid producing an excessive volume.

Each target wine w is a blend of wines extracted from several tanks. We denote by $V_{w,b}$ the volume of wine w that is pumped from base tank b. We have a direct relation with V_w:

$$\forall w \in 1..\mathscr{W},\ \ V_w = \sum_{b=1}^{\mathscr{B}} V_{w,b} \tag{2}$$

Furthermore, all the volumes extracted from the same base tank b must leave a minimum volume s_b^- in the tank.

$$\forall b \in 1..\mathscr{B},\ \ s_b^- \leq \text{vol}_b - \sum_{w=1}^{\mathscr{W}} V_{w,b} \tag{3}$$

When transferring wine between two tanks, a subpart is generally wasted in the pipes. Hence it is impossible to transfer very small volumes. If δ_V is the minimum volume that can be transferred between two tanks, we define the following *disjunctive constraint*:

$$\forall w \in 1..\mathscr{W}, \forall b \in 1..\mathscr{B},\ (V_{w,b} = 0)\ \vee\ (\delta_V \leq V_{w,b}) \tag{4}$$

In addition to volume, each target wine is described in terms of aroma compound concentration. For a given wine w, we denote by $\hat{c}_{w,a}$ the desired concentration of aroma a.

The actual concentrations of aroma a in target wine w ($C_{w,a}$) are to be as close as possible to $\hat{c}_{w,a}$ within an interval $[c_{w,a}^-, c_{w,a}^+]$, where $c_{w,a}^-$ (resp. $c_{w,a}^+$) denotes the minimum (resp. maximum) admissible concentration of aroma a in wine w. The relation between volumes and concentrations can be formulated as follows:

$$\forall w \in 1..\mathscr{W},\ \forall a \in 1..\mathscr{A},\ \ c_{w,a}^-\ \leq\ C_{w,a} = \frac{1}{V_w} \sum_{b=1}^{\mathscr{B}} (V_{w,b}\,.\,c_{b,a})\ \leq\ c_{w,a}^+ \tag{5}$$

In a similar way, we can model constraints on alcohol content or price per liter for the target wines. These can be treated like additional aromas.

3 A mixed COP formulation for wine blending

We can model the wine blending problem as a mixed constrained optimization problem (COP) – also known as mixed integer nonlinear program (MINLP). The bound constraints are directly modeled below by bounded domains, i.e. intervals.

3.1 *Variables*

For each wine $w \in 1..\mathscr{W}$ and each base $b \in 1..\mathscr{B}$, we create:

- a variable $V_{w,b}$ with a domain $D(V_{w,b}) = [0, \overline{V_{w,b}}] = [0, \min(\text{vol}_w^+, \text{vol}_b)]$ representing the volume coming from the base b to the wine w, and
- a 0/1 variable $P_{w,b}$. The introduction of this 0/1 variable will avoid an explicit definition of the disjunctive constraint (4).

For each volume of a wine $w \in 1..\mathscr{W}$, we also define a variable V_w of domain $[\text{vol}_w^-, \text{vol}_w^+]$ (see (1)).

3.2 *Constraints*

The system of constraints of our COP is described below.

- The channeling constraint (2) becomes:

$$\forall w,\ V_w - \sum_{b=1}^{\mathscr{B}} V_{w,b} = 0 \tag{2.i}$$

- The surplus constraint (3) also remains similar:

$$\forall b \in 1..\mathscr{B}, s_b^- \leq \mathrm{vol}_b - \sum_{w=1}^{\mathscr{W}} V_{w,b} \tag{3.i}$$

For handling realistic volumes (due to (4)), for each wine $w \in 1..\mathscr{W}$ and each base $b \in 1..\mathscr{B}$, we add:

$$P_{w,b}.\delta_V \leq V_{w,b} \leq P_{w,b}.\overline{V_{w,b}} \tag{4.i}$$

where δ_V is the minimal volume that can be transferred between two tanks.

Aroma concentration requirements (see (5)) are decomposed into two constraints, and both parts of inequalities are multiplied by the positive volume V_w. $\forall w \in 1..\mathscr{W}$ and $\forall a \in 1..\mathscr{A}$, we have:
for the lower bound,

$$0 \leq \sum_{b=1}^{\mathscr{B}} V_{w,b} \,.\, (c_{b,a} - c_{w,a}^-) \tag{5.i -}$$

and for the upper bound:

$$0 \leq \sum_{b=1}^{\mathscr{B}} V_{w,b} \,.\, (c_{w,a}^+ - c_{b,a}) \tag{5.i +}$$

3.3 *A Min-Max for reaching the highest quality of wines*

Objective function

In this application, the significant criterion is wine quality. Bear in mind that a target wine is defined by a set of desired concentrations $\hat{c}_{w,a}$ for each of its aromas. Therefore, a way to optimize the quality of *one* target wine is to minimize a weighted sum of differences between the desired concentrations $\hat{c}_{w,a}$ and the obtained concentrations $C_{w,a}$ (see (5)). Another goal is to reach the desired volume of each target wine to produce, i.e. to minimize a difference between the desired volume $\hat{\mathrm{vol}}_w$ and the actual one V_w. Note that we could aggregate other criteria by weighted sums, such as errors on alcohol content or price.

Finally, we want to minimize the maximal error on the set of target wines. Thus, we obtain the following objective function:

$$\max_{w \in 1..\mathscr{W}} \Omega_w (\lambda_{vol_w} \,.\, e_{vol_w} + \sum_{a \in 1..\mathscr{A}} (\lambda_{w,a} \,.\, e_{w,a})) \tag{6}$$

where:

- Ω_w is a parameter reflecting how important is a given wine w (Ω_w is assumed to be in $[0,1]$). This weight ensures a more accurate blending to the best wines among the targets.

