Chapter 18

CONSTRAINT PROGRAMMING AND GENETIC ALGORITHM

Prakash R. Kotecha, Mani Bhushan
and Ravindra D. Gudi*

Department of Chemical Engineering
Indian Institute of Technology Bombay
Powai Mumbai-400 076, India
ravigudi@iitb.ac.in

1. Introduction

Genetic Algorithms (GA) have found wide acceptance in Chemical Engineering (Yee *et al.*, 2003; Tarafder *et al.*, 2005, 2007; Agrawal *et al.*, 2007) because of their simplicity, and ability to determine optimal or near optimal solutions for both single and multi-objective, non linear optimization problems. In this chapter, we present another AI based technique, Constraint Programming (CP), which has been finding increasing use in Chemical Engineering. Recent applications of CP in Chemical Engineering involving combinatorial optimization include the design of sensor networks based on fault diagnosis perspective (Kotecha *et al.*, 2007), design of robust and reliable sensor network (Kotecha *et al.*, 2008a), design of precise and reliable sensor networks (Kotecha *et al.*, 2008b), short term batch scheduling of chemical plants (Maravelias and Grossmann, 2004; Roe *et al.*, 2005), scheduling of job shop plants (Jain and Grossmann, 2001), and multistage scheduling problems (Harjunkoski and Grossmann, 2002). In this chapter,

we compare CP and GA techniques for solving some of these combinatorial optimization problems. Both these techniques share some common features despite having completely different working principles. In the next section, we give a brief background of CP along with its working principle and demonstrate its working on a simple single objective problem. We present some of its unique features along with discussing its similarities and differences with GA followed by a brief description of its extension to multi-modal and multi-objective optimization (m3o) problems. We subsequently present two case studies (a job shop scheduling problem and a sensor network design problem) widely discussed in the Chemical Engineering literature and demonstrate the performance of both GA and CP on each of these problems. We finally conclude the chapter by summarizing the results and present possible future extensions.

2. Constraint Programming

Constraint Programming is an intelligent, non gradient, tree based, implicit enumerative search technique. It has been developed by the Artificial Intelligence and Computer Science Community for solving Constraint Satisfaction Problems (CSPs) arising in these fields. It has found applications in diverse areas (Van Hentenryck, 2002) such as computer graphics, software engineering, databases, hybrid systems, finance, circuit design, etc. CP has been found to be particularly successful in areas involving combinatorial optimization such as planning, resource allocation and scheduling. A brief description of CP is presented next.

2.1. Working Principle

CP is a domain reduction technique relying primarily on constraint propagation for reducing the domain of the variables. CP solves an optimization problem by transforming it to a CSP. A CSP is a feasibility problem that is similar to an optimization problem except for the fact that it does not have an objective function. Hence, any solution satisfying the set of constraints is an acceptable solution as there is no notion of optimality. Consider a single objective optimization problem as shown in formulation SO.

$$\mathbf{SO} \quad \begin{array}{ll} Optimize & f(\mathbf{x}) \\ s.t & G(\mathbf{x}) \end{array}, \tag{1}$$

where $f(\mathbf{x})$ is a multivariable single objective function that needs to be optimized subject to the set of constraints represented by $G(\mathbf{x})$. The set of constraints represented by $G(\mathbf{x})$ can directly include, without any transformation, constraints involving the non-equality operator ($\neq$), implication constraints or disjunctive constraints. The optimization problem in SO can be converted to a CSP (or feasibility problem) as shown in SF

$$\mathbf{SF} \quad \begin{array}{ll} Solve & G(\mathbf{x}) \\ & F = f(\mathbf{x}) \end{array}. \tag{2}$$

The problem statement in SF is a feasibility problem and as mentioned earlier any set of decision variables which satisfies the set of constraints represented by $G(\mathbf{x})$ is a solution. The feasibility problem in SF ignores the objective function and instead adds a new variable (F) evaluating the value of the objective function of the original problem SO. The following constraint is also added to SF after determining every feasible solution to ensure that the value of the objective function progressively improves throughout the exploration of the search space.

$$F^{k+1} \lhd F^k, \tag{3}$$

where F^k, F^{k+1} indicates the value of the objective function of the kth and $(k+1)$th feasible solution and $\lhd$ is the dominant operator (Deb, 2001). To keep the formulation generic, the dominant operator has been used to avoid the distinction between a maximization problem and a minimization problem. For maximization problems, $\lhd(\trianglelefteq)$ will represent the $>$ ($\geq$) operator and for a minimization problem, it will represent the $<$ ($\leq$) operator. The initial value of the variable (F), F^0, is set to $-\infty(\infty)$ for a maximization (minimization) problem. In this chapter, we will be using the term "suboptimal" solution to denote solutions that are feasible but do not correspond to the global optima. Figure 1 shows the working principle of CP to solve a single objective optimization problem and additional details can be obtained from literature (Lustig and Puget, 2001; Grossmann and Biegler, 2004; Roe *et al.*, 2005).

The first step of the CP algorithm is to propagate the set of constraints to eliminate infeasible values from the domain thus reducing the potential search space. However, this step eliminates only those values in a domain

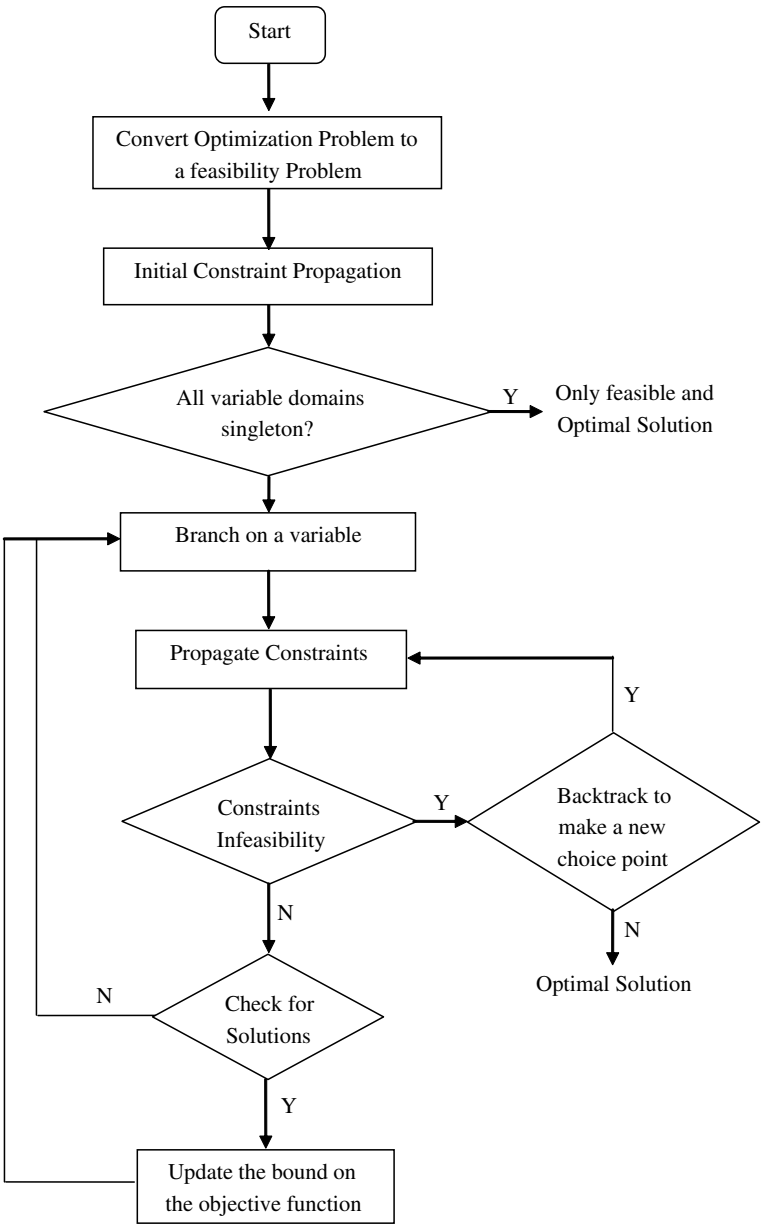

Figure 1. CP algorithm for solving an optimization problem.

that can be identified to be infeasible without making any choice on other decision variables (i.e., fixing any other variable arbitrarily to a particular value from its domain). If this step leads to a singleton set for all the decision variables, then the problem has only one feasible solution which then corresponds to the optimal solution. However, if the reduced domains of all variables are not singleton, a choice point is made, i.e., a decision variable is selected and arbitrarily assigned a value from its (latest updated) domain. The constraints are once again propagated to possibly reduce the domain of other variables. If necessary, additional choice points are made on the other variables during the search till either the domain of any one of the decision variable becomes null or the domain of all variables become singleton. It should be noted that constraint propagation is performed after every choice point. If the domain of any decision variable becomes null, the choice point is said to have led to a failure and backtracking is performed to revise the latest choice point. This procedure of choosing choice point and propagating constraints is continued till the whole search space is explored. It should be noted that on realizing a failure, the latest choice point corresponding to this failure cannot be removed from the domain of the variable. This is because this failure may have been reached for a particular set of choice points. However a choice point can be removed from the domain of a variable if all subsequent choice points on all the other variables lead to a failure. At any point, a feasible solution is said to be obtained if the domain of all variables are singleton. Once a feasible solution is obtained, the value of F is updated and the procedure is continued till the complete search space is explored. However, if every choice point leads to a failure, the problem is an infeasible problem. In literature there are intelligent ways of selecting a choice point leading to fewer failures (Jaffar and Maher, 1994; Grossmann and Biegler, 2004; Roe *et al.*, 2005; Rossi *et al.*, 2006). We next discuss a simple example to illustrate the working principle of CP.

2.2. Example

We now demonstrate the working of CP on a small example to illustrate its superior modeling power compared to traditional optimization techniques and guarantee of global optimality. Consider the following two variable constrained single objective Integer Programming (IP) optimization

problem (SOE).

$$\max x_1 - x_2 \tag{4}$$

$$s.t \quad x_1 \neq x_2; \tag{5}$$

$$x_1 < 2 \Rightarrow x_2 > 4; \tag{6}$$

$$x_1 + x_2 < 5; \tag{7}$$

$$x_1 = \{0, 1, 2, 3, 4, 5\}; \quad x_2 = \{0, 1, 2, 3, 4, 5\}. \tag{8}$$

It can be seen that the constraints involve non-equality, implication and strict inequalities. The use of CP enables the handling of the problem in SOE without any modification of the constraints to the form $G(\mathbf{x}) \leq 0$. As mentioned earlier, the first step is to convert the optimization problem to a feasibility problem as shown in SFE (refer to Fig. 2). From Fig. 2, it can be seen that the initial constraint propagation reduces the domains of variables x_1 and x_2. The value 5 is removed from the domain of x_1 and x_2 as it would violate constraint (7). The values {0, 1} are removed from the domain of x_1 since they would lead to violation of constraints (6) and (7). Thus, the smallest feasible value of x_1 is 2 and hence any value of x_2 greater than 3 would violate constraint (7). It can be seen that the initial constraint propagation removes a large number (but not all) of infeasible values. Figure 2 shows the remaining steps in CP to converge to the globally optimal solution of 4. It can be seen from step 5 that the choice point of $x_1 = 2$ and $x_2 = 1$ leads to a failure. However, the value of 1 is not removed from the domain of x_2 because other combinations of x_1 with $x_2 = 1$ may not violate any constraint. However, at the end of step 5, the value of 2 is removed from the domain of x_1 because all combinations of x_2 with $x_1 = 2$ have been explored and found to lead to failures. Thus, we see that CP can intelligently search the whole space without requiring/determining any gradient information or enumerating the whole search space. The case studies presented later in this chapter will reinforce the power of constraint propagation even for complex problems.

2.3. Features of CP

In this section, we list some of the advantages and disadvantages of CP along with a brief mention of other CP features (Kotecha *et al.*, 2007;

Example:

SOE $\max \quad x_1 - x_2$
$s.t \quad x_1 \neq x_2; \quad x_1 < 2 \Rightarrow x_2 > 4; \quad x_1 + x_2 < 5; \quad x_1 = \{0,1,2,3,4,5\}; \quad x_2 = \{0,1,2,3,4,5\}$

SFE $Solve \quad x_1 \neq x_2; \quad x_1 < 2 \Rightarrow x_2 > 4; \quad x_1 + x_2 < 5; \quad x_1 = \{0,1,2,3,4,5\}; \quad x_2 = \{0,1,2,3,4,5\}$
$F = x_1 - x_2;$

	Operation	**Domain**		**Description**	**Best Sol**
		x_1	x_2		
Step 1	Initial Domains	{0,1,2,3,4,5}	{0,1,2,3,4,5}	Initialization	$F^0 = -\infty$
Step 2	Initial Constraint Propagation	{2,3,4}	{0,1,2}	Select a choice point as the domains of the variables are not singleton	$F^0 = -\infty$
Step 3	Choice Point $x_1 = 2$; Propagating Constraints	2	{0,1}	Select a choice point for x_2 as its domain is not singleton	$F^0 = -\infty$
Step 4	Choice Point $x_2 = 0$	2	0	Feasible Solution; Update the bound on the objective function; Select another choice point as the whole search space has not been explored	$F^1 = 2$
Step 5	Choice Point $x_2 = 1$	2	1	Failure; Backtrack to select a new choice point	$F^1 = 2$
Step 6	Constraint Propagation	{3,4}	{0,1}	Select a choice point as the domains of the variables are not singleton	$F^1 = 2$
Step 7	Choice Point $x_1 = 3$	3	{0,1}	Select a choice point for x_2 as its domain is not singleton	$F^1 = 2$
Step 8	Constraint Propagation	3	0	Feasible Solution; Update the bound on the objective function; Select another choice point as the whole search space has not been explored	$F^2 = 3$
Step 9	Choice Point $x_2 = 1$	4	0	Feasible solution; As the whole space has been explored, this is the optimal solution	$F^3 = 4$

Figure 2. Demonstration of CP on a two variable optimization problem.

Kotecha *et al.*, 2008a). Some of the important benefits of CP over traditional mathematical programming techniques are listed under the following categories:

(i) *Modeling*: The expressive modeling ability of CP can be used to develop compact models with minimal efforts. For example, traditional mathematical programming techniques require the constraints of an optimization problem to be of the form $G(\mathbf{x}) \leq 0$. However, CP permits a richer description of the problem and does not impose any such restriction. As mentioned earlier, it can directly include, without any transformation, constraints involving the non-equality operator ($\neq$), implication constraints or disjunctive constraints. Also, there is no restriction on the variables to be non negative which is in contrast to some existing techniques such as the popular simplex method that uses additional artificial variables to incorporate this non negativity constraint. Additionally, CP modeling environments have problem specific constructs which result in compact models thereby enabling efficient constraint propagation leading to significant reduction in computational efforts.

