9

Theory and Applications of Simulated Annealing for Nonlinear Constrained Optimization[1]

Benjamin W. Wah[1], Yixin Chen[2] and Tao Wang[3]
[1]*Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois,*
[2]*Department of Computer Science, Washington University,*
[3]*Synopsys, Inc.*
USA

1. Introduction

A general *mixed-integer nonlinear programming problem* (MINLP) is formulated as follows:

$$(P_m): \quad \min_{z} \quad f(z), \qquad \text{subject to} \quad h(z) = 0 \ \text{ and } \ g(z) \leq 0, \tag{1}$$

where $z = (x, y)^T \in \mathsf{Z}$; $x \in \mathbb{R}^v$ and $y \in \mathbb{D}^w$ are, respectively, bounded continuous and discrete variables; $f(z)$ is a lower-bounded objective function; $g(z) = (g_1(z), \ldots, g_r(z))^T$ is a vector of r inequality constraint functions;[2] and $h(z) = (h_1(z), \ldots, h_m(z))^T$ is a vector of m equality constraint functions. Functions $f(z)$, $g(z)$, and $h(z)$ are general functions that can be discontinuous, non-differentiable, and not in closed form.

Without loss of generality, we present our results with respect to minimization problems, knowing that maximization problems can be converted to minimization ones by negating their objectives. Because there is no closed-form solution to P_m, we develop in this chapter efficient procedures for finding locally optimal and feasible solutions to P_m, demonstrate that our procedures can lead to better solutions than existing methods, and illustrate the procedures on two applications. The proofs that our procedures have well-behaved convergence properties can be found in the reference [27], We first define the following terms.

[1] Research supported by the National Science Foundation Grant IIS 03-12084 and a Department of Energy Early Career Principal Investigator Grant.

[2] Given two vectors V_1 and V_2 of the same dimension, $V_1 \geq V_2$ means that each element of V_1 is greater than or equal to the corresponding element of V_2; $V_1 > V_2$ means that at least one element of V_1 is greater than the corresponding element of V_2 and the other elements are greater than or equal to the corresponding elements of V_2.

Source: Simulated Annealing, Book edited by: Cher Ming Tan, ISBN 978-953-7619-07-7, pp. 420, February 2008, I-Tech Education and Publishing, Vienna, Austria

Definition 1. A *mixed neighborhood* $\mathsf{N}_m(z)$ for $z = (x, y)^T$ in the mixed space $\mathbb{R}^v \times \mathbb{D}^w$ is:

$$\mathcal{N}_m(z) = \left\{(x', y)^T \mid x' \in \mathcal{N}_c(x)\right\} \cup \left\{(x, y')^T \mid y' \in \mathcal{N}_d(y)\right\}, \tag{2}$$

where $\mathsf{N}_c(x) = \{x' : \|x' - x\| \leq \epsilon \text{ and } \epsilon \to 0\}$ is the *continuous neighborhood* of x, and the *discrete neighborhood* $\mathsf{N}_d(y)$ is a *finite* user-defined set of points $\{y' \in \mathbb{D}^w\}$ such that $y' \in \mathsf{N}_d(y) \Leftrightarrow y \in \mathsf{N}_d(y')$ [1]. Here, $\epsilon \to 0$ means that ϵ is arbitrarily close to 0.

Definition 2. Point z of P_m is a *feasible point* iff $h(z) = 0$ and $g(z) \leq 0$.

Definition 3. Point $z^\star$ is a *constrained local minimum* (CLM_m) of P_m iff $z^\star$ is feasible, and $f(z^\star) \leq f(z)$ with respect to all feasible $z \in \mathsf{N}_m(z^\star)$.

Definition 4. Point $z^\star$ is a *constrained global minimum* (CGM_m) of P_m iff $z^\star$ is feasible, and $f(z^\star) \leq f(z)$ for every feasible $z \in \mathsf{Z}$. The set of all CGM_m of P_m is Z_{opt}.

Note that a discrete neighborhood is a user-defined concept because it does not have any generally accepted definition. Hence, it is possible for $z = (x, y)^T$ to be a CLM_m to a neighborhood $\mathsf{N}_d(y)$ but not to another neighborhood $\mathcal{N}_{d_1}(y)$. The choice, however, does not affect the validity of a search as long as one definition is consistently used throughout. Normally, one may choose $\mathsf{N}_d(y)$ to include discrete points closest to z, although a search will also be correct if the neighborhood includes "distant" points.

Finding a CLM_m of P_m is often challenging. First, $f(z)$, $g(z)$, and $h(z)$ may be non-convex and highly nonlinear, making it difficult to even find a feasible point or a feasible region. Moreover, it is not always useful to keep a search within a feasible region because there may be multiple disconnected feasible regions. To find high-quality solutions, a search may have to move from one feasible region to another. Second, $f(z)$, $g(z)$, and $h(z)$ may be discontinuous or may not be differentiable, rendering it impossible to apply existing theories based on gradients.

A popular method for solving P_m is the penalty method (Section 2.1). It transforms P_m into an unconstrained penalty function and finds suitable penalties in such a way that a global minimum of the penalty function corresponds to a CGM_m of P_m. Because it is computationally intractable to look for global minima when the penalty function is highly nonlinear, penalty methods are only effective for finding CGM_m in special cases.

This chapter is based on the theory of extended saddle points in mixed space [25, 29] (Section 2.2), which shows the one-to-one correspondence between a CLM_m of P_m and an *extended saddle point* (ESP) of the corresponding penalty function. The necessary and sufficient condition allows us to find a CLM_m of P_m by looking for an ESP of the corresponding penalty function.

One way to look for those ESPs is to minimize the penalty function, while gradually increasing its penalties until they are larger than some thresholds. The approach is not sufficient because it also generates stationary points of the penalty function that are not

CLM_{m} of P_{m}. To avoid those undesirable stationary points, it is possible to restart the search when such stationary points are reached, or to periodically decrease the penalties in order for the search to escape from such local traps. However, this simple greedy approach for updating penalties may not always work well across different problems.

Our goals in this chapter are to design efficient methods for finding ESPs of a penalty formulation of P_m and to illustrate them on two applications. We have made three contributions in this chapter.

First, we propose in Section 3.1 a constrained simulated annealing algorithm (CSA), an extension of conventional simulated annealing (SA) [18], for solving P_m. In addition to probabilistic descents in the problem-variable subspace as in SA, CSA does probabilistic ascents in the penalty subspace, using a method that controls descents and ascents in a unified fashion. Because CSA is sample-based, it is inefficient for solving large problems. To this end, we propose in Section 3.2 a constraint-partitioned simulated annealing algorithm (CPSA). By exploiting the locality of constraints in many constraint optimization problems, CPSA partitions P_m into multiple loosely coupled subproblems that are related by very few global constraints, solves each subproblem independently, and iteratively resolves the inconsistent global constraints.

Second, we show in Section 4 the asymptotic convergence of CSA and CPSA to a constrained global minimum with probability one in discrete constrained optimization problems, under a specific temperature schedule [27]. The property can be proved by modeling the search as a strongly ergodic Markov chain and by showing that CSA and CPSA minimize an implicit virtual energy at any constrained global minimum with probability one. The result is significant because it extends conventional SA, which guarantees asymptotic convergence in discrete unconstrained optimization, to that in discrete constrained optimization. It also establishes the condition under which optimal solutions can be found in constraint-partitioned nonlinear optimization problems.

Last, we evaluate CSA and CPSA in Section 5 by solving some benchmarks in continuous space and by demonstrating their effectiveness when compared to other dynamic penalty methods. We also apply CSA to solve two real-world applications, one on sensor-network placements and another on out-of-core compiler code generation.

2. Previous work on penalty methods

Direct and penalty methods are two general approaches for solving P_m. Since direct methods are only effective for solving some special cases of P_m, we focus on penalty methods in this chapter.

A penalty function of P_m is a summation of its objective and constraint functions weighted by penalties. Using penalty vectors $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^r$, the general penalty function for P_m is:

$$L_p\big((z,\alpha,\beta)^T\big) = f(z) + \alpha^T P(h(z)) + \beta^T Q(g(z)), \tag{3}$$

where P and Q are transformation functions. The goal of a penalty method is to find suitable α^* and β^* in such a way that z^* that minimizes (3) corresponds to either a CLM_m or a

CGM_m of P_m. Penalty methods belong to a general approach that can solve continuous, discrete, and mixed constrained optimization problems, with no continuity, differentiability, and convexity requirements.

When $P(g(z))$ and $Q(h(z))$ are general functions that can take positive and negative values, unique values of α^* and β^* must be found in order for a local minimum z^* of (3) to correspond to a CLM_m or CGM_m of P_m. (The proof is not shown.) However, the approach of solving P_m by finding local minima of (3) does not always work for discrete or mixed problems because there may not exist any feasible penalties at z^*. (This behavior is illustrated in Example 1 in Section 2.1.) It is also possible for the penalties to exist at z^* but (3) is not at a local minimum there. A special case exists in continuous problems when constraint functions are continuous, differentiable, and regular. For those problems, the Karush-Kuhn-Tucker (KKT) condition shows that unique penalties always exist at constrained local minima [21]. In general, existing penalty methods for solving P_m transform $g(z)$ and $h(z)$ in (3) into non-negative functions before finding its local or global minima. In this section, we review some existing penalty methods in the literature.

2.1 Penalty methods for constrained global optimization

Static penalty methods. A *static-penalty method* [21, 22] formulates P_m as the minimization of (3) when its transformed constraints have the following properties: a) $P(h(z)) \geq 0$ and $Q(g(z)) \geq 0$; and b) $P(h(z)) = 0$ iff $h(z) = 0$, and $Q(g(z)) = 0$ iff $g(z) \leq 0$. By finding suitable penalty vectors α and β, an example method looks for z^* by solving the following problem with constant $\rho > 0$:

$$(P_1): \qquad \min_z L_s\left((z,\alpha,\beta)^T\right) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i \, |h_i(z)|^\rho + \sum_{j=1}^{r} \beta_j \, \left(g_j(z)^+\right)^\rho \right], \quad (4)$$

where $g_j(z)^+ = \max(0,\ g_j(z))$, and $g(z)^+ = (g_1(z)^+,\ \ldots,\ g_r(z)^+)^T$.

Given z^*, an interesting property of P_1 is that z^* is a CGM_m of P_m iff there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that z^* is a global minimum of $L_s((z,\ \alpha^{**},\ \beta^{**})_T)$ for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. To show this result, note that α_i and β_j in P_1 must be greater than zero in order to penalize those transformed violated constraint functions $|h_i(z)|^\rho$ and $(g_j(z)^+)^\rho$, which are non-negative with a minimum of zero. As (4) is to be minimized with respect to z, increasing the penalty of a violated constraint to a large enough value will force the corresponding transformed constraint function to achieve the minimum of zero, and such penalties always exist if a feasible solution to P_m exists. At those points where all the constraints are satisfied, every term on the right of (4) except the first is zero, and a global minimum of (4) corresponds to a CGM_m of P_m.

Example 1. Consider the following simple discrete optimization problem:

$$\min_{\substack{y\in\{-3,-2,\\ -1,0,1,2\}}} f(y) = \begin{cases} 0 & \text{if } y \geq 0 \\ y & \text{if } y = -1, -2 \\ -4 & \text{if } y = -3 \end{cases} \qquad \text{subject to } y = 0. \tag{5}$$

Obviously, $y^* = 0$. Assuming a penalty function $L_p((y,\ \alpha)^T) = f(y) + \alpha y$ and $\mathsf{N}_d(y) = \{y-1,\ y+1\}$, there is no single α^* that can make $L_p((y,\ \alpha^*)^T)$ a local minimum at $y^* = 0$ with respect to $y = \pm 1$. This is true because we arrive at an inconsistent α^* when we solve the following inequalities:

$$0 = L_p\big((0,\alpha^*)^T\big) \ \leq \ \begin{cases} L_p\big((-1,\alpha^*)^T\big) = f(-1) - \alpha^* \ = -1 - \alpha^* \\ L_p\big((1,\alpha^*)^T\big) = f(1) + \alpha^* = 0 + \alpha^* \end{cases} \implies \begin{cases} \alpha^* \leq -1 & \text{when } y = -1 \\ \alpha^* \geq 0 & \text{when } y = 1. \end{cases}$$

On the other hand, by using $L_s((y,\ \alpha)^T) = f(y) + \alpha\ |y|$ and by setting $\alpha^* = \dfrac{4}{3}$, the CGM_d of (5) corresponds to the global minimum of $L_s((y,\ \alpha^{**})^T)$ for any $\alpha^{**} > \alpha^*$. ■

A variation of the static-penalty method proposed in [16] uses discrete penalty values and assigns a penalty value $\alpha_i(h_i(z))$ when $h_i(z)$ exceeds a discrete level ℓ_i (*resp.*, $\beta_j(g_j(z))$ when $g_j(z)^+$ exceeds a discrete level ℓ_j), where a higher level of constraint violation entails a larger penalty value. The penalty method then solves the following minimization problem:

$$(P_2): \quad \min_z L_s\big((z,\alpha,\beta)^T\big) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(h_i(z))\ h_i^2(z) + \sum_{j=1}^{r} \beta_j(g_j(z))\big(g_j(z)^+\big)^2 \right]. \tag{6}$$

A limitation common to all static-penalty methods is that their penalties have to be found by trial and error. Each trial is computationally expensive because it involves finding a global minimum of a nonlinear function. To this end, many penalty methods resort to finding local minima of penalty functions. However, such an approach is heuristic because there is no formal property that relates a CLM_m of P_m to a local minimum of the corresponding penalty function. As illustrated earlier, it is possible that no feasible penalties exist in order to have a local minimum at a CLM_m in the penalty function. It is also possible for the penalties to exist at the CLM_m but the penalty function is not at a local minimum there.

Dynamic penalty methods. Instead of finding α^{**} and β^{**} by trial and error, a *dynamic-penalty method* [21, 22] increases the penalties in (4) gradually, finds the global minimum z^* of (4) with respect to z, and stops when z^* is a feasible solution to P_m. To show that z^* is a CGM_m when the algorithm stops, we know that the penalties need to be increased when z^* is a global minimum of (4) but not a feasible solution to P_m. The first time z^* is a feasible solution to P_m, the solution must also be a CGM_m. Hence, the method leads to the smallest

α^{**} and β^{**} that allow a CGM_m to be found. However, it has the same limitation as static-penalty methods because it requires computationally expensive algorithms for finding the global minima of nonlinear functions.

