

Comparison of Simulated Annealing, Interval Partitioning and Hybrid Algorithms in Constrained Global Optimization

Chandra Sekhar Pedamallu¹ and Linet Özdamar²

¹ Nanyang Technological University, School of Mechanical and Aerospace Engineering, Singapore.

On overseas attachment to:

University of Szeged, Institute of Informatics, Szeged, Hungary.

pcs.murali@gmail.com

² Izmir University of Economics, Sakarya Cad. No.156, 35330, Balçova, Izmir, Turkey.

Fax : 90(232)279 26 26

linetozdamar@lycos.com, lozdamar@gmail.com

Abstract

The continuous Constrained Optimization Problem (COP) often occurs in industrial applications. In this study, we compare three novel algorithms developed for solving the COP. The first approach consists of an Interval Partitioning Algorithm (IPA) that is exhaustive in covering the whole feasible space. IPA has the capability of discarding sub-spaces that are sub-optimal and/or infeasible, similar to available Branch and Bound techniques. The difference of IPA lies in its use of Interval Arithmetic rather than conventional bounding techniques described in the literature. The second approach tested here is the novel dual-sequence Simulated Annealing (SA) algorithm that eliminates the use of penalties for constraint handling. Here, we also introduce a hybrid algorithm that integrates SA in IPA (IPA-SA) and compare its performance with stand-alone SA and IPA algorithms. All three methods have a local COP solver, Feasible Sequential Quadratic Programming (FSQP) incorporated so as to identify feasible stationary points. The performances of these three methods are tested on a suite of COP benchmarks and the results are discussed.

Key words: Constrained Global Optimization, Interval Partitioning Algorithms, Simulated Annealing, Hybrid Algorithms

1 Introduction

Many important real world problems can be expressed in terms of a set of nonlinear constraints that restrict the real domain over which a given performance criterion is optimized, that is, as a Constrained Optimization Problem (COP). The COP is ex-

pressed as: *minimize* $f(\mathbf{x})$: $\mathbf{x} = (x_1, \dots, x_n)^t \in \xi \subset \mathbb{R}^n$ where ξ is the feasible domain. ξ is defined by k inequality constraints ($g_i(\mathbf{x}) \leq 0, i = 1, \dots, k$), $(m - k)$ equality constraints ($h_i(\mathbf{x}) = 0, i = k + 1, \dots, m$) and domain lower and upper bounds ($LB_x \leq \mathbf{x} \leq UB_x$). The expressions $g(\mathbf{x})$ and $h(\mathbf{x})$ may involve nonlinear and linear relations. The objective function, $f(\mathbf{x})$, is minimized by an optimum solution vector $\mathbf{x}^* = (x_1, \dots, x_n)^t \in \xi \subset \mathbb{R}^n$ where $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \xi$. In the general COP with a non-convex objective function (and a general non-convex feasible domain), discovering the location of the global optimum is NP-hard. Derivative based solution approaches developed for solving the COP might often be trapped in infeasible and/or sub-optimal sub-spaces if the combined topology of the constraints is too rugged.

Existing global optimization algorithms designed to solve the COP can be categorized as deterministic and stochastic methods. Surveys on global optimization are abundant in the literature (see the recent one [34]). Examples of deterministic approaches are: Lipschitzian methods [37]; branch and bound methods [1]; reformulation techniques [41]; interior point methods [12]; Branch and Reduce (BARON [39] and interval methods [15], [19]. An extensive list of up to date references of stochastic approaches for the COP is maintained by Coello Coello (<http://www.cs.cinvestav.mx/~constraint/>) where evolutionary approaches, genetic algorithms, ant colony approaches, simulated annealing and many other techniques are cited for continuous and discrete problems.

Here we focus on the Interval Partitioning Algorithms (IPA) and Simulated Annealing (SA) to solve the COP. IPA is a branch and bound technique that uses inclusion functions. Similar to the branch and bound technique, IPA is complete and reliable in the sense that it explores the whole feasible domain and discards sub-spaces in the feasible domain only if they are guaranteed to exclude feasible solutions and/or local stationary points better than the ones already found. On the other hand, SA is a black box stochastic algorithm that generates a sequence of random solutions converging to a global optimum. SA employs a slow annealing process that accepts worse solutions more easily in the beginning stages of the search as compared to later phases [22]. Using this feature, SA escapes from local optima and overcomes the difficulties encountered by derivative based numerical methods. A convergence proof for SA in the real domain is provided in [8]. We now discuss the available literature on SA and IPA designed for the COP.

As Hedar and Fukushima [17] also mention in their report, publications concerning the implementation of SA in the COP are rather scarce. Some successful special case SA applications for constrained engineering problems exist in the literature (e.g., structural optimization problems [2], [25], SA combined with genetic algorithms in economic dispatch [47], in power generator scheduling [48], [49], [50], in thermoelastic scaling behavior [51]). There has also been theoretical work conducted related to the use of SA in the general COP. For instance, Wah and Wang [44], [45] introduce a new penalty method where penalty parameters are also perturbed by SA. Hedar and Fukushima [17] apply multi-start SA from solutions that are preferably pareto

optimal according to the infeasibility and optimality criteria. The latter technique aims at achieving a better exploration in both feasible and infeasible regions. Similar to constrained SA applications, interval research on the COP is also relatively scarce when compared with bound constrained optimization. Hansen and Sengupta [16] discuss the inequality COP whereas Ratschek and Rokne [38] describe interval techniques for the COP with both inequality and equality constraints. Numerical results using these techniques are published later in Wolfe [46] and Kearfott [20]. Dallwig et al. [6] propose the software (GLOPT) for solving bound constrained optimization and the COP where a new reduction technique is proposed. More recently, Kearfott [21] presents a software named GlobSol and Markot [26] develops an IPA for the COP with inequalities where new adaptive multi-section rules and a new box selection criterion are presented [27]. Here, we propose a deterministic IPA algorithm having an adaptive tree search management approach that coordinates calls to a local solver, Feasible Sequential Quadratic Programming (FSQP – [52], [24]). In IPA, FSQP is activated within the confinement of each sub-space stored in the list of boxes to be explored. The second approach proposed is the dual-sequence SA, DSA, where the sequences of infeasible and feasible solutions are traced separately. In each SA iteration, a feasible candidate neighbor is compared with the last feasible solution obtained in the feasible sequence, and similarly an infeasible one is compared with the last infeasible solution. Thus, two sequences are constructed in parallel, and, the problems (e.g., the magnitude of penalty parameters) encountered by penalty methods are avoided. This approach is different from other approaches. For instance in Hedar and Fukushima's algorithm [17], each new sequence started from a non-dominated solution is a single sequence. The diversification scheme implemented in DSA is also much simpler and requires minimal memory space. DSA also incorporates FSQP as a local solver, but it has its own invoking policy. In the third approach proposed here, we create a hybrid IPA-DSA algorithm by integrating DSA into IPA where DSA works within the confinement of the specific sub-domain to be explored. This time, FSQP is invoked by DSA. This hybrid approach targets the total coverage of the search domain while enabling a global search in the sub-domains explored. Further, the total search space is reduced by IPA's reliable elimination of infeasible and sub-optimal sub-spaces. All three approaches are compared using a test suite of COP benchmarks. In the next sections we provide the basics of Interval Arithmetic, and brief descriptions of IPA, DSA and IPA-DSA.

2 Basics of Interval Arithmetic

Denote the real numbers by x, y, \dots , the set of compact intervals by $\mathbb{I} := \{[a, b] \mid a \leq b; a, b \in \mathbb{R}\}$ and the set of n dimensional intervals (also called simply intervals or boxes) by \mathbb{I}^n . Italic letters will be used for intervals. Every interval $x \in \mathbb{I}$ is denoted by $[\underline{x}, \bar{x}]$, where its bounds are defined by $\underline{x} = \inf x$ and $\bar{x} = \sup x$. For every $a \in \mathbb{R}$, the interval point $[a, a]$ is also denoted by a . The width of an interval x is the real number $w(x) = \bar{x} - \underline{x}$. Given two real intervals x and y , x is said to be tighter than y if $w(x) < w(y)$.