(ii) *Optimality*: Unlike mathematical programming techniques, CP does not rely on integer relaxations or gradients but instead uses constraint propagation to reduce the domain of the variables. Thus, it avoids convergence to suboptimal (or local) solutions irrespective of the nature (convex/non-convex) of the problem. Hence, CP can guarantee globally optimal solutions for single objective problems and globally optimal pareto points for multi-objective optimization problem irrespective of the combinatorial and convex nature of the problem.

(iii) *Realizations*: The existence of multiple solutions (realizations) is a typical characteristic of both single and multi-objective combinatorial problems. The importance of determining such realizations has been discussed in literature (Tarafder *et al.*, 2007). Such realizations can be determined using CP without repeatedly re-solving the optimization problems.

(iv) *Next best solutions*: The search strategy of CP can be modified so as to determine K-best feasible solutions (K being an user defined

parameter) of a single objective optimization problem instead of determining the globally optimal solution. Depending on the requirement of the designer, the set of distinct or non-distinct K-best feasible solutions can be determined.

(v) *Soft-constraints/soft-objectives:* In certain cases, the enumerative nature of CP can be appropriately used to solve problems involving either soft constraints or soft objectives (Kotecha *et al.*, 2008a).

Inspite of the above advantages, as with any other technique, CP also suffers from certain drawbacks as mentioned below

(i) Though CP enables richer modeling compared to the traditional gradient based mathematical techniques, it nevertheless requires an explicit optimization formulation for the propagation of constraints. This is in contrast to GA which has the capability of handling constructive constraints.

(ii) Most traditional optimization techniques use some mathematical property such as convexity or bounds obtained by appropriate relaxations to confirm local/global optimality of the obtained solution. However, since CP is purely a search based technique, it may require considerable amount of computational resource even after the determination of an optimal solution to confirm this optimality.

Comparative works: In the literature, CP and Integer Programming (IP) have been compared for a wide range of problems like the modified generalized assignment problem (Darby-Dowman *et al.*, 1997), the template design problem (Proll and Smith, 1998), the progressive party problem (Smith *et al.*, 1996), the change problem (Heipcke, 1999), the short term batch scheduling problem (Maravelias and Grossmann, 2004), the job shop problem (Jain and Grossmann, 2001) and the sensor network design problems (Kotecha *et al.*, 2008c). However, to the best of our knowledge, there have been no comparisons of CP with GA. As will be explained later, CP and GA share some interesting common features that help in addressing complexity issues in optimization problems. However, they differ in their search mechanisms and it would be interesting to analyze their relative performances on some benchmark problems. In this chapter, we make such a comparison of GA with CP for two representative problems

reported in the literature and list the advantages and drawbacks of each technique.

Hybrid CP based methods: Recent developments in the field of combinatorial optimization include the development of hybrid methods and integration schemes (Hooker *et al.*, 1999; Van Hentenryck, 2002) which use the complementary properties of the traditional IP optimization techniques and CP to efficiently handle combinatorial problems. Most of these schemes have been developed for scheduling problems (Jain and Grossmann, 2001; Harjunkoski and Grossmann, 2002; Maravelias and Grossmann, 2004; Roe *et al.*, 2005) wherein the original problem is decomposed into a MILP Master problem and a CP subproblem leading to a large reduction in computational efforts.

CP tools: Some of the available CP solvers associated with different optimization software are ILOG Solver (ILOG, Last Accessed: July 2008), Kalis (DashOptimization, Last Accessed: July 2008), CHIP (Dincbas *et al.*, 1988), ECLiPSe (Wallace *et al.*, 1997). CP has been quite popular for solving scheduling problems and hence a large number of scheduling specific developments have been incorporated in various CP tools. The ILOG Scheduler is one such example that has been specially built for efficient modeling and solving of scheduling problems. It provides a large number of constructs that can be used to easily model scheduling problems.

Till recently, CP was primarily used for determining either a feasible solution or determining one single optimal solution for single objective optimization problems. However, recent works (Kotecha *et al.*, 2007, 2008a) have used CP to determine globally optimal pareto solutions and efficiently solve m3o problems to global optimality. To the best of our knowledge, CP is the only technique that can provide guarantee on the global optimality of the pareto front even for non linear Integer Programming problems. In the following sections, we briefly describe some of the CP based techniques that have been proposed in literature for the determination of multiple optimal solutions and the determination of pareto optimal fronts along with their realizations. A detailed discussion and specific observations on each of these techniques and their variants can be obtained from literature (Kotecha *et al.*, 2008d).

2.4. Determination of multiple optimal solutions

A two step CP based strategy has been used in literature (Kotecha *et al.*, 2007) to determine the multiple optimal solutions of a single objective optimization problem.

The two steps of this strategy are (i) the determination of a single optimal solution, and (ii) the determination of all feasible solutions of an appropriate feasibility problem. The first step is as explained in the Sec. 2.1. The second step involves the transformation of the optimization problem into a CSP whose solution will correspond to the optimal solution of the optimization problem. Consider the optimization problem in (1). Let the optimal solution of this problem be $f^{optimal}$. The second step involves modifying the optimization problem in (1) to

$$\begin{aligned} Solve \quad & G(\mathbf{x}) \\ & f(\mathbf{x}) = f^{optimal} \end{aligned} \cdot \tag{9}$$

It is easy to observe that any feasible solution to (9) will also be an optimal solution to (1) due to the constraint $f(\mathbf{x}) = f^{optimal}$ and all the feasible solutions of (9) will correspond to the total multiple optimal solutions of (1). The individual steps in the above strategy are summarized in Fig. 3.

This strategy has the following advantages over the traditional techniques for determining multiple optimal solutions

(a) it does not require re-solving the optimization problem for determining every optimal solution.
(b) it does not require *a priori* specification of the total number of multiple optimal solutions.
(c) it does not require the construction and addition of any cuts.
(d) it does not alter the nature (linear to non-linear) or increase the dimensionality of the problem even in the presence of integer variables.

Though this strategy involves solving the problem twice, the computational effort required in the second step will be comparatively less due to the presence of the constraint $f(\mathbf{x}) = f^{optimal}$. As will be demonstrated in the case study, this constraint is used during constraint propagation and can considerably reduce the search space. The above strategy can be easily modified to determine only a pre-defined number of optimal solutions.

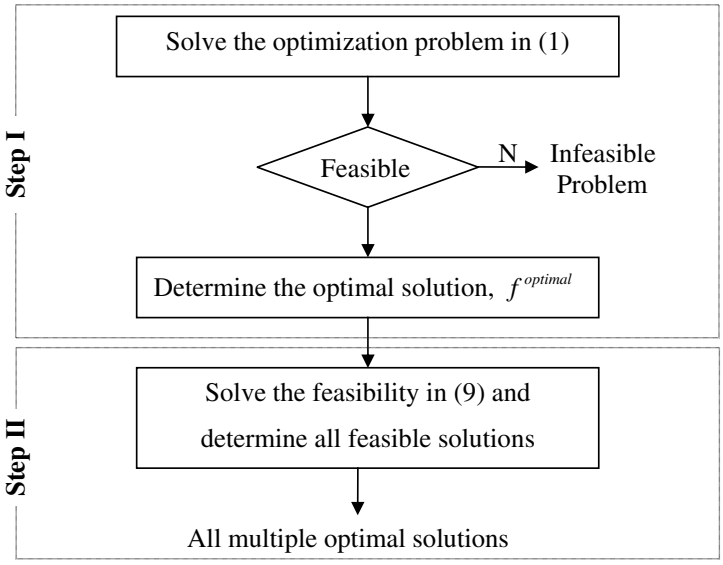

Figure 3. Two step strategy for determining multiple optimal solutions.

2.5. Determination of pareto fronts along with multiple realizations

CP has been successfully used to determine globally optimal pareto fronts along with their realizations. All the techniques proposed in literature for the determination of pareto fronts using CP harness the ability of CP to efficiently solve feasibility problems. The initial work proposed (Kotecha *et al.*, 2007) used the ability of CP to determine all feasible solutions without the addition of integer cuts. The strategy involved the determination of all feasible solutions followed by sorting these feasible solutions to identify the non dominated solutions. This strategy was demonstrated on the benchmark Tennessee Eastman (TE) problem for the design of sensor networks for efficient fault diagnosis. However, this technique requires the determination of all feasible solutions and can be computationally intensive. Subsequently, a two-step strategy for the efficient determination of pareto optimal solutions has been proposed (Kotecha *et al.*, 2008d). Similar to the previous approaches, this strategy also solves a feasibility problem but in addition employs integer cuts to ensure that inferior points are not explored. These integer cuts are designed based on the principle of pareto optimality

and are added after the determination of each feasible solution ensuring that future solutions are non dominated. As this strategy does not require the determination of all feasible solutions, it can lead to potential computational benefits. This technique has been demonstrated on the benchmark TE problem for the design of sensor networks for efficient fault diagnosis. The two step strategy has also been used for the design of robust and reliable sensor networks (Kotecha *et al.*, 2008a) and for the design of reliable and precise sensor networks (Kotecha *et al.*, 2008b). We now present a brief description of this two step strategy and detailed information can be obtained from the cited literature (Kotecha *et al.*, 2008d).

Two step strategy: MO and MF are the corresponding multi-objective counterparts of SO and SF. The problem statement represented in MO is a multi-objective optimization problem with m multivariable objectives.

$$\textbf{MO} \quad \begin{array}{ll} Optimize & f_j(\mathbf{x}); \quad j = 1, 2, \ldots, m \\ s.t & G(\mathbf{x}) \end{array}. \tag{10}$$

MF represents the feasibility problem of MO with each of the objectives evaluated. $F$is a $m$dimensional vector whose jth element corresponds to the value of the jth objective function.

$$\textbf{MF} \quad \begin{array}{ll} Solve & G(\mathbf{x}) \\ & F_j = f_j(\mathbf{x}); \quad j = 1, 2, \ldots, m \end{array}. \tag{11}$$

This strategy consists of two distinct steps: (i) the determination of pareto optimal points, and (ii) the determination of realizations (pareto optimal solutions). The first step involves the conversion of the optimization problem to a feasibility problem (MO to MF) and determining a single feasible solution. Further exploration of the search space is performed with the additional criterion that every subsequent solution should be better than the current solution in at least one of the m objectives. Mathematically, this criterion can be represented by the following Type I cut

$$\textbf{Type I} \qquad \bigvee_{j=1,2,\ldots,m} (F_j \lhd F_j^t), \tag{12}$$

where F_j^t represents the jth objective function corresponding to the tth feasible solution, $\vee$ represents the commonly used OR operation in Boolean logic and $\lhd$ indicates the dominant operator. The successive addition of

Type I cut is continued till either the problem becomes infeasible or the search space is completely explored. The set of pareto points can then be determined by a simple post-optimal sorting of these Type I cuts. The second step of this strategy involves the construction of a single Type II cut corresponding to each Type I cut as

$$\textbf{Type II} \qquad \left\{ \bigwedge_{j=1,2,\ldots,m} (F_j = F_j^i) \right\} \vee \left\{ \bigvee_{j=1,2,\ldots,m} (F_j \lhd F_j^i) \right\}. \quad (13)$$

The T Type II cuts (one for each Type I cut) are subsequently added to the feasibility problem and all solutions of this modified feasibility problems are determined. This solution set directly corresponds (does not require any sorting) to all the pareto optimal points along with all possible realizations for each of the pareto point. The two step strategy to determine globally optimal pareto fronts/pareto solutions is summarized in Fig. 4.

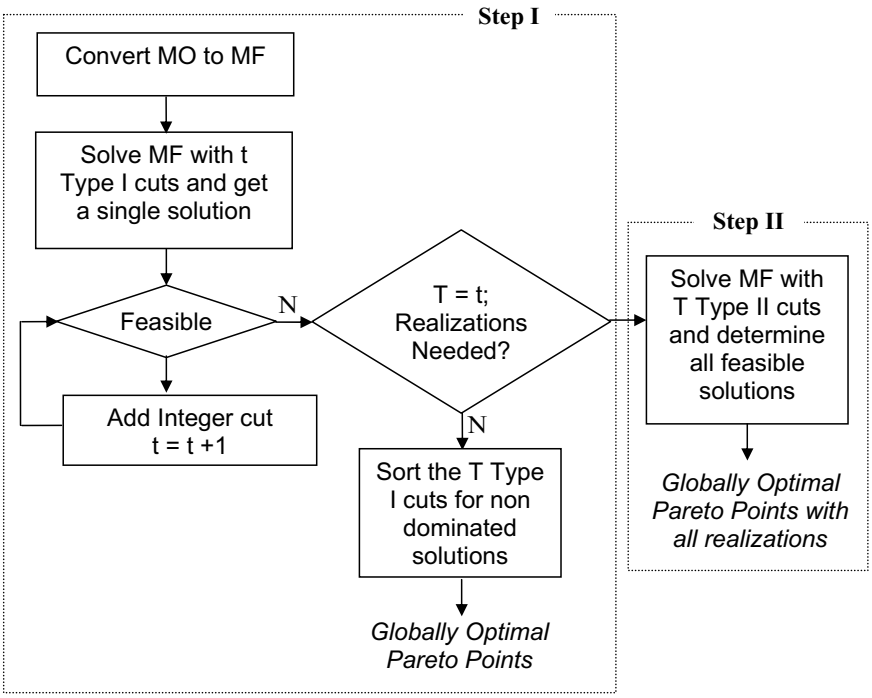

Figure 4. Two step strategy to determine globally optimal pareto solutions.

2.6. Comparison of CP and GA

In this section, we compare and contrast both the CP and GA techniques which have originated from different backgrounds. CP was developed by the Computer Science & Artificial Intelligence community whereas the stochastic GA technique finds its roots in the theory of evolution. From an application point of view, GA has found relatively more applications in Chemical Engineering than CP whereas CP has found considerable applications in scheduling problems.

