Information Discovery from Constraints for Evolutionary Computational Model

Anurag Sharma

A thesis submitted in partial fulfilment of the requirements of the

Degree of Doctor of Philosophy

Faculty of Education, Science, Technology and Mathematics

February 2014

Abstract

Many science and engineering applications require finding solutions to optimization problems by satisfying a set of constraints. These problems are typically ill-structured and intractable. They are formalized as constraint problems (CPs). Evolutionary algorithms (EAs) are known to be good solvers for optimization problems ubiquitous in various problem domains. EAs have also been used to solve CPs, however traditional EAs are 'blind' to constraints as they do not extract and exploit information from the constraints to better 'inform' search for solutions. In this thesis, the constraint problems are investigated and characterized. A variation of EA is studied and proposed which extracts information from constraints to better guide the search process. A novel computational model is developed as a prototype and validated with several benchmark problems. The model is called ICHEA (for Intelligent Constraint Handling Evolutionary Algorithm) and it is developed to work on domains with both quantitative and qualitative data and constraints. ICHEA modeling is designed to be independent of problem parameters and mostly focuses on maximally utilizing information from constraint in its search. It takes the divide and conquer approach for search through incrementally taking a sequence of constraints to solve CPs. It models constraints preferences and constraints strengths and exploits their relationships in search. The computational model developed allows for what-if analysis for exploring search spaces and for revising solution paths. *Incrementality* is a requisite feature as it models solving dynamic CPs where constraints arrive at run time or where constraints change over time. The challenge undertaken is to maximally utilize solutions already developed to a point in time to process any new, arriving sub-sequence of constraints rather than search for a solution anew each time a constraint changes or a new constraint arrives.

The experimental results from ICHEA from benchmark test problems demonstrate improvements on efficiency for the continuous domain and also produce competitive results for discrete domains. The main objective is not only to outperform other algorithms and approaches in efficiency and efficacy but also to provide a generic (i.e. problem independent) computational model that is suitable for constraint handling and solving. Several experiments have been conducted to benchmark ICHEA with other models. The results from these

experiments are encouraging. ICHEA can be extended to support mixed data (continuous and discrete) domains of CPs. It can also be designed based to benefit from a multi-agent environment where each subsequence of constraints clustered on constraint relationships and hierarchies is handled by autonomous agent to get more efficient results. Several future extensions to the current investigation are proposed to address some remaining research gaps from the reflections, and new research questions that emerged from the current work.

Acknowledgment

First of all I thank the Almighty for giving me strength and knowledge to complete this dissertation. I am indebted to my late father Dr. Vivekanand Sharma who himself was a great scholar for his inspiration in the quest for knowledge, love, encouragement and guidance with a rich spiritual and moral upbringing.

I would like to convey my profound gratitude to my supervisor Professor Dharmendra Sharma for his valuable contribution and expertise. I discussed all of my ideas with him and he gave me specific directions to follow on. His guidance, motivation and encouragement helped in every step of this research work. As an international student I received his tremendous support not only for the research work but also for helping me to settle down and be comfortable in this new country by committing his personal time. I would also like to thank Dr. Cecil Schmidt of Washburn University, Topeka for providing the source code for his work and Dr. Trung Thanh Nguyen of The Centre of Excellence for Research in Computation Intelligence and Applications (CERCIA), University of Birmingham, United Kingdom for providing the data files from his experimental results.

I would also like to extend my appreciation to the Faculty of Information Sciences and Engineering (now Faculty of Education, Science, Technology and Mathematic) for their support of my research in terms of providing conference fee and scholarship.

My special thanks to my wife Aleshni and daughter Akshara to keep me motivated and for all their love and understanding throughout the journey.

CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

Many engineering problems ranging from resource allocation and scheduling to fault diagnosis and design involve constraint handling as an essential component that require finding solutions to satisfy a set of constraints over real numbers or discrete representation of constraints (Craenen et al., 2003; Shang and Fromherz, 2003; Craenen, 2005). The very basic form of constraint problems (CP) is constraint satisfaction problems (CSPs). CSP is a problem with a finite set of variables, each associated to a finite domain, and a set of constraints which restrict the values that these variables can simultaneously take. CSP can be defined as triple $\langle X, D, C \rangle$ where $X = \{x_1, x_2, \dots x_n\}$ is an input vector of size n in a finite search space S, D is a collection of domains and C is a set of constraints. Each variable x_i has a finite domain D_i. A set of constraints $C = \{c_1, c_2, \dots c_m\}$ are defined in the form of functions (Tsang, 1993):

$$c_i(x_1, x_2, \dots x_n) = \begin{cases} 1, & if\ satisfied \\ 0, & if\ violated \end{cases} \tag{1.1}$$