Given $(x_1, \dots, x_n) \in \mathbb{I}$, the corresponding box \mathbf{x} is the Cartesian product of intervals, $\mathbf{x} = x_1 \times \dots \times x_n$, where $\mathbf{x} \in \mathbb{I}^n$. A subset of \mathbf{x} , $\mathbf{y} \subseteq \mathbf{x}$, is a sub-box of \mathbf{x} . The notion of width is defined as follows:

$$w(x_1 \times \dots \times x_n) = \max_{1 \leq i \leq n} w(x_i) \quad \text{and} \quad w(x_i) = \bar{x}_i - \underline{x}_i \quad (1)$$

Interval Arithmetic operations are set theoretic extensions of the corresponding real operations. Given $x, y \in \mathbb{I}$, and an operation $\diamond \in \{+, -, \times, \div\}$, we have: $x \diamond y = \{x \diamond y \mid x \in x, y \in y\}$.

Due to properties of monotonicity, these operations can be implemented by real computations over the bounds of intervals. Given two intervals $x = [a, b]$ and $y = [c, d]$, we have:

$$\begin{aligned} [a, b] + [c, d] &= [a + c, b + d], \\ [a, b] - [c, d] &= [a - d, b - c], \\ [a, b] \times [c, d] &= [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}], \\ [a, b] \div [c, d] &= [a, b] \times [1/d, 1/c] \quad \text{if } 0 \notin [c, d]. \end{aligned}$$

The associative law and the commutative law are preserved over these operations, however, the distributive law does not hold. In general, only a weaker law is verified, called subdistributivity.

Interval arithmetic is particularly appropriate to represent outer approximations of real quantities. The range of a real function f over an interval x is denoted by $f(x)$, and it can be computed by interval extensions.

Definition 1. (*Interval extension*): An interval extension of a real function $f : D_f \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is a function $F : \mathbb{I}^n \rightarrow \mathbb{I}$ such that $\forall \mathbf{x} \in \mathbb{I}^n, \mathbf{x} \in D_f \Rightarrow f(\mathbf{x}) = \{f(x) \mid x \in \mathbf{x}\} \subseteq F(\mathbf{x})$.

Interval extensions are also called *interval forms* or *inclusion functions*. This definition implies the existence of infinitely many interval extensions of a given real function. In a proper implementation of interval extension based inclusion functions the outward rounding must be made to be able to provide a mathematical strength reliability.

The most common extension is known as the natural extension. Natural extensions are obtained from the expressions of real functions, and are inclusion monotonic (this property follows from the monotonicity of interval operations). Hence, given a real function f , whose natural extension is denoted by F , and two intervals x and y such that $x \subseteq y$, the following holds:

$F(x) \subseteq F(y)$. We denote the lower and upper bounds of the function interval range over a given box y as $\underline{F}(y)$ and $\bar{F}(y)$, respectively.

Here, it is assumed that for the studied COP, the natural interval extensions of f , g and h over \mathbf{x} are defined in the real domain. Furthermore, (F and similarly, G and H) is α -convergent over \mathbf{x} , that is, for all $\mathbf{y} \subseteq \mathbf{x}$, $w(F(\mathbf{y})) - w(f(\mathbf{y})) \leq cw(\mathbf{y})^\alpha$ where c and α are positive constants.

An *interval constraint* is built from an interval function and a relation symbol, which is extended to intervals. A constraint being defined by its expression (atomic formula and relation symbol), its variables, and their domains, we will consider that an interval constraint has interval variables (variables that take interval values), and that each associated domain is an interval.

The main feature of interval constraints is that if its solution set is empty, i.e., it has no solution over a given box y , then it follows that the solution set of the COP is also empty and the box y can be reliably discarded. In a similar manner, if the upper bound of the objective function range, $\bar{F}(y)$, over a given box y is less than or equal to the objective function value of a known feasible solution (the Current Lower Bound, CLB), then y can be reliably discarded since it cannot contain a better solution than the CLB.

Below we formally provide the conditions where a given box y can be discarded reliably based on the ranges of interval constraints and the objective function.

In a partitioning algorithm, each box y is assessed for its optimality and feasibility status by calculating the ranges for F , G , and H over the domain of y .

Definition 2. (*Cut-off test based on optimality:*) If $\bar{F}(y) < CLB$, then box y is called a sub-optimal box.

Definition 3. (*Cut-off test based on feasibility:*) If $\underline{G}_i(y) > 0$, or $0 \notin H_i(y)$ for any i , then box y is called an *infeasible* box.

Definition 4. If $\bar{F}(y) \leq CLB$, and $\bar{F}(y) > CLB$, then y is called an *indeterminate* box with regard to optimality. Such a box holds the potential of containing x^* if it is not an infeasible box.

Definition 5. If $\underline{G}_i(y) < 0$, and $\bar{G}_i(y) > 0$, or $(0 \in H_i(y) \neq 0)$ for some i , and other constraints are consistent over y , then y is called an *indeterminate* box with regard to feasibility and it holds the potential of containing x^* if it is not a sub-optimal box.

The IPA described in the following section uses the feasibility and optimality cut-off tests in discarding boxes reliably and sub-divides indeterminate boxes repetitively until either they are discarded or they are small enough (these boxes have a potential of holding x^* and finally the local solver identifies it). However, at certain points in this process, available indeterminate boxes are occasionally (given that some conditions hold) subjected to local search before they reach the tolerance size. The latter is undertaken to speed up convergence.

3 The Interval Partitioning Algorithm (IPA)

Under reasonable assumptions, IPA is a reliable convergent algorithm that sub-divides indeterminate boxes to reduce the uncertainties related to feasibility and optimality by nested partitioning. In terms of subdivision direction selection (the

choice of the variables to partition in a given indeterminate box), convergence depends on whether the direction selection rule is balanced [5]. The contraction and the α -convergence properties enable this. Here, the rule that selects the variable with the widest variable domain is utilized (Rule A) for this purpose. The reduction in the uncertainty levels of boxes finally lead to their elimination due to sub-optimality or infeasibility while helping IPA in ranking remaining indeterminate boxes in a better fashion.

A box that becomes feasible after nested partitioning still has uncertainty with regard to optimality unless it is proven that it is sub-optimal. The convergence rate of IPA might be very slow if we require nested partitioning to reduce a box to a point interval that is the global optimum. Hence, we need to use the local search procedure FSQP that might also identify stationary points in larger boxes. FSQP calls are coordinated by a special adaptive tree search procedure that is developed in Pedamallu et al. [36].

The adaptive tree management system maintains a stage-wise branching scheme that is conceptually similar to the iterative deepening approach [23]. The iterative deepening approach explores all nodes generated at a given tree level (stage) before it starts assessing the nodes at the next stage. Exploration of boxes at the same stage can be done in any order, the sweep may start from best-first box or the one on the most right or most left of that stage (depth-first). On the other hand, in the proposed adaptive tree management system, a node (parent box) at the current stage is permitted to grow a sub-tree forming partial succeeding tree levels and nodes in this sub-tree are explored before exhausting the nodes at the current stage. In the proposed IPA, if a feasible solution (CLB) is not identified yet, boxes in the sub-tree are ranked according to descending total constraint infeasibility, otherwise they are ranked in ascending order of $\underline{F}(\mathbf{y})$. A box is selected among the children of the same parent according to either box selection criterion, and the child box is partitioned again continuing to build the same sub-tree. This sub-tree grows until the Total Area Deleted (TAD) by discarding boxes fails to improve in two consecutive partitioning iterations in this sub-tree. Such failure triggers a call to FSQP for all boxes that have not been previously subjected to local search. The boxes that have undergone local search are placed back in the list of pending boxes and exploration is resumed among the nodes at the current stage. Feasible and improving solutions found by FSQP are stored (that is, if a feasible solution with a better objective function value is found, CLB is updated and the solution is stored).

