J Heuristics
DOI 10.1007/s10732-014-9241-6

The robust binomial approach to chance-constrained optimization problems with application to stochastic partitioning of large process networks

Oana Stan · Renaud Sirdey · Jacques Carlier · Dritan Nace

Received: 19 January 2012 / Revised: 28 May 2013 / Accepted: 19 February 2014

Abstract In this paper, we study an interpretation of the sample-based approach to chance-constrained programming problems grounded in statistical testing theory. On top of being simple and pragmatic, this approach is theoretically well founded, non parametric and leads to a general method for leveraging existing heuristic algorithms for the deterministic case to their chance-constrained counterparts. Throughout this paper, this algorithm design approach is illustrated on a real world graph partitioning problem which crops up in the field of compilation for parallel systems. Extensive computational results illustrate the practical relevance of the approach, as well as the robustness of the obtained solutions.

Keywords Chance-constrained optimization · Heuristic design · Graph partitioning

1 Introduction

In this paper, we consider optimization problems of the following general form:

$$\begin{aligned} \min_{x} \quad & g\,(x) \\ \text{s.t.} \quad & \mathbb{P}\,(G(x,\xi) \leq 0) \geq 1-\varepsilon \end{aligned} \tag{CCP}$$

O. Stan (✉) · R. Sirdey
CEA, LIST, Embedded Real Time System Laboratory, Point Courrier 172,
91191 Gif-sur-Yvette, France
e-mail: oana.stan@cea.fr

J. Carlier · D. Nace
UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne,
BP 20529, 60205 Compiègne, France

Published online: 09 March 2014

In the above model, $x \in \mathbb{R}^n$ is the decision variable vector, $\xi \in \Omega \longrightarrow \mathbb{R}^D$ represents a random vector and $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the objective function. We suppose that the probability space is $(\Omega, \Sigma, \mathbb{P})$, with Ω, the sample space, Σ, the set of events, i.e. subsets of Ω, and $\mathbb{P}$, the probability distribution on Σ. $G : \mathbb{R}^n \times \mathbb{R}^D \longrightarrow \mathbb{R}^m$ is the constraint function, $0 \leq \varepsilon \leq 1$ is a scalar defining a prescribed probability level and $\mathbb{P}(e)$ is the probability measure on the set Σ.

Introduced in the seminal work of Charnes et al. (1958), chance-constrained programs have been extensively studied under diverse flavors with many different solution techniques. Since even for simple cases (e.g. linear programs) problem (CCP) may be extremely difficult to solve, the vast majority of existing approaches model the problem by making particular assumptions about the distribution of the stochastic parameters (usually assuming either or both independence and Gaussian distribution). However, since in many real world situations these assumptions are not verified, another approach consists in approximating the chance constraints, such as in the sample-based approach, which makes use of experimental data and, more importantly, requires no assumptions on the distribution of the parameters. We propose a non parametric sample-based method which takes advantage of basic results from statistical testing theory, remaining flexible and applicable within a heuristic framework to real applications in which uncertainty can arise.

Moreover, by directly exploiting the available experimental data, without any model assumption, and by making use of existing algorithms developed for the deterministic case, our approach can be adapted to a wide range of optimization problems, without any major difficulty of integration in terms of both software engineering and execution times.

In order to illustrate our general approach, we will consider a particular problem arising in the field of compilation for real-time embedded systems, the partitioning of process networks on a clusterized parallel architecture. This problem is an extension of the more abstract problem of graph partitioning, for which the deterministic version is known to be NP-hard (Garey et al. 1976). The specific class of partitioning problems considered in this paper consists in assigning the weighted vertices of a graph to a fixed set of partitions, in order to minimize the sum of costs for edges having their extremities in different partitions, without exceeding the limited capacity of each partition and by taking into account the uncertainty affecting the vertices weights. Known for the single-dimensional deterministic case as the Node Capacitated Graph Partitioning problem (Ferreira et al. 1998), this problem has, to the best of our knowledge, received little attention from the stochastic programming community.

For solving the chance-constrained version of the capacitated graph partitioning, we applied a new method which consists in combining sampling with an existing randomized greedy heuristic, described in Stan et al. (2012) and which has proved to be reasonably efficient for the placement of the processes in the deterministic case.

As shown in the sequel, the proposed heuristic (greedy algorithm) consists in providing, with a preset level of confidence, more robust solutions for the above-mentioned problem.

The remainder of this paper is organized as follows: Section 2 presents other related works and discusses the motivations and the basic idea behind our approach. Afterwards, we report the basic results of statistical hypothesis testing theory that we use

and the general sample-based method of resolution we propose. Section 3 is dedicated to our application case. After a formal definition of our problem and a survey on existing methods for graph partitioning, we briefly recall the randomized greedy algorithm our approach is based on and present our stochastic resolution strategy. Experimental results are provided and analyzed in Sect. 4. Finally, some concluding remarks and further research directions are discussed in Sect. 5.

2 Stochastic approach

As one may expect, chance-constrained optimization problems are inherently difficult to address and although this class of problems have been studied for the last fifty years, only a few advances towards practical resolution methods have been reported. Among the main reasons that make chance-constrained programs computationally intractable in the general case are their combinatorial nature, the possible non-convexity of the feasible set and the difficulty to integrate multi-dimensional complex probability distributions.

Before presenting the main contributions of this paper, the next section provides an overview of the existing resolution approaches to chance-constrained programs with an emphasis on other heuristics methods.

2.1 State of art: chance-constrained optimization

Among the wide range of literature on the subject, there are a lot of theoretical studies dedicated to the convexity of chance-constrained. This branch of research focuses on finding convex restrictions to the space of feasible solutions such that standard methods can be then applied for a more efficient optimization. For instance, it has been shown that the only generic case for which the difficulties encountered when solving chance-constrained programs can be overcome, is the normal distribution. We may refer the reader to Prekopa (1995) for theoretical background and an extensive list of references.

Other approaches, from the field of robust optimization, consist in proposing, by relaxation techniques, equivalent deterministic programs to chance-constrained problems. However, these methods can be applied only for particular classes of problems, such as linear (Bertsimas and Sim 2004), semidefinite or quadratic programs (Ben-Tal and Nemirovski 1999) and the probabilistic considerations are accompanied by restrictions on the structure of the uncertain vector ξ (e.g. independence of the components).

Another exact approach consists in supposing that the probability distribution is known, discrete as well as having a bounded support, and subsequently solving the obtained combinatorial problem (Dentcheva et al. 2000). An example is the model used in Gaivoronski et al. (2011) for solving the quadratic knapsack problem with probability constraints which, at first view, seems identical with the formulation we propose in this paper. Although the structure is similar, this model makes the assumption that the distribution of the random constraint matrix $m \times n$ is known and has the form $\sum_{A\in\Omega} p_A \delta_A$, with $\sum_{A\in\Omega} p_A = 1$, Ω the event set and δ_ξ the Dirac distribution

centered at point $\xi \in \mathbb{R}^{m \times n}$. Under these conditions, for example, an equivalent to a linear chance-constrained program is the following mixed integer linear programming (MILP):

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b + (1 - \chi_A)L, \quad A \in \Omega \\ & \sum_{A \in \Omega} p_A \chi_A \geq (1 - \varepsilon) \\ & \chi_A \in \{0, 1\}, \ A \in \Omega. \end{aligned}$$

in which $c \in \mathbb{R}^n$ is the cost vector, $x \in \mathbb{R}^n$ is the decision variable vector, $\chi_A \in \mathbb{R}^m$ is a vector of binary variables and L is a suitable large problem-dependent constant.

Other methods for finding feasible solutions to this kind of optimization problems is through sampling (Calafiore and Campi 2006; Pagnoncelli et al. 2009). These methods are also making different assumptions on the model and distribution of the uncertain data and furthermore the solutions found are often highly conservative.

Actually, since dealing with uncertainty in optimization problems is highly complicated and difficult, the approaches that guarantee to find optimal solutions are more appropriate when solving small size instances and they also require a lot of computational effort. In contrast, approaches based on heuristics or metaheuristics are capable of finding good and even optimal solutions to problem instances of realistic size, in a smaller computation time. We refer the reader to Bianchi et al. (2006) for an extensive survey on the existing metaheuristics for dealing with stochastic combinatorial optimization problems.

However, to the best of our knowledge and as pointed in the survey (Bianchi et al. 2006), there are only a few heuristics proposed for solving the problem we consider here, formulated in (CCP), a program without recourse with uncertainty affecting the constraints.

In Loughlin and Ranjithan (1999), the approach consists in using a Monte-Carlo simulation in a genetic algorithm fitness function. For each uncertain parameter, a statistical distribution must be obtained or assumed and the sampling is carried out using either Monte-Carlo sampling or Latin hypercube sampling. If the estimated reliability of meeting one or more constraints is less than the prescribed probability level, the current solution is penalized. As such, the use of sampling is different from our approach and no theoretical guarantees are provided for establishing the number of necessary realizations.

Another method for solving chance-constrained programs, suggested in Aringhieri (2004), combines a tabu search heuristic with simulation. The evaluation of the feasibility of a solution is realized using two different methods. The first one consists in randomly generating T values for each random variable and computing the average over them in order to evaluate the constraints. The second method uses the central theorem limit to obtain a normal approximation of a sum of independent random variables. Although the first method is sample based, no statistical tools are used in order to determine and reduce the T, which is the dimension of the sample employed

to estimate the constraint feasibility. Furthermore, the second evaluation makes the simplifying assumption of independence of the random variables.

