# Constrained Wine Blending

Philippe Vismara[1,2], Remi Coletta[1], and Gilles Trombettoni[1]

[1] LIRMM, UMR5506 Université Montpellier II - CNRS, Montpellier, France
{Philippe.Vismara, Remi.Coletta, Gilles.Trombettoni}@lirmm.fr
[2] MISTEA, UMR0729 Montpellier SupAgro - INRA, Montpellier, France

**Abstract.** Assemblage consists in blending base wines in order to create a target wine. 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 non linear constrained Min-Max problem (minimizing the gap to the desired concentrations for every aromatic criterion) efficiently handled by the `Ibex` interval branch and bound.

## 1 Introduction

Assemblage is the subtle blending of 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`), 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 [6]. 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 [11]. 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 [8]. 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 and nonlinear constraints.

An interesting algorithmic research on wine blending has been presented in [8]. An artificial neural network approach has been 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 preferred to resort to monocriterion optimization. In addition, no performance (CPU time) results are shown in [8]. Another research dealt with the blending problem [7]. 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. The problem is modeled by a mixed (discrete and continuous) nonlinear program. We transform it into a non-linear (pure) continuous CSP handled by a rigorous interval Branch and Bound (B&B). We have built a constrained optimization model for minimizing in each target wine the gap between desired aromatic concentrations and obtained concentrations, while taking into account the minimal transfer disjunctive constraint. Absolute value and max operators have been removed from the obtained system.

## 2 The wine assemblage problem

Figure 1 illustrates the definition of wine assemblage.

We consider a set of base wines numbered from 1 to $\mathcal{B}$. We denote by $\text{vol}_b$ the volume of the base $b \in 1..\mathcal{B}$.

For different reasons, it is sometimes impossible to completely empty a tank. Let $s_b^-$ be the minimum volume that must remain in tank $b$. (We have: $0 \leq s_b^- \leq \text{vol}_b$.) All base wines are analyzed in order to measure the concentration of selected key aroma compounds. These compounds are numbered from 1 to $\mathcal{A}$. 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. Hence, we consider a set of target wines, numbered from 1 to $\mathcal{W}$. For each wine $w$, we aim to produce an optimal volume $\hat{\text{vol}}_w$. The final volume $V_w$ of wine $w$ should be as close as possible to $\hat{\text{vol}}_w$ and must remain greater (resp. smaller) than a given lower bound $\text{vol}_w^-$ (resp. an upper bound $\text{vol}_w^+$), i.e.:

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

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 has been pumped from base tank $b$.

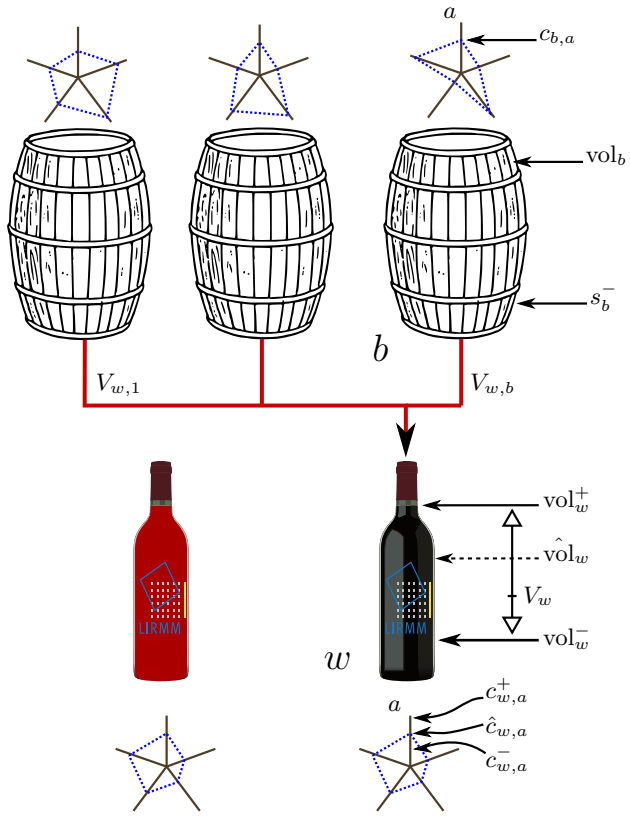


**Fig. 1.** Wine assemblage

We have a direct relation with $V_w$:

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

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

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

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

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

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 $C_{w,a}$ concentrations 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:

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

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 MINLP formulation for wine blending

We can model the wine blending problem as a mixed nonlinear program (MINLP). We show in Section 4 how to straightforwardly transform the MINLP into a numerical CSP (NCSP) handled by interval methods. This explains why the bound constraints are directly modeled below by bounded domains, i.e., intervals.

### 3.1 Variables

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

- a 0/1 variable $P_{w,b}$ and
- a variable $V'_{w,b}$ with a domain $D(V'_{w,b}) = [\delta_V, \min(\mathrm{vol}^+_w, \mathrm{vol}_b)]$ representing the volume coming from the base $b$ in the wine $w$.
  (We have: $V_{w,b} \equiv P_{w,b}\,.\,V'_{w,b}$. The introduction of these 0/1 variables of course avoids an explicit definition of the disjunctive constraint (4).)

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

### 3.2 Constraints

The system of constraints of our MINLP is described below.

- The channeling constraint (2) becomes:

$$\forall w,\ V_w - \sum_{b=1}^{\mathcal{B}}(P_{w,b}\,.\,V'_{w,b}) = 0 \tag{2.i}$$

– The surplus constraint (3) also remains similar:

$$\forall b \in 1..\mathcal{B}, s_b^- \leq \text{vol}_b - \sum_{w=1}^{\mathcal{W}} (P_{w,b} \,.\, V'_{w,b}) \tag{3.i}$$

– To enhance the performance results, we have added a constraint redundant to (3.i). This constraint simply ensures that the sum of the volumes of target wines is inferior to the sum of the base volumes:

$$\sum_{w=1}^{\mathcal{W}} V_w \leq \sum_{b=1}^{\mathcal{B}} \text{vol}_b \tag{6}$$

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

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

hence:

$$0 \leq \sum_{b=1}^{\mathcal{B}} P_{w,b} \,.\, V'_{w,b} \,.\, (c_{b,a} - c_{w,a}^-) \tag{5.i -}$$

and for the upper bound:

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

hence:

$$0 \leq \sum_{b=1}^{\mathcal{B}} P_{w,b} \,.\, V'_{w,b} \,.\, (c_{w,a}^+ - c_{b,a}) \tag{5.i +}$$

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

In this application, the significant criterion is wine quality. Nevertheless, we could extend the definition of $E$ below to errors on alcohol content or price. Bear in mind that a target wine is defined by a set of desired concentrations $\hat{c}_{w,a}$ for each of its aroma. Therefore, a way to optimize the quality of the target wines is to minimize a weighted sum of differences between the concentrations desired $\hat{c}_{w,a}$ and the concentrations obtained $C_{w,a}$ (see (5)). Furthermore, we want to minimize the maximal error on the set of target wines:

$$\max_{w \in 1..\mathcal{W}} \Omega_w (\lambda_{vol_w} \,.\, e_{vol_w} + \sum_{a \in 1..\mathcal{A}} (\lambda_{w,a} \,.\, e_{w,a})) \tag{7}$$

where:

- $\Omega_w$ is a parameter reflecting how important is a given wine $w$ ($\Omega_w$ 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 respect of the volume requirement of wine $w$ compared to the satisfaction of aroma concentrations. For a given target wine $w$, we assume that $\lambda_{vol_w} + \sum_{a \in 1..\mathcal{A}} \lambda_{w,a} = 1$.