The above adaptive tree management scheme is achieved by maintaining two lists of boxes, B_s and B_{s+1} that are the lists of boxes to be explored at the current stage s and at the next stage $s + 1$, respectively. Initially, the set of indeterminate or feasible boxes in the pending list B_s consists only of \mathbf{x} and B_{s+1} is empty. As child boxes are added to a selected parent box, they are ordered according to the current ranking criterion. Boxes in the sub-tree stemming from the selected parent at the current stage are explored and partitioned until there is no improvement in TAD in two consecutive partitioning iterations. At that point, partitioning of the selected parent box is stopped and all boxes that have not been processed by local search are sent to FSQP module and processed to identify feasible and improving point solutions if FSQP is

successful in doing so.¹ From that moment onwards, child boxes generated from any other selected parent in B_s are stored in B_{s+1} irrespective of further calls to FSQP in the current stage. When all boxes in B_s have been assessed (discarded or partitioned), the search moves to the next stage, $s + 1$, starting to explore the boxes stored in B_{s+1} .

The tree continues to grow in this manner taking up the list of boxes of the next stage after the current stage's list of boxes is exhausted. The algorithm stops either when there are no boxes remaining in B_s and B_{s+1} or when there is no improvement in CLB as compared with the previous stage. The proposed IPA algorithm is described below.

IP with adaptive tree management

Step 0. Set tree stage, $s = 1$. Set future stage, $r = 1$. Set non-improvement counter for TAD: $nc = 0$. Set B_s , the list of pending boxes at stage s equal to \mathbf{x} , $B_s = \{\mathbf{x}\}$, and $B_{s+1} = \emptyset$.

Step 1. If the number of function evaluations or CPU time reaches a given limit, or, if both $B_s = \emptyset$ and $B_{s+1} = \emptyset$, then STOP.

Else, if $B_s = \emptyset$ and $B_{s+1} \neq \emptyset$, then set $s \leftarrow s + 1$, set $r \leftarrow s$, and continue. Pick the first box \mathbf{y} in B_s and continue.

1.1 If \mathbf{y} is infeasible or suboptimal, discard \mathbf{y} , and go to Step 1.

1.2 Else if \mathbf{y} is sufficiently small, evaluate m , its mid-point, and if it is a feasible improving solution, update CLB, reset $nc \leftarrow 0$, and store m . Remove \mathbf{y} from B_s and go to Step 1.

Step 2. Select variable(s) to partition (sort variables according to descending width and select first v variables whose widths exceed the average width of candidate variables).

Step 3. Partition \mathbf{y} into 2^v non-overlapping child boxes. Check TAD, if it improves, then reset $nc \leftarrow 0$, else set $nc \leftarrow nc + 1$.

Step 4. Remove \mathbf{y} from B_s , add 2^v boxes to B_r .

4.1. If $nc > 2$, apply FSQP to all (previously unprocessed by FSQP) boxes in B_s and B_{s+1} , reset $nc \leftarrow 0$. If FSQP is called for the first time in stage s , then set $r \leftarrow s + 1$. Go to Step 1.

4.2. Else, go to Step 1.

¹ It should be noted that, whether or not FSQP fails to find an improving solution, IPA will continue to partition indeterminate boxes as long as they pass both cutoff tests. Finally, the algorithm encloses potential improving solutions in sufficiently small boxes where FSQP can identify them.

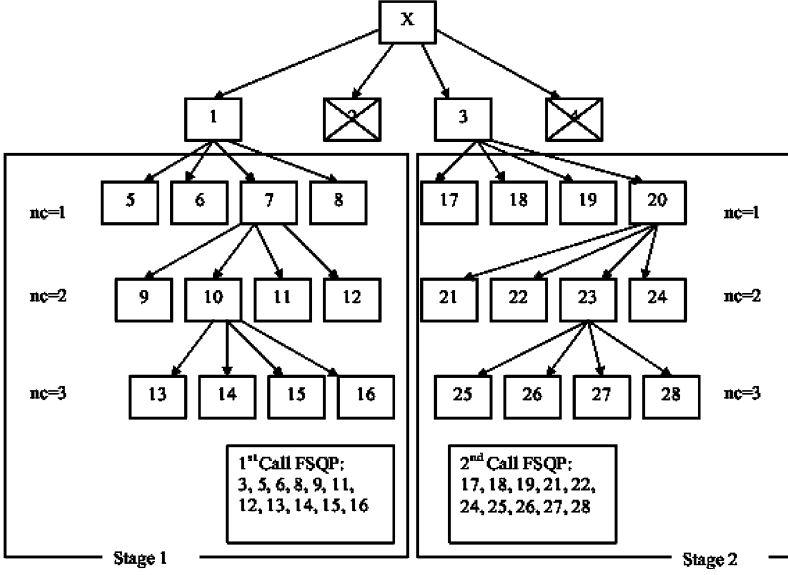


Fig. 1 Implementation of the adaptive iterative deepening procedure

The adaptive tree management system in IP is illustrated in Fig. 1 on a small tree where node labels indicate the order of nodes visited.

4 The Dual-sequence SA Algorithm: DSA

In contrast to the deterministic exhaustive IPA, DSA is a stochastic global search algorithm that does not necessarily cover the whole search space within a finite number of iterations. DSA is shown to be superior to single sequence SA applications where constraint handling is carried out via penalty functions [35]. DSA also utilizes FSQP to identify stationary points nearby good solutions. Before we describe DSA algorithm, we provide notation related to infeasibility degrees of constraints.

We denote the degree of infeasibility of constraint i at a given solution vector \mathbf{x} by $INF_i(\mathbf{x})$ and define it below for infeasible equality and inequality constraints.

$$INF_i(\mathbf{x}) = \begin{cases} 0 & \text{if } h_i(\mathbf{x}) = 0 \text{ or } g_i(\mathbf{x}) \leq 0; \\ |h_i(\mathbf{x})| & \text{or } g_i(\mathbf{x}) \text{ otherwise.} \end{cases} \quad (2)$$

Total infeasibility degree of a solution \mathbf{x} is given as: $TIF(\mathbf{x}) = \sum_{i=1}^m INF_i(\mathbf{x})$.

In Fig. 2 we provide the pseudocode of DSA and the relevant notation. DSA starts with a random initial sample, so called “seed”, within the hypercube defined by $[LB_x, UB_x]$. The iteration counter of DSA is denoted as q . In each iteration, a candidate solution, \mathbf{x}^q , that is a neighbor to \mathbf{x}^{q-1} is generated by perturbing the value of a selected coordinate of \mathbf{x}^{q-1} within the coordinate lower and upper bounds. This is

Notation and Functions used in DSA:

q : iteration counter
 q_max : maximum allowed number of iterations for DSA
 Gq : global counter for consecutive number of non-improving feasible solutions
 Gq_max : maximum allowable value of Gq before new sequence is started
 Lq : local counter for consecutive number of non-improving feasible or infeasible solutions
 Lq_max : maximum allowable value of Lq before accepting a worse solution probabilistically
 x^q : solution vector obtained by perturbing x^{q-1} .
 $temp$: annealing temperature.
 t_f : minimum temperature allowed.
 $feas_obj^q$: $f(x)$ of the last feasible solution encountered until iteration q .
 f^* : best feasible $f(x)$ found until iteration q .
Perturb(x^q): Function that selects a coordinate j of the solution vector x^{q-1} and increases or decreases x_j randomly in the interval $[LB_j, UB_j]$.
Paccept(x^q): Function that calculates acceptance probability for x^q . (if a randomly generated float number in $[0,1]$ is less than acceptance probability, it reports that x^q is acceptable.)
Re-anneal: Set $temp$ to one.

