Solving Large-Scale Nonlinear Programming Problems by Constraint Partitioning[⋆]

Benjamin W. Wah and Yixin Chen

Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory,
University of Illinois, Urbana-Champaign,
Urbana, IL 61801, USA
{wah, chen}@manip.crhc.uiuc.edu
http://www.manip.crhc.uiuc.edu

Abstract. In this paper, we present a constraint-partitioning approach for finding local optimal solutions of large-scale mixed-integer nonlinear programming problems (MINLPs). Based on our observation that MINLPs in many engineering applications have highly structured constraints, we propose to partition these MINLPs by their constraints into subproblems, solve each subproblem by an existing solver, and resolve those violated global constraints across the subproblems using our theory of extended saddle points. Constraint partitioning allows many MINLPs that cannot be solved by existing solvers to be solvable because it leads to easier subproblems that are significant relaxations of the original problem. The success of our approach relies on our ability to resolve violated global constraints efficiently, without requiring exhaustive enumerations of variable values in these constraints. We have developed an algorithm for automatically partitioning a large MINLP in order to minimize the number of global constraints, an iterative method for determining the optimal number of partitions in order to minimize the search time, and an efficient strategy for resolving violated global constraints. Our experimental results demonstrate significant improvements over the best existing solvers in terms of solution time and quality in solving a collection of mixed-integer and continuous nonlinear constrained optimization benchmarks.

1 Introduction

In this paper, we study mixed-integer nonlinear programming problems (MINLPs) of the following general form:

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

where variable $z = (x, y)$, and $x \in \mathbb{R}^v$ and $y \in \mathbb{D}^w$ are, respectively, the continuous and the discrete parts. The objective function f is continuous and differentiable with respect to x, whereas the constraint functions $h = (h_1, \ldots, h_m)^T$ and $g = (g_1, \ldots, g_r)^T$ are general functions that can be discontinuous, non-differentiable, and not in closed form.

[⋆] Research supported by National Science Foundation Grant IIS 03-12084.

MINLPs defined by P_m include discrete problems and continuous nonlinear programming problems (CNLPs) as special cases. Ample applications exist in production management, operations research, optimal control, and engineering designs.

Because there is no closed-form solution to P_m, we aim at finding local optimal solutions to the problem. We, however, focus on solving some of the more difficult instances that cannot be solved by existing solvers.

An example MINLP that cannot be solved by existing solvers is TRIMLON12. This is an instance of the TRIMLON benchmark [9] with $I = J = 12$. The goal is to produce a set of product paper rolls from raw paper rolls by assigning continuous variables $m[j]$ and $y[j]$ and integer variables $n[i,j]$, where $i = 1, \ldots, I$ and $j = 1, \ldots, J$, in order to minimize f as a function of the trim loss and the overall production cost.

$$\begin{array}{lll}
\text{objective:} & \min_{z=(y,m,n)} f(z) = \sum_{j=1}^{J}(c[j]\cdot m[j] + C[j]\cdot y[j]) & \text{(OBJ)} \\
\text{subject to:} & B_{min} \le \sum_{i=1}^{I}(b[i]\cdot n[i,j]) \le B_{max} & \text{(C1)} \\
& \sum_{i=1}^{I} n[i,j] - N_{max} \le 0 & \text{(C2)} \\
& y[i] - m[j] \le 0 & \text{(C3)} \\
& m[j] - M\cdot y[j] \le 0 & \text{(C4)} \\
& Nord[i] - \sum_{j=1}^{J}(m[j]\cdot n[i,j]) \le 0. & \text{(C5)}
\end{array}$$

An instance can be specified by defining I and J, leading to $(I+2)J$ variables and $5J + I$ constraints. For example, there are 168 variables and 72 constraints in TRIMLON12.

A key observation we have made on many application benchmarks, including TRIMLON12, is that their constraints do not involve variables that are picked randomly from their variable sets. Invariably, many constraints in these benchmarks are highly structured because they model relationships that have strong spatial or temporal locality, such as those in physical structures and task scheduling.

Figure 1a illustrates this point by depicting the constraint structure of TRIMLON12. It shows a dot where a constraint (with a unique ID on the x axis) is related to a variable (with a unique ID on the y axis). With the order of the variables and the constraints arranged properly, the figure shows a strong regular structure of the constraints. Figures 1b and 1c further illustrate the regular constraint structure of two other benchmarks.

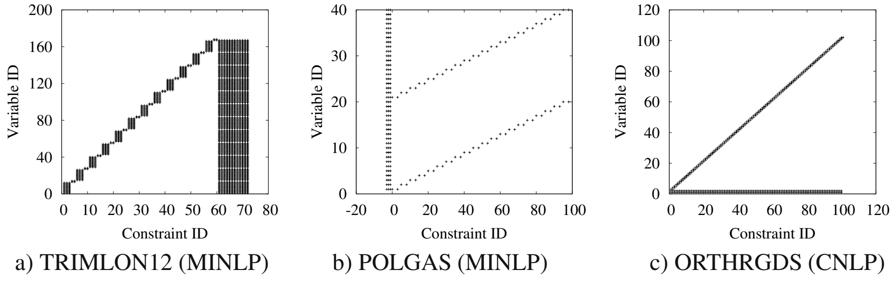


Fig. 1. Regular structures of constraints in some MINLP and CNLP benchmarks. A dot in each graph represents a variable associated with a constraint.