Constraint satisfaction sets (feasible regions for each constraint) $\{S_1, S_2, .. S_m\}$ can also be defined where:

$$S_i = \{X \in S \mid c_i(X) = 1, i \in \{1, \dots, m\}\} \tag{1.2}$$

The solution of a CSP is $s \in S$ when all the constraints c_i are satisfied, which can be given as:

$$\sum_{i=1}^{m} c_i(s) = m \tag{1.3}$$

There are many applications of CPs. The following are examples of problems where constraint satisfaction has been successfully applied (Brailsford, 1999; Pemberton and Galiber, 2001; "CTVR: Home," 2011):

- Operations Research (scheduling, timetabling)
- Bioinformatics (DNA sequencing)

- Electrical engineering (fault location, circuit layout computation)
- Robotics (control of modular, hyper-redundant robots)
- Vehicle routing
- Telecommunications
- Hubbell telescope/satellite scheduling
- Numerical computation (solving polynomial constraints)

The CPs can be divided into two classes: *constrained optimizing problems* (COPs) and *constraint satisfaction problems* (CSPs) (Eiben et al., 1998). The difference between these classes is that in the first an optimal solution that satisfies all constraints should be found, while in the second class any solution as long as all the constraints are satisfied is acceptable. Table 1.1 shows the difference between these two classes of constraint problems.

Table 1.1. Types of constrained problems

	Objective function	
Constraints	Yes	No
Yes	Constrained optimization problem	Constraint satisfaction problem
No	Free optimization problem	Not in this category

Constraint problems are called *static constraint problems* (SCPs) if the environment of a given problem remains static i.e. constraints do not change along with time and vice-versa for *dynamic constraint problems* (DCPs). A DCP is a sequence of SCP, each one resulting from some changes in the definition of the previous one. The notion of DCP has been first introduced in (Dechter, 1992). In this form of the problem, initial constraints are subject to change. The constraints can be time dependent or modification due to change in environment or user's requirement. These changes may affect any component of SCP: variables additions or removals, change of domain's initial scope, constraints additions or removals, or simply constraint definitions (Verfaillie and Jussien, 2005). As a result of these changes, assignments that were solutions to a previous CP in the sequence may become invalid, which means that search may have to be restarted to find a solution to the new problem. Even very small changes in a CP can have profound effects on search performance (Wallace et al., 2009). (Sabin et al., 2003) further divides DCP based on their formulation and handling approach but we will use the generic term of DCP of any type of changeable constraints. The class of

problem being solved in this thesis is CPs with either discrete or continuous domain. Dealing with mixed nature of dataset has been a future goal.

The classical algorithms that solve CPs include branch and bound, backtrack algorithm, iterative forward search algorithm, local search but heuristic methods such as evolutionary algorithms (EAs) have mixed success and for many difficult problems these are the only available choices (Brailsford, 1999; Craenen, 2005; Müller, 2005). EAs are based on a population of promising solutions based on the given evaluation function, to which the Darwin's evolution process of selection, reproduction and survival of the fittest individuals are applied in order to find an optimal solution (Salcedo-Sanz, 2009). EAs are acknowledged as good solvers for complex problems (like NP hard and NP complete) of various problem domains, however EAs suffer from some of its inherent problems to solve CPs as they do not make use of information from constraints and only blindly search in the solution space using its heuristic search algorithms (Culberson, 1998; Craenen, 2005).

1.1. Motivation

The main motivation of the thesis is to investigate exploitation of information from constraints for evolutionary search. Traditional EAs are 'blind' to constraint as they do not extract the information and use from the constraints but search the solution through random heuristic greedy approach (Culberson, 1998; Craenen et al., 2000, 2003). The search operators not necessarily produce feasible *offspring* from feasible *parents*. This causes the search engine to spent extra computational effort in searching for the solution into the wider search space without only concentrating in the restricted smaller feasible search space. Constraints can reduce the search space and it can make the heuristic search more efficient by extracting information from constraints to guide the search engine, search in promising search space only where degree of feasibility is high. The constraint handling model should be generic enough that does not require initial feasible solution and produces more promising progenies from parents without using *repair* functions which slows down the process (Coello Coello, 2002). Most importantly it must avoid using problem dependent *penalty* functions. Some research has been reported on working with feasible search space or searching for solution in the boundary between the feasible and infeasible region known as strategic oscillation (Glover and Kochenberger, 1996). Normally, these are problem dependent, or require complex calculation to perform crossover and mutation to produce feasible progeny from feasible

parents. Moreover, finding an initial feasible solution is itself an NP-hard problem (Coello Coello, 2002).