Another tabu search heuristic is proposed in Tanner and Beier (2007) for solving joint chance constrained stochastic programs with random parameters having discrete distributions. The main focus of the paper is on exploiting the scenario structure: identifying subsets of scenarios that are more important in finding good solutions, adding or removing scenarios at each iteration step. Though the ideas presented are interesting, it seems that the maintenance of the set of scenarios to work with can be computationally demanding.

A beam search heuristic, based on the classical Branch and Bound scheme, is suggested in Beraldi and Ruszczynski (2005) for solving chance constrained programs with integer variables. In order to evaluate which nodes to explore further, the heuristic is using the lower bound of the optimal solution, computed using the notion of p-efficient point. It is however worthwhile mentioning that the definition of p-efficient point is employing the conditional marginal distribution function, and thus this method supposes as known and calculable the distribution of the uncertain variables.

2.2 Basic ideas and motivations

Most of the studies mentioned above are making assumptions (e.g., existent analytical form of the distribution, independence of the random vector components) which are rather either restrictive, or difficult to verify or not always adequate to represent the uncertainty of real-life applications.

We have found that, in many real world situations, the probability distribution is not explicitly known or its integration is too difficult. One example shown in this paper is given by the execution times of medium-grained computer programs which are random variables difficult to fully describe analytically. However, in practice, we have at our disposal some observations for the uncertain data, obtained, for example, when performing tests on the target architecture. These observations can be directly employed in order to construct an equivalent optimization problem, more robust and compatible with the variations of the real data, with the condition that the available sample is sufficiently representative of the entire distribution.[1]

To the best of our knowledge, the only tractable approximation of the probabilistic constrained programs, which does not impose any restrictions on the structure of the uncertain data (in particular with respect to random vector component independence), is the one derived from the general scenario approach.

The optimization problem (CCP) can be then approximated by the convex program:

$$\begin{aligned} \min_{x} \quad & g\,(x) \\ \text{s.t.} \quad & G(x, \tilde{\xi}^{(i)}) \leq 0; \quad i = 1 \ldots NS \end{aligned} \qquad (RCP_{NS})$$

[1] An assumption that can be in practice checked, to some extent, using static program analysis techniques. An assumption which also relies reasonably on the expertise of test engineers in terms of designing validation cases representative of real-world system operation.

where $\xi^{(1)}, \ldots, \xi^{(NS)}$ is a sample of size NS of independent and identically distributed observations of ξ and $\tilde{\xi}^{(i)}$ is a realization of $\xi^{(i)}$. Let us recall that ξ is a random vector and that no assumptions are required on its joint probability distribution, in particular with respect to the independence of its components. The scenario approach searches for solutions which satisfy the probabilistic constraints for all the realizations of ξ. The acronym RCP_{NS} refers to the fact that this new formulation is a robust program where, instead of having m constraints, we have $NS \times m$ constraints. As such, this approach provides a conservative approximation to the original program, by finding feasible but suboptimal solutions. Theoretical justification of this approximation scheme can be found in (de Farias and Van Roy 2003; Calafiore and Campi 2005).

Our idea is to take advantage of the experimental data and revisit the scenario approach using elementary tools from statistical hypothesis testing theory and directly exploiting the available sample.

Also, in order to face the computational complexity which is one of the major drawbacks of the sample-based method, we propose a general way of integrating it in almost any heuristic algorithm. In this manner, even if the application case requires a high level of precision for the probability constraint threshold ε, which involves the analysis of a large sample, our approximation remains computationally tractable and, as we shall see in the next section, statistically significant.

Our algorithm design methodology consists in leveraging existing heuristics for the deterministic case without significant destructuring them (i.e. at small cost in terms of software engineering) and with acceptable performance hit. Furthermore, this method could be applicable to pretty much any such algorithm.

Hence, the contribution of this paper is more centered on demonstrating the practical relevance of our redesign-for-the-stochastic-case methodology than on demonstrating the intrinsic quality of the algorithms involved.

2.3 Statistical hypothesis testing

Before presenting the statistical results on which our method is based, let us introduce the following notation:

x	Decision vector
ξ	Uncertainty vector
p_0	$\mathbb{P}(G(x, \xi) \leq 0)$
$\xi^{(1)}, \ldots, \xi^{(NS)}$	i.i.d. random variables corresponding to NS observations of ξ
$\tilde{\xi}^{(i)}$	realization of observation $\xi^{(i)}$
χ_i	Bernoulli variable equal to 1 if $G\left(x, \xi^{(i)}\right) \leq 0$ and 0 otherwise.

So the random variable $\chi = \sum_{i=1}^{NS} \chi_i$ follows, *by definition*, a Binomial distribution with parameters NS and p_0 ($\chi \sim \mathbb{B}(NS, p_0)$). Let us now consider a realization $\tilde{\chi}$ of χ. If $\tilde{\chi}$ (corresponding to the number of times the inequality $G(x, \xi) \leq 0$ is satisfied on a sample of size NS) is sufficiently large (for instance, larger than $k(NS, 1-\varepsilon, \alpha)$), we say that the constraint $\mathbb{P}(G(x, \xi) \leq 0) \geq 1 - \varepsilon$ is *statistically satisfied*.

Table 1 Examples values for $k(NS, 0.90, 0.05)$ in function of NS

NS	$k(NS, 0.90, 0.05)$
10	–
20	–
30	29
40	38
50	48
100	95
1000	915

The value of the threshold $k(NS, 1-\varepsilon, \alpha)$ (to which, for simplicity sake, we will refer, from now on, as k) will be chosen so that the probability we accept the constraint by error is smaller than a fixed α, in which case p_0 is strictly smaller than $1-\varepsilon$:

$$\mathbb{P}(\chi \geq k) \leq \alpha \tag{1}$$

For any fixed $p_0 < 1-\varepsilon$, $\mathbb{P}(\chi \geq k)$ is smaller than $\mathbb{P}(\chi' \geq k)$ when $\chi' \sim \mathbb{B}(NS, 1-\varepsilon)$. So we can choose k such that $\mathbb{P}(\chi' \geq k) \leq \alpha$.

Thus, given x, the parameter α can be interpreted as the type I error of the statistical hypothesis test:

$$\begin{cases} \mathrm{H}_0 : \mathbb{P}\left(G(x, \xi) \leq 0\right) < 1-\varepsilon \\ \mathrm{H}_1 : \mathbb{P}\left(G(x, \xi) \leq 0\right) \geq 1-\varepsilon \end{cases}$$

H_0 is (intuitively) the hypothesis which we wish to reject only if we have statistically significant reasons to do so (which is the correct setting if we wish to confidently achieve robustness), as, recall, it is well known that the two hypothesis of such a test are not symmetric.

Hence, we can conclude, with a *confidence level* of at least $1-\alpha$, that $p_0 \geq 1-\varepsilon$.

Table 1 shows some minimal values for k in function of the sample size NS, $\varepsilon = 0.10$ and $\alpha = 0.05$. For example, for establishing that an inequality holds with a preset probability level of $1-\varepsilon = 0.90$ and with a confidence level $1-\alpha = 0.95$, for a sample of size 50, the threshold k needed is at 48 and $P(\chi \geq 48 | p_0 = 0.90; 50) \approx 0.034$. It should also be noted that, for a practical use, the parameters ε and α should be of the same order of magnitude.

Table 2 gives a deeper insight about the minimal number of constraints to respect in function of ε, the prescribed probability level and α, the confidence level when NS, the size of the sample, is equal to 30 (the folklore minimal size for which a population is considered statistically significant), 100 and respectively 1000. It should be remarked that for respecting higher probability and confidence levels, a more important sample size is needed but however, a sample size of 1000 seems sufficient even when $\varepsilon = 0.01$ and $\alpha = 0.01$.

We can also establish in advance the minimal size of the sample required for a fixed level of the probability $1-\varepsilon$ (with $\varepsilon \in]0, 1[$) and a prespecified confidence level $1-\alpha$ (with $\alpha \in]0, 1[$). In particular, if $p_0 = 1-\varepsilon$ and:

$$\mathbb{P}(\chi = NS) > \alpha$$

Table 2 Values of k in function of α and ε

	NS = 30			NS = 100			NS = 1000		
ε	0.01	0.05	0.1	0.01	0.05	0.1	0.01	0.05	0.1
α									
0.01	–	–	–	–	99	96	996	965	921
0.05	–	–	29	–	98	95	995	961	915
0.1	–	–	29	–	98	94	994	959	912

then we can affirm that the sampling size is insufficient (which is true for $NS = 10$ and $NS = 20$, see Table 1). The above formula provides an easy way to determine the minimal number of realizations that need to be drawn in order to statistically significantly (α) achieve the desired probability level (ε). We remark that its computation does not depend on the number of decision variables as in Calafiore and Campi (2006), nor on complicated complexity measures from Vapnik–Chervonenkis theory as in Vidyasagar (1999).

2.4 Chance constraints and sampling

The statistical theory above can be applied for obtaining a statistically significant approximation model to the initial program (CCP). In order to obtain a relevant equivalent program, we make the following assumptions about the random vector ξ, represented by a sample of size NS of observations $\xi^{(i)}$, with $i = 1, \ldots, NS$:

Assumption 1 *NS, the size of the sample for the uncertain vector ξ, is sufficiently representative and finite.*

Assumption 2 *The sample for ξ is composed of independent and identically distributed (i.i.d.) observations: $\xi^{(1)}, \ldots, \xi^{(NS)}$.*

We would like to attract the attention of the reader that we are not treating time series. As such, our assumption of independence is on the different observations of the random vector and not on its components which (as already stated) can be dependent. Additionally, these assumptions remain quite general. As many previous studies do not mention, they are also necessary in the case of methods using a probability model, as the model itself must be validated e.g. on a Kolmogorov–Smirnov hypothesis test *using an i.i.d. sample of experimental data*.

