

A threaded approach of the quadratic bi-blending algorithm

Juan F.R. Herrera · Leocadio G. Casado ·
Eligius M.T. Hendrix · Inmaculada García

Published online: 15 May 2012

© The Author(s) 2012. This article is published with open access at Springerlink.com

Abstract Blending algorithms aim for solving the problem of determining the mixture of raw materials in order to obtain a cheap and feasible recipe with the smallest number of raw materials. An algorithm that solves this problem for two products, where available raw material is limited, has two phases. The first phase is a simplicial branch-and-bound algorithm which determines, for a given precision, a Pareto set of solutions of the bi-blending problem as well as a subspace of the initial space where better feasible solutions (with more precision) can be found. The second phase basically consists in an exhaustive reduction of the mentioned subspace by deleting simplicial subsets that do not contain solutions. This second phase is useful for future refinement of the solutions. Previous work only focused on the first phase neglecting the second phase due to computational burden. With this in mind, we study the parallelization of the different phases of the sequential bi-blending algorithm and focus on the most time consuming phase, analyzing the performance of several strategies.

J.F.R. Herrera (✉) · L.G. Casado

Dpt. of Comput. Archit. and Electron., Univ. of Almería, Agrifood Campus of Int. Excell., ceiA3,
04120, Almería, Spain
e-mail: juanfrh@ual.es

L.G. Casado
e-mail: leo@ual.es

E.M.T. Hendrix · I. García
Computer Architecture, Málaga University, 29071, Málaga, Spain

E.M.T. Hendrix
e-mail: eligius.hendrix@wur.nl

I. García
e-mail: igarciaf@uma.es

E.M.T. Hendrix
Operations Research and Logistics, Wageningen University, Wageningen, The Netherlands

Keywords Shared memory · Parallel processors · Multithreaded · Branch-and-bound · Global optimization

1 Introduction

The problem of finding the best robust recipe that satisfies quadratic design requirements is a global optimization problem for which a guaranteed optimal solution is hard to obtain, because it can have several local optima. The feasible area may be nonconvex and even may consist of several compartments. In practice, companies are dealing with so-called multiblending problems where the same raw materials are used to produce several products [1, 3]. This complicates the search process if we intend to guarantee the optimality and robustness of the final solutions. Exhaustive search for a blending algorithm and its components are described in [4, 6, 8], while a bi-blending approach appears in [9].

Section 1.1 describes the blending problem and Sect. 1.2 defines the blending problem to obtain two mixture designs (bi-blending). Section 2 describes the sequential version of the bi-blending algorithm, and Sect. 3 its parallel model. Section 4 shows the computational results and Sect. 5 summarizes the conclusions and future work.

1.1 Blending problem

The blending problem is the basis of our study of the bi-blending. The considered blending problem is described in [8] as a Semicontinuous Quadratic Mixture Design Problem (SQMDP). Here, we summarize the main characteristics of the blending problem.

The set of possible mixtures is mathematically defined by the unit simplex $S = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1.0; x_i \geq 0\}$, where x_i , $i = 1, \dots, n$, represents the fraction of the raw material i in a mixture x . The objective is to find a recipe x that minimizes the cost of the material, $f(x) = c^T x$, where vector c gives the cost of the raw materials. Additionally, the number of raw materials in the mixture x , given by $\sum_{i=1}^n \delta_i(x)$, should be minimized, where

$$\delta_i(x) = \begin{cases} 1 & \text{if } x_i > 0, \\ 0 & \text{if } x_i = 0. \end{cases}$$

The semicontinuity of the variables is due to a minimum acceptable dose (*md*) that the practical problems reveal, i.e., either $x_i = 0$ or $x_i \geq md$. The number of resulting subsimplices (faces) is $2^n - 1$. All points x in an initial simplex P_u , $u = 1, \dots, 2^n - 1$, are mixtures of the same group of raw materials. The index u representing the group of raw materials of simplex P_u is given by $u = \sum_{i=1}^n 2^{i-1} \delta_i(x)$, $\forall x \in P_u$.