All the parameters, including the aroma concentrations, are measured with a given uncertainty. $\varepsilon_a$ 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 $\varepsilon_a$. 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:

$$\boldsymbol{e_{w,a}} = \max\left(\frac{|C_{w,a} - \hat{c}_{w,a}|}{\hat{c}_{w,a}} - \varepsilon_a, 0\right) \tag{8}$$

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:

$$\boldsymbol{e_{vol_w}} = \max\left(\frac{\hat{\text{vol}}_w - V_w}{\hat{\text{vol}}_w} - \varepsilon_{vol}, 0\right) \tag{9}$$

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 $\text{vol}_w^+$ and the target $\hat{\text{vol}}_w$. This is illustrated by Figure 2.

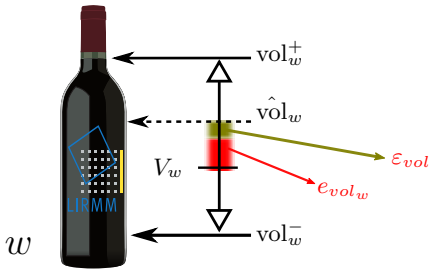


**Fig. 2.** Visualization of the gap $e_{vol_w}$

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:

$$\forall w \in 1..\mathcal{W}, \quad \Omega_w(e_{vol_w}.\lambda_{vol_w} + \sum_{a \in 1..\mathcal{A}} (e_{w,a} \,.\, \lambda_{w,a})) \leq E \tag{10}$$

**Removing max and absolute value operators**

In order to increase 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 (8). Indeed, $\lambda_{w,a}$ is positive so that minimizing $E$ entails minimizing every variable $e_{w,a}$. Hence:

$$\forall w \in 1..\mathcal{W}, \forall a \in 1..\mathcal{A}, \quad ((e_{w,a} + \varepsilon_a)\, \hat{c}_{w,a} \geq |C_{w,a} - \hat{c}_{w,a}|) \;\wedge\; (e_{w,a} \geq 0) \tag{11}$$

The same simplification can be applied to (9), as follows:

$$\forall w \in 1..\mathcal{W}, \quad ((e_{vol_w} + \varepsilon_{vol})\, \hat{\text{vol}}_w \geq (\hat{\text{vol}}_w - V_w)) \;\wedge\; (e_{vol_w} \geq 0) \tag{12}$$

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 (11). Indeed, remember that every variable $e_{w,a}$ must be minimized and observe that $\hat{c}_{w,a}$ is positive. Thus,
$\forall w \in 1..\mathcal{W}, \forall a \in 1..\mathcal{A}$,

$$(e_{w,a} + \varepsilon_a)\, \hat{c}_{w,a} \geq \frac{1}{V_w}.\sum_{b=1}^{\mathcal{B}}(P_{w,b}.V'_{w,b}.c_{b,a}) - \hat{c}_{w,a}$$

$$(e_{w,a} + \varepsilon_a)\, \hat{c}_{w,a} \geq -\frac{1}{V_w}.\sum_{b=1}^{\mathcal{B}}(P_{w,b}.V'_{w,b}.c_{b,a}) + \hat{c}_{w,a}$$

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

$$e_{w,a} \geq 0 \tag{13}$$

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

$$V_w.(e_{w,a} + \varepsilon_a - 1)\,.\,\hat{c}_{w,a} + \sum_{b=1}^{\mathcal{B}}(P_{w,b}.V'_{w,b}.c_{b,a}) \geq 0 \tag{15}$$

As a result, we have succeeded in suppressing from our initial model all the absolute value and max operators. Although our interval nonlinear constraint solver can handle these operators, the performance is thus increased and the simplified model can also be implemented in other 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 Min-Max: one variable $E \in [0, +\infty]$, $\mathcal{W}.\mathcal{A}$ variables $e_{w,a} \in [0, 1]$ (that absorb the unary constraints (13)) and $\mathcal{W}$ variables $e_{vol_w} \in [0, 1]$ that absorb the unary constraints of (12).