Procedure DSA

Generate a starting solution for a new sequence;

```

Do {
  If ( $Gq > Gq\_max$ ) {
    /* Diversification I */
    Generate a starting solution for a new sequence;
     $x^{q-1} \leftarrow$  new seed;
    If ( $x^{q-1}$  is feasible)  $feas\_obj^{q-1} \leftarrow f(x^{q-1})$ ;  $TIF^{q-1} \leftarrow \infty$ ;
    Else  $TIF^{q-1} \leftarrow TIF(x^{q-1})$ ;  $feas\_obj^{q-1} \leftarrow \infty$ ;
    Reset  $Gq, Lq \leftarrow 0$ ; Re-anneal;
  }
   $x^q \leftarrow Perturb(x^{q-1})$ ;
  If ( $x^q$  is feasible) {
    Set  $Fq \leftarrow Fq + 1$ ;
    If ( $Fq > Fq\_max$ ) Reset  $Fq \leftarrow 0$ ;  $TIF^q \leftarrow \infty$ ; /* Diversification II */
    Else  $TIF^q \leftarrow TIF^{q-1}$ ; /* Update inf. sequence information */
     $feas\_obj^q \leftarrow f(x^q)$ ; /* Update feas. sequence information */
    If ( $f(x^q) < f^*$ ) Update  $f^*$ ; Reset  $Gq \leftarrow 0$ ; /* continue feasible sequence */
    Else  $Gq \leftarrow Gq + 1$ ; /* increase counter for Diversification I */
    If ( $f(x^q) < feas\_obj^{q-1}$ ) Invoke probabilistic FSQP call;
  } /* endif */
  Else {
     $TIF^q \leftarrow TIF(x^q)$ ; /* Update inf. sequence information */
     $feas\_obj^q \leftarrow feas\_obj^{q-1}$ ; /* Update feas. sequence information */
  }
  If ( $TIF^q < TIF^{q-1}$ ) OR ( $feas\_obj^q < feas\_obj^{q-1}$ ) Reset  $Lq \leftarrow 0$ ; /* better sol. */
  Else { /* worse sol. */
     $Lq \leftarrow Lq + 1$ ; /* increase intensification counter */
    Invoke Paccept( $x^q$ ); /* calculate prob. of acceptance */
    If ( $Lq > Lq\_max$ ) AND ( $x^q$  is accepted) { /* accept. of worse sol. enabled */
      Reset  $Lq \leftarrow 0$ ; /* reset intensification counter */
      Reduce  $temp$ ;
      If ( $temp < t_f$ ) Re-anneal;
      If ( $q \approx q\_max$ ) Invoke FSQP call; /* FSQP call near worse solution */
    } /* endif */
    Else  $x^q \leftarrow x^{q-1}$ ; /* preserve old solution */
  } /* endelse */
   $q \leftarrow q + 1$ ;
} while ( $q \leq q\_max$ );
  
```

Fig.2 Pseudocode of DSA

achieved by procedure Perturb. In DSA, every solution is classified so as to belong to the feasible or infeasible sequence.

In each iteration, the information regarding both sequences is updated according to the neighbor's feasibility status. If \mathbf{x}^q is feasible, then, $feas_obj^q$ becomes equal to $f(\mathbf{x}^q)$ and TIF^q takes its old value TIF^{q-1} since there is no change in the infeasible sequence. Otherwise, if \mathbf{x}^q is infeasible, TIF^q is set equal to $TIF(\mathbf{x}^q)$ and $feas_obj^q$ takes its old value. These are indicated in Fig. 2 by *update sequence information* comments.

Diversification in DSA: In DSA, we introduce two diversification schemes. These are indicated as comments in Fig. 2. The first scheme (Diversification I) is controlled by a counter, G_q that stops a sequence of solutions once it stagnates. Stagnation occurs when the best feasible solution found so far, f^* , does not improve during a long sequence of moves. Thus, G_q records the number of consecutive non-improving feasible solutions and it is reset to zero whenever a feasible objective value lower than f^* , is encountered. When G_q exceeds the limit G_{q_max} , ($G_{q_max} = 0.1 \cdot q_max$), the procedure generates a new seed, which is not a neighbor to the previous solution, \mathbf{x}^{q-1} . In other words, a new sequence of solutions is started, replaces \mathbf{x}^{q-1} , and all parameters are re-initialized. The temperature $temp$ is re-annealed, G_q is set to 0. According to the feasibility status of the new seed, $feas_obj^{q-1}$ and TIF^{q-1} are re-set.

The second diversification scheme (Diversification II) in DSA is designed for avoiding traps set by recurrent feasible solutions. We define a counter F_q that counts the number of feasible solutions obtained so far. When $F_q > F_{q_max}$, we re-set TIF^q to a large number so that new and worse infeasible solutions are accepted and the search moves away from the vicinity of a trapping feasible solution. F_q is reset to zero after it exceeds F_{q_max} ($F_{q_max} = 0.01 \cdot q_max$).

Intensification in DSA: In DSA, we do not immediately allow probabilistic acceptance of a worse solution. A number of consecutive hill-climbing iterations are applied before enabling its acceptance. We carry out this intensification process by a control counter, L_q that records the consecutive number of feasible or infeasible non-improving solutions. Each non-improving solution is directly rejected and not given any probabilistic chance of acceptance until L_q reaches its limit, L_{q_max} . Here, we set $L_{q_max} = n$, that is, we set it to the number of dimensions in the COP, because we would like to have a chance to find a better neighbor in each coordinate before we may accept a worse solution. This approach has an intensification effect on the search.

When L_q exceeds L_{q_max} , it is re-set to zero, and DSA is permitted to accept a worse candidate solution \mathbf{x}_q with the annealing probability defined in Equation (3). After a worse solution is accepted, the next cycle of hill-climbing starts. This intensification approach reduces the number of moves that stray from a good sequence.

$$\text{Prob}(\text{accept}) = \begin{cases} \exp\left(-\frac{f(\mathbf{x}^q) - feas_obj^{q-1}}{f(\mathbf{x}^q)temp}\right) & \text{if } \mathbf{x}^q \text{ is feasible} \\ \exp\left(-\frac{TIF(\mathbf{x}^q) - TIF^{q-1}}{TIF(\mathbf{x}^q)temp}\right) & \text{otherwise} \end{cases} \quad (3)$$

According to the equation above, if \mathbf{x}^q turns out to be feasible, $f(\mathbf{x}^q)$ is assessed against $feas_obj^{q-1}$ and if infeasible, assessment is carried out according to the comparison between $TIF(\mathbf{x}^q)$ and TIF^{q-1} . This enables the procedure to compare a feasible candidate solution \mathbf{x}^q (generated after an infeasible solution) with the last feasible solution in the feasible sequence, rather than with \mathbf{x}^{q-1} . Hence, \mathbf{x}^q might not be immediately accepted if it is not better than $feas_obj^{q-1}$. The latter eliminates trapping in feasible areas. In any case, if a new feasible solution is good enough, its neighborhood is scanned by FSQP. The situation is similar when an infeasible candidate solution is generated. That particular infeasible candidate is accepted or rejected according to the last solution's TIF in the infeasible sequence. Thus, an infeasible solution arriving just after a feasible solution can still be accepted if it improves TIF . In this manner, DSA enhances the exploration power of SA in infeasible regions and reduces wasted function calls in feasible regions.

The cooling temperature $temp$ found in Equation (3) is managed as follows. Since the probability of acceptance is based on normalized deterioration, initially $temp$ is set to 1.0. Whenever a worse solution is accepted, the annealing temperature $temp$ is reduced geometrically ($temp \leftarrow temp/(1 + \theta)$, here, $\theta = 0.005$). This approach is different from the standard SA algorithm that waits for an epoch length to reduce $temp$. Here, the intensification duration becomes an adaptive epoch length. Ozdamar and Demirhan [33] test this approach against several SA algorithms from the literature and find that this is the most effective method in bound constrained optimization. In order to increase Prob(accept) in later stages of the search and give more freedom to DSA, the temperature $temp$ is re-annealed when it falls below its minimum allowable value, t_f .

DSA stops when the maximum number of moves, q_max , is reached. The best feasible solution, f^* , encountered during the search is reported.