Moreover, the first assumption is not very restrictive, since even if the number of initial observations is not sufficient, we can resort to statistical methods for resampling, such as bootstrapping (Efron and Tibshirani 1994). However, it is important that the initial sample is representative of the distribution. We underline that we are not concerned in this paper by the acquisition of representative experimental data. This stage has to be realized a priori at system level, for example during the validation stage and needs to be done regardless of the method used for solving the chance-constrained programming. If we take the case of a video encoder for example, the validation tests should provide representative samples of video sequences which can be used for building our approximation program. Afterwards, in order to validate the

robust approach, we need other video samples, statistically identical but, of course, different from the first ones.

Let define the binary variable $\tilde{\chi}_i$ for realization $\tilde{\xi}^i$:

$$\tilde{\chi}_i = \begin{cases} 1 & \text{if } G\left(x, \tilde{\xi}^{(i)}\right) \leq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Since the sum $\sum_{i=1}^{NS} \chi_i$ follows a Binomial distribution of parameters NS and p_0 (again, by construction), we can determine $k(NS, 1-\varepsilon, \alpha)$. Therefore, we can use $\tilde{\chi}_i$, the realization of the variables χ_i, and replace the constraint probability

$$\mathbb{P}(G(x, \xi) \leq 0) \geq 1 - \varepsilon$$

to obtain the (RBP) formulation, equivalent to (CCP):

$$\min_x \quad g(x) \tag{RBP}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{i=1}^{NS} \tilde{\chi}_i \geq k(NS, 1-\varepsilon, \alpha) \\ & G(x, \tilde{\xi}^{(i)}) \leq (1 - \tilde{\chi}_i)L; \qquad i = 1, \ldots, NS \\ & \tilde{\chi}_i \in \{0, 1\}; \qquad i = 1, \ldots, NS \end{aligned} \tag{2}$$

The first constraint assures that the number of constraints which are satisfied for the given sample are superior to the threshold k, fixed in advance in function of NS, ε and α. Constraints 2 verify the respect of the constraint for each realization i, making the link between x, $\tilde{\xi}^{(i)}$ and $\tilde{\chi}_i$, with L a constant of large size, depending on the problem structure but generally easy to find. For example, for a knapsack constraint $\sum_{i=1}^{m} w_i x_i \leq C$ with $w_i \geq 0$ the weights of the m items to be placed, supposed uncertain, x_i binary variables and C the maximal capacity allowed, $L = \sum_{i=1}^{m} w_i$.

Minimizing the objective function $g(x)$ under these constraints is equivalent to solving the initial program (CCP) with a confidence level of at least $1 - \alpha$. We again emphasize that the validity of this approximation is independent of any particular assumption on the joint distribution of the random vector ξ, in particular with respect to inter-component dependencies. This is appropriate especially for the cases when such assumptions are not always verified.

In practice, although it is well illustrated on that problem, it should be stressed out that our approach is not really appropriate in a mathematical programming setting, since, for example, the reformulation of an original linear problem contains many binary variables and it is more complex to deal with. However, the method can be easily and efficiently adapted to heuristic approaches. Furthermore, we can make use of the existing heuristic algorithms developed for the deterministic version of a problem and extend them for treating the stochastic case.

Table 3 General schema for a constructive algorithm

Deterministic	Stochastic
Input: g and G functions, ξ	**Input:** g and G functions
1: $R = \{r:$ residual decisions$\}$	**Input:** $\xi_1, \ldots, \xi_{NS}$, ε, α
2: $S^* = \emptyset$	1: $R = \{r:$ residual decisions$\}$
3: **while** $R \neq \emptyset$ **do**	2: $S^* = \emptyset$
4: $D = \{r \in R : \mathcal{O}(r) = True\}$	3: **while** $R \neq \emptyset$ **do**
5: **if** $D \neq \emptyset$ **then**	4: $D = \{r \in R : \mathcal{O}_s(r) = True\}$
6: $d^* = \underset{d \in D}{\text{argmin}}\ g(S^* \cup \{d\})$	5: **if** $D \neq \emptyset$ **then**
7: $S^* = S^* \cup \{d^*\}$	6: $d^* = \underset{d \in D}{\text{argmin}}\ g(S^* \cup \{d\})$
8: $R = R \setminus \{d^*\}$	7: $S^* = S^* \cup \{d^*\}$
9: **else**	8: $R = R \setminus \{d^*\}$
10: break;	9: **else**
11: **end if**	10: break;
12: **end while**	11: **end if**
Output: S^*	12: **end while**
	Output: S^*

As we shall illustrate in the sequel through the example of graph partitioning, any constructive algorithm relying on an oracle for testing solution admissibility can be turned into an algorithm for the stochastic case by modifying the said oracle so as to count the number of constraint violations and take an admissibility decision based on the threshold k.

Table 3 shows, as an example, the general structure of a greedy algorithm for the deterministic case as well as its adaptation for the stochastic case. The input is problem specific and consists, for the deterministic case, in giving the structure of the objective g, the constraint function G, the parameter vector ξ as well as the domain of definition for the decision variables. For the chance-constrained version, in which we consider ξ as random, we also specify a sample of size NS for ξ, the probability level ε and in order to apply the robust binomial approach the confidence level α. In both cases, R represents the set of decisions not yet made (or residual), D the set of admissible decisions, $g(S)$ the solution value for solution S, d^* the current optimal decision and S^* the optimal overall solution, build in a greedy fashion. While there are residual decisions to be made, an oracle is evaluating them for deciding the admissible decisions. Between the admissible decisions, only the one with the greatest improvement on the optimal solution value is kept and the overall solution S^* is updated. If no admissible decision is found by the oracle, the algorithms stops. As seen, the only major difference when considering chance constraints is in establishing the set of admissible solutions, **by using a stochastic oracle $\mathcal{O}_s$ instead of the original one $\mathcal{O}$** (line 3). The deterministic oracle is establishing the admissibility of a residual decision by verifying the respect of the constraints, while the stochastic oracle is applying the robust binomial approach and it verifies if a residual decision is stochastically significant with a confidence level of $1 - \alpha$ for the given sample by comparing the number of constraints respected by the sample with the threshold k, established in advance in function of NS, ε and α (see the procedures for $\mathcal{O}$ and $\mathcal{O}_s$ in Table 4).

Of course, any optimization algorithm relying on an oracle to determine whether or not a solution is admissible (e.g. a neighboring method) can be turned into an

Table 4 Deterministic oracle versus stochastic oracle

Deterministic oracle $\mathcal{O}$	Stochastic oracle $\mathcal{O}_s$
Input: $r \in R, G, \xi$ 1: **if** $G(r,\xi) < 0$ **then** 2: **return** *True* 3: **end if** 4: **return** *False* **Output:** *True*, *False*	**Input:** $r \in R, G, \tilde{\xi}_1, \ldots, \tilde{\xi}_{NS}$ **Input:** $k(NS, \varepsilon, \alpha)$ 1: $nbRespConstr = 0$ 2: **for** $i = 1$ **to** NS **do** 3: **if** $G(r,\tilde{\xi}_i) < 0$ **then** 4: $nbRespConstr++$ 5: **end if** 6: **if** $nbRespConstr \geq k$ **then** 7: **return** *True* 8: **end if** 9: **end for** 10: **return** *False* **Output:** *True*, *False*

algorithm solving the stochastic case using the same method. For example, since the only difference between a generic local search method for the deterministic case and its adaptation to the stochastic version consists in deciding which one of the neighbors of the current solution is a possible admissible solution, the deterministic oracle has to be replaced by a stochastic one. The structure of the oracles $\mathcal{O}$ and $\mathcal{O}_s$ could be the same as before or they could be implemented more efficiently, using the fact that the neighbors are obtained from a current admissible solution for the deterministic and, respectively, the stochastic case.

Such a context assures a practical and tractable implementation of our approach even for cases when a very high number of constraints is demanded.

These situations can arise when ε is set to be really small (e.g. less than 10^{-5}) and thus, it is required to have a large minimal size of the sample. For example, a problem with probability level $\varepsilon = 10^{-5}$ and, accordingly, a confidence level $\alpha = 10^{-5}$, requires a sample of minimal size 10^6 which, although large, is not prohibitive. Additionally, in order to obtain a more rapid computation, the operation of counting the constraint violations can be parallelized without major effort[2].

In order to test our approach, we applied it to the problem of stochastic partitioning of large process networks, described in the next section.

3 Partitioning of process networks: an illustrative example

3.1 Experimental methodology

As already emphasized, this paper is centered in demonstrating the practical relevance of solving a stochastic problem by integrating the robust binomial approach into an existing heuristic developed for the deterministic case. As such, we are mainly interested in showing that having at our disposal an algorithm for the deterministic case, it

[2] An one line OpenMP pragma will do the trick.

is relatively easy in terms of software engineering (notably) to adapt it to the chance-constrained version of the same problem. In the latter case, the solutions found are of consistent quality (with respect to the ones provided by the original algorithm) and more importantly, guaranteed to be robust to data variations with a confidence level of $1 - \alpha$ and a required probability level of $1 - \varepsilon$.

Since for partitioning networks of processes we have already developed a multi-start constructive algorithm, we took advantage of the existing implementation in order to adapt the admissibility oracle and solve the stochastic case.