In addition to the constraints (2.i), (3.i), (6), (5.i-), (5.i+) defined in Section 3.2, we define new constraints for the Min-Max: (10), (12), (14), (15). The objective function simply consists in minimizing the value of the variable $E$.

## 4 Solving the MINLP with an interval B&B

We wanted a free solver in order to embed it in the final dedicated tool for Nyseos. In addition, the MINLP detailed above could be handled by any MINLP solver such as Baron [13] or Couenne [2], but all of them are *not* rigorous (safe). This means that they sometimes miss the best solution due to round-off errors related to floating-point arithmetic. It is known that cases where the best solution is missed by unsafe solvers are rare but do occur in practice. Using a safe optimizer was reassuring for Nyseos. Furthermore, since:

- our modeling of the wine blending problem contains only one type of 0/1 variables,
- the interval solver `Ibex` features the very efficient `IbexOpt` interval B&B [14],
- the authors have a good command of `Ibex` [5, 4],

we decided to simply encode the MINLP problem as an NCSP, i.e., a standard *continuous* system of nonlinear constraints (i.e., over the real numbers). To do so, the 0/1 variables are encoded by real-valued variables $P'_{w,b}$ of domain $[0, 1]$. To ensure these variables take 0/1 values, we simply add the following quadratic constraints:

$$\forall w \in 1..\mathcal{W} \text{ and } \forall b \in 1..\mathcal{B},\ 4(P_{w,b} - \frac{1}{2})^2 = 1 \tag{16}$$

This means that the initial disjunctive constraints that produce mixed constraints in the MINLP model are handled by continuous quadratic constraints.

Our good command of the interval solver `Ibex` enabled to produce an efficient strategy, but it would be interesting to compare in a future work our interval B&B with a MINLP solver like Couenne [2]. Quadratic solvers are not our first alternative choice because our model will probably be extended with other nonlinear (and non quadratic) constraints about color or wine varieties. To our knowledge, only one rigorous interval B&B, called `IBBA` [12], is endowed with a simple mechanism handling integral variables. `IBBA` could be compared with Ibex, although the two solvers are merging.

### 4.1 Constrained global optimization with an interval B&B

A continuous constrained global optimization problem is defined as follows.

**Definition 1 (Constrained global optimization)**
*Consider a vector of variables $x = (x_1, ..., x_n)$ varying in a domain $[x] = [x_1] \times \cdots \times [x_n]$, a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$, vector-valued functions $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^p$. We have $g = (g_1, ..., g_m)$ and $h = (h_1, ..., h_p)$.*

*Given the system $S = (f, g, h, [x])$, the constrained global optimization problem consists in finding:*

$$\min_{x \in [x]} f(x) \quad \textit{subject to} \quad g(x) \leq 0 \wedge h(x) = 0.$$

*$f$ denotes the objective function; $g$ and $h$ are inequality and equality constraints respectively.*

Our `IbexOpt` constrained global optimizer [14] computes a floating-point vector $x$ $\epsilon$-minimizing[1]:

$$f(x) \ s.t. \ g(x) \leq 0 \wedge (-\epsilon_{eq} \leq h(x) \leq +\epsilon_{eq}).$$

Note that equalities $h_j(x) = 0$ are relaxed by "thick" equations $h_j(x) \in [-\epsilon_{eq}, +\epsilon_{eq}]$, i.e., two inequalities: $-\epsilon_{eq} \leq h_j(x) \leq +\epsilon_{eq}$. `IbexOpt` guarantees the global optimum of the relaxed system, although $\epsilon_{eq}$ can often be chosen almost arbitrarily small. (Most of the global optimizers like `Baron` [13] or `Couenne` [2] cannot offer any guarantee.)

In our wine blending problem, we set a constant $\epsilon^1_{eq}$ equal to `1e-4` in the equality constraints (16). Another $\epsilon^2_{eq}$ is set to `1e-1` in the equality constraints (2.i). This corresponds to 1 dl (deciliter), 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.