FSQP calls: Throughout the search, DSA interacts with FSQP by invoking a probabilistic call to local search whenever a *better* feasible solution is found. Hence, FSQP contributes to SA by providing exploration ability around the feasible solutions identified by DSA. The relation between FSQP and DSA is such that DSA provides the current feasible solution \mathbf{x}^q to FSQP as a starting point. FSQP seeks for a local or global stationary point around \mathbf{x}^q and simply updates f^* if it finds a better solution. The sequence of solutions generated by DSA is not affected by invoking FSQP.

In order to reduce the number of FSQP calls, we introduce an annealing type of activation probability that depends on the annealing temperature $temp$ as well as on how much the new feasible solution \mathbf{x}^q improves $f(\mathbf{x})$ compared to the last feasible solution, $feas_obj^{q-1}$. The probability of calling FSQP is calculated as follows.

$$\text{Prob}(\text{call_FSQP}) = \exp\left(\frac{-1}{(feas_obj^{q-1} - f(\mathbf{x}^q)) \cdot \sqrt{temp}}\right) \quad (4)$$

FSQP is also allowed to explore around feasible or infeasible worse solutions accepted near the end of the search, that is, when the iteration index q exceeds, for instance, $0.9 \cdot q_max$. The latter FSQP calls do not require the calculation of Prob(call_FSQP).

5 The Hybrid IPA-DSA Algorithm

The hybrid IPA-DSA algorithm is basically the IPA algorithm described in Sect. 3 with the exception that rather than invoking FSQP in Step 4.1, the DSA algorithm is invoked. In the hybrid algorithm DSA works as described in Sect. 4, however, the search space is confined by the boundaries of box y . DSA invokes FSQP using its own probabilistic calling scheme. Our goal in developing this hybrid is to use the advantage of activating global search in larger boxes within an exhaustive algorithm framework. We make the following parameter adjustments in the hybrid algorithm. In each box y that DSA is activated, the number of iterations allowed is $q_{max} = 500 \cdot \text{sizeof}(y)/\text{sizeof}(x)$, and similarly, the number of iterations the solver FSQP is allowed in a given box y is equal to $150 \cdot \text{sizeof}(y)/\text{sizeof}(x)$. Thus, these parameters are based on the box size.

6 Numerical Experiments and Comparisons with Other Deterministic and Stochastic Search Methods

6.1 Test Problems

We test the performance of all methods on 32 COPs collected from different sources in the literature. These are listed in Table 1 with their characteristics (number of non-linear/linear equalities and inequalities as well as expression types), optimal or best results reported, and their source references. A few test problems are tested in their revised versions. Most of the problems involve polynomial constraints and objective functions with nine exceptions where trigonometric expressions are found (problems P3, P22, P23, P24, P25, P26, P27, P30, P31). Some problems have a single variable in their objective function (P4, P6, P7, P8, P10, P11, P12, P17), and perturbations made by DSA in such problems do not affect the objective function directly. This might lead to some difficulties in finding the direction of descent.

6.2 Comparison with Single Sequence Penalty-based SA Algorithms

DSA and the hybrid IPA-DSA are compared with five SA algorithms that handle constraint feasibility by using an objective function augmented with constraint infeasibility degrees. All these algorithms are single sequence techniques, that is, solutions are not differentiated according to feasibility status. Similar to DSA, they all call FSQP in the same manner to identify feasible stationary solutions.

We call penalty based SA algorithms SAP. In SAP, the augmented objective $f'(\mathbf{x})$ of the COP includes a penalty function, $p(\mathbf{x})$ that is non-decreasing in $TIF(\mathbf{x})$. $f'(\mathbf{x})$ is defined as

$$\min f'(\mathbf{x}) = \begin{cases} f(\mathbf{x}) + p(\mathbf{x}) & \text{if } TIF(\mathbf{x}) > 0 \\ f(\mathbf{x}) & \text{otherwise} \end{cases} \quad (5)$$

Table 1 List of Test problems

Second column: N: dimension, NE: Nonlinear equations, LE: linear equations, NI: Nonlinear inequalities, LI: Linear inequalities

*: indicates that result is obtained in this chapter

Prob. Id.	N, # NE, # LE, # NI, # LI	Notes	$f(\mathbf{x}^*)$	Source
P1	13,0,0,0,9	Quadratic $f(\mathbf{x})$	-15	[29] (G1)
P2	8,0,0,3,3	Linear $f(\mathbf{x})$, 2nd degree NI	3514.81*	[29] (G10)
P3	20,0,0,1,1	Trigonometric $f(\mathbf{x})$, 20th degree NI	0.8036	[29] (G2)
P4	21,0,0,1,10	Linear single variable $f(\mathbf{x})$, 2nd degree concave NI	-8695.012	[11] (Chapter 2, Test problem 7)
P5	11,0,0,1,5	Linear single variable $f(\mathbf{x})$, 2nd degree concave NI	-39	[11] (Chapter 2, Test problem 6)
P6	6,0,0,7,0	Linear single variable $f(\mathbf{x})$, 2nd degree NI	-30665.53	[11] (Chapter 2, Test problem 2)
P7	3,0,0,1,0	Linear single variable $f(\mathbf{x})$, 2nd degree NI	-3	[10] (e_1.def)
P8	5,0,0,1,6	Linear single variable $f(\mathbf{x})$, 2nd degree NI	-13	[43] (Example 1)
P9	2,0,0,2,2	Linear single variable $f(\mathbf{x})$, 2nd degree NI, convex and non-convex	-2.8284	[43] (Example 4)
P10	3,0,0,2,1	Linear single variable $f(\mathbf{x})$, 2nd degree NI	0.741	[42] (Example 1)
P11	3,0,0,2,0	Linear single variable $f(\mathbf{x})$, 2nd degree NI	-0.5	[42] (Example 2)
P12	8,0,0,3,3	Linear single variable $f(\mathbf{x})$, 2nd degree NI	100	[14]
P13	5,0,0,6,0	Nonlinear $f(\mathbf{x})$, 2nd degree NI	-30665.53	[14]
P14	5,0,0,6,0	Nonlinear $f(\mathbf{x})$, 2nd degree NI	-31535.37	[7]
P15	5,0,0,6,0	Nonlinear $f(\mathbf{x})$, 2nd degree NI	-31022	[32]
P16	10,0,1,0,0	2nd degree $f(\mathbf{x})$ with many, interactive terms	-0.375	[11] (Problem 2.10)
P17	21,0,0,1,10	Linear single variable $f(\mathbf{x})$, 2nd degree NI	-4150.410	Revised P4
P18	7,0,0,12,2	Nonlinear $f(\mathbf{x})$, Geometric fractional NI, 2nd degree NI	1227.183	[7]
P19	13,0,0,12,1	Linear $f(\mathbf{x})$, Geometric fractional NI, 2nd degree NI	97.59	[7]
P20	14,6,1,0,0	2nd degree $f(\mathbf{x})$, 2nd degree NE	-1.765	[4] (Alkyl.gms)
P21	10,0,7,0,0	2nd degree $f(\mathbf{x})$	0.814*	[4] (Genhs28.gms)
P22	14,0,0,2,0	Quadratic $f(\mathbf{x})$, trigonometric NI	0.00*	[4] (revised robot.gms)
P23	6,0,0,4,3	2nd $f(\mathbf{x})$, trigonometric NI	-2355.39*	[4] (revised mathopt3.gms)
P24	14,2,0,0,0	Quadratic $f(\mathbf{x})$, trigonometric NE	0.00	[4] (robot.gms)
P25	6,4,3,0,0	Quadratic $f(\mathbf{x})$, trigonometric NE	-1071.6	[4] (mathopt3.gms)
P26	11,0,0,4,2	Linear $f(\mathbf{x})$, trigonometric NI	0*	[4] (revised hs087.gms)
P27	9,0,0,8,2	Linear $f(\mathbf{x})$, trigonometric NI, 2nd degree NI	5319.58*	[4] (revised hs109.gms)
P28	17,0,0,7,4	Linear $f(\mathbf{x})$, 4th degree NI, 3rd degree NI, 2nd degree NI	-1000*	[4] (revised rk23.gms)
P29	4,0,0,0,6	2nd $f(\mathbf{x})$	-333.333*	[4] (revised hs044.gms)
P30	4,3,0,0,2	2nd $f(\mathbf{x})$, trigonometric NE	5126.49	[40] (G5)
P31	2,0,0,2,0	Trigonometric $f(\mathbf{x})$, 2nd degree NI	-0.1	[40] (G8)
P32	10,0,0,5,3	2nd $f(\mathbf{x})$, 2nd degree NI	24.306	[40] (G7)