Similarities between CP and GA: The following are some of the similarities between GA and CP:

(i) Both these techniques fall under the paradigm of AI. These techniques are independent of the geometry of the problem and do not make use of any gradient information.
(ii) Both these techniques can be used for solving both single objective optimization and multi-objective optimization to determine the pareto-optimal solutions (along with realizations) in a single run.
(iii) Both these techniques offer high modeling flexibility and do not restrict the designer to specify constraints in the form of $G(x) \leq 0$. Neither do these techniques require the decision variables to be non negative.
(iv) Both these techniques can determine multiple realizations and K-best solutions without the use of any integer cuts and without re-solving the problem.
(v) Both these techniques can prove to be computationally intensive as these do not have a bound on the objective function due to the absence of relaxations. This is evident from the fact that these techniques very often continue the exploration of the search space to prove the optimality (or satisfy the convergence criteria) despite attaining the optimality.

Differences between CP and GA: CP is a deterministic technique in nature and hence its results are easily reproducible whereas GA is a stochastic technique and the results are reproducible only if the same set of random numbers is generated. Other prominent differences between CP and GA can be categorized into three subdivisions and are summarized in Table 1. We would like to emphasize that the distinctions reported in this table are between the most commonly used implementations of CP and GA.

Table 1. Differences between CP and GA.

Sl. no.	Parameter	Constraint programming	Genetic algorithm
I. Modeling			
1	Handling of constraints	CP predominantly relies on the constraints to reduce the domain of the variables and determine the optimal solutions	GA predominantly relies on the fitness function and ignores the constraints during its principal operations such as crossover and mutation thereby possibly generating large number of infeasible solutions
2	Search Space	The search space is predominantly decided by the set of constraints and is intelligently explored. As the constraints are used for domain reduction, defining a wider domain for variables domain is acceptable as redundant values can get eliminated	The search space is defined by the domain of variables with no role for constraints. Hence the domain of variables has to be chosen appropriately failing which many infeasible solutions can get generated (These infeasible solutions have an inferior fitness value and may get eliminated at a later stage). In recent implementations of GA, this drawback has been overcome by using the repair operator (Michalewicz, 1994; Duda, 2005)
3	Replicates	Works with a single solution and no replicates (solutions identical both in the objective function and decision variables) are ever created/re-visited	Works with a population of solutions and replicates may get created
4	Understanding, ease of implementation	Well understood technique but constraint propagation techniques need to be effectively implemented	Though some attempts have been made, GAs are not yet mathematically well understood. Nevertheless, operations like crossover and mutation are relatively simple to implement

(*Continued*)

Table 1. (*Continued*)

Sl. no.	Parameter	Constraint programming	Genetic algorithm
5	Modeling capabilities	Superior modeling power than the traditional optimization techniques but nevertheless requires an explicit optimization framework	GA can handle constructive constraints and does not require the constraints to be written in terms of explicit equations thereby making it suitable for a wide range of optimization problems
II. Computational Performance			
1	Progression of objective function	The objective function progressively improves throughout the search	Though NSGA-II is designed to progressively improve, it can nevertheless lose globally optimal pareto points due to the distance crowding operator (in cases where the size of population is less than the number of solutions in the best front obtained from the pooling of parents and offspring)
2	Parameter sensitivity	Since the search space of CP is primarily decided by constraints, a change in the parameters involved in the constraints can vary the search space of the problem. This can lead to variation in time required by CP to attain optimality. Hence, CP may be viewed as a technique susceptible to parametric changes	The search space of GA is determined by the domain of the decision variables as specified by the user. This domain may not change due to small variations in the parameters. Hence, the time required by GA to explore a given fraction of search space (as specified by choice of number of generations and population size) may be less sensitive to parametric changes
3	Number of runs required	Single run enables the determination of globally optimal solutions (both for multiple and single objective optimization problems)	Multiple runs are required (in terms of different set of random numbers) to verify/improve optimality

(*Continued*)

Table 1. (*Continued*)

Sl. no.	Parameter	Constraint programming	Genetic algorithm
III. Optimality			
1	User defined parameters	No user defined parameters (the choice points can be arbitrarily selected with no effect on optimality but may affect the time required for optimality)	A considerable number of user defined parameters like size of population, number of generations, mutation probability, crossover probability need to be carefully selected for reaching the optimal solution.
2	Global optimality	Guaranteed optimality for linear and non linear IP problems, irrespective of convexity	No guarantee on the optimality for even well structured problems
3	Realizations	All realizations in single and multi objective optimization can be guaranteed to be obtained	No guarantee of determining all realizations in either single or multi objective optimization. The maximum number of realizations that can be obtained is restricted by the population size
4	Stopping criterion	Well defined stopping criterion i.e., exploration of the complete search space	Arbitrary stopping criteria (such as fixed number of generations or fixed amount of time)

3. Case Studies

In this section, we evaluate both CP and GA approaches using two case studies taken from the Process Systems Engineering literature. The first case study is a single objective optimization problem involving scheduling of tasks in a job shop while the second case involves the design of sensor networks with multiple objectives. Both the techniques are implemented on a 3.20 GHz Pentium IV CPU with 1 GB RAM and using Windows XP Operating System. The CP solver used is ILOG SOLVER 6.1 and ILOG SCHEDULER 6.1 with ILOG OPL Studio (Van Hentenryck, 2002; ILOG, 2003) as the modeling language. The GA is implemented using an in house developed elitist code on MATLAB 7.5. Unless otherwise mentioned, the

following GA parameters have been used: The crossover probability is set to 0.9. The mutation probability is set to $1/n_d$, where n_d is the actual number of decision variables. The population size is fixed at $20n_d$. The random numbers are generated using the *rand* function in MATLAB with the seed value set to zero.

3.1. Single objective optimization: Job shop scheduling

In this section, we will be using a job shop scheduling (Jain and Grossmann, 2001) combinatorial optimization problem for comparing CP and GA regarding their capability to (i) efficiently model a problem, (ii) determine the globally optimal solution, and (ii) determine all the multiple optimal solutions.

3.1.1. Problem definition

This problem involves the processing of a given set of orders (I) on a set of dissimilar parallel machines (M). For each order, a release date and due date is specified. The processing of an order i on any machine m can start only after its release date and has to be completed on or before its due date. Each pair of order and machine has a different cost (c_{im}) and processing time (p_{im}) associated with them. It should be noted that more than one order can be processed on a machine. The two types of decision variables in this problem are (i) assignment of an order to a machine, and (ii) the start time of each order on the assigned machine subject to the following three constraints: (i) the processing of an order cannot start before its release date, (ii) the processing of the order should end before the due date, and (iii) if more than one order is processed on a single machine, then the time of processing of any two orders should not overlap i.e., a second order cannot start before the completion of the first order. The exact number of variables and constraints are dependent on the number of orders and machines in a given problem.

3.1.2. CP formulation

This problem has been addressed in literature (Jain and Grossmann, 2001) and four different explicit optimization formulations, namely (i) MILP

formulation, (ii) CP formulation, (iii) combined MILP-CP formulation and, (iv) MILP/CP hybrid model have been proposed. The discrete time CP scheduling model presented here was proposed in literature (Jain and Grossmann, 2001) and is based on the ILOG's OPL modeling language which has a wide range of constructs for scheduling problems. These constructs make the modeling task easier leading to compact models and also have efficient constraint propagation schemes to reduce the computational effort. ILOG OPL considers the set of orders to be processed as activities that have to be completed and the set of machines available to process these orders as resources. Each activity is associated with a start time, processing time, and end time. The CP model is as given in F1:

F1

$$\min \sum_{i \in I} C_{iz_i}, \tag{14}$$

$$\text{s.t} \quad i.start \geq r_i \quad \forall i \in I, \tag{15}$$

$$i.start \leq d_i - p_{z_i}, \tag{16}$$

$$i.duration = p_{z_i} \quad \forall i \in I, \tag{17}$$

$$i requiresT \quad \forall i \in I, \tag{18}$$

$$activityHasSelected \,\text{Re}\, source(i, T, t_m) \Leftrightarrow z_i = m \; \forall i \in I, m \in M, \tag{19}$$

$$z_i \in M \quad \forall i \in I, \tag{20}$$

$$i.start \in \Im \quad i \in I, \tag{21}$$

$$i.duration \in \Im \quad i \in I. \tag{22}$$

The machine selected to process order i is represented by the variable subscript z_i. Equation (14) is the objective function to determine the cost of a feasible schedule. The start time of the order is given by "$i.start$" and constraint (15) ensures that the start time of any order is not earlier than its release time. Constraint (16) ensures that the order is completed before its corresponding due date. The term p_{z_i} is machine dependent and corresponds to the processing time of order i on the selected machine m. The appropriate value of p_{z_i} is determined using constraint (17). ILOG OPL provides the definition of a special type of resource known as "*unary resource*".

The uniqueness of this type of resource is that they can be used by only one activity at any given point of time. This is identical to the problem at hand and hence the machines are defined as "*unary resources*" represented by T. Constraint (18) enforces that any order i requires a machine/resource from the set T. In addition to the assignment of an order to a specific machine, this constraint also makes sure that the orders on the same machine are sequenced appropriately. The construct "*activityHas-SelectedResource*( )" in constraint (19) is a boolean construct used to assign the machine processing order i to the variable z_i. The variable z_i is assigned the value of the machine m if the machine t_m processes the order i. Constraints (20)–(22) define the domains of the variables z_i, the start time and the durations of the orders respectively. Additional details on various constructs available in ILOG OPL Studio can be obtained from the cited literature. We now discuss some issues related with the GA implementation before presenting the results.

3.1.3. GA implementation

In this chapter, we have made use of an elitist GA. The set of decision variables and constraints is as specified in the earlier section on problem definition.

Decision variables: The two types of decision variables are (i) the machine processing order $i\,(x_{im})$, and (ii) the start time of the order $i\,(ts_i)$. As mentioned earlier, specifying appropriate domains in GA is very critical to its performance. The range of x_{im} is the integer values between 1 and the total number of machines m whereas ts_i can take any value between the release and the due date of order i.

Constraints: In addition to the constraints stated earlier, we have also included the following constraint as it was observed to significantly help GA achieve better solutions

$$\sum_{i \in I} p_{im} x_{im} \leq \max_i\{d_i\} - \min_i\{r_i\} \quad \forall m \in M. \tag{23}$$

This constraint ensures that the total processing time for all the orders assigned to a machine is less than the difference of the latest due date and the earliest release date of orders on that machine. This constraint is used

(Jain and Grossmann, 2001) to tighten the LP relaxation of the MILP model. To aid in the determination of appropriate penalty for constraint violation, all the three types of constraints are normalized by transforming them to the form (Deb, 2001)

$$g(x) \geq 0. \tag{24}$$

Then the penalty associated with a constraint c, w_c is calculated as follows

$$w_c = \begin{cases} |g(x)|, & \text{if } g(x) < 0 \\ 0, & \text{otherwise} \end{cases}. \tag{25}$$

The above equation ensures that the penalty is zero for all feasible solutions giving them better fitness value and higher rate of survival.

Fitness function: The fitness function is used while deciding the candidate solutions for the next generation. For a feasible solution, the fitness function is given as

$$f_{feasible} = \sum_{i \in I} c_{im} x_{im}, \tag{26}$$

where c_{im} is the cost associated with processing order i on machine m. However, an infeasible solution has to be penalized and thus the fitness function is given as,

$$f_{infeasible} = \sum_{i \in I} c_{im} x_{im} + \sum_{c \in C} w_c + \sum_{i \in I} \max_m \{C_{im}\}. \tag{27}$$

the first term on the RHS of (27) corresponds to the cost of processing all the orders on the selected machines. The second term corresponds to the penalty associated with each of the constraint while the last term is an upper bound on the cost. The term $\max_m\{C_{im}\}$ denotes the cost of processing an order i on a machine with the maximum cost. The third term is necessary to ensure that an infeasible solution does not have a better fitness function value than a feasible solution.

GA parameters: The number of generations in GA is restricted to 500 with the rest of parameters as mentioned earlier. We now present the comparative results for the two techniques.

3.1.4. Data

Jain and Grossmann (Jain and Grossmann, 2001) considered 5 different problems of varying sizes (as determined by the number of orders and machines). For each problem size, two sets of data were considered to demonstrate that the difficulty of solving a scheduling problem can vary significantly with data. The processing times in the first data set are longer and have fewer feasible schedules and the total cost of processing all the tasks is higher. All the required parameters for these 10 instances (5 problems with 2 data sets each) can be obtained from the online supplement available with (Jain and Grossmann, 2001). However, we will restrict ourselves to only 6 instances by solving problems of three different sizes. In particular, we have used three problems labeled as Problem 1, Problem 4 and Problem 5 (Jain and Grossmann, 2001). To maintain consistency, we will also refer to these problems as Problem 1, Problem 4 and Problem 5. The dimensionality of these three problems is given in Table 2.

All the required parameters for these 6 instances are given in the Appendix I. In the rest of the chapter, we have used labels to refer to each of the six instances. For example, the label P1S1 indicates the first data set of Problem 1 while P4S2 indicates the second data set of Problem 4.

3.1.5. Results

Table 3 shows the comparison of the problem sizes for the six instances between CP and GA. It can be seen that as the problem size increases from P1 to P5, the number of variables and constraints increases significantly in the CP formulation as compared to that in GA. This shows the superior modeling power of GA which does not require an explicit optimization formulation as opposed to CP. However, it should be noted that a compact

Table 2. Details of orders and machines for the three problems.

Problem	Number of orders	Number of machines
Problem 1	3	2
Problem 4	15	5
Problem 5	20	5

Table 3. Problem statistics for six different instances.

	Problem Statistics			Constraint Programming*		Genetic Algorithm (Number of generations = 500)			
Problem label	Number of orders (n)	Number of machines (m)	Problem[$]	Number of variables	Number of constraints	Number of variables ($n_d = 2n$)	Number of constraints[†] ($n + 2m$)	Population size ($20n_d$)	Size of search space ($m^n \prod_{i \in I} (d_i - r_i)$)
P1S1	3	2	Problem 1 Set 1	15	21	6	7	120	19040
P1S2	3	2	Problem 1 Set 2	15	21	6	7	120	19040
P4S1	15	5	Problem 4 Set 1	120	105	30	25	600	$4.675 * 10^{31}$
P4S2	15	5	Problem 4 Set 2	120	105	30	25	600	$4.675 * 10^{31}$
P5S1	20	5	Problem 5 Set 1	160	140	40	30	800	$3.843 * 10^{42}$
P5S2	20	5	Problem 5 Set 2	160	140	40	30	800	$3.843 * 10^{42}$

[$] As classified in (Jain and Grossmann, 2001).

* As indicated by ILOG Scheduler 6.1 and ILOG Solver 6.1.

