

A Framework for Parallel Nonlinear Optimization by Partitioning Localized Constraints ^{*}

You Xu and Yixin Chen

Department of Computer Science and Engineering
Washington University in St. Louis
{yx2, chen} @cse.wustl.edu

Abstract

We present a novel parallel framework for solving large-scale continuous nonlinear optimization problems based on constraint partitioning. The framework distributes constraints and variables to parallel processors and uses an existing solver to handle the partitioned subproblems. In contrast to most previous decomposition methods that require either separability or convexity of constraints, our approach is based on a new constraint partitioning theory and can handle nonconvex problems with inseparable global constraints. We also propose a hypergraph partitioning method to recognize the problem structure. Experimental results show that the proposed parallel algorithm can efficiently solve some difficult test cases.

1 Introduction

Nonlinear optimization is an important problem that has abundant applications in science and engineering. In this paper, we study continuous nonlinear programming (NLP) problems formulated as follows:

$$\begin{aligned} \text{(P)} : \quad & \min_x && f(x), && (1) \\ & \text{subject to} && h(x) = 0 \text{ and } g(x) \leq 0, \end{aligned}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^r$ are twice continuously differentiable functions. We denote by $\mathcal{F} = \{x \in \mathbb{R}^n | g(x) \leq 0, h(x) = 0\}$ the feasible set that we assume nonempty. We further define $\mathcal{C} = \{h(x), g(x)\}$ as the constraints set and $\mathcal{X} = \{x_i, i = 1, \dots, n\}$ as the variable set.

^{*}This work is supported by a Microsoft Research New Faculty Fellowship and an ECPI grant from the Department of Energy.

The parallel optimization framework presented in this paper is designed to handle P by solving a sequence of specially constructed smaller-scale (in terms of both number of variables and number of constraints) optimization problems in parallel. By using partitioning and parallel computing, we aim to make the solver more scalable.

1.1 Observation

Our key observation is that most application-based NLPs have structured and localized arrangements of constraints. Most constraints only involve a small subset of the variables instead of all.

To formalize this observation and exploit the sparse nature of nonlinear optimization problems, for a given function f , we define the related variable set $v(f)$ as the set of variables involved in the closed form of f .

For instance, if an NLP problem with five variables $\{x_1, \dots, x_5\}$ has an objective function f as $f = 2x_1 + \sin(x_2) - x_4x_5$, the related variable set $v(f)$ is $\{x_1, x_2, x_4, x_5\}$. The definition of related variable set for a single function can easily be generalized to a set of functions, where $v(f_1, f_2, \dots, f_n)$ is defined as the union of $v(f_1), v(f_2), \dots, v(f_n)$. For problem P , \mathcal{X} is actually $v(f, g, h)$.

The structure of constraints can be exploited based on the definition above. To visualize this idea, we index all constraints and variables and denote them by c_1, c_2, \dots, c_m and x_1, x_2, \dots, x_n . Then, we plot a *relationship graph* using the constraint index as the x-axis and variable index as the y-axis. A dot is printed on (i, j) if and only if $x_j \in v(c_i)$.

Figure 1a shows the constraint-variable relationship graph of the Bratu3D problem in the CUTE library [3]. The coefficient matrix of Bratu3D is a well-known symmetric sparse matrix [5]. The corresponding relationship graph is also sparse, symmetric and strongly localized. Figure 1b shows the constraint-variable relationship graph of the Britgas problem in the CUTE library. It is a real-world problem that formulates the high pressure gas network problem for British Gas [3]. It is also sparse. For any given constraint, it is only related to a small set of variables.

For many nonlinear optimization problems, the constraints are not only sparse but also has a regular structure. This is usually because of the spatial or temporal nature of the problem, as constraints are usually local spatial or temporal restrictions. Therefore, our intuitive idea is to group constraints and variables such that all the constraints in each group are only related to a small set of variables. We use the term “partitioning” to denote such a grouping procedure. Other literature have also used the term “decomposition”.