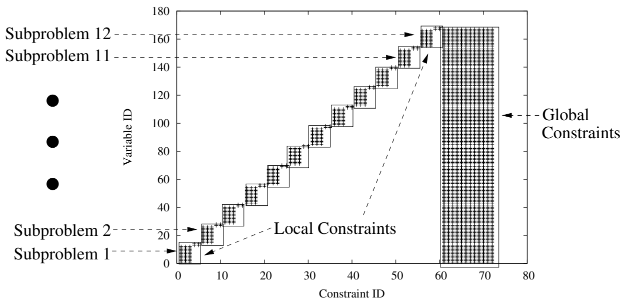
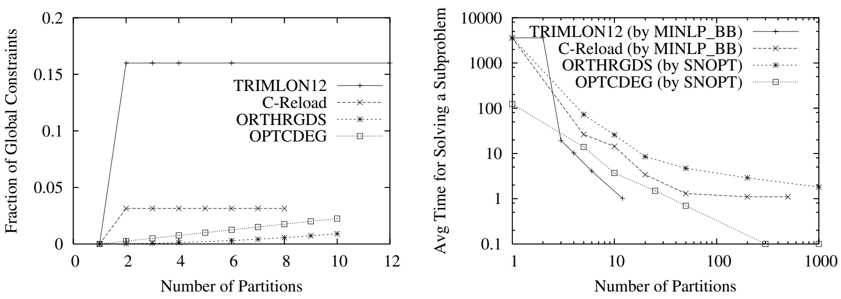


Fig. 2. An illustration of the partitioning of the constraints in TRIMLON12 into 12 subproblems



a) Monotonic increase in fraction of global constraints

b) Exponential decrease in average time for solving a subproblem

Fig. 3. Trade-offs between the number of global constraints to be resolved and the average time for evaluating a subproblem. As the number of partitions increases, the number of global constraints to be satisfied increases, while the average time to solve a subproblem decreases.

Based on the regular constraint structure of a problem instance, we can cluster its constraints into multiple loosely coupled partitions. To illustrate the idea, consider the partitioning of the constraints in TRIMLON12 by index $j \in S_J = \{1, \cdots, 12\}$. Suppose S_J is partitioned into N disjoint subsets in such a way that $S_1 \cup \cdots \cup S_N = S_J$. Then the k^{th} subproblem, $k = 1, \ldots, N$, has variables $y[j], m[j], n[j,i]$, where $i = 1, \cdots, I$, $j \in S_k$, and a common objective function (OBJ). (C1)-(C4) are its local constraints because each involves only local indexes on j. (C5), however, is a global constraint because it involves a summation over all j.

Figure 2 illustrates the decomposition of TRIMLON12 into $N = 12$ partitions, where S_J is partitioned evenly and $S_k = \{k\}$. Of the 72 constraints, 60 are local and 12 are global. Hence, the fraction of constraints that are global is 16.7%.

The fraction of constraints that are global in a problem instance depends strongly on its constraint structure and the number of partitions. Using the straightforward scheme in TRIMLON12 to partition the constraints evenly, Figure 3a illustrates that the fraction of global constraints either increases monotonically or stays unchanged with respect to the number of partitions for four benchmarks.

In contrast, the time required to solve a subproblem decreases monotonically as the number of partitions is increased. When a problem is partitioned by its constraints,

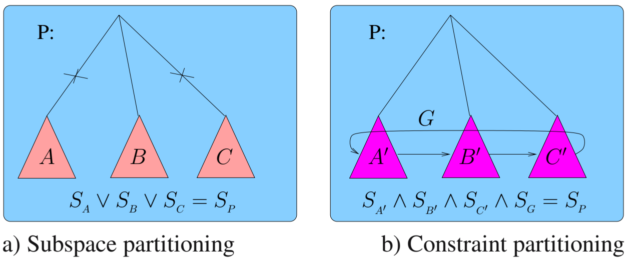

P:
A
B
C
$S_A \vee S_B \vee S_C = S_P$
a) Subspace partitioning
P:
G
A′
B′
C′
$S_{A'} \wedge S_{B'} \wedge S_{C'} \wedge S_G = S_P$
b) Constraint partitioning

Fig. 4. An illustration of subspace partitioning and constraint partitioning. Subspace partitioning decomposes P into a disjunction ($\vee$) of subproblems, where the complexity of each subproblem is similar to that of P. In contrast, constraint partitioning decomposes P into a conjunction ($\wedge$) of subproblems and a set of global constraints (G) to be resolved, where the complexity of each subproblem is substantially smaller than that of P.

each subproblem is much more relaxed than the original problem and can be solved in exponentially less time than the original. Figure 3b illustrates this exponential decrease of the average time for solving a subproblem with increasing number of partitions. The overheads between no partitioning and partitioning can be several orders of magnitude.

The partitioning of a problem by its constraints creates a new issue not addressed in past studies, namely, the resolution of global constraints relating the subproblems.

Traditional methods solve MINLPs by *subspace partitioning*. This decomposes a problem by partitioning its variable space into a disjunction ($\vee$) of subspaces and by exploring each subspace one at a time until the problem is solved (Figure 4a). Although pruning and ordering strategies can make the search more efficient by not requiring the search of every subspace, the complexity of searching each subspace is very similar to that of the original problem. In contrast, constraint partitioning decomposes the constraints of a problem into a conjunction ($\wedge$) of subproblems that must all be solved in order to solve the original problem. Each subproblem is typically much more relaxed than the original and requires significantly less time to solve (Figure 3b). However, there are global constraints (S_G in Figure 4b) that may not be satisfied after solving the subproblems independently. These global constraints include constraints in P that span across variables in multiple subproblems and new constraints added to maintain the consistency of shared variables across the subproblems. As a result, the subproblems may need to be solved multiple times in order to resolve any violated global constraints. The number of times that the subproblems are to be solved depends strongly on the difficulty in resolving the violated global constraints.

