The Parallel Surrogate Constraint Approach to the Linear Feasibility Problem

Hakan Özaktaş, Mustafa Akgül, and Mustafa Ç. Pınar*

Department of Industrial Engineering
Bilkent University
06533 Bilkent, Ankara, Turkey

Abstract. The linear feasibility problem arises in several areas of applied mathematics and medical science, in several forms of image reconstruction problems. The surrogate constraint algorithm of Yang and Murty for the linear feasibility problem is implemented and analyzed. The sequential approach considers projections one at a time. In the parallel approach, several projections are made simultaneously and their convex combination is taken to be used at the next iteration. The sequential method is compared with the parallel method for varied numbers of processors. Two improvement schemes for the parallel method are proposed and tested.
Key Words. Linear and convex feasibility, projection methods, distributed computing, parallel algorithms.
Subject Classifications (AMS). 90C25, 90C26, 90C60.

1 Introduction to Projection Methods for the Convex Feasibility Problem

In this paper some experimental results with the surrogate constraint algorithm described in [Yang & Murty 92] are given. The algorithm has both sequential and parallel versions. Emphasis is put on the parallel implementation.

The problem is to find a feasible point with respect to a set of linear inequalities explicitly defined as $Ax \leq b$, where $x \in R^n$ and A is an m by n matrix. It is assumed that the polyhedron defined by these inequalities is nonempty.

An iteration of the algorithm is carried out by projecting the current point onto a surrogate constraint—defined by a set of violated constraints at the current iteration. The algorithm may be seen as an extended version of the classical methods like Cimmino's algorithm and the relaxation algorithm for linear inequalities.

Before going into the sequential and parallel algorithms, an overview of several projection algorithms to the feasibility problem will be given. Such algorithms are useful when the problem is to determine a feasible point in a nonempty set defined by the intersection of many convex sets.

* The authors are indebted to K. Madsen for providing financial support to this project.

A projection of a point $x^k \in R^n$ onto a convex set Ω gives a point (if there are any) $x \in \Omega$ which has the minimal Euclidean distance to x^k [Hir.-Urr. & Lemar. 93]. However, projections are usually defined more generally as the nearest points contained in the convex bodies with respect to appropriate distance definitions.

The method of successive orthogonal projections of Gubin, Polyak and Raik works by projecting the current point onto a convex set at each iteration until a point which is contained in the intersection of these convex sets is found. At a typical iteration of the SOP algorithm (actually in many of these algorithms) an overrelaxed or an underrelaxed step is taken in the computed projection direction. Hence an iterative step becomes:

$$x^{k+1} = x^k + \lambda_k (P_{C_i}(x^k) - x^k) \qquad 0 < \lambda_k < 2 \tag{1}$$

where P_{C_i} is the projection operator onto the closed convex set C_i and λ_k is the so called relaxation parameter. When $\lambda_k = 1$, the next point generated is the exact orthogonal projection of the current point. On the other hand, when $\lambda_k > 1$, one has longer steps, which is the case of overrelaxation and when $\lambda_k < 1$, one has shorter steps, which is the case of underrelaxation [Censor & Zenios 95].

One has to compute the projection of x^k onto C_i for which $x^k \notin C_i$. This projection requires the solution of the following problem:

$$P_{C_i}(x^k) = \min_{x \in C_i} ||x^k - x|| \tag{2}$$

In this case, the minimization is made over the Euclidean distance, and the nearest point of C_i to x^k is found.

For the cases where $C_i = \{x \in R^n : f_i(x) \leq 0\}$, $f_i(x)$ convex and differentiable, the moving direction $P_{C_i}(x^k) - x^k$ is given by $\nabla f_i(x^{k+1})$. However, to determine the direction and the next point, one needs to solve the minimization problem defined above and in a way, to find the point x^{k+1}, in advance. The cyclic subgradients projection method overcomes this difficulty by picking up an alternate moving direction, $\nabla f_i(x^k)$ (or a suitable subgradient at x^k, for the nondifferentiable case) [Censor & Lent 82], [Censor & Zenios 95].

When an explicitly defined linear feasibility problem is considered, both SOP and CSP methods are equivalent to the relaxation procedure of Agmon, Motzkin and Schoenberg [Censor & Zenios 95]. In that case, the intersection of many halfspaces define the convex set. At an iteration point x^k, the routine proceeds by considering a violated inequality $a^i x^k > b_i$ and calculating the next point as:

$$x^{k+1} = x^k - \lambda_k \frac{a^i x^k - b_i}{||a^i||^2} a^i \tag{3}$$

where $0 < \lambda_k < 2$. Clearly the gradient of $f_i(x) = a^i x - b_i$, is a^i regardless of x, hence there is no essential difference between these routines.

Cimmino's algorithm considers all violated inequalities at a time. Projections are made separately and their convex combination is taken. The routine is quite slow when the number of constraints is large. The interest in the surrogate

constraint algorithm stems from this fact. In the Yang-Murty algorithm instead of making projections for all violated constraints, block projections are made. However, the paper by Yang and Murty reports computational results only with the sequential surrogate constraint method. Our point of departure is to investigate the performance of the parallel version of this algorithm. Implementation of a similar but somewhat different parallel algorithm is described in [Gar.-Pal. & Gon.-Cas. 96] which will also be discussed in Section 3.

2 The Sequential Surrogate Constraint Method

The matrix A and the RHS vector b are split into p blocks so that one has $A^t x \leq b^t$, $t = 1, ..., p$. Each block consists of m_t rows. During the iterations, projections are made not to individual half spaces defined by violated inequalities, but to sets defined by surrogate constraints. The surrogate constraint for each block is $\pi^t A^t x \leq \pi^t b^t$, where π^t is an appropriate weight vector.

The sequential surrogate constraint algorithm of Yang and Murty is given as follows:

Step 0. Generate or read a feasible problem, with $A \in R^{m \times n}$, $b \in R^m$ with previously known values of n, m, p, $m_1, \ldots, m_p$. Initially, let $k = 0$ and $t = 1$. Fix a value of λ so that $0 < \lambda < 2$.

Step 1. Check, if $A^t x^k \leq b^t$. If so, then let $x^{k+1} = x^k$, otherwise let

$$x^{k+1} = x^k - \lambda \frac{(\pi^{t,k} A^t x^k - \pi^{t,k} b^t)(\pi^{t,k} A^t)}{||\pi^{t,k} A^t||^2} \tag{4}$$

where $\pi_i^{t,k} > 0$, if constraint i is violated and $\pi_i^{t,k} = 0$ otherwise, and $\sum_{i=1}^{m_t} \pi_i^{t,k} = 1$ is required for convenience. Update the value of the counter of violated inequalities.

Step 2. If $t < p$, let $k = k + 1$, $t = t + 1$ and go to *Step 1*. If $t = p$ and if the total number of violated constraints in the major iteration is zero then stop, the current solution is feasible. Otherwise, assign zero as the new value of the counter of violated constraints, let $t = 1$, $k = k + 1$ and go to *Step 1*.

Note that one can make the feasibility check also by setting $x_{\text{old}} = x^k$ at the end of each major iteration, i.e. when $t = p$. If at the end of the following major iteration, the current x is same with x_{old}, then this solution is feasible.

This algorithm is finitely convergent, if the feasibility check in *Step 1* is adjusted to allow for a certain degree of tolerance. Hence, one needs to make the comparison between $a^i x^k$ and $b_i + \varepsilon$ where ε is a small positive value [Yang & Murty 92].

During our implementation, A has been generated as a sparse matrix without any structure. The blocks are divided almost evenly, since the matrix doesn't have a special structure. Implementation results of the sequential algorithm is given in Table 1. Details related to problem generation, weight selection rules and matrix storage are given in Section 5.

p // m, n	2 //	4 //	8 //	16 //
500, 1000	15.4(7.2)	27.8(6.2)	51.8(5.6)	104.6(5.6)
2000, 1000	119(59)	203(50)	365.4(44.8)	651.8(39.8)
5000, 2500	106.6(52.8)	191.8(47.4)	367(45)	681.6(43.6)

Table 1. Average number of iterations (numbers in the parentheses represent the major iterations) of an implementation of the sequential algorithm. Matrices are randomly generated with a sparsity value of 0.02. m and n are the row and column sizes respectively and p represents the number of blocks. Five test problems for each size have been solved.

3 The Parallel Surrogate Method

The parallel algorithm of Yang and Murty uses the same structure. The problem is divided into blocks evenly and surrogate constraints are considered for block projections. But in this case, the projections are made simultaneously and a convex combination of these is taken. Thus, a single combined step is taken at a major iteration when compared to p distinct movements in the sequential algorithm.

Hence, the algorithm will be modified as follows:

Step 0. Generate or read a feasible problem. Let $k = 0$ and fix λ so that $0 < \lambda < 2$.

Step 1. For $t = 1, ..., p$,
check, if $A^t x^k \leq b^t$. If so, then let $P_t(x^k) = x^k$, otherwise let

$$P_t(x^k) = x^k - \frac{(\pi^{t,k} A^t x^k - \pi^{t,k} b^t)(\pi^{t,k} A^t)}{||\pi^{t,k} A^t||^2} \tag{5}$$

where $\pi^{t,k}$, is the same as that of the sequential algorithm. When the entire matrix is processed, let $P(x^k) = \sum_{t=1}^{p} \tau_t P_t(x^k)$ where $\sum_{t=1}^{p} \tau_t = 1$, $\tau_t \geq 0$, and $\tau_t > 0$ for all blocks which violate feasibility. The next point is generated as:

$$x^{k+1} = x^k + \lambda(P(x^k) - x^k) \tag{6}$$

Update the total number of violated constraints, in all blocks.

Step 2. If the total number of violated constraints in the major iteration is zero then stop, the current solution is feasible. Otherwise, assign zero to the number of violated constraints, let $k = k + 1$ and go to *Step 1*.

Our experimental results reveal that the parallel algorithm as given in this pure form is much slower than the sequential algorithm. Comparing the results in Table 1 and Table 2 (assuming that an iteration of the sequential routine is more or less equivalent to one major iteration of the parallel routine) indicates

p // $m,\ n$	2 //	4 //	8 //	16 //
500, 1000	27.6	69.4	209.4	507
2000, 1000	267.4	1422.8	3393.8	6921.4
5000, 2500	1087.6	3274.8	6524.2	13069.2

Table 2. Average number of major iterations of an implementation of the parallel algorithm. p represents the number of blocks and hence the processors. Same test problems with the sequential experiments given in Table 1, are used.

that the two algorithms have incomparable performances. However, it is still desirable to benefit from the effects of parallelization. Clearly, one should be able to obtain some better results when it is possible to distribute the feasibility checks of the blocks to distinct machines.

An examination of the two algorithms reveals the problem of the parallel routine and suggests a partial remedy. The sequential routine takes several steps (the number being equal to the number of blocks which have infeasibility) which accumulate, whereas the parallel routine provides a single step (which is a convex combination of the steps generated from infeasible blocks) during a major iteration. The overall effect of this fact is much worse, as seen from the test results.

Let us recall the sequential moving direction (with an appropriate magnitude based on the Euclidean distance to the surrogate hyperplane) for a certain infeasible block t:

$$-d_t = -\frac{(\pi^{t,k}A^t x^k - \pi^{t,k}b^t)(\pi^{t,k}A^t)}{||\pi^{t,k}A^t||^2} \tag{7}$$

The next test point is calculated as, $x^{k+1} = x^k - \lambda d_t$.

The parallel algorithm also uses the direction given in (7) for the corresponding block. However, instead of accumulating these steps, a convex combination of these, hence a shorter step is taken. The movement in the parallel routine, given in (5) and(6) can be rewritten as (assuming that $d_t^k = 0$ when block t is feasible):

$$x^{k+1} = x^k + \lambda\left(\sum_{t=1}^{p} \tau_t P_t(x^k) - x^k\right), \qquad \left(\sum_{t=1}^{p} \tau_t = 1\right)$$

$$= x^k + \lambda\left(\sum_{t=1}^{p} \tau_t (x^k - d_t^k) - x^k\right)$$

$$= x^k - \lambda\left(\sum_{t=1}^{p} \tau_t d_t^k\right)$$

$$x^{k+1} = x^k - \lambda \bar{d}^k, \qquad \left(\bar{d}^k = \sum_{t=1}^{p} \tau_t d_t^k \right) \tag{8}$$

One can suggest the usage of longer steps having magnitudes comparable to those obtained in the sequential algorithm. An idea is to implement the algorithm by increasing the step size by multiplying it with p at each major iteration.

Using this idea somewhat improves the performance during implementation. However, it will yield unnecessarily long steps which slows down the algorithm, in the cases where some of the blocks reach feasibility immediately and some do not. Therefore, choosing the number of violated blocks at a given iteration as the multiplicative parameter will be a better strategy to approach the trajectory obtained from the sequential algorithm. Thus, one can obtain the next point alternatively as:

$$x^{k+1} = x^k - (\lambda \bar{p}^k) \bar{d}^k, \tag{9}$$

where $\bar{p}^k$ is the number of blocks which have infeasibility at the k^{th} iteration.

The results obtained with this adjusted step sizing rule is given in Table 3. It can be seen that, for relatively small values of p, the results are better than the pure application of the parallel algorithm. Furthermore, some of these results are better than those obtained by the sequential algorithm, assuming that one major iteration of a parallel routine is more or less equivalent to a normal iteration of the sequential routine. However, utilizing purely (9) might yield us

p // $m,\ n$	2 //	4 //	8 //	16 //
500, 1000	7.2	7	6.6	7
2000, 1000	84	152.2	–	–
5000, 2500	86.6	979.6	–	1164.6

Table 3. Average number of major iterations of an implementation of the parallel algorithm with new step sizing policy. Same test problems with the previous experiments given in Table 1, and Table 2 are used. '–' represents a typical case where the routine does not converge to a feasible point.

a nonconvergent routine, especially when p is relatively large. We were not able to obtain the complete test results for $p = 8$ and $p = 16$. The reason is that, the generated steps might still become too long, in some cases. Thus a regulatory mechanism of the step size is required to use (9) successfully.

An improved step for a similar parallel projection algorithm outlined in [Gar.-Pal 93] has been given in [Gar.-Pal. & Gon.-Cas. 96] recently. This paper discusses the slow performance of the parallel Cimmino algorithm (without any adjustment) and proposes an accelaration procedure. The partitioning mechanism and projective subroutines in the resulting algorithm are different than

that of the Yang-Murty algorithm, however the overall algorithm is similar. The pure parallel algorithm uses the following combined step:

$$x^k - \lambda \left(\sum_{t=1}^{p} \tau_t d_t^k \right) \tag{10}$$

which is identical with (8). The improved step yields the following test point:

$$x^{k+1} = x^k - \frac{1}{2} \lambda_k \frac{\sum_{i=1}^{p} ||d_t^k||^2}{||\sum_{t=1}^{p} d_t^k||^2} \sum_{t=1}^{p} d_t^k \tag{11}$$

It has been established in [Gar.-Pal. & Gon.-Cas. 96] that the algorithm with adjusted step sizing policy performs better under some assumptions, both theoretically and practically.

We have used the García Palomares-Gonzáles Castaño step given by (11) in the parallel surrogate algorithm of Yang-Murty, for the same test problems. The results are given in Table 4. It can be seen that this step improves the parallel version of the Yang-Murty algorithm, however it seems that using the step given in (9) performs better when p is relatively small.

Another interesting observation is that for a given problem size the number of iterations of the parallel algorithm is almost constant as the number of blocks increases. This is in contrast to the results of Table 2 where the number of major iterations increases directly for a given problem as more blocks are used. Based on this limited evidence we can conclude that the new step provides a significant stabilizing effect on the parallel surrogate constraint algorithm.

p // $m,\ n$	2 //	4 //	8 //	16 //
500, 1000	24.2	24.2	26.6	27.8
2000, 1000	193	200.2	193.4	197.8
5000, 2500	612.8	643	646.6	669

Table 4. Average number of major iterations of an implementation of the parallel algorithm with the García Palomares-Gonzáles Castaño step. Same test problems with the previous experiments given in Table 1, Table 2 and Table 3 are used.

4 Practical Considerations for the Parallel Algorithm

In the parallel routine the number of submatrix blocks should be equal to the number of processors. Each processor deals exactly with a single block, therefore it is quite natural to divide the matrix into p submatrices evenly or almost

evenly, so that each submatrix has $\lceil \frac{m}{p} \rceil$ rows. Each processor checks its constraints (which are equal or almost equal in size) for feasibility and computes the projection if necessary. When a processor finishes its task, it waits for the others to finish as well.

During the subroutine operations (feasibility checks and projection calculations for distinct blocks) each processor works independently and no message passing within the machines is required. When all processes are finished the projection data is transferred to one of the processors, where the new point is calculated according to (6). The calculation of the cumulative projection and the new point, is the sequential part of the algorithm. After this step, the new point is broadcasted to all processors and the procedure is repeated until a feasible point is found.

The initialization phase is carried out on each processor independently to avoid further communication within the blocks. If the problem is to be randomly generated, the initial seed is broadcasted to all machines. If data has to be read from the files, this is done by all machines.

A distributed implementation of the algorithm is being developed using PVM 3.3.11 on several Sparc workstations. The algorithm is being governed by a main C routine, which makes calls to a C subroutine (for parallel block operations) and to several PVM functions suitable to C programs (see [Geist et al. 94]). These results will be reported elsewhere.

5 Implementation Issues

It is assumed that the matrix underlying the feasibility problem is sparse. This matrix is stored both in the rowwise format and the columnwise format, to ease the access. When computing $A^t x^k$ or $a^i x^k$, the rowwise format is used and when computing $\pi^{t,k} A^t$ the columnwise format is used. The widely known standard formats are used almost without any changes [Pissa. 84], but to ease the control over the storage arrays, we have added one more cell to each, which marks the end of the array.

Sample test problems have been generated as follows: First of all, a matrix with a given size and sparsity percentage is generated so that the nonzero elements are distributed uniformly, so that each nonzero value is distributed in between -5.0 and 5.0. The random distribution has been done in two ways. In the first, an exact number of nonzero entries is fixed and they are distributed uniformly (with parameters depending on the problem size) to rows of the matrix so that each row has at least one nonzero element. Then, their column numbers are generated uniformly. In the second, a simulated sequence of a Poisson process is obtained and points are generated with exponential interarrival times with parameter equal to the sparsity of the matrix. The exponential density is truncated between 1 and n and the generated interarrival time is rounded to the nearest integer and the next nonzero entry position is found by adding this value to the column number of the previously placed nonzero entry. When one row is finished, the process is continued in the next row. This is somewhat better than

the first, since the nonzero entries are generated sequentially and one does not require a reordering when storing the data in rowwise format.Here we utilize the property stating that in a Poisson process, given that n arrivals have occurred within a time interval $(0,t)$, the distribution of the arrival times $S_1,\ldots,S_n$ have the same distribution as the order statistics of n independent random variables distributed on this interval $(0,t)$ (see for example [Ross 83]). We assume that the idea is applicable to a discrete interval since it is quite a long interval, and that the truncation of the exponential distribution does not have a significant effect on the uniformity of the nonzeros over the matrix.

After generating the random matrix, a random x is generated, so that each of its elements are in the interval $(-4.5,4.5)$. Following this, Ax is computed and the b vector is generated so that $b_i = A_i x + u_i$ where u_i is a discrete uniform random variable between 0 and 1. In this way a feasible polyhedron is created. Our aim is to try to keep this polyhedron somewhat small and distant to the initial point of the algorithm, so that trivial convergence in a few steps will not occur.

An important issue is the selection of the weight vector $\pi^{t,k}$. Weights may be distributed equally among all violated constraints or they can be assigned, proportional to the amount of violations. Or a suitable combination of the two approaches may also be used. In our tabulated results we have used the hybrid approach (which has also been used in [Yang & Murty 92]):

$$\pi_i^{t,k} = \frac{0.2(A_i^t x^k - b_i)}{\sum_{h:A_h^t x^k > b_h^t}(A_h^t x^k - b_h^t)} + \frac{0.8}{\text{number of violated constraints}} \qquad (12)$$

For convenience it is assumed that $\sum \pi_i^{t,k} = 1$. The relaxation parameter λ has been taken to be 1.7 and the tolerance limit for feasibility, ε is 10^{-9}.

6 Conclusions and Future Work

In our test problems it has been observed that the parallel surrogate constraint method without any adjustment is quite poor when compared to the sequential surrogate constraint method. The reason is that the magnitudes of the steps obtained in the parallel algorithm remain quite small with respect to the accumulated sequential steps.

In order to compensate for this loss, we have magnified the parallel step length by multiplying it with the number of infeasible blocks at that time. In this way we have obtained some favorable results. We have also used a longer step which has been recently used in a similar (but not the same) parallel routine by [Gar.-Pal. & Gon.-Cas. 96]. The results are also improved when compared to the pure version of the parallel algorithm. This approach also seems to stabilize the number of iterations when more blocks are used to partition the problem.

It seems that increasing the step simply by multiplying it with the current number of infeasible blocks, yields quite an improvement when the number of blocks is relatively small. However, this idea should be used with some sort of

regulation over the step given by (9), since the idea gives unnecessarily long steps which may prevent convergence, especially when the number of processors is relatively large.

The issue that is of interest at this point is to obtain an adjusted parallel algorithm and establish convergence. This can be possibly done by using the improved step and adapting a control mechanism. We are also foreseeing the completion of the PVM implementation of the parallel algorithm.

References

[Censor & Lent 82] Censor, Y., Lent, A.: Cyclic Subgradient Projections. *Mathematical Programming*, 24:233–235, 1982.

[Censor & Zenios 95] Censor, Y., Zenios, S. A.: *Parallel Optimization: Theory, Algorithms and Applications* (to be published by Oxford University Press). October 18, 1995.

[Gar.-Pal. 93] García Palomares, U. M.: Parallel Projected Aggregation Methods for Solving the Convex Feasibility Problem. *SIAM Journal on Optimization*, 3–4:882–900, November 1993.

[Gar.-Pal. & Gon.-Cas. 96] García Palomares, U. M., González Castaño, F. J.: Acceleration technique for solving convex (linear) systems via projection methods. Technical Report OP960614, Universidade de Vigo, ESCOLA TÉCNICA SUPERIOR DE ENXEÑEIROS DE TELECOMUNICACIÓN, Lagoas Marcosende 36200 Vigo, España, 1996.

[Geist et al. 94] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press., Cambridge, Massachusetts, 1994.

[Hir.-Urr. & Lemar. 93] Hiriart-Urruty, Jean-Baptiste and Lemaréchal, Claude: *Convex Analysis and Minimization Algorithms.* Springer-Verlag, Berlin, 1993.

[Pissa. 84] Pissanetzky, Sergio: *Sparse Matrix Technology.* Academic Press Inc., London, 1984.

[Ross 83] Ross, Sheldon M.: *Stochastic Processes.* John Wiley & Sons Inc., 1983.

[Yang & Murty 92] Yang, K., Murty, K.G.: New Iterative Methods for Linear Inequalities. *Journal of Optimization Theory and Applications*, 72:163–185, January 1992.