In order to have a self-contained paper as well as for comparison purposes, the original greedy algorithm for the deterministic problem is given in Sect. 3.6 and the associated computational results are presented in Sect. 4. Therefore, we are not claiming that this existing algorithm is a best-in-class graph partitioning algorithm. What we do claim is that, using a slight adaptation of this algorithm, we can easily obtain robust solutions. Thus our experiments focus on showing that the algorithm for the stochastic version provides results consistent with those of the original one and attempt to quantify the "price of robustness". We also claim that our method for leveraging an algorithm solving the stochastic case from one for the deterministic case is generally applicable.

3.2 Problem statement

We begin by a formal description of the application case considered in this paper for testing our approach.

The process networks partitioning problem can be stated as follows:

Let $G = (V, A)$ be a directed graph where the set of vertices $V = \{v_1, v_2, \ldots, v_{|V|}\}$ represents the tasks and the arcs $(v, w) \in A$ correspond to the channels of a process network. Let N be the set of disjoint nodes on a parallel architecture on which we want to map our graph. The resources (essentially memory footprint and computing core occupancy resources) are given by the set R and the capacities of the nodes are given by the multi-dimensional array $C \in \mathbb{R}^{+|R|}$. For the sake of simplicity, this study will be limited to the case of homogeneous nodes, hence we suppose all nodes have the same capacity.

Let us also define two functions. $s : V \longrightarrow \mathbb{R}^{+|R|}$, is defined as a size function for the vertex weights, with $s(v)_r$ being the weight of vertex v for resource r. The second function, defined for the edges, is the affinity function $q : A \longrightarrow \mathbb{R}^{+|R|}$ where $q((v, w)) > 0$ denotes the weight of edge $(v, w) \in A$ and $q((v, w)) = 0$ if no edge (v, w) exists between the vertices v and w. In the remaining of this paper, we will use the following simplified notation: $Q_{vw} = q((v, w))$ for each arc $(v, w) \in A$ and $S_{vr} = s(v)_r$, for $r \in R$ and $v \in V$.

The partitioning problem we work on consists in finding an assignment of vertices to nodes, denoted $f : V \longrightarrow N$, that satisfies the capacity constraints for all resources:

$$\sum_{v \in V : f(v) = n} S_{vr} \leq C_r, \quad \forall n \in N, \forall r \in R, \tag{3}$$

by minimizing the objective function:

$$\sum_{(v,w)\in A: f(v)\neq f(w)} Q_{vw}$$

As described below, a qualitative analysis of the sources of uncertainty (mainly the execution times), motivates our choice for a model in which the weights of the vertices, directly proportional to the execution times, are dependent random variables. It also shows the difficulty of obtaining an analytical description of the distribution of execution times and justifies our recourse to a non parametric sample-based approach.

Hence, the stochastic case we consider here is relatively new, even if the deterministic graph partitioning has already been extensively studied. The results of a survey on the main related works are described in a later section.

3.3 Uncertainty sources

In the problem of process networks allocation, one of the main sources of uncertainties lies in the intrinsic indeterminism of execution times for computing kernels of intermediate granularity. This indeterminism is due in part to some of the characteristics of the processor architecture such as the cache memories and memory access controllers and is also inherently due to data dependent control flows (conditional branches and loops).

Even if it is reasonable to assume, in embedded computing, that the probability distributions of execution time have a bounded support (no infinite loops), we have to cope with the fact that the distributions are intrinsically multimodal. For example, for the computing kernel "for $i = 1$ to n if x then S_1 else S_2" with n taking values between 1 and N, S_1 and S_2 being two linear sequences of instructions, the distribution has $2N$ modes. Hence, it is difficult to model these probabilities laws through usual distributions such as the normal or uniform ones, which are unimodal. Furthermore, in the case of a process network, we cannot overlook the problem of dependency between these random variables. An easy example consists in a target tracking pipeline for which the execution times of each of the pipeline elementary tasks depend, to a certain degree, on the number of effectively treated targets.

Thus, it is appropriate to assume that the execution times are random variables characterized by complicated multimodal joint distributions, presumably better defined as unions of orthotopes rather than, a Gaussian or even a mixture of Gaussians, although we do not build further on this assumption in this paper. As such, it is rather difficult to fully describe or estimate the parameters for such distributions, even by static program analysis or by dynamic analysis (i.e., testing). Nevertheless, a static analysis of the code could allow us to approximate the support of the probability law and give us a feedback on the existing modes that have or have not been sampled, i.e. on the representativeness of the tasks performed.

3.4 State of art

3.4.1 Deterministic graph partitioning

Since the graph partitioning problem and especially the bisection problem (a particular version of the problem for $|N| = 2$, also NP-hard) have been of great interest in the past, many different resolution methods were developed for treating the deterministic case. There are several surveys (see Fjällström 1998; Elsner 1997; Bichot and Durand 2010) resuming the existing algorithms for deterministic graph partitioning.

Due to the NP-hardness of graph partitioning, the literature addressing the exact resolution of this problem is relatively sparse. Among the most successful exact deterministic approaches are the branch-and-price or column generation methods (Johnson et al. 1993; Mehrotra and Trick 1997). Interesting results are also obtained in Ferreira et al. (1998), in which the polyhedral structure of the problem is analyzed and classes of strong valid inequalities are included in a branch-and-cut algorithm. We should also mention the existence of a few approaches exploiting lower bounds for the problem. Particularly new lower bounds of rather good quality were found using semidefinite programming (Lisser and Rendl 2003) as well as multi-commodity flows (Sensen 2001). Nevertheless, these exact methods can handle only relatively small graphs, being too slow to be applied to larger graphs, with, for example, more than a thousand vertices. Mainly for this reason, these methods are not adequate to our application where we have to partition instances with a number of vertices varying roughly between 500 and 4000 on 16–64 nodes.

Therefore, we turn our attention to heuristics, the usual and more practical methods for tackling such problems. There are a large number of such methods, either global or local, that differ with respect to cost (time and memory space required to run the algorithm) and partition quality, i.e. the optimal solution or the cut size. One of the earliest and most popular algorithms, due to Kernighan and Lin (1970), originally proposed for the bisection case, is of quite high complexity ($O(|E|)$ for Fiduccia adaptation (Fiduccia and Mattheyses 1982) or in the original version $O(|E|^2 log(|E|)))$ (for a graph with $|E|$ edges) and demands a lot of computational effort for being adapted to the capacitated generalized problem. Among local metaheuristics, one of the most used to solve the graph partitioning problem is simulated annealing, mainly because of its simplicity (Johnson et al. 1989; Kirkpatrick 1984). However, it highly depends on the structure of the problem and for large sized instances, the required execution time may become prohibitive. For very large graphs, rather good results were found by global approaches, such as the multilevel and hierarchical methods (Hendrickson and Leland 1995; Karypis and Kumar 1998) or the more recent method of fusion–fission (Bichot 2007).

3.4.2 Stochastic graph partitioning

Previous work related to the stochastic form of the problem treated in the present paper is quite scarce. Fan et Pardalos studied a problem relatively close to ours: partition the vertex set of a graph into several disjoints subsets so that the sum of weights of the edges between the disjoint subsets is minimized, with a cardinality constraint on each

subset and the uncertainty affecting the edge weights. In Fan and Pardalos (2010), assuming there is no information on the probability distribution other than that the weights on the links are independent and bounded in known intervals, they formulate the problem using a robust optimization model, similar to Bertsimas and Sim (2004). The equivalent linear programming formulation is then solved by an algorithm based on a decomposition method. In a more recent study, Fan et al. (2011) introduce the two-stage stochastic graph partitioning, assuming that the distribution of edge weights has finite explicit scenarios. Having as objective to minimize the expected weight of edges in the set of cuts over all scenarios, they present a nonlinear stochastic mixed integer model and propose an equivalent integer programming formulation for solving the problem using CPLEX. Taskin (2009) study the stochastic edge-partition problem, where the edge weights are uncertain, and are realized only after the node-to-subgraph assignments have been made. They introduce a two-stage cutting plane algorithm with integer variables in both stages and, to overcome the computational difficulties, they also prescribe a hybrid integer/constraint programming method as alternative.

Also, Barbu and Song-Chun (2003) addresses a graph partitioning problem with probabilities on the graph edges, using Markov-based techniques related to the simulated annealing method in order to solve a class of image segmentation problems.

The approaches above differ in several aspects from our study. First, in our case, the problem formulation is not the same, dealing with multidimensional capacity constraints on the nodes instead of cardinality constraints. Consequently, uncertainty is addressed in a different manner, the assumption of uncertainty being made on the weights of the vertices rather than on the weights of the edges. Finally, we remark that the existing methods are exact and thus, mostly suited for small-size instances of the problem, the numerical experiments being performed on graphs with at most 100 vertices. On the contrary, for the processes placement problem we are interested in practice to partition much larger graphs.

3.5 Relative affinity

Before describing the randomized greedy heuristic our stochastic algorithm is based on, let us recall the notion of relative affinity, initially introduced in David et al. (1991) (see also Stan et al. 2012 for details).

Let S and T be two disjoint subsets of V.

Definition 1 The affinity of S for T is given by :

$$\alpha(S, T) = \sum_{(v,w)\in\delta(S,T)} Q_{vw}.$$

with $\delta(S, T) = \{(v, w) : v \in S; w \in T\}$.

It follows that $\alpha(S, T) = \alpha(T, S)$.