- $e_{w,a}$ denotes the discrepancy between $\hat{c}_{w,a}$ and $C_{w,a}$.
- e_{vol_w} denotes the discrepancy between the volume $\hat{\text{vol}}_w$ of wine w desired and the volume V_w obtained.
- $\lambda_{w,a} \in [0,1]$ defines the weight of aroma a in the wine w. $\lambda_{vol_w} \in [0,1]$ weights the satisfaction of the volume requirement of the target wine w compared to the satisfaction of aroma concentrations. For a given target wine w, we assume that $\lambda_{vol_w} + \sum_{a \in 1..\mathscr{A}} \lambda_{w,a} = 1$.

Remarks

A natural question that arises is who determines these weights. The Ω_w weights reflecting the importance of a given target wine are fixed by the winery/client. The user interface asks the client a quantitative or qualitative value (e.g., a number of stars) that is translated into a coefficient between 0 and 1. The weights $\lambda_{w,a}$ are chosen by Nyseos or by the oenologists working in the wineries, provided they have a knowledge of the impact of such or such chemical component (aroma) on the final taste of a given wine. These weights can be tuned individually or by family of aromas.

Eq.(6) is one way to resort to monocriterion optimization based on the multiple criteria we have to take into account. For a given target wine, the weighted error on aroma concentrations (i.e., $\sum_{a \in 1..\mathscr{A}} \lambda_{w,a} \cdot e_{w,a}$) is a fine way to obtain the desired taste. An alternative amounts to aggregating the $e_{w,a}$ errors with a least-square formula (or 2-norm), minimizing the sum of the square of the errors (that are positive in our case). This least-square variant is studied in the experimental part (see Section 5.4).

The way of aggregating the different target wine errors also leads to interesting discussions. In particular, the min-max approach could be replaced by a *leximin* approach [8]. That approach would order the target wines by decreasing status, stating that the first wine is infinitely prefered to the second, itself infinitely prefered to the third one, and so on, in a lexicographic order. Although interesting, the leximin approach does not meet the needs of the company. Indeed, in a traditional wine blending (made by oenologists in a winery), the target wine blendings are computed one after the other, in decreasing order of preference, which simulates the leximin criterion. However, this approach can produce a very good first-class wine at the price of a very bad second-class wine. Nyseos prefers a slight deterioration of the first-class wine provided the second-class one reaches (nearly) the desired taste. The tool proposed precisely aims at producing several wines simultaneously for this purpose. This is obtained by the Ω_w coefficients we associate to each wine and the simpler min-max criterion we use (see Eq.(6)).

Taking into account uncertainties in measures

All the parameters, including the aroma concentrations, are measured with a given uncertainty. ε_a denotes the measure error related to the concentration of aroma a. We thus want to minimize the gap between $\hat{c}_{w,a}$ and $C_{w,a}$ within the limit given by this uncertainty ε_a. In other words, if the gap between the desired and obtained concentrations remains below the uncertainty, it will be considered as being null in the objective function. Thus, the variable $e_{w,a}$ describes the normalized concentration error in each aroma a for each wine w, as follows:

$$\boldsymbol{e}_{w,a} = \max\left(\frac{|C_{w,a} - \hat{c}_{w,a}|}{\hat{c}_{w,a}} - \varepsilon_a, 0\right) \tag{7}$$

We can also describe the gap e_{vol_w} between the volume of a target wine V_w obtained and the volume $\hat{\text{vol}}_w$ desired with a similar expression:

$$\boldsymbol{e}_{vol_w} = \max\left(\frac{\hat{\text{vol}}_w - V_w}{\hat{\text{vol}}_w} - \varepsilon_{vol}, 0\right) \tag{8}$$

Compared to the previous formula, the removal of the absolute value simply means that no error is taken into account if the volume V_w computed falls between the maximum volume vol_w^+ and the target $\hat{\text{vol}}_w$. This is illustrated by Figure 2.

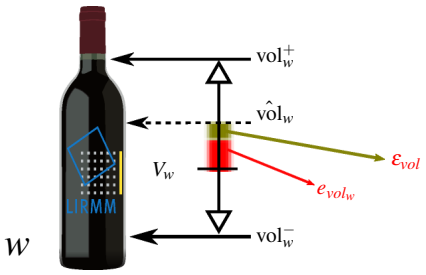


Fig. 2 Visualization of the gap e_{vol_w}

Following a usual way to define a Min-Max problem, we add a variable $E \in [0, +\infty]$ to be minimized and the following constraints:

$$\forall w \in 1..\mathscr{W},\ \ \Omega_w(\lambda_{vol_w}.e_{vol_w} + \sum_{a \in 1..\mathscr{A}} (\lambda_{w,a} \,.\, e_{w,a})) \leq E \tag{9}$$

Removing max and absolute value operators

In order to enhance the portability of our model and improve the performance, we attempt to remove max and absolute value operators. Observe that the maximum operator can be defined by

$$e = max(x,y) \ \equiv\ e \geq x \wedge e \geq y \wedge (e = x \vee e = y).$$

In addition, if the quantity e must be minimal for any reason, the last conjunct can be removed, thus simplifying the max operator. We can apply this simplification to (7). Indeed, $\lambda_{w,a}$ is positive so that minimizing E entails minimizing every variable $e_{w,a}$. Hence:

$$\forall w \in 1..\mathscr{W}, \forall a \in 1..\mathscr{A},\ \ ((e_{w,a} + \varepsilon_a)\,\hat{c}_{w,a} \geq |C_{w,a} - \hat{c}_{w,a}|) \ \wedge\ (e_{w,a} \geq 0) \tag{10}$$

The same simplification can be applied to (8), as follows:

$$\forall w \in 1..\mathscr{W},\ \ ((e_{vol_w} + \varepsilon_{vol})\,\hat{\text{vol}}_w \geq (\hat{\text{vol}}_w - V_w)) \ \wedge\ (e_{vol_w} \geq 0) \tag{11}$$