The keys to the success of using constraint partitioning to solve MINLPs and CNLPs, therefore, depend on the identification of the constraint structure of a problem instance and the efficient resolution of its violated global constraints. To this end, we study four related issues in this paper.

a) *Automated analysis of the constraint structure of a problem instance and its partitioning into subproblems.* We present in Section 4.1 the analysis of an instance specified in some standard form (such as AMPL [5] and GAMS). We show methods for determining the structure of an instance after possibly reorganizing its variables and its constraints, and identify the dimension by which the constraints can be partitioned.

b) *Optimality of the partitioning.* The optimality relies on trade-offs between the number of violated global constraints to be resolved (Figures 3a) and the overhead for evaluating a subproblem (Figure 3b). We present in Section 4.1 a metric for comparing the various partitioning schemes and a simple and effective heuristic method for selecting the optimal partitioning according to the metric.

c) *Resolution of violated global constraints.* We present in Section 3 the theory of extended saddle points (ESP) for resolving violated global constraints. The theory was originally developed for solving AI planning problems [15] whose constraints are not necessarily continuous, differentiable, and in closed form. Since continuity and differentiability of the continuous subspace is generally true in CNLPs and MINLPs, they can be exploited to speed up tremendously the solution of each subproblem.

d) *Demonstration of improvements over existing solvers.* We demonstrate the success of our approach in Section 5 by solving some large-scale CNLP and MINLP benchmarks that cannot be solved by other leading solvers.

2 Previous Work

In this section, we survey existing penalty methods for solving CNLPs and MINLPs and partitioning methods for decomposing large problems into subproblems.

Penalty Methods for Constrained Programming. Penalty methods belong to a general approach that can solve continuous, discrete, and mixed constrained optimization problems, with no continuity, differentiability, and convexity requirements. A penalty function of P_m is a summation of its objective and constraint functions (possibly under some transformations) weighted by penalties. The goal of a penalty method is to find suitable penalty values in such a way that the z^* which minimizes the penalty function corresponds to a local optimal solution of P_m.

Penalty methods can be classified into global (*resp.*, local) optimal penalty methods that look for constrained global (*resp.*, local) optimal solutions.

Global optimal penalty methods rely on the one-to-one correspondence between a *constrained global minimum* (CGM) of P_m and a global minimum z^* of the following penalty function with non-negative (transformed) constraint functions [13]:

$$L_s(z, c) = f(z) + c \cdot \left[\sum_{i=1}^{m} (h_i(z))^\rho + \sum_{i=1}^{r} (\max(0, g_i(z)))^\rho \right], \tag{2}$$

where ρ is a constant no less than 1, and c is a positive penalty parameter that is larger than a finite c^*. Here, c^* can be finite or infinite, depending on the value of ρ, and can be statically chosen or dynamically adjusted.

Methods based on finding the global minimum of (2) are of limited practical importance because the search of a global minimum of a nonlinear function is very computationally expensive. Techniques like simulated annealing are too slow because they only achieve global optimality with asymptotic convergence.

To avoid expensive global optimization, *local optimal penalty methods* have been developed for finding *constrained local minima* (CLM) instead of CGM. One approach is the Lagrange-multiplier method developed for solving CNLPs with continuous and differentiable objective and constraint functions. It relies on the *Karush-Kuhn-Tucker*

(KKT) condition [1], a first-order necessary condition on a CLM that is also a regular point. Because the condition is expressed as a system of simultaneous equations, its solution leads to unique Lagrange multipliers at a CLM. When the condition is nonlinear and not solvable in closed form, iterative procedures have been developed. However, there is no efficient solution procedure for resolving inconsistent assignments when the nonlinear equations are partitioned into subproblems and solved independently.

Another local optimal penalty method for solving CNLPs is the ℓ_1-penalty method based on the following ℓ_1-penalty function [8]:

$$\ell_1(z,c) = f(z) + c \cdot \max\Big(0, |h_1(z)|, \cdots, |h_m(z)|, g_1(z), \cdots, g_q(z)\Big). \tag{3}$$

Its theory shows that there is a one-to-one correspondence between a CLM and an unconstrained local minimum of (3) when c is larger than a finite c^*. The method cannot support the constraint partitioning of P_m for two reasons. First, the theory was derived under the continuity and differentiability assumptions on constraints similar to those in the first-order KKT condition. In fact, c^* can be proved to be the maximum of all Lagrange multipliers of the corresponding Lagrangian formulation. Second, since there is only one penalty c on the maximum of all constraint violations, it is difficult to partition (3) by its constraints and to reach a consistent value of c across the subproblems.

Existing Partitioning Methods. Partitioning is popular in existing methods for solving NLPs. Many MINLP solution methods are based on subspace partitioning and decompose the search space of a problem instance into subproblems. Examples include the following. a) *Generalized Benders decomposition* (GBD) [6] decomposes a problem space into multiple subspaces by fixing the values of its discrete variables, and by using a master problem to derive bounds and to prune inferior subproblems. b) *Outer approximation (OA)* [4] is similar to GBD except that the master problem is formulated using primal information and outer linearization. c) *Generalized cross decomposition (GCD)* [10] iterates between a phase solving the primal and dual subproblems and a phase solving the master problem. d) *Branch-and-reduce methods* [14] solve MINLPs and CNLPs by a branch-and-bound algorithm and exploit factorable programming to construct relaxed problems. All these methods require the original problem to have special decomposable structures and the subproblems to have some special properties, such as nonempty and compact subspaces with convex objective and constraint functions.

Another class of decomposition methods is *separable programming methods* based on duality [1]. By decomposing a large problem into multiple much simpler subproblems, they have similar advantages as our constraint partitioning approach. However, they are limited in their general applications because they have restricted assumptions, such as linearity or convexity of functions. In this paper, we study a general constrained optimization approach with no restricted assumptions on constraint functions. Instead of using duality, we build our theoretical foundation on a novel penalty formulation discussed in the next section.

3 Constraint Partitioning by Penalty Formulations

In this section, we summarize our theory of extended saddle points (ESP). Our goal in solving P_m is to find a constrained local minimum $z^* = (x^*, y^*)$ with respect to

$\mathcal{N}_m(z^*)$, the mixed neighborhood of z^*. Due to space limitations, we only summarize some high-level concepts without the precise formalism [15].