There are many variations of dynamic penalty methods. A well-known one is the *non-stationary method* (NS) [17] that solves a sequence of minimization problems with the following in iteration t:

$$(P_3): \quad \min_z L_t\big((z,\alpha,\beta)^T\big) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(t)\ |h_i(z)|^\rho + \sum_{j=1}^{r} \beta_j(t)\big(g_j(z)^+\big)^\rho \right] \quad (7)$$

where $\alpha_i(t+1) = \alpha_i(t) + C \cdot |h_i(z(t))|, \quad \beta_j(t+1) = \beta_j(t) + C \cdot g_j(z(t))^+.$

Here, C and ρ are constant parameters, with a reasonable setting of C = 0.01 and ρ = 2. An advantage of the NS penalty method is that it requires only a few parameters to be tuned.

Another dynamic penalty method is the *adaptive penalty method* (AP) [5] that makes use of a feedback from the search process. AP solves the following minimization problem in iteration t:

$$(P_4): \quad \min_z L_t\big((z,\alpha,\beta)^T\big) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(t)\ h_i(z)^2 + \sum_{j=1}^{r} \beta_j(t)\big(g_j(z)^+\big)^2 \right], \quad (8)$$

where $\alpha_i(t)$ is, respectively, increased, decreased, or left unchanged when the constraint $h_i(z) = 0$ is respectively, infeasible, feasible, or neither in the last ℓ iterations. That is,

$$\alpha_i(t+1) = \begin{cases} \frac{\alpha_i(t)}{\lambda_1} & \text{if } h_i(z(i)) = 0 \text{ is feasible in iterations } t-\ell+1,\ldots,t \\ \lambda_2 \cdot \alpha_i(t) & \text{if } h_i(z(i)) = 0 \text{ is infeasible in iterations } t-\ell+1,\ldots,t \\ \alpha_i(t) & \text{otherwise,} \end{cases} \quad (9)$$

where ℓ is a positive integer, $\lambda_1,\ \lambda_2 > 1$, and $\lambda_1 \neq \lambda_2$ in order to avoid cycles in updates. We use ℓ = 3, λ_1 = 1.5, and λ_2 = 1.25 in our experiments. A similar rule applies to the updates of $\beta_j(t)$.

The *threshold penalty method* estimates and dynamically adjusts a near-feasible threshold $q_i(t)$ (*resp.*, $q'_j(t)$) for each constraint in iteration t. Each threshold indicates a reasonable amount of violation allowed for promising but infeasible points during the solution of the following problem:

$$(P_5): \quad \min_z L_t\big((z,\alpha,\beta)^T\big) = \min_z \left\{ f(z) + \alpha(t) \left[\sum_{i=1}^{m} \left(\frac{h_i(z)}{q_i(t)} \right)^2 + \sum_{j=1}^{r} \left(\frac{g_j(z)^+}{q'_j(t)} \right)^2 \right] \right\}. \quad (10)$$

There are two other variations of dynamic penalty methods that are not as popular: the death penalty method simply rejects all infeasible individuals [4]; and a penalty method that uses the number of violated constraints instead of the degree of violations in the penalty function [20].

Exact penalty methods. Besides the dynamic penalty methods reviewed above that require solving a series of unconstrained minimization problems under different penalty values, the *exact penalty methods* are another class of penalty methods that can yield an optimal solution by solving a single unconstrained optimization of the penalty function with appropriate penalty values. The most common form solves the following minimization problem in continuous space [35, 6]:

$$\min_x L_e\big((x,c)^T\big) = \min_x \left[f(x) + c\Big(\sum_{i=1}^{m} |h_i(x)| + \sum_{j=1}^{r} g_j(x)^+ \Big) \right]. \quad (11)$$

It has been shown that, for continuous and differentiable problems and when certain constraint qualification conditions are satisfied, there exists $c^* > 0$ such that the x^* that minimizes (11) is also a global optimal solution to the original problem [35, 6]. In fact, c needs to be larger than the summation of all the Lagrange multipliers at x^*, while the existence of the Lagrange multipliers requires the continuity and differentiability of the functions.

Besides (11), there are various other formulations of exact penalty methods [11, 12, 10, 3]. However, they are limited to continuous and differentiable functions and to global optimization. The theoretical results for these methods were developed by relating their penalties to their Lagrange multipliers, whose existence requires the continuity and differentiability of the constraint functions.

In our experiments, we only evaluate our proposed methods with respect to dynamic penalty methods P_3 and P_4 for the following reasons. It is impractical to implement P_1 because it requires choosing some suitable penalty values a priori. The control of progress in solving P_2 is difficult because it requires tuning many ($\ell \cdot (m+r)$) parameters that are hard to generalize. The method based on solving P_5 is also hard to generalize because it depends on choosing an appropriate sequence of violation thresholds. Reducing the thresholds quickly leads to large penalties and the search trapped at infeasible points, whereas reducing the thresholds slowly leads to slow convergence. We do not evaluate exact penalty methods because they were developed for problems with continuous and differentiable functions.

2.2 Necessary and sufficient conditions on constrained local minimization

We first describe in this section the theory of extended saddle points (ESPs) that shows the one-to-one correspondence between a CLM_m of P_m and an ESP of the penalty function. We then present the partitioning of the ESP condition into multiple necessary conditions and the formulation of the corresponding subproblems. Because the results have been published earlier [25, 29], we only summarize some high-level concepts without the precise formalism and their proofs.

Definition 5. For penalty vectors $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^r$, we define a *penalty function* of P_m as:

$$L_m\big((z,\alpha,\beta)^T\big) = f(z) + \alpha^T|h(z)| + \beta^T g(z)^+ = f(z) + \sum_{i=1}^{m} \alpha_i |h_i(z)| + \sum_{j=1}^{r} \beta_j g_j(z)^+. \quad (12)$$

Next, we informally define a constraint-qualification condition needed in the main theorem [25]. Consider a feasible point $z' = (x', y')^T$ and a neighboring point $z'' = (x' + \vec{p}, y')^T$ under an infinitely small perturbation along direction $\vec{p} \in X$ in the x subspace. When the *constraint-qualification condition* is satisfied at z', it means that there is no $\vec{p}$ such that the rates of change of all equality and active inequality constraints between z'' and z' are zero. To see why this is necessary, assume that $f(z)$ at z' decreases along $\vec{p}$ and that all equality and active inequality constraints at z' have zero rates of change between z'' and z'. In this case, it is not possible to find some finite penalty values for the constraints at z'' in such a way that leads to a local minimum of the penalty function at z' with respect to z''. Hence, if the above scenario were true for some $\vec{p}$ at z', then it is not possible to have a local minimum of the penalty function at z'. In short, constraint qualification at z' requires at least one equality or active inequality constraint to have a non-zero rate of change along each direction $\vec{p}$ at z' in the x subspace.

Theorem 1. Necessary and sufficient condition on CLM_m of P_m [25]. Assuming $z^* \in \mathsf{Z}$ of P_m satisfies the constraint-qualification condition, then z^* is a CLM_m of P_m iff there exist some finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ that satisfies the following extended saddle-point condition (ESPC):

$$L_m\left((z^*, \alpha, \beta)^T\right) \leq L_m\left((z^*, \alpha^{**}, \beta^{**})^T\right) \leq L_m\left((z, \alpha^{**}, \beta^{**})^T\right) \tag{13}$$

for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$ and for all $z \in \mathsf{N}_m(z^*)$, $\alpha \in \mathbb{R}^m$, and $\beta \in \mathbb{R}^r$.

Note that (13) can be satisfied under rather loose conditions because it is true for a range of penalty values and not for unique values. For this reason, we call $(z^*, \alpha^{**}, \beta^{**})^T$ an *extended saddle point* (ESP) of (12). The theorem leads to an easy way for finding CLM_m. Since an ESP is a local minimum of (12) (but not the converse), z^* can be found by gradually increasing the penalties of those violated constraints in (12) and by repeatedly finding the local minima of (12) until a feasible solution to P_m is obtained. The search for local minima can be accomplished by any existing local-search algorithm for unconstrained optimization.

Example 1 (cont'd). In solving (5), if we use $L_m((y, \alpha)^T) = f(y) + \alpha|y|$ and choose $\alpha^* = 1$ we have an ESP at $y^* = 0$ for any $\alpha^{**} > \alpha^*$. This establishes a local minimum of $L_m((y, \alpha)^T)$ at $y^* = 0$ with respect to $\mathsf{N}_d(y) = \{y - 1, y + 1\}$. Note that the α^* that satisfies Theorem 1 is only required to establish a local minimum of $L_m((y, \alpha)^T)$ at $y^* = 0$ and is, therefore, smaller than the α^* $(= \frac{4}{3})$ required to establish a global minimum of $L_m((y, \alpha)^T)$ in the static-penalty method. ■

An important feature of the ESPC in Theorem 1 is that it can be partitioned in such a way that each subproblem implementing a partitioned condition can be solved by looking for any α^{**} and β^{**} that are larger than α^* and β^*.

Consider P_t, a version of P_m whose constraints can be partitioned into N subsets:

$$\begin{array}{lllll} (P_t): & \min\limits_{z} & f(z) & & \\ & \text{subject to} & h^{(t)}(z(t))=0, & g^{(t)}(z(t))\le 0 & \text{(local constraints)} \\ & \text{and} & H(z)=0, & G(z)\le 0 & \text{(global constraints).} \end{array} \tag{14}$$

Each subset of constraints can be treated as a subproblem, where Subproblem t, $t = 1, \ldots, N$, has local *state vector* $z(t) = (z_1(t), \ldots, z_{u_t}(t))^T$ of u_t mixed variables, and $\cup_{t=1}^{N} z(t) = z$. Here, $z(t)$ includes all the variables that appear in any of the m_t local equality constraint functions $h^{(t)} = \left(h_1^{(t)}, \ldots, h_{m_t}^{(t)}\right)^T$ and the r_t local inequality constraint functions $g^{(t)} = \left(g_1^{(t)}, \ldots, g_{r_t}^{(t)}\right)^T$. Since the partitioning is by constraints, $z(1), \ldots, z(N)$ may overlap with each other. Further, $z(g)$ includes all the variables that appear in any of the p global equality constraint functions $H = (H_1, \ldots, H_p)^T$ and the q global inequality constraint functions $G = (G_1, \ldots, G_q)^T$.

We first define $\mathcal{N}_m(z)$, the mixed neighborhood of z for P_t, and decompose the ESPC in (13) into a set of necessary conditions that collectively are sufficient. Each partitioned ESPC is then satisfied by finding an ESP of the corresponding subproblem, and any violated global constraints are resolved by finding some appropriate penalties.

Definition 6. $\mathcal{N}_{p_0}(z)$, the *mixed neighborhood* of z for P_t when partitioned by its constraints, is:

$$\mathcal{N}_{p_0}(z) = \bigcup_{t=1}^{N} \mathcal{N}_{p_1}^{(t)}(z) = \bigcup_{t=1}^{N} \left\{ z' \;\middle|\; z'(t) \in \mathcal{N}_{m_1}(z(t)) \text{ and } z_i' = z_i \;\forall z_i \notin z(t) \right\}, \tag{15}$$

where $\mathcal{N}_{m_1}(z(t))$ is the mixed neighborhood of $z(t)$ (see Definition 2).

Intuitively, $\mathcal{N}_{m_1}(z(t))$ is separated into N neighborhoods, where the t^{th} neighborhood only perturbs the variables in $z(t)$ while leaving those variables in $z \backslash z(t)$ unchanged.

Without showing the details, we can consider P_t as a MINLP and apply Theorem 1 to derive its ESPC. We then decompose the ESPC into N necessary conditions, one for each subproblem, and an overall necessary condition on the global constraints across the subproblems. We first define the penalty function for Subproblem t.

Definition 7. Let $\Phi((z,\gamma,\eta)^T) = \gamma^T |H(z)| + \eta^T G(z)^+$ be the sum of the transformed global constraint functions weighted by their penalties, where $\gamma = (\gamma_1, \ldots, \gamma_p)^T \in \mathbb{R}^p$ and $\eta = (\eta_1, \ldots, \eta_q)^T$ are the penalty vectors for the global constraints. Then the penalty function for P_t in (14) and the corresponding penalty function in Subproblem t are defined as follows:

$$L_m\left((z,\alpha,\beta,\gamma,\eta)^T\right) = f(z) + \sum_{t=1}^{N} \left\{ \alpha(t)^T |h^{(t)}(z(t))| + \beta(t)^T \left(g^{(t)}(z(t))\right)^+ \right\} + \Phi((z,\gamma,\eta)^T), \tag{16}$$

$$\Gamma_m((z,\alpha(t),\beta(t),\gamma,\eta)^T) = f(z) + \alpha(t)^T|h^{(t)}(z(t))| + \beta(t)^T\big(g^{(t)}(z(t))\big)^+ + \Phi((z,\gamma,\eta)^T), \quad (17)$$

where $\alpha(t) = (\alpha_1(t),\ldots,\alpha_{m_t}(t))^T \in \mathbb{R}^{m_t}$ and $\beta(t) = (\beta_1(t),\ldots,\beta_{r_t}(t))^T \in \mathbb{R}^{r_t}$ are the penalty vectors for the local constraints in Subproblem *t*.

Theorem 2. *Partitioned necessary and sufficient ESPC on CLMm* of Pt [25]. Given $\mathcal{N}_{p_0}(z)$, the ESPC in (13) can be rewritten into *N* + 1 necessary conditions that, collectively, are sufficient:

$$\begin{aligned}\Gamma_m((z^*,\alpha(t),\beta(t),\gamma^{**},\eta^{**})^T) &\leq \Gamma_m((z^*,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**})^T)\\ &\leq \Gamma_m((z,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**})^T) \quad (18)\end{aligned}$$

$$L_m\big((z^*,\alpha^{**},\beta^{**},\gamma,\eta)^T\big) \leq L_m\big((z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**})^T\big) \quad (19)$$

for any $\alpha(t)^{**} > \alpha(t)^* \geq 0$, $\beta(t)^{**} > \beta(t)^* \geq 0$, $\gamma^{**} > \gamma^* \geq 0$, and $\eta^{**} > \eta^* \geq 0$, and for all $z \in \mathcal{N}_{p_1}^{(t)}(z^*)$, $\alpha(t) \in \mathbb{R}^{m_t}$, $\beta(t) \in \mathbb{R}^{r_t}$, $\gamma \in \mathbb{R}^p$, $\eta \in \mathbb{R}^q$, and $t = 1,\ldots,N$.