We can also remove the absolute value operator above that can be transformed into a max operator as follows:

$$\begin{aligned} e = |x| \ &\equiv\ e = max(x,-x) \\ &\equiv\ e \geq x \ \wedge\ e \geq -x \ \wedge\ (e = x \vee e = -x) \end{aligned}$$

Once more, if the quantity e must be minimal for any reason, the last conjunct can be removed, thus replacing the absolute value operator with two inequalities. We can apply this simplification to (10). Indeed, remember that every variable $e_{w,a}$ must be minimized and observe that $\hat{c}_{w,a}$ is positive. Thus,
$\forall w \in 1..\mathscr{W}, \forall a \in 1..\mathscr{A}$,

$$(e_{w,a} + \varepsilon_a)\,\hat{c}_{w,a} \geq \frac{1}{V_w} \cdot \sum_{b=1}^{\mathscr{B}} (V_{w,b}.c_{b,a}) - \hat{c}_{w,a}$$

$$(e_{w,a} + \varepsilon_a)\,\hat{c}_{w,a} \geq -\frac{1}{V_w} \cdot \sum_{b=1}^{\mathscr{B}} (V_{w,b}.c_{b,a}) + \hat{c}_{w,a}$$

Multiplying both parts of these inequalities by the positive volume V_w, we finally obtain the following three categories of constraints: $\forall w \in 1..\mathscr{W}, \forall a \in 1..\mathscr{A}$,

$$e_{w,a} \geq 0 \tag{12}$$

$$V_w.(e_{w,a} + \varepsilon_a + 1)\,.\,\hat{c}_{w,a} - \sum_{b=1}^{\mathscr{B}} (V_{w,b}.c_{b,a}) \geq 0 \tag{13}$$

$$V_w.(e_{w,a} + \varepsilon_a - 1)\,.\,\hat{c}_{w,a} + \sum_{b=1}^{\mathscr{B}} (V_{w,b}.c_{b,a}) \geq 0 \tag{14}$$

As a result, we have succeeded in suppressing from our initial model all the absolute value and max operators. Although some interval nonlinear constraint solvers like `IbexOpt` can handle these operators,[1] the performance is thus increased and the simplified model can also be implemented in most of the solvers.

3.4 *Summary*

In addition to the variables $P_{w,b}$, $V_{w,b}$ and V_w defined in Section 3.1, we define new variables for the objective function: one variable $E \in [0, +\infty]$, $\mathscr{W}.\mathscr{A}$ variables $e_{w,a} \in [0,1]$ (that absorb the unary constraints (12)) and $\mathscr{W}$ variables $e_{vol_w} \in [0,1]$ that absorb the unary constraints of (11).

In addition to the constraints (2.i), (3.i), (4.i), (5.i-), (5.i+) defined in Section 3.2, we define new constraints for the Min-Max: (9), (11), (13), (14). The objective function simply consists in minimizing the value of the variable E.

Recall that the aroma concentrations of a wine blending satisfy linear constraints [22]. However, the disjunctive constraint (4) satisfying the minimum volume transfer requirement makes appear Boolean variables and the problem becomes a mixed integer program (MIP). Furthermore, the optimization part mixes error variables and volumes in bilinear constraints (see (13) and (14)) that make the problem nonlinear. These bilinear constraints make the problem nonconvex. Therefore, MIP solvers like IBM Ilog CPLEX cannot handle this problem since they do not accept quadratic constrained programs that are not convex [1].

[1] Inverse operations of *abs* and *max* must be implemented for the constraint propagation (second phase of HC4-Revise [6]) and generalized gradients must be developed for the polyhedral relaxation.

4 Solving the COP with B&B

4.1 *Constrained global optimization*

A mixed constrained optimization problem (COP), is defined as follows.

Definition 1 (Constrained optimization problem)
Consider vectors of real $X = (x_1,...,x_{|X|})$ and integer $Y = (y_1,...,y_{|Y|})$ variables, varying in a domain $\mathscr{D}(X) \times \mathscr{D}(Y) = D(x_1) \times \cdots \times D(x_{|X|}) \times D(y_1) \times \cdots \times D(y_{|Y|})$, a function $f : \mathbb{R}^{|X|} \times \mathbb{N}^{|Y|} \to \mathbb{R}$, vector-valued functions $G : \mathbb{R}^{|X|} \times \mathbb{N}^{|Y|} \to \mathbb{R}^m$ and $H : \mathbb{R}^{|X|} \times \mathbb{N}^{|Y|} \to \mathbb{R}^p$. (We have $G = (g_1,...,g_m)$ and $H = (h_1,...,h_p)$.)

Given the system $S = (f, G, H, X, Y, \mathscr{D}(X), \mathscr{D}(Y))$, the constrained optimization problem consists in finding:

$$\min_{X \in \mathscr{D}(X), Y \in \mathscr{D}(Y)} f(X,Y) \;\; subject\ to \;\; G(X,Y) \leq 0 \wedge H(X,Y) = 0$$

where f denotes the objective function; G and H are inequality and equality constraints respectively.

We have used two constrained global optimizers for handling our wine blending problem: `Couenne` [5] and `IbexOpt` [24]. They compute a floating-point vector (X,Y) ε_{obj}-minimizing[2]:

$$f(X,Y) \; s.t. \; G(X,Y) \leq 0 \wedge (-\varepsilon_{eq} \leq H(X,Y) \leq +\varepsilon_{eq}).$$

Note that equalities $h_j(X,Y) = 0$ are relaxed by "thick" equations $h_j(X,Y) \in [-\varepsilon_{eq}, +\varepsilon_{eq}]$, i.e. two inequalities: $-\varepsilon_{eq} \leq h_j(X,Y) \leq +\varepsilon_{eq}$.