For some problems, partitioning strategies are straightforward. Take the “Distribution of Electrons on a Sphere” (ELEC) problem in the COPS benchmark [6] as an example. Given n_p electrons, the goal is to find the equilibrium state distribution of the electrons positioned on a conduction sphere, which minimizes the Coulomb potential. The problem can be formulated as

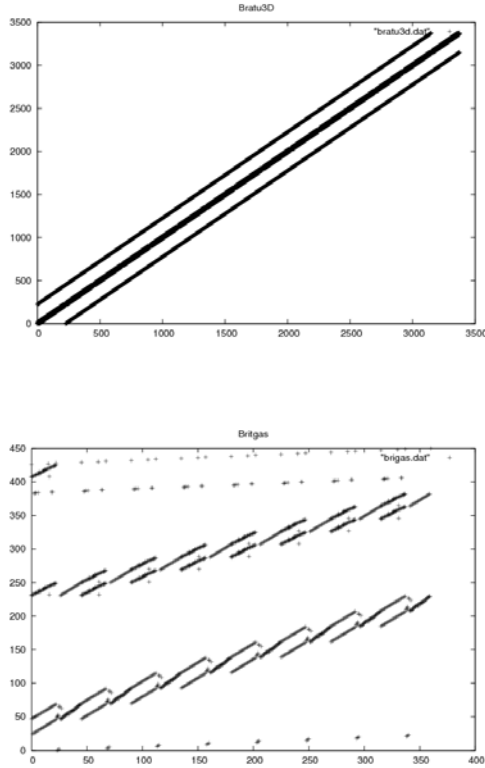


Figure 1: Constraint-Variable relationship for Bratu3D and Britgas

$$\min f(x, y, z) = \sum_{i=1}^{n_p-1} \sum_{j=i+1}^{n_p} ((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2)^{-\frac{1}{2}}, \quad (2)$$

$$\text{s.t.} \quad x_i^2 + y_i^2 + z_i^2 = 1, \quad i = 1, \dots, n_p \quad (3)$$

In ELEC, all constraints are highly localized and only related to three variables which are the 3-D coordinates for an electron. Since each constraint is localized to three related variables, objective function can be minimized for every three variables subject to one constraint without affecting the feasibility of other constraints. In this paper, we define constraints of this kind as *local constraints*.

Generally, for large-scale NLP problems, localized arrangement of constraints and variables provide opportunities for reducing the computational cost. However, in reality, not all constraints are localized. We still take an example from

the CUTE library. The “EIGMINB” problem in CUTE is to find the eigenvector and the smallest eigenvalue for a given symmetric matrix A . Namely, the goal is to find a unit vector q and scalar d such that $Aq = dq$ for which d is least. This problem is formulated as an optimization problem with a constraint $\|q\| = 1$ and the objective function $\min d$. If we treat every separate element of the q vector as a real variable, it is obvious that this particular constraint involves all but one variable d . Thus, this constraint is not naturally separable. In our framework, we identify constraints of this sort as *global constraints* and define a biased penalty function for resolving such global constraints during parallel optimization.

1.2 Framework Outline

In our framework, the classification of *global constraints* and *local constraints* are based on the problem structure. Similar to the decomposition in linear programming, the sparse structure of constraints is the attribute we want to utilize. Our method is to rearrange the constraints and variables into a form that is as block-separable as possible.

Definition For a given nonlinear optimization problem with constraint set $\mathcal{C} = \{c_i(x)\}$, $\mathcal{P}(\mathcal{C})$ is the powerset of \mathcal{C} . Set $\mathcal{S} \subset \mathcal{P}(\mathcal{C})$ is a *block-separable constraint partition* if and only if $\{\mathcal{S}_i\}$ is a partition of \mathcal{C} and $\{v(\mathcal{S}_i)\}$ is a partition of \mathcal{V} . Thus, $\forall i \neq j, \forall \mathcal{S}_i, \mathcal{S}_j$ in \mathcal{S} , $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$, $\bigcup \mathcal{S}_i = \mathcal{C}$ and $v(\mathcal{S}_i) \cap v(\mathcal{S}_j) = \emptyset$, $\bigcup v(\mathcal{S}_i) = \mathcal{V}$.