Recipes have to satisfy certain requirements. For relatively simple blending instances, bounds and linear inequality constraints define the design space $X \subset S$, see [1, 3, 16]. In practice, however, quadratic requirements appear [4, 8]. The feasible space, according to quadratic constraints, is defined as Q .

Moreover, the design must have an ε -robustness with respect to the quadratic requirements in order to maintain the feasibility of the result when small variations in the mixture appear. One can define robustness $R(x)$ of a design $x \in Q$ with respect to Q as $R(x) = \max\{R \in \mathbb{R}^+ : (x + r) \in Q, \forall r \in \mathbb{R}^n, \|r\| \leq R\}$.

Given previous aspects, SQMDP is defined as follows:

$$\begin{aligned} \min f(x), \quad & \sum_{i=1}^n \delta_i(x) \\ \text{s.t.} \quad & x \in X \cap Q \\ & R(x) \geq \varepsilon. \end{aligned}$$

In [4], tests are described based on so-called infeasibility spheres that identify areas where a feasible solution cannot be located. In [8], we described a B&B algorithm to solve SQMDP using rejection tests based on linear, quadratic, and robustness constraints. A threaded version of the B&B blending (SQMDP) algorithm was presented in [13], following a similar strategy to the one used in a parallel interval global optimization algorithm [5].

1.2 Bi-blending problem

As described in [9], when designing several products simultaneously, each product has its own demand and quality requirements which are posed as design constraints. Here, we summarize the main characteristics of the problem. Let index j represent a product with demand D_j . The amount of available raw material i is given by B_i . Now, the main decision variable is matrix \mathbf{x} , where variables $x_{i,j}$ represent the fraction of raw material i in recipe of product j .

In principle, all products $x_{*,j}$ can make use of all n raw materials; $x_{*,j} \in \mathbb{R}^n$, $j = 1, 2$. This means that $x_{i,1}$ and $x_{i,2}$ denote fractions of the same ingredient for products 1 and 2. The main restrictions that give the “bi” character to the bi-blending problem are the capacity constraints:

$$\sum_{j=1}^2 D_j x_{i,j} \leq B_i; \quad i = 1, \dots, n. \quad (1)$$

The cost function of the bi-blending problem can be written as $F(\mathbf{x}) = \sum_{j=1}^2 D_j \times f(x_{*,j})$. Having two mixtures $x_{*,1}$ and $x_{*,2}$ sharing ingredients, the other optimization criterion on the number of different raw materials is defined as minimizing $\omega(\mathbf{x}) = \sum_{i=1}^n \delta_i(x_{*,1}) \vee \delta_i(x_{*,2})$, where \vee denotes the bitwise *or* operation. The concept of Pareto optimality is used to minimize both objective functions. The Pareto front consists of minimum costs F_p^* for each number of raw materials $p = 1, \dots, n$.

The Quadratic Bi-Blending problem (QBB) is defined as follows:

$$\begin{aligned}
 & \min F(\mathbf{x}), \omega(\mathbf{x}) \\
 & \text{s.t. } x_{*,1} \in X_1 \cap Q_1, \quad x_{*,2} \in X_2 \cap Q_2 \\
 & \quad R_j(x_{*,j}) \geq \varepsilon; \quad j = 1, 2 \\
 & \quad \sum_{j=1}^2 D_j x_{i,j} \leq B_i; \quad i = 1, \dots, n.
 \end{aligned} \tag{2}$$

The next section describes a method to solve the bi-blending problem.

2 Algorithm to solve the QBB problem

We are interested in methods that find solutions \mathbf{x}_p of the QBB problem up to a guaranteed accuracy, e.g., $F(\mathbf{x}_p) - F_p^* \leq \delta$. Solving (2) in an exhaustive way (the method obtains all global solutions with a predefined precision) requires the design of a specific branch-and-bound algorithm. B&B methods can be characterized by four rules: *Branching*, *Selection*, *Bounding*, and *Elimination* [10, 14]. A *Termination rule* can be incorporated, for instance, based on the smallest sampling precision. In the branch-and-bound method, the search region is subsequently partitioned in more and more refined subsets (branching) over which bounds of an objective function value and bounds on the constraint functions are computed (bounding) which are used to determine whether the subset can contain an optimal solution.