In our wine blending problem, ε_{eq} is set to `1e-1` in the equality constraints (2.i). This corresponds to 100 ml, i.e. less than 0.1% of the target volumes (at least 500 liters). This means that the volumes are computed with an approximation significantly better than the ineluctable errors made during the actual blending, i.e. the errors induced by measures and loss of residual matter during the wine transfer from a base to a target tank.

Interval/spatial B&B

The COP is handled by an interval *Branch & Contract & Bound* schema, called *spatial B&B* in `Couenne`. The process starts with an initial domain $D(x) \times D(y)$ that is recursively subdivided by a branching operator. The domains $D(x_i)$ are intervals of real values (bounded by floating-point numbers) for real variables x_i and the domains $D(y_j)$ are intervals of integers for integer variables y_j. Following the vocabulary used in interval analysis, we call **box** an n-dimensional domain made of $n = |X| + |Y|$ real and integer intervals.

The search tree of the B&B is traversed in best first order, where a node/box with a smallest minimum cost (i.e., a smallest *lower bound* as explained below) is selected first. The following operators are run at each node of the B&B:

Branch: A variable is chosen and its domain is split into two sub-domains. This makes the overall process combinatorial.

Contract: A filtering process *contracts* the studied box, i.e. improves the bounds of its intervals, without loss of solutions.

[2] ε_{obj}-minimize $f(X,Y)$ means minimize $f(X,Y)$ with a precision ε_{obj} on the objective, i.e. find (X,Y) such that for all Z_1, Z_2 we have $f(Z_1,Z_2) \geq f(X,Y) - \varepsilon_{obj}$.

Bound: Lower bounding guarantees that no feasible solution exists below a computed *lower bound* (of the optimum). The improvement of the lower bound is similar to a contraction, considering a cost variable corresponding to the objective cost is added in the system.

Improving the upper bound amounts to finding a good (although generally not the best) feasible point, so as to cut branches in the search tree with a higher cost.

The process starts with an initial box and ends when the difference between the upper and lower bounds reaches a given precision ε_{obj} or when all the explored nodes reach a size inferior to a given precision.

Algorithmic operators

Several algorithmic operators are used in the nodes of the B&B tree to reduce the search space and improve the lower bound of the objective function:

- **Constraint propagation.** The state-of-the-art `HC4` [6,21,5] continuous constraint propagation algorithm (HC stands for Hull-Consistency) is used to contract the handled box. The constraints are revised/handled one by one using an AC3-like progagation loop until a quasi-fixed point is obtained in terms of filtering. Each constraint is revised by a procedure called HC4-Revise that traverses twice the expression tree of the function. The first bottom-up phase evaluates the expression using *interval arithmetic* [15,17] (leaves are intervals); the second top-down phase uses inverse functions of the operators to filter the domains.[3]
- **Polyhedral convex relaxation.** Different strategies can produce a linear approximation of the system, i.e. of the numeric constraints and the objective function. The polytope obtained allows one to contract the box by calling a linear programming solver for improving the lower bound and/or the bounds of the n variable intervals. Let us mention three of these polyhedral convex relaxation strategies.
  Like `HC4`, Affine arithmetic [14,20] works with the expression tree of each function. Traversing the expression tree in a bottom-up and recursive way, for each mathematical operator, *affine arithmetic* combines the affine forms of the children/operands to build a new affine form. This interval algorithm finally builds two parallel hyperplanes for enclosing the feasible space of every constraint.
  A second and simple `X-Newton` strategy [2] uses a specific interval Taylor to produce a linear form for every function of the system.
  Finally, the most common polyhedral relaxation is based on the *reformulation* of the system (constraints and objective function). It can be achieved in a pre-processing step to help producing a polytope during search. The reformulation roughly amounts to introducing auxiliary variables for replacing nonlinear primitive operators. Hyper-planes are then computed during search (according to the current bound/domain constraints) to produce an outer approximation of the corresponding constraints. For instance, McCormick proposed in [19] to produce four specific hyper-planes defining a convex envelope of bilinear operators.
  Affine arithmetic and `X-Newton` are implemented in `IbexOpt` while a system reformulation strategy is implemented in `Couenne`.
- **Upperbounding.** Several types of algorithms can be used to find good feasible points possibly improving the upper bound. The most standard upperbounding algorithm uses

[3] Another interval constraint programming operator, called 3B in [18,25], is available in both solvers but is counterproductive in this application. It is based on a refutation reasoning that removes a sub-interval at a bound of a given domain if `HC4` can prove that the corresponding sub-problem contains no solution.

local optimization [5]. Local optimization is not called at every iteration/node in `Couenne` since it is costly. In `IbexOpt`, two original algorithms try to improve the upper bound by heuristically extracting an inner (entirely feasible) region that contains only solution points [3]. Roughly, the `InHC4` algorithm is a dual algorithm of `HC4` and `InnerPolytope` is a dual algorithm of `X-Newton`.

Using `IbexOpt` *or* `Couenne` *for wine blending*

`Couenne` and `IbexOpt` are open-source deterministic branch and bound global optimizers. `Couenne` is distributed on COIN-OR (COmputational INfrastructure for Operations Research), a project for the development of mathematical open-source softwares for the Operations Research community (see `www.coin-or.org/`). `IbexOpt` is implemented in `Ibex` (Interval Based EXplorer) and enriches this `C++` library devoted to interval solving [10,9].