[†] Bound constraints on the individual variable are not counted. Also, the constraint that the start time of any order cannot be less than the release date is not included because this constraint is always satisfied as the start time is always selected between the release date and the due date.

model representation need not always translate to a reduced computational time. The interested reader can also refer to literature (Jain and Grossmann, 2001) wherein the problem sizes for a MILP formulation are given. The size of the MILP increases even more drastically than CP. As is obvious, there is no difference in the problem size or search space between the two instances of a particular problem. From Table 3, it can also be seen that whereas the search space for GA increases exponentially from Problem 1 to Problem 5, the population size has not been increased proportionately. Hence the ratio of number of solutions evaluated (calculated as number of generations $\times$ size of population) to the size of search space in GA decreases from P1S1 to P5S2.

Table 4 shows the computational performance of both CP and GA for determining the globally optimal solution along with the number of multiple optimal solutions. For Problem 1, it can be seen that both GA and CP are able to determine the globally optimal solutions in a small amount of time. For P1S1, both GA and CP are able to determine all the three multiple optimal solutions. However, GA is able to determine only two of the three globally optimal solutions for P1S2.

For P4S1, CP is able to determine the globally optimal solution of 115 in less than 111 seconds whereas GA is not able to determine the globally optimal solution even after the completion of 500 generations (102.5 seconds). The best solution reported by GA is 123. Similar results are also observed for problem P4S2. However, the time taken for CP to determine the globally optimal solution is significantly less for this instance of the data set (P4S2). This shows the susceptibility of CP to the change in the data of a problem. It can also be seen that these two instances of Problem 4 are characterized by a large number of realizations and CP is able to determine them in a reasonable amount of time. Due to a large number of realizations, we have restricted ourselves to determining only 10,000 globally optimal solutions. However for P4S1, GA is not able to determine more than one solution (best) and for P4S2 it is able to determine only 22 solutions with an objective of 109 in 500 generations. As mentioned earlier during CP discussion, the second step in CP (i.e., the solution of the CSP while determining multiple optimal solutions) is solved efficiently as 10,000 solutions are discovered in a reasonable time. This case study also shows the possibility of GA not being able to determine all realizations with a fixed population size.

Table 4. Computational performance for the six different instances.

	Constraint Programming[a]				Genetic Algorithm[b]				
Problem label	Time	Objective	Time for realizations[#]	No. of realizations	Time for optimal solution	Objective	Time for best sol	No. of realizations (multiple best sols)	Time for 500 generations
P1S1	~ 0	26	~ 0	3	0.4	26	NA	3	12.76
P1S2	~ 0	18	~ 0	3	0.62	18	NA	2	13.62
P4S1	110.52	115	10.06	10,000[m]	NA	123[s]	1.22	1	102.50
P4S2	3.8	102	15.32	10,000[m]	NA	109[s]	64.25	22	80.14
P5S1	103.47	169[s]	104.64	100[m]	NA	378[i]	53.99	4	138.45
P5S2	535.5	140	99.67	100[m]	NA	157[s]	1.51	8	118.60

[a] indicates the time required on ILOG CP Solver 6.1 and Scheduler 6.1.
[b] indicates the time required by a elitist GA on MATLAB 7.5.
[#] indicates the time taken by CP to determine the multiple optimal solutions (only Step II and does not include the time for determining the optimality in Step I)
[m] indicates the time reported to determine the specified number of solutions.
[s] Suboptimal Solution.
[i] Infeasible Solution; NA indicates that GA has not determined the globally optimum solution and hence the notion of time for globally optimum solution does not exist.

For example, if we assume that GA is able to determine the globally optimal solution, it still cannot determine all the 10,000 realizations because the population size is restricted to 600. A higher population size increases the possibility of determining many more realizations but increases the computational burden.

For P5S1, the optimal solution has been reported (Jain and Grossmann, 2001) as 158. This solution was determined by using a hybrid approach as a standalone CP was not able to determine the optimal solution in 19 hrs (using ILOG OPL Studio 2.1, ILOG Scheduler 4.4 and ILOG Solver 4.4 single processor versions on a dual processor SUN Ultra 60 workstation, refer (Jain and Grossmann, 2001)). Hence, we decided not to determine the globally optimal solution. As can be seen from Table 4, a solution of 169 is obtained in less than 104 seconds. While executing CP, it was found that for further improvement to 168, CP required 1449.11 seconds. In contrast to CP, GA is not able to determine even a single feasible solution in 500 generations. The best objective reached is 378 and this corresponds to an infeasible solution because the maximum possible cost for a feasible solution is $209\left(=\sum_{i\in I}\max_m\{C_{im}\}\right)$. To determine the effect of the number of generations, GA was executed for 10,000 generations. However, no improvement in the objective was observed indicating that even a single feasible solution has not been determined in 10,000 generations. For P5S2, CP determines the global optimal solution of 140 in 535.5 seconds whereas GA converges to a suboptimal solution of 157 in around 1.51 seconds. However, CP was seen to achieve the same objective function value of 157 in 1.72 seconds. As in Problem 4, this problem also has a large number of realizations and due to the complexity of the problem, we have reported the time taken to determine only 100 realizations. For P5S2, GA is able to determine only 8 solutions with a suboptimal cost value whereas CP is able to determine 100 solutions which are globally optimal.

The reasons for the inability of GA to determine the globally optimal solution in four of these six instances can possibly be that the search space is very large for these four instances and GA explores only a small portion of this search space. Moreover, the set of constraints is not used in exploring the search space. In contrast, CP explores the complete search space intelligently with the help of constraints and is able to determine the globally optimal solutions along with their realizations. Thus, we see that despite

the larger problem sizes due to explicit modeling, CP is able to efficiently determine the globally optimal solutions compared to GA.

We next discuss the performance of GA for the above six instances of the problem. In particular, we show the improvement in the objective function value and the number of multiple solutions that are determined in each generation. We also show the number of feasible solutions in each generation along with the time required for each generation.

Figure 5 shows the improvement in the objective function for GA as the evolution progresses for all these three problems and their instances. Due to the elitist nature, the objective function is bound to improve (decrease in this case) monotonically. From Fig. 5, it can be seen that GA converges for all the six problems in less than 200 generations.

For P1S1, GA determines the globally optimal solution (26) in the first generation whereas for P1S2 it determines the globally optimal solution (18) in the 9th generation. For P4S1, GA converges to a suboptimal solution of

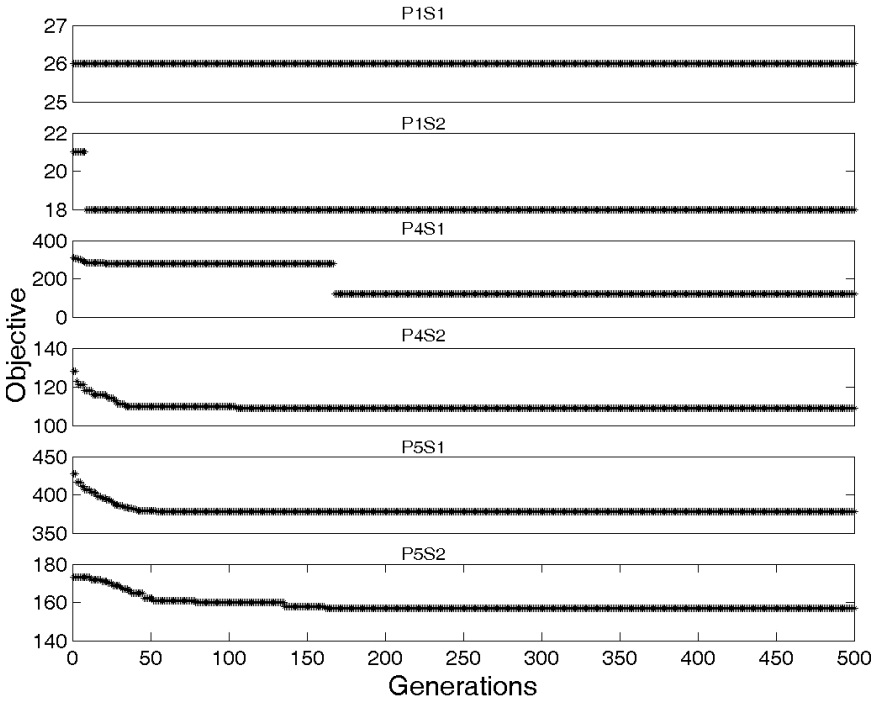

Figure 5. Improvement in the objective function with generations in GA.

123 in the 168th generation while for P4S2, GA converges to a suboptimal solution of 109 in the 105th generation. For P5S1, GA converges to a solution with an objective of 378 which is an infeasible solution as discussed earlier. This can also be seen from Fig. 6 which shows the total number of feasible solutions in a generation. For P5S2, GA converges to a suboptimal solution of 157 in the 162nd generation.

Figure 6 shows the number of feasible solution in each generation. It is obvious that the number of feasible solutions cannot be greater than the population size in a particular generation. The number of feasible solutions should increase monotonically with every generation because of the elitist nature. It is to be noted that due to the stochastic nature of GA, the number of feasible solutions need not remain constant even after the determination of the optimal solution because an infeasible (or inferior) solution from the set of population can be replaced by a feasible (better) solution.

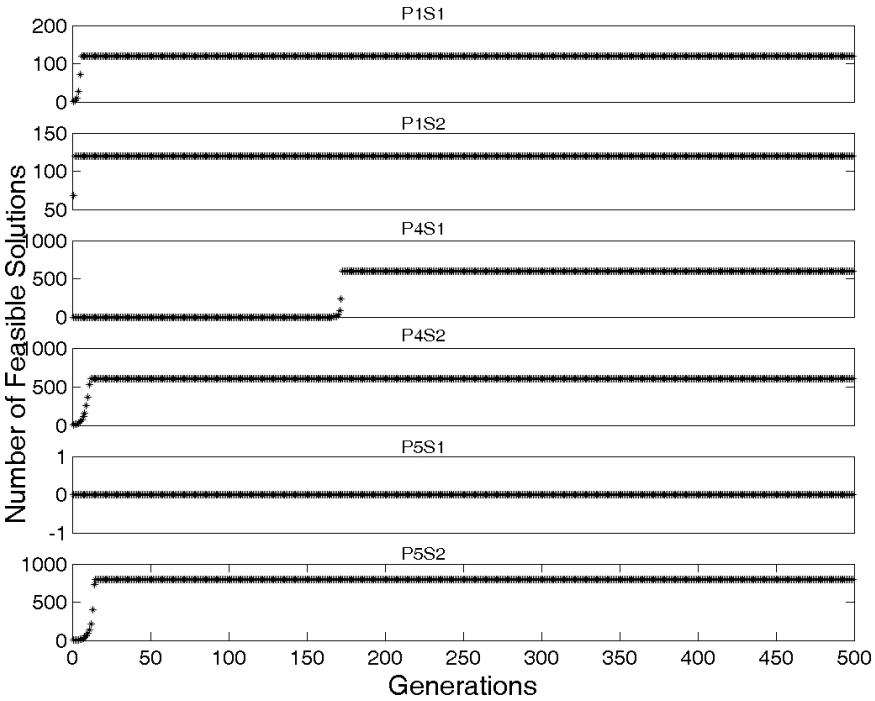

Figure 6. Number of feasible solutions in each generation in GA.

For Problem 1, the population size is 120 and an equal number of feasible solutions are obtained in 6th and 2nd generations for P1S1 and P1S2 respectively. It takes 6 generations for P4S1 to determine feasible solution equaling the size of population (600) whereas 12 generations are required for P4S2. For P5S1, as expected the number of feasible solutions is always zero whereas it takes 15 generations to determine 800 feasible solutions for P5S2. However, many of these solutions may be replicates of one another.

Figure 7 shows the number of current best solutions in each generation. The number of realizations (of the current best solution) need not increase monotonically because there can be an improvement in the objective function which may cause a decrease in the number of realizations. However, the number of multiple optimal solutions can only monotonically increase after the globally optimal solution (or the best solution) is reached because of the elitist nature. In all the cases, GA appears to have converged from the

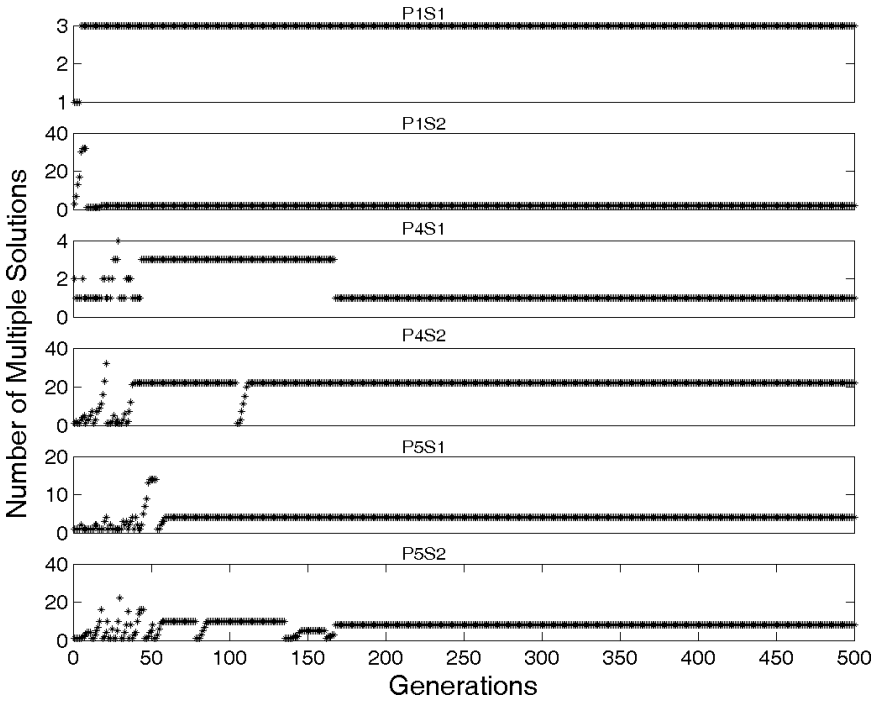

Figure 7. Number of realizations (best) in each generation in GA.

perspective of multiple best solutions. The number of generations required for convergence from the perspective of multiple optimal solutions will never be less than the number of generations for convergence from the perspective of the objective function value. This is because, GA is a stochastic search strategy and all the realizations need not be obtained in the same generation in which the best solution is determined.