A detailed description of Algorithm 1 can be found in [9]. Algorithm 1 makes use of a working list Λ_j and a final list Ω_j for each product j . We give a summary of the used B&B rules:

Branching: Simplex C is divided by its longest edge or that edge with the cheapest and the most expensive vertices, when all edges have the same length.

Bounding: Two bound values have to be calculated for each simplex:

Cost: $f^L(C)$ is a lower bound of the cost of a simplex C and it is equal to the minimum cost of the vertices of the simplex, because the simplices are convex and the cost function is linear.

Amount of each raw material: $b_i^L(C)$ is a lower bound of the raw material i in the simplex C . It is obtained in an analogous way as the lower bound of the cost.

Selection: A hybrid Best-Depth search is performed. The cheapest simplex, based on the sum of the cost of its vertices, is selected and a Depth-first is done until no further subdivision is possible (see Algorithm 1, lines 7 and 16). Depth-first search is used to reduce the memory requirement of the algorithm.

Rejection: Several individual tests based on linear, quadratic and robustness constraints are applied to simplices of one product, see [8]. In addition, tests are applied taking into account both products:

Capacity test: Let $\beta_{i,j}^L = D_j \times \min \{x_i : x \in C \in \Lambda_j \cup \Omega_j\}$ be a lower bound of the demand of material i in the current search space of product j . Then, a simplex C

Algorithm 1 B&B algorithm

1:	Set $ns := 2 \times (2^n - 1)$	<i>Number of simplices</i>
2:	Set the working list $\Lambda_1 := \{C_1, \dots, C_{2^n-1}\}$	
3:	Set the working list $\Lambda_2 := \{C_{2^n}, \dots, C_{ns}\}$	
4:	Set the final lists $\Omega_1 := \emptyset$ and $\Omega_2 := \emptyset$	
5:	Set $j := 1$	
6:	while $\Lambda_1, \Lambda_2 \neq \emptyset$ do	
7:	Select a simplex $C = C_k$ from Λ_j	<i>Selection rule</i>
8:	repeat	
9:	Evaluate C	
10:	Compute $f^L(C)$ and $b_i^L(C), i = 1, \dots, n$	<i>Bounding rule</i>
11:	if C cannot be eliminated then	<i>Rejection rule</i>
12:	if C satisfies the termination criterion then	<i>Termination rule</i>
13:	Store C in Ω_j	
14:	else	
15:	Divide C into C_{ns+1}, C_{ns+2}	<i>Branching rule</i>
16:	$C := \arg \min\{f^L(C_{ns+1}), f^L(C_{ns+2})\}$	<i>Select the cheapest simplex</i>
17:	Store $\{C_{ns+1}, C_{ns+2}\} \setminus C$ in Λ_j	
18:	$ns := ns + 2$	
19:	end if	
20:	end if	
21:	until C is rejected or stored in Ω_j	
22:	$j := (j \bmod 2) + 1$	<i>Alternate product</i>
23:	end while	
24:	return $\mathbf{x}_p, F_p^U, p = 1, \dots, n,$ and $\Omega_j, j = 1, 2$	

of product j does not satisfy the capacity test if

$$D_j \times b_i^L(C) + \beta_{i,j'}^L > B_i, \tag{3}$$

where j' denotes the other product.

Pareto test: Let $\phi_{u,j}^L = D_j \times \min\{f(v) : v \in C \subset P_{u,j}, C \in \Lambda_j \cup \Omega_j\}$ be a vector containing the cost value of the cheapest nonrejected mixture for initial simplex $P_{u,j}, u = 1, \dots, 2^n - 1$. Then a simplex C of product j does not satisfy the Pareto test if

$$D_j f^L(C) + \phi_{u,j'}^L > F_{\omega(x,y)}^U; \quad x \in C, y \in P_{u,j'}. \tag{4}$$

Global upper bound values $F_p^U, p = 1, \dots, n,$ are updated as follows. Every time a new vertex, satisfying individual tests, is generated by branching rule, it will be combined with all vertices of the other product that also meets individual tests to check the existence of a combination that satisfies (1) and improves F_p^U .