`IbexOpt` is rigorous since all its operators are based on interval arithmetic and take into account roundoff errors. `Couenne` is not rigorous since some operators use interval arithmetics, but not all of them. Therefore it could sometimes miss the optimum. Also, because of our good command of the Ibex solver, we first implemented the wine blending solver in Ibex [26]. Unfortunately, Ibex is restricted to continuous variables for now, i.e. it cannot handle integer variables like the $P_{w,b}$ 0/1 variables. Therefore, to manage them, we encoded the 0/1 variables as real-valued variables $P_{w,b}$ of domain $[0,1]$. To ensure these variables take 0/1 values, and not a value inside the domain, we simply added the following quadratic constraints:

$$\forall w \in 1..\mathscr{W} \text{ and } \forall b \in 1..\mathscr{B},\ 4(P_{w,b} - \frac{1}{2})^2 = 1 \tag{15}$$

However, as shown in [26], `IbexOpt` gave good results on the first real instances we have (named `WineBlending1` and `WineBlending2` further), but did not scale so well on more difficult instances, i.e.:

- an artificial instance `WineBlending1+2` generated by mixing the two real instances (see Section 5.1),
- optimization processes launched in the prototypal interactive configuration tool (see Section 5.3), and
- a new series given by the company for checking how the solver scales up with the number of base wines (see Section 5.4).

That is why we have implemented a second wine blending tool using `Couenne` that turns out to be satisfactory.

5 Experiments

We modeled and solved several instances of wine assemblage. Section 5.1 details the results obtained on two instances given by Nyseos. We also report in Section 5.2 a validation of our approach in a real tasting session. Section 5.3 investigates whether our optimization algorithm could be used interactively, inside a configuration tool. Finally, Section 5.4 shows how our tool scales up with the number of base wines on a generic and difficult wine blending instance.

For the ε-optimization, we have required an accuracy ε_{obj} (optimum precision) below `1e-4`. The same precision is required for the solution (box) size: under this size, a box is not

studied (nor split) by the interval Branch & Bound. The optimum accuracy is smaller than the errors ε_a made by the chemical tools when they measure the $c_{b,a}$ aroma concentrations.

5.1 *Tests on first wine blending instances*

We first modeled a small and artificial instance of wine blending. It was used to rapidly adapt the COP model presented above until a rapid solving could be obtained. It contains 21 variables and was solved in a fraction of a second by `Couenne`.

The instance called `WineBlending1` is the first real instance provided by the Nyseos company. The instance consists in producing $\mathscr{W} = 2$ target wines from $\mathscr{B} = 7$ bases wines, taking into account $\mathscr{A} = 11$ aromas.

The global optimization problem, modeled as described in Section 3.4, contains 55 variables and 102 constraints:

- 2 volume (relaxed) channeling constraints,
- 7 base surplus constraints,
- 44 aroma concentration constraints,
- 49 constraints coming from the Min-Max encoding,

The second instance (`WineBlending2`) consists in assembling $\mathscr{W} = 3$ target wines from $\mathscr{B} = 6$ bases, taking into account $\mathscr{A} = 7$ aromas.

The Min-Max problem contains 64 variables and 100 constraints:

- 3 volume (relaxed) channeling constraints,
- 6 base surplus constraints,
- 42 aroma concentration constraints,
- 49 constraints coming from the Min-Max encoding,

We have also generated an artificial instance in order to check how our B&Bs scale up. The instance was generated by mixing the two real instances. It consists in assembling $\mathscr{W} = 5$ target wines from $\mathscr{B} = 13$ bases, taking into account $\mathscr{A} = 7$ aromas (those in common between both instances). It contains 176 variables and 169 constraints.

Performance results using `Couenne`

All the experiments have been run using `Couenne` on a `Dell Power Edge R610` computer (with a `3.46 Ghz X5690` processor, 6 cores, 36 Go of RAM). Note that the solver cannot exploit the multicore architecture.

To enhance the performance of `Couenne`, we removed the 3B filtering procedure from the B&B strategy (i.e., `aggressive_fbbt` was set to `no`).

The results obtained on the 3 instances are good, as shown in Table 1.

Observe that a solution with no error can be obtained in less than one minute in the three instances. This performance range is very satisfactory in that it would enable oenologists to choose between several blendings elaborated with different constraints.

	WineBlending1	WineBlending2	WineBlending1+2
CPU time (sec.)	3.81	3.25	31
#nodes (#iterations)	1007 (8829)	984 (11674)	6159 (84370)
Optimum	1e-8	1.5e-6	0

Table 1 Performance results obtained by the Couenne MINLP optimizer. The second line indicates the CPU time in second, the third line indicates the number of branching nodes (number of iterations), the last line gives the error obtained at the end: a value even less than the required precision 1e-4 means that a solution (with no error) has actually been found, as shown in Figure 4.

Performance results using IbexOpt

The performance results obtained by IbexOpt on the first two real instances are detailed in [26]. The article also details with which algorithmic features of IbexOpt the results were improved. The performance results are good on WineBlending1 and WineBlending2. Nevertheless, we obtained no satisfactory answer on the bigger instance WineBlending1+2 within the timeout. We found only some feasible points and the ε_{obj} precision on the optimum remained bad: the difference between the upper bound and the lower bound of the optimum obtained in 5 minutes was about 0.05 (recall that ε_{obj}=1e-4 is required). The IbexOpt strategy did not scale up well either to the configuration queries studied in Section 5.3 (decreasing the admissible errors $e_{w,a}$). For the first wine blending, the hardest instance ($e_{1,8}$) could not be solved within the timeout; the same is observed for the second wine blending in all the instances.

We finally analyzed whether these mitigate performance results came from the artificial encoding of Boolean variables or from the solver itself. We observed that WineBlending1+2 and the hard configuration queries cannot be handled either by Couenne when these instances are encoded as pure continuous systems.

These experiments strongly suggest to prefer a mixed discrete/continuous model for wine blending and encourage us to generalize Ibex and IbexOpt to mixed systems in a long-term. For the constraint propagation part, we should follow several ideas experimented in RealPaver [7]. To our knowledge, only one rigorous interval B&B, called IBBA [23], can handle integer variables, but this solver is not maintained anymore. Since several IBBA features are currently added to IbexOpt (by J. Ninin), we hope designing in a long-term an efficient and rigorous tool dedicated to wine blending.