In SAP, the augmented $f'(\mathbf{x})$ is used to assess solutions rather than $f(\mathbf{x})$. The annealing probability of acceptance is expressed below.

$$\text{Prob}(\text{accept}) = \exp \left(\frac{-(f'(\mathbf{x}^q) - f'(\mathbf{x}^{q-1}))}{f'(\mathbf{x}^q) \text{temp}} \right) \quad (6)$$

SAP contains the intensification scheme in DSA and the first diversification scheme. The second diversification scheme in DSA is omitted since the feasible solution sequence is not traced separately in SAP. In other words, all features of SAP and DSA are the same with this exception and this performance comparison only involves the maintenance of a single sequence or a dual sequence.

We now list the types of penalty functions $p(\mathbf{x})$ considered in this comparison.

Penalty functions. We utilize five different expressions for $p(\mathbf{x})$. These are adapted from the GA literature. The first two penalty functions are static in the sense that they are not affected by the status of the search, i.e., on the number of function evaluations already spent until the current solution is obtained. The third one is dynamic, and the last two are classified as annealing penalty functions. These are listed below.

1. *MQ* [30]. MQ is a static penalty function that depends only on the number of infeasible constraints rather than on $TIF(\mathbf{x})$. The penalty function is expressed below.

$$p(\mathbf{x}) = B - \frac{Bs}{m} \quad (7)$$

Here, s is the number of feasible constraints (equalities plus inequalities), m is the total number of constraints, and B is a large positive number (10^9 is the suggested value by the developers of the function) that guarantees that an infeasible solution has a larger objective function value than a feasible solution.

2. *QP* (Quadratic Penalty function). QP is the static classical quadratic penalty function expressed below.

$$p(\mathbf{x}) = B \sum_{i=1}^m INF_i^2(\mathbf{x}) \quad (8)$$

Again, B is a sufficiently big positive number (e.g., 10^6) such that a solution having a larger $TIF(\mathbf{x})$ is guaranteed to have a worse objective function value than another with a smaller $TIF(\mathbf{x})$.

3. *JH* [18]. JH is a dynamic penalty function that depends both on the progress of the search and on $TIF(\mathbf{x})$. The idea in JH is to allow exploration in earlier stages of the search and become strict in later stages. Here, for representing the progress of the search we take the number of SAP iterations, q , made until the current solution is reached. The penalty function is adapted as follows:

$$p(\mathbf{x}) = (qC)^\alpha \sum_{i=1}^m INF_i^\beta(\mathbf{x}) \quad (9)$$

The value of the parameter C is suggested as 0.5, and for α, β , the values of 1 or 2 are suggested. Here, we prefer the former value because q becomes large in later stages of the search.

The advantage of JH and the following two annealing penalties over static ones is that the infeasibility (and its related $p(\mathbf{x}^q)$) of a candidate solution, \mathbf{x}^q , can be traded off with the reduction in $f(\mathbf{x}^q)$. That is, $(f(\mathbf{x}^q) + p(\mathbf{x}^q)) < f(\mathbf{x}^{q-1})$ where \mathbf{x}^{q-1} is the feasible predecessor solution. This increases the exploration capability of SAP.

4. *MA* [28]. *MA* is an annealing penalty function where the quadratic penalty weight is divided by the annealing temperature, *temp*, that is reduced as the search runtime increases. Similar to JH, the penalty becomes quite high near the end of the search, and infeasible solutions tend to be rejected. The function $p(\mathbf{x})$ is adapted as follows.

$$p(\mathbf{x}) = \frac{0.5 \sum_{i=1}^m INF_i^2(\mathbf{x})}{temp} \quad (10)$$

Here, we let *temp* take the value of the geometric cooling temperature that the SAP algorithm uses in that iteration. Hence, the rejection criterion of non-improving solutions in SAP is aligned with $p(\mathbf{x})$.

5. *CSBA* [3]. *CSBA* is an annealing penalty function of multiplicative type where $f(\mathbf{x})$ is multiplied by an exponential function whose arguments are $TIF(\mathbf{x})$ and *temp*. As *temp* goes to zero, the penalty function goes to one leading to increased $f(\mathbf{x})$. The penalty function is adapted as follows.

$$p(\mathbf{x}) = f(\mathbf{x}) e^{\frac{-\sqrt{temp}}{TIF(\mathbf{x})}} \quad (11)$$

6.3 Comparison with IPA

The IPA described here utilizes an adaptive tree management. For the purpose of the comparison, here, we also include two more IPA versions: IPA with best-first tree management scheme and IPA with depth-first scheme. These two versions are similar to IPA-adaptive scheme. However, in the best-first approach, the one with least $TIF(\mathbf{y})$ among indeterminate boxes is selected for re-partitioning until a feasible solution is identified, and, then, once a feasible solution is identified, the box with the lowest $\underline{F}(\mathbf{y})$ is selected. On the other hand, in the depth-first approach, the box on the left branch of the last partitioned box is always selected for re-partitioning. Calls to FSQP are managed in the same way (according to the improvement in TAD) in all three IPA versions.

Furthermore, we also provide the hybrid IPA-DSA algorithm in three versions: with adaptive, best-first and depth-first tree management schemes.

6.4 Comparison with Other Deterministic Approaches

To complete the comparison portfolio, we also solve the test problems with five well-known solvers that are linked to the commercial software GAMS (www.gams.com) and also with stand-alone FSQP [52], [24] whose code has been provided by AEM (www.aemdesign.com/FSQPmanyobj.htm). The solvers used in this comparison are BARON 7.0 [39], Conopt 3.0 [9], MINOS 5.5 [31], Snopt 5.3.4 [13] and FSQP. Among these solvers, BARON is an exhaustive global solver and all others are local solvers. We allow every solver to complete its run without imposing additional stopping criteria except the maximum CPU time.

6.5 Results

In the experiments, SAP, DSA, and the hybrid IPA-DSA are re-run 100 times for every test problem using a 1.7 GHz PC with 256 MB RAM in Windows operating system. In SAP and DSA, every problem is run with a maximum iteration counter of $q_{max} = 2000(n + m)$ or maximum CPU time of 900 CPUseconds. In IPA-DSA hybrid, the number of function evaluations, excluding those made by DSA, is limited to $1000(n + m)$ and DSA is allowed a number of function calls that depends on the relative size of the box as described in Sect. 5. One run of IPA-DSA takes a longer time than stand-alone DSA and therefore, the maximum number of function evaluations is halved. Similarly, the stand-alone IPA methods are allowed $1000(n + m)$ function evaluations. All methods in the comparison have a limited run time of 900 CPUseconds.

In Tables 2 and 3, we illustrate the results obtained by the hybrid stochastic and stochastic methods. In Table 2, we summarize the results obtained by the three tree management schemes in the hybrid IPA-DSA method. In Table 3, we provide the results obtained by stand-alone DSA and five SAP methods.