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

$$\mathcal{N}_m(z) = \Big\{(x', y) \mid x' \in \mathcal{N}_c(x)\Big\} \cup \Big\{(x, y') \mid y' \in \mathcal{N}(y)\Big\}, \tag{4}$$

where $\mathcal{N}_c(x) = \{x' : \|x' - x\| \le \epsilon$ and $\epsilon \to 0\}$ is the continuous neighborhood of x, and the discrete neighborhood $\mathcal{N}(y)$ is a finite *user-defined set of points $\{y' \in \mathbb{D}^w\}$.*

Definition 2. *Point z^* is a CLM_m, a constrained local minimum of P_m with respect to points in $\mathcal{N}_m(z^*)$, if z^* is feasible and $f(z^*) \le f(z)$ for all feasible $z \in \mathcal{N}_m(z^*)$.*

Definition 3. *The penalty function of P_m with penalty vectors $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^r$ is:*

$$L_m(z, \alpha, \beta) = f(z) + \alpha^T |h(z)| + \beta^T \max(0, g(z)). \tag{5}$$

Theorem 1. *Necessary and sufficient ESPC on CLM_m of P_m* [15]. *Assuming $z^* \in \mathbb{R}^v \times \mathbb{D}^w$ of P_m satisfies a constraint-qualification condition (not shown due to space limitations), then z^* is a CLM_m of P_m iff there exist finite $\alpha^* \ge 0$ and $\beta^* \ge 0$ that satisfies the following extended saddle-point condition (ESPC):*

$$L_m(z^*, \alpha, \beta) \;\le\; L_m(z^*, \alpha^{**}, \beta^{**}) \;\le\; L_m(z, \alpha^{**}, \beta^{**}) \tag{6}$$

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

Note that the condition in (6) is rather loose because it only needs to be satisfied for any α^{**} and β^{**} that are larger than some critical α^* and β^*. The theorem is important because it establishes a one-to-one correspondence between a CLM_m z^* of P_m and an ESP of the corresponding unconstrained penalty function in (5) when penalties are sufficiently large. Moreover, it leads to a way for finding CLM_m. Since an ESP is a local minimum of (5) (but not the converse), z^* can be found by increasing gradually the penalties of violated constraints in (5) and by finding repeatedly local minima of (5) until a feasible solution to P_m is obtained. This is practical because there exist many search algorithms for locating the local minima of unconstrained nonlinear functions.

The ESPC in Theorem 1 has two features that distinguish it from the traditional penalty theory. First, because the ESPC can be satisfied by many possible penalty values, the search of these penalties can be carried out in a partitioned fashion in which each subproblem is solved by looking for any penalty values that are larger than α^* and β^*. This is not possible if the search were formulated as the solution of a system of nonlinear equations as in the KKT condition, or as the search of a single penalty term in the ℓ_1-penalty function in (3). Second, the condition is developed for general constraint functions and does not require continuity and differentiability as in the KKT condition. Further, it can be implemented by looking for the local minima of a nonlinear penalty function, and not for the global minima as in the general penalty theory.

Consider P_t, a version of P_m whose constraints can be partitioned into N stages. Stage t, $t = 1, \ldots, N$, has local *state vector* $z(t) = (z_1(t), \ldots, z_{u_t}(t))^T$, where $z(t)$

includes all the variables that appear in any of the local constraints in stage t. Note that since the partitioning is by constraints, $z(1), \ldots, z(N)$ may overlap with each other.

$$\begin{aligned} (P_t): \quad & \min_z \; J(z) && && (7)\\ \text{subject to} \quad & h^{(t)}(z(t)) = 0, && g^{(t)}(z(t)) \le 0 && \text{(local constraints)}\\ \text{and} \quad & H(z) = 0, && G(z) \le 0 && \text{(global constraints).} \end{aligned}$$

Here, $h^{(t)} = (h_1^{(t)}, \ldots, h_{m_t}^{(t)})^T$ and $g^{(t)} = (g_1^{(t)}, \ldots, g_{r_t}^{(t)})^T$ are local-constraint functions in stage t that involve $z(t)$; and $H = (H_1, \ldots, H_p)^T$ and $G = (G_1, \ldots, G_q)^T$ are global-constraint functions that involve $z \in \mathcal{X} \times \mathcal{Y}$.

Without showing the details [15], we first describe intuitively $\mathcal{N}_b(z)$, the mixed neighborhood of z in P_t. $\mathcal{N}_b(z)$ is made up of N neighborhoods, each perturbing z in one of the stages of P_t, while keeping the overlapped variables consistent across the other stages. Next, by considering P_t as a MINLP and by defining the corresponding penalty function, we apply Theorem 1 and derive the ESPC of P_t. Finally, we decompose the ESPC into N necessary conditions, one for each stage, and an overall necessary condition on the global constraints across the subproblems.

The partitioned condition in stage t can be satisfied by finding the ESPs in that stage. Because finding an ESP is equivalent to solving a MINLP, we can reformulate the search in stage t as the solution of the following optimization problem:

$$\left(P_t^{(t)}\right): \quad \min_{z(t)} \; J(z) + \gamma^T |H(z)| + \eta^T \max(0, G(z)) \qquad (8)$$
$$\text{subject to} \quad h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \le 0.$$

The weighted global-constraint violations in the objective of $P_t^{(t)}$ are important because they lead to points that minimize such violations. When they are large enough, solving $P_t^{(t)}$ will lead to points, if they exist, that satisfy the global constraints.