The wines produced

The layout of the solution computed for WineBlending1 is shown in Figure 3.

The details of the aroma concentrations ($C_{w,a}$) in the target wines are illustrated by $\mathscr{W} = 2$ (resp. $\mathscr{W} = 3$) radar graphs for WineBlending1 (resp. WineBlending2) in Figure 4. The fact that the blue lines ($\hat{c}_{w,a}$) fall between the green lines ($\hat{c}_{w,a} - \varepsilon_a$ and $\hat{c}_{w,a} + \varepsilon_a$) highlights visually that the best solution has been obtained within ε_a tolerances, i.e. the total error is null.

5.2 *Tasting session*

An interesting (qualitative) validation of our tool was carried out in collaboration with an oenologist. He was asked by the Nyseos company to elaborate a (target) wine by blending

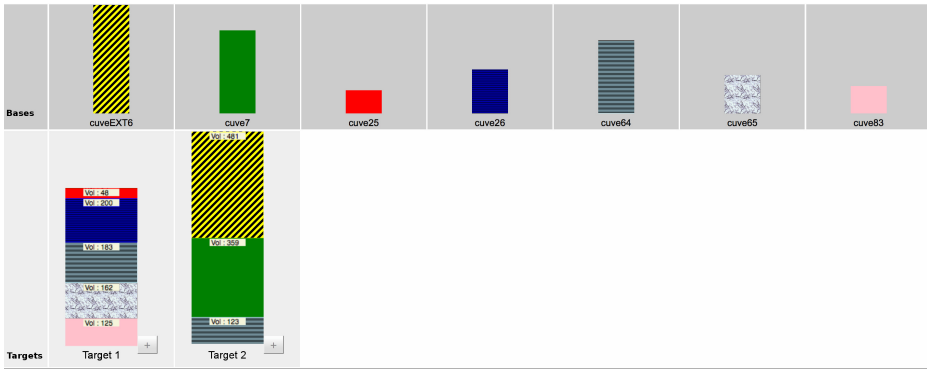


Fig. 3 Layout of the results for `WineBlending1`: volumes of the target wines (below) obtained by blending the bases (above).

several given base wines. Nyseos wrote down the volumes the oenologist selected for the assemblage and carried out a chemical analysis of the final blend to measure its aromatic criteria. Then, using our tool, Nyseos created a similar blend with the same base wines, but using eventually different volumes extracted from each base wine. Nyseos finally compared the blendings used to obtain the human-made wine with the computer-made wine and asked the oenologist to carry out a blind-test on the two wines.

As a result, *despite the blendings being significantly different, the oenologist could not distinguish between the two wines.*

This one experiment is of course far from being representative, but is nonetheless a promising indication of the relevance of our tool.

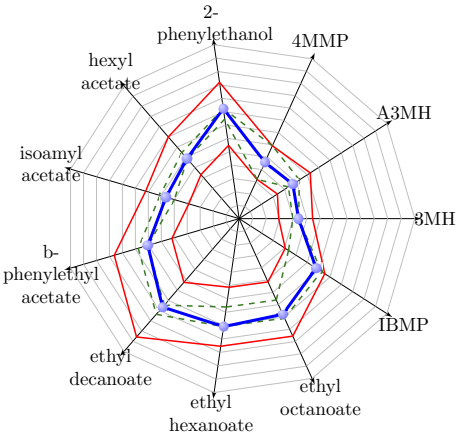
5.3 *Towards a configuration tool for wine assemblage*

To better fit the wishes of the client, we can imagine using our optimization algorithm interactively, inside a configuration tool. The user would be able to interact with the system via radar graphs corresponding to the different target wines, such as shown in Fig. 4. This would be particularly useful if a first optimization process failed in finding a perfect solution (with error E equal to 0).

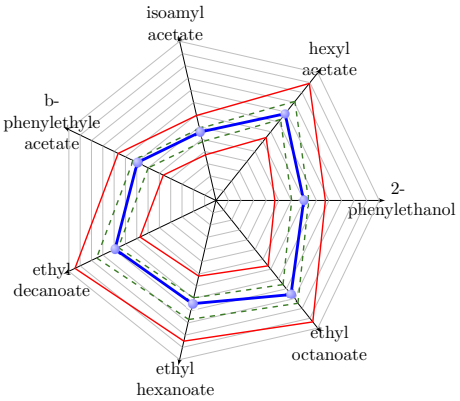
A way to modify the blending is to increase (or decrease) the importance (weight) Ω_w of a wine. A slider under each radar graph could for instance be used for this purpose. An optimization process could then recompute a new solution with this specification. Following the same idea, the user could modify the weight $\lambda_{w,a}$ of a given aroma in a wine (e.g., with a popup menu appearing when the mouse cursor position is on the corresponding axis of a radar graph), and the tool would run a new optimization.

Another interesting interaction would allow the tool to show to the client a new solution improving the concentration of an aroma a in a target wine w. More precisely, as shown in Figure 5, the tool could be able to compute new solutions minimizing any error $e_{w,a}$ such that:

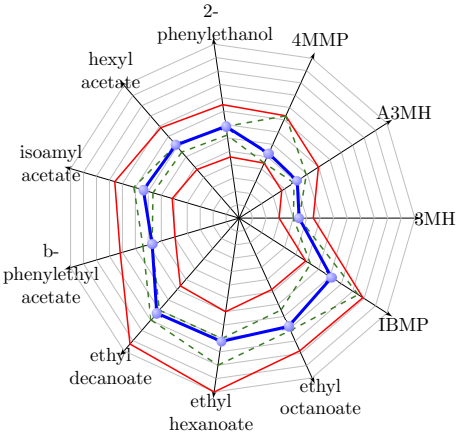
- the solution remains of course feasible, and
- the global error remains the same (or slightly worse) or,
  alternatively, the global error is not taken into account anymore.