[1] $\epsilon$-minimize $f(x)$ means minimize $f(x)$ with a precision $\epsilon_{obj}$ on the objective, i.e., find $x$ such that for all $y$, we have $f(y) \geq f(x) - \epsilon_{obj}$.

### 4.2 Algorithmic features of Ibex

`IbexOpt` is implemented in `Ibex` (Interval Based EXplorer) and enriches this `C++` library devoted to interval solving [5].

`IbexOpt` [14] follows an interval *Branch & Contract & Bound* schema. The process starts with an initial box $[x]$ that is recursively subdivided by a branching operator. The tree is traversed in best first search, in which a box with a smallest minimum cost is selected first. `IbexOpt` applies the following operators at each node (box) of the B&B:

**Branch:** A variable $x_i$ is chosen and its interval $[x_i]$ is split into two sub-boxes.

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

**Bound:** The improvement of the lower bound is similar to a contraction (considering an additional variable corresponding to the objective cost). The lower bound guarantees that no feasible solution exists lower. Improving the upper bound amounts in 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 $[x]$ and ends when the difference between the upper and lower bounds reaches a given precision $\epsilon_{obj}$ or when all the explored nodes reach a size inferior to a given precision.

At each node of the B&B, `IbexOpt` is called with efficient operators for reducing the search space and improving the lower bound of the objective function:

- The state-of-the-art `HC4` [3, 10] (continuous) constraint propagation algorithm is first used to contract the handled box.
- The operator `X-Newton` uses a specific interval Taylor to convexify the search space, contract the box and improve the lower bound [1].
- Two original algorithms are used to improve the upper bound by heuristically extracting an inner (entirely feasible) region that contains only solution points. This explains why equations are slightly relaxed. Roughly, the `InHC4` algorithm is a dual algorithm of `HC4` and `InnerPolytope` is a dual algorithm of `X-Newton`.

The default optimizer uses, as bisection heuristic, the `SmearSumRel` variant of Kearfott's branching heuristic using the *Smear* function [9]. The `SmearSumRel` and `SmearMaxRel` branching heuristics are described in [14].

## 5 Experiments on first instances

We have modeled and solved several instances of wine assemblage. We also report in Section 5.5 a validation of our approach during a real tasting session.

For the $\epsilon$-optimization, we have always required an accuracy $\epsilon_{obj}$ (goal precision) below `1e-4`. The same precision is required for the solution (box) size: under this size, a box is not studied (and split) by the interval Branch & Bound. The goal accuracy is better than the errors $\epsilon_a$ made by the chemical tools when they measure the aroma concentrations (e.g, for the $c_{b,a}$'s).

### 5.1 Wine blending instances

We have modeled three instances of wine assemblage. The first one (`WineBlending0`) is a small and artificial instance. It was used to rapidly adapt the MINLP/NCSP model presented above until a rapid solving could be obtained. It contains 21 variables. `WineBlending0` is solved in 0.18 seconds and only 6 branching nodes, independently from the $\epsilon_{obj}$ goal precision (`1e-4` or `1e-8`).

#### Real instance 1

The second instance (`WineBlending1`) is a real instance provided by the Nyseos company. The instance consists in producing $\mathcal{W} = 2$ target wines from $\mathcal{B} = 7$ bases wines, taking into account $\mathcal{A} = 11$ aroma.

The Min-Max problem, modeled as described in Section 3.4, contains 55 variables and 116 constraints:

- 2 volume (relaxed) channeling constraints,
- 7 base surplus constraints,
- 44 aroma concentration constraints,
- 49 constraints coming from the Min-Max encoding,
- 14 quadratic constraints modelling the disjunctive realistic volume constraints.

#### Real instance 2

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

The Min-Max problem contains 64 variables and 118 constraints:

- 3 volume (relaxed) channeling constraints,
- 6 base surplus constraints,
- 42 aroma concentration constraints,
- 49 constraints coming from the Min-Max encoding,
- 18 quadratic constraints modeling the disjunctive realistic volume constraints.