If the constraints are block-separable, the Block Coordinate Descent [15] method introduced in the next subsection can be used to solve it. However, more often than not, there are some constraints that are related to variables across multiple groups and there is no way to avoid them by regrouping. To handle this, we introduce the concept of general constraint partition as follows.

Definition For a given nonlinear optimization problem with constraint set $\mathcal{C} = \{c_i(x)\}$, $\mathcal{P}(\mathcal{C})$ is the powerset of \mathcal{C} . Set $\mathcal{S} \subset \mathcal{P}(\mathcal{C})$ is a *general constraint partition* if \mathcal{S}' is a partition of \mathcal{C} with $m + 1$ ($m \in \mathbb{Z}^+$) parts and $\{v(\mathcal{S}'_j), j = 1, \dots, m\}$ is a partition of \mathcal{X} .

In the definition of general constraint partition, we allow constraints in the $m + 1^{th}$ group to relate to variables across multiple partitions. Similar to a block-separable partition, all the constraints within the same group in first m groups are only related to a small portion of variables and any constraints that in different groups do not share any variables. Therefore, all the variables are also hereby partitioned into m non-intersecting groups. Based on this definition, we formally define the *global constraint* set \mathcal{C}_2 as the set of all constraints residing in the $m + 1^{th}$ group. The set of constraints in the first m groups is correspondingly defined as the *local constraint* set, denoted by \mathcal{C}_1 . Now in the EIGMINB example, we can assign constraint $\|q\| = 1$ as a global constraint.

Our proposed framework consists of three steps. The first step is to analyze the problem structure and find a general constraint partition with $m + 1$ loosely coupled groups. This is introduced in Section 3. Then, in the next step, we use a biased objective function defined in a general form as

$$\Phi(x, \delta, \mathcal{C}_2) = f(x) + p(\delta, \mathcal{C}_2(x)), \quad (4)$$

where δ is a parameter vector and $p(\cdot)$ is a penalty function with $p > 0$ when any constraint \mathcal{C}_2 is unsatisfied. Φ function is defined such that the minimization of it will eventually resolve the general constraints in \mathcal{C}_2 . Subproblems are defined as the minimization of Φ with only local constraints and some v value. During the subproblem solving procedure, the feasibility of the local constraints are maintained. There are various ways to design the function Φ . Finally, a parallel algorithm presented in Section 2 is employed to solve the transformed problem.

1.3 Existing parallel optimization methods

In this subsection, we survey existing methods for solving large-scale optimization problems through parallel decomposition.

Unconstrained optimization. When an optimization problem is unconstrained, the only way to partition it is distributing variables into smaller subproblems. The proposed decomposition algorithms usually exploit the structure in objective function and decompose it into smaller scale subproblems. A synchronization step is followed to gather the solutions from subproblems. This procedure will iterate until some convergence conditions are satisfied. Coordinate Descent Method (CDM) [12], Parallel Gradient Descending (PGD) [13] and Block Coordinate Descent (BCD) [15] all decompose the objective function orthogonally along the variable coordinate. Those methods are highly parallel. PGD employs the same idea and emphasizes on the distribution of gradient as well. Its global convergence result is proved for convex and differentiable objective function [12]. Parallel Variable Distribution (PVD) [8] algorithm introduces a ‘forget-me-not’ term to allow the remaining variables (the variables not belonging to a subproblem) to change in a restricted fashion.

Constrained optimization with block-separable constraints. Algorithms in this category deal with constrained optimization with block-separable constraints. Usually the subproblem P_l only contains constraints in the current block l . The Block Jacobi Method [14], the Updated Conjugate Subspaces method [10] and the Coordinate Decent Method [16] all fall in this category. These methods are also highly parallel. However, since they neglect the global properties of the whole system, they might only gain a limited improvement during each iteration.

In summary, the previous parallel algorithms shed lights to the development of parallel algorithms for handling general global constraints. There are also some previous works for treating general constraints. However, they are restricted to convex and smooth functions [12] or quasi-convex and hemivariate functions [15]. In this paper, we further relax the assumptions over the objective

and constraint functions. In particular, we do not assume the convexity of the constraints.