The other major motivation is that “traditional EAs are not intelligent” (Xing et al., 2006; Vivekanandan et al., 2013). The EAs do not learn from past experience in the context of constraint handling. For instance if some constraint is changed for a constraint problem solved by an EA, it has to be restarted to get the new results. A good example for constraint handling problems is university time tabling where some constraints change every semester/year and the time-table has to be regenerated without using any previous knowledge. If the existing technique in any university requires lots of user input then it is a lot of work for the technicians every time they generate a new time table. This drawback of EAs of not imitating real human behavior of learning from past can be labeled as an “unintelligent artificial intelligence” solution. The proposed approach is to provide set of partial solutions that are solved incrementally according to their preferences that leads to the final CSP solution. This also helps in solving DCPs by reusing partial solutions if some constraints are added, updated or removed without regenerating the whole solution.

1.2. Objectives

The main objective of the thesis include development of a variation of EA that uses constraints to guide evolutionary search to get efficient solutions without making it problem dependent as it is known that traditional EAs are ‘blind’ to constraints (Craenen et al., 2003). The other objectives are to

evaluate the performance of the proposed variation of EA with the state-of-the-art algorithm used in solving CPs using benchmark test sets;

develop an algorithm to support revision of previous/partial solutions to arrive to final solution using incremental approach of solving partial solution in each increment. This last objective would help navigating the search space for what-if analysis through partial solutions and the solution space.

1.3. Research Questions

The proposed research centers on the following research questions

1. What are the characteristics of CPs?
2. Define the concepts of dynamic CPs. Can incrementality in the search for solutions be modeled through constraints?
3. Why EAs is chosen for the proposed research.
4. Current evolutionary search operators do not necessarily produce feasible progenies from feasible parents (Craenen et al., 2003). Is it possible to define a new operator that resolves this problem?
5. Can EA support revision of previous/partial solutions to arrive to final solution by caching partial solutions for incremental learning purpose? Would it still be efficient enough to be acceptable for real time applications?
6. Can what-if analysis be done once the solution is prepared (without restarting the system)? Is it possible for EA to mimic human memory to use previous knowledge to come up with the final solution?
7. Can EA be an adaptive algorithm to solve CPs? If the solution is prepared then is it possible to add/remove/update constraints without regenerating the whole solution?
8. Is it possible to handle multi-constraint problem through multi-agents where each agent is in charge of handling a constraint or subset of constraints? Any update/change on the constraints is handled by its owner agent who relays the changes to other agents to get the solution based on current constraints.

1.4. Hypothesis

EAs are used to find a solution for CPs in a finite search space as shown in Fig. 1.1 where constraints are shown as three different circles c1, c2 and c3. The circle shaped points inside the circles are feasible data and star shaped points are infeasible data in a search space. The shaded region indicates the solution space where all three constraints are satisfied. If the search space is very large then intuitively it would be more efficient to search for the solution only in the feasible search space if it is easy to locate feasible data points notwithstanding the complexity of the problem. An intelligent heuristic search mechanism is possible where the

knowledge from the constraints can be extracted to guide the evolutionary search to look for the solution only in the feasible search space that reduces the size of the search space. This may produce a solution more efficiently and more likely it has a higher chance of finding a feasible solution (Sharma and Sharma, 2012a).

EAs have proven to be quite an effective tool for solving some constrained optimization problem (Maciej Norberciak, 2006), however from our knowledge EAs do not make the experience count in the context of constraint handling techniques. They do not use past experience to solve the given problem. Normally, if the problem is slightly changed or a new constraint is added then the algorithms must be restarted to get the new solution which can be very time-consuming. In this case EA methods can be expressed as unintelligent artificial intelligence approach that does not utilize past experience. As human being memorizes good and bad events for decision making so they do not venture into the area if it is highly likely to be a failure. This additional intelligence can be incorporated into the EA to help in finding a feasible solution for CPs. EAs have been able to successfully solve many complex engineering problems and this thesis would elaborate its capability in solving CPs where relatively little research has been done on developing efficient evolutionary solutions (Kramer, 2010).

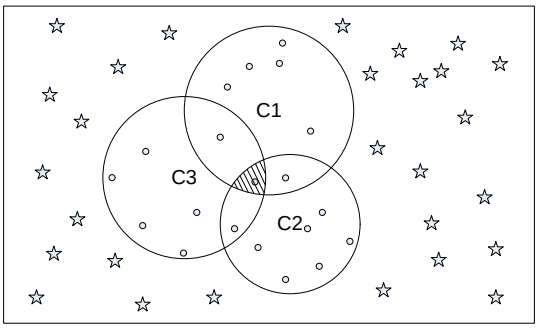


Fig. 1.1. Constraints in a finite search space

The constraint regions are shown as circles c1, c2 and c3 with some circle shaped data points.