All the globally optimal realizations (3) are obtained for P1S1. For P1S2, the number of realizations initially increases and suddenly decreases due to an improvement in the objective function (as can be seen in Fig. 5). However, only 2 of the 3 globally optimal realizations are determined using GA. Similarly for P4S1, the number of realizations generally increases in the beginning and then decreases due to an improvement in the objective function. Only one solution is obtained with the best objective value (123). For P4S2, a total of 22 realizations are obtained for an objective of 109. For P5S1, a total of 4 solutions are reported by GA with an objective of 378. However, all of them are infeasible (also seen in Fig. 6) because the objective value is greater than the maximum possible value of 209 for a feasible solution. For P5S2, 8 solutions are obtained with an objective value of 157 in the 168th generation.

The computational time required for each generation is shown in Fig. 8. As expected, the time for P1S1 (and P1S2) is less than P4S1 (and P4S2). This is because these problems have a larger population size (600 against 120). Similar, observations can be made for P5S1 and P5S2.

As mentioned earlier, the stochastic nature of GA makes it very susceptible to the selection of random numbers. All the six instances of the problem were implemented with 100 different sets of random number (generated by initializing the seed in the *rand* function from 1 to 100 in MATLAB). The objective function at the end of the 500 generations for every seed is shown in Fig. 9 and the number of realizations at the end of the 500 generations is shown in Fig. 10. As can be seen from Fig. 9, the objective function for P1S1 always remains at 26. Except for P1S1, the best objective function for all the other problems does not remain constant. Table 5 shows various statistics including the best possible solution for the six instances.

From Table 5 (and Fig. 9), it is interesting to note that GA is able to determine feasible schedules with an objective of 165 for P5S1. It may be recalled that GA was not able to determine even a single feasible solution

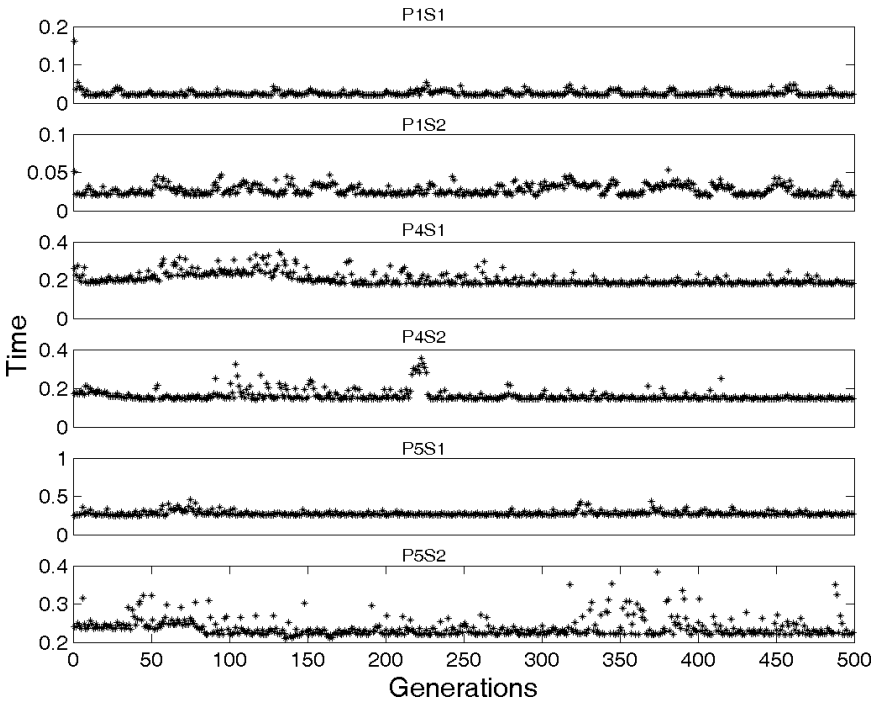

Figure 8. Time required for each generation.

when the random numbers were generated with the seed set to zero. On the other hand, for some values of seed, GA is not able to determine even a single feasible solution for Problem P4S1 (as the maximum value is 283 while the maximum possible cost for a feasible solution is 154).

3.2. Multi-objective optimization

In this section, we present a multi-objective optimization problem taken from the PSE literature and compare the performance of CP and GA on it. We only provide preliminary details about the problem and a complete description can be obtained from (Bhushan *et al.*, 2008).

3.2.1. Problem definition

This design problem involves the selection of the variables (along with the number of sensors for each of the selected variable) to be measured

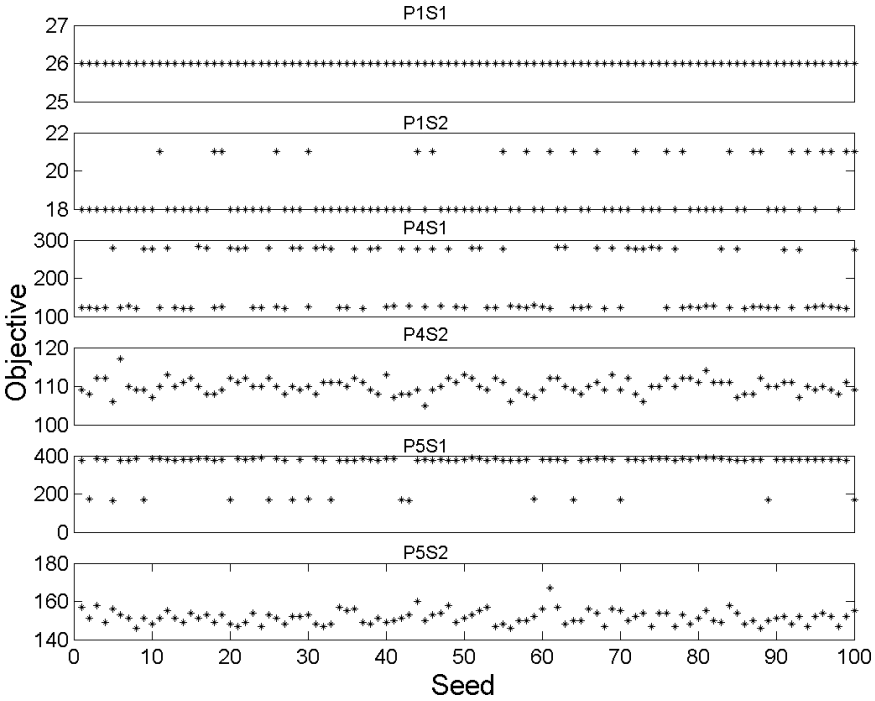

Figure 9. Objective function value for various sets of random number.

in the presence of uncertainties so as to aid in efficient fault diagnosis. These uncertainties can be in either the fault occurrence or sensor failure probabilities or in both these data. An explicit MILP formulation with four different objectives namely (i) minimization of nominal system unreliability of detection (U), (ii) maximization of robustness to the uncertainties in the probability data (ϕ), (iii) maximization of the network distribution (N), and (iv) minimization of the sensor network cost (or the maximization of the cost saving (x_s)) has been proposed in literature (Bhushan *et al.*, 2008). However, these objectives were considered in various lexicographic orderings and hence the trade-offs between them were not fully evaluated.

3.2.2. CP formulation

This problem was reformulated into an explicit CP model (Kotecha *et al.*, 2007) using the expressive power of CP leading to a significant reduction

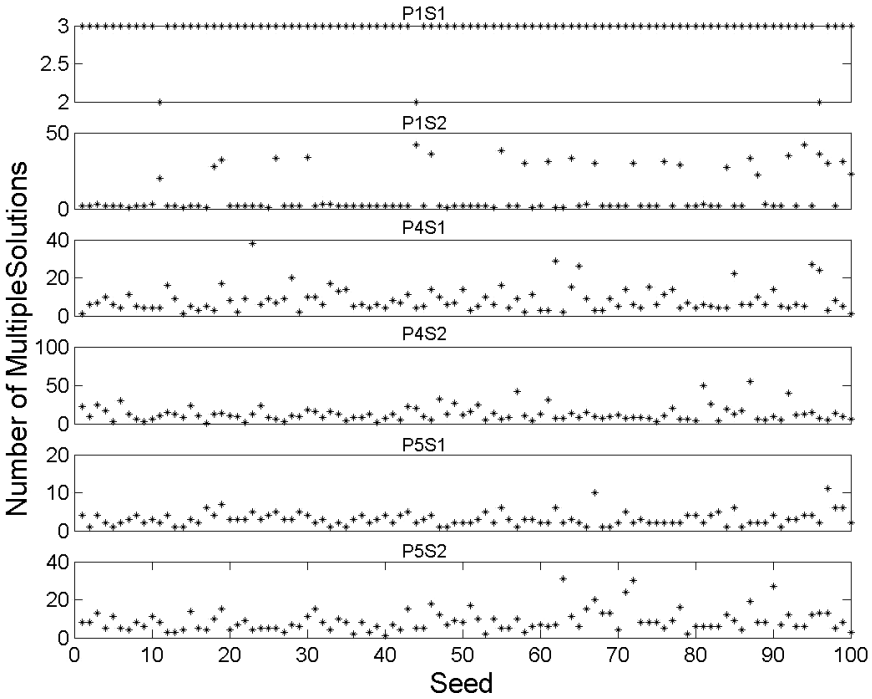

Figure 10. Number of realizations for various sets of random number.

Table 5. Statistics on the six instances with various sets of random number.

Problem tag	Mean	Standard deviation	Global optima	Minimum (Best) solution among the 100 seeds	Maximum (Worst) solution among the 100 seeds
P1S1	26	0	26	26	26
P1S2	18.72	1.2812	18	18	21
P4S1	185.67	75.7262	115	120	283
P4S2	109.96	1.9387	102	105	117
P5S1	349.26	75.4334	158	165	389
P5S2	151.63	3.5990	140	146	167

in the dimensionality of the optimization problem. In this chapter, we will be using the explicit CP based optimization formulation (reproduced in F2) used in literature (Kotecha *et al.*, 2008d). To be consistent with literature and without loss of generality, we will fix the value of one of the objectives

(minimization of nominal system unreliability of detection (U)) to its optimal value and determine the tradeoffs between the other three objectives.

$$\begin{aligned} &\text{Max} \quad [\phi, N, x_s] \\ &\qquad U = U^{Optimal'} \end{aligned} \tag{28}$$

$$\text{s.t} \quad \sum_{j=1}^{n} c_j x_j + x_s = C^*, \tag{29}$$

$$U = \log_{10}(f_i) + \sum_{j=1}^{n} b_{ij} x_j \log_{10}(s_j) + \phi_i, \quad \forall i \in (I_f \cup I_s), \tag{30}$$

$$U_i = \log_{10}(f_i) + \sum_{j=1}^{n} b_{ij} x_j \log_{10}(s_j), \quad \forall i \in I \backslash (I_f \cup I_s), \tag{31}$$

$$U = \max_{\forall i \in I}(U_i), \tag{32}$$

$$\phi_i^* = \phi_{fi}^*, \quad \forall i \in (I_f \backslash I_s), \tag{33}$$

F2

$$\phi_i^* = \phi_{si}^*, \quad \forall i \in (I_s \backslash I_f), \tag{34}$$

$$\phi_i^* = \phi_{fi}^* + \phi_{si}^*, \quad \forall i \in (I_s \cap I_f), \tag{35}$$

$$\left.\begin{aligned} \phi_i < \phi_i^* \Rightarrow \tilde{\phi}_i = \phi_i \\ \phi_i \geq \phi_i^* \Rightarrow \tilde{\phi}_i = \phi^* \end{aligned}\right\}, \quad \forall i \in (I_s \cup I_f), \tag{36}$$

$$\phi = \min_{\forall i \in (I_f \cup I_s)}(\tilde{\phi}_i), \tag{37}$$

$$\phi_{si}^* = -\sum_{j \in J_s} B_{ij}(\log_{10} s_j) x_j, \quad \forall i \in I_s, \tag{38}$$

$$N = \sum_{j=1}^{n} \min(x_j, 1), \tag{39}$$

$$(U_i, U) \in \mathbb{Z}^-; \ (x_j, x_s, \phi, \phi_i^*, \phi_i, \phi_{si}^*, N) \in \mathbb{Z}^+, \tag{40}$$

where I is the set of all faults, I_f is the set of faults with uncertain occurrence probabilities, and I_s is the set of faults that affect variables which can only be measured by sensors with uncertain failure probabilities. Note that the ith fault is characterized as uncertain if (i) its fault occurrence probability is not

known accurately ($i \in I_f$), (ii) failure probability of any sensor available for measuring the variables affected by that fault (the corresponding b_{ij} is 1; b_{ij} is the (i, j)th element of the cause-effect matrix B) is uncertain ($i \in I_s$), or (iii) both i and ii hold ($i \in (I_s \cap I_f)$). In Eq. (30), ϕ_i is the slack in the detection unreliability constraint for the ith uncertain fault. A nonzero ϕ_i ensures that the system detection unreliability is robust to changes in the data used to estimate the detection unreliability of the ith uncertain fault. ϕ_i^* is the corresponding value of the slack that is necessary to ensure complete robustness of the system detection unreliability to the uncertain data used to calculate the detection unreliability of fault i, and is required because the upper values of the probabilities are bounded by 1. Depending on the type of uncertainty in the faults (cases i, ii, or iii), Eqs. (33)–(35) can be used to determine the value of ϕ_i^*. The objective ϕ represents the network's ability to tolerate uncertainty in the probability data without affecting the nominal system detection unreliability and is defined as the minimum of all those ϕ_i which are below their corresponding ϕ_i^* values. Equation (36) ensures that if ϕ_i is less than ϕ_i^*, then the maximum meaningful value of the slack ($\tilde{\phi}_i$) is ϕ_i. On the other hand, if $\phi_i \geq \phi_i^*$ then the maximum meaningful value of the slack is ϕ^*. The values of ϕ^* and ϕ_{fi}^* can be determined a priori (Bhushan *et al.*, 2008) by the following equations

$$\phi^* = \max \left\{ \begin{array}{l} \max\limits_{i \in I_f \setminus I_s} (\phi_{fi}^*), \max\limits_{i \in I_s \setminus I_f} \left(- \sum\limits_{j \in J_s} B_{ij} (\log_{10} s_j) x_j^* \right), \\ \max\limits_{i \in I_s \cap I_f} \left(\phi_{fi}^* - \sum\limits_{j \in J_s} B_{ij} (\log_{10} s_j) x_j^* \right) \end{array} \right\}, \quad (41)$$

$$\phi_{fi}^* = -\log_{10} f_i, \quad \forall i \in I_f. \quad (42)$$

In the above equation, x_j^* is an upper bound on x_j and corresponds to the maximum number of sensors that can be installed on the jth variable on the basis of available cost and is given by $\lfloor C^*/c_j \rfloor$ (where $\lfloor x \rfloor$ indicates rounding off x to the nearest integer not greater than x). In F2, while ϕ denotes the robustness to uncertainty in the probability data, N is the network distribution and is a measure of robustness to modeling errors. Network distribution is defined as the total number of measured variables

and is calculated as shown in Eq. (39). The variable x_s in the objective function represents the cost saving and is equal to the difference in the available and the used cost (Eq. (29)).