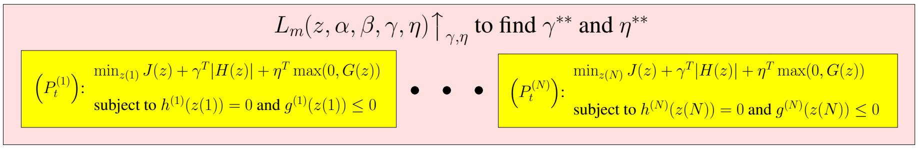
4 Partitioning and Resolution Strategies

Figure 5 presents CPOPT, a partition-and-resolve procedure for solving P_t. It first partitions the constraints into N subproblems (Line 2 of Figure 5b, discussed in Section 4.1). With fixed γ and η, it then solves $P_t^{(t)}$ in stage t using an existing solver (Line 6). To allow $P_t^{(t)}$ to be solvable by an existing solver that requires a differentiable objective function, we transform $P_t^{(t)}$ into the following equivalent MINLP:

$$\min_{z(t)} \; J(z) + \gamma^T a + \eta^T b \qquad (9)$$
$$\begin{aligned} \text{subject to} \quad & h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \le 0,\\ & -a \le H(z) \le a \quad \text{and} \quad G(z) \le b, \end{aligned}$$

where a and b are non-negative auxiliary vectors. After solving each subproblem, we increase γ and η on the violated global constraints (Line 7, discussed in Section 4.2). The process is repeated until a CLM_m to P_t is found or when γ and η exceed their maximum bounds (Line 9, discussed in Section 4.2).

We describe below the partitioning of the constraints and the update of the penalties.


$L_m(z,\alpha,\beta,\gamma,\eta)\uparrow_{\gamma,\eta}$ to find γ^{**} and η^{**}
$\left(P_t^{(1)}\right)$: $\min_{z(1)} J(z)+\gamma^T|H(z)|+\eta^T\max(0,G(z))$
subject to $h^{(1)}(z(1))=0$ and $g^{(1)}(z(1))\le 0$
• • •
$\left(P_t^{(N)}\right)$: $\min_{z(N)} J(z)+\gamma^T|H(z)|+\eta^T\max(0,G(z))$
subject to $h^{(N)}(z(N))=0$ and $g^{(N)}(z(N))\le 0$

a) The partition-and-resolve framework to look for CLM_m of P_t

```
1. procedure CPOPT
2.      call automated_partition(); // automatically partition the problem //
3.      $\gamma \longleftarrow \gamma_0$; $\eta \longleftarrow \eta_0$; // initialize penalty values for global constraints//
4.      repeat      // outer loop //
5.          for $t = 1$ to $N$ // iterate over all $N$ stages to solve $P_t^{(t)}$ in stage $t$ //
6.              apply an existing solver to solve $P_t^{(t)}$;
7.              call update_penalty(); // update penalties of violated global constraints //
8.          end_for;
9.      until stopping condition is satisfied;
10. end_procedure
```

b) CPOPT: Implementation of the partition-and-resolve framework

Fig. 5. The partition-and-resolve procedure to look for CLM_m of P_t

4.1 Strategies for Partitioning Constraints into Subproblems

Our goal in Line 2 of Figure 5b is to partition the constraints in such a way that minimizes the overall search time. Since the enumeration of all possible ways of partitioning is computationally prohibitive, we restrict our strategy to only partitioning by index vectors of problems modeled by the AMPL language [5].

Definition 4. *An index vector V in an AMPL model is a finite ordered array of discrete elements that are used to index variables and constraints.*

For example, TRIMLON12 described in Section 1 has two index vectors: $I = J = \{1, \cdots, 12\}$. A variable or a constraint function can be indexed by one or more index vectors: $n[i, j], i \in I, j \in J$, is indexed by I and J; and (C5) is indexed by I alone.

Definition 5. *A partitioning index vector (PIV) of an AMPL model is an index vector in the model that is used for partitioning the constraints.*

Definition 6. *Constraint partitioning by PIV. Given a PIV of an AMPL model, an N-partition by the PIV is a collection of subsets of the PIV, $S_1, \cdots, S_N$, where a) $S_i \in PIV$; b) $S_1 \cup \cdots \cup S_N = PIV$; and c) $S_i \cap S_j = \varnothing$ for $i \neq j$ and $i, j = 1 \ldots N$.*

The constraints of a problem can be partitioned along one or more index vectors. With multiple index vectors, the Cartesian-product space of the PIVs is partitioned into subsets. For instance, we have shown in Section 1 the partitioning of TRIMLON12 by J into $N = 12$ subproblems; that is, PIV $= \{J\}$, and $S_1 = \{1\}, \cdots, S_{12} = \{12\}$. This allows all the constraints indexed by J (C1 to C4) to be grouped into local constraints, and those not indexed by J (C5) to be the global constraints.

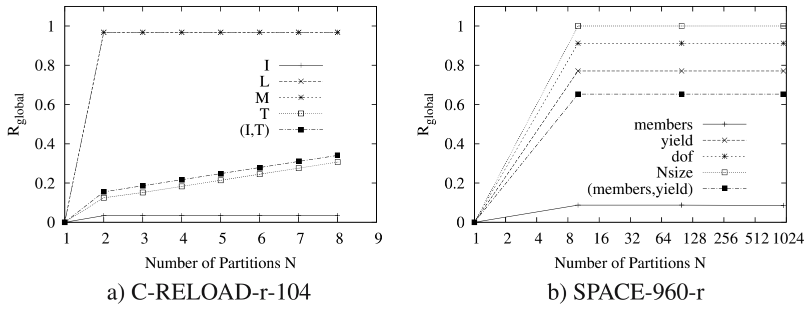


Fig. 6. Ratio of global constraints when partitioned by different PIVs for two MINLPs

We argue that it is reasonable and effective to partition constraints by their index vectors. First, indexing is essential in modeling languages like AMPL and GAMS for representing a complex problem in a compact form. Without it, it will be very cumbersome to use a unique name for each variable, especially when there are thousands of variables and constraints. Second, index vectors in large application problems are typically associated with physical entities. When constraints are partitioned by their index vectors, the partitions can be interpreted meaningfully. For example, index vector J in TRIMLON12 corresponds to the possible cuts of paper rolls, and a subproblem partitioned by J entails the optimization of the individual paper production in each cut.