### 5.2 Results obtained by the default optimizer

All the results reported in this paper have been obtained on a 2010 MacBook laptop with a 2.4 GHz Intel Core 2 Duo process.
We have first run the default optimizer of `Ibex` with a solution and goal precisions set to 1e-4 and with a timeout set to 5 minutes.

The optimizer reaches the timeout for `WineBlending1` and `WineBlending2` although rather good solutions are computed:

- In 5 min and 7894 branching nodes, an accuracy 0.002 is obtained on the goal (i.e., the maximum error on $E$) for `WineBlending1`.
- In 5 min and 8520 branching nodes, an accuracy 0.0054 is obtained on the goal for `WineBlending2`.

### 5.3 Algorithmic analysis and improvements

The results above have been obtained with the by-default constrained optimization strategy available in `Ibex` and briefly described in Section 4.2.

We have first analyzed which of the default features have an impact on performance and which ones do not. This analysis was fruitful.

Concerning the contraction/filtering part, the interval linearization operator `X-Newton` is surprisingly counterproductive. On the contrary, all the contraction is performed by the `HC4` constraint programming operator. Since the wine blending model built is mainly linear, this result is counterintuitive.

The opposite and more intuitive observation has been made on the inner regions extracted for improving the upperbound. The `InHC4` operator, issued from constraint programming principles (working constraint per constraint), appeared to be useless. On the contrary, the `InnerPolytope` algorithm (dual of *X-Newton*) that can extract an inner polytope from the feasible region is crucial. Without this feature, we could not obtain any answer from the optimizer.

| | | `WineBlending1` | `WineBlending2` |
|---|---|---|---|
| `SmearSum` | cputime (sec.) | > 300 | 57 |
| | precision | 0.00014 | 1e-10 |
| | #nodes | 21880 | 4146 |
| `SmearSumRel` | cputime (sec.) | > 300 | > 300 |
| | precision | 0.0018 | 0.00017 |
| | #nodes | 25864 | 24270 |
| `SmearMax` | cputime (sec.) | > 300 | 111 |
| | precision | 0.0013 | 1e-10 |
| | #nodes | 25864 | 8851 |
| `SmearMaxRel` | cputime (sec.) | 0.35 | 27.4 |
| | precision | 1e-10 | 1e-10 |
| | #nodes | 23 | 2048 |

**Table 1.** Comparison between the Smear-based branching heuristics. An entry is a multiline containing 3 information about the results obtained by a given branching strategy on a given instance. The first line of a multiline contains the CPU time, with a timeout set to 5 min; the second line gives the error obtained at the end (`1e-10` means that the last simplex call achieved by `InnerPolytope` finds a solution with no error (rounded to `1e-10`)); the third line reports the number of branching nodes.

A second feature has a major impact on performance: the branching (bisection) strategy. We have observed that the basic bisection heuristics available in `Ibex` are inefficient or show a poor performance. The `largestFirst` heuristic, selecting a variable with the largest width, prevents the B&B from finding any feasible point during the tree search. The `roundRobin` also generally shows a poor performance, except if we rearrange statically the variables as preconised by a first analysis reported below. Only the Smear-based strategies behave well:

the historical `SmearMax` and `SmearSum` heuristics [9]; the `SmearSumRel` variant called in the default optimization [14], and the `SmearMaxRel` variant that plays a fondamental role in our wine blending problem.

Based on this analysis, we have designed a dedicated strategy. The `X-Newton` operator has been first disconnected, thus bringing a speedup of a factor 4 on the two real instances. Second, we have compared the performance of the four Smear heuristic variants mentioned above. Table 1 gathers the results obtained.

We remark that `SmearSum` and, in particular, `SmearMaxRel`, show a good performance. Therefore `SmearMaxRel` has been chosen for our wine blending dedicated strategy.