Termination: Nonrejected simplices that reach the required size α are stored in Ω_j .

The result of Algorithm 1 is a set of δ -guaranteed Pareto bi-blending recipe-pairs \mathbf{x}_p with their corresponding costs $F_p^U, p = 1, \dots, n,$ and lists $\Omega_j, j = 1, 2,$ that contain mixtures that have not been thrown out. During the execution of the Algorithm 1,

Algorithm 2 Combination algorithm

```

1: for  $j = 1, 2$  do
2:   for all  $C \in \Omega_j$  not tagged as valid do
3:     if  $\exists C' \in \Omega_{j'}$  that satisfies (5) and (6) then
4:       Tag  $C'$  as valid
5:       Continue with the next  $C$            Remaining  $C' \in \Omega_{j'}$  are not visited
6:     else
7:       Remove  $C$ 
8:     end if
9:   end for
10: end for

```

lower bounds $\beta_{i,j}^L$ and $\varphi_{u,j}^L$ are updated based on nonrejected vertices to discard simplices that do not satisfy (3) or (4). These lower bounds are used to avoid expensive computation related to the combination of simplices of both products.

Once Algorithm 1 has been finished, there may exist simplices in $\Omega_j, j = 1, 2$, that do not contain a solution. Algorithm 2 combines simplices of the two products in order to reduce the solution area [11], which can be used for finding more accurate Pareto-optimal solutions. The found solutions are still valid up to the used accuracy. The set of combinations that are left over after running Algorithm 2, can be used as input data for a second execution with more accurate results.

Algorithm 2 checks for each simplex $C \in \Omega_j, j = 1, 2$, whether it satisfies $\exists C' \in \Omega_{j'}$ such that

$$D_j f^L(C) + D_{j'} f^L(C') \leq F_{\omega(x,y)}^U; \quad x \in C, y \in C', \tag{5}$$

and

$$b_i^L(C) + b_i^L(C') \leq B_i; \quad i = 1, \dots, n. \tag{6}$$

Otherwise, C is rejected. In the first iteration ($j = 1$) of the outer loop, simplex C' is marked as valid if it is used to validate (5) and (6). This means, it will not be checked in the next iteration ($j = 2$).

3 Parallel strategy

The bi-blending problem is solved in two independent phases: the B&B phase (Algorithm 1) provides lists Ω_1 and Ω_2 with simplices that reached the termination criterion; the combination phase (Algorithm 2) filters out simplices without solutions. The computational characteristic of Algorithms 1 and 2 are completely different. While Algorithm 1 works with irregular data structures, Algorithm 2 is confronted with more regular ones. Algorithm 2 is run after finishing Algorithm 1. Hence, parallel models of both algorithms are analyzed separately.

The number of final simplices of Algorithm 1 depends on several factors: the dimension, the accuracy α of the termination rule, the feasible region of the instances to solve, etc. Preliminary experimentation shows that this number of final simplices

can be relatively large. Algorithm 2 is computationally much more expensive than Algorithm 1. Therefore, we first study the parallelization of Algorithm 2.

Algorithm 2 uses a nested loop, and two lists Ω_1 and Ω_2 . For each simplex $C \in \Omega_j$, a simplex $C' \in \Omega_{j'}$ must be found that satisfies (5) and (6) to keep C on the list. In the worst case (when the simplex can be removed), list $\Omega_{j'}$ is explored completely (all simplices $C' \in \Omega_{j'}$ are examined).

In the following, $\text{Pos}(C, \Omega_j)$ represents the position of the simplex C in Ω_j , NTh denotes the total number of threads and Th identifies a thread $Th = 0, \dots, NTh - 1$. There are several possible ways to build a parallel threaded approach of Algorithm 2. In this paper, we deal with two strategies:

Strategy 1: applying $NTh/2$ threads at each list Ω_j , $j = 1, 2$. Thus, iterations 1 and 2 of the outer loop are performed concurrently. This strategy requires $NTh \geq 2$. Each thread Th checks simplices in $\{C \in \Omega_j : \text{Pos}(C, \Omega_j) \bmod (NTh/2) = Th \bmod (NTh/2)\}$. After exploring both lists, the deletion of the simplices is performed by one thread per list Ω_j .