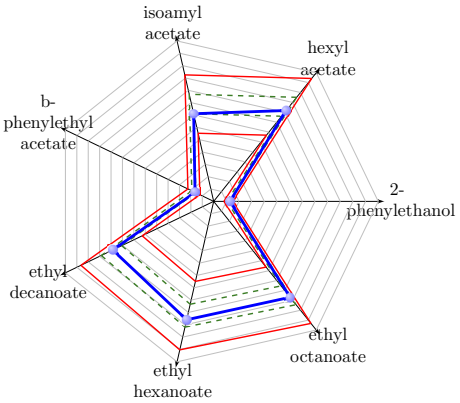
(a) WineBlending 1, target 1



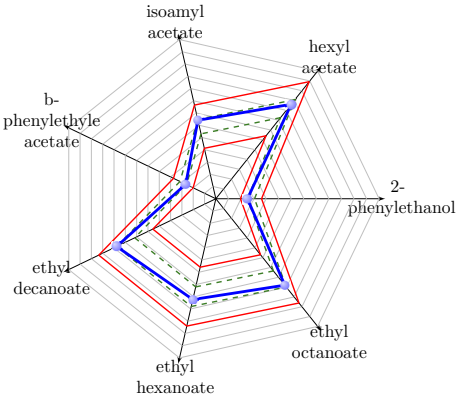
(b) WineBlending 2 target 1



(c) WineBlending 1, target 2



(d) WineBlending 2, target 2



(e) WineBlending 2, target 3

Fig. 4 Solutions obtained by our solver in instances `WineBlending1` (subfigures (a) and (c)) and `WineBlending2` (subfigures (b), (d), (e)). Every axis in a radar graph shows a computed aroma concentration $C_{w,a}$ (shown in thick line with balls), comprised within the imposed limits ($c^-_{w,a}$ and $c^+_{w,a}$) represented by solid lines, and as close as possible to the desired concentration $\hat{c}_{w,a}$. The 2 dashed curves represent the tolerances $\hat{c}_{w,a} - \varepsilon_a$ and $\hat{c}_{w,a} + \varepsilon_a$ on the desired concentration of aroma a in wine w.

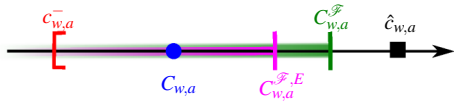


Fig. 5 Configuration tool: Given an optimized concentration $C_{w,a}$ obtained with global error E, we can compute the feasible concentrations $C^{\mathscr{F},E}_{w,a}$ and $C^{\mathscr{F}}_{w,a}$ which are the closest to $\hat{c}_{w,a}$. $C^{\mathscr{F}}_{w,a}$ is computed with any global error while $C^{\mathscr{F},E}_{w,a}$ respects the global error E.

We have carried out experiments to study the feasibility of such a configuration tool. The protocol was the following. First, we took the instances and modified them for finding a best solution entailing a global error. These harder instances were simply obtained by artificially decreasing the admissible measure errors ε_a related to the concentration of aroma a. Second, we stored the global error E obtained by optimization and injected it (slightly relaxing it) in new instances where the global error was a constraint to be fulfilled and the $e_{w,a}$'s became the new goal to be minimized. Third, we ran the new instances with the new goals and checked whether `Couenne` was able to find a new solution and measured the performance.

The results for the first wine blending are the following. The artificial harder instance is solved in 143 seconds, 34,788 nodes and 898,288 iterations for finding a global error E equal to 0.003239. A brief analysis shows that this global error comes from 7 of 11 $e_{w,a}$ error terms. Minimizing the 7 corresponding $e_{w,a}$ errors while keeping the global error below 0.00327 (i.e., computing $C^{\mathscr{F},E}_{w,a}$) requires the following CPU times (in second): 4.66, 5.32, 18.3, 30.2, 44.3, 65.1, 315. The performance is satisfactory except for the last minimization process of $e_{1,8}$ that exceeds five minutes.

The results for the second wine are the following. The artificial harder instance is solved in 48 seconds, 23,219 nodes and 987,677 iterations for finding a global error E equal to 0.003085. A brief analysis shows that this global error comes from 6 $e_{w,a}$ errors. Minimizing the 6 corresponding $e_{w,a}$ errors while keeping the global error below 0.0031 requires the following CPU times (in second): 0.42, 0.47, 1.26, 7.34, 48.2, 270. The performance is satisfactory except for the last minimization process of $e_{2,6}$ that exceeds four minutes.

Concluding remarks

It appears that the CPU times fall down to a fraction of a second if the constraint on the global error is removed (i.e., when computing $C^{\mathscr{F}}_{w,a}$). The same picture is observed if the global error is slightly more relaxed (more than 1%). This suggests to embed in our future configuration tool a relaxation percentage that would allow a reasonable computation of the alternative solutions. An empirical study on more instances must be conducted to select a "robust" relaxation percentage of E.

5.4 *Scaling issue with L1-norm and L2-norm*

The Nyseos company recently provided a generic wine blending instance producing 3 target wines by blending up to 20 base wines. Let us call this generic instance `WineBlending3`. The number of aromas that can be measured has also significantly increased up to 25. For the wineries, the number of target wines is not an issue and is typically one, two or three. On the contrary, increasing the number of base wines is very interesting for numerous wineries.

This generic instance allowed us to test how our optimization tool scales up. Based on this generic instance, we could generate instances with different numbers of wine bases: we tested 10, 15 and 20 bases.

Table 2 Performance results obtained with `Couenne` on the `WineBlending3` instance. The maximum error corresponds to the maximum value of an $e_{w,a}$ in the optimal solution.