Definition 2 The total affinity of S (similarly for T) is given by

$$\beta(S) = \alpha(S, V\backslash S).$$

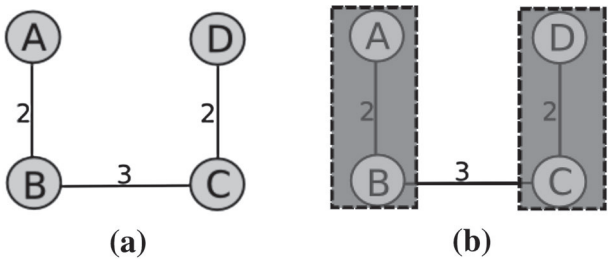

Fig. 1 **a** A graph example; **b** 2-partition using the relative affinity (David et al. 1991)

Definition 3 The relative affinity of S for T is defined as

$$\gamma(S,T) = \frac{1}{2}\alpha(S,T)\left(\frac{1}{\beta(S)} + \frac{1}{\beta(T)}\right)$$

where $\frac{\alpha(S,T)}{\beta(S)}$ represents the contribution to the total affinity of S of the edges adjacent to S and T.

Let us illustrate these notions through a simple example (David et al. 1991) on the undirected graph shown in Fig. 1a. We suppose that we have only one resource and that all the vertices have unitary weights and we want to partition the graph into two nodes of capacity equal to 2. A greedy partitioning using the total affinity would have begun by putting together the vertices B and C, resulting in a solution of cost 4. Instead, a greedy partitioning based on relative affinity would match the vertices A and B (and C and D), with $\gamma(\{A\},\{B\}) = \gamma(\{C\},\{D\}) = 0.7$ and $\gamma(\{B\},\{C\}) = 0.6$, obtaining a solution of cost 3 (see Fig. 1b).

3.6 Randomized greedy algorithm for the deterministic case

In order to fully illustrate our methodology for leveraging an existing algorithm solving the deterministic version of a problem to the stochastic case let us describe our starting point (which we do not claim to be the ultimate graph partitioning heuristic, emphasis being made on the leverage-for-the-stochastic-case methodology).

Initially described in Sirdey and David (2009), the randomized greedy algorithm we are adapting, is based on the *relative affinities* of *admissible assignments* and *admissible fusions*.

Let W be a set of vertices not yet assigned to a node.

Definition 4 An assignment of vertex v to node n is admissible if it satisfies the capacity constraints for node n, such that for every resource $r \in R$:

$$S_{vr} + \sum_{w \in V \setminus W : f(w) = n} S_{wr} \leq C_r$$

Definition 5 A fusion between the nodes n and m is admissible if for every resource $r \in R$:

$$\sum_{v \in V \setminus W : f(v)=n} S_{vr} + \sum_{v \in V \setminus W : f(v)=m} S_{vr} \leq C_r$$

The assignments are favored over fusions and, when tie-breaking with respect to relative affinity, the heuristic prioritizes the assignment of vertices with heavier weights on less loaded nodes and the fusion of the most loaded nodes. We also formally define the relations of *heavier vertex* and *more loaded node* which are being used in the algorithm for the multidimensional case.

Definition 6 The vertex v is smaller or lighter than the vertex w if:

$$\max_{r \in R} \frac{S_{vr}}{C_r} < \max_{r \in R} \frac{S_{wr}}{C_r} \tag{4}$$

Definition 7 The node n is more loaded than the node m if:

$$\max_{r \in R} \frac{1}{C_r} \left(C_r - \sum_{v \in V \setminus W : f(v)=n} S_{vr} \right) < \max_{r \in R} \frac{1}{C_r} \left(C_r - \sum_{v \in V \setminus W : f(v)=m} S_{vr} \right)$$

The algorithm, to which we will refer as RG_PART, takes as input the set of unassigned vertices W (initially equal to V), the set of nodes N, the set of resources R and the vertex weights S_{vr}. A basic version of the algorithm is given underneath.

Algorithm 1 RG_PART

Input: W, N, R, S_{vr} for each $\{v \in V, r \in R\}$

1: Initialization $W = V$

2: Assign the first $\min(|V|, |N|)$ vertices in lexicographic order to the $|N|$ nodes and update the set W

3: Find an admissible assignment (v^*, n^*) $(v^* \in W, n^* \in N)$ cf. Definition 4, if any, with maximal relative affinity:

$$\gamma_1 = \gamma(\{v^*\}, \{v \in V \setminus W : f(v) = n^*\})$$

4: Find an admissible fusion (n_1^*, n_2^*) $(n_1^* \in N, n_2^* \in N)$ cf. Definition 5, if any, with maximal relative affinity:

$$\gamma_2 = \gamma(\{v \in V \setminus W : f(v) = n_1^*\}, \{v \in V \setminus W : f(v) = n_2^*\})$$

5: If $\gamma_1 \geq \gamma_2$ then assign v^* to n^* and update the set W. Else merge n_1^* and n_2^*

6: If W is empty or there is neither any admissible assignment nor any admissible fusion, stop. Else, go to Step 3.

Output: assignment f

Since greedy algorithms tend to sometimes get trapped with poor quality solutions (due to the fact that the algorithm may, by construction, fail to compensate an early bad decision), a type of diversification strategy is required. This is the reason why a

randomized version of the algorithm is executed several times (i.e. in a *a multi-start* fashion). The randomization strategy consists in executing the algorithm first on the list of vertices sorted by their decreasing weights (see step 2 of the algorithm and for multi-resource case, Eq. 4) and several times afterwards using randomized versions of the list of vertices.

The algorithm being given for the deterministic version of our problem, we can now turn to the case we are interested in, the one in which the weights of the vertices are uncertain.

3.7 Randomized greedy algorithm: chance-constrained version

The sample based approach described previously can be easily adapted for solving the stochastic version of the capacitated graph partitioning problem. All we have to do is to combine the statistical hypothesis test explained before with the heuristic algorithm RG_PART by counting the number of constraint violations.

For the stochastic version of our graph partitioning problem, formally stated in Sect. 3.2, we make the assumptions that the task weights S_{vr} are random variables and that we dispose of a relevant sample of NS independent and identically distributed realizations of the uncertain vector of task weights. For $k = 1$ to NS, let $\tilde{S}_{vr}^{(k)}$ be the realization of the k-th observation.

Let us also note the event $e_{nr} = \{\sum_{v \in V : f(v)=n} S_{vr} \leq C_r\}$. The capacity constraint, expressed for the deterministic case in Eq. (3), becomes:

$$\mathbb{P}\left(\bigwedge_{n \in N} \bigwedge_{r \in R} e_{nr}\right) \geq 1 - \varepsilon.$$

In order to ensure that the probabilistic constraint is satisfied with a given confidence level at every step of the algorithm, it is necessary to redefine the notions of *admissible assignment* and *admissible fusion*.

Definition 8 An assignment of vertex v to node n is stochastically admissible if the sum:

$$\sum_{k=1}^{NS} \tilde{\chi}(\{\exists n' \neq n, \exists r : \sum_{w : f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\} \vee \{\exists r : \tilde{S}_{vr}^{(k)} + \sum_{w : f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\}),$$

is less than or equal to $NS - k(NS, 1 - \varepsilon, \alpha)$, with $\tilde{\chi}(\mathscr{P}_a) = 1$ if and only if $\mathscr{P}_a$ is true.

This calculation can be simplified by using an ad hoc data structure, a boolean bi-dimensional array of size $|N| \times NS$, denoted t, indicating for the partial current partitioning if, for every node, the sample k has already induced a violation.

Thus, the assignment of a vertex v to a node n is *stochastically admissible* if:

$$\sum_{k=1}^{NS} \tilde{\chi}(t[n',k] \vee \{\exists r : \tilde{S}_{vr}^{(k)} + \sum_{w:f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\}) \leq NS - k(NS, 1-\varepsilon, \alpha)$$

With the use of the boolean array, the computation of an admissible assignment increases in complexity linearly, with a factor of NS, compared to the deterministic case.

If the vertex v is effectively assigned to node n then the boolean array is updated with:

$$t[n,k] := t[n,k] \vee (\exists r : \tilde{S}_{vr}^{(k)} + \sum_{w:f(w)=n} \tilde{S}_{wr}^{(k)} > C_r)$$

Definition 9 A fusion between nodes n and m is stochastically admissible if the sum:

$$\sum_{k=1}^{NS} \tilde{\chi}(\{\exists n', r : \sum_{w:f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\} \vee \{\exists r : \sum_{w:f(w)=n} \tilde{S}_{wr}^{(k)} + \sum_{v:f(v)=m} \tilde{S}_{vr}^{(k)} > C_r\}),$$

is less than or equal to $NS - k(NS, 1-\varepsilon, \alpha)$, with $\tilde{\chi}(\mathscr{P}_f) = 1$ if and only if the predicate $\mathscr{P}_f$ is true.

Analogously, we can simplify $\mathscr{P}_f$ by using the same boolean matrix $|N| \times NS$. Once the fusion is realized, the entries for nodes n and m are updated as follows:

$$t[n,k] := t[n,k] \vee (\exists r : \sum_{w:f(w)=n} \tilde{S}_{wr}^{(k)} + \sum_{v:f(v)=m} \tilde{S}_{vr}^{(k)} > C_r)$$
$$t[m,k] := false$$

As for the computation complexity, we remark a linear increase with a factor of NS in comparison to the deterministic version.

Also, since we have to deal with a sample of size NS, we can redefine the way we compare the vertices and the nodes weights, by taking into account the average over all realizations as follows.