Theorem 2 shows that the original ESPC in Theorem 1 can be partitioned into *N* necessary conditions in (18) and an overall necessary condition in (19) on the global constraints across the subproblems. Because finding an ESP to each partitioned condition is equivalent to solving a MINLP, we can reformulate the ESP search of the t^{th} condition as the solution of the following optimization problem:

$$\begin{aligned}\left(P_t^{(t)}\right): \qquad &\min_{z(t)} \quad f(z) + \gamma^T|H(z)| + \eta^T G(z)^+\\ &\text{subject to} \quad h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \leq 0.\end{aligned} \quad (20)$$

The weighted sum of the global constraint functions in the objective of (20) is important because it leads to points that minimize the violations of the global constraints. When γ^T and η^T are large enough, solving $P_t^{(t)}$ will lead to points, if they exist, that satisfy the global constraints. Note that $P_t^{(t)}$ is very similar to the original problem and can be solved by the same solver to the original problem with some modifications on the objective function to be optimized.

In summary, we have shown in this section that the search for a CLM_m of P_m is equivalent to finding an ESP of the corresponding penalty function, and that this necessary and sufficient condition can be partitioned into multiple necessary conditions. The latter result allows the original problem to be decomposed by its constraints to multiple subproblems and to the reweighting of those violated global constraints defined by (19). The major benefit of this decomposition is that each subproblem involves only a fraction of the original constraints and is, therefore, a significant relaxation of the original problem with much lower complexity. The decomposition leads to a large reduction in the complexity of the original problem if the global constraints is small in quantity and can be resolved efficiently. We demonstrate in Section 5 that the number of global constraints in many benchmarks is indeed small when we exploit the locality of the constraints. In the next section, we describe our extensions to simulated annealing for finding ESPs.

3. Simulated annealing for constrained optimization

In this section, we present three algorithms for finding ESPs: the first two implementing the results in Theorems 1 and 2, and the third extending the penalty search algorithms in Section 2.1. All three methods are based on sampling the search space of a problem during their search and can be applied to solve continuous, discrete, and mixed-integer optimization problems. Without loss of generality, we only consider P_m with equality constraints, since an inequality constraint $g_j(z) \leq 0$ can be transformed into an equivalent equality constraint $g_j(z)^+ = 0$.

3.1 Constrained simulated annealing (CSA)

Figure 1 presents CSA, our algorithm for finding an ESP whose $(z^*, \alpha^{**})^T$ satisfies (13). In addition to probabilistic descents in the z subspace as in SA [18], with an acceptance probability governed by a temperature that is reduced by a properly chosen cooling schedule, CSA also does probabilistic ascents in the penalty subspace. The success of CSA lies in its strategy to search in the joint space, instead of applying SA to search in the subspace of the penalty function and updating the penalties in a separate phase of the algorithm. The latter approach would be taken in existing static and the dynamic penalty methods discussed in Section 2.1. CSA overcomes the limitations of existing penalty methods because it does not require a separate algorithm for choosing penalties. The rest of this section explains the steps of CSA [30, 28].

```
1.  procedure CSA
2.      set starting point z ← (z, α)^T and initialize α ← 0;
3.      set starting temperature T ← T_0 and cooling rate 0 < κ < 1;
4.      set N_T ← number of trials per temperature;
5.      while stopping condition is not satisfied do
6.          for k ← 1 to N_T do
7.              generate trial point z' ∈ N_m(z) using G(z, z');
8.              if z' is accepted according to A_T(z, z') then z ← z'
9.          end_for
10.         reduce temperature by T ← κ T;
11.     end_while
12. end_procedure
```

Figure 1. CSA: Constrained simulated annealing (see text for the initial values of the parameters). The differences between CSA and SA lie in their definitions of state $\mathbf{z}$, neighborhood $\mathsf{N}_m(\mathbf{z})$, generation probability $G(\mathbf{z}, \mathbf{z}')$ and acceptance probability $A_T(\mathbf{z}, \mathbf{z}')$.

Line 2 sets a starting point $\mathbf{z} \leftarrow (z, \alpha)^T$, where z can be either user-provided or randomly generated (such as using a fixed seed 123 in our experiments), and α is initialized to zero.
Line 3 initializes control parameter *temperature* T to be so large that almost any trial point $\mathbf{z}'$ will be accepted. In our experiments on continuous problems, we initialize T by first randomly generating 100 points of x and their corresponding neighbors

$x' \in \mathsf{N}_c(x)$ in close proximity, where $|x'_i - x_i| \leq 0.001$, and then setting $T = \max_{x,x',i}\left\{ |L_m((x',1)^T) - L_m((x,1)^T)|, |h_i(x)| \right\}$. Hence, we use a large initial T if the function is rugged ($|L_m((x',1)^T) - L_m((x,1)^T)|$ is large), or the function is not rugged but its constraint violation ($|h_i(x)|$) is large. We also initialize κ to 0.95 in our experiments.

Line 4 sets the number of iterations at each temperature. In our experiments, we choose $N_T \leftarrow \zeta(20n + m)$ where $\zeta \leftarrow 10(n + m)$, n is the number of variables, and m is the number of equality constraints. This setting is based on the heuristic rule in [9] using $n + m$ instead of n.

Line 5 stops CSA when the current $\mathbf{z}$ is not changed, *i.e.*, no other $\mathbf{z}'$ is accepted, in two successive temperature changes, or when the current T is small enough (*e.g.* $T < 10^{-6}$).

Line 7 generates a random point $\mathbf{z}' \in \mathsf{N}_m(\mathbf{z})$ from the current $\mathbf{z} \in \mathcal{S} = \mathcal{Z} \times \Lambda$,, where $\Lambda = \mathbb{R}^m$ is the space of the penalty vector. In our implementation, $\mathsf{N}_m(\mathbf{z})$ consists of $(z', \alpha)_T$ and $(z, \alpha')_T$, where $z' \in \mathcal{N}_{m_1}(z)$ (see Definition 1), and $\alpha' \in \mathcal{N}_{m_2}(\alpha)$ is a point neighboring to α when $h(z) \neq 0$:

$$\mathcal{N}_m(\mathbf{z}) = \left\{(z', \alpha)^T \in \mathcal{S} \text{ where } z' \in \mathcal{N}_{m_1}(z)\right\} \cup \left\{(z, \alpha')^T \in \mathcal{S} \text{ where } \alpha' \in \mathcal{N}_{m_2}(\alpha)\right\} \quad (21)$$

$$\text{and } \mathcal{N}_{m_2}(\alpha) = \left\{\alpha' \in \Lambda \text{ where } (\alpha'_i < \alpha_i \text{ or } \alpha'_i > \alpha_i \text{ if } h_i(z) \neq 0) \text{ and } (\alpha'_i = \alpha_i \text{ if } h_i(z) = 0)\right\}. \quad (22)$$

According to this definition, α_i is not perturbed when $h_i(z) = 0$ is satisfied.

$\mathsf{G}(\mathbf{z}, \mathbf{z}')$, the *generation probability* from $\mathbf{z}$ to $\mathbf{z}' \in \mathsf{N}_m(\mathbf{z})$, satisfies:

$$0 \geq G(\mathbf{z}, \mathbf{z}') \leq 1 \quad \text{and} \quad \sum_{\mathbf{z}' \in \mathcal{N}_m(\mathbf{z})} G(\mathbf{z}, \mathbf{z}') = 1. \quad (23)$$

Since the choice of $G(\mathbf{z}, \mathbf{z}')$ is arbitrary as long as it satisfies (23), we select $\mathbf{z}'$ in our experiments with uniform probability across all the points in $\mathsf{N}_m(\mathbf{z})$, independent of T:

$$G(\mathbf{z}, \mathbf{z}') = \frac{1}{|\mathcal{N}_m(\mathbf{z})|}. \quad (24)$$

As we perturb either z or α but not both simultaneously, (24) means that $\mathbf{z}'$ is generated either by choosing $z' \in \mathcal{N}_{m_1}(z)$ randomly or by generating α' uniformly in a predefined range.

Line 8 accepts $\mathbf{z}'$ with acceptance probability $A_T(\mathbf{z}, \mathbf{z}')$ that consists of two components, depending on whether z or α is changed in $\mathbf{z}'$:

$$A_T(\mathbf{z}, \mathbf{z}') = \begin{cases} \exp\left(-\frac{(L_m(\mathbf{z}') - L_m(\mathbf{z}))^+}{T}\right) & \text{if } \mathbf{z}' = (z', \alpha)^T \\ \exp\left(-\frac{(L_m(\mathbf{z}) - L_m(\mathbf{z}'))^+}{T}\right) & \text{if } \mathbf{z}' = (z, \alpha')^T. \end{cases} \quad (25)$$

The acceptance probability in (25) differs from the acceptance probability used in conventional SA, which only has the first case in (25) and whose goal is to look for a global minimum in the z subspace. Without the α subspace, only probabilistic descents in the z subspace are carried out.

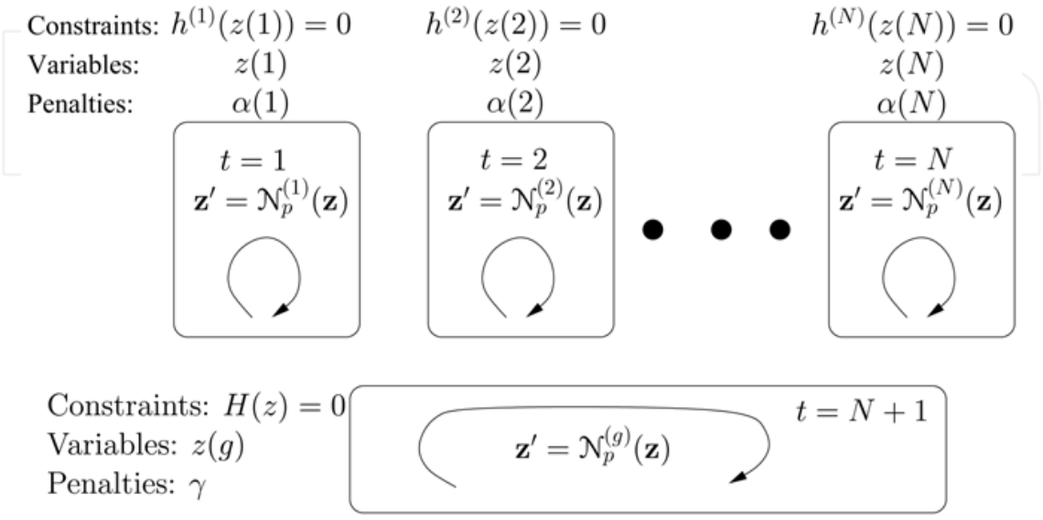


Figure 2. CPSA: Constraint-partitioned simulated annealing.

In contrast, our goal is to look for an ESP in the joint $\mathsf{Z} \times \Lambda$ space, each existing at a local minimum in the z subspace and at a local maximum in the α subspace. To this end, CSA carries out *probabilistic descents* of $L_m\big((z,\alpha)^T\big)$ with respect to z for each fixed α. That is, when we generate a new z' under a fixed α, we accept it with probability one when $\delta_z = L_m\big((z',\alpha)^T\big) - L_m\big((z,\alpha)^T\big)$is negative; otherwise, we accept it with probability $e^{-\delta_z/T}$. This step has exactly the same effect as in conventional SA; that is, it performs descents with occasional ascents in the z subspace.

However, descents in the z subspace alone will lead to a local/global minimum of the penalty function without satisfying the corresponding constraints. In order to satisfy all the constraints, CSA also carries out *probabilistic ascents* of $L_m\big((z,\alpha)^T\big)$ with respect to α for each fixed z in order to increase the penalties of violated constraints and to force them into satisfaction. Hence, when we generate a new α' under a fixed z, we accept it with probability one when $\delta_\alpha = L_m\big((z,\alpha')^T\big) - L_m\big((z,\alpha)^T\big)$ is positive; otherwise, we accept it with probability $e^{-\delta_\alpha/T}$. This step is the same as that in conventional SA when performing ascents with occasional descents in the α subspace. Note that when a constraint is satisfied, the corresponding penalty will not be changed according to (22).

Finally, Line 10 reduces T by the following *cooling schedule* after looping N_T times at given T:

$$T \longleftarrow \kappa \cdot T \quad \text{where the cooling-rate constant } \kappa \leftarrow 0.95 \text{ (typically } 0.8 \leq \kappa \leq 0.99). \tag{26}$$

At high T, (25) allows any trial point to be accepted with high probabilities, thereby allowing the search to traverse a large space and overcome infeasible regions. When T is reduced, the acceptance probability decreases, and at very low temperatures, the algorithm behaves like a local search.

3.2 Constraint-Partitioned Simulated Annealing (CPSA)

We present in this section CPSA, an extension of CSA that decomposes the search in CSA into multiple subproblems after partitioning the constraints into subsets. Recall that, according to Theorem 2, P_t in (14) can be partitioned into a sequence of N subproblems defined in (20) and an overall necessary condition defined in (19) on the global constraints across the subproblems, after choosing an appropriate mixed neighborhood. Instead of considering all the constraints together as in CSA, CPSA performs searches in multiple subproblems, each involving a small subset of the constraints. As in CSA, we only consider P_t with equality constraints.

Figure 2 illustrates the idea in CPSA. Unlike the original CSA that solves the problem as a whole, CPSA solves each subproblem independently. In Subproblem t, t = 1, …,N, CSA is performed in the $(z(t), \alpha(t))^T$ subspace related to the local constraints $h^{(t)}(z(t)) = 0$. In addition, there is a global search that explores in the $(z(g), \gamma)^T$ subspace on the global constraints $H(z) = 0$. This additional search is needed for resolving any violated global constraints.

```
1.  procedure CPSA
2.      set starting point z ← (z, α, γ)^T and initialize α = γ ← 0;
3.      set starting temperature T ← T^0 and cooling rate 0 < κ < 1;
4.      set N_T ← number of trials per temperature;
5.      while stopping condition is not satisfied do
6.          for k ← 1 to N_T do
7.              set t to be a random integer between 1 and N + 1;
8.              if 1 ≤ t ≤ N then
9.                  generate z' ∈ N_p^(t)(z) using G^(t)(z, z');
10.                 if z' is accepted according to A_T(z, z') then z ← z';
11.             else /* t = N + 1 */
12.                 generate z' ∈ N_p^(g)(z) using G^(g)(z, z');
13.                 if z' is accepted according to A_T(z, z') then z ← z';
14.             end_if
15.         end_for
16.         reduce temperature by T ← κ T;
17.     end_while
18. end_procedure
```

Figure 3. The CPSA search procedure.