3.2.3. GA implementation

We have used the NSGA — II technique (Deb, 2001) to determine the pareto optimal fronts along with their realizations. The various decision variables and constraints are as discussed below

Decision Variables: From the formulation F2, it can be seen that a large number of decision variables such as $x_j, x_s, \phi, \phi_i^*, \phi_i, \phi_{si}^*, N$ are added to express the problem in an explicit optimization framework. However, the only decision variables necessary in this problem are the number of sensors that need to be deployed on each variable i.e., x_j. As mentioned earlier, all these variables are bounded between 0 and x_j^* and hence the search space for these problems is $\prod_{j=1}^{n} x_j^*$.

Constraints: The three types of constraints that define the problem are

(i) the cost constraint represented by (29),
(ii) the value of ϕ should be positive,
(iii) the set of constraints shown below (corresponding to equations (31) and (32))

$$U \geq \log_{10}(f_i) + \sum_{j=1}^{n} B_{ij} x_j \log_{10}(s_j) + \phi_i, \quad \forall i \in (I_f \cup I_s). \qquad (43)$$

The constraints are normalized as explained in the previous case study. Equations (30) are used to determine the values of ϕ_i whereas Eqs. (33)–(35) are used to determine ϕ_i^*; The values of ϕ_i are subsequently used to determine the meaningful slacks $\tilde{\phi}_i$ using Eq. (36) while Eq. (37) is used to evaluate the robustness (ϕ) of the system. The network distribution is calculated using Eq. (39).

GA parameters: The number of generations in GA is set to 2000 while the rest of the parameters are as specified earlier. As discussed earlier, CP is known to determine the globally optimal pareto front along with their

realizations and hence the true pareto can be estimated a priori. Thus at any generation, the distance of the pareto front (Q) determined by GA to the true pareto front (P^*) (determined by CP) can be estimated using the generational distance (GD) (Deb, 2001) given as:

$$GD = \frac{\left(\sum_{i=1}^{|Q|} d_i^p\right)^{1/p}}{|Q|}. \tag{44}$$

For $p = 2$, the parameter d_i is the Euclidean norm (in the objective space) between the solution $i \in Q$ and the nearest member of the true pareto front

$$d_i = \min_{k=1}^{|P^*|} \sqrt{\sum_{m=1}^{M} (f_m^{(i)} - f_m^{*(k)})^2} \tag{45}$$

where $f_m^{*(k)}$ is the mth objective function value of the kth member of P^*. Ideally, due to the elitist nature of NSGA-II, the generational distance should monotonically decrease with the progress of the evolution.

3.2.4. Data

In this chapter, we will be using the Tennessee Eastman Problem for comparing the performance of CP and GA to solve multi-objective optimization problems. The TE flowsheet (shown in Appendix II) has been widely discussed in Process Systems Engineering literature as a benchmark problem. For the sake of brevity, further details of the TE problem are not presented here and the interested reader is referred to the literature (Downs and Vogel, 1993; Bhushan *et al.*, 2008). It has $50 (= n_d)$ potential measurements and 33 faults (16 bidirectional and 1 unidirectional; given in Appendix II). The cost of the sensors, the sensor failure probability, and fault occurrence probability data are taken from literature (Kotecha *et al.*, 2007; Bhushan *et al.*, 2008). All these data have been reported in Appendix II. Similar to literature (Kotecha *et al.*, 2007; Bhushan *et al.*, 2008), we have considered that the sensor failure probabilities of sensors 3 and 4 and occurrence probabilities of faults 1 and 9, are not known accurately. We present the results for the case when the maximum available monetary resource (C^*) is 1000 units (Kotecha *et al.*, 2007).

3.2.5. Results

In this section, we compare the performance of both CP and GA on various criteria for the above case study.

Model statistics: Table 6 shows the number of constraints and decision variables for both the CP and GA formulation. As expected, the GA formulation is more compact than the CP formulation and hence the number of variables and constraints are comparatively fewer.

Pareto Points: It can be seen from Table 6 that CP is able to determine 17 globally optimal pareto points whereas GA is able to determine only 14 of the pareto points. However, all these 14 pareto points discovered by GA are part of the globally optimal pareto front. The details of the pareto points determined by CP and GA are given in Table 7. All these 14 globally optimal pareto points are obtained in the 489th generation. The three points that are not determined by GA are with the tag CO, CP and CQ (marked in bold).

Realizations: From Table 6, it can be seen that CP discovers all the possible 102 realizations on the global pareto front whereas GA is able to determine only 49 (at the end of 2000 generations). The number of realizations of the three pareto points not discovered by GA is 16 and hence it can be seen that GA is able to determine 49 realizations out of 86 realizations for the remaining 14 pareto points. Details of the realizations associated with each pareto point are given in Table 7.

Table 6. Model statistics and computational performance of CP and GA.

Problem statistics								Time (secs)		
CP		GA		Pareto points		Pareto solutions			GA	
No. of vars	No. of cons	No. of vars	No. of cons	CP	GA	CP	GA	CP[a]	Time for best front[b]	Time[c]
81	37	50	14	17	14	102	49	10	3915.3	5175.6

[a] indicates the time required on ILOG CP Solver 6.1
[b] indicates the time required on MATLAB 7.5 for determining the best front i.e., time for 489 generations
[c] indicates the time required on MATLAB 7.5 for 2,000 generations

Table 7. Details of pareto points.

Constraint programming					Genetic algorithm				
	Pareto points					Pareto points			
Tag	ϕ	N	x_s	No. of realizations	Tag	ϕ	N	x_s	No. of realizations
CA	0	1	900	2	GA	0	1	900	1
CB	0	2	800	9	GB	0	2	800	5
CC	0	3	700	16	GC	0	3	700	6
CD	0	4	600	14	GD	0	4	600	6
CE	0	5	500	6	GE	0	5	500	4
CF	0	6	400	1	GF	0	6	400	1
CG	0	7	250	3	GG	0	7	250	1
CH	0	8	100	3	GH	0	8	100	1
CI	3	2	600	1	GI	3	2	600	1
CJ	3	3	500	5	GJ	3	3	500	4
CK	3	4	400	10	GK	3	4	400	6
CL	3	5	300	10	GL	3	5	300	7
CM	3	6	200	5	GM	3	6	200	5
CN	3	7	100	1	GN	3	7	100	1
CO	**6**	**2**	**200**	**1**					
CP	**6**	**3**	**100**	**5**					
CQ	**6**	**4**	**0**	**10**					

Computational Time: In this work, CP and GA have been implemented on different platforms; hence exact comparison of their computation times cannot be made. However, to get an approximate idea, we have reported the time taken by CP and GA to determine their respective pareto fronts in Table 6. It can be seen that CP is able to determine the pareto front in 10 seconds whereas GA takes 3915.3 seconds to identify 14 of the 17 globally optimal pareto points. The actual time taken by GA to complete 2000 generations is 5175.6 seconds. Hence for this case study, the computational performance of CP is superior to GA.

From Fig. 11(a), it can be seen that GA is able to find 1000 feasible solutions in 44 generations. Also from Fig. 11(b), it can be seen that the time for each generation is high initially and decreases as the number of feasible solution increases. This behavior can be attributed to the fact that in the absence of infeasible solutions, there is no computational effort required for determining the penalties associated with the constraints.

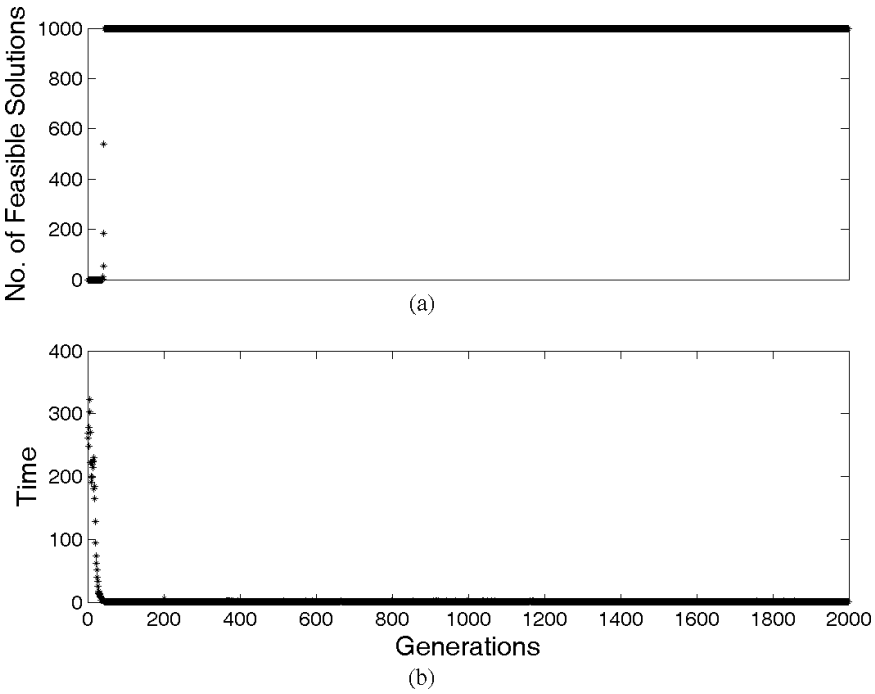

Figure 11. (a) Number of feasible solutions and (b) time taken for each generation.

It can be noted that the number of true pareto points determined by GA should always increase monotonically with the generations whereas the number of pareto points in the best front need not follow this behavior. This is due to the fact that a pareto point can be discovered (in a current generation) which could be dominating more than one another pareto point obtained till the previous generation, causing a decrease in the number of pareto points. However, a true pareto point can never be dominated and hence the number of true pareto points has to increase monotonically with the generations. Figure 12 shows the number of pareto points reported by GA and also the number of true pareto points in this set of pareto points. It can be seen that the number of true pareto points increase monotonically whereas the number of pareto points does not display such behavior. In the 482nd generation, 14 pareto points are discovered of which 10 are globally pareto optimal points. However, in the 489th generation, 14 pareto points are determined which are all globally pareto optimal.

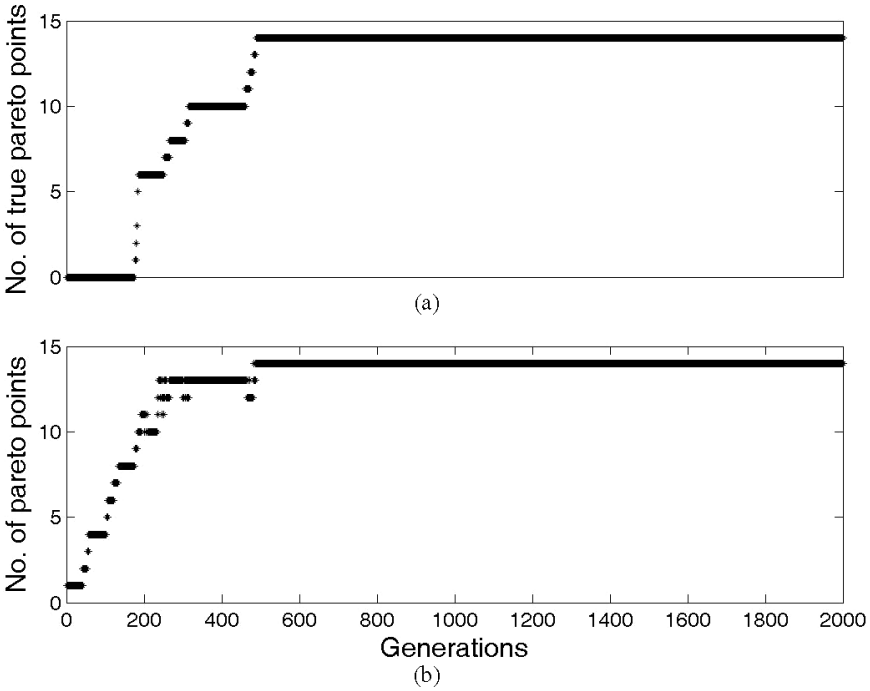

Figure 12. (a) Number of true pareto points and (b) Number of pareto points in each generation.

From Fig. 13(a), it can be seen that the generational distance from the true pareto front monotonically decreases with the generations and finally settles at zero. This should not be used to interpret that the true globally optimal pareto front is determined because the GD will be zero even if GA determines only m pareto points (all of these being present in the true pareto front) out of a total of $n(n > m)$ true pareto points. In the current case, the value of n is 17 and the value of m is 14. Also, since all these 14 points are part of the globally optimal pareto front, the GD becomes zero. This anomaly is due to the fact that the definition of GD (as in (44) and (45)) only takes into account the distance of a pareto point (determined by GA) to the nearest true pareto point. It does not take into account two other important factors namely (i) the number of true pareto points that have been determined by GA, and (ii) the number of realizations discovered by GA compared to the total number of true realizations.

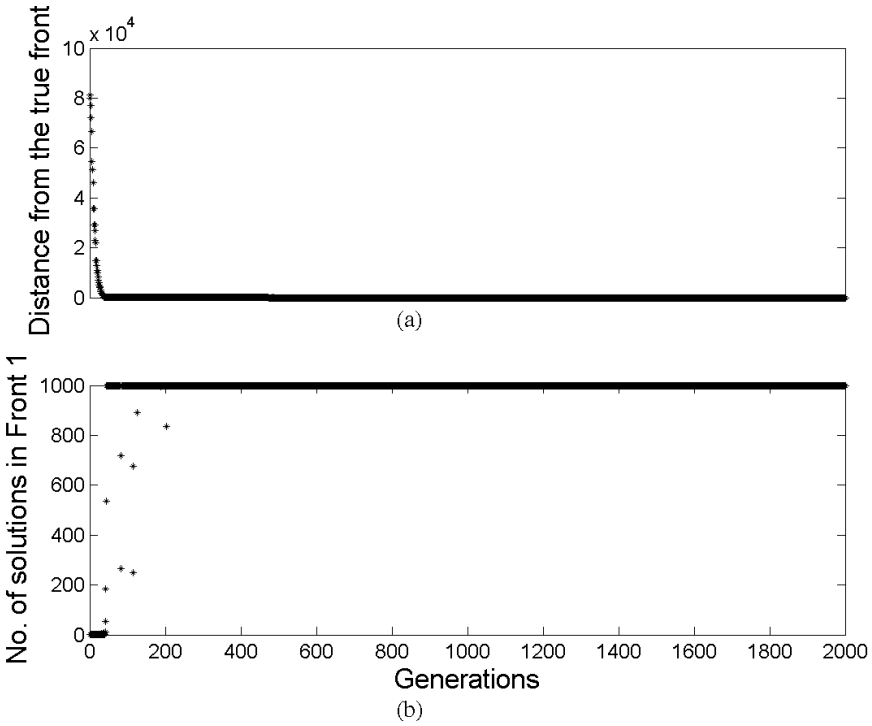

Figure 13. (a) Distance from the true and front and (b) Number of solutions in the best front in each generation.

Figure 14 shows the number of pareto solutions determined by GA and also the number of true pareto optimal solutions among those determined by GA. Ideally, the number of true pareto solutions should increase (for reasons mentioned for the true pareto points) whereas the number of pareto solutions can either increase or decrease. However, in Fig. 14(a) the number of true pareto solutions is not increasing monotonically. This can happen under the following scenario: The set of solutions passed on to a subsequent generation includes all the pareto optimal points of the combined population consisting of parent and offpsring population at the current generation. However, if the number of pareto optimal points (in the combined population) is greater than the population size, the crowding distance metric (Deb, 2001) is used to determine which of the current set of pareto optimal solutions are passed to the next generation. This crowding distance metric

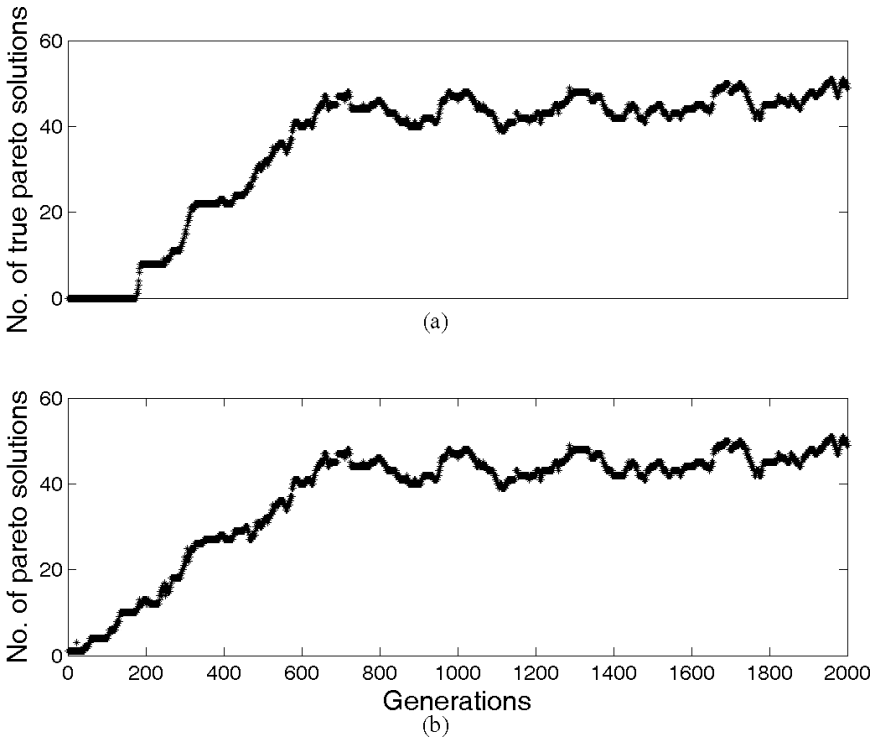

Figure 14. (a) Number of true pareto solutions and (b) Number of pareto solutions in each generation.

however need not necessarily rank a globally optimal pareto point higher than a locally pareto optimal point and hence this strategy can even eliminate a globally optimal pareto point/solution. This is the reason for the non monotonic increase of the number of true pareto optimal solutions. However, this anomaly cannot happen if the number of solutions in the best front of the current generation is less than the population size. As can be seen from Fig. 13(b), the number of solutions in the best front is equal to 1000 (the population size). It should be noted that the loss of globally optimal solutions can happen even for the true pareto optimal points in Fig. 12(a). However, for this case study, such phenomenon does not happen for the number of pareto points possibly due to the existence of a large number of realizations.

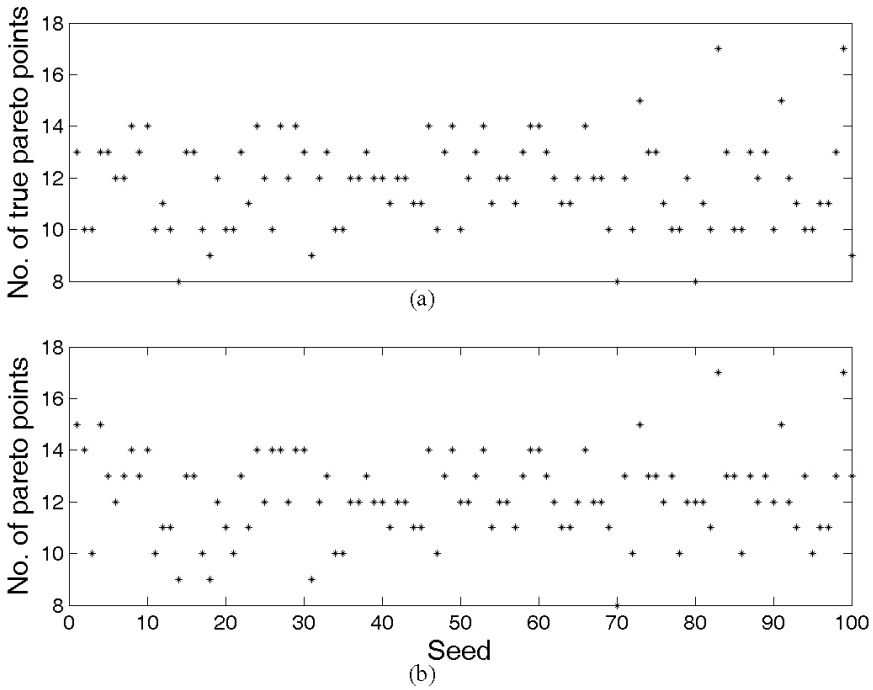

Figure 15. (a) Number of pareto points and (b) Number of true pareto points for various sets of random number.

As in the previous case study, we solved this multi-objective optimization problem with 100 different sets of random numbers (generated by initializing the seed in the *rand* function between 1 and 100 in MATLAB). The number of generations for each set of random number was set to 1,000. Figure 15 shows the number of pareto points as well as the number of true pareto points determined by GA at the end of 1,000 generations for each set of random numbers. It can be seen that GA was able to determine all the 17 globally optimal pareto points in two instances (with seed values 83 and 99). As seen in Fig. 16, these two instances are characterized by 59 and 62 pareto optimal solutions. It should be evident that these pareto optimal solutions will also be globally pareto optimal. However, there is also one instance wherein GA is able to determine only 8 pareto points (all of them globally optimal) with 13 realizations.

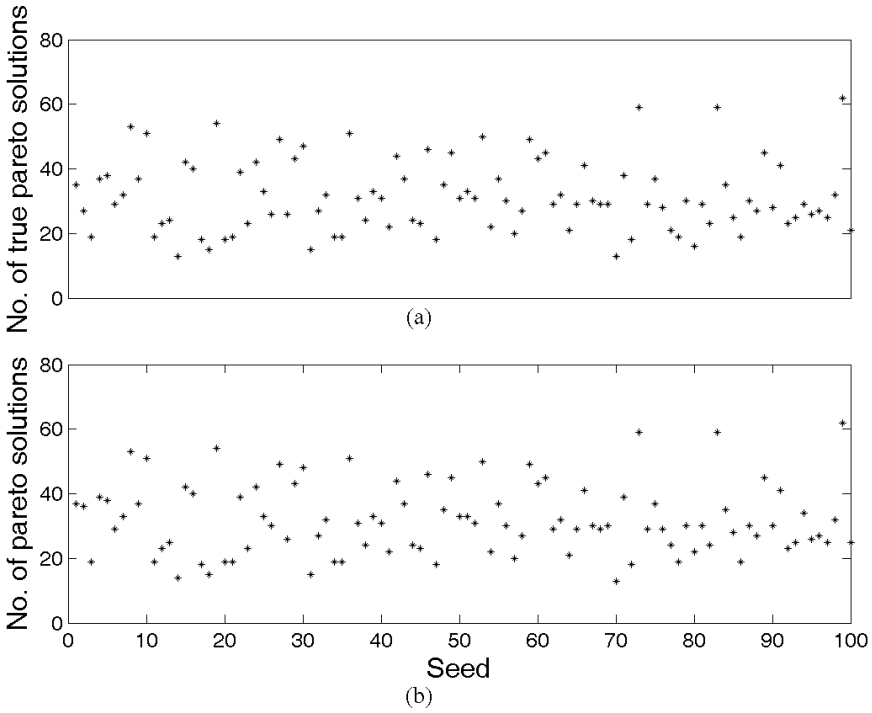

Figure 16. (a) Number of pareto solutions and (b) Number of true pareto solutions for various sets of random number.

Figure 17 shows the distance of the pareto front determined by GA to the globally optimal pareto front. It can be seen that the GD is zero in most of the instances. However, as indicated earlier, it should not be used to interpret that the complete pareto optimal front has been determined. The maximum GD is 6.6622 where GA determined 13 pareto points (with 85 as the seed). Among these 13 pareto points, only 10 are present in the globally optimal pareto front. Thus, we see that GA can possibly determine globally optimal pareto points and solutions but is dependent on the set of random numbers that are used during the various GA operations.

It can be seen from the above case study that CP is able to solve the multi-objective optimization more efficiently than GA. However, it should be mentioned that CP requires the problem to be specified in an explicit optimization framework and this requirement restricts its usage to a limited set of problems. On the other hand, GA does not require explicit optimization

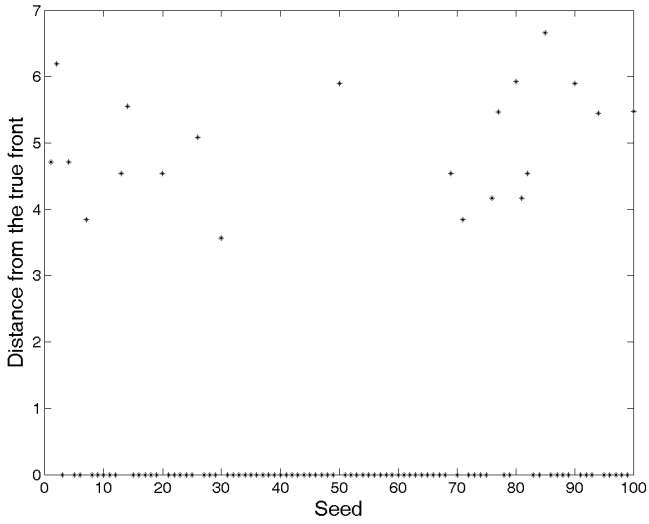

Figure 17. Distance of the pareto front from the true front for various sets of random number.

formulation and hence addresses a wider set of problems, but nevertheless does not guarantee the global optimality of the solution.

4. Conclusions

In this chapter, we have compared the performance of CP and GA in terms of their capabilities to efficiently model the problems, determine the optimal solutions and the realizations for both single objective and multi objective optimization problems. GA was able to model the problem more compactly (less number of variables) than CP as it does not require posing the problem in an explicit optimization framework. However, CP was able to efficiently determine globally optimal solutions and realizations because of its powerful constraint propagation mechanism. The strengths of GA and CP need to be used in a single framework and it may be worthwhile to explore such hybrid strategies. For example, as seen in some instances (especially P5S1), CP can require a considerable amount of time for a small improvement in the objective function. In such cases, a CP-GA based hybrid strategy can be used wherein CP is used to quickly determine some feasible solutions which can be used to populate the initial population in GA. Such a strategy

can help the designer to discover better solutions in a relatively lesser time without any guarantee on the optimality of the solutions.

References

Agrawal, N., Rangaiah, G.P., Ray, A.K. and Gupta, S.K. (2007). Design stage optimization of an industrial low-density polyethylene tubular reactor for multiple objectives using NSGA-II and its jumping gene adaptations. *Chemical Engineering Science*, **62**(9), pp. 2346–2365.

Bhushan, M., Narasimhan, S. and Rengaswamy, R. (2008). Robust sensor network design for fault diagnosis. *Computers & Chemical Engineering*, **32**(4–5), pp. 1067–1084.

Darby-Dowman, K., Little, J., Mitra, G. and Zaffalon, M. (1997). Constraint logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem. *Constraints*, **1**(3), pp. 245–264.

DashOptimization. http://www.dashoptimization.com. (Last Accessed: July 2008).

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA, John Wiley & Sons, Inc.

Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Bertier, F. (1988). The constraint programming language CHIP. *International Conference on 5th Generation Computer Systems*. FGCS-88., Tokyo, Japan.

Downs, J.J. and Vogel, E.F. (1993). A plant-wide industrial-process control problem. *Computers & Chemical Engineering*, **17**(3), pp. 245–255.

Duda, J. (2005). Lot-Sizing in a Foundry Using Genetic Algorithm and Repair Functions. Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, **3448/2005**, pp. 101–111.

Grossmann, I.E. and Biegler, L.T. (2004). Part II. Future perspective on optimization. *Computers & Chemical Engineering*, **28**(11), pp. 1193–1218.

Harjunkoski, I. and Grossmann, I.E. (2002). Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & Chemical Engineering*, **26**(11), pp. 1533–1552.

Heipcke, S. (1999). Comparing constraint programming and mathematical programming approaches to discrete optimisation — the change problem. *Annals of Operations Research*, **50**(6), pp. 581–595.

Hooker, J., Ottosson, G., Thorsteinsson, E.S. and Kim, H. (1999). On integrating constraint propagation and linear programming for combinatorial optimization. *Proceedings of the 16th National Conference on Artificial Intelligence* (*AAAI-99*), Orlando, Florida, USA, The AAAI Press/The MIT Press.

ILOG (2003). *ILOG OPL Studio 3.7 Language Manual*.

ILOG. www.ilog.com. Last Accessed: July 2008.

Jaffar, J. and Maher, M.J. (1994). Constraint Logic Programming: A Survey. *Journal of Logic Programming*, **19/20**, pp. 503–581.

