Abstract

Following from part I, which presents a retrospective on optimization, we focus here on areas that are recent active research topics and are likely to strongly influence the future of optimization algorithms and formulations. First, we discuss recent developments in deterministic global optimization algorithms applied to both nonlinear programs and mixed-integer programs. Second, we discuss logic-based optimization and its influence in both modeling and solving mixed-integer optimization problems. Third, we discuss issues and approaches related to large-scale optimization algorithms and applications. Finally, we summarize recent progress in scientific computing and software engineering as applied to optimization applications.

© 2004 Elsevier Ltd. All rights reserved.

Keywords: Nonlinear programs; Optimization; Scientific computing

1. Introduction

In the first paper (Biegler & Grossmann, 2004), we provided a general review on optimization for process systems engineering that have been extensively studied and applied, namely, nonlinear programming, mixed-integer nonlinear programming, dynamic optimization, and optimization under uncertainty. As was shown in that review, considerable progress has been made in these areas, although it is clear that major research questions remain to be solved, particularly in regard to effective solution methods for optimization under uncertainty.

In this second part we outline future directions of research, some of which may also have been the subject of past research, but are still largely unsolved and therefore are likely to be the subject of further significant research work over the next decade or so. In Section 1, we describe developments in global optimization, which still represents a major challenge in nonlinear optimization. Section 2 then describes developments in logic-based optimization as a promising approach to more effectively model and solve discrete optimization problems. In Section 3, we then address the issue of large-scale computation in which the major goal is to increase the capability of optimizing models that are orders of magnitude larger than what is currently possible. We continue in Section 4 by describing developments in advanced scientific computation and software engineering that will allow to further exploit the computational power to address very large problems. Section 5 then concludes with a brief summary of the paper.

2. Global optimization

One of the major limitations of the NLP and MINLP methods reviewed in Biegler and Grossmann (2004) is that they are not guaranteed to find the global optimum in nonconvex problems. In some respect considerable work has been performed in the area of global optimization over the last 25 years. However, it is mostly in the last 10 years that many significant developments have taken place not only by both operations researchers, but also to a large extent by chemical engineers. In fact, a good milestone of when the level of research activity in this area started to rapidly increase was 1991, the year of when the Journal of Global Optimization first was published. Despite the many advances, global optimization will remain in the foreseeable future as a major challenge that is likely to occupy the attention of future research efforts.

2.1. Stochastic methods

Global optimization methods can generally be classified as stochastic and deterministic. Stochastic methods, which
were briefly reviewed in Biegler and Grossmann (2004), are methods that often rely on physical analogies in order to generate trial points that mimic the approach to an equilibrium condition. Examples include simulated annealing (Kirkpatrick et al., 1983), and genetic algorithms (Holland, 1975). While these methods are relatively simple to implement, they do require that the user model the problem in terms of recursive moves, which is not always easy to do, especially for continuous variables. Furthermore, these methods are not rigorous, and tend to have difficulty in handling complex constrained problems. However, stochastic methods are suitable for problems where the function evaluations are cheap, and often produce good suboptimal solutions. Successful applications include, for instance, scheduling and product design problems. It should also be noted that a variant of stochastic methods are heuristic search methods, as is, for instance, the case of Tabu search (Glover, 1990), which consists of a sequence of local moves followed by random explorations into new regions (see also Battiti & Tecchiolli, 1994). Here again no rigorous global optimality can be guaranteed, although the performance of Tabu search has been found to be more effective than simulated annealing or genetic algorithms since fewer adjustable parameters are required.

2.2. Deterministic methods for NLP and MINLP

The development of deterministic methods for nonconvex continuous optimization has received considerable attention in the literature. These methods include, for instance: (a) Lipschitzian methods (e.g. Hansen, Jaumard, & Lu, 1992a,b); (b) branch and bound methods (e.g. Al-Khayyal, 1992; Al-Khayyal & Falk, 1983; Horst & Tuy, 1987); (c) cutting plane methods (e.g. Tuy, Thi, & Thai, 1985); (d) difference of convex (DC) and reverse convex methods (e.g. Tuy, 1987); (e) outer-approximation (OA) methods (e.g. Horst, Thoai, & De Vries, 1992); (f) primal-dual methods (e.g. Ben-Tal, Eiger, & Gershovitz, 1994; Floudas & Visweswaran, 1990, 1993; Shor, 1980); (g) reformulation-linearization methods (e.g. Sherali & Alameddine, 1992; Sherali & Tunc bilek, 1992); and (h) interval methods (e.g. Hansen, 1980).

Rigorous global optimization approaches for addressing nonconvexities in NLP and MINLP problems can be developed when special structures are assumed in the continuous terms (e.g. bilinear, linear fractional, concave separable). Specifically, the idea is to use convex envelopes or underestimators to formulate lower-bounding convex MINLP problems. These are then combined with global optimization techniques for continuous variables (Falk & Soland, 1969; Floudas, 2000; Grossmann, 1996; Horst & Tuy, 1996; Quesada & Grossmann, 1995; Ryoo & Sahinidis, 1995; Zamora & Grossmann, 1999), which usually take the form of spatial branch and bound methods (Soland, 1971). Most of the methods proposed for solving these problems rely on the spatial branch and bound method, which is a deterministic algorithm that divides the feasible region of continuous variables and compares lower bound and upper bound for fathoming each subregion (see Fig. 1). The subregion that contains the optimal solution is found by eliminating subregions that are proved not to contain the optimal solution. An example of such a method for nonconvex NLP problems is the one by Quesada and Grossmann (1995) who proposed a spatial branch and bound algorithm for concave separable, linear fractional and bilinear programs, and making use of linear and nonlinear underestimating functions.

As for methods for nonconvex MINLP, Ryoo and Sahinidis (1995) have developed a branch and bound method that branches on the continuous and discrete variables. This method, which relies on bounds reduction and use of underestimators, has been implemented in BARON. Adjiman, Androulakis, &Floudas (1997, 2000) proposed the SMIN-αBB and GMN-αBB algorithms for twice-differentiable nonconvex MINLPs. By using a valid convex underestimation of general functions as well as for special functions, Adjiman and Floudas (1996) developed the αBB method which is a branch and bound procedure that branches on both the continuous and discrete variables according to specific options. Smith and Pantelides (1999) proposed the reformulation/spatial branch and bound algorithm for nonconvex MINLP which is implemented in gPROMS modeling environment (Barton & Pantelides, 1994). The branch-and-contract method (Zamora & Grossmann, 1998b, 1999) for global optimization of process models which have bilinear, linear fractional, and concave separable functions in the continuous variables and linear 0–1 variables, uses bound contraction and applies the outer-approximation algorithm at each node of the tree for the spatial search. Kesavan and Barton (2000b) developed a generalized branch-and-cut (GBC) algorithm, and showed that their earlier decomposition algorithm (Kesavan & Barton, 2000a) is a specific instance of the GBC algorithm with a set of heuristics. Lee and Grossmann (2001) developed a two-level branch and bound method for solving nonconvex disjunctive programming problems.

All of these methods rely on a spatial branch and bound procedure. The difference lies on how to perform the
branching on the discrete and continuous variables. Some methods perform the spatial tree enumeration on both the discrete and continuous variables on the bounding problem (LBMP) shown below. Other methods perform a spatial branch and bound on the continuous variables and solve the corresponding MINLP problem (LBMIP) at each node using any of the methods reviewed in Biegler and Grossmann (2004). Finally, other methods, branch on the discrete variables of problem (LBMP), and switch to a spatial branch and bound on nodes where a feasible value for the discrete variables is found. The methods also rely on procedures for tightening the lower and upper bounds of the variables, since these have a great effect on the quality of the underestimators. Since the tree searches are not finite (except for $\epsilon$-convergence), these methods can be computationally expensive. However, their major advantage is that they can rigorously find the global optimum. It should also be noted that specific cases of nonconvex MINLP problems have been handled. An example is the work of Pörn and Westerlund (2004), who have addressed the solution of MINLP problems with pseudo-convex objective function and convex inequalities through an extension of the ECP method.

2.3. Convex relaxations

In order to describe in some detail how to construct a lower bounding MINLP problem, or NLP if there are no discrete variables $y$, consider the (MIP) problem

$$\min \ Z = f(x, y)$$

s.t. \ $g_j(x, y) \leq 0 \ j \in J$

$$x \in X, \ y \in Y$$

where the functions $f(x, y)$ and $g(x, y)$ are generally nonconvex. The lower bounding MINLP problem has the general form

$$\min \ Z = \tilde{f}(x, y)$$

s.t. \ $\tilde{g}_j(x, y) \leq 0 \ j \in J$

$$x \in X, \ y \in Y$$

where $\tilde{f}$, $\tilde{g}$, are valid convex underestimators such that $f(x, y) \leq \tilde{f}(x, y)$, and the inequalities $g_j(x, y) \leq 0$ are satisfied if $g(x, y) \leq 0$. A geometric example that illustrates the use of underestimators in nonconvex functions to predict a lower bound GLB is shown in Fig. 1.

We discuss below the major classes of nonconvexities, and their corresponding underestimators, as well as strategies for reduction of bounds.

2.4. Concave minimization

Concave minimization and its various special forms is the most often treated class of global optimization problems. Recent algorithms proposed for the solution of the minimization of a concave function over a convex feasible region include Tuy and Horst (1988) and Benson and Horst (1991). Another type of commonly treated global optimization problem is Lipschitz optimization. For a review of this area see Hansen et al. (1992a,b).

Univariate concave separable function $f_j(x)$ can be underestimated by a secant line which matches concave function at the upper and lower bound (Falk & Soland, 1969):

$$f_j(x) \geq f_j(x^*) + (x - x^*) \frac{f_j(x^*) - f_j(x)}{x^* - x} \quad (C1)$$

2.5. Bilinear and fractional models

An important class of nonconvex optimization problems correspond to nonlinear programming problems with bilinear or linear fractional terms. These commonly arise, for instance, in engineering design problems (e.g. see Floudas & Pardalos, 1990; Grossmann, 1990; Papalambros & Wilde, 1988; Reklaitis & Ravindran, 1983).

Al-Khayyal (1992) presented a review of models and applications of bilinear programming. The bilinear and linear fractional terms are factorable functions for which McCormick (1976) has presented a general approach for deriving underestimator functions that can be incorporated in global optimization algorithms. Al-Khayyal and Falk (1983) proposed an algorithm for bilinear programs with linear constraints in which linear estimators over the bilinear terms are used. Swaney (1993) addressed the asymptotic behavior that can occur in this type of algorithm when a solution does not lie at an extreme point. Algorithms for bilinear programming models have also been developed by Sherali and Alameddine (1992). These authors presented a linearization reformulation technique that embeds the method proposed by Al-Khayyal and Falk (1983) and predicts stronger bounds for the global optimum. However, the main limitation is that the size of the linear programming underestimator problems grows exponentially with the number of constraints in the original problem.

Falk and Palocsay (1992) proposed an algorithm for optimizing the sum of linear fractional functions subject to linear constraints. The algorithm consists of a sequence of linear programming problems in which bounds on feasible subsets are added. These bounds are tightened iteratively to reduce the search space. These authors also developed convergence properties for this algorithm by extending the approach presented by Dinkelbach (1967). However, the rate of convergence of this method can be slow. Konno, Yajima, and Matsui (1991) addressed the minimization of the sum of two linear fractional functions over a polytope for which they applied parametric linear programming algorithms. Recently Falk and Palocsay (1992) presented a generalization of their approach to the case of products of ratios of linear terms.

Floudas and Vicente (1990) presented an algorithm based on a Benders-based decomposition approach that can be used to solve bilinear and/or fractional programming problems. In this method a sequence of subproblems and
relaxed dual subproblems are solved. Although the advantage of this method is that there is freedom for choosing the structure of the subproblems (e.g. they can correspond to linear programs or some advantageous type of problem), one potential difficulty is that the number of relaxed dual subproblems that has to be solved at each iteration grows exponentially with the number of variables that are selected to decompose the problem.