Figure 3 describes the CPSA procedure. The first six lines are similar to those in CSA.

To facilitate the convergence analysis of CPSA in a Markov-chain model, Lines 7-14 randomly pick a subproblem for evaluation, instead of deterministically enumerating the subproblems in a round-robin fashion, and stochastically accept a new probe using an acceptance probability governed by a decreasing temperature. This approach leads to a memoryless Markovian process in CPSA.

Line 7 randomly selects Subproblem i, $i = 1 \ldots, N+1$, with probability $P_s(t)$, where $P_s(t)$ can be arbitrarily chosen as long as:

$$\sum_{t=1}^{N+1} P_s(t) = 1 \text{ and } P_s(t) > 0. \tag{27}$$

When t is between 1 and N (Line 8), it represents a local exploration step in Subproblem t. In this case, Line 9 generates a trial point $\mathbf{z}' \in \mathcal{N}_p^{(t)}(\mathbf{z})$ from the current point $\mathbf{z} = (z, \alpha, \gamma)^T \in \mathcal{S}$ using a *generation probability* $G^{(t)}(\mathbf{z}, \mathbf{z}')$ that can be arbitrary as long as the following is satisfied:

$$0 \le G^{(t)}(\mathbf{z}, \mathbf{z}') \le 1 \quad \text{and} \quad \sum_{\mathbf{z}' \in \mathcal{N}_p^{(t)}(\mathbf{z})} G^{(t)}(\mathbf{z}, \mathbf{z}') = 1. \tag{28}$$

The point is generated by perturbing $z(t)$ and $\alpha(t)$ in their neighborhood $\mathcal{N}_p^{(t)}(\mathbf{z})$:

$$\mathcal{N}_p^{(t)}(\mathbf{z}) = \left\{(z', \alpha(t), \gamma) \in \mathcal{S} \mid z' \in \mathcal{N}_{p_1}^{(t)}(z)\right\} \cup \left\{(z, \alpha'(t), \gamma) \in \mathcal{S} \mid \alpha'(t) \in \mathcal{N}_{p_2}^{(t)}(\alpha(t))\right\} \tag{29}$$

$$\begin{aligned} \mathcal{N}_{p_2}^{(t)}(\alpha(t)) = \big\{&\alpha'(t) \in \Lambda^{(\alpha(t))} \text{ where } (\alpha_i'(t) < \alpha_i(t) \text{ or } \alpha_i'(t) > \alpha_i(t) \text{ if } h_i(z(t)) \neq 0) \\ &\text{and } (\alpha_i'(t) = \alpha_i(t) \text{ if } h_i(z(t)) = 0)\big\}, \end{aligned} \tag{30}$$

and $\mathcal{N}_{p_1}^{(t)}(z)$ is defined in (15) and $\Lambda^{(\alpha(t))} = \mathbb{R}^{m_t}$. This means that $\mathbf{z}' \in \mathcal{N}_p^{(t)}(\mathbf{z})$ only differs from $\mathbf{z}$ in $z(t)$ or $\alpha(t)$ and remains the same for the other variables. This is different from CSA that perturbs $\mathbf{z}$ in the overall variable space. As in CSA, α_i is not perturbed when $h_i(z(t)) = 0$ is satisfied. Last, Line 10 accepts $\mathbf{z}'$ with the Metropolis probability $A^T(\mathbf{z}, \mathbf{z}')$ similar to that in (25):

$$A_T(\mathbf{z}, \mathbf{z}') = \begin{cases} \exp\left(-\frac{(L_m(\mathbf{z}') - L_m(\mathbf{z}))^+}{T}\right) & \text{if } \mathbf{z}' = (z', \alpha, \gamma)^T \\ \exp\left(-\frac{(L_m(\mathbf{z}) - L_m(\mathbf{z}'))^+}{T}\right) & \text{if } \mathbf{z}' = (z, \alpha', \gamma)^T \text{ or } \mathbf{z}' = (z, \alpha, \gamma')^T. \end{cases} \tag{31}$$

When $t = N + 1$ (Line 11), it represents a global exploration step. In this case, Line 12 generates a random trial point $\mathbf{z}' \in \mathcal{N}_p^{(g)}(\mathbf{z})$ using a generation probability $G^{(g)}(\mathbf{z}, \mathbf{z}')$ that satisfies the condition similar to that in (28). Assuming $\mathcal{N}_{m_1}(z(g))$ to be the mixed neighborhood of $z(g)$ and $\Lambda^{(g)} = \mathbb{R}^p$, $\mathbf{z}'$ is obtained by perturbing $z(g)$ and γ in their neighborhood $\mathcal{N}_p^{(g)}(\mathbf{z})$::

$$\mathcal{N}_p^{(g)}(\mathbf{z}) = \left\{(z', \alpha, \gamma)^T \in \mathcal{S} \text{ where } z' \in \mathcal{N}_{p_1}^{(g)}(z)\right\} \cup \left\{(z, \alpha, \gamma')^T \in \mathcal{S} \text{ where } \gamma' \in \mathcal{N}_{p_2}^{(g)}(\gamma)\right\} \tag{32}$$

$$\mathcal{N}_{p_1}^{(g)}(z) = \left\{z' \text{ where } z'(g) \in \mathcal{N}_{m_1}(z(g)) \text{ and } z_i' = z_i \ \forall z_i \notin z(g)\right\} \tag{33}$$

$$\mathcal{N}_{p_2}^{(g)}(\gamma) \;=\; \{\gamma' \in \Lambda^{(g)} \text{ where } (\gamma'_i < \gamma_i \text{ or } \gamma'_i > \gamma_i \text{ if } H_i(z) \neq 0) \text{ and } (\gamma'_i = \gamma_i \text{ if } H_i(z) = 0)\}. \quad (34)$$

Again, **z**′ is accepted with probability $A_T(\mathbf{z}, \mathbf{z}')$ in (31) (Line 13). Note that both $\mathcal{N}_p^{(t)}(\mathbf{z})(\mathbf{z})$ and $\mathcal{N}_p^{(g)}(\mathbf{z})$: ensure the ergodicity of the Markov chain, which is required for achieving asymptotic convergence.

When compared to CSA, CPSA reduces the search complexity through constraint partitioning. Since both CSA and CPSA need to converge to an equilibrium distribution of variables at a given temperature before the temperature is reduced, the total search time depends on the convergence time at each temperature. By partitioning the constraints into subsets, each subproblem only involves an exponentially smaller subspace with a small number of variables and penalties. Thus, each subproblem takes significantly less time to converge to an equilibrium state at a given temperature, and the total time for all the subproblems to converge is also significantly reduced. This reduction in complexity is experimentally validated in Section 5.

3.3 Greedy ESPC Search Method (GEM)

In this section, we present a dynamic penalty method based on a greedy search of an ESP. Instead of probabilistically accepting a probe as in CSA and CPSA, our greedy approach accepts the probe if it improves the value of the penalty function and rejects it otherwise.

One simple approach that does not work well is to gradually increase α^{**} until $\alpha^{**} > \alpha^{*}$, while minimizing the penalty function with respect to z using an existing local-search method. This simple iterative search does not always work well because the penalty function has many local minima that satisfy the second inequality in (13), but some of these local minima do not satisfy the first inequality in (13) even when $\alpha^{**} > \alpha^{*}$. Hence, the search may generate stationary points that are local minima of the penalty function but are not feasible solutions to the original problem.

To address this issue, Figure 4 shows a global search called the *Greedy ESPC Search Method* [32] (GEM). GEM uses the following penalty function:

$$L_g\big((z,\alpha)^T\big) = f(z) + \sum_{i=1}^{m} \alpha_i |h_i(z)| + \frac{1}{2}||h(z)||^2. \quad (35)$$

Lines 5-8 carries out Ng iterative descents in the z subspace. In each iteration, Line 6 generates a probe $z' \in \mathcal{N}_{m_1}(z)$ neighboring to z. As defined in (24) for CSA, we select z' with uniform probability across all the points in $\mathcal{N}_{m_1}(z)$. Line 7 then evaluates $L_g((z', \alpha)^T)$ and accepts z' only when it reduces the value of L_g. After the N_g descents, Line 9 updates the penalty vector α in order to bias the search towards resolving those violated constraints.

When α^{**} reaches its upper bound during a search but a local minimum of L_g does not correspond to a CLM_m of P_m, we can reduce α^{**} instead of restarting the search from a new starting point. The decrease will change the terrain of L_g and "lower" its barrier, thereby

allowing a local search to continue in the same trajectory and move to another local minimum of L_g. In Line 10, we reduce the penalty value of a constraint when its maximum violation is not reduced for three consecutive iterations. To reduce the penalties, Line 11 multiplies each element in α by a random real number uniformly generated between 0.4 to 0.6. By repeatedly increasing α** to its upper bound and by reducing it to some lower bound, a local search will be able to escape from local traps and visit multiple local minima of the penalty function. We leave the presentation of the parameters used in GEM and its experimental results to Section 5.

1. **procedure GEM**
2. set ϱ to be a positive real constant;
3. set starting point $\mathbf{z} \leftarrow (z, \alpha)^T$ and initialize α;
4. **repeat**
5. **for** k $\leftarrow$ 1 **to** N_g /* $N_g \leftarrow 20$, a positive integer in our experiments */
6. generate random trial point $z' \in \mathcal{N}_{m_1}(z)$;
7. **if** $(L_g((z,\alpha)^T) > L_g((z',\alpha)^T))$ **then** $z' \leftarrow z$; **end_if**
8. **end_for**
9. update $\alpha \longleftarrow \alpha + \varrho|h(z)|$;
10. **if** (condition to decrease α is satisfied) **then**
11. reduce α in order to allow the search to escape from local traps;
12. **end_if**
13. **until** stopping conditions are satisfied;
14. **end_procedure**

Figure 4. Greedy ESPC search method (GEM).

4. Asymptotic convergence of CSA and CPSA

In this subsection, we show the asymptotic convergence of CSA and CPSA to a constrained global minimum in *discrete* constrained optimization problems. Without repeating the definitions in Section 1, we can similarly define a discrete nonlinear programming problem (P_d), a discrete neighborhood ($N_d(y)$), a discrete constrained local minimum (CLM_d), a discrete constrained global minimum (CGM_d), and a penalty function in discrete space (L_d).

4.1 Asymptotic convergence of CSA

We first define the asymptotic convergence property. For a global minimization problem, let Ω be its search space, Ω_s be the set of all global minima, and $\omega(j) \in \Omega$, $j = 0, 1, \ldots$, be a sequence of points generated by an iterative procedure ψ until some stopping conditions hold.

Definition 8. Procedure ψ is said to have *asymptotic convergence to a global minimum*, or simply *asymptotic convergence* [2], if ψ converges with probability one to an element in Ω_s; that is, $\lim_{j\to\infty} P(\omega(j) \in \Omega_s) = 1$, independent of ω (0), where $P(w)$ is the probability of event w.

In the following, we first state the result on the asymptotic convergence of CSA to a CGM_d of P_d with probability one when T approaches 0 and when T is reduced according to a specific cooling schedule. By modeling CSA by an inhomogeneous Markov chain, we show that the

chain is strongly ergodic, that the chain minimizes an implicit virtual energy based on the framework of generalized SA (GSA) [24, 23], and that the virtual energy is at its minimum at any CGM_d. We state the main theorems without proofs [27] and illustrate the theorems by examples.

CSA can be modeled by an inhomogeneous Markov chain that consists of a sequence of homogeneous Markov chains of finite length, each at a specific temperature in a cooling schedule. Its *one-step transition probability matrix* is $P_T = [P_T(\mathbf{y}, \mathbf{y}')]$, where:

$$P_T(\mathbf{y},\mathbf{y}') = \begin{cases} G(\mathbf{y},\mathbf{y}')A_T(\mathbf{y},\mathbf{y}') & \text{if } \mathbf{y}' \in \mathcal{N}_d(\mathbf{y}) \\ 1 - \sum\limits_{\mathbf{y}'' \in \mathcal{N}_d(\mathbf{y})} G(\mathbf{y},\mathbf{y}'')A_T(\mathbf{y},\mathbf{y}'') & \text{if } \mathbf{y}' = \mathbf{y} \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

Example 2. Consider the following simple discrete minimization problem:

$$\begin{aligned} \min_{y} \quad & f(y) = -y^2 \\ \text{subject to} \quad & h(y) = |(y-0.6)(y-1.0)| = 0, \end{aligned} \tag{37}$$

where $y \in \mathsf{Y} = \{0.5,\ 0.6, \ldots,\ 1.2\}$. The corresponding penalty function is:

$$L_d\big((y,\alpha)^T\big) = -y^2 + \alpha \cdot |(y-0.6)(y-1.0)|. \tag{38}$$

By choosing $\alpha \in \Lambda = \{2,\ 3,\ 4,\ 5,\ 6\}$, with the maximum penalty value $\alpha^{\max}$ at 6, the state space is $\mathcal{S} = \{(y,\alpha)^T \in \mathcal{Y} \times \Lambda\}$ with $|\mathcal{S}| = 8 \cdot 5 = 40$ states. At $y = 0.6$ or $y = 1.0$ where the constraint is satisfied, we can choose $\alpha^* = 1$, and any $\alpha^{**} > \alpha^*$, including $\alpha^{\max}$, would satisfy (13) in Theorem 1.