We have also experimentally analyzed which variables are selected by the optimization process in slow and fast runs. We have empirically learnt that the CSP variables $P_{w,b}$ and $V'_{w,b}$ should often be selected, whereas the variables added for the Min-Max optimization should be rarely chosen. This study has led us to a dedicated heuristic that offers (only) the same performance as `SmearMaxRel` (23 seconds versus 27 seconds on `WineBlending2`). Of course, we need more instances to better learn from the data.

### 5.4 The wines produced

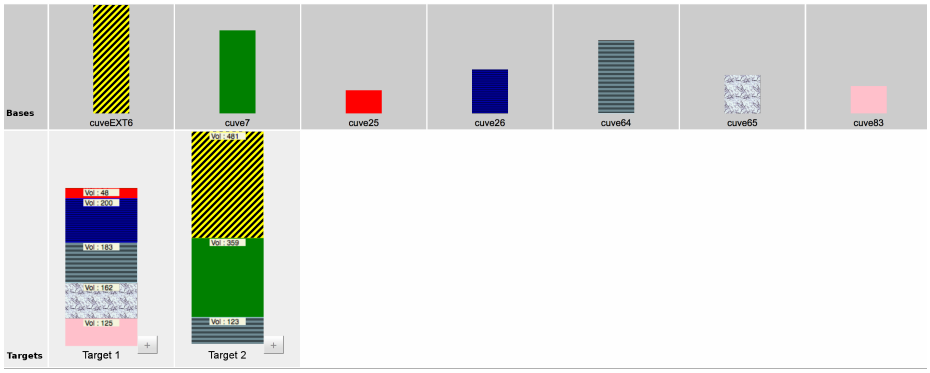


**Fig. 3.** Layout of the results for `WineBlending1`: volumes of the target wines (below) obtained by blending the bases (above)

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 $\mathcal{W} = 2$ radar graphs 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 $\varepsilon_a$ tolerances.

The details of the aroma concentrations ($C_{w,a}$) in the target wines for `WineBlending2` are illustrated by $\mathcal{W} = 3$ radar graphs in Figure 5.

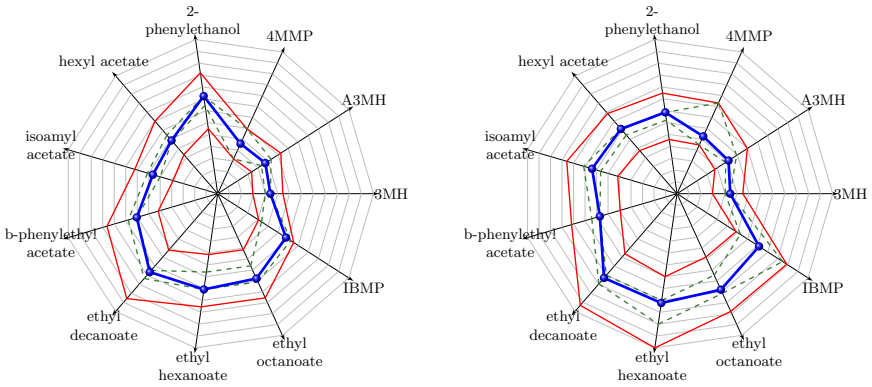


**Fig. 4.** Solution obtained by our Min-Max model on the two wines elaborated in instance `WineBlending1`. Every axis in a radar graph shows a computed aroma concentration $C_{w,a}$ (shown in thick line with balls), comprised within the imposed limits ($c^-_{w,a}$ and $c^+_{w,a}$) represented by solid 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 $w$.