Definition 10 The vertex v is smaller or lighter in average than the vertex w if:

$$\max_{r \in R} \frac{\sum_{k=1}^{NS} \tilde{S}_{vr}^{(k)}}{NS * C_r} < \max_{r \in R} \frac{\sum_{k=1}^{NS} \tilde{S}_{wr}^{(k)}}{NS * C_r}$$

Definition 11 The node n is more loaded than the node m in average if:

$$\max_{r \in R} \frac{1}{C_r} \left(C_r - \frac{\sum_{v \in V \setminus W : f(v)=n} \tilde{S}_{vr}^{(k)}}{NS} \right) < \max_{r \in R} \frac{1}{C_r} \left(C_r - \frac{\sum_{v \in V \setminus W : f(v)=m} \tilde{S}_{vr}^{(k)}}{NS} \right)$$

The above definitions can then be easily integrated in the algorithm described in Sect. 3.6, without any major destructuring. As such, the algorithm for solving the chance-constrained version of the node capacitated graph partitioning problem, named RG_PART_STOCH is as follows.

Algorithm 2 RG_PART_STOCH

Input: W, N, R, ε, α, NS, $\tilde{S}_{vr}^{(k)}$ for each $\{v \in V, r \in R, k = 1...NS\}$

1: Initialization $W = V$

2: Assign the first $\min(|V|, |N|)$ vertices in lexicographic order to the $|N|$ nodes and update the set W

3: Find an admissible stochastic assignment (v^*, n^*) $(v^* \in W, n^* \in N)$ cf. Definition 8, if any, with maximal relative affinity:

$$\gamma_1 = \gamma(\{v^*\}, \{v \in V \backslash W : f(v) = n^*\})$$

4: Find an admissible stochastic fusion (n_1^*, n_2^*) $(n_1^* \in N, n_2^* \in N)$ cf. Definition 9, if any, with maximal relative affinity:

$$\gamma_2 = \gamma(\{v \in V \backslash W : f(v) = n_1^*\}, \{v \in V \backslash W : f(v) = n_2^*\})$$

5: If $\gamma_1 \geq \gamma_2$ then assign v^* to n^* and update the set W. Else merge n_1^* and n_2^*.

6: If W is empty or there is neither any admissible assignment nor any admissible fusion, stop. Else, go to Step 3.

Output: assignment f

It should be noted that the only remarkable differences between the algorithm RG_PART and its stochastic counterpart RG_PART_STOCH are in Step 3 and Step 4 when deciding if the current assignment or fusion is admissible. Additionally, the algorithm for the chance-constrained case needs as input the NS realizations of S_{vr}, the tasks weights for each resource, ε the prescribed probability level and α the confidence level.

By using the statistical hypothesis testing within a heuristic approach, we also overcome the computational effort of taking into account the uncertainties of the weights of the vertices. We could even further improve the performances of the heuristic by parallelizing the computations of admissible assignments and of admissible fusions.

4 Computational results

In this section, we report on the computation experiments of applying the above sample-based randomized greedy heuristic to the chance-constrained version of graph partitioning with uncertainty affecting the weights of the vertices. All these experiments have been carried out on a Linux PC workstation, with a 3.80 GHz Pentium(R) processor, 3 GB of memory and Ubuntu 10.04 as operating system. In the rest of the section, we report about the benchmark and the random variables used in our computation, different evaluation measures and we discuss the results of the heuristic for the chance-constrained version in comparison with the heuristic for the deterministic case.

Table 5 Computational results of RG_PART heuristic for deterministic case: grid problems

Inst.	#Vertices	#Nodes	C	Multi
Grid 4 × 4	16	4	4	8
Grid 10 × 10	100	5	20	28
Grid 23 × 23	529	14	40	150

4.1 Benchmark and uncertain parameters generation

Since, to the best of our knowledge, there are no probabilistic instances defined for the graph partitioning problem, we tested our algorithm on two modified sets of test problems, originally intended for the deterministic case.

The first set of instances consists of some examples of grids, representative in size for our application. Besides, these instances are easy to modify and we can use them to test different configurations of the parameters for our method. The second set is defined by instances publicly available defined in Johnson et al. (1993) and initially used for bisection. The tests on this second set were performed in order to confirm the effectiveness of our stochastic algorithm (both in terms of solution quality and running time) on a set of representative instances.

It should be noted that the instance "Grid 23 × 23", from the first data set, with 529 vertices and 16 nodes, is the closest in size to the real instances we have to deal with in our application context, at least as a first step.

The number of vertices for Johnson instances varies between 124 and 1000 and, for both sets, we consider the case of mono-dimensional resources.

In the deterministic case, the tests were performed for unitary weights for edges and vertices.

We have generated the random variables representing the weights of the vertices by simulating a joint bimodal distribution. The two modes are uniform in their intervals and selected in an equally likely manner.

The first mode is represented by the hypercube:

$$[0.8, 0.9]^{|V|},$$

and the second one, by the hypercube:

$$[1.1, 1.2]^{|V|}.$$

4.2 Results for the deterministic version

Table 5 shows the experimental results obtained by applying the RG_PART heuristic for graph partitioning on the first data set with deterministic vertices weights. All the results were obtained for the monodimensional case (the capacity of each node is indicated in column "C") with unitary weights for edges and vertices. The column "Multi" in Table 5 shows the solutions found by running the multi-start version of the heuristic (with 10 iterations).

The RG_PART heuristic was applied on the larger sizes instances of Johnson et al. (1993), with unitary weights for edges and vertices. As illustrated by Table 6, the

Table 6 Computational results of RG_PART heuristic for deterministic case: Johnson instances

Name	$\|V\|$	C	Best known	Multi
Gsub.500	500	250	206	236
G1000.0025	1,000	500	95	118
G1000.005	1,000	500	445	509
G1000.01	1,000	500	1,362	1,461
G1000.02	1,000	500	3,382	3,526
G124.02	124	62	13*	15
G124.04	124	62	63*	68
G124.08	124	62	178	183
G124.16	124	62	449	471
G250.01	250	125	29*	36
G250.02	250	125	114	127
G250.04	250	125	357	378
G250.08	250	125	828	855
G500.005	500	250	49*	61
G500.01	500	250	218	253
G500.02	500	250	626	669
G500.04	500	250	1,744	1,825
U1000.05	1,000	500	1*	6
U1000.10	1,000	500	39*	69
U1000.20	1,000	500	222	299
U1000.40	1,000	500	737	866
U500.05	500	250	2*	12
U500.10	500	250	26*	68
U500.20	500	250	178*	196
U500.40	500	250	412	412

solutions are reasonably close to the optimum (*) or to the best known solutions (column "Best known"). Furthermore, for most instances we observed that the solutions values found have an average differential approximation ratio (Demange and Paschos 1996) of 5.22 % compared to the best known value.

Although these results are only of moderate quality, our goal in the experimental part of this paper, as already stated in Sect. 3.1, is to provide them for self-contentedness and for serving, in the next section, as a starting point for measuring "the price of robustness" of the solutions obtained by the algorithm derived for the stochastic case.

4.3 Results for the chance-constrained version

We have tested our adaptation of the algorithm for the stochastic case on the same problems varying the parameters ε and α in the range $\{0.01, 0.05\}$. To obtain a set of stochastic instances, we have considered that the weights of the vertices are random variables with the aforementioned bimodal distribution and we generated corresponding samples of size 100 and respectively 1000. Choosing a smaller size for the sample

Table 7 Computational results of the stochastic method for $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$: grid problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Grid 4×4	6	14	$\approx$0	4.71	12	$\approx$0
Grid 10×10	6	38	0.02	23.3	29	0.01
Grid 23×23	16	182	1.12	44.1	173	0.99

Table 8 Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.05$, $\alpha = 0.05$: grid problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Grid 4×4	6	14	$\approx$0	4.712	12	$\approx$0
Grid 10×10	6	37	0.16	23.273	37	0.13
Grid 23×23	16	182	11.23	44.13	172	9.65

Table 9 Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.01$, $\alpha = 0.01$: grid problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Grid 4×4	6	14	≈ 0	4.74	10	≈ 0
Grid 10×10	6	37	0.15	23.36	37	0.13
Grid 23×23	16	182	10.75	44.183	193	9.67

may make the solution infeasible, and larger values of NS increase computation time of the problem.

The method has been implemented in C and, for each instance, 10 random iterations of our algorithm were executed. Tables 7, 8 and 9 summarize the numerical results for the grid problems for different values of the parameters NS, ε and α. The computational results for the second data set, the Johnson instances, are reported in Tables 10, 11 and 12.

For each instance from the data sets, we performed two tests. The first test consists in keeping the same node capacity as for deterministic case (see columns C from Tables 5, 6) and progressively increasing the number of nodes used in the deterministic case until the probabilistic constraint is satisfied.

The numerical results of this test, reported in section "1st test" of Tables 7, 8, 9, 10, 11 and 12 are: the minimal number of nodes for which the probabilistic constraint is respected (column "#Nodes"), the solution value (column "Sol") and the average execution time for 10 iterations in seconds (column "Time").

For the second test, we maintain the same number of nodes as in the deterministic case, but we gradually increase the capacity of all nodes (starting from one used in the deterministic case) until finding a feasible solution, satisfying the probabilistic constraint.

The results of this second test, reported in section "2nd test" of Tables 7, 8, 9, 10, 11 and 12 are: the minimal capacity of each node for which we obtain a feasible solution (column "C"), the solution value (column "Sol") and the average execution time for 10 iterations in seconds (column "Time").