In the Markov chain, we define $\mathsf{N}_d(\mathbf{y})$ as in (21), where $\mathcal{N}_{d_1}(y)$ and $\mathcal{N}_{d_2}(\alpha)$ are as follows:

$$\mathcal{N}_{d_1}(y) = \{y-0.1, y+0.1 |\ 0.6 \le y \le 1.1\} \cup \{y+0.1 |\ y = 0.5\} \cup \{y-0.1 |\ y = 1.2\}, \tag{39}$$

$$\begin{aligned} \mathcal{N}_{d_2}(\alpha) = {} & \{\alpha-1, \alpha+1 |\ 3 \le \alpha \le 5,\ y \ne 0.6 \text{ and } y \ne 1.0\} \cup \{\alpha-1 |\ \alpha = 6, \\ & y \ne 0.6 \text{ and } y \ne 1.0\} \cup \{\alpha+1 |\ \alpha = 2,\ y \ne 0.6 \text{ and } y \ne 1.0\}. \end{aligned} \tag{40}$$

Figure 5 shows the state space S of the Markov chain. In this chain, an arrow from $\mathbf{y}$ to $\mathbf{y}' \in \mathsf{N}_d(\mathbf{y})$ (where $\mathbf{y}' = (y',\ \alpha)^T$ or $(y,\ \alpha')^T$) means that there is a one-step transition from $\mathbf{y}$ to $\mathbf{y}$ whose $P_T(\mathbf{y},\ \mathbf{y}') > 0$. For $y = 0.6$ and $y = 1.0$, there is no transition among the points in the α dimension because the constraints are satisfied at those y values (according to (22)).

There are two ESPs in this Markov chain at $(0.6,\ 5)^T$ and $(0.6,\ 6)^T$, which correspond to the local minimum at $y = 0.6$, and two ESPs at $(1.0,\ 5)^T$ and $(1.0,\ 6)^T$, which correspond to the local minimum at $y = 1.0$. CSA is designed to locate one of the ESPs at $(0.6,\ 6)^T$ and $(1.0,\ 6)^T$. These correspond, respectively, to the CLM^d at $y^* = 0.6$ and $y^* = 1.0$. ■

Let $\mathbf{y}_{\text{opt}} = \{(y^*, \alpha^{\max})^T | \ y^* \in \mathcal{Y}_{\text{opt}}\}$, and N_L be the maximum of the minimum number of transitions required to reach $\mathbf{y}_{\text{opt}}$ from all $\mathbf{y} \in \mathcal{S}$. By properly constructing $\mathcal{N}_d(\mathbf{y})$, we state without proof that P_T is irreducible and that N_L can always be found. This property is illustrated in Figure 5 in which any two nodes can always reach each other.

Let N_T, the number of trials per temperature, be N_L. The following theorem states the strong ergodicity of the Markov chain, where strong ergodicity means that state $\mathbf{y}$ of the Markov chain has a unique stationary probability $\pi_T(\mathbf{y})$. (The proof can be found in the reference [27].)

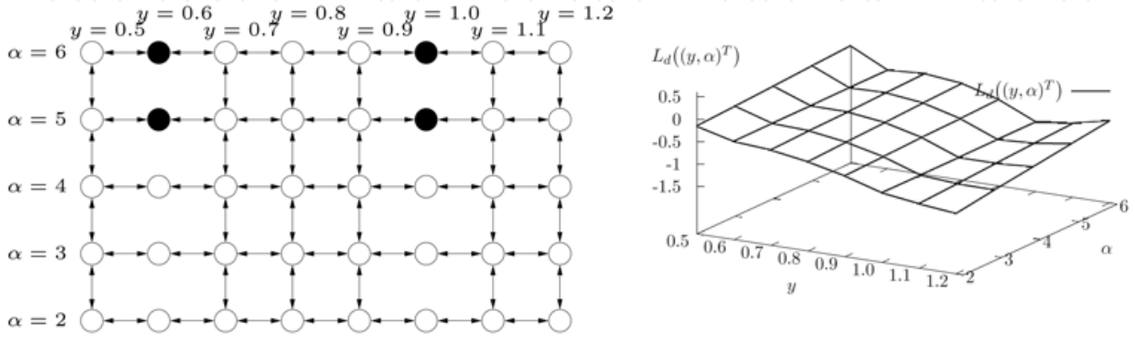


Figure 5. The Markov chain with the transition probabilities defined in (36) for the example problem in (37) and the corresponding penalty-function value at each state. The four ESPs are shaded in (a).

Theorem 3. The inhomogeneous Markov chain is *strongly ergodic* if the sequence of temperatures $\{T_k, k = 0, 1, 2, \ldots\}$ satisfies:

$$T_k \geq \frac{N_L \triangle_L}{log_e(k+1)}, \tag{41}$$

where $T_k > T_{k+1}$, $\lim_{k\to\infty} T_k = 0$, and $\triangle_L = 2 \max_{\mathbf{y}\in\mathcal{S},\mathbf{y}'\in\mathcal{N}_d(\mathbf{y})} \left\{ |L_d(\mathbf{y}') - L_d(\mathbf{y})| \right\}$.

Example 2 (cont'd). In the Markov chain in Figure 5, Δ_L = 0.411 and N_L = 11. Hence, the Markov chain is strongly ergodic if we use a cooling schedule $T_k \geq \frac{4.521}{log_e(k+1)}$. Note that the cooling schedule used in CSA (Line 10 of Figure 1) does not satisfy the condition.

Our Markov chain also fits into the framework of *generalized simulated annealing* (GSA) [24, 23] when we define an irreducible Markov kernel $P_T(\mathbf{y}, \mathbf{y}')$ and its associated *communication cost* $v(\mathbf{y}, \mathbf{y}')$, where v: $\mathcal{S} \times \mathcal{S} \to [0, +\infty]$ and $\mathbf{y}' \in \mathcal{N}_d(\mathbf{y})$:

$$v(\mathbf{y}, \mathbf{y}') = \begin{cases} (L_d(\mathbf{y}') - L_d(\mathbf{y}))^+ & \text{if } \mathbf{y}' = (y', \alpha)^T \\ (L_d(\mathbf{y}) - L_d(\mathbf{y}'))^+ & \text{if } \mathbf{y}' = (y, \alpha')^T. \end{cases} \tag{42}$$

Based on the communication costs over all directed edges, the *virtual energy* $W(\mathbf{y})$ (according to Definition 2.5 in [23, 24]) is the cost of the minimum-cost spanning tree rooted at $\mathbf{y}$:

$$W(\mathbf{y}) = \min_{g\in G(\mathbf{y})} V(g), \tag{43}$$

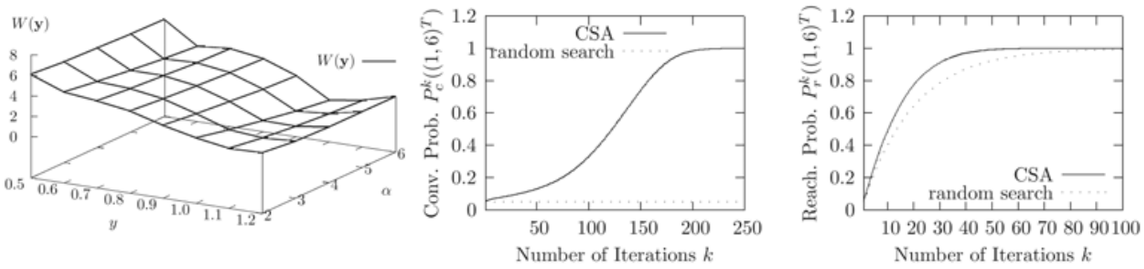
where $G(\mathbf{y})$ is the set of spanning trees rooted at $\mathbf{y}$, and $V(g)$ is the sum of the communication costs over all the edges of g.

The following quoted result shows the asymptotic convergence of GSA in minimizing $W(i)$:

Proposition 1 "(Proposition 2.6 in [14, 23, 24]). For every $T > 0$, the unique stationary distribution π_T of the Markov chain satisfies:

$$\pi_T(i) \longrightarrow \exp\left(-\frac{W(i) - W(E)}{T}\right) \quad \text{as} \quad T \longrightarrow 0, \tag{44}$$

where $W(i)$ is the virtual energy of i, and $W(E) = \min_{i \in \mathcal{S}} W(i)$."



a) Virtual energy $W(\mathbf{y})$ b) Convergence prob. at $(1, 6)^T$ c) Reachability prob. at $(1, 6)^T$

Figure 6. Virtual energy of the Markov chain in Figure 5a and the convergence behavior of CSA and random search at $(1.0, 6)^T$.

In contrast to SA that strives to minimize a single unconstrained objective, CSA does not minimize $L_d((y, \alpha)^T)$. This property is illustrated in Figure 5b in which the ESPs are not at the global minimum of $L_d((y, \alpha)^T)$. Rather, CSA aims to implicitly minimize $W(\mathbf{y})$ according to GSA [24, 23]. That is, $y^* \in \mathsf{Y}_{\text{opt}}$ corresponds to $\mathbf{y}^* = (y^*, \alpha^{\max})^T$ with the minimum $W(\mathbf{y})$, and $W((y^*, \alpha^{\max})^T) \prec W((y, \alpha)^T)$ for all $y \neq y^*$ and $\alpha \in \Lambda$ and for all $y = y^*$ and $\alpha \neq \alpha^{\max}$. The following theorem shows that CSA asymptotically converges to $\mathbf{y}^*$ with probability one. (See the proof in the reference [27].)

Theorem 4. Given the inhomogeneous Markov chain modeling CSA with transition probability defined in (36) and the sequence of decreasing temperatures that satisfy (41), the Markov chain converges to a CGM_d with probability one as $k \to \infty$.

Example 2 (cont'd). We illustrate the virtual energy $W(\mathbf{y})$ of the Markov chain in Figure 5a and the convergence behavior of CSA and random search.

One approach to find $W(\mathbf{y})$ that works well for a small problem is to enumerate all possible spanning trees rooted at $\mathbf{y}$ and to find the one with the minimum cost. Another more efficient way adopted in this example is to compute $W(\mathbf{y})$ using (44). This can be done by first numerically computing the stationary probability $\pi_T(\mathbf{y})$ of the Markov chain at a given T using the one-step transition probability $P_T(\mathbf{y}, \mathbf{y}')$ in (36), where π_T evolves with iteration k as follows:

$$P_c^{k+1} = P_c^k \, P_T \qquad \text{for any given initial convergence probability vector } P_c^0, \tag{45}$$

until $||P_c^{k+1} - P_c^k|| \leq \varepsilon$. In this example, we set $\varepsilon = 10^{-16}$ as the stopping precision. Since $\pi_T = \lim_{k\to\infty} P_c^k$, independent of the initial vector P_c^0, we set $P_c^0(i) = \frac{1}{|\mathcal{S}|}$ for $i = 1, \cdots, |\mathcal{S}|$.

Figure 6a shows $W((y,\ \alpha)^T)$ of Figure 5a. Clearly, $L^d((y,\ \alpha)^T) \neq W((y,\ \alpha)^T)$. For a given y, $W((y,\ \alpha)^T)$ is non-increasing as α increases. For example, $W((0.6,\ 3)^T) = 4.44 \geq W((0.6,\ 4)^T) = 4.03$, and $W((0.8,\ 2)^T) = 4.05 \geq W((0.8,\ 6)^T) = 3.14$. We also have $W((y,\ \alpha)^T)$ minimized at $y = 1.0$ when $\alpha = \alpha^{\max} = 6$: $W((0.6,\ 6)^T) = 3.37 \geq W((0.8,\ 6)^T) = 3.14 \geq W((1.0,\ 6)^T) = 0.097$. Hence, $W((y,\ \alpha)^T)$ is minimized at $(y^*,\ \alpha^{\max})^T = (1.0,\ 6)^T$, which is an ESP with the minimum objective value. In contrast, $L_d((y,\ \alpha)^T)$ is non-decreasing as α increases. In Figure 5b, the minimum value of $L_d((y,\ \alpha)^T)$ is at $(1.2,\ 2)^T$, which is not a feasible point.

To illustrate the convergence of CSA to $y^* = 1.0$, Figure 6b plots $P_c^k(\mathbf{y}^*)$ as a function of k, where $\mathbf{y}^* = (1.0,\ 6)^T$. In this example, we set $T_0 = 1.0$, $N_T = 5$, and $\kappa = 0.9$ (the cooling schedule in Figure 1). Obviously, as the cooling schedule is more aggressive than that in Theorem 3, one would not expect the search to converge to a CGM_d with probability one, as proved in Theorem 4. As T approaches zero, $W(\mathbf{y}^*)$ approaches zero, and $P_c^k(\mathbf{y}^*)$ monotonically increases and approaches one. Similar figures can be drawn to show that $P_c^k(\mathbf{y})$, $\mathbf{y} \neq \mathbf{y}^*$, decreases to zero as T is reduced. Therefore, CSA is more likely to find $\mathbf{y}^*$ as the search progresses. In contrast, for random search, $P_c^k(\mathbf{y}^*)$ is constant, independent of k.

Note that it is not possible to demonstrate asymptotic convergence using only a finite number of iterations. Our example, however, shows that the probability of finding a CGM_d improves over time. Hence, it becomes more likely to find a CGM_d when more time is spent to solve the problem.

Last, Figure 6c depicts the *reachability probability* $P_r^k(\mathbf{y}^*)$ of finding $\mathbf{y}^*$ in any of the first k iterations. Assuming all the iterations are independent, $P_r^k(\mathbf{y}^*)$ is defined as:

$$P_r^k(\mathbf{y}^*) = 1 - \prod_{i=0}^{k} \Big(1 - P(\mathbf{y}^* \text{ found in the } i^{\text{th}} \text{ iteration})\Big). \tag{46}$$

The figure shows that CSA has better reachability probabilities than random search over the 100 iterations evaluated, although the difference diminishes as the number of iterations is increased.

It is easy to show that CSA has *asymptotic reachability* [2] of $\mathbf{y}^*$; that is, $\lim_{k\to\infty} P_r^k(\mathbf{y}^*) = 1$. Asymptotic reachability is weaker than asymptotic convergence because it only requires the algorithm to hit a global minimum sometime during a search and can be guaranteed if the algorithm is ergodic. (Ergodicity means that any two points in the search space can be reached from each other with a non-zero probability.) Asymptotic reachability can be accomplished in any ergodic search by keeping track of the best solution found during the search. In contrast, asymptotic convergence requires the algorithm to converge to a global minimum with probability one. Consequently, the probability of a probe to hit the solution increases as the search progresses.

4.2 Asymptotic convergence of CPSA

By following a similar approach in the last section on proving the asymptotic convergence of CSA, we prove in this section the asymptotic convergence of CPSA to a CGM_d of P_d.

CPSA can be modeled by an inhomogeneous Markov chain that consists of a sequence of homogeneous Markov chains of finite length, each at a specific temperature in a given cooling schedule. The state space of the Markov chain can be described by state $\mathbf{y} = (y, \alpha, \gamma)^T$, where $y \in \mathcal{D}^w$, where $y \in \mathsf{D}^w$ is the vector of problem variables and α and γ are the penalty vectors.