Regarding the construction of underestimators, consider first a bilinear function $x_i x_j$. Valid convex/concave envelopes are as follows (Al-Khayyal & Falk, 1983; McCormick, 1976):

$$ x_i x_j \geq x_i^U x_j + x_i x_j^L - x_i^U x_j^L \quad \text{(BL)} $$

$$ x_i x_j \geq x_i^L x_j + x_i x_j^U - x_i x_j^L \quad \text{(BL)} $$

$$ x_i x_j \leq x_i^U x_j + x_i x_j^L - x_i^U x_j^L \quad \text{(BL)} $$

$$ x_i x_j \leq x_i^L x_j + x_i x_j^U - x_i^L x_j^U \quad \text{(BL)} $$

For a linear fractional function $x_i / x_j$, Quesada and Grossmann (1995) developed nonlinear convex underestimators.

$$ \frac{x_i}{x_j} \geq \frac{x_i}{x_j} \left( x_i + \sqrt{x_i^2 + \alpha^2} \right)^2 \quad \text{(LF1)} $$

$$ \frac{x_i}{x_j} \geq \frac{x_i}{x_j} \left( x_i + \sqrt{x_i^2 + \alpha^2} \right)^2 \quad \text{(LF2)} $$

Alternate convex underestimator/envelopes are the following:

$$ \frac{x_i}{x_j} \geq \frac{x_i}{x_j} \left( x_i + \sqrt{x_i^2 + \alpha^2} \right)^2 \quad \text{(LF3)} $$

It can easily be shown that the function $Z(x)$ is convex, and underestimates $Z(x)$ for sufficiently large $\alpha$. However, this underestimation may yield very loose lower bounds. Maranas and Floudas (1994a,b) showed that the choice of $\alpha$ is restricted to be a non-negative parameter which is greater or equal to the negative one half of the minimum eigenvalue of the Hessian of $f(x)$ over the box $x_i \leq x_i \leq x_i^U, i = 1, 2, \ldots, n$. That is

$$ (\alpha \geq \max(0, -1/2 \text{min})) $$

Since the term added to $f(x)$ has the effect of overpowering the nonconvexity characteristics of $f(x)$ with the addition of the term (2\alpha) to all of the eigenvalues of its Hessian. The smaller the value of $\alpha$, the tighter the underestimator $Z(x)$. A detailed and elegant analysis of this issue can be found in Adjiman and Floudas (1996). Also, this underestimation of general functions, combined with the underestimation of special classes, have been implemented within a spatial branch and bound method, known as the $\alpha$-BB method (Adjiman et al., 2000).

### 2.7. Variable bounds reduction

While the use of the above underestimating functions provides lower bounds to the global optimum, the quality of these bounds is strongly a function of the bounds on the variables. This simply follows from the fact that the underestimating functions such as in (C1), (BL), (LF1), (LF2), (LF3), are all expressed in terms of the lower and upper bounds of the variables. In order to obtain the “optimum” bounds one can reduce the size of the hyperrectangle defined by the lower and upper bounds, $x_i^L \leq x_i \leq x_i^U$, by expanding the convex relaxation of the corresponding branch and bound node. Assuming that the (MIP) problem has linear or convex separable discrete variables $y$, following contraction subproblem for the variables $x_i$ can be defined.

$$ \min \text{ or max } x_i \quad \text{s.t.} \quad \tilde{f}(x, y) \leq \text{OUB} \quad \text{for } x \in X, \quad y \in Y $$

The parameter OUB in (CS) corresponds to the current upper bound of the objective. The optimization direction, min or max.
max, is selected depending upon which of the bounds, $x^i$ or $\tilde{x}^i$, to be contracted.

The simplest approach in applying problem (CS) is to optimize the variable bounds prior to performing the global optimization (e.g. Quesada & Grossmann, 1993). The drawback, however, is that this can be an expensive proposition if there are many variables and the (CS) problem is nonlinear. Therefore, two major schemes that have been proposed are as follows. First, a bound contraction strategy can be developed (e.g. Zamora & Grossmann, 1999) in which the reduction is performed by sequentially choosing variables that potentially can lead to the greatest reduction. Also, the subproblems are repeated until a certain tolerance in the reduction of bounds is achieved. The second approach, often known as range reduction (Ryoo & Sahinidis, 1995, 1996) is to generate the following bounding inequalities for the variables involved in the solution of the relaxation of problem (LBMIP) at each node of the branch and bound tree:

$$x_j \leq x^L_j + \frac{\tilde{x}^L_j - x^L_j}{\lambda_j} \quad \text{(UBR)}$$

for the case when the constraint $x^L_j - \tilde{x}_j \leq 0$, $j \in J$, is active at the solution of the subproblem (LBMIP).

$$x_j \geq \frac{x^U_j - \tilde{x}^L_j}{\lambda_j} \quad \text{(LBR)}$$

for the case when the constraint $x^L_j - \tilde{x}_j \leq 0$, $j \in J$, is active. In both cases, a Lagrange multiplier $\lambda_j > 0$ is required for the application of these bounding inequalities.

### 2.8. Future challenges

While significant progress has been made in global optimization, there are clearly still a number of major challenges that need to be addressed.

First, existing methods for global optimization of NLP and MINLP can still be computationally very expensive to solve. Special reformulation strategies such as the ones that are outlined in Smith and Pantelides (1999) in order to handle algebraic models in terms of basic functional forms (bilinear, fractional, concave separable), need to be studied further and implanted effectively. Also, computational strategies that are, for instance, implemented in BARON (Sahinidis, 1996) or in $\alpha$-BB (Adjiman, Androulakis, Maranas, & Floudas, 1996) need to be further developed and investigated, as well as implemented in modern parallel machines (see Section 4). Second, the extension of global optimization to problems of optimization under uncertainty (Floudas, Gumus, & Ierapetritou, 2001; Ierapetritou, Pistikopoulos, & Floudas, 1996), or to dynamic and mixed-integer dynamic problems (Singer & Barton, 2001), will clearly require either radically new ideas, or a quantum improvement in computational capabilities. On the other hand, it is important to recognize that current capabilities for global optimization are already quite impressive, so there is reason for optimism in the future of this area.

### 3. Logic-based methods

One of the emerging areas related to discrete optimization is logic-based optimization. The major motivation in this area lies in developing symbolic representations that can facilitate the modeling of discrete constraints, and motivate more effective solution techniques that can help to reduce the computational complexity of discrete/continuous optimization problems. We first review briefly logic representations, and then discuss the development of logic-based optimization techniques.

#### 3.1. Logic representations

Propositional logic can generally be used as a basis for developing modeling and solution techniques for discrete optimization problems (see Mendelson, 1987, for general review on logic). The basic unit of a propositional logic expression, which can correspond to a state or to an action, is called a literal which is a single variable that can assume either of two values, true or false. Associated with each literal $Y_i$ its negation $\neg Y_i$ is such that $[\neg Y_i \lor Y_i]$ is always true. A disjunctive clause is a set of literals separated by OR operators $[\lor]$, and is also called a disjunction. A proposition is any logical expression and consists of a set of clauses $P_i$, $i = 1, \ldots, r$ that are related by the logical operators OR $[\lor]$, AND $[\land]$, IMPLICATION $[\Rightarrow]$.

In synthesis logic propositions usually refer to relations of existence of units in a superstructure. These are commonly expressed by a set of conjunctions of clauses

$$A = L_1 \land L_2 \land \cdots \land L_q \quad \text{(L1)}$$

where $L_i$ is a logical proposition expressed with Boolean variables $Y_i$ in terms of implications, OR, EXCLUSIVE OR and AND operators. In synthesis problems $Y_i$ is a Boolean variable representing the existence of unit $i$ and $\neg Y_i$ its nonexistence. There are two ways of transforming the propositions in $A$. In the simplest case, the logic propositions are converted into the conjunctive normal form (CNF) by removing the implications through contrapositions in each of the clauses $L_i$ in (L1) and applying De Morgan’s theorem. In this way each clause in the CNF form consists of only OR operators with non-negated and negated Boolean variables as follows:

$$A = \lor_{a \in P_1} Y_{a} \land \lor_{a \in \bar{P}_1} \neg Y_{a} \land \cdots \land \lor_{a \in P_s} Y_{a} \land \lor_{a \in \bar{P}_s} \neg Y_{a} \quad \text{(L2)}$$

where $P_i$ and $\bar{P}_i$ are subsets of the Boolean variables that correspond to some of the 0–1 variables, and $s$ is the number of clauses.
In the second representation, the logic propositions in the CNF form are converted into the disjunctive normal form (DNF) (see Clocksin & Mellish, 1984) by moving the AND operators inwards and the OR operators outwards by applying elementary Boolean operations. The DNF form is as follows:

\[ \bigvee_{i=1}^{r} \bigwedge_{j=1}^{n} (Y_i) \bigvee \left[ \bigwedge_{i=1}^{n} (Y_i) \bigwedge \bigvee_{i=1}^{n} (\neg Y_i) \right] \bigvee \cdots \bigvee \left[ \bigwedge_{i=1}^{n} (Y_i) \bigwedge \bigvee_{i=1}^{n} (\neg Y_i) \right] \]

where \( Q_i \) and \( \bar{Q}_i \) are the index sets of the Boolean variables which correspond to a partition of all the 0–1 variables \( Y_i \) \( i = 1, \ldots, p \) in non-negated and negated terms. Each clause separated by a disjunction represents the assignment of units in a feasible configuration, where it is assumed that each Boolean variable has a one-to-one correspondence with the 0–1 binary variables of the MILP model. Therefore, \( r \) represents the number of alternatives in the superstructure. While the DNF form is more convenient to manipulate, the drawback is that the transformation from CNF to DNF has exponential complexity in the worst case.

In order to obtain an equivalent mathematical representation for any propositional logic expression, this can be easily performed using the CNF form as a basis. We must first consider basic logical operators to determine how each can be transformed into an equivalent representation in the form of an equation or inequality. These transformations are then used to convert general logical expressions into an equivalent mathematical representation (Cavaler & Soyster, 1987; Cavaler, Pardalos, & Soyster, 1990).

To each literal \( P_i \), a binary variable \( y_i \) is assigned. Then the negation or complement of \( P_i (\neg P_i) \) is given by \( 1 - y_i \). The logical value of true corresponds to the binary value of 0. The basic operators used in propositional logic and the representation of their relationships are shown in Table 1.

With the basic equivalent relations given in Table 1 (e.g. see Williams, 1985), one can systematically model an arbitrary propositional logic expression which is given in terms of OR, AND, IMPLICATION, operators, as a set of linear equality and inequality constraints. One approach is to systematically convert the logical expression into its equivalent conjunctive normal form representation which involves the application of pure logical operators. The conjunctive normal form is a conjunction of clauses, \( Q_1 \land Q_2 \land \cdots \land Q_r \).

Hence, for the conjunctive normal form to be true, each clause \( Q_i \) must be true independent of the others. Also since a clause \( Q_i \) is just a disjunction of literals, \( P_1 \lor P_2 \lor \cdots \lor P_r \), it can be expressed in the linear mathematical form as the inequality:

\[ y_1 + y_2 + \cdots + y_r \geq 1 \]

Table 1: Representation of logical relations with linear inequalities

<table>
<thead>
<tr>
<th>Logical relation</th>
<th>Comments</th>
<th>Boolean expression</th>
<th>Representation as linear inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical OR</td>
<td></td>
<td>( P_1 \lor P_2 \lor \cdots \lor P_r )</td>
<td>( y_1 + y_2 + \cdots + y_r \geq 1 )</td>
</tr>
<tr>
<td>Logical AND</td>
<td></td>
<td>( P_1 \land P_2 \land \cdots \land P_r )</td>
<td>( y_1 \leq 1 )  ( y_2 \leq 1 )  ( \vdots )  ( y_r \leq 1 )</td>
</tr>
<tr>
<td>IMPLICATION</td>
<td></td>
<td>( P_1 \Rightarrow P_2 )</td>
<td>( \neg P_1 \lor P_2 )</td>
</tr>
<tr>
<td>Equivalence</td>
<td>if and only if ( P_1 \Rightarrow P_2 \land (P_2 \Rightarrow P_1) )</td>
<td>( \neg P_1 \lor P_2 \land \neg (\neg P_2 \lor P_1) )</td>
<td>( y_1 = y_2 )</td>
</tr>
<tr>
<td>EXCLUSIVE OR</td>
<td>exactly one of the variables is true</td>
<td>( P_1 \lor P_2 \lor \cdots \lor P_r )</td>
<td>( y_1 + y_2 + \cdots + y_r = 1 )</td>
</tr>
</tbody>
</table>

3.2. Symbolic and mathematical methods for logic inference

The most common logic inference problem is the satisfiability problem where, given the validity of a set of propositions, one has to prove the truth or validity of a conclusion which may be either a literal or a proposition. This inference problem is one of the basic issues in artificial intelligence and data bases. However, the general satisfiability problem for propositional logic is NP-complete (Cook, 1971; Karp, 1972). Therefore, research has focused on identifying classes of problems within the general satisfiability problem that can be solved efficiently. Knowledge-based systems commonly require the use of Horn clauses systems which have at most one non-negated literal in each clause. The inference problem for this class of propositional logic problems can be solved in linear time using unit resolution (Dowling & Gallier, 1984). The unit resolution technique (e.g. see Clocksin & Mellish, 1984) is one of the most common inference techniques, and in simple terms, it consists of solving sequentially each logic clause one at a time.

Chandru and Hooker (1989) have extended the class of problems that can be solved in linear time to include extended Horn clause systems. One of the most effective logic-based methods for solving the general satisfiability problem is the algorithm of Davis and Putnam (1960) as treated by Loveland (1978). This approach is closely related to the branch and bound method for mixed-integer programming. Jeroslow and Wang (1990) have developed branching heuristics to improve the performance of the...
Davis–Putnam procedure. It must be noted that although the previous work has been restricted to propositional logic, the techniques used for this class are essential to higher order representations like predicate logic that involve additional logic operators like for all.

3.3 Disjunctive programming

Recently there has been a new trend of representing discrete/continuous optimization problems by models consisting of algebraic constraints, logic disjunctions and logic relations (Balas, 1985): the context of synthesis problems the disjunctions in (GDP) express relationships between the disjunctive sets. In the form of the feasible space in which the continuous variables, \( x \) and \( c \), can be interpreted as weight factors that determine the validity of the inequalities in the corresponding disjunctive term. Note also that (CHk) reduces to the result by Balas and Mehrotra (1999) for the case of linear constraints.

In problem (GDP), \( Y_k \) are the Boolean variables that establish whether a given term in a disjunction is true \([h_k(x) \leq 0]\), while \( \Omega(Y) \) are logical relations assumed to be in the form of propositional logic involving only the Boolean variables. \( Y_k \) are auxiliary variables that control the part of the feasible space in which the continuous variables, \( x \), and the variables \( c_i \) represent fixed charges which are activated to a value \( y_{ik} \) if the corresponding term of the disjunction is true. Finally, the logical conditions, \( \Omega(Y) \), express relationships between the disjunctive sets. In the context of synthesis problems the disjunctions in (GDP) typically arise for each unit \( i \) in the following form:

\[
\begin{bmatrix}
Y_i \\
\ h_k(x) \leq 0 \\
\ c_i = y_{ik} 
\end{bmatrix} \lor \begin{bmatrix}
\neg Y_i \\
\ B^x \geq 0 \\
\ c_i = 0 
\end{bmatrix}, \quad i \in I
\]

In (CHk) the variables \( y_{ik} \) can be interpreted as disaggregated variables that are assigned to each disjunctive term, while \( \lambda_{ik} \), can be interpreted as weight factors that determine the validity of the inequalities in the corresponding disjunctive term. Note also that (CHk) reduces to the result by Balas (1985) for the case of linear constraints.

From the above, it clearly follows that the nonlinear programming relaxation of (GDP) is given by

\[
\min \ Z = \sum_k c_k + f(x)
\]

s.t.

\[
\begin{align}
\nu \in \mathbb{D} & : \quad h_k(x) \leq 0, \quad k \in SD \\
\Omega(Y) & : \quad Y \in R^n, \quad \nu \in \text{true}, \text{false}^n
\end{align}
\]

In problem (GDP), \( Y_k \) are the Boolean variables that establish whether a given term in a disjunction is true \([h_k(x) \leq 0]\), while \( \Omega(Y) \) are logical relations assumed to be in the form of propositional logic involving only the Boolean variables. \( Y_k \) are auxiliary variables that control the part of the feasible space in which the continuous variables, \( x \), and the variables \( c_i \) represent fixed charges which are activated to a value \( y_{ik} \) if the corresponding term of the disjunction is true. Finally, the logical conditions, \( \Omega(Y) \), express relationships between the disjunctive sets. In the context of synthesis problems the disjunctions in (GDP) typically arise for each unit \( i \) in the following form:

\[
\begin{bmatrix}
Y_i \\
\ h_k(x) \leq 0 \\
\ c_i = y_{ik} 
\end{bmatrix} \lor \begin{bmatrix}
\neg Y_i \\
\ B^x \geq 0 \\
\ c_i = 0 
\end{bmatrix}, \quad i \in I
\]
ties, and corresponding to the Boolean variable to construct a special purpose branch and bound method as has been proposed by Lee and Grossmann (2000). The basic idea in this method is to directly branch on the constraints corresponding to particular terms of each disjunction, while considering the convex hull of the remaining disjunctions or disjunctive terms. Compared to the conventional branch and bound method applied to the equivalent MINLP problem, the disjunctive branch and bound often yields tighter lower bounds. Alternatively, problem (RDP) can also be used to reformulate problem (GDP) as a tight MINLP problem of the form

\[
\text{min } Z = \sum_{i \in I} y_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
(\lambda_k)h_{ik}(x) \leq 0, \quad i \in D_k, \quad k \in SD
\]

\[
x \in \mathbb{R}^n, \quad \lambda_k \leq 0, \quad i \in D_k, \quad k \in SD
\]

\[
y_i \in \{0,1\}, \quad k \in SD
\]

in which \( x \) is a small tolerance to avoid numerical difficulties, and \( \lambda_k \) and \( y_i \) are binary variables that represent the Boolean variables \( Y_k \). All the algorithms that were discussed in the section on MINLP methods can be applied to solve this problem.

We consider next OA and GBD algorithms for solving problem (GDP). As described in Türkay and Grossmann (1996), for fixed values of the Boolean variables, \( Y_k = true \) and \( Y_k = false \) for \( i \neq i \), the corresponding MINLP subproblem is as follows:

\[
\text{min } Z = \sum_{i \in SD} c_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
\begin{align*}
h_{ik}(x) & \leq 0, \quad i \in D_k, \quad k \in SD \\
x \in \mathbb{R}^n, \quad c_i = 0, \quad k \in SD
\end{align*}
\]

Note that for every disjunction \( k \in SD \) only constraints corresponding to the Boolean variable \( Y_k \) is that is true are imposed, thus leading to a reduction in the size of the problem. Also, fixed charges \( y_{ik} \) are only applied to these terms. Assuming that \( K \) subproblems (NLPD) are solved in which sets of linearizations are generated for subsets of disjunction terms \( L_k = \{l \} \).

\[
\text{min } Z = \sum_{i \in SD} c_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
\begin{align*}
h_{ik}(x) & \leq 0, \quad i \in D_k, \quad k \in SD \\
x \in \mathbb{R}^n, \quad c_i = 0, \quad k \in SD
\end{align*}
\]

Note that for every disjunction \( k \in SD \) only constraints corresponding to the Boolean variable \( Y_k \) is that is true are imposed, thus leading to a reduction in the size of the problem. Also, fixed charges \( y_{ik} \) are only applied to these terms. Assuming that \( K \) subproblems (NLPD) are solved in which sets of linearizations are generated for subsets of disjunction terms \( L_k = \{l \} \).

\[
\text{min } Z = \sum_{i \in SD} c_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
\begin{align*}
h_{ik}(x) & \leq 0, \quad i \in D_k, \quad k \in SD \\
x \in \mathbb{R}^n, \quad c_i = 0, \quad k \in SD
\end{align*}
\]

Note that for every disjunction \( k \in SD \) only constraints corresponding to the Boolean variable \( Y_k \) is that is true are imposed, thus leading to a reduction in the size of the problem. Also, fixed charges \( y_{ik} \) are only applied to these terms. Assuming that \( K \) subproblems (NLPD) are solved in which sets of linearizations are generated for subsets of disjunction terms \( L_k = \{l \} \).

\[
\text{min } Z = \sum_{i \in SD} c_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
\begin{align*}
h_{ik}(x) & \leq 0, \quad i \in D_k, \quad k \in SD \\
x \in \mathbb{R}^n, \quad c_i = 0, \quad k \in SD
\end{align*}
\]

Note that for every disjunction \( k \in SD \) only constraints corresponding to the Boolean variable \( Y_k \) is that is true are imposed, thus leading to a reduction in the size of the problem. Also, fixed charges \( y_{ik} \) are only applied to these terms. Assuming that \( K \) subproblems (NLPD) are solved in which sets of linearizations are generated for subsets of disjunction terms \( L_k = \{l \} \).

\[
\text{min } Z = \sum_{i \in SD} c_i + f(x)
\]

s.t. \( g(x) \leq 0 \)

\[
\begin{align*}
h_{ik}(x) & \leq 0, \quad i \in D_k, \quad k \in SD \\
x \in \mathbb{R}^n, \quad c_i = 0, \quad k \in SD
\end{align*}
\]

Note that for every disjunction \( k \in SD \) only constraints corresponding to the Boolean variable \( Y_k \) is that is true are imposed, thus leading to a reduction in the size of the problem. Also, fixed charges \( y_{ik} \) are only applied to these terms. Assuming that \( K \) subproblems (NLPD) are solved in which sets of linearizations are generated for subsets of disjunction terms \( L_k = \{l \} \).
where the vector $s$ is partitioned into the variables $(s_{x_1}, s_{x_2}, \ldots, s_{x_n})$ for each disjunction $i$ according to the definition of the matrix $B^i$ (i.e. $s_i$ refers to non-zero rows of this matrix). The linearization set is given by $K^i = \{ | \text{true}, i = 1, \ldots, l \}$ that denotes the fact that only a subset of inequalities was enforced for a given subproblem $I$. It is interesting to note that the logic-based outer-approximation algorithm represents a generalization of the modeling/decomposition strategy Türkay and Grossmann (1989) for the synthesis of process flowsheets.

Türkay and Grossmann (1996) have also shown that while a logic-based generalized Benders method (Geoffrion, 1972) cannot be derived as in the case of the OA algorithm, one can exploit the property for MINLP problems that performing one Benders iteration (Türkay & Grossmann, 1996) on the MILP master problem of the OA algorithm, is equivalent to generating a generalized Benders cut. Therefore, a logic-based version of the generalized Benders method consists of performing one Benders iteration on the MILP master problem (MIPDF). It should also be noted that slacks can be introduced to (MGDP) and to (MIPDF) to reduce the effect of nonconvexities as in the augmented-penalty MILP problem (MIPDF). It should be noted that the logic-based outer-approximation algorithm, as in the case of the OA algorithm, can exploit the property for MINLP problems that performing one Benders iteration (Türkay & Grossmann, 1996) on the MILP master problem of the OA algorithm, is equivalent to generating a generalized Benders cut. Therefore, a logic-based version of the generalized Benders method consists of performing one Benders iteration on the MILP master problem (MIPDF).

3.4. Constraint programming

Constraint programming (Puget, 1994; Van Hentenryck, 1989) is an alternative approach to discrete and continuous problem solving that developed in the computer science and artificial intelligence communities. It has proved successful in several applications, particularly in scheduling and logistics. It does not use relaxations in the way that mathematical programming uses them, but it applies sophisticated methods of logical inference (primarily domain reduction and constraint-propagation) to reduce the domain of possible values a discrete or continuous variable may take. It also formulates models more succinctly due to its much richer modeling language.

The basic idea in constraint programming (CP) is to use compact languages for expressing optimization problems in terms of variables that are continuous, integer, and/or Boolean, and constraints that can be expressed in algebraic form (e.g. $h(x) \leq 0$), as disjunctions (e.g. $\{A_1 \leq b_1 \lor A_2 \leq b_2\}$), or as conditional logic statements (e.g. If $g(x) \leq 0$ then $r(x) \leq 0$). In addition, the language can support special implicit functions such as the all different ($x_1, x_2, \ldots, x_n$) constraint for assigning different values to the integer variables $x_1, x_2, \ldots, x_n$.

Constraint programming uses primarily an implicit enumeration coupled with constraint-propagation as the inference engine. At each node, constraint-propagation is used to reduce the domains of all the variables. The domain of a variable can be continuous, discrete, Boolean, etc. Constraint-propagation can result in empty domains in which case a node is fathomed. Branching is performed whenever the domain of a variable consists of more than one element (discrete and Boolean domains) or the bounds are not within a certain tolerance (continuous domains). CP was originally developed to solve feasibility problems. It has now been extended to solve optimization problems. This is achieved by solving a feasibility problem in which the objective function of the problem is rewritten as a constraint that forces it to be equal to a new variable. The domain of this new variable gives upper and lower bounds on the objective-function values. Whenever a feasible solution is obtained during the search, additional constraints that restrict the objective-function values are imposed throughout the search tree. The search terminates when all the nodes have been fathomed. The effectiveness of the CP methods depends primarily on the constraint-propagation algorithms that are used to reduce the domain of a variable.

The language of constraint programming consists of C++ procedures, although the recent trend has been to provide higher level languages such as OPL. Other commercial CP software packages include ILOG Solver (ILOG, 1999), CHIP (Dutchas et al., 1988), ECLiPSe (Wallace, Novello, & Schimpf, 1997), and Prolog IV. In these packages the tree in the implicit enumeration procedure is normally enumerated with a depth first search in which the lower bound is given by partial solutions, and the upper bound by the best feasible solution. At each of the nodes in the tree search, constraint-propagation is performed through domain reduction of the variables. This involves, for instance, the reduction of bounds in the case of continuous variables, and/or domains in the case of discrete variables. The former uses procedures for tightening bounds for linear and monotonic functions, while the latter is performed either by inference techniques, or by special procedures. Consider as an example the all different ($x_1, x_2, \ldots, x_n$) constraint, which requires that $x_1, x_2, \ldots, x_n$ take different values. When the domains of the $x_i$ are already partially reduced due to branching, etc. graph-theoretical methods applied to the all-different condition can reduce them further. Another example is the "edge-finding" method for reducing the domain of possible start times for job scheduling problems. Fig. 2 presents a simple example of such a method to resolve a disjunction about the relative processing of jobs $i$ and $j$ that are assigned to the same machine.

As another example consider multivalued discrete variables $Y_k$ that have a value indicating the unit to which input $k$ is assigned. Here it is sufficient to use a single constraint, all-different ($Y_1, \ldots, Y_k$), which requires each unit to be assigned at most one input.
Multivalued variables have the advantage that constraint-propagation algorithms (a form of logical inference) may be able to reduce their domains efficiently. The domain of a variable is the set of values it may assume. Suppose that in the assignment problem just mentioned, \( Y_1 \in 1, 2, 3, 5, Y_2 \in 1, 2, 3, 5, Y_3 \in 1, 2, 3, 4, 5 \). That is, input 1 must be assigned to unit 1, input 2 may be assigned to unit 2, 3 or 5, and so forth. It can be deduced from this information that input 2 cannot be assigned to unit 5. In fact, the domains can be reduced to \( Y_2 \in 2, 3, Y_3 \in 1, 2, Y_4 \in 5, \) and \( Y_5 \in 4 \). This reduces the amount of branching on each variable and even fixes two of the variables. This deduction can be accomplished rapidly using an algorithm related to maximum, cardinality bipartite matching (Régan, 1993).

Even though more constructs are available, not all of them have efficient constraint-propagation engines. Caution and sound judgment should be used when exploiting the advantage of having more constructs with which to model the problem. This is because the success of the CP solution approach is highly dependent on the propagation mechanism behind constraints written using these constructs.

Recently, a number of papers have compared the performance of CP- and MILP-based approaches for solving a number of different problems, for example, the modified generalized assignment problem (Darby-Dowman, Little, Mitra, & Zaffalon, 1997), the template design problem (Proll & Smith, 1998), the progressive party problem (Smith, Brailsford, Hubbard, & Williams, 1997), and the change problem (Heipcke, 1999a). These papers showed that MILP is very efficient when the relaxation is tight and the models have a structure that can be effectively exploited. CP works better for highly constrained discrete optimization problems where expressiveness of MILP is a major limitation.

Most of the recent attempts (Heipcke, 1999b; Rodosek, Wallace, & Hajian, 1999) to integrate CP and MILP use constraint-propagation along with linear programming in a single search tree to obtain bounds on the objective and to reduce the domains of the variables. In these approaches a complete CP model and at the least a corresponding partial MILP model are required. This is because CP is a richer modeling tool and not all CP constraints may be easily reformulated as MILP constraints. These approaches in some sense perform redundant computations because a constraint-propagation problem and a simplex problem are solved at every node. For some problems this may be justified because they are intractable for either of the two methods. Rodosek et al. (1999) presented a systematic approach for transforming a CP model into a corresponding MILP model. However, automatic translation from a CP model to an MILP model may result in a poor model involving numerous “big-M” constraints (poor LP relaxations). In this case, the advantage of performing “global reasoning” using LP relaxation is essentially lost. If automatic translation is not used, then the user has to model the problems for both approaches.

Hooker, Ottosson, Thorsteinsson, and Kim (1999) have argued that a new modeling paradigm may be required to perform efficient integration of MILP- and CP-based approaches. Bockmayr and Kasper (1998) did an interesting analysis of CP and MILP approaches, and presented a unifying framework, Branch and Infer, that can be used to develop various integration strategies. They divide constraints for both MILP and CP into two different categories, primitive and non-primitive. Primitive constraints are those for which there exists a polynomial-time solution algorithm, and non-primitive constraints are those for which this is not true. The interesting aspect about this classification is that some of the primitive constraints in CP are non-primitive in MILP and vice versa. They also discussed how non-primitive constraints can be used to infer primitive constraints and the use of symbolic constraints for MILPs. Raman and Grossmann (1993, 1994) earlier modeled discrete/continuous optimization problems with disjunctions and symbolic constraints in the form of logic propositions. This model, which they denoted as a generalized disjunctive program (GDP), can be converted all or in part into an MILP. They presented the idea of w-MIP representability, which is similar to the idea of primitive constraints. They showed that it is computationally efficient to transform w-MIP representable disjunctions into linear constraints, and proposed a hybrid branch-and-bound algorithm that handles the non-w-MIP representable disjunctions directly.

From the work that has been performed, it is not clear whether a general integration strategy will always perform better than either a CP or an MILP approach by itself. Jain and Grossmann (2001) have reported an encouraging decomposition method for single-stage scheduling problems, in which the assignments of jobs to machines is performed with an MILP model, and the sequencing with a CP model. This hybrid scheme was shown to yield reductions of several orders of magnitude compared with only MILP or CP models. Chandru and Hooker (1999) give an interesting operations research perspective on consistency methods and logical inference. Also, Hooker (2000) deals with the subject of MILP and CP integration in detail.
4. Large-scale optimization

The success of optimization strategies in process engineering constantly motivates the desire to formulate and solve larger problems over a wider set of domains. To discuss the advances and future needs for this topic it is important to first consider the characteristics that make optimization problems large.

1. Process engineering models evolve as additional detail is added to improve the accuracy of design and performance decisions. These models evolve in complexity from lumped parameter to spatially distributed representations. Also, designs that require steady state models can be improved by considering process performance under dynamic conditions. Moreover, specific constitutive behaviors can be generalized by evolving from empirical relations to more fundamental phenomenological relationships. For all of these cases, one needs to deal with model equations that evolve from algebraic to differential-algebraic to partial differential equations in multiple dimensions. More recently, these have been extended to deal with probabilistic models and statistical simulations as well.

2. Temporal scaling is common in both business and engineering models. These can take the form of dynamic multiperiod models that are executed over planning horizons, or process models consisting of differential-algebraic equations (DAEs). For optimization of time dependent models, stability of the time evolution of the equations can be an additional concern. Moreover, when these problems are applied in on-line applications, computations are resource limited and efficient optimization algorithms are essential. Finally, in addition to their large size, time dependent models have a forward dynamic structure that should be exploited. This can be done through decomposition strategies but these may also require storage of the entire state history.

3. Scaling in size is common in the growth of homogeneous partial differential equation (PDE) models. Due to the need for accurate spatial discretizations for challenging applications in computational mechanics, fluid flow or reacting systems, PDE formulations are often orders of magnitude larger than typical process models. After choosing an appropriate meshing strategy and assembling the relevant finite element (or finite volume or finite difference) equations, systems of discretized PDEs may consist of $n = O(10^6)$ state variables and equations for typical production simulations. Leading edge simulations will have $n = O(10^9)$ state variables and equations. A similar scaling occurs in the discretization of probability distributions in considering models with uncertainty. Again, it is not uncommon to multiply the size of nominal process models by several orders of magnitude to yield problems with $10^{6-9}$ variables and equations.

4. Integration of models leads to the creation of very large models for process optimization. A typical instance of this occurs in flowsheet optimization, which requires the integration of heterogeneous process models. More recently, this is faced with multi-environment models that may also run on different processors with vastly different solution strategies and calculation times. In addition to flowsheets, they can include PDE models (such as computational fluid dynamics (CFD)) computational chemistry and scheduling models.

These factors lead to the need to extend and reinvent optimization algorithms. In particular, the following areas need to be addressed for large-scale algorithms.

Problem size: for large-scale problems, many of the conceptual algorithms for NLP and MINLP do not change. On the other hand, additional care is needed to deal with the scale-up of subproblems, particularly with the solution of linear systems. This scale-up is influenced by the size of the process model as well as the number of variables available for optimization (degrees of freedom).

Growth of combinatorics: the combinatorics of both NLP and MINLP algorithms are affected significantly by increases in problem size. In NLPs this is usually observed in finding optimal active constraint sets. In MINLP and global optimization algorithms, this is observed in the exponential growth of the branch and bound tree and the need to enumerate many more alternatives.

Effects of problem structure: with increases in size, it is imperative to exploit specific problem structures. These are usually done through the application of decomposition strategies that lead to the repeated solution of smaller subproblems. There is often an opportunity (e.g. with convex problems) to generate bounds on the optimal solution in order to accelerate convergence and provide termination criteria.

In the remainder of this section we present concepts for large-scale algorithms for NLP and MINLP optimization. For continuous variable optimization, we will primarily emphasize the first two aspects, as they also have an influence on MINLPs. We then discuss decomposition strategies primarily for MINLP problems.

4.1. Large-scale continuous variable optimization

Optimal design problems that incorporate large-scale (especially DAE or PDE) models generally have decision variables that consist of $O(1)$ decisions for physically-based designs or $O(10^3)$ to $O(10^6)$ variables for finely-parameterized shape optimization. Decision variables for optimal control problems scale with up to $O(n)$ decision variables. Finally, inverse problems that incorporate DAE or PDE models have decisions that include $O(1)$ decisions for state problems or PDEs with homogeneous material parameters, or they may have $O(n)$ decisions for heterogeneous materials or time dependent parameters.
Large-scale constrained nonlinear programs are typically solved using variants of Newton’s method applied to the optimality conditions of the NLP. This Newton-based approach, extended to deal with inequality constraints, was discussed in Biegler and Grossmann (2004) as successive quadratic programming (SQP). Variants that incorporate the elimination of the state variables and linearized equations are known as reduced space SQP or rSQP methods. Linear systems that arise in NLP methods have evolved over the past 20 years from reliable dense matrix solvers (such as QR factorizations in NPSOL, Gill, Murray, & Saunders, 1987) to direct sparse factorizations in codes such as SOCS (Betts, 2001), IPOPT (Biegler & Wächter, 2003), LOQO (Vanderbei & Shanno, 1997) and KNITRO (Byrd, Hribar, & Nocedal, 1997). However, with consideration of larger, structured problems we often arrive at a practical performance limit for these general purpose solvers (usually with $n < 10^6$ variables) and specialized linear system solvers need to be considered differently in future implementations.

In addition to developing large-scale NLP algorithms, there are a number of specific decomposition issues related to handling increasingly large process optimization problems with algebraic models. These could include the following.

- **Decomposition of large nonlinear systems** for problems that exhaust the limitations of current tools. For instance, in process engineering one needs to consider the handling of (multiple) RTO models for refinery wide optimization and multiple plant design models, which may run on different platforms, exploit different structure and could be developed under different modeling environments.
- **Coupling of multiple design models** where mass and energy balance models need to be coupled with detailed equipment design and costing. Also included are important process aspects such as operability, controllability and safety, which can be hard to formulate for the optimization problem (often due to discontinuities in their calculation).
- **Multiperiod models** reflect life cycle considerations for the process and allow the design to consider different operating scenarios, consideration of uncertainty and evolution of the process over time. These models lead to very large bordered block diagonal (or almost block diagonal) KKT matrices, where each diagonal block could represent the KKT matrix of any of the process optimization problems discussed thus far.

Often these problems can be dealt with through specialized linear decompositions that exploit the structure of the overall KKT matrix. Biegler, Schmid, and Ternet (1997) describes a tailored rSQP approach that is based on this concept and is useful for problems with few decision variables. Finally, for large-scale DAE and PDE models there are a number of structural and model specific considerations that need to be addressed, as presented next.

### 4.1.1. Large-scale DAE optimization

As described in Biegler and Grossmann (2004), the DAE optimization problem can be converted into an NLP by approximating state and control profiles by a family of polynomials on finite elements $(t_0 < t_1 < \cdots < t_m = t_f)$. Here, one can use either monomial or Lagrange basis representations for the differential profiles. Continuity of the differential profiles is enforced by with additional equations. Moreover, we use Radau collocation points because they allow us to set constraints easily at the end of each element and to stabilize the system more efficiently if high index DAEs are present. In addition, the control and algebraic profiles are approximated using a Lagrange polynomial basis representation. As a result, the differential variables are required to be continuous throughout the time horizon, while the control and algebraic variables are allowed to have discontinuities at the boundaries of the elements. It should be mentioned that the bounds on differential variables are enforced directly at element boundaries; however, they can be enforced at all collocation points by writing appropriate point constraints.

The resulting NLP problem can be solved using an rSQP or reduced space barrier method. This method is efficient for solving DAE optimization problems, especially when the dimension of the state variables is much larger than that of the control variables ($n \gg n - m$). The efficiency of the solution procedure is also improved by performing matrix factorizations in each element. This allows us to preserve and exploit the structure of the problem and to detect ill-conditioning due to unstable modes in the DAE system. To develop a stable decomposition we reconsider the rSQP approach from Biegler and Grossmann (2004). Using the KKT conditions of the QP (or barrier subproblem) we have:

$$
\begin{bmatrix}
Q & A \\
A^T & 0
\end{bmatrix}
\begin{bmatrix}
d \\
\lambda
\end{bmatrix}
= 
\begin{bmatrix}
a \\
c
\end{bmatrix} \quad \text{(KKT)}
$$

where $a = \nabla f$, $A = \nabla c$, $Q = \nabla^2 L(x, \lambda)$ (the second derivative matrix of the Lagrange function with respect to $x$, possibly with barrier terms) or its approximation (the iteration $k$ subscript is suppressed for brevity). We now consider the exploitation of the structure of the QP. Here the $n \times n$ matrix $Q$ can either be considered in a large, sparse form or approximated in the reduced space. To solve the QP in the reduced space, let:

$$A^T = [y^T; z^T], \quad A = [c\mid N]$$

and select an $n \times n$ nonsingular matrix $[YZ]$, where $A^T Z = 0$ and $Z$ and $Y$ form null and range space bases for the linearized equality constraints. We similarly partition the search direction into range ($p_Y$) and null space ($p_Z$) components: $d = p_Y + p_Z$.

The former component deals with the dependent variables of the model, while the latter component determines the
search direction for decision variables of the optimization problem. With this representation we can write:

\[ [Y|Z] = \begin{bmatrix} I & -C^{-1}N \end{bmatrix}, \quad [Y|Z]^{-1} = \begin{bmatrix} I & C^{-1}N \\ 0 & I \end{bmatrix} \]

Defining \( M = \text{diag}[Y|Z], I \) and pre- and post-multiplying the linear system from the (4) the equivalent system is given as:

\[
\begin{bmatrix}
Y^T Q Y & Y^T Q Z & Y^T A \\
Z^T Q Y & Z^T Q Z & 0 \\
A^T Y & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_Y \\
p_Z \\
lambda
\end{bmatrix}
= 
\begin{bmatrix}
Y^T a \\
Z^T a \\
c
\end{bmatrix}
\]

(RKKT)

Because \( p_Y \) and \( p_Z \) are calculated from the last two rows, we can ignore the \( Y^T Q Y \) and \( Y^T Q Z \) terms and approximate \( Z^T Q Y \) by \( w_k \) to get:

\[
\begin{bmatrix}
0 & 0 & Y^T A \\
0 & Z^T Q Z & 0 \\
A^T Y & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_Y \\
p_Z \\
lambda
\end{bmatrix}
= 
\begin{bmatrix}
Y^T a \\
Z^T a + w_k \\
c
\end{bmatrix}
\]

(SKKT)

Note that neglecting these terms does not affect the search direction, only the multiplier estimates \( \lambda \). And as the search direction converges to zero, \( \lambda \) also converges to its correct value. Also we see that these assumptions lead to a simple block diagonal decomposition for the rSQP strategy.

4.2. Elemental decomposition

The partitioning in (8) allows us to perform efficient decomposition of \( A \) described in Cervantes and Biegler (2000). First, consider the Jacobian of the discretized system of equations.

\[
A^T = \begin{bmatrix}
x_i^1 & x_i^2 & x_i^3 & \ldots & x_i^{n+1} \\
x_i^1 & x_i^2 & x_i^3 & \ldots & x_i^{n+1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
x_i^1 & x_i^2 & x_i^3 & \ldots & x_i^{n+1} \\
x_i^1 & x_i^2 & x_i^3 & \ldots & x_i^{n+1}
\end{bmatrix}
\]

where \( I \) represents the identity matrix of appropriate size, and \( Df \) is a matrix containing the coefficients of the continuity equations of the \( r \) th element. \( Z^r_c, Z^r_z, V^r_c, U^r_c \) and \( P^r_c \) represent the Jacobian of the collocation equations with respect to the differential states and their time derivatives \( z_c \), \( dz_c/dt \), the algebraic states \( y_c^a \), control variables \( u_c^a \), and parameters \( p \), at collocation point \( q \) and element \( i \). \( Z^r_c, Z^r_z, V^r_c, U^r_c \) and \( P^r_c \) correspond to the Jacobian of any additional design constraints. The factorization of this matrix is performed over smaller matrices, each one representing a finite element. In most cases these matrices have the same sparsity structure and consequently allow re-use of the sparse matrix pivot sequence.

To explore this decomposition, consider the rows and columns of \( A^T \), corresponding to the finite element \( i \):

\[
A^T = \begin{bmatrix}
Z_i^{-1} & D_{i}^z & 0 & 0 & P_i \\
Z_i^{-1} & D_{i}^z & 0 & 0 & P_i \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
Z_i^{-1} & D_{i}^z & 0 & 0 & P_i \\
I & D^c & 0 & -I & 0
\end{bmatrix}
\]

Here, it is assumed that the additional point constraints (the last row in the matrix) can be separated by elements. If no parameters \( p \) are present, the decomposition of this matrix can be performed directly, as all the variables can be eliminated locally. In the case that a parameter is present, the last column of \( A^T \) which corresponds to the parameters, will be coupled to the entire system. In this case, we create separate dummy parameters for each finite element.

For rSQP the partitioning of \( A^T = [C|N] \) for the \( m \) dependent and \( n \)-th decision variables, respectively, is performed by applying an LU factorization with partial pivoting on each rectangular system \( A' \). Following Cervantes and Biegler (2000) this LU factorization yields a dichotomous system in each element. Thus if an unstable mode is present in the DAE, \( A' \) is required to be partitioned so that, in effect, the end conditions of any increasing mode are fixed or become decision variables. In the partitioning, if a differential variable \( z_c \) has an increasing mode, \( dz_c/dt, \text{null} \) would be identified and shifted from a column in \( C \) to a column in \( N \). Correspondingly, a column corresponding to a control variable or a parameter could be shifted from \( N \) to \( C \). By considering the variables that span the columns of the null
space to be fixed, the decomposition approach then becomes equivalent to solving a discretized, linear boundary value problem that remains stable.

Also, as shown in Cervantes and Biegler (2000), the whole matrix $A$ need not be stored, and we perform a forward elimination in order to obtain the step for the dependent variables $p_i$. After the basis is selected, we can represent the overall matrix $A$ with the following structure and partition:

$$A^T = \begin{bmatrix}
I & 0 & N^1 \\
T^1 & C^1 & \bar{N}^1 \\
I & T^1 & C^2 - I & N^2 \\
I & \bar{N}^2 \\
\vdots & \vdots & \ddots & \ddots \\
I & T^N & C^N & \bar{N}^N \\
\end{bmatrix} = [C][N]$$

The corresponding right-hand sides are:

$$e^T = [c_0 \, c_1 \, c_2 \, \cdots \, c_N]$$

where $(\cdot)^T$ indicates terms from the continuity equations that link the finite elements. By pre-multiplying $T_i$, $C_i$, and $N_i$ by $C^{-1}$ in each element, we can develop a forward decomposition strategy that allows us to calculate $C^{-1}N$ and $C^{-1}c$. It should be noted that in this implementation only the sparse LU factors of $C_i$ need be stored and reused.

As described in Biegler, Cervantes, and Wächter (2002), the elemental decomposition strategy is very efficient and has been applied to problems up to two million variables. Coupled with this decomposition is also the placement of finite elements in order to maintain accuracy of the discretized DAE model and to determine the optimal breakpoint locations in the optimal control profile. In Biegler et al. (2002) we develop formulations that facilitate movement of finite elements in order to satisfy accuracy and consistency properties. While several challenging optimal control problems were solved accurately, this moving finite element strategy is still at a heuristic level and a more rigorous analysis is needed to put this strategy on a more fundamental basis. Finally, as the number of discretized variables increases an interior point or barrier algorithm needs to be applied to handle variable bounds. Research challenges for barrier algorithms are discussed below after we consider large-scale PDE systems.

### 4.2.1. Large-scale PDE optimization

In addition to the challenges faced with the optimization of DAE systems, large-scale optimization of PDE systems generally requires the application of iterative linear solvers, as the KKT matrix is too large to be factored. Consequently, iterative linear solvers (usually, Preconditioned Newton-Krylov methods) also need to be considered in the application of NLP algorithms. To lend further insight into this challenge, we consider four levels of implementation of NLP algorithms with PDE solvers.

At the most basic level, we consider a black-box or nested analysis and design (NAND) implementation. This interface requires very little interaction of the NLP solver with the PDE model. Gradients for the NI-P solver (e.g. SQP or rSQP) are obtained by finite difference, a nonlinear elimination of the state variables is assumed, and usually only few decision variables can be considered. On the other hand, the NAND implementation can suffer from repeated and time-consuming solution of the PDE model and intermediate convergence failures of the model.

One can also consider partially open implementations that are tailored to the PDE modeling system using rSQP and (SKKT) above. Here two variations can be distinguished by whether accurate Jacobian elements of the PDE model can be made available to the NLP algorithm.

- With the **direct tailored approach** reduced gradients and search directions for rSQP are calculated directly using Newton steps from the PDE modeling system ($-C^{-1}e$) and the "sensitivity" of Newton steps to the decision variables ($-C^{-1}N$). Nevertheless, these steps require the iterative solution of $(n-m)$ preconditioned linear systems. This approach is favored only when $n-m$ is small.
- On the other hand, the **adjoint tailored approach** calculates reduced gradients and search directions for rSQP by using the transpose of the model Jacobian to form $-C^T a$. This approach requires far fewer iterative linear solutions than the direct approach and also provides multiplier estimates for the NLP solver.

However, to obtain the best performance, a fully open method should be considered. This implementation allows access to the complete discretization of the PDE model and accurate first and second derivatives that make up the linear KKT system in the space of all of the state and decision variables. To solve this full KKT system, an approximate elimination of states and adjoints is applied as a preconditioner. Therefore, because the iterative solution of the linearized PDE equations is now part of the NLP algorithm, this approach leads to a very fast simultaneous analysis and design (SAND) optimization strategy.

To demonstrate this approach, Biros and Ghattas (2001) developed the Lagrange–Newton–Krylov–Schur (LNKS)
We exploit the structure of the problem by first partitioning it into states and decisions, as opposed to subdomain and interface spaces. A basic component of this method is the Krylov–Schur solver and its preconditioner, which accelerates the computations of the Newton step for (KKT). Biros and Ghattas (2001) introduce and analyze these preconditioner(s) and also examine the parallelizability and scalability of the LNKS algorithm. Here we present a summary of this method and discuss its effectiveness on a large-scale optimization problem. Consider the NLP without bound constraints:

\[
\min \ f(x) \\
\text{s.t.} \ c(x) = 0
\]

We exploit the structure of the problem by first partitioning it into state variables \( y \in \mathbb{R}^n \) and decision variables \( z \in \mathbb{R}^{m-n} \). The KKT conditions are used to convert the constrained optimization to a system of nonlinear equations and in LNKS a Newton method is applied. To compute the Newton step we solve the system (KKT) above using an appropriate Newton–Krylov method. At the core of the algorithm lies the preconditioner, which is an inexact factorization of the KKT matrix:

\[
\begin{bmatrix}
Q_y & 0 \\
Q_z \Sigma^{-1} & I
\end{bmatrix}
\begin{bmatrix}
d_z \\
C_y N y
\end{bmatrix}
= 
\begin{bmatrix}
a_x \\
\lambda
\end{bmatrix}
\]

(PKKT)

There are a number of ways to solve (PKKT). For instance, reduced space SQP (rSQP) is equivalent to a block-row elimination: given \( d_z \), solve the last block of equations for \( d_y \), then solve the first to find \( \lambda \), and finally solve the middle one for \( d_x \), the search direction for the decision variables. Therefore rSQP can be written as a particular block-LU factorization of the KKT matrix:

\[
\begin{bmatrix}
Q_y \Sigma^{-1} C_y^{-1} & 0 & I \\
Q_z \Sigma^{-1} & I & N^T C_y^{-T} \\
I & 0 & 0
\end{bmatrix}
\begin{bmatrix}
C & N & 0 \\
0 & Q_x & 0 \\
0 & Q_z - Q_x \Sigma^{-1} N & C^T
\end{bmatrix}
\]

(LU-KKT)

Note that these factors are permutable to block triangular (LU) form and that \( Q_x \) is the reduced Hessian with respect to \( x_k \), given by:

\[
Q_x = Q_x + S^T Q_z \Sigma^{-1} S - S^T Q_z S
\]

where \( S = C^{-1} N \). Based on the Schur-type factorization we use the following preconditioner for (PKKT):

\[
\begin{bmatrix}
0 & 0 & I \\
0 & I & N^T C^{-T} \\
I & 0 & 0
\end{bmatrix}
\begin{bmatrix}
C & N & 0 \\
0 & Q_x & 0 \\
0 & 0 & C^T
\end{bmatrix}
\]

The key components of this preconditioner are \( C^{-1} \) and \( Q_x \), the preconditioners for the solution of the PDE model and the reduced space Hessian, respectively. In particular, a natural choice for \( Q_x \) is a BFGS-like method, which is commonly used in rSQP methods. In addition, the Newton–Krylov solver MINRES was used in a parallel environment to solve (PKKT). For a 3-D finite element model of the Navier–Stokes equations (with over 615,000 state variables, 8900 decision variables) Biros and Ghattas (2001) obtained optimal solutions in only 4.1 wallclock hours on Cray T3E. This preconditioned method was an order of magnitude faster than rSQP methods and showed excellent scalability of the method on parallel processors. More recently, Ghattas and coworkers have solved finite element models with over three million state variables.

4.2.2. Inequality constraints

Many optimal design problems with DAE and PDE models have bounds on decision variables and also inequalities on state variables at points in temporal and spatial domains; this can lead to O(n) inequalities, which are often nonlinear. Examples of these include avoiding material failure, reducing vorticity in fluid flow, and limits on temperature, concentration and operation. In classical SQP and rSQP methods, inequalities are handled through the active set solution of the quadratic programming subproblem. However, determination of the active set of inequalities is an NP-hard problem and therefore may be extremely expensive for large systems. Recently, large-scale interior point (or barrier) NLP algorithms have been considered to address these challenges (see Biegler et al., 2002; Cervantes, Wächter, Tütüncü, & Biegler, 2000; Vanderbei & Shanno, 1997).

As discussed in Biegler and Grossmann (2004), inequality constraints are lumped into a barrier term and the combinatorial problem is transformed to solving a set of nonlinear equations. Moreover, for large-scale problems, the barrier approach also has an extremely important structural feature. For the following simplified NLP:

\[
\min \ f(x) \\
\text{s.t.} \ c(x) = 0
\]

application of the barrier method in Biegler and Grossmann (2004) leads to the following KKT matrix for the Newton algorithm:

\[
\begin{bmatrix}
Q + \Sigma & A^T \\
A^T & 0
\end{bmatrix}
\begin{bmatrix}
d \\
\lambda
\end{bmatrix}
= 
\begin{bmatrix}
a - X^{-1} \mu e \\
\Sigma \mu e
\end{bmatrix}
\]

(IPKKT)

where \( X = \text{diag}(\mu) \), \( \Sigma = \{1 \, 1 \cdots 1\} \), \( v = X^{-1} \mu e \), and \( \Sigma = X^{-1} V \), a diagonal matrix. Note that
the addition of barrier terms leads to only a small change in the KKT matrix without bound constraints, and linear decomposition procedures for the KKT matrix (e.g. range and null space, LNKS or specialized multiperiod decompositions) remain unchanged. As a result, barrier approaches allow for an efficient and straightforward incorporation of bound constraints, even for very large problems.

To assess the performance of barrier approaches for large-scale problems, Mittelmann (2002) and Wächter (2002) present an extensive comparison of the interior point algorithm IPOPT. Both full and reduced space versions have been implemented for this algorithm and efficient performance of this algorithm has been promoted through the incorporation of exact second order information. In the full and reduced space implementations of IPOPT, these are provided directly through automatic differentiation or through differenced preconditioned conjugate gradient (PCG) methods, respectively.

Nevertheless, there are still some open questions with this approach and a number of improvements need to be considered, including the following.

- **Negative curvature**: because second derivatives can be incorporated directly or can be differenced along PCG steps, improved provisions must be made to deal with negative eigenvalues in the reduced Hessian matrix. This is a difficult problem with line search methods.
- **Preconditioner for CG iterations**: in Cervantes et al. (2000), a preconditioner that separates the two Hessian terms, $P_k = (Z^T_k Q_k Z_k + Z^T_k \Sigma_k Z_k)^{-1}$ leads to a significant reduction in CG iterations. Nevertheless, this requires the construction of $Z^T_k \Sigma_k Z_k$ which can become costly for large problems.
- **Globalization steps**: IPOPT contains an efficient filter line search procedure (Wächter & Biegler, 2002) that guarantees global and superlinear convergence. However, this may require a constraint restoration phase that needs still needs to be adapted to large-scale problems and to the particular Newton subproblems considered in IPOPT.

### 4.2.3. Mathematical programs with equilibrium constraints (MPECs)

Because barrier solvers, like IPOPT, are effective in dealing with the combinatorics of active set selection, they are also very useful to handle the combinatorics of nested optimization problems. These can be generalized as mathematical programs with equilibrium constraints or (MPECs). MPECs represent an exciting new field in mathematical programming. While applications of MPECs have long been recognized in game theory, transportation planning, economics and engineering design, little work has been done in developing efficient algorithms for their solution. A broad survey of these applications can be found in Luo, Pang, and Ralph (1996). In process engineering, these problems stem from bilevel and multilevel optimization problems as well as optimization of hybrid (discrete and continuous) systems. Included in this class are optimization problems with phase equilibrium constraints and in equilibrium stage processes. A general form for the MPEC problem is given by:

$$
\begin{align*}
\min & \quad f(x, y, w) \\
\text{s.t.} & \quad c(x, y, w) = 0 \\
& \quad x, y, w \geq 0 \\
& \quad y_i w_i = 0, \quad i = 1, \ldots, n_w.
\end{align*}
$$

where we now consider an augmented set of variables $(x, y, w)$. The complementarity conditions in (MPEC) are difficult to handle even when present as part of a square system of equations. This has motivated the development of a number of specialized algorithms based on smoothed reformulation of complementarity constraints, interior point and pivoting techniques among others. A survey of MPEC algorithms can be found in the article by Harker and Pang (1990). However, obtaining solutions to complementarity constrained nonlinear programs pose a bigger challenge. MPECs fail to satisfy constraint qualifications, such as linear independence of constraint gradients (LICQ) and Mangasarian-Fromovitz constraint qualification (MFCQ), commonly assumed to hold for nonlinear programs (NLPs). Furthermore the feasible region of the MPEC lacks closedness, convexity and could even be disjoint (Luo et al., 1996). These features can render (MPEC) intractable with a nonlinear programming strategy, as even conditions for existence of solutions for general MPECs have not been fully established. In spite of these deficiencies, complementarity formulations have seen widespread use in numerous applications, and this motivates the study of their properties and development of numerical algorithms.

The use of complementarity constraints in modeling process engineering problems is a relatively new and exciting field of research. In particular, chemical processes involve systems that are governed by chemical equilibrium. These lead naturally to bilevel optimization problems (Clark & Westerberg, 1990; Sahin & Ciric, 1998) which can be re-formulated as MPECs by replacing the inner minimization problem with its stationary conditions. A class of parametric feasibility problems has have important applications in process engineering can be posed as bilevel (Floudas & Grossmann, 1987) and tri-level optimization problems (Swaney & Grossmann, 1985). The bilevel program was solved by posing it as a mixed-integer nonlinear program (MINLY) (Floudas & Grossmann, 1987).

A number of algorithms for the solution of MPECs are discussed in Luo et al. (1996). The penalty interior point algorithm (PIPA) is based on an interior point algorithm and aims to solve the stationary conditions for MPECs derived in the same work. These authors also propose a piecewise sequential quadratic programming (PSQP) algorithm which employs a strategy similar in spirit to the active set strategy to identify the zero variables involved in the complementarity constraints. Jiang and Ralph (2000) pose mathematical...
programs with nonlinear complementarity constraints as a nonsmooth program by reformulating the constraints, and a class of functions parameterized by a scalar is used to obtain a smooth nonlinear program. Two globally convergent algorithms based on sequential quadratic programming (SQP) are applied as exact penalty methods for nonlinear programs. In a parallel development, researchers (Andreani & Martínez, 2001; Anitescu, 2001; Leyffer, 2000) have investigated the applicability of NLP algorithms for the solution of MPECs. Leyffer (2000) demonstrates encouraging results that support general NLP algorithms as compared to specialized MPEC algorithms. Anitescu (2001) proves convergence behavior of a certain class of NLP algorithms when applied to MPECs. Recently, Raghunathan and Biegler (2003) extended the IPOPT algorithm to solve MPECs. In particular, they reformulate and relax the (MPC) problem by considering the following barrier subproblem:

\[ \min f(x, y, w) - \mu \left( \sum_i x_i + \sum_i \ln y_i \right) \]
\[ \text{s.t. } c(x, y, u) = 0 \]
\[ y_i w_i + s_i = \delta \mu, \quad i = 1, \ldots, n_w \]

(MPEC-B)

Note that here the barrier parameter \( \mu \) operates both on the barrier terms as well as the complementarity conditions. A related relaxation with \( y_i^w + s_i = \delta \mu_l \) This approach has a number of advantages over the direct solution of (MPEC). First, for positive values of \( \mu \) the subproblem satisfies LICQ and the Newton steps for IPOPT are well defined. Therefore, the algorithm converges arbitrarily closely to the solution of (MPEC), if it exists. Raghunathan and Biegler (2003) considered a number of properties for the IPOPT-based approach and performed an extensive numerical comparison on the MacPEC problem set from Leyffer (2000) (with over 120 cases). Moreover, they applied this approach to the optimal design of distillation columns that may have dry or vaporless trays. More recently, this approach was also applied to large-scale dynamic MPEC problems that include the startup of batch distillation columns (Raghunathan, Wächter, & Biegler, 2002).

4.3. MINLP decomposition

In the remainder of this section we consider decomposition strategies that can be applied to the broader class of MINLP problems in the context of multiperiod design, planning and scheduling problems. When choosing a decomposition method it is important to consider how to exploit the structure of the model most efficiently and also to choose a degree of decomposition that allows solution in reasonable time while still finding an optimal or near-optimal solution. Several decomposition schemes have been proposed in the literature. Bender decomposition (Benders, 1962) and dual decomposition or Lagrangean relaxation (see e.g. Fisher, 1981) exploit the primal and dual structures of the model, respectively. Cross-decomposition (see e.g. Van Roy, 1983) exploits both the primal and dual structures and is applicable to models where both the primal and dual subproblems are easy to solve. Bilevel decomposition (e.g. Iyer & Grossmann, 1998) exploits the structure of models that include different hierarchical levels, such as the hierarchy from design, to planning, to scheduling. Roszczenψki (1997) gives a comprehensive review on decomposition methods for stochastic problems, including cutting plane methods, augmented Lagrangean decomposition, splitting methods and nested decomposition. We discuss only Lagrangean relaxation and bilevel decomposition in further detail below, since an in depth discussion of all decomposition methods is beyond the scope of this paper.

4.3.1. Bilevel decomposition

One approach to exploit the hierarchical structure of combined design, planning and/or scheduling models is to decompose the model into an upper level problem at the higher hierarchical level, and a lower level problem at a lower hierarchical level. Iyer and Grossmann (1998) proposed such a bilevel decomposition algorithm for an MILP design and planning problem, where the upper level involves mainly design decisions while the lower level involves mainly planning decisions. Van den Heever and Grossmann (2000) extended this approach to MINLPs through the use of GDP. Consider an original model (P) where superscript "d" denotes design variables and superscript "p" denotes planning variables.

\[ \min f(x^d, y^d, x^p, y^p) \]
\[ \text{s.t. } h(x^d, y^d, x^p, y^p) \leq 0 \]
\[ x \in \mathbb{N}, \ y \in [0, 1] \]

To derive the upper level design problem (DP), all the discrete planning variables are relaxed. This results in a much smaller number of nodes in the branch and bound search facilitating a faster solution. Also, some of the constraints and/or variables may be aggregated at this level, indicated by \( \Lambda \).

\[ \min f(x^d, y^d, x^p, y^p) \]
\[ \text{s.t. } Ah(x^d, y^d, x^p, y^p) \leq 0 \]
\[ 0 \leq y^p \leq 1 \]
\[ x \in \mathbb{N}, \ y^d \in \mathbb{N}, \ y^d \in [0, 1] \]

After (DP) is solved, the discrete design variables are fixed (indicated by the bar on \( d \)) and the lower level planning problem (PP) is solved for the fixed design.

\[ \min f(x^d, y^d, x^p, y^p) \]
\[ \text{s.t. } h(x^d, y^d, x^p, y^p) \leq 0 \]
\[ x \in \mathbb{N}, \ y^d \in [0, 1] \]

Subproblems (DP) and (PP) are solved iteratively and design and integer cuts are added at each iteration to ensure an opti-
normal solution. Note that even though both (DP) and (PP) are in a reduced space, both consider the design and planning model as a whole. A further benefit of this approach is that it significantly reduces the computational effort compared to solving the combined problem as a whole, while still guaranteeing the optimal solution to the original combined model in the convex case. Papageorgiou and Pantelides (1996) proposed a similar decomposition approach for combined campaign planning and scheduling of multipurpose batch/semicontinuous plants. In their work, the upper level problem concerns mainly campaign planning decisions while the scheduling decisions are aggregated, and the lower level problem is solved with some of the campaign planning variables fixed. Again both levels consider the problem as a whole. In the experience of the authors, the bilevel decomposition approach works particularly well for large-scale industrial applications over a long time horizon, especially when combined with the aggregation of time periods.

4.3.2. Lagrangean relaxation

This is an approach that is often applied to models with a block diagonal structure. In such models, distinct blocks of variables and constraints can be identified that are linked with a few “linking” constraints and variable (see Fig. 3). Some applications include scenario decomposition for planning under uncertainty (Carne & Schultz, 1999), unit commitment in power plants (Nowak & Römisch, 1998), midterm production planning (Gupta & Maranas, 1999), and oilfield investment planning (Van den Heever, Grossmann, Vasantharajan, & Edwards, 2001) and combined transportation and scheduling (Equi, Gallo, Marziale, & Weintraub, 1999), to name but a few.

Consider a model (L) that has been partitioned into blocks of constraints $p = 1 \ldots P$, where the blocks are linked by a constraint set $h$:

$$\min \sum_{p} f_{p}(x_{p})$$

s.t. $g_{p}(x_{p}) \leq 0, \forall p$

$h(x_{1}, \ldots, x_{P}) \leq b$

$x \in X$

The basic idea behind Lagrangean relaxation as applied to the decomposition of block diagonal structures, is to dualize the linking constraint set, $h$, by removing it and replacing it with a penalty in the objective function involving the associated Lagrangean multipliers, $\lambda$, as seen in model (LR):

$$\min \sum_{p} f_{p}(x_{p}) + \lambda(h(x_{1}, \ldots, x_{P}) - b)$$

s.t. $g_{p}(x_{p}) \leq 0, \forall p$

$x \in X$

Model (LR) is now decomposable into $p$ subproblems and, for any choice of $\lambda$, also yields a lower bound to the optimal solution of (L) if the constraints are convex. The case where variables link the blocks can be dealt with by introducing duplicates for each linking variable, setting the duplicates equal, and dualizing this equality constraint. This is referred to as Lagrangean decomposition (Guignard & Kim, 1987).

Obtaining the tightest lower bound to (L) requires the solution of the Lagrangean dual problem (LD):

$$\max \sum_{p} f_{p}(x_{p}) + \lambda(h(x_{1}, \ldots, x_{P}) - b)$$

s.t. $g_{p}(x_{p}) \leq 0, \forall p$

$x \in X$

If all the constraints are convex and all the variables are continuous, the optimum of (LD) will equal the optimum of (L). However, a duality gap might exist in the presence of integer variables or other non-convexities, which means that the optimal solution to the dual problem will be strictly less than the true optimum of (L). Bazaraa, Sherali, and Shetty (1994) and Guignard (1995) give comprehensive graphical interpretations of the duality gap in the case of integer variables and non-convex constraints, respectively. Solving (LD) can be difficult to implement and time-consuming, although Fisher (1981) reports on some algorithms for this purpose.

On the other hand, solving the dual to optimality is therefore often circumvented by using an iterative heuristic approach where (LD) is solved to generate lower bounds to (L) and a heuristic method is used to generate feasible solutions to (L) which are also upper bounds. $\lambda$ is updated at each iteration with some updating rule, for example, a subgradient method (see e.g. Fisher, 1981). This decomposition method reduces the computational effort by solving several subproblems instead of the original problem, and the associated algorithms lend themselves to parallelization to reduce the computational effort even more. For a thorough background on the application of Lagrangean relaxation, we refer the reader to Guignard (1995) and Fisher (1981, 1985).

4.3.3. Aggregation

For some models, decomposition alone is not enough to obtain a good solution in reasonable time, and some form of aggregation is required to further reduce the model size. Rogers, Plante, Wong, and Evans (1991) give a good review on the use of aggregation/disaggregation in optimization. These authors define the major components of this framework, namely, aggregation analysis, disaggregation analysis.
and error analysis. The first component involves determining which elements of the model to combine into a single element and how to define the single element, while the second component conversely involves deriving a more refined model from the aggregate one. Error analysis determines the error introduced by aggregation and disaggregation. These three components can be addressed sequentially or iteratively to reduce the computational effort of solving the original problem, with the iterative approach aiming at decreasing the error at each iteration.

It should be noted that the solution to the aggregate formulation is not necessarily feasible for the disaggregate case. However, for certain models it may be possible to formulate the aggregation in such a way as to yield a strict bound to the original problem, and guarantee feasibility for the disaggregate level, as shown by Iyer et al. (1998) for the aggregation of oil wells for oil production planning. One approach to reduce the number of constraints is to linearly combine some of them into a surrogate constraint where the aggregation coefficients are modified iteratively (see e.g. Ermoliev, Kryazhimskii, & Ruszcynski, 1997). Wilkinson, Cortier, Shah, and Pantelides (1996) use a constraint aggregation approach to solve a large-scale production and distribution planning problem for multiple production sites. In their work an upper level aggregate model is solved to set production targets and also yield a strict upper bound to the original problem, after which the detailed scheduling can be optimized for each site individually with fixed targets, thus decreasing the computational effort significantly. Wilkinson et al. (1998) proposed aggregate formulations for large-scale process scheduling problems using ideas of approximation of difference equations, as well as decomposition approaches for solving these models. In the case of multiperiod models, an approach that works well is to aggregate the time periods. This is especially true when the model involves two hierarchical time levels, such as combined design and planning or combined planning and scheduling. Van den Heever and Grossmann (2000) combined the bilevel decomposition approach mentioned above with the aggregation of time periods by aggregating in the upper level problem with subsequent disaggregation in the lower level planning problem. An additional subproblem is solved after each iteration to determine the best new aggregation scheme (which periods should be grouped together) and information from the aggregation subproblem is used at each iteration to eliminate variables in the lower level problem. It was found that the error introduced by the aggregation of the time periods was very small, mainly due to the optimal aggregation subproblem. Other aggregations schemes include the aggregation of products into families of similar products for the scheduling of multiproduct plants (Kondili, Pantelides, & Sargent, 1993). Where uncertainty is incorporated through a scenario-based model, scenario aggregation can speed up the solution time significantly. The scenario aggregation approach was applied to a mixed-integer linear multiproduct production planning problem by Jorsten and Leisten (1994) who exploited the coupling between continuous and integer planning variables to allow application of the scenario-aggregation algorithm originally proposed by Rockafeller and Wets (1991) for continuous models.

Apart from decomposition and aggregation techniques, some other heuristic approaches address the solution of large-scale planning and scheduling problems. One such heuristic is a capacity shifting heuristic presented by Ahmed and Sahinidis (2000) for a class of process planning problems. These authors show that the error of their heuristic algorithm vanishes asymptotically as the problem size increases. This is a very nice result, considering that the solution time increases exponentially with the number of time periods for an exact solution algorithm.

5. Scientific computing

Future developments for large-scale optimization are also driven by rapid advances in scientific computing, both in performance growth of computing hardware and in object oriented software development.

5.1. Performance increases in computer hardware

Leading the way toward state of the art computing are rapid advances in supercomputers. Based on the extrapolation of historical data (e.g. the CDC6600 at 1 Mflop/s in 1965, Cray XMP at 1 Gflop/s in 1983, ASCI Red at 1 Tflop/s in 1997) one can conservatively estimate a peak computing speed of 1 Pflop (1015) in 2009.

Moreover, at supercomputing centers and national laboratories, large clusters based on commodity hardware will exceed 10 Tflop/s peak speeds during 2002. For instance, the Pittsburgh Supercomputing Center’s Compaq AlphaServer system, achieved peak speeds of 6 Tflop/s in fall 2001. The critical challenge will be to design algorithms capable of exploiting this power. Furthermore, a number of aggressive development projects by a number of hardware vendors suggest that the growth of computers may be accelerated over the next few years. Probably the most significant one is IBM’s Blue Gene computer project, which is expected to achieve 1 Pflop/s by 2005. This advancing technology will influence the growth of “accessible” computer resources at various centers.

At the level of desktop computing, Gordon Moore, Intel’s founder, first observed the “doubling of transistor density on a manufactured die every year” as early as 1965. Surprisingly, “Moore’s law” is still true almost 40 years later. Moore expects this trend to continue for another 20 years and by 2012, Intel should have the ability to integrate 1 billion transistors (see http://www.intel.com/update/archive/issue2/feature.htm) onto a production die. Moreover, with rapid developments in carbon-based nanotechnology, it is predicted that carbon-based components can be synthesized in ways that allow them to self-assemble
into the desired components. If successful, such components would transcend Moore’s Law and be hundreds of times finer in scale than silicon chips.

Finally, the ubiquitous presence of standard high performance processors and the need for advanced computing has led to the development of cheap, high performance clusters. In particular, the Beowulf class computers have leveraged the availability cost effective components (microprocessors, motherboards, disks and network interface cards) and publicly available, hardware independent software, in particular, the Linux operating system, GNU compilers and programming tools and MPI and PVM message passing libraries.

Finally, experience with high performance computing over the past decade has led to a natural environment for the evolution of Beowulf clusters. Such clusters allow the possibility of large-scale parallel computing for the price of standard components. More information of these clusters can be found on http://www.beowulf.cheme.cmu.edu. To illustrate, a sketch of the current Beowulf cluster (developed by Prof. Steinar Hauan) in the Chemical Engineering Department at CMU is given below:

5.2. Software platforms for model-based optimization

Over the past decade the integration of tools and process design environments has become a major activity in the process industries. The impacts of this integration on standardization of work processes and incorporating design and operation issues into the supply chain (Ramage, 1998; van Schijndel & Pistikopoulos, 1999) are widely recognized as key corporate activities. Marquardt and Nagl (1998) surveyed the development of standards for tool integration and classified these as: presentation integration—integrated tool set with common look and feel presented to user; data integration—sharing and managing relations among data objects; control integration—notification and activation of tools using, e.g. message passing and platform integration—execution of integrated suite of tools on heterogeneous, distributed computer network. Examples of these standards include STEP and PDXI for data integration as well as CAPE-OPEN for data and control integration.

Moreover, commercial examples of tool integration include VTPLAN (Bayer), Plantelligence (Aspen Tech) and SimSight/Simulation Manager (SimSci/Bayer). Academic projects in this area include efforts to support conceptual, design and front end engineering and are exemplified by the n-dim (Carnegie Mellon), KBDS/epce (Edinburgh), and the IMPROVE and CHEOPS (Aachen) Systems.

More generally, to allow for the integration of models, equation set objects (ESO) have been defined and developed as part of the CAPE-Open protocols. Developed through the gProms group at PSE plc, these protocols will allow simultaneous dynamic optimization strategies to interface to a number of existing process models and modeling environments.

Moreover, with the development of special purpose models in engineering design and operations, there is a steady development of general purpose modeling platforms instead of simulators for specific applications. These platforms are especially distinguished by general purpose optimization algorithms, rather than specific procedures and powerful, high level modeling languages. Using automatic differentiation tools, such as ADIFOR, MAPLE or ADOL-C, many of these also provide exact derivative (and sometimes Hessian) information for the optimization algorithm. Among these are GAMS, AMPL and AIMMS for (largely) algebraic optimization models, gProms, ADOPT (Schlegel, Binder, Cruse, Oldenburg, & Marquardt, 2001), OCC (Jockenhoevel, Biegler, & Wächter, 2003) and DynoPC (Lang, Cervantes, & Biegler, 1998) for DAE optimization models, and DAKOTA (http://www.endo.sandia.gov/DAKOTA), MP-Salsa (Lehoucq & Salingar, 1998) and Sundance (Long, 2001) for PDE optimization models.

Finally, much research has been devoted to developing general purpose efficient methods to solve large-scale NLPs. As discussed in Biegler and Grossmann (2004), these methods have been studied and refined for more than a decade, and many users rely on them through the use of modeling environments like GAMS, AMPL and AIMMS and these solvers are also embedded into many other applications.

With so many different well-refined implementations for NLP solvers available, why should more be developed? It is clear that the overriding motivation for developing specialized optimization software is to better exploit the properties of specialized classes of NLPs. For the most part, general purpose NLP solvers can only exploit the general sparsity of an NLP and use direct linear solvers that require explicit nonzero entries from the Jacobian (and only in some cases, Hessian) matrices. These solvers generally can not take advantage of opportunities for problem specific specialized lin-
ear algebra or possibly even parallelism. Examples of such applications include DAE constrained and PDE constrained optimization. Here, large-scale simulation codes have been developed for solving nonlinear DAEs and PDEs with millions (even possibly, billions) of discretized state variables. This has been made possible due to advances in massively parallel iterative linear solvers and multi-processor computers (Womble et al., 1999). Most current optimization software can not take advantage of this work.

To implement the next generation of optimization software, there is a need to allow a sophisticated user to specialize the algorithm to exploit particular classes of optimization problems. This might include problem specific data structures, linear solvers and even opportunities for parallelism. It may also require altering the logic of the algorithm and including problem independent QP solvers, globalization methods (e.g. line searches, trust regions, etc.), quasi-Newton methods as well as many other refinements (e.g. heuristics). Also, if users are allowed to specialize components in the algorithm, the optimization software needs to include optional, but yet relatively efficient, run-time tests to validate the various computations. This includes validating functions and gradients, linear solvers and every other major computation that has clear well-defined post conditions. A number of efforts that lead to these features are described next.

5.3. Object-oriented software libraries for next generation optimization

Efficient and robust coupling of optimization algorithms with new or legacy simulation codes presents several software engineering challenges. Here, object-oriented programming is the modern approach to large-scale coding. Its crux is algorithmic encapsulation and data hiding. In the context of highly-parallel simulations, these approaches are antagonistic to the optimal use of computing resources. Modern simulation codes use specialized data structures; sharing and exposing these data structures to optimizers can compromise robustness. On the other hand, large-scale applications (such as PDEs) may involve millions of unknowns, and passing copies of this data to the optimizers is prohibitive. Therefore, new software technologies need to be devised to allow efficient integration of optimizers and model solvers. In addition, real-time optimization and simulation codes have the additional burden of managing large data sets and data streams effectively in order to avoid I/O-related bottlenecks. Several algorithmic and large-scale PDE-constrained optimization issues are being addressed in a number of commercial software packages, such as gProms and Aspen’s Engineering Suite. Thus, the goal of object oriented software developments is to allow plug and play features in the formulation of optimization models, choice of problem specific options in the algorithm and even in the modification of the basic optimization algorithm.

To emphasize the advantages of next generation optimization software, we briefly describe a prototype developed at Carnegie Mellon, rSQP++. rSQP++ is an object-oriented framework that allows efficient optimization package based on rSQP and developed for a broad variety of engineering modeling systems (Bartlett, 2001). It is also designed to support many different SQP algorithms and to allow for external configuration of specialized application-specific linear algebra objects such as matrices and linear solvers. rSQP++ is extremely flexible in allowing a number of algorithmic options (e.g. merit functions, Hessian updates, second order correction terms), decomposition options (coordinate space, orthogonal decomposition), linear algebra (sparse, dense, banded, ABD, etc.) and QP solvers (QP-KWIK, QPOPT, QPSchur, etc.). Currently, a version of rSQP++ is being developed to also include barrier methods, based on the IPOPT algorithm. The UML object diagram of rSQP++ above illustrates the structure of this program. Moreover, in rSQP++, it is possible for a client to modify the SQP algorithms to meet other specialized needs.
without having to modify any of the source code within the rSQP++ framework, or even having to recompile existing SQP algorithms. Much of this capability is accomplished through a set of carefully constructed interfaces to various linear algebra objects such as matrices and linear solvers. The initial development of rSQP++ was done in a serial environment and therefore issues related to the use of massively parallel iterative solvers are just starting to be addressed (Bartlett & Biegler, 2003). In order to support parallelism more effectively, a flexible abstract vector interface is being integrated into rSQP++ to allow more flexibility in vector implementations. This vector encapsulation allows fully parallel linear algebra. The rSQP++ framework has been benchmarked on a wide variety of math programming problems and has been demonstrated on large linear model predictive control problems (Bartlett, Biegler, Backstrom, & Gopal, 2002) as well as nonlinear model predictive control problems (Bartlett, 2003). In addition, it has been used for the optimization of CVD reactors, modeled with the general purpose finite element package, MPSalsa (van Bloemen Waanders, Bartlett, Salinger, Pawlowski, & Biegler, 2001).

6. Conclusions

Over the past quarter century optimization has become an essential tool in Process Systems Engineering, and has found applications in synthesis, design, control, operations and planning. In part I (Biegler & Grossmann, 2004) we discussed the elements of optimization algorithms and concepts that have made this possible. In this paper we focus on emerging areas of optimization research that were not foreseen 25 years ago. In particular, the following four areas constitute active areas of optimization research and will strongly influence future applications of optimization.

First, deterministic algorithms for global optimization on nonconvex nonlinear and mixed-integer nonlinear programming problems have seen great advances in the past 10 years. Based mostly on spatial branch and bound concepts, these algorithms have been improved by the development of new tight convex relaxations for a variety of functions, and second by more effective strategies for reducing the bounds of the variables. Second, logic-based optimization has started to contribute significantly to the systematic formulation of discrete/continuous problems, particularly in the formulation of integer and mixed-integer constraints. Furthermore, logic-based algorithms, such as constraint-propagation, can potentially lead to significant reductions in the combinatorial search by using logic inference techniques, and constraint-propagation methods. On the other hand, disjunctive optimization methods can lead to a significant strengthening of the lower bounds, as well as a more direct exploitation of reduction in dimensionality in subproblems with fixed Boolean variables. Third, due to the increased scope of decision-making in engineering and business applications, optimization problems will always increase in size and complexity. Thus large-scale optimization remains a challenging research area for NLP and MINLP applications. Moreover, with consideration of PDE models and optimization problems under uncertainty, optimization problems with millions of variables are becoming increasingly common. Finally, advances in scientific computing, both in hardware and software engineering, need to be harnessed to develop new optimization algorithms. Currently, many engineering modeling platforms are being redesigned to take exploit advances in parallel computing architectures and object oriented programming. This allows a window of opportunity to improve the performance of optimization algorithms and their links to larger and more complex engineering models.

Given these challenges, research in optimization remains an exciting and active field. Coupled to fundamental research is the application of optimization to domains that extend far beyond traditional process engineering to areas like computational chemistry and biotechnology. We look forward to optimization formulations and algorithms that have significant impacts on these fields as well.

References