Strategy 2: applying NTh threads at the inner loop to perform an iteration of the outer loop. Each thread Th checks simplices $C \in \Omega_j$ that meets $\text{Pos}(C, \Omega_j) \bmod NTh = Th$. Now, the idea is to check just one list in parallel, removing nonfeasible simplices before exploring the other list. Deletion of the simplices (tagged for this purpose) is only performed by one of the threads at the end of each iteration j .

To avoid contention between threads in the Ω_j exploration, simplices are not deleted (line 7 of Algorithm 2) but tagged to be removed. Otherwise, the list can be modified by several threads, when simplices are removed, requiring the use of mutual exclusion.

A difficulty of parallelizing Algorithm 1 is that the pending computational work for the B&B search of one product is not known beforehand, i.e., it is an irregular algorithm. The search in one product is affected by the shared information with the other product. Moreover, the computational cost of the search in each product can be quite different due to the different design requirements. A study on the prediction of the pending work in B&B Interval Global Optimization algorithms can be found in [2]. Although authors describe their experience in B&B parallel algorithms [5, 7, 12, 15], these papers tackle only one B&B algorithm. QBB actually uses two B&B algorithms, one for each product, sharing $\beta_{i,j}^L$, $\varphi_{u,j}^L$ and F_p^U (see Eqs. (3) and (4)). The problem is to determine how many threads to assign to each product, if we want that both parallel B&B executions spend approximately the same (or similar) computing time. This will be addressed in a future study. Preliminary results show that the B&B phase is computationally negligible when compared to Comb. phase. Therefore, we will use just one static thread per product. This allows us to illustrate the challenge of load balancing.

4 Experimental results

To evaluate the performance of the parallel algorithm, we have used a pair of five-dimensional products, called UniSpec1-5 and UniSpec5b-5. Both of them are modifications of two seven-dimensional instances (UniSpec1 and UniSpec5b, respectively)

Table 1 Computational effort

	B&B phase		Comb. phase	
	BiBlendSeq	BiBlendPar	BiBlendSeq	BiBlendPar
NEvalS	2,536,862	2,537,430		
NEvalV	168,186	168,299		
QLR	887,609	888,004		
Pareto	54,050	54,050	27,284	27,284
Capacity	18,277	18,211	105,499	105,521
$ \Omega_S $	308,443	308,465	175,660	175,660
$ \Omega_V $	49,317	49,324	24,861	24,861

taken from [8] by removing raw materials 6 and 7 from the cases. This instance was solved with a robustness $\varepsilon = \sqrt{2}/100$, an accuracy $\alpha = \varepsilon$, and a minimal dose $md = 0.03$. The demand of each product is $D^T = (1, 1)$. The availability of raw material RM1 and RM3 is restricted to 0.62 and 0.6, respectively; while the others are not limited. Two solutions were found for UniSpec1-5 & UniSpec5b-5 with a different number of raw materials involved [9].

The algorithms were coded in C. For controlling the parallelization, POSIX Threads API was used to create and manipulate threads. Previous studies, as those presented in [15], show a less than linear speedup using OpenMP for B&B algorithms. A study on the parallelization of the combinatorial phase with OpenMP pragmas will be addressed in the future. The code was run on a Dell PowerEdge R810 with one octo-core Intel Xeon L7555 1.87 GHz processor, 24 MB L3 cache, 16 GB of RAM, and Linux operating system with 2.6 kernel.

Table 1 provides information about the computational work performed by the sequential algorithm (BiBlendSeq) and the parallel version (BiBlendPar) in terms of: the number of evaluated simplices (NEvalS) and vertices (NEvalV), the number of simplices rejected by linear infeasibility, quadratic infeasibility, or lack of robustness (QLR), Pareto test (Pareto) and Capacity test (Capacity). $|\Omega_S|$ and $|\Omega_V|$ give respectively the sum of the number of simplices and vertices in both lists Ω_j , $j = 1, 2$. Notice that differences between the sequential and parallel execution is negligible, so there is no detrimental neither incremental anomalies in the B&B phase. The Comb. phase is able to reduce drastically (almost 50%) the search space (number of simplices). This feeds the idea of using the complete algorithm in a iterative way to refine the solutions of the bi-blending problem with smaller values of the accuracy α .