	L1 norm		L2 norm	
# bases	cpu (sec.)	max error	cpu (sec.)	max error
10	10.8	0%	20.5	0%
15	48.3	16.1%	36.5	11.1%
20	19.4	18.3%	157	15.3%

Table 2 reports results obtained on 3 instances. They differ in the number of base wines. We also tested these instances using two different objective functions: the one described by (6), labeled L_1-norm in the table, and the variant where the errors $e_{w,a}$ (see (7)) are squared in the formula, minimizing a least square criterion for each target wine (L_2-norm).

Concluding remarks

The main conclusion is that our wine blending optimization tool seems to scale up well with the number of bases. Also, the least-square criterion used for a target wine seems more costly than the simple sum-based aggregation, but seems to lower the maximal error over aromatic concentrations. This suggests to compare these two criteria on other instances before choosing one of them for the tool.

6 Conclusion

We have reported in this paper a first attempt to handle the wine assemblage problem with constraint programming and mathematical programming techniques. This approach helps the oenologist to blend a *set* of target wines while limiting taste saturation.

Automatic wine blending can be modeled as a constrained optimization problem accepting disjunctive constraints that are critical in practice. These constraints ensure that a minimal amount of wines is transferred from a base tank to a target wine tank. We have resorted to mono-criterion optimization and worked to obtain a model with no max and no absolute value operators.

The `Couenne` interval/spatial B&B has allowed us to find in reasonable CPU time the best solution to several real and realistic instances given by the Nyseos company. In particular, we have shown on a recent generic instance that our constrained optimization tool scales up well with the number of base wines. A tasting session carried out by an oenologist has qualitatively validated our approach.

Other encouraging results suggest the possibility to use our approach within an interactive configuration tool dedicated to wine assemblage.

References

1. IBM ILOG CPLEX Optimization Studio V12.6.0 documentation (2015)

2. Araya, I., Trombettoni, G., Neveu, B.: A Contractor Based on Convex Interval Taylor. In: Proc. CPAIOR, pp. 1–16. LNCS 7298 (2012)
3. Araya, I., Trombettoni, G., Neveu, B., Chabert, G.: Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. J. Global Optimization (JOGO) p. 22 pages (2014). To appear, online in 2014
4. Belotti, P.: Couenne, a user's manual (2013). `www.coin-or.org/Couenne/`
5. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branch and Bounds Tightening Techniques for Non-convex MINLP. Optimization Methods and Software **24**(4–5), 597–634 (2009)
6. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising Hull and Box Consistency. In: Proc. ICLP, pp. 230–244 (1999)
7. Berger, N., Granvilliers, L.: Some Interval Approximation Techniques for MINLP. In: Proc. SARA, pp. 26–33 (2009)
8. Bouveret, S., Lemaitre, M.: Computing leximin-optimal solutions in constraint networks. Artificial Intelligence J. **173**, 343–364 (2009)
9. Chabert, G.: Interval-Based EXplorer (2013). `www.ibex-lib.org`
10. Chabert, G., Jaulin, L.: Contractor Programming. Artificial Intelligence **173**, 1079–1100 (2009)
11. Dagan, L.: Potentiel aromatique des raisins de Vitis vinifera L. cv. Petit Manseng et Gros Manseng. Contribution à l'arôme des vins de pays Côtes de Gascogne. Ph.D. thesis, École nationale supérieure agronomique (Montpellier) (2006)
12. Datta, S., Nakai, S.: Computer-aided optimization of wine blending. Journal of Food Science **57**(1), 178–182 (1992). DOI 10.1111/j.1365-2621.1992.tb05450.x. URL http://dx.doi.org/10.1111/j.1365-2621.1992.tb05450.x
13. Ferrier, J.G., Block, D.E.: Neural-network-assisted optimization of wine blending based on sensory analysis. American Journal of Enology and Viticulture **52**(4), 386–395 (2001). URL http://www.ajevonline.org/content/52/4/386.abstract
14. de Figueiredo, L., Stolfi, J.: Affine Arithmetic: Concepts and Applications. Numerical Algorithms **37**(1–4), 147–158 (2004)
15. Knuppel, O.: Bias/profil: A fast interval library. Computing **53**, 277–287 (1994)
16. Koak, J.H., Kang, B.S., Hahm, Y.T., Park, C.S., Baik, M.Y., Y., K.B.: Blending of Different Domestic Grape Wines using Mixture Design and Optimization Technique. Food science and biotechnology **19**(4), 1011–1018 (2010)
17. Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W., Krämer, W.: filib++, a Fast Interval Library Supporting Containment Computations. ACM TOMS **32**(2), 299–324 (2006)
18. Lhomme, O.: Consistency Techniques for Numeric CSPs. In: IJCAI, pp. 232–238 (1993)
19. McCormick, G.: Computability of Global Solutions to Factorable Nonconvex Programs - part 1 - Convex Underestimating Problems. Mathematical Programming **10**, 147–175 (1976)
20. Messine, F., , Laganouelle, J.L.: Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. Journal of Universal Computer Science **4**(6), 589–603 (1998)
21. Messine, F.: Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes. Ph.D. thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse (1997)
22. Moore, D.B., Griffin, T.G.: Computer blending technology. American Journal of Enology and Viticulture **29**(1), 50–53 (1978). URL http://www.ajevonline.org/content/29/1/50.abstract
23. Ninin, J., Messine, F., Hansen, P.: A Reliable Affine Relaxation Method for Global Optimization. Tech. Rep. RT-APO-10-05, IRIT (2010). URL ftp://ftp.irit.fr/IRIT/APO/RT-APO-10-05.pdf
24. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner Regions and Interval Linearizations for Global Optimization. In: AAAI, pp. 99–104 (2011)
25. Trombettoni, G., Chabert, G.: Constructive Interval Disjunction. In: Proc. CP, LNCS 4741, pp. 635–650 (2007)
26. Vismara, P., Coletta, R., Trombettoni, G.: Constrained Wine Blending. In: Proc. CP, Constraint Programming, LNCS 8124, pp. 864–879. Springer (2013)