Given a MINLP specified in AMPL, we present in the following our approach to automatically partition the problem by its constraints. We propose a metric to measure the quality of partitioning, present an algorithm to select the optimal PIV, illustrate the trade-offs between the number of partitions and the overall complexity, and show an efficient heuristic for determining the optimal number of partitions.

a) *Metric of partition-ability.* Since the time to solve a partitioned problem is largely driven by the overhead in resolving its inconsistent global constraints, we define R_{global} to be the ratio of the number of global constraints to the number of all constraints. This metric also needs to account for the shared variables in multiple subproblems that must be consistent with each other. For simplicity, we assume each shared variable v that appears in k subproblems to be equivalent to $k-1$ global constraints, where the i^{th} constraint involves the consistency between the i^{th} copy and the $i+1^{st}$ copy. Note that the metric is heuristic because the exact overhead depends on the difficulty of resolving the inconsistent global constraints and not on the number of global constraints.

b) *Selection of PIV.* To select the best PIV that minimizes R_{global}, we observe from the benchmarks tested that the best PIV for a problem instance is independent of the number of partitions N. To illustrate this observation, Figure 6 plots the value of R_{global} for various PIVs as a function of N for two benchmarks. It shows that the best PIV that minimizes R_{global} is the same for all N. Based on this property, we first fix an arbitrary value of N in our implementation. As there are usually less than five index vectors in a model file, we just enumerate all possible combinations of PIVs, compute R_{global} for each case, and pick the one that minimizes R_{global}.

c) *Number of partitions.* Based on the best PIV selected, we decide next the number of partitions. Experimentally, we have observed a convex relationship between N and

Table 1. Trade-offs between N and the total solution time on the SPACE-960-r MINLP

Number of partitions N	1	15	$\boxed{30}$	60	120	240	480
Time per subproblem	>3600	8.4	$\boxed{3.3}$	3.1	2.8	2.7	2.6
Time per iteration	>3600	126	$\boxed{99}$	186	336	648	1248
Number of iterations	1	1	$\boxed{1}$	2	2	2	5
Total time to solve problem	>3600	126	$\boxed{99}$	372	672	1296	6240

```
1. procedure optimal_number_of_partitions (PIV)
2.     $N \longleftarrow |PIV|$;  $last\_time \longleftarrow \infty$ ;
3.     repeat
4.         evaluate a subproblem under $N$ partitions, and record the solution time $T_p(N)$;
5.         $overall\_time \longleftarrow T_p(N) \cdot N$;
6.         if ($overall\_time > last\_time$) then return ($2N$);
7.         $last\_time \longleftarrow overall\_time$;
8.         $N \longleftarrow N/2$ ;
9.     end_repeat
10. end_procedure
```

Fig. 7. An iterative algorithm to estimate the optimal number of partitions

the total solution time. We illustrate this observation in Table 1 for various values of N on the SPACE-960-r MINLP from the MacMINLP library [12]. It shows the average time to solve a subproblem, the total time to solve N subproblems in one iteration, the number of iterations needed to resolve the inconsistent global constraints, and the overall time to solve the problem. The best N for this problem is 30.

The convex relationship is intuitively reasonable. When the number of partitions is small or when there is no partitioning, the global constraints will be few in number and easy to revolve, but each subproblem is large and expensive to evaluate. On the other hand, when there are many partitions, each subproblem is small and easy to evaluate, but there will be many global constraints that are hard to resolve.

The convex relationship allows us to determine an optimal number of partitions that minimizes the overall solution time. We start with the maximum number of partitions in the original problem (Line 2 of Figure 7) and evaluate a few subproblems in order to estimate $T_p(N)$, the average time to solve a subproblem when there are N partitions (Line 4). We also evaluate $overall\_time$, the time to solve all the subproblems once (Line 5). Assuming the number of iterations for resolving the global constraints to be small, $overall\_time$ will be related to the time to solve the original problem by a constant factor. This assumption is generally true for the benchmarks tested when N is close to the optimal value (as illustrated in Table 1). Next, we reduce N by half (Line 8) and repeat the process. We stop the process when we hit the bottom of the convex curve and report $2N$ that leads to the minimum $overall\_time$ (Line 6).

The algorithm requires $T_p(N)$, which can be estimated accurately based on the observation that it has little variations when the constraints are partitioned evenly. Table 2 illustrates this observation and shows that the standard deviation of the time to evaluate a subproblem is very small for two values of N. As a result, we only evaluate one subproblem in each iteration of Figure 7 in order to estimate $T_p(N)$ (Line 4).

Table 2. Average and standard deviation of solution time per subproblem for two benchmarks

Problem instance	ORTHRGDS		SPACE-960-r	
Number of partitions N	1000	20	100	10
Avg. time per subproblem ($T_p(N)$)	1.8	8.5	2.8	9.4
Std. dev. of time per subproblem	0.021	0.31	0.013	0.015

For the SPACE-960-r MINLP in Table 1, we set N to 480, 240, 120, 60, 30, 15. We stop at $N = 15$ and report $N = 30$ when $overall\_time$ starts to increase. The total time for solving the six subproblems is only 22.9 seconds, which is small when compared to the 160.45 seconds required by CPOPT for solving the original problem (see Table 3).

4.2 Strategies for Updating Penalty Values

After solving each subproblem, we use the following formulas to update the penalty vectors γ and μ of violated global constraints (Line 7 of Figure 5b):

$$\gamma \longleftarrow \gamma + \rho^T |H(z)|, \qquad \eta \longleftarrow \eta + \varrho^T \max(0, G(z)), \tag{10}$$

where ρ and ϱ are vectors for controlling the rate of updating γ and η.