We provide the results in the following format. For all SA methods and hybrid IPA-DSA, we provide the average absolute deviations from the global optima (and standard deviations) of three results: the average deviation of the worst solution obtained in 100 runs (1st results column in Tables 2 and 3) over 32 problems, the average deviation of the best solution in 100 runs (2nd column in Tables 2 and 3) over

Table 2 Hybrid IPA-DSA results

Hybrid IPA-DSA	Abs. Dev. (Worst)	Abs. Dev. (Best)	Abs. Dev. (Average)	Ratio of Unsolved Probs. Over 100 runs	CPU Secs
IPA-DSA (Adaptive Tree)					
Average	327.40	96.73	144.58	0.000	38.37
Std. Dev.	800.53	327.87	356.42	0.000	136.98
# of Optimal Solutions	17	23			
# of Unsolved Probs.	0	0			
IPA-DSA (Best-first Tree)					
Average	237.990	95.80	152.44	0.001	44.55
Std. Dev.	653.53	328.02	372.54	0.004	160.23
# of optimal Solutions	15	21			
# of Unsolved Probs.	1	0			
IPA-DSA (Depth-first Tree)					
Average	256.17	100.50	137.41	0.014	12.48
Std. Dev.	634.01	327.27	305.58	0.073	45.53
# of Optimal Solutions	16	20			
# of Unsolved Probs.	2	0			

Table 3 Hybrid IPA-DSA results

SA Methods	Abs. Dev. (Worst)	Abs.Dev. (Best)	Abs. Dev. (Average)	Ratio of Unsolved Probs. Over 100 runs	CPU (Secs)
without Penalty					
DSA					
Average	218.62	109.54	136.29	0.004	1.842
Std. Dev.	571.53	332.49	357.25	0.02	2.325
# of Optimal Solutions	15	21			
# of Unsolved Probs.	1	0			
with Penalty (SAP)					
Morales-Quezada					
Average	543.87	111.82	298.11	0.02	1.55
Std. Dev.	1279.56	330.91	666.60	0.08	2.74
# of Optimal Solutions	14	20			
# of Unsolved Probs.	4	0			
Static Quadratic					
Average	340.39	107.46	156.85	0.01	1.98
Std. Dev.	730.49	331.39	362.95	0.03	4.19
# of Optimal Solutions	13	22			
# of Unsolved Probs.	4	0			
Joines and Houck					
Average	385.82	107.26	157.45	0.01	1.73
Std. Dev.	841.84	331.44	366.32	0.05	3.55
# of Optimal Solutions	13	20			
# of Unsolved Probs.	4	0			
Michalewicz and Attia					
Average	622.50	116.22	334.69	0.01	0.75
Std. Dev.	1381.33	334.51	657.46	0.04	0.81
# of Optimal Solutions	9	20			
# of Unsolved Probs.	6	0			
Carlson et al.					
Average	647.40	116.22	334.69	0.03	0.54
Std. Dev.	1403.84	334.51	657.46	0.10	0.63
# of Optimal Solutions	8	20			
# of Unsolved Probs.	7	0			

32 problems and the average of 100 runs' average deviation (3rd column in Tables 2 and 3) over 32 problems. Finally, the average ratio of unsolved problems in 100 runs where no feasible solution was found (4th column) and the average CPU times per run are reported (5th column). We also report the number of optimal solutions found among worst and best solutions (3rd row of every method), and the number of problems where no feasible solution was found at all in 100 runs, i.e., total failure of the procedure (4th row of every method).

In Table 4, we provide the results for deterministic methods. These are illustrated in terms of the average absolute deviation from the global optimum, standard deviation, number of optimal solutions obtained and number of problems for which no feasible solution was found.

Let us first compare the results in Tables 2 and 3. We observe that the average results obtained by DSA and the hybrid IPA-DSA (adaptive tree) methods are not

Table 4 Hybrid IPA-DSA results

Deterministic Methods	Abs. Dev.	CPU Secs	Deterministic Methods	Abs. Dev.	CPU Secs
FSQP			MINOS		
Average	300.75	0.04	Average	393.90	0.27
Std. Dev.	939.19	0.07	Std. Dev.	955.30	0.16
# of Optimal Solutions	16		# of Optimal Solutions	11	
# of Unsolved Probs.	0		# of Unsolved Probs.	3	
Baron (Exhaustive)			Conopt		
Average	108.99	0.13	Average	516.90	0.29
Std. Dev.	374.99	0.12	Std. Dev.	1215.03	0.33
# of Optimal Solutions	13		# of Optimal Solutions	9	
# of Unsolved Probs.	0		# of Unsolved Probs.	7	
Snopt					
Average	442.84	0.38			
Std. Dev.	960.95	0.58			
# of Optimal Solutions	10				
# of Unsolved Probs.	5				
IPA					
IPA Adaptive Tree	Abs. Dev.	CPU Secs	IPA Best-first Tree	Abs. Dev.	CPU Secs
Average	114.48	39.27	Average	117.70	282.00
Std. Dev.	330.59	158.54	Std. Dev.	342.01	352.53
# of Optimal Solutions	17		# of Optimal Solutions	17	
# of Unsolved Probs.	0		# of Unsolved Probs.	3	
IPA (Depth-first Tree)					
Average	380.74	4.73			
Std. Dev.	980.11	10.12			
# of Optimal Solutions	17				
# of Unsolved Probs	2				

significantly different. However, in terms of the number of optimal solutions found and in terms of the criterion of achieving feasible solutions in every run, the hybrid method is better than DSA. If we consider the CPU times taken by both methods, the hybrid method is seen to be computationally very expensive. In general, the best-first tree management approach takes longer CPU times due to its memory and sorting requirements and the depth-first approach is fastest among the hybrids. Yet, in terms of achieving a feasible solution in every run, the depth-first approach is inferior to the other two approaches (a feasible solution is not identified in about 1.4 runs over 100). In terms of the worst solutions obtained, DSA's performance is superior to those of SAP, and the best-first and depth-first approaches in the hybrid method perform as well as DSA in this respect. These results show that using DSA as a stand-alone solver is a reasonable choice for the COP because CPU times are small and average case and worst case quality of solutions are acceptable. On the other hand, if one wishes to improve the best solution found at the expense of longer computation times, the hybrid IPA-DSA method can be used.

We now take a look at Table 4 and compare deterministic methods with SA-based approaches. As expected, the best method among GAMS solvers is the exhaustive approach BARON. The computation time taken by this solver is very small, however, the number of optimal solutions found is much lower than the best solutions found by all SA-based methods. Again, this advantage comes with more computational effort. For DSA, we can obtain best results in 100 runs while BARON carries out only one run. However, one should not forget that BARON would not be able to find a better solution within the allowed CPU time while it is always possible to obtain a better result by running a stochastic method more than once. Using the stand-alone IPA-adaptive tree might not be a good option under these circumstances. Though the number of optimal solutions obtained is higher than BARON and problem solving capability is as good as that of BARON, the computation times are higher.

We can summarize our findings as follows. Considering the deterministic and stochastic COP methods tested in these experiments, SA-based methods seem to be the best option to use in terms of CPU times and solution quality. If a user is willing to afford longer computation times, then he/she can resort to the hybrid IPA-DSA solver to have a higher chance of identifying the optimum solution to the COP at hand. The IPA-DSA adaptive tree approach described here uses its exhaustive partitioning and exploration tools that result in the highest number of optimal solutions and best capability in identifying feasible solutions in every run.

7 Conclusions

We have described different SA-based approaches to solve the COP. Among the SA methods described here, the performance of the novel dual-sequence SA (DSA) and those of SA approaches with different penalty functions (SAP) are illustrated. Furthermore, a new stand-alone exhaustive interval partitioning algorithm (IPA) with an adaptive tree management scheme is described. This method is also tested with different tree management schemes. IPA provides us with an exhaustive exploration

framework that guides DSA in conducting the search. By combining IPA with DSA, we obtain a hybrid algorithm (IPA-DSA) that uses the advantage of exploring the whole search space by systematic partitioning while discarding sub-spaces that are guaranteed to exclude feasible solutions or better stationary points reliably. Finally, we have included the well-known deterministic commercial solvers in the experiments. Our empirical results indicate that it is desirable to use DSA to solve the COP if one wishes to obtain a high quality solution within small computation times. However, the solution quality can be even better if the hybrid IPA-DSA is used (in terms of the number of optimal solutions found and identifying feasible solutions). The performance of the exhaustive deterministic method BARON is also quite good and fast, but it lacks the flexibility of finding even better solutions by multiple re-runs.