1.5. Methodology

There are many non-Artificial Intelligence algorithms approaches that work well with small scale problems like backtracking but in most large scale or complex problems they are not very efficient (Brailsford, 1999; Bliek et al., 2001). EAs are acknowledged as good solvers for many complex problems like NP hard and NP complete problems of various domains. It is

also easier to model the problem into EA as it is inspired from various nature inspired algorithms where user has to provide the objective function of a given problem and the evolutionary search engine does the computational task (Goldberg, 2002; Mezura-montes and Coello, 2006). EAs have become a popular choice to solve different types of optimization problems and CP is an instance of it. So EA is chosen as it is very promising to provide good solutions for this research.

The main focus of this research is to solve CPs by understanding constraints and extracting as much information as possible from them. A variation of EA is developed that is not 'blind' to constraints as traditionally being done to disregard the knowledge hidden in the constraints; instead the information from constraints is utilized to guide the evolutionary search. Some artifacts are developed that demonstrate the intelligent behavior of EA by producing more efficient results. Currently, the penalty functions are used in general to determine the fitness of an individual chromosome in a population which is generally problem dependent and cannot be reused (Kramer, 2010). The proposed research aims at developing a generic, problem independent approach in terms of CP solving where no penalty function will have to be provided and constraints can be added or removed at any stage.

The reason for making the EA intelligent is to make a generic model for CP solving and reusing the existing solution to get the results in timely fashion rather than outperform other problem specific algorithms in terms of efficiency. This in turn can also be used for DCPs where an update/addition/removal of a constraint will not force the system to restart. To make the dynamic behavior more efficient the constraints can be deployed to different agents to solve the problem in multi-agent environment. We named the proposed algorithm ICHEA (Intelligent Constraint Handling Evolutionary Algorithm) which is a variation of EAs concentrating mostly on information from constraints for its heuristic search.

1.6. ICHEA Framework

The general framework of information discovery from constraints for evolutionary computational Models is given in Fig. 1.2. The constraints can be simply in the form of mathematical functions or a list of records require to be processed to extract information about constraints like clashes (constraints) in the university timetable is determined from student data sets and course data sets. Once information about constraints is retrieved there are two ways to handle them. Traditionally an objective function can be created that will

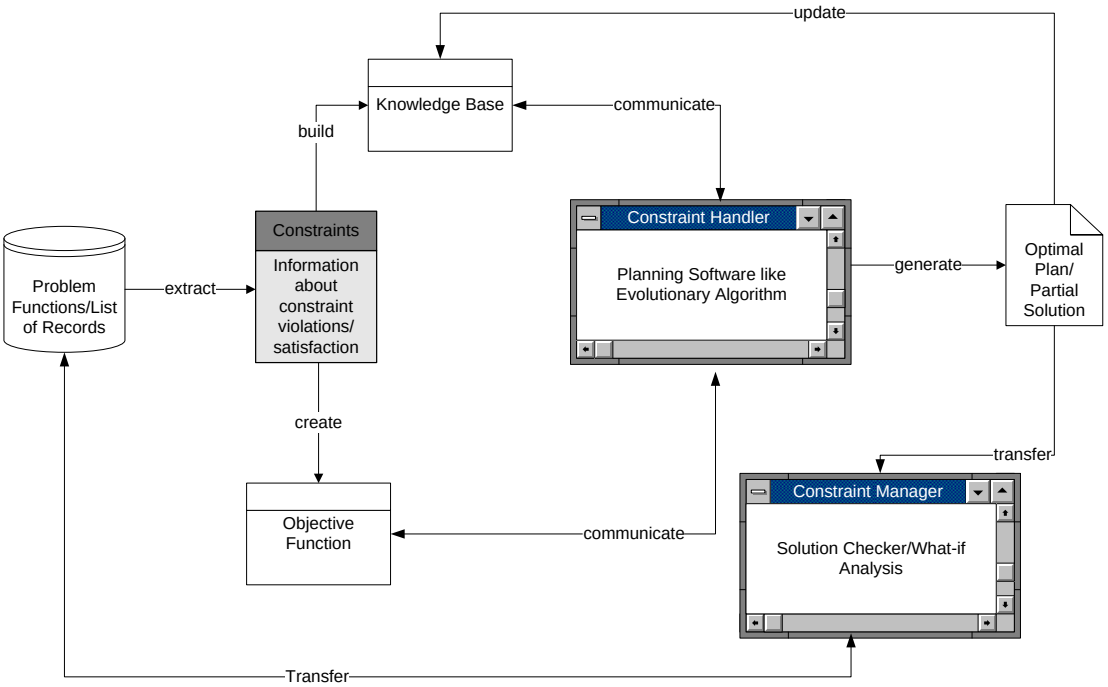


Fig. 1.2. A framework for information discovery from constraints for evolutionary computational model