We update each element of ρ and ϱ dynamically until the corresponding global constraint is satisfied. Vector ρ is initialized to ρ_0 and is updated as follows. For each global constraint H_i, $i = 1, \cdots, p$, we use c_i to count the number of consecutive subproblem evaluations in which H_i is violated since the last update of ρ_i. After solving a subproblem, we increase c_i by 1 if H_i is violated; if c_i reaches threshold K, which means that H_i has not been satisfied in K consecutive subproblem evaluations, we increase ρ_i by:

$$\rho_i \longleftarrow \rho_i \cdot \alpha, \qquad \text{where } \alpha > 1, \tag{11}$$

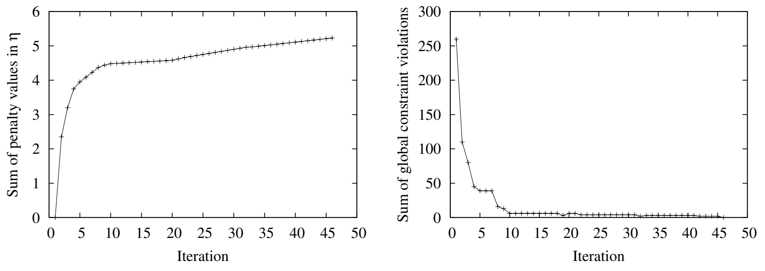
and reset c_i to 0. If H_i is satisfied, we reset ρ_i to ρ_0 and c_i to 0. In our implementation, we choose $\rho_0 = 0.01$, $K = 3$ and, $\alpha = 1.25$. We update ϱ in the same manner.

The procedure in Figure 5 may generate fixed points of (5) that do not satisfy Theorem 1. This happens because an ESP is a local minimum of (5) but not the converse. One way to escape from infeasible fixed points of (5) is to allow periodic decreases of γ and η (Line 7 of Figure 5b). These decreases "lower" the barrier in the penalty function and allow local descents in the inner loop to escape from an infeasible region. In our implementation, we scale down γ and η by multiplying each penalty by a random value between 0.4 and 0.6 if we cannot decrease the maximum violation of the global constraints or improve the objective after solving five consecutive subproblems.

Example. Consider the partitioning of TRIMLON12 into 12 subproblems along index J and the solution of the following $P_t^{(t)}$ in Stage j:

$$\min_{z=(y,m,n)} \; f(z) + \sum_{i=1}^{I} \left(\eta[i] \cdot \max \left(0, Nord[i] - \sum_{j=1}^{J} m[j] \cdot n[i,j] \right) \right)$$

subject to: local constraints (C1) - (C4) for Subproblem j, $j = 1, \cdots, 12$,



a) Sum of penalty values in η b) Sum of global constraint violations (C5)

Fig. 8. Illustration of solving TRIMLON12 by CPOPT

where η is the penalty vector for the global constraints (C5). Using the penalty update strategy discussed, Figure 8 shows the change on the sum of all penalty values in η and the sum of the violations on the global constrains as CPOPT is run. The search terminates in 46 iterations when all the global constraints are resolved.

5 Experimental Results

In this section, we compare the performance of CPOPT to that of other leading solvers. In CPOPT, if $P_t^{(t)}$ is a MINLP, CPOPT first generates a good starting point by solving it as a CNLP using SNOPT [7] without the integrality requirement, before solving it by MINLP_BB [11]. If $P_t^{(t)}$ is a CNLP, CPOPT applies SNOPT to solve it directly.

We have compared CPOPT to two of the best MINLP solvers, MINLP_BB [11] and BARON [14], on a collection of MINLP benchmarks from the MacMINLP library [12]. MINLP_BB implements a branch-and-bound algorithm with a sequential-quadratic-programming (SQP) solver for solving continuous subproblems, whereas BARON is a mixed-integer constrained solver implementing the branch-and-reduce algorithm. Of the 43 benchmarks in MacMINLP, we only show the results on 22 in Table 3. The remaining 21 benchmarks are all small problems and can be solved easily by all three solvers in tens of seconds or less. For these 21 benchmarks, the average solution times for CPOPT, BARON, and MINLP_BB are, respectively 8.40 seconds, 4.59 seconds, and 5.45 seconds. CPOPT is slower in solving these small problems due to its overhead in partitioning and in resolving the violated global constraints.

Note that although branch-and-bound methods, such as BARON and MINLP_BB, are theoretically complete methods that will converge to global optima, it is difficult to achieve global optimality in practice. BARON reports the best feasible solution found during its search until it times out in the 3600-sec time limit. For large problems, the gap between the lower and upper bounds usually does not vanish before termination, which implies that the solution found may not be optimal. Similarly, MINLP_BB reports the best solution found before it times out or runs out of memory.

We have also compared CPOPT to two of the best CNLP solvers, Lancelot (a solver implementing an augmented Lagrangian method) [3] and SNOPT (an SQP solver) [7] on the CNLPs from the CUTE library [2]. Table 3 summarizes only the results on

Table 3. Results on solving MINLP benchmarks from the MacMINLP library [12] and CNLP benchmarks from the CUTE library [2]. Results on MINLP_BB and BARON were obtained by submitting jobs to the NEOS server (*http://www-neos.mcs.anl.gov/neos/*) and BARON's site (*http://archimedes.scs.uiuc.edu/baron/baron.html*), respectively; results of other solvers were collected on an AMD Athlon MP2800 PC running RH Linux AS4 and a time limit of 3,600 sec. All timing results are in sec and should be compared only within a solver. For each instance, n_c and n_v represent, respectively, the number of constraints and the number of variables. Solutions with the best quality are boxed. "–" means that no feasible solutions were found in the time limit.