Table 10 Computational results of the stochastic method for $NS = 100$, $\varepsilon = 0.05$, $1 - \alpha = 0.95$: Johnson problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Gsub.500	3	301	5.57	288.300	244	5.55
G1000.0025	3	135	58.97	575.800	131	70.11
G1000.005	3	649	62.23	575.900	513	72.10
G1000.01	3	1,865	65.30	575.900	1,456	77.36
G1000.02	3	4,481	68.78	575.940	3,579	74.10
G124.02	3	18	0.16	71.650	21	0.11
G124.04	3	91	0.16	71.650	72	0.11
G124.08	3	233	0.16	71.670	199	0.11
G124.16	3	585	0.16	71.680	475	0.12
G250.01	3	40	0.75	144.200	38	0.64
G250.02	3	162	0.76	144.260	128	0.63
G250.04	3	485	0.78	144.250	393	0.64
G250.08	3	1,074	0.77	144.200	862	0.65
G500.005	3	68	5.14	288.370	67	5.20
G500.01	3	308	5.36	288.340	269	5.04
G500.02	3	860	5.42	288.280	679	5.44
G500.04	3	2,287	5.56	288.270	1,835	5.60
U1000.05	3	17	67.76	576.100	16	73.50
U1000.10	3	101	65.55	576.100	110	77.74
U1000.20	3	417	67.80	576.200	303	75.23
U1000.40	3	1,370	68.26	576.300	1,018	77.00
U500.05	3	10	5.05	288.390	7	5.27
U500.10	3	88	5.58	288.270	66	5.38
U500.20	3	278	5.49	288.200	396	5.43
U500.40	3	663	5.23	288.380	574	5.28

It is worthwhile noting that the solutions obtained in the second test, by increasing the node capacity, are of better quality than the solutions of the first experiment (see columns "Sol") and can be adjustable more accurately. For example, in Table 8 for Grid 10×10, for finding a feasible solution, we must add two more nodes but in this case the solution found is too conservative since all the constraints are verified. Since, however, in practice it is easier to modify the number of nodes than the capacity of each node, we also investigated the results found by the first test.

Our main purpose with these tests is to get an idea of the cost of the robustness of the solutions, independently of concrete application constraints.

In evaluating the performance of our heuristic method, between the main aspects we consider are: the capacity and the number of nodes needed for finding a feasible solution, the time factor and the robustness and quality of the solutions.

In our first test, we were interested in the number of nodes needed for the stochastic case compared to the deterministic one. Our computational results show that the ratio between the number of nodes for stochastic partitioning and the number of nodes for

Table 11 Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.05$, $1 - \alpha = 0.95$: Johnson problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Gsub.500	3	298	26.72	288.240	252	18.94
G1000.0025	3	136	139.30	576.030	134	119.48
G1000.005	3	653	143.70	576.060	528	123.52
G1000.01	3	1,866	141.00	576.060	1,470	125.70
G1000.02	3	4,482	140.86	576.040	3,599	127.95
G124.02	3	17	1.39	71.662	17	0.91
G124.04	3	87	1.37	71.654	68	0.92
G124.08	3	237	1.36	71.660	182	0.93
G124.16	3	599	1.35	71.653	479	0.91
G250.01	3	39	5.84	144.310	39	4.00
G250.02	3	163	5.81	144.310	129	4.04
G250.04	3	483	5.78	144.295	387	3.99
G250.08	3	1,080	5.75	144.257	872	3.98
G500.005	3	69	26.67	288.240	68	18.88
G500.01	3	320	26.74	288.240	258	19.00
G500.02	3	853	26.87	288.250	668	19.15
G500.04	3	2,283	26.76	288.250	1,829	19.18
U1000.05	3	18	140.90	576.050	6	125.70
U1000.10	3	74	139.40	576.060	115	126.80
U1000.20	3	417	141.40	576.030	339	126.80
U1000.40	3	1,370	143.51	576.080	1,032	132.60
U500.05	3	16	26.73	288.300	2	19.49
U500.10	3	105	26.90	288.260	75	19.32
U500.20	3	289	27.15	288.250	289	19.21
U500.40	3	663	26.73	288.240	569	18.89

deterministic partitioning for the same instance is 1.5, except for Grid 23 $\times$ 23, for which the ratio is equal to $\approx$1.14. The same ratio of 1.5 was found for the Johnson instances.

For the second test, we analyzed the required increase in capacity for solving the stochastic version of the problems. The stochastic solutions of the instances reported in Tables 7, 8 and 9 are obtained for an equally large increase in the capacity of the nodes in the order of 1.1. For the Johnson instances, the capacity of nodes for stochastic partitioning is superior to the nominal capacity with $\approx$1.15. As one may expect, keeping the same probability and confidence levels and changing the sample size does not significantly affect the minimal capacity of the nodes for which a valid solution is found. On the contrary, imposing a higher probability and confidence levels demands a minimal capacity of nodes slightly larger (in the order of 0.001). Following the run of each instance, we have also observed a particular behavior consisting in a threshold effect of the solutions, sensible to the node capacity variations. One example is the problem U1000.10 for which an augmentation of the capacity from 576.06 to 576.20 results in a largely better solution (69 against 115).

Table 12 Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.01$, $1 - \alpha = 0.99$: Johnson problems

	1st test			2nd test		
Name	#Nodes	Sol	Time (s)	C	Sol	Time (s)
Gsub.500	3	298	25.32	288.610	240	18.94
G1000.0025	3	137	141	576.470	132	121.51
G1000.005	3	654	140.77	576.520	519	127.54
G1000.01	3	1,870	141.66	576.520	1,467	125.74
G1000.02	3	4,475	141.23	576.530	3,544	128.78
G124.02	3	17	1.35	71.865	17	0.9
G124.04	3	87	1.34	71.825	73	0.92
G124.08	3	237	1.33	71.851	187	0.92
G124.16	3	599	1.33	71.831	484	0.91
G250.01	3	39	5.73	144.548	40	4
G250.02	3	163	5.73	144.530	132	4
G250.04	3	483	5.65	144.515	383	4.06
G250.08	3	1,085	5.65	144.523	856	3.95
G500.005	3	69	25.33	288.490	68	19.38
G500.01	3	320	25.32	288.540	258	19
G500.02	3	853	25.2	288.530	687	19.6
G500.04	3	2,283	25.25	288.520	1,852	19.3
U1000.05	3	20	141.79	576.550	1	125.76
U1000.10	3	74	140.69	576.520	90	128.03
U1000.20	3	421	143.14	576.570	339	131.14
U1000.40	3	1,376	145.14	576.580	1,137	127.47
U500.05	3	16	26.73	288.570	2	19.17
U500.10	3	105	25.75	288.560	62	19.03
U500.20	3	289	25.4	288.560	289	19.15
U500.40	3	663	25.08	288.570	569	19.41

Concerning the time factor, the overall execution time of our method depends mainly on the number of vertices and on the size of the sample. We note that the running time needed to solve Johnson instances is considerably higher than the time required for the grid problems, the reason being the presence of instances of larger size (e.g., G1000.0025–G1000.02, U1000.05–U1000.40). As expected, the larger is the sample size, the higher is the computation time, with an average of 48.04 s. for a sample size of 1000 (Table 11) against 25.93 s. for a sample size of 100 (Table 10) for the second test. It should also be noted that the computation time for the first test is, in average, superior to the time for finding solutions in the second one. By comparison of Tables 11 and 12, it appears that when a higher probability level ε and confidence level α are imposed, a slightly higher execution time is needed.

Although these results could be improved (e.g. by code optimization and parallelism), such execution durations are already acceptable in our application context with respect to the usual compilation duration of a dataflow process network on a many core architecture.

The running times found for the stochastic version of the algorithm confirm the theoretical remarks (see Sect. 3.7) on a linear increase in complexity with a factor of NS in comparison of the deterministic case.

In order to measure the quality and the robustness of the stochastic solutions, the algorithm RG_PART was re-run with the same input parameters as the ones found with the chance-constrained method. We kept the same number of nodes and respectively the same capacity of each node as the ones for which the chance-constrained methods found feasible solutions and we considered unitary weights for arcs and unitary weights for tasks (which is the expected value of the distribution of our uncertain data).

For the first test, consisting in increasing the number of nodes, the quality of the stochastic solutions is, as expected, almost always worse than for the deterministic version and than for the solutions found by the second test. One exception is the instance U500.05, from Table 10 but this result is assumed to be due to the heuristic nature of our approach, which, by construction, provides no guarantees with respect to monotony.

Instead, the stochastic solutions of the second test are quite often close in quality to the solutions found when running RG_PART algorithm. By analyzing Tables 10 and 11, for $\varepsilon, \alpha = 0.05$ we found out that there are 14 and respectively 15 instances with a gap in the stochastic solution quality of less than 5 % from the deterministic solutions. When analyzing the results for a probability level of 0.99 and a level of confidence of 0.99 (Table 12), we remark a number of 14 stochastic solutions close (a relative 5 % gap) to the deterministic solutions.

By comparing the quality of solutions for different values of the input parameters $(NS, \varepsilon, \alpha)$ it comes out that for the same probability and confidence levels, the obtained solutions when varying the sample size are quite similar, revealing that the performance of our algorithm does not deteriorate as the number of samples increases. It should be noted that it is however necessary to determine the minimal size of the sample needed to solve the problem with the required probability level. The required sample size for $\varepsilon, \alpha = 0.01$, is at least 459, which justifies our choice not to conduct tests for these values on the samples of size 100.