Acknowledgement. We wish to thank Professor Andre Tits (Electrical Engineering and the Institute for Systems Research, University of Maryland, USA) for providing the source code of CFSQP.

References

1. Al-Khayyal F A, Sherali H D (2000) SIAM Journal on Optimization, 10:1049–1057
2. Bennis W A, Dhingra A K (1995) International Journal of Numerical Methods in Engineering, 38:2753–2773
3. Carlson S, Shonkwiler R, Babar S, Aral M (1998) Annealing a genetic algorithm over constraints. SMC 98 Conference, available from <http://www.math.gatech.edu/shenk/body.html>
4. Coconut Test Problems on Constrained Optimization Problems – Library2. <http://www.mat.univie.ac.at/~neum/glopt/coconut/>
5. Csendes T, Ratz D (1997) SIAM Journal of Numerical Analysis 34:922–938
6. Dallwig S, Neumaier A, Schichl H (1997) GLOPT – A Program for Constrained Global Optimization. In: Bomze I M, Csendes T, Horst R, Pardalos P M (eds) Developments in Global Optimization. Kluwer, Dordrecht
7. Dembo R S (1976) Mathematical Programming 10:192–213
8. Dekkers A, Aarts E (1991) Mathematical Programming 50:367–393
9. Drud A S (1996) CONOPT: A System for Large Scale Nonlinear Optimization. Reference Manual for CONOPT Subroutine Library, ARKI Consulting and Development A/S, Bagsvaerd Denmark
10. Epperly T G (1995) Global optimization of nonconvex nonlinear programs using parallel branch and bound. Ph. D dissertation, University of Wisconsin-Madison, Madison
11. Floudas C A, Pardalos P M (1990) A collection of Test Problems for Constrained Global Optimization Algorithms. Volume 455 of Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg New York
12. Forsgren A, Gill P E, Wright M H (2002) SIAM Review 44:525–597
13. Gill P E, Murray W, Saunders M A (1997) SNOPT: An SQP algorithm for large-scale constrained optimization. Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA
14. Hansen P, Jaumard B, Lu S-H (1991) Mathematical Programming 52:227–254

15. Hansen E R(1992) *Global Optimization Using Interval Analysis*. Marcel Dekker, New York
16. Hansen E, Sengupta S (1980) Global constrained optimization using interval analysis. In: Nickel K L (eds), *Interval Mathematics*
17. Hedar A-R, Fukushima M (2006) Derivative-free filter simulated annealing method for constrained continuous global optimization. *Journal of Global Optimization* (to appear)
18. Joines J, Houck C (1994) On the use of non-stationary penalty functions to solve non-linear constrained optimization problems with GAs. *Proceedings of the First IEEE International Conference on Evolutionary Computation*, IEEE Press. 579–584
19. Kearfott R B (1996a) *Rigorous Global Search: Continuous Problems*. Kluwer, Dordrecht, Netherlands
20. Kearfott R B (1996b) A Review of Techniques in the Verified Solution of Constrained Global Optimization Problems. In: Kearfott R B, Kreinovich V (eds) *Applications of Interval Computations*. Kluwer, Dordrecht, Netherlands
21. Kearfott R B(2003) An overview of the GlobSol Package for Verified Global Optimization. Talk given for the Department of Computing and Software, McMaster University, Cannada
22. Kirkpatrick A, Gelatt Jr C D, Vechi M P (1983) *Science* 220:671–680
23. Korf R E (1985) *Artificial Intelligence* 27:97–109
24. Lawrence C T, Zhou J L, Tits A L (1997) *User's Guide for CFSQP version 2.5: A Code for Solving (Large Scale) Constrained Nonlinear (minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints*. Institute for Systems Research, University of Maryland, College Park, MD
25. Leite J P B, Topping B H V (1999) *Computers and Structures* 73:545–564
26. Markot M C (2003) *Reliable Global Optimization Methods for Constrained Problems and Their Application for Solving Circle Packing Problems*. PhD dissertation, University of Szeged, Hungary
27. Markot M C, Fernandez J, Casado L G, Csentes T (2006) New interval methods for constrained global optimization. *Mathematical Programming* 106:287–318
28. Michalewicz Z, Attia N (1994) Evolutionary optimization of constrained problems. *Proceedings of the Third Annual Conference on Evolutionary Programming*, World Scientific
29. Michalewicz Z (1995) Genetic algorithms, numerical optimization, and constraints. *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann
30. Morales K A, Quezada C C (1998) A universal eclectic genetic algorithm for constrained optimization. *Proceedings 6th European Congress on Intelligent Techniques and Soft Computing*, EUFIT'98
31. Murtagh B A, Saunders M A (1987) *MINOS 5.0 User's Guide*. Report SOL 83-20, Department of Operations Research, Stanford University, USA
32. Myung H, Kim J-H, Fogel D B (1995) *Evolutionary Programming* 449–463
33. Özdamar L, Demirhan M (2000) *Computers and Operations Research*, 27:841–865
34. Pardalos P M, Romeijn H E (2002) *Handbook of Global Optimization Volume 2*. Springer, Boston Dordrecht London
35. Pedamallu C S, Özdamar L (2006) Investigating a hybrid simulated annealing and local search algorithm for constrained optimization. *Inpress EJOR* (<http://dx.doi.org/10.1016/j.ejor.2006.06.050>)
36. Pedamallu C S, Özdamar L, Csentes T (2006) An interval partitioning approach for continuous constrained optimization. Accepted for publication in *Models and Algorithms in Global Optimization*. Springer, Berlin Heidelberg New York

37. Pinter J D (1997) LGO – A program system for continuous and Lipschitz global optimization. In: Bomze I M, Csendes T, Horst R, Pardalos P M (eds) *Developments in Global Optimization*. Kluwer Academic Publishers, Dordrecht Boston London
38. Ratschek H, Rokne J (1988) *New Computer Methods for Global Optimization*. Ellis Horwood, Chichester
39. Sahinidis N V (2003) *Global Optimization and Constraint Satisfaction: The Branch-and-Reduce Approach*. In: Bliek C, Jermann C, Neumaier A (eds) *COCOS 2002, LNCS*. Springer, Boston Dordrecht London
40. Schoenauer M, Michalewicz Z (1999) *Evolutionary Computation* 7:19–44
41. Smith E M B, Pantelides C. C (1999) *Computers and Chemical Engineering* 23:457–478
42. Swaney R E (1990) Global Solution of algebraic nonlinear programs. *AIChE Annual Meeting*, Chicago, IL
43. Visweswaran V, Floudas C A (1990) *Computers and Chemical Engineering* 14:1419–1434
44. Wah B W, Wang T (2000) *International Journal on Artificial Intelligence Tools* 9:3–25
45. Wah B W, Chen Y X (2000) Optimal anytime constrained simulated annealing for constrained global optimization. In: Dechter R (ed.) *LNCS 1894*. Springer, Berlin Heidelberg New York
46. Wolfe M A (1994) *Journal of Computational and Applied Mathematics* 50:605–612
47. Wong K P, Fung C C (1993) *IEE Proceedings: Part C* 140:509–515
48. Wong K P, Wong Y W (1995) *IEE Proceedings: Generation Transmission and Distribution* 142:372–380
49. Wong K P, Wong Y W (1996) *IEEE Transactions on Power Systems* 11:112–118
50. Wong K P, Wong Y W (1997) *IEEE Transactions on Power Systems* 12:776–784
51. Wong Y C, Leung K S, Wong C K (2000) *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 30: 506–516
52. Zhou J L, Tits A L (1996) *SIAM Journal on Optimization* 6:461–487

Advances in Metaheuristics for Hard Optimization

Siarry, P.; Michalewicz, Z. (Eds.)

2008, XVI, 481 p. 167 illus., Hardcover

ISBN: 978-3-540-72959-4