According to the generation probability $G^{(t)}(\mathbf{y}, \mathbf{y}')$ and the acceptance probability $A^T(\mathbf{y}, \mathbf{y}')$, the *one-step transition probability matrix* of the Markov chain for CPSA is $P^T = [P^T(\mathbf{y}, \mathbf{y}')]$, where:

$$P_T(\mathbf{y},\mathbf{y}') = \begin{cases} P_s(t)G^{(t)}(\mathbf{y},\mathbf{y}')A_T(\mathbf{y},\mathbf{y}') & \text{if } \mathbf{y}' \in \mathcal{N}_p^{(t)}(\mathbf{y}),\ t = 1,\ldots,N \\ P_s(N+1)G^{(g)}(\mathbf{y},\mathbf{y}')A_T(\mathbf{y},\mathbf{y}') & \text{if } \mathbf{y}' \in \mathcal{N}_p^{(g)}(\mathbf{y}) \\ 1 - \sum_{t=1}^{N}\Big[\sum_{\mathbf{y}''\in\mathcal{N}_p^{(t)}(\mathbf{y})} P_T(\mathbf{y},\mathbf{y}'')\Big] - \sum_{\mathbf{y}''\in\mathcal{N}_p^{(g)}(\mathbf{y})} P_T(\mathbf{y},\mathbf{y}'') & \text{if } \mathbf{y}' = \mathbf{y} \\ 0 & \text{otherwise.} \end{cases} \tag{47}$$

Let $\mathbf{y}_{\text{opt}} = \{(y^*, \alpha^{\max}, \gamma^{\max})^T \mid y^* \in \mathcal{Y}_{\text{opt}}\}$, and N_L be the maximum of the minimum number of transitions required to reach $\mathbf{y}_{\text{opt}}$ from all $\mathbf{y} \in \mathcal{S}$. Given $\{T_k, k = 0, 1, 2, \ldots\}$ that satisfy (41) and N_T, the number of trials per temperature, be N_L, a similar theorem as in Theorem 3 can be proved [8]. This means that state $\mathbf{y}$ of the Markov chain has a unique stationary probability $\pi_T(\mathbf{y})$.

Note that Δ_L defined in Theorem 3 is the maximum difference between the penalty-function values of two neighboring states. Although this value depends on the user-defined neighborhood, it is usually smaller for CPSA than for CSA because CPSA has a partitioned neighborhood, and two neighboring states can differ by only a subset of the variables. In contrast, two states in CSA can differ by more variables and have larger variations in their penalty-function values. According to (41), a smaller Δ_L allows the temperature to be reduced faster in the convergence to a CGM_d.

Similar to CSA, (47) also fits into the framework of GSA if we define an irreducible Markov kernel $PT(\mathbf{y}, \mathbf{y}')$ and its associated communication cost $v(\mathbf{y}, \mathbf{y}')$, where $v: \mathcal{S} \times \mathcal{S} \to [0, +\infty]$:

$$v(\mathbf{y},\mathbf{y}') = \begin{cases} (L_d(\mathbf{y}') - L_d(\mathbf{y}))^+ & \text{if } \mathbf{y}' = (y', \alpha, \gamma)^T \\ (L_d(\mathbf{y}) - L_d(\mathbf{y}'))^+ & \text{if } \mathbf{y}' = (y, \alpha', \gamma)^T \text{ or } \mathbf{y}' = (y, \alpha, \gamma')^T. \end{cases} \tag{48}$$

In a way similar to that in CSA, we use the result that any process modeled by GSA minimizes an implicit virtual energy $W(\mathbf{y})$ and converges to the global minimum of $W(\mathbf{y})$ with probability one. The following theorem states the asymptotic convergence of CPSA to a CGM_d. The proof in the reference [27] shows that $W(\mathbf{y})$ is minimized at $(y^*, \alpha^{\max}, \gamma^{\max})^T$ for some $\alpha^{\max}$ and $\gamma^{\max}$.

Theorem 5. Given the inhomogeneous Markov chain modeling CPSA with transition probability defined in (47) and the sequence of decreasing temperatures that satisfy (41), the Markov chain converges to a CGM_d with probability one as $k \to \infty$.
Again, the cooling schedule of CPSA in Figure 3 is more aggressive than that in Theorem 5.

5. Experimental results on continuous constrained problems

In this section, we apply CSA and CPSA to solve some nonlinear continuous optimization benchmarks and compare their performance to that of other dynamic penalty methods. We further illustrate the application of the methods on two real-world applications.

5.1 Implementation details of CSA for solving dontinuous problems

In theory, any neighborhoods $\mathsf{N}_{c_1}(x)$ and $\mathsf{N}_{c_2}(\alpha)$ that satisfy (21) and (22) can be used. In practice, however, appropriate neighborhoods must be chosen in any efficient implementation.
In generating trial point $\mathbf{x}' = (x', \alpha)^T$ from $\mathbf{x} = (x, \alpha)^T$ where $x' \in \mathsf{N}_{c_1}(x)$, we choose x' to differ from x in the i^{th} element, where i is uniformly distributed in $\{1, 2, \ldots, n\}$:

$$x' = x + \theta \otimes \mathbf{e_1} = x + (\theta_1 e_{1,1}, \theta_2 e_{1,2}, \ldots, \theta_n e_{1,n})^T \tag{49}$$

and $\otimes$ is the vector-product operator. Here, $\mathbf{e}_1$ is a vector whose i^{th} element is 1 and the other elements are 0, and θ is a vector whose i^{th} element θ_i is Cauchy distributed with density $f_d(x_i) = \frac{1}{\pi}\frac{\sigma_i}{\sigma_i^2 + x_i^2}$ and scale parameter σ_i. Other distributions of θ_i studied include uniform and Gaussian [30]. During the course of CSA, we dynamically update σ_i using the following modified 1-to-1 rate rule [9] in order to balance the ratio between accepted and rejected configurations:

$$\sigma_i \longleftarrow \begin{cases} \frac{\sigma_i[1+\beta_0(p_i - p_u)]}{1-p_u} & \text{if } p_i > p_u \\ \frac{\sigma_i}{[1+\beta_1(p_v - p_i)/p_v]} & \text{if } p_i < p_v \\ \text{unchanged} & \text{otherwise,} \end{cases} \tag{50}$$

where p_i is the fraction of x' accepted. If p_i is low, then too many trial points of $\mathbf{x}'$ are rejected, and σ_i is reduced; otherwise, the trial points of $\mathbf{x}'$ are too close to $\mathbf{x}$, and σ_i is increased. We set $\beta_0 = 7$, $\beta_1 = 2$, $p_u = 0.3$, and $p_v = 0.2$ after experimenting different combinations of parameters [30]. Note that it is possible to get somewhat better convergence results when problem-specific parameters are used, although the results will not be general in that case.
Similarly, in generating trial point $\mathbf{x}'' = (x, \alpha')^T$ from $\mathbf{x} = (x, \alpha)^T$ where $\alpha' \in \mathsf{N}_{c_2}(\alpha)$, we choose α' to differ from α in the j^{th} element, where j is uniformly distributed in $\{1, 2, \ldots, m\}$:

$$\alpha' = \alpha + \nu \otimes \mathbf{e_2} = \alpha + (\nu_1 e_{2,1}, \nu_2 e_{2,2}, \ldots, \nu_m, e_{2,m})^T. \tag{51}$$

Here, the j_{th} element of $\mathbf{e}_2$ is 1 and the others are 0, and the ν_j is uniformly distributed in $[-\phi_j, \phi_j]$. We adjust ϕ_j according to the degree of constraint violations, where:

$$\phi = w \otimes h(x) = (w_1 h_1(x), w_2 h_2(x), \ldots, w_m h_m(x))^T. \tag{52}$$

When $h_i(x) = 0$ is satisfied, $\phi_i = 0$, and α_i does not need to be updated. Otherwise, we adjust φ_i by modifying w_i according to how fast $h_i(x)$ is changing:

$$w_i \longleftarrow \begin{cases} \eta_0\ w_i & \text{if } h_i(x) > \tau_0 T \\ \eta_1\ w_i & \text{if } h_i(x) < \tau_1 T \\ \text{unchanged} & \text{otherwise,} \end{cases} \tag{53}$$

where $\eta_0 = 1.25$, $\eta_1 = 0.95$, $\tau_0 = 1.0$, and $\tau_1 = 0.01$ were chosen experimentally. When $h_i(x)$ is reduced too quickly (*i.e.*, $h_i(x) < \tau_1 T$ is satisfied), $h_i(x)$ is over-weighted, leading to a possibly poor objective value or difficulty in satisfying other under-weighted constraints. Hence, we reduce α_i's neighborhood. In contrast, if $h_i(x)$ is reduced too slowly (*i.e.*, $h_i(x) > \tau_0 T$ is satisfied), we enlarge α_i's neighborhood in order to improve its chance of satisfaction. Note that w_i is adjusted using T as a referenc because constraint violations are expected to decrease when T decreases. Other distributions of ϕ_j studied include non-symmetric uniform and non-uniform [30].

Finally, we use the cooling schedule defined in Figure 1, which is more aggressive than that in (41). We accept the $\mathbf{x}'$ or $\mathbf{x}''$ generated according to the Metropolis probability defined in (25). Other probabilities studied include logistic, Hastings, and Tsallis [30]. We set the ratio of generating $\mathbf{x}'$ and $\mathbf{x}''$ from $\mathbf{x}$ to be $20n$ to m, which means that x is updated more frequently than α.

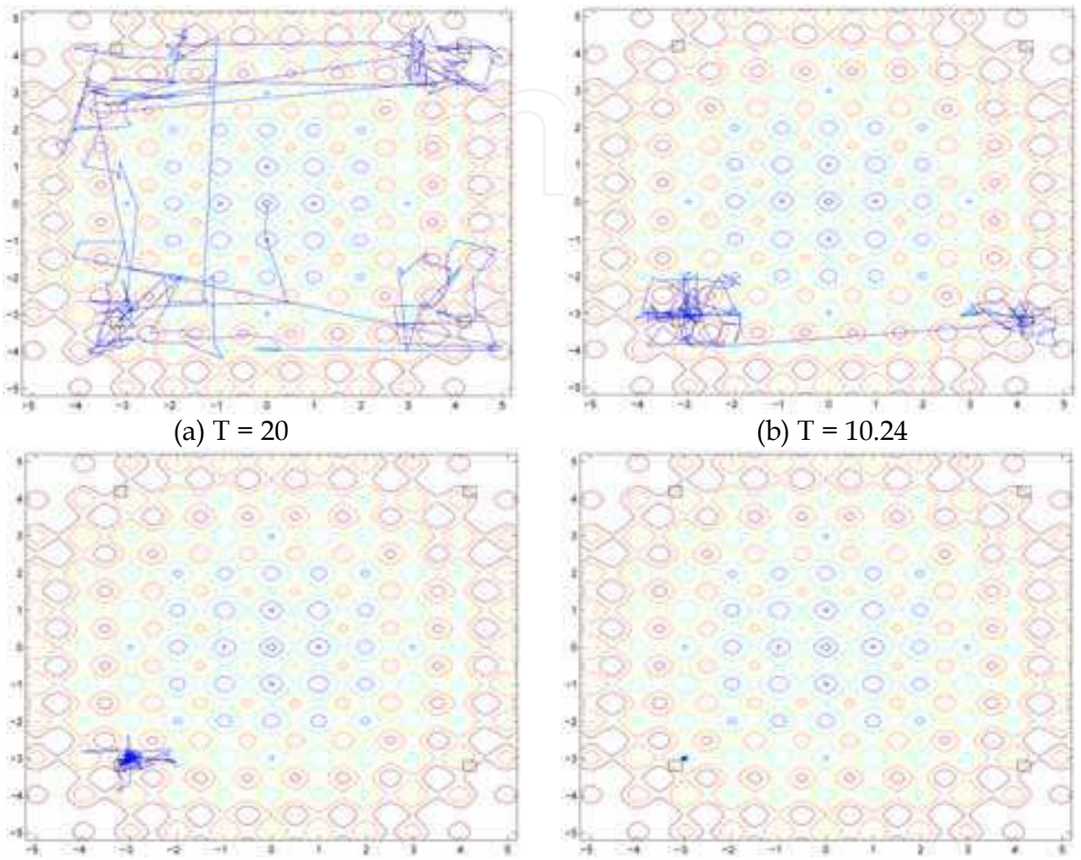
Example 3. Figure 7 illustrates the run-time behavior at four temperatures when CSA is applied to solve the following continuous constrained optimization problem:

$$\begin{aligned} \min_{x_1,\, x_2} \quad & f(x) = 10n + \sum_{i=1}^{2} \left(x_i^2 - 10\ \cos(2\pi x_i) \right) \quad \text{where } x = (x_1, x_2)^T \\ \text{subject to} \quad & |(x_i - 3.2)(x_i + 3.2)| = 0, \quad i = 1, 2. \end{aligned} \tag{54}$$

The objective function $f(x)$ is very rugged because it is made up of a two-dimensional Rastrigin function with 11^n (where $n = 2$) local minima. There are four constrained local minima at the four corners denoted by rectangles, and a constrained global minimum at $(-3.2, -3.2)$.