Concerning the robustness of the solutions found by the presented approach, we measured the number of times the deterministic solution is not satisfied on the used samples. The percentage of samples on which the deterministic solution is not satisfying the capacity constraints 3 is, in average, for Tables 10, 11 and 12 between 48.24 and 50.04 %.

Analyzing the overall results, we observe that our stochastic heuristic confirms the capacity of computing good solutions, within an admissible average running time, even for large instances. The quality of the solutions is comparable to the deterministic case (i.e. the "price of robustness" is not too high) and moreover we guarantee that our solutions are robust to the uncertainties affecting the weights of the vertices.

5 Conclusion

In this paper, we introduced a non-parametric and general sample-based method for chance-constrained programs which, using statistical hypothesis testing, can be applied to leverage existing heuristic algorithms for the deterministic case for solving the stochastic version of the same problem. The proposed methodology is suitable to approximate chance-constrained problems where an analysis of the uncertainty data reveals complex probability distributions, for which analytical descriptions are difficult to compute and general assumptions are inappropriate. As such, the approach can be successfully applied to many practical engineering problems, in which samples on random variables are available, providing robust solutions guaranteed with a predefined statistically significant level of confidence.

Having at our disposal an admissibility oracle-based algorithm already developed for the deterministic version of a problem, it is reasonably straightforward to make the necessary changes in order to treat the stochastic case. For obtaining solutions to the chance-constrained problem (with probability level of $1 - \varepsilon$) which are statistically meaningful with a confidence level of $1 - \alpha$, all we have to do is to modify the oracle deciding on the admissibility of a solution. In the stochastic method, this admissibility oracle integrates the robust binomial approach, by verifying that the number of constraints which are respected exceeds a threshold k established based on the sample size, the prescribed probability level $1 - \varepsilon$ and the confidence level $1 - \alpha$.

We can also remark that the robust binomial approach is general. It can be easily adapted to any stochastic program, including the individual chance-constrained programs, by counting the number of times the probablistic constraints are satisfied in a realization of the sample.

As an illustration of the practical relevance of this approach, we addressed the problem of chance-constrained partitioning of communicating process networks, which arises in compilation for embedded parallel systems. After a brief analysis of the random data, we proposed an heuristic algorithm, combining sampling and statistical hypothesis testing with a randomized greedy method originally designed for the deterministic case.

Numerical results showed that the obtained solutions have often a quality consistent with those computed for the deterministic version.

More importantly, the solutions found are robust and guaranteed with a preset statistical significance level, to hold to data variations affecting the constraints. We also showed that not taking into account the stochastic nature of our data and considering only the deterministic case may lead to non feasible solutions with quite high probability (in average 50 % of cases). Furthermore, this approach can solve with an acceptable computation time problems close in dimensions to the real instances a compiler would have to treat.

In further work, we plan to investigate two directions. The first one is to design a parallelized and more efficient implementation of our method. This would allow the treatment of problems with large instances or with a high number of constraints or those which demand a high level of robustness (meaning values of ε inferior to 10^{-5}). Other direction concerns the application case, in which a more thorough analysis and characterization of execution time distribution (using for example methods of

static code analysis) would allow us to have a more accurate characterization of the uncertainties affecting the weights of the processes. Other future directions of research consist in the design and implementation of more heuristic strategies, combining the sample approximation with other approximate algorithms, originally developed for solving deterministic problems.

Acknowledgments The authors thank the anonymous referees for several suggestions that led to improvements in the paper.

References

Aringhieri, R.: Solving chance-constrained programs combining tabu search and simulation. In: Proceedings of the 3rd International Workshop on Experimental and Efficient Algorithms (WEA04). Lecture Notes in Computer Science, vol. 3059, pp. 30–41. Springer, Berlin (2004)

Barbu, A., Zhu, S.-C.: Stochastic graph partition: generalizing the Swendsen–Wang method. Technical Report Paper 2003010120, UCLA Department of Statistics (2003)

Ben-Tal, A., Nemirovski, A.: Robust solutions of uncertain linear programs. Oper. Res. Lett. **25**, 1–13 (1999)

Beraldi, P., Ruszczynski, A.: Beam search heuristic to solve stochastic integer problems under probabilistic constraints. Eur. J. Oper. Res. **167**(1), 35–47 (2005)

Bertsimas, D., Sim, M.: The price of robustness. Oper. Res. **52**(1), 35–53 (2004)

Bianchi, L., Dorigo, M., Gambardella, L., Gutjahr, W.: A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput **8**(2), 239–287 (2006)

Bichot, C.H.: A new method, the fusion fission, for the relaxed-way graph partitioning problem, and comparisons with some multilevel algorithms. J. Math. Model. Algorithms **6**(3), 319–344 (2007)

Bichot, C., Durand, N.: Partitionnement de graphe. Lavoisier, Paris (2010)

Calafiore, G., Campi, M.: Uncertain convex programs: randomized solutions and confidence levels. Math. Program. **102**, 25–46 (2005)

Calafiore, G., Campi, M.: The scenario approach to robust control design. IEEE Trans. Automat. Control **51**(5), 742–753 (2006)

Charnes, A., Cooper, W.W., Symonds, G.H.: Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. Manag. Sci. **4**(3), 235–263 (1958)

David, V., Fraboul, C., Rousselot, J.Y., Siron, P.: Etude et realisation d'une architecture modulaire et reconfigurable: Projet MODULOR. Technical Report, 1/3364/DERI.ONERA (1991)

de Farias, D., Van Roy, B.: On constraint sampling in the linear programming approach to approximate linear programming. In: Proceedings of the 42nd IEEE Conference on Decision and Control, vol. 3, pp. 2441–2446 (2003)

Demange, M., Paschos, V.: On an approximation measure founded on the links between optimization and polynomial approximation theory. Theor. Comput. Sci. **158**, 117–141 (1996)

Dentcheva, D., Prékopa, A., Ruszczynski, A.: Concavity and efficient points of discrete distributions in probabilistic programming. Math. Program. **89**, 55–77 (2000)

Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. CRC Press, Boca Raton (1994)

Elsner, U.: Graph partitioning—a survey. Technical Report, TU Chemnitz SFB393/97-27 (1997)

Fan, N., Pardalos, P.: Robust optimization of graph partitioning and critical node detection in analyzing networks. In: Proceedings of the 4th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2010), pp. 170–183 (2010)

Fan, N., Zheng, Q., Pardalos, P.: On the two-stage stochastic graph partitioning problem. In: Proceedings of the 5th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2011), pp. 500–509 (2011)

Ferreira, C.E., Martin, A., de Souza, C., Weismantel, R., Wolsey, L.: The node capacitated graph partitioning problem: a computational study. Math. Program. **81**, 229–256 (1998)

Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proceedings of the 19th Design Automation Conference. DAC '82, pp. 175–181. IEEE Press, Piscataway (1982)

Fjällström, P.O.: Algorithms for graph partitioning: a survey. Linköping Electron. Articles Comput. Inf. Sci. **3**, 10 (1998)

Gaivoronski, A., Lisser, A., Lopez, R., Xu, H.: Knapsack problem with probability constraints. J. Glob. Optim. **49**, 397–413 (2011)
Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**(3), 237–267 (1976)
Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM). ACM, New York (1995)
Johnson, D., Aragon, C., McGeoch, L., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. Oper. Res. **37**(6), 865–892 (1989)
Johnson, E., Mehrotra, A., Nemhauser, G.L.: Min-cut clustering. Math. Program. **62**, 133–151 (1993)
Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**, 359–392 (1998)
Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(1), 291–307 (1970)
Kirkpatrick, S.: Optimization by simulated annealing: quantitative studies. J. Stat. Phys. **34**, 975–986 (1984)
Lisser, A., Rendl, F.: Graph partitioning using linear and semidefinite programming. Math. Program. **95**, 91–101 (2003)
Loughlin, D.H., Ranjithan, S.: Chance-constrained genetic algorithms. In: GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 369–376 (1999)
Mehrotra, A., Trick, M.: Cliques and clustering: a combinatorial approach. Oper. Res. Lett. **22**, 1–12 (1997)
Pagnoncelli, B.K., Ahmed, S., Shapiro, A., Pardalos, P.M.: Sample average approximation method for chance constrained programming: theory and applications. J. Optim. Theory Appl. **142**, 399–416 (2009)
Prekopa, A.: Stochastic Programming. Kluwer Academic Publishers, Dordrecht (1995)
Sensen, N.: Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows. In: Meyer auF der Heide, F. (ed.) Lecture Notes in Computer Science, vol. 2161, pp. 391–403. Springer, Berlin (2001)
Sirdey, R., David, V.: Approches heuristiques des problèmes de partitionnement, placement et routage de rèseaux de processus sur architectures parallèles clusterisées. Technical Report, CEA LIST DTSI/SARC/09-470/RS (2009)
Stan, O., Sirdey, R., Carlier, J., Nace, D.: A heuristic algorithm for stochastic partitioning of process networks. In: ICSTCC (2012)
Tanner, M.W., Beier, E.B.: A general heuristic method for joint chance-constrained stochastic programs with discretely distributed parameters (2007). http://www.optimization-online.org/DB_HTML/2007/08/1755.html
Taskin, Z.C., Smith, J.C., Ahmed, S., Schaefer, A.: Cutting plane algorithms for solving a stochastic edge-partition problem. Discret. Optim. **6**(4), 420–435 (2009)
Vidyasagar, M.: Randomized algorithms for robust controller synthesis using statistical learning theory. In: Learning Control and Hybrid Systems. Lecture Notes in Control and Information Sciences, vol. 241, pp. 3–24. Springer, Berlin (1999)