2 Algorithmic Framework

In this section we present our algorithm framework for solving the following optimization problem P^t with a general constraint partition

$$(P^t) : \quad \begin{array}{ll} \min & f(x) \\ \text{subject to} & h^t(x) = 0, \quad g^t(x) \leq 0 \quad 1 \leq t \leq m \\ \text{and} & H(x) = 0, \quad G(x) \leq 0. \end{array}$$

where $\mathcal{C}_1 = \{g^t, h^t | 1 \leq t \leq m\}$ includes m block-separable constraints (local constraints) and $\mathcal{C}_2 = \{H, G\}$ contains global constraints. We further assume that the objective function f and constraints in $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2\}$ are all continuous and differentiable.

In our framework, all global constraints are incorporated to a biased function denoted as $\Phi(x, \delta, \mathcal{C}_2)$. Instead of solving P^t directly, we try to solve the following transformed problem:

$$(P_\Phi) : \quad \begin{array}{ll} \min & \Phi(x, \delta, \mathcal{C}_2) \\ \text{subject to} & h^t(x) = 0, \\ & g^t(x) \leq 0, \quad 1 \leq t \leq m, \end{array}$$

where δ is a penalty parameter which can be a vector or a positive scalar. We define constraint violation metric ν for inequality constraint $g(x) \leq 0$ as $\nu(g, x^0) = g^+(x^0) = \max(0, g(x^0))$ and $\nu(h, x^0) = |h(x^0)|$ for equality constraint $h(x) = 0$. This definition of constraint violation can be extended to a set of constraints \mathcal{C} by defining $\nu(\mathcal{C}_2, x^0) = \sum_{c \in \mathcal{C}_2} \nu(c(x^0))$. The design of the function Φ requires that when δ approaches its limit (a finite value or infinity), the function value of $\Phi : \mathbb{R}^n \mapsto [0, \infty)$ is $f(x)$ if $\nu(\mathcal{C}_2(x)) = 0$ and has a limit of $+\infty$ otherwise. Therefore, by forcing δ to approach its limit, the minimization of Φ will push x to a feasible point of P with the objective function also optimized.

Our framework for solving P_Φ is shown in Algorithm 1. Algorithm 1 presents **constraint partitioning optimization (CPO)**, a constraint partitioning based optimization framework for solving problem P . It first analyzes the problem's constraint structure and partitions the constraints into m subproblems as in P^t (Line 2-3 of Algorithm 1, discussed in Section 3). After the transformation, the constraint structure becomes block separable. With a fixed δ , it then transforms P_t to P_Φ and uses a ‘‘Parallel Inner Iteration’’ procedure to solve P_Φ . Suppose now \mathcal{V} is partitioned into m blocks x_1, \dots, x_m and $x_t \in \mathbb{R}^{n_t}$. We can distribute the m groups of constraints and variables into m parallel processors. The Parallel Inner Iteration is given in Algorithm 2, in which Lines 3 to 8 are distributed to multiple processors and executed in parallel.

```

1 Initialize  $x^{(0)}$  as the starting point ;
2 Partition the original problem  $P$  to  $P^t$  ;
3 Reformulate  $P^t$  to  $P_\Phi$  ;
4 while constraint violation larger than  $\eta^*$  or objective deduction larger than  $\epsilon^*$  do
5     Call Parallel Inner Iteration;
6     if  $\nu(\mathcal{C}_2, x^{(k)}) < \eta^*$  then
7         Output  $x^{(k)}$  as solution;
8     end
9     if  $\nu(\mathcal{C}_2, x^{(k)}) < \eta^k$  then
10        Call update  $\delta$  ;
11        Call update  $\eta^k$  ;
12         $k \leftarrow k + 1$ ;
13    else
14        Call update  $\delta$  ;
15         $k \leftarrow k + 1$ ;
16    end
17 end

```

Algorithm 1: The constraint partitioning optimization (CPO) framework

Various forms of Φ can be used. In this paper, we consider two cases, one based on the l_1 exact penalty function (denoted as CPO_{l_1}) and the other based on an augmented Lagrangian method (denoted as CPO_{al}).