Assuming a penalty function $L_c((x,\ \alpha)^T) = f(x) + \alpha_1 |(x_1 - 3.2)(x_1 + 3.2)| + \alpha_2 |(x_2 - 3.2)(x_2 + 3.2)|$ and that samples in x are drawn in double-precision floating-point space, CSA starts from $x = (0, 0)^T$ with initial temperature $T_0 = 20$ and a cooling rate $\kappa = 0.95$. At high temperatures (*e.g.* $T_0 = 20$), the probability of accepting a trial point is high; hence, the neighborhood size is large according to (50). Large jumps in the x subspace in Figure 7a are due to the use of the Cauchy distribution for generating remote trial points, which increases the chance of getting

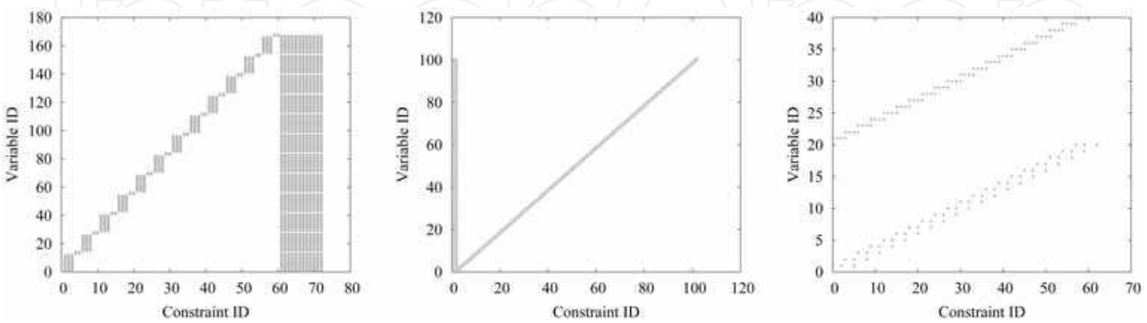
out of infeasible local minima. Probabilistic ascents with respect to α also help push the search trajectory to feasible regions. As T is reduced, the acceptance probability of a trial point is reduced, leading to smaller neighborhoods. Finally, the search converges to the constrained global minimum at $x^*= (-3.2, -3.2)^T$. ■



(a) T = 20 (b) T = 10.24

(c) T = 8.192 (d) T = 0.45

Figure 7. Example illustrating the run-time behavior of CSA at four temperatures in solving (54).



a) TRIMLOSS b) ORTHREGC c) OPTCDEG3

Figure 8. Strongly regular constraint-variable structures in some continuous optimization problems. A dot in each graph represents a variable associated with a constraint.

5.2 Implementation details of CPSA for solving continuous problems

We have observed that the constraints of many application benchmarks do not involve variables that are picked randomly from their variable sets. Invariably, many constraints in existing benchmarks are highly structured because they model spatial and temporal relationships that have strong locality, such as those in physical structures, optimal control, and staged processing.

Figure 8 illustrates this point by depicting the regular constraint structure of three benchmarks. It shows a dot where a constraint (with unique ID on the x axis) is related to a variable (with a unique ID on the y axis). When the order of the variables and that of the constraints are properly arranged, the figure shows a strongly regular constraint-variable structure.

In CPSA, we follow a previously proposed automated partitioning strategy [26] for analyzing the constraint structure and for determining how th constraints are to be partitioned. The focus of our previous work is to solve the partitioned subproblems using an existing solver SNOPT [15]. In contrast, our focus here is to demonstrate the improvement of CPSA over CSA and on their asymptotic convergence property.

Based on P_m with continuous variables and represented in AMPL [13], our partitioning strategy consists of two steps. In the first step, we enumerate all the indexing vectors in the AMPL model and select one that leads to the minimum R_{global}, which is the ratio of the number of global constraints to that of all constraints. We choose R_{global} as a heuristic metric for measuring the partitioning quality, since a small number of global constraints usually translates into faster resolution. In the second step, after fixing the index vector for partitioning the constraints, we decide on a suitable number of partitions. We have found a convex relationship between the number of partitions (N) and the complexity of solving P_m. When N is small, there are very few subproblems to be solved but each is expensive to evaluate; in contrast, when N is large, there are many subproblems to be solved although each is simple to evaluate. Hence, there is an optimal N that leads to the minimum time for solving P_m. To find this optimal N, we have developed an iterative algorithm that starts from a large N, that evaluates one subproblem under this partitioning (while assuming all the global constraints can be resolved in one iteration) in order to estimate the complexity of solving P_m, and that reduces N by half until the estimated complexity starts to increase. We leave the details of the algorithm to the reference [26].

Besides the partitioning strategy, CPSA uses the same mechanism and parameters described in Section 5.1 for generating trial points in the x, α, and γ subspaces.

5.3 Implementation details of GEM for solving continuous problems

The parameter in GEM were set based on the package developed by Zhe Wu and dated 08/13/2000 [32]. In generating a neighboring point of x for continuous problems, we use a Cauchy distribution with density $f_d(x_i) = \frac{1}{\pi}\frac{\sigma_i}{\sigma_i^2+x_i^2}$ for each variable x_i, $i = 1,\ldots, n$, where σ_i is a parameter controlling the Cauchy distribution. We initialize each σ_i to 0.1. For the last 50 probes that perturb x_i, if more than 40 probes lead to a decrease of L_m, we increase σ_i by a factor of 1.001; if less than two probes lead to a decrease of L_m, we decrease σ_i by a factor of 1.02. We increase the penalty σ_i for constraint h_i by $\alpha_i = \alpha_i + \varrho_i|h_i(x)|$, where ϱ_i is set to 0.0001 in our experiments. We consider a constraint to be feasible and stop increasing its penalty when its violation is less than 0.00001.

5.4 Evaluation results on continuous optimization benchmarks

Using the parameters of CSA and CPSA presented in the previous subsections and assuming that samples were drawn in double-precision floating-point space, we report in this section some experimental results on using CSA and CPSA to solve selected problems from CUTE [7], a constrained and unconstrained testing environment. We have selected those problems based on the criterion that at least the objective or one of the constraint functions is nonlinear. Many of those evaluated were from real applications, such as semiconductor analysis, chemical reactions, economic equilibrium, and production planning. Both the number of variables and the number of constraints in CUTE can be as large as several thousand.

Table 1 shows the CUTE benchmark problems studied and the performance of CPSA, CSA GEM in (35), P_3 in (7), and P_4 in (8). In our experiments, we have used the parameters of P_3 and P_4 presented in Section 2.2. For each solver and each instance, we tried 100 runs from random starting points and report the average solution found (Q_{avg}), the average CPU time per run of those successful runs (T_{avg}), the best solution found (Q_{best}), and the fraction of runs there were successful (P_{succ}). We show in shaded boxes the best Q_{avg} and Q_{best} among the five solvers when there are differences. We do not list the best solutions of P_3 and P_4 because they are always worse than those of CSA, CPSA, and GEM. Also, we do not report the results on those smaller CUTE instances with less than ten variables (BT*, AL*, HS*, MA*, NG*, TW*, WO*, ZE*, ZY*) [30] because these instances were easily solvable by all the solvers studied.

When compared to P_3, P_4, and GEM, CPSA and CSA found much better solutions on the average and the best solutions on most of the instances evaluated. In addition, CPSA and CSA have a higher success probability in finding a solution for all the instances studied.

The results also show the effectiveness of integrating constraint partitioning with CSA. CPSA is much faster than CSA in terms of T_{avg} for all the instances tested. The reduction in time can be more than an order of magnitude for large problems, such as ZAMB2-8 and READING6. CPSA can also achieve the same or better quality and success ratio than CSA for most of the instances tested. For example, for LAUNCH, CPSA achieves an average quality of 21.85, best quality of 9.01, and a success ratio of 100%, whereas CSA achieves, respectively, 26.94, 9.13, and 90%.

The nonlinear continuous optimization benchmarks evaluated in this section are meant to demonstrate the effectiveness of CSA and CPSA as dynamic penalty methods. We have studied these benchmarks because their formulations and solutions are readily available and because benchmarks on nonlinear discrete constrained optimization are scarce. These benchmarks, however, have continuous and differentiable functions and, therefore, can be solved much better by solvers that exploit such properties. In fact, the best solution of most of these problems can be found by a licensed version of SNOPT [15] (version 6.2) in less than one second of CPU time! In this respect, CSA and CPSA are not meant to compete with these solvers. Rather, CSA and CPSA are useful as constrained optimization methods for solving discrete, continuous, and mixed-integer problems whose constraint and objective functions are not necessarily continuous, differentiable, and in closed form. In these applications, penalty methods are invariably used as an effective solution approach. We illustrate in the following section the effectiveness of CSA for solving two real-world applications.

Problem	CPSA (with $\kappa = 0.8$ and 100 runs)				CSA (with $\kappa = 0.8$ and 100 runs)				GEM				P_3 Penalty Method				P_4 Penalty Method			
ID	Q_{avg}	Q_{best}	T_{avg}	P_{succ}	Q_{avg}	Q_{best}	T_{avg}	P_{succ}	Q_{avg}	Q_{best}	T_{avg}	P_{succ}	Q_{avg}	T_{avg}	P_{succ}		Q_{avg}	T_{avg}	P_{succ}	
AVION2	$9.47 \cdot 10^7$	$9.47 \cdot 10^7$	7.93	100%	$9.47 \cdot 10^7$	$9.47 \cdot 10^7$	204.74	100%	$9.47 \cdot 10^7$	$9.47 \cdot 10^7$	3213.49	100%	$9.47 \cdot 10^7$	2798	100%		$9.47 \cdot 10^7$	2578	100%	
BATCH	$2.14 \cdot 10^5$	$2.00 \cdot 10^5$	35.03	20%	-	-	-	-	$2.42 \cdot 10^5$	$1.94 \cdot 10^5$	13242.43	5%	–	–	–		–	–	–	
CRESC4	29.23	1.78	3.48	100%	34.24	1.98	7.37	100%	82.84	2.17	31.82	100%	82.84	31.82	100%		82.84	31.82	100%	
CSFI1	−38.87	-49.08	0.06	100%	−28.65	-49.08	1.24	100%	−17.34	-49.05	2.95	98%	−23.26	0.98	91%		−20.57	1.54	93%	
DEMBO7	174.80	174.79	2.58	100%	174.80	174.80	65.86	83%	174.80	174.80	232.16	57%	174.81	198.23	42%		174.81	203.64	40%	
DIPIGRI	680.63	680.63	0.17	100%	680.65	680.63	1.79	100%	680.64	680.63	11.18	100%	680.64	9.45	100%		680.64	14.38	100%	
DNIEPER	$1.87 \cdot 10^4$	$1.87 \cdot 10^4$	80.36	25%	-	-	-	-	$1.87 \cdot 10^4$	$1.87 \cdot 10^4$	3071.67	3%	–	–	–		–	–	–	
EXPFITA	1.20	0.06	8.15	100%	2.35	0.10	18.45	100%	1.24	1.12	60.18	100%	1.24	63.94	100%		1.26	76.18	100%	
FLETCHER	14.65	11.65	0.07	100%	4227.11	11.65	0.7	100%	20.28	11.72	3.75	100%	23.97	4.76	100%		23.76	6.42	100%	
GIGOMEZ2	1.95	1.95	0.02	50%	1.95	1.95	0.55	48%	1.95	1.95	9.67	50%	–	–	–		1.95	12.04	22%	
HIMMELBI	−1734.83	-1735.39	195.28	100%	−1735.55	−1735.57	*5091.47	100%	–	–	–	–	–	–	–		–	–	–	
HIMMELBJ	−1910.45	−1910.56	17.83	100%	−1909.98	-1910.33	*619.19	100%	−1909.39	-1910.05	3514.92	99%	−1904	2304.04	94%		−1908	1953.94	57%	
HIMMELP2	−62.05	-62.05	0.02	100%	−62.05	-62.05	0.44	100%	−62.05	−62.05	0.95	97%	−62.05	0.95	89%		−62.05	0.95	19%	
HIMMELP6	−59.01	-59.01	0.04	100%	−59.01	-59.01	0.62	100%	−59.01	−59.01	1.58	100%	−59.01	2.34	100%		−59.01	1.77	100%	
HONG	22.53	22.53	0.06	100%	22.53	22.53	0.82	100%	22.57	22.57	8.68	100%	22.57	8.9	100%		22.57	10.3	100%	
HUBFIT	0.017	$1.69 \cdot 10^{-2}$	0.02	100%	0.017	$1.69 \cdot 10^{-2}$	0.36	100%	0.017	$1.69 \cdot 10^{-2}$	14.73	100%	0.017	15.62	100%		0.017	16.07	100%	
LAUNCH	21.85	9.01	12.33	100%	26.94	9.13	*495.89	90%	22.80	9.01	1205.03	87%	24.583	1403.40	40%		24.54	1543.04	34%	
LIN	−0.02	-0.02	0.1	100%	−0.02	-0.02	1.21	100%	−0.019	−0.02	3.02	100%	−0.019	3.01	100%		−0.019	3.21	100%	
LOADBAL	76.35	0.78	6.34	100%	100.71	33.53	*226.07	100%	13.40	2.66	2350.60	100%	13.45	2454.65	100%		14.54	1934.34	100%	
LOOTSMA	1.41	1.41	0.04	100%	1.41	1.41	0.52	100%	1.41	1.41	1.56	100%	1.41	2.75	100%		1.41	2.43	100%	
MESH	-10^5	-10^5	17.37	100%	-10^5	-10^5	*625.03	100%	-10^5	-10^5	14453.76	100%	-10^5	14560	100%		-10^5	12139	100%	
MISTAKE	−1.00	-1.00	0.44	100%	−1.00	-1.00	11.18	100%	−1.00	−1.00	63.84	90%	−1.00	70.48	45%		−1.00	77.74	98%	
MRIBASIS	30.11	21.52	31.27	100%	31.04	29.32	1031.34	100%	31.56	31.22	24345.34	100%	34.03	20434	90%		–	–	–	
MWRIGHT	2564.35	1.53	0.04	100%	12029.65	1.53	0.6	100%	$1.3 \cdot 10^5$	35.83	9.83	100%	$1.4 \cdot 10^5$	11.84	100%		$2 \cdot 10^5$	10.34	100%	
ODFITS	−2225.33	-2379.4	5.56	100%	−1442.48	-2379.4	6.95	100%	9393.45	2699.04	21.15	100%	9497.43	20.48	100%		10320.3	24.32	100%	
OPTCNTRL	549.61	549.49	12.36	100%	549.61	549.49	103.34	100%	550.00	550.00	1376.45	100%	550.00	1487.73	100%		550.00	1432.54	100%	
OPTPRLOC	−16.42	-16.42	6.23	100%	−16.42	-16.42	296.49	100%	−16.42	−16.42	1381.46	100%	−16.42	872.43	100%		−16.42	787.42	100%	
PENTAGON	0.00	0.00	0.4	100%	0.00	0.00	6.92	100%	0.00	0.00	47.32	93%	0.00	49.38	75%		0.00	36.34	94%	
POLAK5	50.00	50.00	0.03	100%	50.00	50.00	0.43	100%	50.00	50.00	1.68	100%	50.00	1.83	100%		50.00	1.98	100%	
QC	−998.64	−1018.09	0.33	100%	−970.19	-1007.35	5.77	100%	−763.80	−956.14	29.56	100%	−743.65	30.56	100%		−743.22	23.49	100%	
READING6	−58.45	−105.45	202.86	100%	−54.71	-97.3	3059.25	100%	−66.12	−94.72	26027*	100%	−66.33	29820*	100%		−68.12	30223*	100%	
RK23	25906.32	13016.09	0.88	39%	29614.39	16928.29	7.45	13%	–	–	–	–	–	–	–		–	–	–	
ROBOT	5.51	5.46	0.21	100%	5.47	5.46	4.86	100%	5.46	5.46	38.85	100%	5.46	40.66	100%		5.46	38.95	100%	
S316-322	334.13	334.13	0.01	100%	334.13	334.13	0.25	100%	334.30	334.30	1.20	100%	334.30	1.24	100%		334.30	1.50	100%	
SINROSNB	0.00	0.00	0.02	100%	0.00	0.00	0.4	100%	–	–	–	–	–	–	–		–	–	–	
SNAKE	0.00	0.00	0.02	100%	79.12	0.00	0.38	100%	267.08	0.00	2.45	100%	268.54	2.56	100%		269.18	2.48	100%	
SPIRAL	360.21	0.00	0.05	100%	360.86	0.00	0.88	100%	505.70	0.00	2.70	100%	512.56	2.72	100%		505.80	2.67	100%	
STANCMIN	4.29	4.25	0.03	100%	4.25	4.25	0.63	100%	4.25	4.25	2.53	100%	4.25	2.57	100%		4.25	2.54	100%	
SVANBERG	15.73	15.73	0.78	100%	15.73	15.73	16.2	100%	–	–	–	–	–	–	–		–	–	–	
SYNTHES1	2.76	0.76	0.13	100%	2.33	0.76	2.2	100%	2.61	0.76	18.93	100%	2.98	19.23	100%		2.86	13.43	100%	
SYNTHES2	−0.56	-0.56	0.77	100%	−0.56	-0.56	17.63	100%	−0.55	-0.55	95.36	100%	−0.55	92.98	100%		−0.55	92.21	100%	
SYNTHES3	15.08	15.08	4.85	100%	15.09	15.08	48.54	100%	15.08	15.08	336.45	100%	15.08	458.92	100%		15.08	492.3	100%	
TENBARS4	1586.97	1586.97	11.54	77%	2566.82	509.5	15.55	9%	–	–	–	–	–	–	–		–	–	–	
ZAMB2-8	−0.13	-0.15	364.06	100%	1.35	-0.15	6247.57	83%	–	–	–	–	–	–	–		–	–	–	