ID	n_c	n_v	Quality	Time	Quality	Time	Quality	Time
MINLP Test Problem			MINLP_BB		BARON		CPOPT(MINLP_BB)	
C-RELOAD-q-49	1430	3733	–		–	–	$\boxed{-1.13}$	69.45
C-RELOAD-q-104	3338	13936	–	–	–	–	$\boxed{-1.14}$	353.74
Ex12.6.3	57	92	$\boxed{19.6}$	23	$\boxed{19.6}$	423.1	$\boxed{19.6}$	13.43
Ex12.6.4	57	88	$\boxed{8.6}$	70	$\boxed{8.6}$	478.2	$\boxed{8.6}$	2.94
Ex12.6.5	76	130	15.1	4	$\boxed{10.3}$	845.5	10.6	3.33
Ex12.6.6	97	180	$\boxed{16.3}$	18	$\boxed{16.3}$	937.4	$\boxed{16.3}$	149.40
PUMP	34	24	–	–	131124	977	$\boxed{130788}$	84.53
SPACE-960-i	6497	5537	–	–	–	–	$\boxed{7.65E6}$	187.43
SPACE-960-ir	3617	2657	–	–	–	–	$\boxed{7.64E6}$	145.76
SPACE-960	8417	15137	–	–	–	–	$\boxed{7.84E6}$	1206.43
SPACE-960-r	5537	12257	–	–	–	–	$\boxed{5.13E6}$	160.45
STOCKCYCLE	97	480	–	–	436341	n/a	$\boxed{119948.7}$	6.45
TRIMLON4	24	24	12.2	10	$\boxed{8.3}$	11.0	$\boxed{8.3}$	2.73
TRIMLON5	30	35	12.5	14	$\boxed{10.3}$	55.3	$\boxed{10.3}$	24.5
TRIMLON6	36	48	18.8	19	$\boxed{15.6}$	1092.9	$\boxed{15.6}$	15.94
TRIMLON7	42	63	–	–	$\boxed{17.5}$	990.7	18.1	65.34
TRIMLON12	72	168	–	–	–	–	$\boxed{95.5}$	345.50
TRIMLOSS4	64	105	10.8	99	–	–	$\boxed{10.6}$	9.76
TRIMLOSS5	90	161	12.6	190	–	–	$\boxed{10.7}$	76.85
TRIMLOSS6	120	215	–	–	–	–	$\boxed{22.1}$	69.03
TRIMLOSS7	154	345	–	–	–	–	$\boxed{26.7}$	59.32
TRIMLOSS12	384	800	–	–	–	–	$\boxed{138.8}$	323.94
CNLP Test Problem			Lancelot		SNOPT		CPOPT(SNOPT)	
CATENARY	166	501	-	-	-	-	$\boxed{-1.35E5}$	245.64
DTOC6	5000	10001	-	-	-	-	$\boxed{1.02E6}$	58.05
EIGMAXB	101	101	$\boxed{0.91}$	1.34	-	-	1.87	24.33
GILBERT	1000	1000	2459.46	1.12	4700.61	689.18	$\boxed{2454.67}$	39.55
HADAMARD	256	129	-	-	-	-	$\boxed{0.99}$	7.88
KISSING	903	127	0.84	123.43	-	-	$\boxed{0.77}$	73.45
OPTCDEG	4000	6001	-	-	$\boxed{45.76}$	10.23	46.98	19.65
ORTHREGC	5000	10005	-	-	3469.05	557.98	$\boxed{2614.34}$	143.65
ORTHREGD	5000	10003	-	-	8729.64	208.27	$\boxed{7932.92}$	123.49
ORTHRGDM	5000	10003	$\boxed{1513.80}$	4.56	10167.82	250.00	2340.34	20.34
ORTHRGDS	5000	10003	912.41	4.20	-	-	$\boxed{894.65}$	105.34
VANDERM1	199	100	-	-	-	-	$\boxed{0.0}$	45.34
VANDERM3	199	100	-	-	-	-	$\boxed{0.0}$	36.70
VANDERM4	199	100	-	-	-	-	$\boxed{0.0}$	52.33

the 14 CUTE benchmarks that either Lancelot or SNOPT has difficulty with. For the remaining CUTE benchmarks that are easy to solve, the average solution times for Lancelot, SNOPT, and CPOPT are, respectively, 23.43 seconds, 13.04 seconds, and 19.34 seconds. For the same reason as before, CPOPT is slower in solving those small problems due to its additional overhead. The results show that, for those difficult-to-solve CUTE benchmarks, CPOPT can find the best solution, that it is one to two orders of magnitude faster, and that it scales well.

References

1. D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 1999.
2. 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.
3. A. R. Conn, N. Gould, and Ph. L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming*, 73:73–110, 1996.
4. M. A. Duran and I. E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:306–307, 1986.
5. R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks Cole Publishing Company, 2002.
6. A. M. Geoffrion. Generalized Benders decomposition. *J. Optim. Theory and Appl.*, 10(4):237–241, 1972.
7. P. E. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.
8. N. I. M. Gould, D. Orban, and Ph. L. Toint. An interior-point ℓ_1-penalty method for nonlinear optimization. Technical report, RAL-TR-2003-022, Rutherford Appleton Laboratory Chilton, Oxfordshire, UK, 2003.
9. I. Harjunkoski, T. Westerlund, R. Pörn, and H. Skrifvars. Different transformations for solving non–convex trim loss problems by MINLP. *European Journal of Operations Research*, 105:594–603, 1998.
10. K. Holmberg. On the convergence of the cross decomposition. *Mathematical Programming*, 47:269–316, 1990.
11. S. Leyffer. Mixed integer nonlinear programming solver. *http://www-unix.mcs.anl.gov/~leyffer/solvers.html*, 2002.
12. S. Leyffer. MacMINLP: AMPL collection of MINLP problems. *http://www-unix.mcs.anl.gov/~leyffer/MacMINLP/*, 2003.
13. R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.
14. N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996.
15. B. Wah and Y. X. Chen. Fast temporal planning using the theory of extended saddle points for mixed nonlinear optimization. *Artificial Intelligence*, (accepted for publication) 2005.