Jain, V. and Grossmann, I.E. (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, **13**(4), pp. 258–276.

Kotecha, P.R., Bhushan, M. and Gudi, R.D. (2007). Constraint programming based robust sensor network design. *Ind. Eng. Chem. Res.*, **46**(18), pp. 5985–5999.

Kotecha, P.R., Bhushan, M. and Gudi, R.D. (2008a). Design of robust, reliable sensor networks using constraint programming. *Computers & Chemical Engineering*, **32**(9), pp. 2030–2049.

Kotecha, P.R., Bhushan, M. and Gudi, R.D. (2008b). Unified approach for the design of robust, reliable and precision sensor networks. *Foundations of Computer-Aided Process Operations (FOCAPO) 2008*, Cambridge, Massachusetts, USA.

Kotecha, P.R., Bhushan, M. and Gudi, R.D. (2008c). Comparison of mathematical programming and constraint programming for the design of sensor networks. *International Conference on Emerging Technologies and Applications in Engineering, Technology and Sciences* (*ICETAETS*), Rajkot, Gujarat, India.

Kotecha, P.R., Bhushan, M. and Gudi, R.D. (2008d). Efficient optimization strategies with constraint programming. *AIChE Journal*. Submitted for publication.

Lustig, I.J. and Puget, J.F. (2001). Program does not equal program: Constraint rogramming and its relationship to mathematical programming. *Interfaces*, **31**(6), pp. 9–53.

Maravelias, C.T. and Grossmann, I.E. (2004). A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Computers & Chemical Engineering*, **28**(10), pp. 1921–1949.

Michalewicz, Z. (1994). *Genetic Algorithms+Data Structures=Evolution Programs*. Berlin, Springer-Verlag.

Proll, L. and Smith, B. (1998). Integer linear programming and constraint programming approaches to a template design problem. *INFORMS Journal on Computing*, **10**, pp. 265–275.

Roe, B., Papageorgiou, L.G. and Shah, N. (2005). A hybrid MILP/CLP algorithm for multipurpose batch process scheduling. *Computers & Chemical Engineering*, **29**(6), pp. 1277–1291.

Rossi, F., Beek, P.V. and Walsh, T., (eds.) (2006). *Handbook of Constraint Programming*, Elsevier Publications.

Smith, B.M., Brailsford, S.C., Hubbard, P.M. and Williams, H.P. (1996). The progressive party problem: Integer linear programming and constraint programming compared. *Constraints*, **1**(1/2), pp. 119–138.

Tarafder, A., Rangaiah, G.P. and Ray, A.K. (2005). Multiobjective optimization of an industrial styrene monomer manufacturing process. *Chemical Engineering Science*, **60**(2), pp. 347–363.

Tarafder, A., Rangaiah, G.P. and Ray, A.K. (2007). A study of finding many desirable solutions in multiobjective optimization of chemical processes. *Computers & Chemical Engineering*, **31**(10), pp. 1257–1271.

Van Hentenryck, P. (2002). Constraint and integer programming in OPL. *INFORMS Journal on Computing*, **14**(4), pp. 345–372.

Wallace, M., Novello, S. and Schimpf, J. (1997). ECLiPSe: A platform for constraint logic programming. *ICL Systems Journal*, **12**(1), pp. 159–200.

Yee, A.K.Y., Ray, A.K. and Rangaiah, G.P. (2003). Multiobjective optimization of an industrial styrene reactor. *Computers & Chemical Engineering*, **27**(1), pp. 111–130.

Appendix I

Scheduling Data for Case Study

The following is the data used for the jobshop scheduling case study discussed in this chapter. Data for three different problems are given here labeled as P1, P4 and P5. Each of these problems has two sets of processing time making a total of 6 data instances. These are labeled as P1S1, P1S2, P4S1, P4S2, P5S1 and P5S2.

Table A.1.1. Scheduling data of problem 1 for case study.

			Cost on machine	
Order (i)	r_i	d_i	1	2
1	2	16	10	6
2	3	13	8	5
3	4	21	12	7

		Durations	
Order (i)	Machine (m)	P1S1	P1S2
1	1	10	5
	2	14	7
2	1	6	3
	2	8	4
3	1	11	5
	2	16	7

Table A.1.2. Scheduling data of problem 4 for case study.

			Cost on machine				
Order (i)	r_i	d_i	1	2	3	4	5
1	2	33	10	6	8	9	9
2	3	34	8	5	6	7	7
3	4	31	12	7	10	11	10
4	5	33	10	6	8	9	8
5	10	34	8	5	6	7	7
6	1	34	12	7	10	11	10
7	2	33	12	10	11	12	11
8	4	25	9	5	7	9	8
9	10	38	10	6	8	9	8
10	1	37	8	5	6	7	6
11	5	30	15	9	12	14	13
12	2	20	13	7	10	12	11
13	4	32	9	5	6	8	7
14	6	20	10	6	8	10	9
15	2	25	8	5	6	7	7

		Durations (p_{im})				Durations (p_{im})	
Order (i)	Machine	P4S1	P4S2	Order (i)	Machine	P4S1	P4S2
1	1	10	5	9	1	2	4
	2	14	7		2	4	6
	3	12	6		3	3	6
	4	11	5		4	2	5
	5	13	6		5	3	5
2	1	6	3	10	1	7	2
	2	8	4		2	14	5
	3	7	3		3	11	3
	4	6	3		4	8	2
	5	7	4		5	10	3
3	1	11	2	11	1	8	2
	2	16	4		2	16	3
	3	13	3		3	12	2
	4	11	2		4	10	2
	5	12	3		5	11	2
4	1	6	3	12	1	3	2
	2	12	6		2	6	6
	3	8	4		3	5	4
	4	7	3		4	4	3
	5	8	4		5	5	3

(*Continued*)

Table A.1.1. (*Continued*)

		Durations (p_{im})				Durations (p_{im})	
Order (i)	Machine	P4S1	P4S2	Order (i)	Machine	P4S1	P4S2
5	1	10	2	13	1	4	1
	2	16	4		2	10	3
	3	12	3		3	7	3
	4	12	2		4	5	2
	5	13	2		5	6	2
6	1	7	1	14	1	2	2
	2	12	3		2	4	5
	3	10	2		3	4	5
	4	8	2		4	3	2
	5	9	2		5	3	3
7	1	10	1	15	1	7	4
	2	13	2		2	14	7
	3	10	1		3	13	6
	4	11	1		4	10	4
	5	12	1		5	11	5
8	1	4	2				
	2	10	5				
	3	8	4				
	4	5	3				
	5	6	3				

Table A.1.3. Scheduling data of problem 5 for case study.

			Cost on Machine				
Order (i)	r_i	d_i	1	2	3	4	5
1	2	33	10	6	8	9	9
2	3	34	8	5	6	7	7
3	4	31	12	7	10	11	10
4	5	33	10	6	8	9	8
5	10	34	8	5	6	7	7
6	1	34	12	7	10	11	10
7	2	33	12	10	11	12	11

(*Continued*)

Table A.1.3. (*Continued*)

			Cost on Machine				
Order (i)	r_i	d_i	1	2	3	4	5
8	4	25	9	5	7	9	8
9	10	38	10	6	8	9	8
10	1	37	8	5	6	7	6
11	5	30	15	9	12	14	13
12	2	20	13	7	10	12	11
13	4	32	9	5	6	8	7
14	6	20	10	6	8	10	9
15	2	25	8	5	6	7	7
16	3	34	9	5	7	9	8
17	3	37	10	6	8	9	8
18	7	38	8	5	6	7	6
19	6	32	15	9	12	14	13
20	0	30	13	7	10	12	11

		Durations (p_{im})				Durations (p_{im})	
Order (i)	Machine	P5S1	P5S2	Order (i)	Machine	P5S1	P5S2
1	1	10	5	4	1	6	3
	2	14	7		2	12	6
	3	12	6		3	8	4
	4	11	5		4	7	3
	5	13	6		5	8	4
2	1	6	3	5	1	10	2
	2	8	4		2	16	4
	3	7	3		3	12	3
	4	6	3		4	12	2
	5	7	4		5	13	2
3	1	11	2	6	1	7	1
	2	16	4		2	12	3
	3	13	3		3	10	2
	4	11	2		4	8	2
	5	12	3		5	9	2
7	1	10	1	14	1	2	2
	2	13	2		2	4	5
	3	10	1		3	4	5
	4	11	1		4	3	2
	5	12	1		5	3	3

(*Continued*)

Table A.1.3. (*Continued*)

		Durations (p_{im})				Durations (p_{im})	
Order (i)	Machine	P5S1	P5S2	Order (i)	Machine	P5S1	P5S2
8	1	4	2	15	1	7	4
	2	10	5		2	14	7
	3	8	4		3	13	6
	4	5	3		4	10	4
	5	6	3		5	11	5
9	1	2	4	16	1	3	2
	2	4	6		2	8	4
	3	3	6		3	7	3
	4	2	5		4	5	2
	5	3	5		5	6	3
10	1	7	2	17	1	6	3
	2	14	5		2	12	6
	3	11	3		3	10	4
	4	8	2		4	7	3
	5	10	3		5	8	4
11	1	8	2	18	1	2	2
	2	16	3		2	8	4
	3	12	2		3	6	3
	4	10	2		4	13	2
	5	11	2		5	4	2
12	1	3	2	19	1	4	1
	2	6	6		2	7	3
	3	5	4		3	6	2
	4	4	3		4	5	2
	5	5	3		5	5	2
13	1	4	1	20	1	5	1
	2	10	3		2	7	2
	3	7	3		3	7	1
	4	5	2		4	6	1
	5	6	2		5	6	1

Appendix II

The Data for Case Study

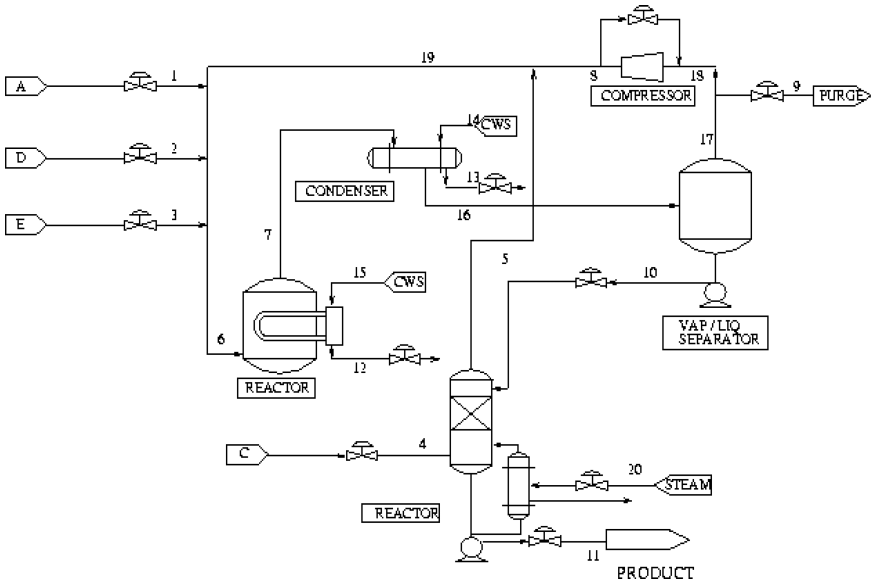

Figure A.2.1. Schematic of the Tennessee Eastman problem.

Table A.2.1. Sensor cost and failure probabilities for potential measurements.

	Var	log s_j	Cost		Var	log s_j	Cost		Var	log s_j	Cost		Var	Log s_j	Cost		Var	log s_j	Cost
1	P_r	−3	100	11	$y_{G,6}$	−3	800	21	$y_{H,7}$	−3	800	31	$x_{E,10}$	−3	700	41	$x_{H,r}$	−3	700
2	P_s	−3	100	12	$y_{H,6}$	−3	800	22	$y_{A,8}$	−3	800	32	$x_{F,10}$	−3	700	42	F_{10}	−3	200
3	P_m	−3	100	13	F_7	−3	300	23	$y_{B,8}$	−3	800	33	$x_{G,10}$	−3	700	43	F_{11}	−3	200
4	F_6	−3	300	14	$y_{A,7}$	−3	800	24	$y_{C,8}$	−3	800	34	$x_{H,10}$	−3	700	44	T_s	−2	500
5	$y_{A,6}$	−3	800	15	$y_{B,7}$	−3	800	25	$y_{D,8}$	−3	800	35	$x_{G,11}$	−3	700	45	VLr_{Xm}	−2	150
6	$y_{B,6}$	−3	800	16	$y_{C,7}$	−3	800	26	$y_{E,8}$	−3	800	36	$x_{H,11}$	−3	700	46	VLr_e	−4	100
7	$y_{C,6}$	−3	800	17	$y_{D,7}$	−3	800	27	$y_{F,8}$	−3	800	37	$x_{D,r}$	−3	700	47	VLs_{Xm}	−2	150
8	$y_{D,6}$	−3	800	18	$y_{E,7}$	−3	800	28	$y_{G,8}$	−3	800	38	$x_{E,r}$	−3	700	48	VLs_e	−4	100
9	$y_{E,6}$	−3	800	19	$y_{F,7}$	−3	800	29	$y_{H,8}$	−3	800	39	$x_{F,r}$	−3	700	49	VLp_{Xm}	−2	150
10	$y_{F,6}$	−3	800	20	$y_{G,7}$	−3	800	30	$x_{D,10}$	−3	700	40	$x_{G,r}$	−3	700	50	VLp_e	−4	100

Table A.2.2. Types of faults and their occurrence probability.

Fault No.	Description	log f_j	Fault No.	Description	log f_j
1,9	F_1 high, low	-2	16,25	$VL_{r_m,bias}$ high, low	-2
2,10	F_2 high, low	-2	17,26	$VL^{set}_{r_m,bias}$ high, low	-2
3,11	F_3 high, low	-2	18,27	$VL_{r_{VP},bias}$ high, low	-2
4,12	F_4 high, low	-2	19,28	$VL_{s_m,bias}$ high, low	-2
5,13	F_8 high, low	-2	20,29	$VL^{set}_{s_m,bias}$ high, low	-2
6,14	F_9 high, low	-2	21,30	$VL_{s_{VP},bias}$ high, low	-2
7,15	T_r high, low	-1	22,31	$VL_{p_m,bias}$ high, low	-2
8	C_d low	-2	23,32	$VL^{set}_{p_m,bias}$ high, low	-2
			24,33	$VL_{p_{VP},bias}$ high, low	-2