CPO_{l_1} : the l_1 exact penalty method. This method transforms the objective function to

$$\Phi_{l_1}(x, \delta, \mathcal{C}_2) = f(x) + \gamma^T |H(x)| + \eta^T G^+(x)$$

where $|H|$ and G^+ are violation vectors for equal and inequality constraints. $|H(x)|$ is $(|H_1(x)|, |H_2(x)|, \dots, |H_r(x)|)$ and $G^+(x)$ is $(G_1^+(x), \dots, G_s^+(x))$. δ in the generic representation of Φ consists of two parts γ and η . Both are positive vectors to penalize the violated global constraints. Based on the Extended Saddle Point theory [4], the saddle point of Φ_{l_1} is the optimal point of P when γ and η are large enough. Thus, in the δ -update procedure in our framework, we increase γ and η if the global constraints are violated. The update δ procedure will increase γ and η by an amount proportional to the violation of the corresponding global constraints.

The advantages of using Φ_{l_1} are twofold. First, it is an exact penalty method, which means γ and η do not have to go to infinity to get the optimal solution. The other advantage is that this method is based on the ESP theory [4] which provides a theoretical framework for analyzing the convergence of the parallel search strategy described in the Parallel Inner Iteration. However, there are some disadvantages of this method. The major disadvantage is that the objective function is not smooth. Φ is not differentiable at some points where $|H|$

```

input:  $x^k, \epsilon, v$ 
1 while  $\|\nabla\Phi_t(x^k, v)\| \geq \epsilon$  do
2    $y = x^k$  ;
3   Parallelization for each subproblem  $P^t$  do
4     Call subproblem solver to solve  $P^t(x_t, x_{\bar{t}})$  and get  $z_t^k$  ;
5     if no solution found then
6       return no feasible solution
7     end
8   end
9   Synchronization Compute  $x^{k+1}$  such that
      
$$\Phi(x^{k+1}) \leq \min_{t \in \{1, \dots, m\}} \Phi_t^i(z_t^k).$$

10 end

```

Algorithm 2: Parallel Inner Iteration for problem Φ

and G^+ are not smooth on \mathcal{F} . The non-smoothness prevents the fathoming of directional gradient via the block-separable structure. For instance, when Φ is not differentiable with respect to x^0 , the equation $\nabla_d \Phi(x^0) = d \nabla \Phi(x^0)$ will no longer be true. Thus, at some point, the Φ value could still be deduced along a certain direction though in each x^t subspace it cannot be further improved. Another disadvantage of this method is the lack of global convergence. In fact, CPO_{l_1} might stop at some stationary point which is a local minimum for Φ but not a feasible point for P , as the Parallel Inner Iteration procedure is actually a local search method. One way to remedy this, as suggested in [4], is to scale down γ and η to escape from local minima. We adopt this method in our implementation. If in 5 consecutive steps, constraint violation cannot be reduced, we scale down γ and η by a factor of 0.2.

To handle the non-smooth nature of the l_1 -penalty function, we further adopt a smooth transformation of the objective function described as follows.

CPO_{al} : the Augmented Lagrangian method. As discussed above, non-smooth objective functions could lead the Parallel Inner Iteration step into a stationary point during minimization. An alternative to the non-smooth l_1 function is the augmented Lagrangian method. Φ now is defined as

$$\Phi(x, \delta, f, \mathcal{C}_1, \mathcal{C}_2) = f(x) + \frac{1}{2} \sum_{j=1}^{m_1} \rho_j \left(h_j(x) + \frac{\lambda_j}{\rho_j} \right)^2 + \frac{1}{2} \sum_{i=1}^{r_1} \rho_{m_1+i} \left(g_i(x) + \frac{\mu_i}{\rho_{m_1+i}} \right)_+^2, \quad (5)$$

where δ now is a compound vector of λ, ρ and μ .

In our implementation, the δ -update step is the same as that in a standard augmented Lagrangian method. Line 10 updates each element of λ by

$$\lambda_j^{k+1} \leftarrow \lambda_j^k + \rho_j * |h_j(x^{k+1})|$$

for equality constraints. The updating rule for inequality constraints is similar [2].

3 Implementation Details

In this section, we introduce some implementation details.

Number of partitions. Intuitively, the total time to solve a partitioned problem is largely driven by the overhead in resolving inconsistent global constraints, since it cannot be parallelized. We want the number of unsatisfied global constraints to be small enough. In one extreme, if all constraints can reside in one partition, the number of global constraints is zero, but the solving time for this subproblem is longer. On the other end, if we allocate a group for each variable and its boundary constraints, then all but the boundary constraints will be global constraints.

We need to find a good trade-off. In the parallelization step, according to Amdahl’s law, we would hope that the number of partitions be as large as possible, so that the speedup in the parallelization step can be higher. Nevertheless, to make things more complicated, the number of partitions is also related to the problem structure. For instance, some problem structures are naturally bipartite and any further decomposition would inevitably destroy the regular structure. Due to such complexity, we take a practical method in the implementation. We support a manual and an automated method. In the manual method, we allow users to tag the constraints and variables manually as a user-defined partitioning scheme. In the automated method, we use a heuristic metric R_g to measure the quality of partitioning. R_g is defined as the ratio of global constraints to the number of constraints. In practice, we employ a third-party partitioning tool to find a partition that minimizes R_g .

Hypergraph based partitioning. Our automated approach is to convert the constraint partitioning problem to a hypergraph partitioning problem. Hypergraph is a generalization of a graph, where an edge can connect any number of vertexes. In our application, we treat variables as vertexes and constraints as hyper-edges. The hypergraph model is equivalent to the constraint-variable relationship graph introduced in Section 1. The advantage of using this model over the graph model is explained in [7, 1]. In the hypergraph partitioning model, our goal is to minimize the edge cuts across partitions, which is actually equivalent to minimizing the number of global constraints. As hypergraph partitioning is widely used in VLSI design and mesh design, there are many existing packages. Although the optimal hypergraph cut problem is NP-hard, lots of efficient heuristics are available. In our implementation, we use the hMetis package [11].

Table 1: Partitioning results on three testing problems.

Problem	Size		Partition Result			
	Var	Con	G	P	T	R
ELEC	900	300	0	10	1s	0.01
Bratu3D	3375	3375	880	6	10s	0.83
Britgas	450	360	68	8	3s	0.12

G: # of global constraints, P: # of partitions
T: partitioning time, R: ratio of T to total solving time

Table 2: Solver performance on three testing problems.

Problem	IPOPT			CPO_{al}			CPO_{t_1}		
	I	Q	T	I	Q	T	I	Q	T
ELEC	101	4.21e4	379s	273	4.21e4	81s	273	4.21e4	92s
Bratu3D	4	0	28s	8	0	11.92s	10	0	12.71s
Britgas	3000	-	-	971	2.47	257s	221	-	-

I: # of iterations. Q: solution quality. T: total solving time

4 Experimental results

In this section we report some preliminary results of the CPO framework. Our implementation is written in C and Python with Numpy modules. The current version uses IPOPT [17] as the nonlinear optimization solver for subproblems. To use IPOPT as a subroutine, we also developed Pyipopt [18], an open source interface to call IPOPT in Python. Our solver takes AMPL models [9] as input.

Currently the full MPI implementation is not completed. Instead, a shared memory model is tested on a Linux workstation with 4 AMD Opteron processors at 1GHz and 8GB RAM. As the full MPI version has not been implemented yet, the parallel solving time in one iteration of our experiment is measured as the longest time spent in solving any subproblem plus the time in the synchronization step. In all experiments, we set η^* , ϵ^* as 1.0^{-8} and ϵ as 1.0^{-4} .

The results on three test problem instances are shown in Table 1 and 2. In Table 1, the Var and Con columns denote the number of variables and constraints, respectively. The G, P and T columns are the number of global constraints, the number of partitions and the time to construct the partition, respectively. The last R column shows the percentage of the partitioning time with respect to the total solving time. The I, Q, T columns in Table 2 are the number of outer iterations, the quality of the solution, and the total solving time, respectively.