Table 2 shows the count of last level cache (LLC) misses in thousands, reported by OProfile,¹ and the running time of the Comb. phase in BiBlendSeq and BiBlendPar ($NTh = 2, 4, 8$) by applying strategies described in Sect. 3. In order to make a fair comparison, the lists generated by the parallel B&B phase have been used as input for both the sequential and parallel Comb. phase. In Strategy 2, results are provided when the algorithm starts by filtering Ω_1 and then Ω_2 ($\Omega_1 - \Omega_2$) or vice versa ($\Omega_2 - \Omega_1$). The speedup with regard to execution time of a parallel algorithm with p process

¹<http://oprofile.sourceforge.net>.

Table 2 LLC misses (in thousands) and speedup obtained in Comb. phase

<i>NTh</i>	Strategy 1			Strategy 2 ($\Omega_1 - \Omega_2$)			Strategy 2 ($\Omega_2 - \Omega_1$)		
	LLC	Time	Sp	LLC	Time	Sp	LLC	Time	Sp
–	2,691	479.00	–	2,691	479.00	–	1,595	333.00	–
2	2,723	463.38	1.03	1,304	231.52	2.07	774	162.72	2.05
4	1,467	223.45	2.14	624	112.97	4.24	372	79.95	4.16
8	815	109.00	4.39	305	57.17	8.38	178	40.32	8.26

units is measured as $S(p) = t(1)/t(p)$, where $t(p)$ is the execution time when p threads are used.

Strategy 1 shows a poor speedup compared to Strategy 2. Strategy 1 has threads working on elements of Ω_1 and Ω_2 , where for each element of one list, the comparison is done with elements of the other list until a valid combination is found or the complete list has been checked (the worst case). This requires that many elements of both lists have to be cached, involving cache misses.

On the other hand, Strategy 2 uses all threads for checking elements of one list. In this way, only the number of elements on the current list (equal to NTh) and the elements of the other list have to be in cache for comparison. This reduces cache misses and, therefore, the running time, even more when the other list has a small size. It is illustrated when Strategy 2 starts with list Ω_2 , where $|\Omega_2| = 286,475$. The other list has a smaller size $|\Omega_1| = 21,990$, which may decrease the number of cache faults. The running time is reduced more than 25%. For this strategy, one can observe a slight super-linear speedup due to cache issues. Using more threads leads to less cache misses, which means that increasing the number of threads promotes the use of the same data in cache, instead of increasing the number of cache misses.

Regarding the B&B phase, which is the same in both strategies, BiBlendPar uses $NTh = 2$. In this phase, a slight speedup equal to 1.03 is obtained: BiBlendSeq spends 7.23 seconds and BiBlendPar spends 7 seconds. A linear speedup is not reached due to the difference of complexity between both products: UniSpec1-5 has simpler quadratic requirements compared to UniSpec5b-5; thread $Th = 1$ only spends 0.86 seconds on exploring the entire search space of UniSpec1-5, while thread $Th = 2$ spends 7 seconds to finalize the search space exploration of UniSpec5b-5.

For a better analysis of computational results, we increase the computational experience with a larger number of processors. Now, the Strategy 2 is run on a Sunfire x4600 with eight quad-core AMD Opteron 8356 2.3 GHz processors, 2 MB L3 cache, 56 GB of RAM, and Linux operating system with 2.6 kernel. The algorithm shows an almost linear speedup for a number of threads less than or equal to the number of cores in a processor. For a larger number of threads, the nonuniform memory access and the small size of the L3 cache produce a strong loss of performance.

5 Conclusions and future work

A parallelization of an algorithm to solve the bi-blending problem has been studied for a small-medium size instance of the problem. This single case illustrates the

difficulties of this type of algorithms. Bi-blending increases the challenges of the parallelization of a B&B algorithm compared to single blending, because it actually runs two B&B algorithms that share information. Additionally, in bi-blending algorithms, a combination of final simplices has to be done after the B&B phase to discard regions without a solution. This combination phase can be computationally several orders of magnitude larger than the B&B phase. Here, we use just one thread for each product in the B&B phase and several threads for the combination phase. Linear speedup is obtained on a shared memory machine with an octo-core processor and large L3 cache using one of the developed strategies. Executions in another shared-memory machine with eight quad-core processors and small L3 cache leads to a poor performance, when the number of threads is greater than the number of cores per processor.