* Only ten runs were made for these problems due to the extensive CPU time required for each run.

Table 1. Experimental results comparing CPSA, CSA, GEM, P_3, and P_4 in solving selected nonlinear continuous problems from CUTE. Each instance was solved by a solver 100 times from random starting points. The best Q_{avg} (*resp.* Q_{best}) among the five solvers are shown in shaded boxes. '−'means that no feasible solution was found in a time limit of 36,000 sec. All runs were done on an AMD Athlon MP2800 PC with RH Linux AS4.

5.5 Applications of CSA on two real-world applications

Sensor-network placement optimization. The application involves finding a suitable placement of sensors in a wireless sensor network (WSN) [31, 33]. Given N sensors, the problem is to find their locations that minimize the false alarm rate, while maintaining a minimum detection probability for every point in a 2-D region A [34]:

$$\begin{aligned} \min \quad & P_F \\ \text{subject to} \quad & P_D(x,y) \geq \beta, \quad \forall (x,y) \in A, \end{aligned} \tag{55}$$

where $P_D(x,\ y)$ denotes the detection probability of location $(x,\ y)$, and P_F denotes the false alarm rate over all locations in A. To compute P_D and P_F, we need to first compute the local detection probability P_{D_i} and the local false alarm rate P_{F_i} for each sensor i, $i = 1,\ldots,N$, as follows:

$$P_{F_i} = \int_{b_i}^{\infty} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{z^2}{2\sigma^2} \right) dz, \tag{56}$$

$$P_{D_i}(x,y) = \int_{b_i}^{\infty} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{z - \sqrt{\frac{e}{d((x,y),(x_i,y_i))^a}}}{2\sigma^2} \right) dz. \tag{57}$$

The probabilistic model is based on a Gaussian noise assumption [34], where b_i, σ, a, and e are constants, and $(x_i,\ y_i)$ is the coordinates of the i^{th} sensor. After all the local decisions have been sent to a fusion center, the center will find that an event happens at $(x,\ y)$ if a majority of the sensors have reported so. Therefore, we have the following equations:

$$P_D(x,y) = \sum_{|S_1|>|S_0|} \prod_{i \in S_0} (1 - P_{D_i}(x,y)) \prod_{j \in S_1} P_{D_j}(x,y), \tag{58}$$

$$P_F = \sum_{|S_1|>|S_0|} \prod_{i \in S_0} (1 - P_{F_i}) \prod_{j \in S_1} P_{F_j}, \tag{59}$$

where S_0 and S_1 denote the set of nodes that, respectively, detect or do not detect an event.

The functions in the above formulation are very expensive to evaluate. In fact, the cost for computing $P_D(x,\ y)$ is $\Theta(2_n)$, since we need to consider all combinations of S_0 and S_1. The cost is so expensive that it is impossible to directly compute $P_D(x,\ y)$ or its derivatives. Thus, the problem has no closed form and without gradient information. Instead, a Monte-Carlo simulation is typically used to estimate $P_D(x,\ y)$ within reasonable time [34]. Previous work in WSN have solved this problem using some greedy heuristic methods that are ad-hoc and suboptimal [34].

We have applied CSA to solve (55) and have found it to yield much better solutions than existing greedy heuristics [34]. In our approach, we find the minimum number of sensors by a binary search that solves (55) using multiple runs of CSA. For example, in a 20 ×20 grid,

CSA can find a sensor placement with only 16 sensors to meet the given thresholds of $P_D \geq$ 95% and $P_F \leq 5\%$, while a previous heuristic method [34] needs 22 sensors. In a 40×40 grid, CSA can find a sensor placement with only 28 sensors to meet the same constraints, while the existing heuristic method [34] needs 43 sensors.

Synthesis of out-of-core algorithms. A recent application uses CSA to optimize the out-of-core code generation for a special class of imperfectly nested loops encoding tensor contractions that arise in quantum chemistry computation [19]. In this task, the code needs to execute some large, imperfectly nested loops. These loops operate on arrays that are too large to fit in the physical memory. Therefore, the problem is to find the optimal tiling of the loops and the placement of disk I/O statements.

Given the abstract code, the loop ranges, and the memory limit of the computer, the out-of-core code-generation algorithm first enumerates all the feasible placements of disk read/write statements for each array. To find the best combination of placements of all arrays, a discrete constrained nonlinear optimization problem is formulated and provided as input to CSA.

The variables of the problem include tile sizes and the placement variables. The constraints include the input-array constraints, which specify that the read statement for an input array can only be placed for execution before the statement where it is consumed. They also include the input-output-array constraints, which specify that the write statement for an output array can only be placed after the statement where it is produced. Lastly, there are a number of other intermediate-array constraints.

Experimental measurements on sequential and parallel versions of the generated code show that the solutions generated by CSA consistently outperform previous sampling approach and heuristic equal-tile-size approach. When compared to previous approaches, CSA can reduce the disk I/O cost by a factor of up to four [19].

6. Conclusions

We have reported in this chapter constrained simulated annealing (CSA) and constraint-partitione simulated annealing (CPSA), two dynamic-penalty methods for finding constrained global minima of discrete constrained optimization problems. Based on the theory of extended saddle points (ESPs), our methods look for the local minima of a penalty function when the penalties are larger than some thresholds and when the constraints are satisfied. To reach an ESP, our methods perform probabilistic ascents in the penalty subspace, in addition to probabilistic descents in the problem-variable subspace as in conventional simulated annealing (SA). Because both methods are based on sampling the search space of a problem during their search, they can be applied to solve continuous, discrete, and mixed-integer optimization problems without continuity and differentiability.

Based on the decomposition of the ESP condition into multiple necessary conditions [25], we have shown that many benchmarks with highly structured and localized constraint functions can be decomposed into loosely coupled subproblems that are related by a small number of global constraints. By exploiting constraint partitioning, we have demonstrated that CPSA can significantly reduce the complexity of CSA.

We have shown the asymptotic convergence of CSA and CPSA to a constrained global minimum with probability one. The result is theoretically important because it extends SA, which guarantees asymptotic convergence in discrete unconstrained optimization, to that in

discrete constrained optimization. Moreover, it establishes a condition under which optimal solutions can be found in constraint-partitioned nonlinear optimization problems.
Lastly, we illustrate the effectiveness of CSA and CPSA for solving some nonlinear benchmarks and two real-world applications. CSA and CPSA are particularly effective when the constraint and objective functions and their gradients are too expensive to be evaluated or are not in closed form.

7. References

E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. J. Wiley and Sons, 1989.

S. Anily and A Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Appl. Prob.*, 24:657–667, 1987.

A. Auslender, R. Cominetti, and M. Maddou. Asymptotic analysis for penalty and barrier methods in convex and linear programming. *Mathematics of Operations Research*, 22:43–62, 1997.

T. Back, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *Proc. of the 4th Int'l Conf. on Genetic Algorithms*, pages 2–9, 1991.

J. C. Bean and A. B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. In *Tech. Rep. TR 92-53, Dept. of Industrial and Operations Engineering, The Univ. of Michigan*, 1992.

D. P. Bertsekas and A. E. Koksal. Enhanced optimality conditions and exact penalty functions. *Proc. of Allerton Conf.*, 2000.

I. Bongartz, A. R. Conn, N. Gould, and P. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Trans. on Mathematical Software*, 21(1):123–160, 1995.

Y. X. Chen. *Solving Nonlinear Constrained Optimization Problems through Constraint Partitioning*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, September 2005.

A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. on Mathematical Software*, 13(3):262–280, 1987.

J. P. Evans, F. J. Gould, and J. W. Tolle. Exact penalty functions in nonlinear programming. *Mathematical Programming*, 4:72–97, 1973.

R. Fletcher. A class of methods for nonlinear programming with termination and convergence properties. In J. Abadie, editor, *Integer and Nonlinear Programming*. North-Holland, Amsterdam, 1970.

R. Fletcher. An exact penalty function for nonlinear programming with inequalities. *Tech. Rep. 478, Atomic Energy Research Establishment, Harwell*, 1972.

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks Cole Pub. Co., 2002.

M. I. Freidlin and A. D. Wentzell. *Random perturbations of dynamical systems*. Springer, 1984.

P. E. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. on Optimization*, 12:979–1006, 2002.

A. Homaifar, S. H-Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In *Proc. of the First IEEE Int'l Conf. on Evolutionary Computation*, pages 579–584, 1994.

S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

S. Krishnan, S. Krishnamoorthy, G. Baumgartner, C. C. Lam, J. Ramanujam, P. Sadayappan, and V. Choppella. Efficient synthesis of out-of-core algorithms using a nonlinear optimization solver. Technical report, Dept. of Computer and Information Science, Ohio State University, Columbus, OH, 2004.

A. Kuri. A universal eclectric genetic algorithm for constrained optimization. In *Proc. 6th European Congress on Intelligent Techniques and Soft Computing*, pages 518–522, 1998.

D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1984.

R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.

A. Trouve. Rough large deviation estimates for the optimal convergence speed exponent of generalized simulated annealing algorithms. Technical report, LMENS-94-8, Ecole Normale Superieure, France, 1994.

A. Trouve. Cycle decomposition and simulated annealing. *SIAM Journal on Control and Optimization*, 34(3):966–986, 1996.

B. Wah and Y. X. Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, 170(3):187–231, 2006.

B. W.Wah and Y. X. Chen. Solving large-scale nonlinear programming problems by constraint partitioning. In *Proc. Principles and Practice of Constraint Programming, LCNS-3709*, pages 697–711. Springer-Verlag, October 2005.

B. W. Wah, Y. X. Chen, and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained optimization. *J. of Global Optimization*, 39:1–37, 2007.

B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 461–475. Springer-Verlag, October 1999.

B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 28–42. Springer- Verlag, October 1999.

T. Wang. *Global Optimization for Constrained Nonlinear Programming*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, December 2000.

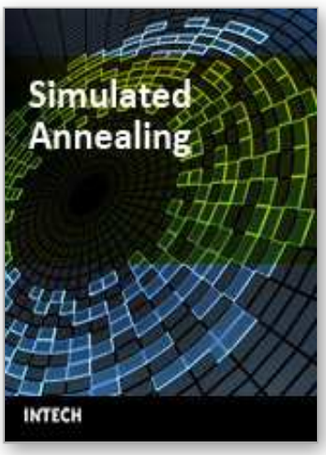
X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, , and C. Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *Proc. First ACM Conf. on Embedded Networked Sensor Systems*, pages 28–39, 2003.

Z. Wu. *The Theory and Applications of Nonlinear Constrained Optimization using Lagrange Multipliers*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 2001.

G. Xing, C. Lu, R. Pless, , and Q. Huang. On greedy geographic routing algorithms in sensing-covered networks. In *Proc. ACM Int'l Symp. on Mobile Ad Hoc Networking and Computing*, pages 31–42, 2004.

G. Xing, C. Lu, R. Pless, and J. A. O'Sullivan. Co-Grid: An efficient coverage maintenance protocol for distributed sensor networks. In *Proc. Int'l Symp. on Information Processing in Sensor Networks*, pages 414–423, 2004.

W. I. Zangwill. Nonlinear programming via penalty functions. *Management Science*, 13:344–358, 1967.



Simulated Annealing

Edited by Cher Ming Tan

ISBN 978-953-7619-07-7
Hard cover, 420 pages
Publisher InTech
Published online 01, September, 2008
Published in print edition September, 2008

This book provides the readers with the knowledge of Simulated Annealing and its vast applications in the various branches of engineering. We encourage readers to explore the application of Simulated Annealing in their work for the task of optimization.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Benjamin W. Wah, Yixin Chen and Tao Wang (2008). Theory and Applications of Simulated Annealing for Nonlinear Constrained Optimization, Simulated Annealing, Cher Ming Tan (Ed.), ISBN: 978-953-7619-07-7, InTech, Available from: http://www.intechopen.com/books/simulated_annealing/theory_and_applications_of_simulated_annealing_for_nonlinear_constrained_optimization

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen