A NEW APPROACH TO OPTIMIZATION OF CHEMICAL PROCESSES

by

T.J. Berna, M.H. Locke & A.W. Westerberg

DRC-06-3-79

January 1979

Department of Chemical Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213

This work was supported by NSF Grant ENG-76-80149.

ABSTRACT

Based on recent work reported by Powell, a new optimization algorithm is presented which merges the Newton-Raphson method and quadratic programming. A unique feature is that one does not converge the equality and tight inequality constraints for each step taken by the optimization algorithm. Hie paper shows how to perform the necessary calculations efficiently for very large problems which require the use of mass memory. Experience with the algorithm on small problems indicates it converges exceptionally quickly to the optimal answer, often in as few iterations (5 to 15) as are needed to perform a single simulation with no optimization using more conventional approaches.

SCOPE

In optimizing the model of a chemical process one is faced with solving a nonlinear programming problem containing anywhere from several hundred to several thousand nonlinear equality and inequality constraints. Before process optimizers were developed, a designer who wished to optimize a chemical process would usually adjust one or more of the independent variables and use the computer to converge the equality constraints (heat and material balance equations) and to evaluate the objective function. Based on the results of this simulation, the designer would check many of the inequality constraints by hand and then would readjust the decision variables and perform another simulation to get a better and/or feasible result.

Some earlier attempts to design a chemical process optimizer (e.g. Friedman and Pinder, 1972) mimicked this process by replacing the designer with an pattern search optimization routine. Although approaches such as this are reasonably effective at improving the process, they are inefficient and have difficulty handling the inequality constraints.

Recently Powell (1977a) published an algorithm which drastically reduces the computational effort required to solve nonlinear programs. The unique feature of Powell's technique is that.he does not have to converge the equality constraints or tight inequality constraints at each iteration. Powell's technique is not suitable as stated for large problems because it requires that the user solve a very large quadratic programming problem (QPP) involving a Hessian matrix of the size of the number of problem variables at .each iteration. Although this method converges rapidly, it requires too much core storage. We extend Powell's work by developing a decomposition scheme which permits one

(1) to solve the same problem but reduce drastically the storage requirements, and

(2) to take computational advantage of the fact the optimization is of a system of interconnected process units.

This paper opens with a brief description of the process optimization problem and some comments on the more significant algorithms already available. We then discuss Powell's algorithm, and, starting with his formulation of the problem, we perform the algebra necessary to arrive at a decomposed problem. We follow this development with a formal statement of the resulting algorithm and an example problem.

In this paper we rely heavily on an earlier paper by two of the authors (Westerberg and Berna (1978)) which describes a decomposition technique for solving large sets of structured linearized equations arising from modeling chemical processes. We do not attempt to present any convergence proofs here, because Powell's results are directly applicable. The reader is referred to Powell (1977b) and Han (1975).

CONCLUSIONS AND SIGNIFICANCE

There are two major difficulties associated with optimizing a modern chemical process model: excessive storage requirements and excessive computational requirements. The technique we present in this paper addresses both of these problems directly. Our method is an extension of work recently published by Powell (1977a). Powell's algorithm is based on the Newton-Raphson method, and it generates a quadratic program at each iteration to improve the current guess at the solution to the original nonlinear program. The primary advantage of Powell's scheme is that it does not need to find a feasible solution to the equality constraints (or tight inequality constraints) at each iteration. The paper demonstrates with an example that this fact dramatically reduces the computational work involved in converging to the optimal solution. Unfortunately, Powell's method as stated becomes impractical for large problems because it requires solving a quadratic program in all the problem variables and not just in the decision variables. We show that the modular nature of chemical processes has allowed us to develop an algorithm which uses mass memory efficiently for very large problems and which solves a quadratic program at each iteration _in the decision variables only_. Therefore, we are applying Powell's algorithm in a way that never requires us to use more than a modest amount of core. Based on a small number of test problems this algorithm appears to require about the same number of gradient and function evaluations to arrive at an optimal solution as available nonoptimizing simulation packages require to obtain a single solution.

The chemical process optimization problem can be stated as follows

Min $(z)

Subject to $g(z) = 0$

h(z) £ 0 (Pi)

zeR^{n+r}

$g:R^{n+r} \rightarrow R^{n}$

$h:R^{n+r} \rightarrow R^{m}$

where the constraints represent material and energy balances, equilibrium relationships, economic factors, etc. For many chemical processes of practical importance n and m have values anywhere from 1000 to 50,000, and r might range anywhere up to about 50. Obviously, these problems are very large.

One technique which has been quite successful for solving highly constrained nonlinear programming problems is the Generalized Reduced Gradient (GRG) algorithm (Abadie and Carpentier, 1969). This technique uses the equality constraints (and tight inequality constraints) to eliminate computationally a large number of variables and equality constraints from the total set. This procedure reduces the apparent dimensionality of the optimization problem. At each iteration all of the eliminated constraints must be satisfied, thus at each iteration the algorithm must solve a very large set of nonlinear algebraic equations.

Some investigators, notably in the oil industry, have been successful in converting problem (Pi) into a large linear programming problem (LP) by linearizing the constraints and the objective function. Because large linear programs are relatively easy to solve the LP-based algorithm solves a sequence of LP's which eventually converge to the desired optimum.

This technique works well for some problems, but it has a drawback. There is no information in the algorithm about the curvature of the constraints, therefore convergence cannot be second order near the solution. Powell (1978) illustrates this difficulty with a small example.

Another class of optimization algorithms is called exact penalty function methods. They use an extended Lagrangian function, one which has a penalty term added to it as well as the constraints with multipliers. Charalambons (1978) describes these methods and claims they are very effective, but the extension of these ideas to very large problems does not yet appear to be available.

From a computational standpoint, we feel the most successful algorithm available may be that recently developed by Powell (1977a). The total number of gradient and function evaluations required by this algorithm to obtain an optimal solution corresponds to the number required by many simulation packages to obtain a single feasible solution. Table 1 illustrates how Powell's algorithm compares with the best known algorithms for solving some small classical problems studied by Colville and others (Powell, 1977a).

Table 1. Comparison of Algorithms (Table 1 of Powell, 1977a)

Problem	Colville (1968)	Biggs (1972)	Fletcher (1975)	Powell
Colville 1	13	8	39 (4)	6 (4)
Colville 2	112	47	149 (3)	17 (16)
Colville 3	23	10	64 (5)	3 (2)
Post Office Problem	—	11	30 (4)	7 (5)
Powell	—	—	37 (5)	7 (6)

The numbers represent the number of times the functions and their gradients were computed; the numbers in parentheses are the number of iterations.

<u>Problem Decomposition</u>

For convenience we restate the optimization problem, but this time we partition the variables into two sets: $x \in R^n$ and $u \in R^r$.

$$
\begin{array}{lll}
 & \min\limits_{x,u} & \Phi(x,u) \\
\text{Subject to} & & g(x,u) = 0 \\
 & & h(x,u) \geq 0 \\
 & & \Phi : R^{n+r} \rightarrow R \\
 & & g : R^{n+r} \rightarrow R^n \\
 & & h : R^{n+r} \rightarrow R^m
\end{array}
\qquad \text{(P2)}
$$

We linearize the equality and inequality constraints about a current guess at the solution, obtaining

$$
\begin{aligned}
\frac{\partial g}{\partial x}\Delta x + \frac{\partial g}{\partial u}\Delta u &= -g \\
\frac{\partial h}{\partial x}\Delta x + \frac{\partial h}{\partial u}\Delta u &\geq -h
\end{aligned}
\qquad (1)
$$

We approximate the objective function $\Phi(x,u)$ to second order in all the problem variables

$$
\begin{aligned}
\Phi(x + \Delta x,\ u + \Delta u) = \Phi(x,u) &+ \left[\frac{\partial \Phi}{\partial x}\ \frac{\partial \Phi}{\partial u}\right]\begin{bmatrix}\Delta x \\ \Delta u\end{bmatrix} \\
&+ \frac{1}{2}\left[\Delta x^T \Delta u^T\right] C \begin{bmatrix}\Delta x \\ \Delta u\end{bmatrix}
\end{aligned}
\qquad (2)
$$

where, as shown by Powell (1977a), C should be the Hessian matrix of the Lagrange function

$$
L(x,u,\lambda,\mu) = \Phi(x,u) - \lambda^T g - \mu^T h \quad .
$$

X. and \s, are vectors of Lagrange and Kuhn-Tucker multiplers, respectively. The approximate problem

$$\text{Min} \quad \hat{\$}(x + Ax,\ u + Au)$$

subject to constraints (1) is a quadratic programming problem in variables Ax, Au.

The necessary conditions for solving this approximate problem are as follows,

1) Stationarity of the Lagrange Function with Respect to Ax and Au

$$C \begin{bmatrix} Ax \\ \wedge\ Au \end{bmatrix} - \begin{pmatrix} \frac{\partial g^{r}}{6x} \\ \frac{r}{bu} \end{pmatrix} \lambda - \begin{pmatrix} \frac{ML^{T}}{bx} \\ \frac{\partial h^{T}}{bu} \end{pmatrix} \mu = - \begin{pmatrix} \frac{\partial \Phi}{bx} \\ \frac{\partial \Phi}{bu} \end{pmatrix}$$

2) Original Constraints

$$\frac{\partial g}{\partial x^T} \Delta x + \frac{\partial g}{\partial u^T} Au = - g$$

$$\frac{\partial h}{\partial x^T} \Delta x + \frac{\partial h}{\partial u^T} \Delta u \geq - h$$

3) Kuhn-Tucker Conditions

$$\mu^T \left[\frac{}{bx^T} \Delta x + \frac{\partial h}{bu^T} \Delta u + h \right] = 0$$

$$\mu \geq 0$$

It is these conditions that a QPP algorithm satisfies. At this point it is convenient to introduce the notation listed in Table 2.

Table 2. Nomenclature for Partial Derivatives

$\phi_x = \frac{\partial \phi}{\partial x}$	$\phi_u = \frac{\partial \phi}{\partial u}$
$\frac{\partial g}{\partial x}$	$\frac{\partial g}{\partial u}$
$K_x = \frac{\partial\, [illegible]^T}{\partial x}$	$K_u = \frac{\partial\, [illegible]^T}{\partial u}$
$C_{xx} = \frac{\partial^2 C}{\partial x \partial x^T}$	$C_{uu} = \frac{\partial^2 C}{\partial u \partial u^T}$
$C_{xu} = \frac{\partial^2 C}{\partial x \partial u^T}$	$C_{ux} = \frac{\partial^2 C}{\partial u \partial x^T}$

Note: We assume that C is symmetric (i.e. $C_{ij} = C_{ji}$)

With this new notation the stationary condition (2) becomes

$$\begin{bmatrix} C_{xx} & C_{xu} & J_x^T & K_x^T \\ C_{ux} & C_{uu} & J_u^T & K_u \\ J_x & J_u & 0 & 0 \\ & K_u & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \\ -\lambda \\ -\mu \end{bmatrix} \begin{matrix} = -b_x \\ = -b_u \\ = -g \\ \geq -h \end{matrix} \qquad (3)$$

$$\mu^T [K_x \quad K_u \quad I] \begin{bmatrix} \Delta x \\ \Delta u \\ h \end{bmatrix} = 0$$

$$\mu \geq 0$$

Note that the coefficient matrix is symmetric and that the lower right portion of the matrix is zero.

Rearranging the matrix in (3) we get

$$\begin{bmatrix} J_x^T & C_{xx} & C_{xu} & K_x^T \\ 0 & J_x & J_u & 0 \\ J_u^T & C_{ux} & C_{uu} & K_u^T \\ 0 & K_x & K_u & 0 \end{bmatrix} \begin{bmatrix} -\lambda \\ \Delta x \\ \Delta u \\ -\mu \end{bmatrix} \begin{matrix} = -b_x \\ = -b_u \\ = -g \\ \geq -h \end{matrix} \qquad (4)$$

$$\mu^T [K_x \quad K_u \quad I] \begin{bmatrix} \Delta x \\ \Delta u \\ h \end{bmatrix} = 0$$

$$\mu \geq 0$$

The coefficient matrix in (4) is very large (2n + r + m) x (2n + r + m) (n and m may be several thousand each; r may be 1 to 50). This matrix is reasonably sparse, and each of the blocks has a structure which can be used to simplify the computational requirements for solving (4).

A very efficient method for solving large sparse linear systems of equations of the form $Mx = b$ is to factor the matrix M into the product of a sparse lower triangular matrix, L, and a sparse upper triangular matrix, U. This technique, known as L/U factorization, can be applied to any matrix which has an inverse, and it is far more efficient to solve linear systems this way, even if they are not sparse, than to compute M^{-1} and use it to premultiply b. Determining L and U for a given matrix M is equivalent to performing a Gaussian elimination on M (see Reid, 1971). Once L and U have been determined,solving is carried out in two steps. The first step is to solve

$$L\,y = b$$

for y by forward substitution, since, as stated, L is lower triangular. The second step is to solve

$$U\,x = y$$

by backward substitution (U is upper triangular).

We now perform symbolically a Gaussian elimination on (4) to eliminate the first two block rows. We use the term block row to refer to all the rows associated with a single symbol, thus our coefficient matrix has four block rows. Each of these first two block rows represents n equations where n is very large. After this reduction we shall discover that the remaining subproblem is a very much smaller QPP, with a Hessian matrix of size rxr rather than the size (n+r) x (n+r) of our original problem. The reduced QPP will have the structure

$$\begin{bmatrix} H & Q \\ Q^T & 0 \end{bmatrix} \begin{bmatrix} u \\ \mu \end{bmatrix} \begin{matrix} = -g \\ \geq -\hat{h} \end{matrix}$$

$$\text{St,} \quad H^T(Q^T Au + \hat{h}) = 0$$

$$\mu \geq 0$$

which is the same symmetric structure as our original problem.

We shall show, after the symbolic reduction step, how to perform the matrix manipulations resulting, discovering thereby how to solve our original QPP with considerably less computational effort than one might expect. In particular we can show that no full matrix of size nxn or mxm need ever be dealt with, where n and m are very large numbers.

<u>Symbolic Reduction of the First Two Block Rows</u>

<u>Step 1</u> Perform block column 1 reductions on the coefficient matrix in (4) in two steps:

(a) Premultiply block row 1 by $(J_x^T)^{-1} = J_x^{-T}$

(b) Make the (3,1) block element zero by the block row operation

New Block Row 3 = Old Block Row 3

$- J_u^T$ x New Block Row 1

The result of this reduction on the augmented matrix in (4) is:

$$\left[\begin{array}{cccc|c} I & J_x^{-T}C_{xx} & J_x^{-T}C_{xu} & J_x^{-T}K_x^T & -J_x^{-T}b_x \\ 0 & J_x & J_u & 0 & -g \\ 0 & C_{ux} - J_u^T J_x^{-T} C_{xx} & C_{uu} - J_u^T J_x^{-T} C_{xu} & K_u^T - J_u^T J_x^{-T} K_x^T & -b_u + J_u^T J_x^{-T} b_x \\ 0 & K_x & K_u & 0 & -h \end{array}\right]$$

<u>Step 2</u> Perform block column 2 reductions on the above augmented matrix, in 3 steps:

(a) premultiply block row 2 by J_x^{-1}

(b) Make the (3,2) block element zero by the block row operation

New Block Row 3 = Old Block Row 3 -

$(C_{uu} - J_u^T J_x^{-T} C_{xx})$ x New Block Row 2

(c) Make the (4,2) block element zero by the block row operation

New Block Row 4 = Old Block Row 4 -

K_x x New Block Row 2

The result of this reduction on the above matrix is

$$\left[\begin{array}{cc|cc|c} I & J_x^{-T}C_{xx} & J_x^{-T}C_{xu} & J_x^{-T}K_x^T & -J_x^{-T}b_x \\ 0 & I & J_x^{-1}J_u & 0 & -J_x^{-1}g \\ \hline 0 & 0 & H & Q & -q \\ 0 & 0 & Q^T & 0 & -\hat{h} \end{array}\right] \qquad (5)$$

where

$$H = (C_{uu} - J_u^T J_x^{-T} C_{xu}) - (C_{ux} - J_u^T J_x^{-T} C_{xx}) J_x^{-1} J_u$$

$$Q^T = K_u - K_x J_x^{-1} J_u$$

$$q = b_u - J_u^T J_x^{-T} b_x - (C_{ux} - J_u^T J_x^{-T} C_{xx}) J_x^{-1} g$$

$$\hat{h} = h + K_x J_x^{-1} g$$

Clearly the (rxr) matrix H is symmetric. The original Kuhn-Tucker conditions become

$$H^T(Q^T Au + \hat{h}) = 0 \qquad (6)$$

$$\backslash M * 0$$

The last two block rows of the reduced augmented matrix (5) together with these reduced Kuhn-Tucker conditions represent a reduced QPP which has a Hessian matrix H which is rxr. The reduced QPP is

$$\text{Min } q^T Au + \frac{1}{2} Au^T H Au \qquad (7)$$

$$\text{Subject to } Q^T Au \wedge \hat{h}$$

The corresponding Lagrange function is

$$L(\Delta u, n) = q^T Au + | Au^T H Au - p,^T (Q^T Au + \hat{h})$$

The necessary conditions for optimality reproduce precisely these last two block rows and conditions (6).

Computational Algorithm for Solving the QPP

We shall now develop a step by step procedure to solve the original QPP, using the above problem "decomposition.[11] We shall note in particular that nowhere do we need to create and use a full nxn matrix.

Before proceeding, however, we need make one further observation about Powell's algorithm. In Powell's algorithm or in minor variations thereof, the matrix C which approximates the Hessian matrix is formed by starting with the identity matrix and performing a quasi-Newton rank 1 update or rank 2 update (in the form of two rank 1 updates) after each iteration.

The Hessian matrix is not necessarily positive definite, but it is symmetric. Powell (1977a) uses rank 2 updates and forces C to be both positive definite and symmetric after each update, but he points out that the positive definite requirement may be relaxed. Symmetry only can be maintained with rank 1 updates. Instead of storing C in matrix form, it is far more efficient to store only its rank 1 updates. We have after ℓ updates

$$\begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ 0 & I_r \end{bmatrix} + \sum_{j=1}^{\ell} \begin{bmatrix} w_{xj} \\ w_{uj} \end{bmatrix} \begin{bmatrix} w_{xj}^T & w_{uj}^T \end{bmatrix}$$

where I_n and I_r are identity matrices of size nxn and rxr, respectively, and the vector $[w_{xj}^T, w_{uj}^T]^T$ is the j^{th} update vector for C.

We now present the QPP algorithm.

1. Develop the L/U factors for J_x such that $J_x = LU$. J_x represents the Jacobian matrix for the equations modeling a chemical process flowsheet. As described in an earlier paper by two of the authors (Westerberg and Berna, 1978), this matrix has a bordered block diagonal form corresponding to the structure of the flowsheet. Each block corresponds to the equations for a single unit. Advantage can be taken of this structure to develop the factors L and u for J_x. J_x is nxn, say several thousand by several thousand, in size. The techniques in the earlier paper use block reduction methods, where each block fits conveniently into the core store of a computer, and, within each block, use existing sparse matrix codes.

2, We note, in looking through the operations indicated in (5), the repeated occurrence of the terms $A = J_x^{-1} J_u$ and its transpose and $v = J_x^{-1} g$. Form these by solving

$$J_x[A,v] = LU[A,v] = [J_u,g]$$

This step is the performing of a forward and backward substitution using L and U on each of the r+1 columns in the right-hand side. Since L and U are both sparse because J_x is and sparse matrix methods were used to find L and U, this step is not an excessive one.

3. Table 3 lists a number of terms which are needed in what follows and which use the rank 1 update vectors of C.

Compute a_{x_j} and s_{v_j} for $j = 1,\cdots,\ell$

Note that a_{x_j} is a vector of length r and s_{v_j} is a scalar.

The number of multiplications required (multiplications count) for this step is $\ell(rn+n)$.

4. We use the definition of H in (5) and of s_{v_j} in Table 3 to discover H has the form

$$H = (I_r + \Sigma w_{u_j} w_{u_j}^T) - A^T(\Sigma w_{x_j} w_{u_j}^T)$$

$$- (\Sigma w_{u_j} w_{x_j}^T)A + A^T(I_n + \Sigma w_{x_j} w_{x_j}^T)A$$

$$= I_r + A^T A + \sum_{j=1}^{\ell} (w_{u_j} - a_{x_j})(w_{u_j} - a_{x_j})^T$$

If one takes advantage of symmetry, this step can be executed in $(\ell+1)\frac{r(r+1)}{2}$ multiplies.

Table 3. Notation and Operations Count for Terms Involving Hessian Matrix C

Symbol	Expression	Multiplications Count
a_{x_j} (r-vector)	$A^T w_{x_j}$	r n
s_{v_j} (scalar)	$w_{x_j}^T v$	n
s_{u_j} (scalar)	$w_{u_j}^T \Delta u$	r
s_{x_j} (scalar)	$w_{x_j}^T \Delta x$	n

5. Compute $q = b_u + A^T b_x + \sum_{j=1}^{l} [w_{u.j} + a_{x.j}] s_j$

This step requires r(n-hJt) multiplications; q is a vector rxl.

6. Compute $Q^T = K_u + K_x A$

The matrix K_x is very sparse, often with only 1 to 5 nonzeros per row.

Its bordered block diagonal structure permits efficient use of mass memory and requires generally fewer than 5 mr multiplications.

7. Solve the reduced QPP in (7) to get Au and \s,

The matrices Q and H are full after all the steps taken and the QPP algorithm need not, indeed should not, attempt to take advantage of sparsity in these matrices.

8. Compute Ax = v + A Au

Although A is full, this operation requires only nr multiplications.

9. Compute s_{u_j} and s_{x_j} for j = 1,###,X

The expressions for computing these scalars are given in Table 3. This step requires X(r+n) multiplications. Note that by storing only the updates for C one may decide to use only the latest 10 to 20 updates if the algorithm seems to be converging slowly. It is easy to discard old information this way, so *I* should never be excessive.

10. Solve the following expression for λ

$$U^T L^T \lambda = \Delta x + \sum_{j=1}^{\ell} w_{x_j} (s_{x_j} + s_{u_j}) - K_x^T \mu + b_x$$

Note that, at most, r Kuhn-Tucker multipliers can be nonzero, therefore evaluation of the right-hand side requires approximately $n\ell + 5r$ multiplications. The forward and backward substitutions can be performed block-by-block as described earlier.

At this point we have accomplished what we set out to do, namely, solve the original QPP in a modest amount of core. In the next section we state the entire optimization algorithm.

The Optimization Algorithm (See Powell (1977a))

Step 0: Initialization

i) Set $k = 0$, $w_x = 0$, $w_u = 0$, $\upsilon_i = 0$ for $i = 1,\cdots,n+m$

(with $C = I$, the first direction taken is the steepest descent)

ii) Initialize all variables $z = [x_o, u_o]$

Step 1: Compute necessary functions and derivatives

i) $k = k + 1$

ii) Evaluate b_x, b_u, J_x, J_u, K_x, K_u, g, h, L and U

iii) Solve $J_x v = - g$ and $J_x A = - J_u$ for v and A, respectively. (This corresponds to steps 1 and 2 of the decomposition algorithm described in the preceding section.)

iv) If $k < 2$ go to Step 3.

Step 2; Update C

i) Compute $\left(\frac{\partial L}{\partial z}\right)_{z=\hat{z}} = \begin{bmatrix} b_x \\ b_u \end{bmatrix} - \begin{pmatrix} J_x^T \\ J_u^T \end{pmatrix} \lambda \begin{pmatrix} K_x^T \\ K_u^T \end{pmatrix} \mu$

ii) $\gamma = \left(\frac{\partial L}{\partial z}\right)_{z=\hat{z}} - \left(\frac{\partial L}{\partial z}\right)_{z=\hat{z}}$; $\sigma_j = (w_x^T \; w_u^T)_j \, \delta \qquad j = 1, \cdots, \ell$

iii) $\eta = \gamma + (1 - \theta) \sum_{j=1}^{\ell} \begin{pmatrix} w_x \\ w_u \end{pmatrix}_j \sigma_j$

where $\theta = \begin{cases} 1 & \text{if } \delta^T\gamma \geq .2\,\sigma \\ \dfrac{.8\sigma}{\sigma - \delta^T\gamma} & \text{otherwise} \end{cases}$

and $\sigma = \sum_{j=1}^{\ell} \sigma_j^2$

iv) $\begin{bmatrix} w_x \\ w_u \end{bmatrix}_{\ell+1} = - \frac{\sum_{j=1}^{\ell} \begin{pmatrix} w_x \\ w_u \end{pmatrix}_j \sigma_j}{\sqrt{\sigma}} \quad \bullet \quad \begin{pmatrix} w_x \\ w_u \end{pmatrix}_{\ell+2} = \frac{\eta}{(\delta^T\gamma)^{1/2}}$

v) $\ell = \ell + 2$

Step 3: Solve QPP

i) Perform Steps 3 through 10 of the decomposition algorithm described in the previous section.

ii) Let $d_u = \Delta u$ and $d_x = \Delta x$

iii) Compute $\left(\frac{\partial \mathcal{L}}{\partial z}\right)^T_{z=\hat{z}} = \begin{bmatrix} b_x \\ b_u \end{bmatrix} - \begin{pmatrix} J_x^T \\ J_u^T \end{pmatrix} \lambda - \begin{pmatrix} K_x^T \\ K_u^T \end{pmatrix} \mu$

Step 4: Determine step size parameter, α

i) For $i = 1, \cdots, n$ set $u_i = \max\{|\lambda_i|, \frac{1}{2}(u_i + |\lambda_i|)\}$

ii) For $i = 1, \cdots, m$ set $u_{i+n} = \max\{\mu_i, \frac{1}{2}(u_{i+n} + \mu_i)\}$

iii) Select the largest value of $\alpha \in (0,1)$ for which

$$\phi(\hat{x},\hat{u},u) < \phi(x,u,u)$$

where $\phi(x,u,u) = f(x,u) + \sum_{i=1}^{n} u_i |g_i(x,u)| - \sum_{i=1}^{m} u_{i+n} \min\{0,h_i\}$

$$\hat{x} = x + \alpha d_x$$

$$\hat{u} = u + \alpha d_u$$

iv) Set $x = \hat{x}$, $u = \hat{u}$, $\delta = \alpha \begin{bmatrix} d_x \\ d_u \end{bmatrix}$

Step 5: Check for convergence

i) Let $\varphi = \delta^T \begin{bmatrix} b_x \\ b_u \end{bmatrix} + \sum_{i=1}^{n} u_i |g_i(x,u)|$

ii) If ($\varphi \le \epsilon$) print results and go to 6

iii) If (k < (maximum number of iterations)) go to 2; otherwise print error message.

Step 6: STOP

In our work we have found it to be important to pay particular attention to two seemingly unimportant comments made by Powell (1977a). The first comment is with respect to the scaling of variables. In the Bracken and McCormick sample problem which we have included in this paper, the variables were initially scaled very poorly. The result was that, when we applied Powell's algorithm to the unsealed problem the QPP predicted very small moves for the variables. We then scaled the variables so that they were all between 0.1 and 10.0; the search converged in 11 iterations. Powell reports that his method is insensitive to the scaling of the constraints, and our experience seems to confirm his observations. We scale the equality constraints in order to determine whether they have been converged to within an acceptable tolerance.

The second point worth noting is that Powell uses a dummy variable to insure that the QPP will always be able to find a feasible solution. This variable, £ > is constrained between zero and one, and it is multiplied by a large positive constant in the objective function of the quadratic program. Powell multiplies the right-hand side of each inequality constraint that has a positive residual and each equality constraint by (l-§). This procedure is helpful especially when starting far away from the solution where the linearizations are more likely to give rise to a set of inconsistent constraints.

<u>Example Problem</u>

Bracken and McCormick (1968) describe a simplified model of an alkylation process. The process is illustrated schematically in Figure 1. Fresh olefins and isobutane are added to the reactor along with a large

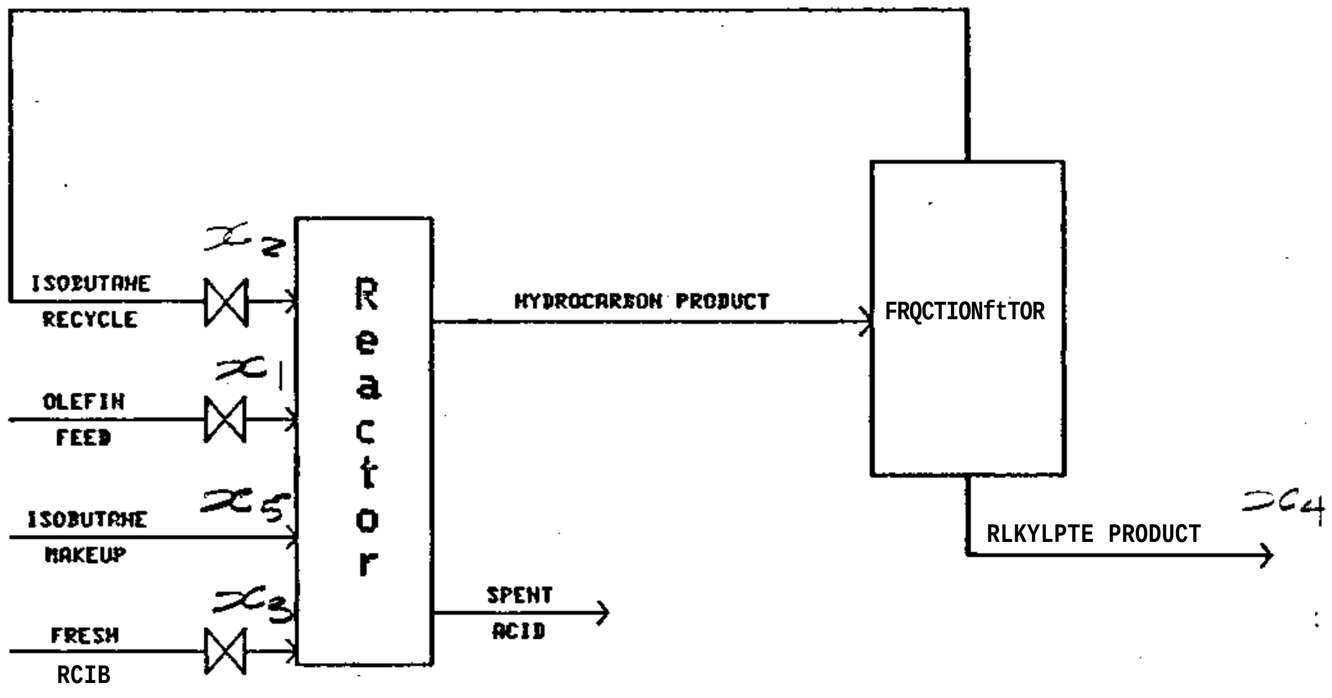

Figure 1. Schematic Diagram of Alkylation Process
(Bracken and McCormick, 1969)

recycle stream which contains unreacted isobutane. Fresh sulfuric acid is added to the reactor to catalyze the reaction, and waste acid is removed. The reactor effluent is sent to a fractionater where the alkylate product is separated from the unreacted isobutane. The formulation used by Bracken and McCormick is given in Table 3. We converted the inequality constraints to equalities by introducing slack variables. In our formulation the variables are all scaled and all divisions have been removed from the constraints and their derivatives by introducing new variables as needed. Our experience has been that the Newton-Raphson method converges well provided that there are no poles (e.g. divisions by a variable which could become zero) in the constraints or in the Jacobian matrix. Table 4 contains a description of our formulation of the problem, and Table 5 contains information about each of the variables in the problem. The optimum solution given in Table 5 is essentially the same as that reported by Bracken and McCormick.

Westerberg and deBrosse (1973) solved this same problem. Their algorithm appeared to be superior to others available, but the present algorithm converges to the desired solution faster than their algorithm. Westerberg and deBrosse require a feasible starting point; to obtain this, they took 8 iterations with their feasible point algorithm. Once feasible, their algorithm required 15 iterations to reach the optimum. The present algorithm requires a total of 11 iterations to reach the optimum from the initial infeasible point. The present algorithm requires only 8 iterations to reach the optimum from Wfesterberg and deBrosse^{1}s first feasible point.

The advantage associated with using the above algorithm is clear; in only 11 function evaluations the method converged to the solution. This has been our experience with small problems; we can obtain an optimum solution in about the same number of iterations as one normally requires to obtain a single solution to the set of equality constraints only.

Table 3. Bracken & McConnick Formulation of Alylation Process Optimization

Number of Variables	10
Equality Constraints	3
Inequality Constraints	28

Max $.063\ x_1 x_2 - 5.04\ x_1 - .035\ x_2 - 10.00\ x_3 - 3.36\ x_5$

Subject to $x_{\min j} \text{ £ } x_j \text{ £ } x_{\max J}$ j = I,-**, 10

$[x.(1.12 + .13167\ x_Q - .00667\ x\^{})] - .99\ x_, \ \^{}\ 0$

$- [x_1(1.12 + .13167\ x_{..} - .00667\ x_{*}] + X./.99\ \^{}\ 0$

$[86.35 + 1.098\ x_g - .038\ X_8^* + .325(x_6 - 89)] - .99\ x_? \ \^{}\ 0$

$- [86.35 + 1.098\ Xg - .038\ Xg^2 + .325(x_6 - 89)3 + x_y/.99\ \^{}\ 0$

$[35.82 - .222\ x_{1()}3 - .9\ x_g\ \^{}\ 0$

$- [35.82 - .222\ x\^{}] + x_g/19 * 0$

$[- 133 + 3\ x_?] - .99\ x_{10} \geq 0$

$- [- 133 + 3\ x_?] + x_{1Q}/.99\ \^{}\ 0$

$1.22\ x_4 - x]L - x_5 - 0$

$$\frac{98,000}{x_4 x_9 + 1000\ x_3} - x_6\ m\ Q$$

$$\frac{x_2 + x_5}{x_1} x_8 = 0$$

Table 4. Akylation Process Formulation as Solved by Present Algorithm

Min $

s.t.

$ + 6,3 $x_4x_?$ - 5.04 x_1 - .35 x_2 - x_3 - 3.36 x_5 = 0.

1.22 x_4 - x_1 - x_5 = 0.

.98 X_3 - $*_6(X_4X_9$ -f" l°0- + $^x{}_3$) = 0.

1.01 x_o + x_- - x_nx_o = 0.
Z J 1 0

x_-(1.12 + .13167x_o - .0067 x_8^2) - .99 x_4 - s. - 0.
1 0 1

-x.(1.12 + .13167 x_{-} - .0067 x^) + x. -7- .99 - s_o = 0.

.8635 + (1.098 x_g - .038 x_g) -7- 100. + .325 (x_g - .89) - .99 $x_?$- s_3 = 0.

-.8635 - (1.098 x_g - .038 Xg^2) -7- 100. - .325 ($x_\&$ - .89) + $x_?$ -r- .99 - s_4= 0,

35.82 - 22.2 $*_{1Q}$ - .90 x_g - s_5 = 0.

-35.82 + 22.2 x_{1Q} + x_g -7- .90 - $s_\&$ = 0.

-1.33 + 3 $x_?$ - .99 x_{1Q} - $s_?$ = 0.

1.33 - 3 x_- + x_{in} -7- .99 - s_o = 0.

x . £ x. £ x i = 1,···,10
min1 1 max1

0 £ $s_\pm$ i = l,##-,8

Values for x_{min} and x_{max} are given in Table 5.

Table 5. Summary of Sample Problem as Solved with Powell's Algorithm

Variable	Lower Bound	Upper Bound	Starting Value	Optimum Value
Φ	—	—	-0.900	-1.765
x_1	0	2.00	1.745	1.704
x_2	0	1.60	1.200	1.585
x_3	0	1.20	1.100	0.543
x_4	0	5.00	3.048	3.036
x_5	0	2.00	1.974	2.000
x_6	.85	.93	0.892	0.901
x_7	.90	.95	0.928	0.950
x_8	3.0	12.0	8.000	10.476
x_9	1.2	4.0	3.600	1.562
x_{10}	1.45	1.62	9.450	1.535
s_1	0	—	0.000	0.000
s_2	0	—	0.000	0.061
s_3	0	—	0.000	0.000
s_4	0	—	0.000	0.019
s_5	0	—	0.000	0.330
s_6	0	—	0.000	0.000
s_7	0	—	0.000	0.000
s_8	0	—	0.000	0.031

ACKNOWLEDGEMENT

This work was funded by NSF Grant ENG-76-80149

REFERENCES

Abadie, J. and J. Carpentier (1969). "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints," in Optimization, Academic Press (New York) pp 37-47.

Biggs, M.C. (1972). "Constrained Minimization Using Recursive Equality Quadratic Programming," in Numerical Methods for Nonlinear Optimization, ed. F.A. Lootsma, Academic Press (London).

Bracken, J. and G.P. McCormick (1968). Selected Applications of Nonlinear programming, Wiley and Sons (New York) pp 37-45.

Charlambous, C. (1978). "Some Recent Advances in Nonlinear Programming," 2nd International Symposium on Large Engineering Systems, Proceedings,

Colville, A.R. (1968). "A Comparative Study on Nonlinear Programming Codes," Report No. 320-2949 (IBM New York Scientific Center).

Fletcher, R. (1975). "An Ideal Penalty Function for Constrained Optimization," J. Inst. Math. Applies., 15, 319.

Friedman, P. and K.-Pinder (1972). "Optimization of a Simulation Model of a Chemical Plant," IEC Proc. Des. Dev., 11, 512.

Han, S.-P. (1975). "A Globally Convergent Method for Nonlinear Programming," Report. No. 75-257 (Dept. of Computer Science, Cornell University).

Powell, M.J.D. (1977a). "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," Paper presented at the 1977 Dundee Conference on Numerical Analysis.

Powell, M.J.D. (1977b). "The Convergence of Variable Metric Methods for Nonlinearly Constrained Optimization Calculations," Paper presented at the "Nonlinear programming 3" symposium held at Madison, Wisconsin.

Powell, M.J.D. (1978). "Algorithms for Nonlinear Constraints that use Lagrangian Functions," Math. Prog., 14, 224.

Reid, J.K. (ed) (1971). Large Sparse Sets of Linear Equations, Academic Press (London).

Westerberg, A.W. and T.J. Berna (1978). "Decomposition of Very Large-Scale Newton-Raphson Based Flowsheeting Problems," accepted for publication in Computer and Chemical Engineering Journal.

Westerberg, A.W. and C.J. deBrosse (1973). "An Optimization Algorithm for Structured Design Systems," AIChE J., 19, (2) 335.