### 5.5 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 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.

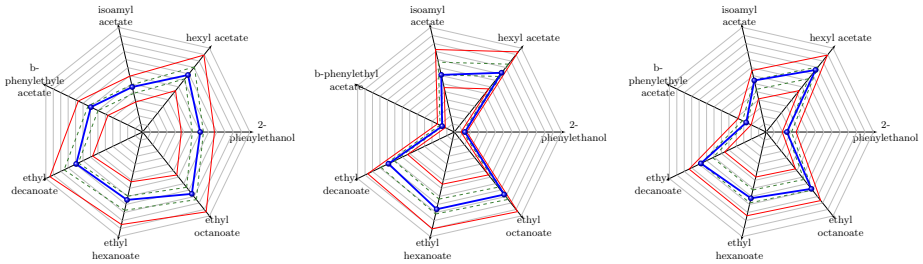


**Fig. 5.** Radar graphs obtained for the instance `WineBlending2`.

The results were as follows: 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.

### 5.6 A future configuration tool for wine assemblage

The first experimental results are preliminary and were obtained on only two real instances. However, they suggest that the current strategy, maybe endowed with a dedicated branching heuristic, can handle efficiently most of the instances useful in practice. Therefore, 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 and Fig. 5.

A way to modify the blending is to increase (or decrease) the importance (weight) $\Omega_w$ 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.

A last and more intrusive possibility is to allow the client to strengthen the maximum admissible concentration $c^+_{w,a}$ (or minimum admissible concentration $c^-_{w,a}$) of aroma $a$ in wine $w$. The client would simply select a bound and our tool would run two optimizations providing two information (assuming $c^+_{w,a}$ is selected). First, the smallest value of concentration $C_{w,a}$ for which there is a solution that respects all the constraints (such a solution is not necessarily optimal for the maximum error $E$). Second, we can also compute the smallest feasible value of $C_{w,a}$ that does not increase the maximum error $E$. These two bounds can be displayed as outstanding values for $c^+_{w,a}$.

## 6 Conclusion

We have reported in this paper a first attempt to handle the wine assemblage problem with constraint programming techniques. The problem can be modeled as a MINLP or a numerical CSP able to define disjunctive constraints that are critical in practice. These constraints ensure that a minimal amount of wines is transferred from a base to target wine. We have resorted to mono-criterion optimization and have worked to obtain a model with no max and no absolute value operators. We have designed an optimization strategy dedicated to wine blending that allows us to find in second the best solution to two real instances given by the Nyseos company (with no error in the volumes or the concentration requirements). A tasting session carried out by an oenologist has qualitatively validated our approach. These encouraging results offer the possibility to use our approach within an interactive configuration tool dedicated to wine assemblage.

## References


1. I. Araya, G. Trombettoni, and B. Neveu. A Contractor Based on Convex Interval Taylor. In *Proc. CPAIOR*, pages 1–16. LNCS 7298, 2012.
2. P. Belotti. Couenne, a user's manual, 2013. `www.coin-or.org/Couenne/`.
3. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.
4. G. Chabert. Interval-Based EXplorer, 2013. `www.ibex-lib.org`.
5. G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
6. L. Dagan. *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*. PhD thesis, École nationale supérieure agronomique (Montpellier), 2006.
7. S. Datta and S. Nakai. Computer-aided optimization of wine blending. *Journal of Food Science*, 57(1):178–182, 1992.
8. J. G. Ferrier and D. E. Block. Neural-network-assisted optimization of wine blending based on sensory analysis. *American Journal of Enology and Viticulture*, 52(4):386–395, 2001.
9. R.B. Kearfott and M. Novoa III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2):152–157, 1990.
10. F. Messine. *Méthodes d'Optimisation Globale basées sur l'Analyse d'Intervalle pour la Résolution des Problèmes avec Contraintes*. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.
11. D. B. Moore and T. G. Griffin. Computer blending technology. *American Journal of Enology and Viticulture*, 29(1):50–53, 1978.
12. J. Ninin, F. Messine, and P. Hansen. A Reliable Affine Relaxation Method for Global Optimization. Technical Report RT-APO-10-05, IRIT, 2010.
13. M. Tawarmalani and N. V. Sahinidis. A Polyhedral Branch-and-Cut Approach to Global Optimization. *Mathematical Programming*, 103(2):225–249, 2005.
14. G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *AAAI*, pages 99–104, 2011.