Our intention is to develop a new version in order to reduce the cache misses and to experiment with larger dimensional problems for the parallel bi-blending algorithm, trying to decrease the computational cost. Another future research question is to develop the n -blending algorithm and its parallel version, which is the problem of interest to the industry.

Acknowledgements This work has been funded by grants from the Spanish Ministry of Science and Innovation (TIN2008-01117 and TIN2010-12011-E) and Junta de Andalucía (P08-TIC-3518 and P11-TIC-7176), in part financed by the European Regional Development Fund (ERDF). Juan F. R. Herrera is a fellow of the Spanish FPU programme. Eligius M. T. Hendrix is a fellow of the Spanish “Ramón y Cajal” contract programme, cofinanced by the European Social Fund.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. Ashayeri J, van Eijs AGM, Nederstigt P (1994) Blending modelling in a process manufacturing: a case study. *Eur J Oper Res* 72(3):460–468
2. Berenguel JL, Casado LG, García I, Hendrix EMT (2011) On estimating workload in interval branch-and-bound global optimization algorithms. *J Global Optim.* doi:10.1007/s10898-011-9771-5
3. Bertrand JWM, Rutten WGMM (1999) Evaluation of three production planning procedures for the use of recipe flexibility. *Eur J Oper Res* 115(1):179–194
4. Casado LG, Hendrix EMT, García I (2007) Infeasibility spheres for finding robust solutions of blending problems with quadratic constraints. *J Glob Optim* 39(4):577–593
5. Casado LG, Martínez JA, García I, Hendrix EMT (2008) Branch-and-bound interval global optimization on shared memory multiprocessors. *Optim Methods Softw* 23(5):689–701
6. Casado LG, García I, Tóth BG, Hendrix EMT (2011) On determining the cover of a simplex by spheres centered at its vertices. *J Glob Optim* 50(4):645–655
7. Estrada JFS, Casado LG, García I (2011) Adaptive parallel interval global optimization algorithms based on their performance for non-dedicated multicore architectures. In: 19th Euromicro international conference on parallel, distributed and network-based processing (PDP), pp 252–256
8. Hendrix EMT, Casado LG, García I (2008) The semi-continuous quadratic mixture design problem: description and branch-and-bound approach. *Eur J Oper Res* 191(3):803–815
9. Herrera JFR, Casado LG, Hendrix EMT, García I (2012) Pareto optimality and robustness in bi-blending problems. *TOP.* doi:10.1007/s11750-012-0253-9
10. Ibaraki T (1976) Theoretical comparisons of search strategies in branch and bound algorithms. *Int J Comput Inf Sci* 5(4):315–344

11. Kubica BJ (2012) A class of problems that can be solved using interval algorithms. *Computing* 94(2):271–280
12. Martínez JA, Casado LG, Alvarez JA, García I (2006) Interval parallel global optimization with Charm++. In: Dongarra J, Madsen K, Wasniewski J (eds) *Applied parallel computing. State of the art in scientific computing. Lecture notes in computer science*, vol 3732. Springer, Berlin/Heidelberg, pp 161–168
13. Martínez JA, Casado LG, García I, Hendrix EMT (2007) Solving the mixture design problem on a shared memory parallel machine. In: Guimarães N, Isaías P (eds) *Proceedings of the IADIS international conference applied computing 2007*, Salamanca, Spain. IADIS Press, pp 259–265. http://www.iadis.net/dl/Search_list_open.asp?code=3346
14. Mitten LG (1970) Branch and bound methods: general formulation and properties. *Oper Res* 18(1):24–34
15. Sanjuan-Estrada JF, Casado LG, García I (2011) Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures. *J Supercomput* 58(3):376–384
16. Williams HP (1993) *Model building in mathematical programming*. Wiley, Chichester