Example 1: Elec300. Consider the ELEC problem with 300 electrons. This problem has 900 variables and 300 constraints. Since the problem is block-separable, the difficulty here is the minimization of objective function and actually it has many local optimal KKT points.

As the problem is block-separable, an arbitrary number of partitions can be used. We manually set it to 10. The partitioning using hMetis takes about 1s. IPOPT takes about 3.6s in each iteration, while our parallel version took only about 0.3s even with synchronization steps in the Parallel Inner Iteration procedure. As there was no global constraints, $\Phi = f$. Our framework took 273 outer iterations to find $\nabla f \leq \epsilon^*$. Also note that in this case, augmented Lagrangian method and l_1 penalty method are actually identical, though the timing results are slightly different. From Table 2, we see that parallel optimization delivers the same quality and has the potential to significantly reduce time.

Example 2: Bratu3d. As we have introduced in the very beginning, the Bratu3d problem has a very sparse constraint structure. The objective function for this problem is simply a constant 0, thus finding a feasible point is the task.

It takes nearly 10s for hMetis to find the best number of partitions: 6. The actual optimization for each subproblem takes only 2 to 3s. Manually we find that if a better constraint partitioning strategy is used for this example, our framework will improve the solving time by an order of magnitude. The l_1 method is a little slower as the total number of iterations is larger.

Example 3: Britgas. British Gas (Britgas) is another problem in CUTE that has a sparse structure. IPOPT fails to solve the problem because the number of iterations exceeded its upper bound. In our experiment, the l_1 method also fails to solve it as the penalty term exceeds its upper bound after about 221 iterations. However, developed as a method for smooth problems, the augmented Lagrangian method solves this problem after 971 iterations. The average time for each inner iteration is about 0.12s and the initial partitioning takes about 3s to find the optimal number of partitions which is 8.

References

- [1] C. Aykanat, A. Pinar, and U. urek. Permuting sparse rectangular matrices into block-diagonal form, 2002.
- [2] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1982.
- [3] I. Bongartz, A. R. Conn, Nick Gould, and Ph.L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Trans. on Mathematical Software*, 21(1):123–160, 1995.
- [4] Yixin Chen. *Solving Nonlinear Constrained Optimization Problems Through Constraint Partitioning*. PhD thesis, Department of Computer Science, UIUC, 2005.
- [5] T. Davis. The university of florida sparse matrix collection. <http://www.cise.ufl.edu/research/sparse/matrices>.

- [6] E. Dolan, J. J. Moré, and T. S. Munson. Benchmarking optimization software with COPS 3.0. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 2004.
- [7] Michael C. Ferris and Jeffrey D. Horn. Partitioning mathematical programs for parallel solution. Technical Report CS-TR-1994-1232, 1994.
- [8] Michael C. Ferris and Olvi L. Mangasarian. Parallel variable distribution. *SIAM Journal on Optimization*, 4(4):815–832, 1994.
- [9] Robert Fourer, David M. Gay, and Brian W. Kernighan. Ampl: A mathematical programming language. Technical report.
- [10] S.P. Han. Optimization by updated conjugate subspaces. *Numerical Analysis*, (140):82–97, 1986.
- [11] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the Design and Automation Conference*, pages 343–348, 1999.
- [12] Z. Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *J. Optim. Theory Appl.*, 72(1):7–35, 1992.
- [13] O. L. Mangasarian. Parallel gradient distribution in unconstrained optimization. *SIAM Journal on Control and Optimization*, 33(6):1916–1925, 1995.
- [14] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer, 2002.
- [15] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optim. Theory Appl.*, 109(3):475–494, 2001.
- [16] Paul Tseng. Dual coordinate ascent methods for non-strictly convex minimization. *Math. Program.*, 59(2):231–247, 1993.
- [17] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, 2002.
- [18] You Xu. Pyipopt: A python interface to ipopt. <http://code.google.com/p/pyipopt/>.