communicate with a planning software like EAs to generate an optimal plan/solution. Another way is to build a knowledge base from constraints that will be communicating with the planning software. The difference between these two approaches is that in the first case, partial solutions can also be reused but the objective function should also be updated if penalty functions are being used. In the second case, partial solutions are generated incrementally that keeps rebuilding the knowledge base. Once the final solution is obtained it can be transferred to the solution viewer/constraint manager software that can be used to either generate reports or do what-if analysis based on the current solutions.

1.7. Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 is dedicated for literature review on constraint handling approaches specifically using EAs. This chapter describes pros and cons of the algorithms used for CPs. It also describes the gaps in the traditional approaches for solving CPs and challenges ahead for the research community. Chapter 3 describes the most basic form of CPs i.e. CSPs. It demonstrates how the information from the constraint can be extracted through novel evolutionary operators to better inform the search. The chapter ends with experiments on benchmark datasets and evaluation of our proposed approach. Chapter 4 mainly concentrates on COPs only where comparative study with other

approaches reported in the literature has been done. This chapter shows clear distinction of CSPs and COPs, and how they are solved together. Chapter 5 elaborates the behavior of dynamic CPs and how these problems can be solved. This chapter anchors the concept of incrementality in dynamic CPs which is equally useful for static CPs. This chapter also focuses on many other issues like diversity management and getting population out of local optimal solution. Chapter 3 – Chapter 5 are based on continuous search space only. Chapter 6 extensively analyzes CPs for discrete search space. It describes another version of ICHEA and its operators to solve discrete CPs. This chapter examines benchmark exam timetabling problem and shows comparative experimental results with other well-known approaches. This chapter also describes the generic fitness function to avoid problem dependent penalty functions. Chapter 7 concludes the thesis with some suggestions for future work. We also appended two appendices at the end of the thesis where Appendix A shows mathematical description and derivations of some proposed formulas. Appendix B describes the benchmark problems and the solutions obtained for them by ICHEA.

Chapter 2

Evolutionary Algorithms for Constraint Processing

2.1. Introduction

CPs are challenging problems that have been solved using many approaches like backtrack, iterative forward search algorithm, local search, Lagrange multiplier, and EAs (Brailsford, 1999), (Bliek et al., 2001) where EAs are acknowledged as good solvers for wide range of complex problem domains (like NP hard and NP complete) (Coello Coello, 2012). Over the past decade there is an increasing interest in solving CPs using different variations of EAs and over 1000 bibliographic references can be found in the literature ("List of References on Constraint-Handling Techniques used with Evolutionary Algorithms," 2013).

EAs are meta-heuristic algorithm based on the analogy of Darwinian evolution theory (Krasnogor and Smith, 2005). Meta-heuristics are general purpose softwares usually require much less work than developing a problem specific application that also produce optimal/near optimal solutions efficiently. This makes them a popular choice for implementation (Ólafsson, 2006). All meta-heuristics have a common approach to solve the given problem by initializing (generally randomly) a set of solutions which are then incrementally evolved using predefined heuristic principles. In each iteration, good solutions are promoted and poor solutions are discarded for the next iteration. As for EAs, it has many different variants like genetic algorithm (GA), evolutionary strategies (ES), evolutionary programming (EP) and genetic programming (GP) but all are based on common principles of Darwinian evolutionary theory of "survival of the fittest". GA encodes the candidates of the population by finite alphabets or integer values, ES and EP use real valued vectors and finite state machine respectively, and GP builds tree data structures (Eiben and Smith, 2003). The current trend has been to decrease the technical differences among these variants and refer them with a general term of EAs (Coello Coello, 2002).

```
Generate initial population
Repeat
     SELECTION of parents
     REPRODUCTION of progenies
     MUTATION of progenies
     SURVIVAL of the fittest solutions for next generation
Until termination condition is satisfied
```

Fig. 2.1. A Pseudocode for Basic EAs

Any EA includes three paradigm of evolutionary process: selection, reproduction and survival. The algorithm runs through number of generations (iterations) until the termination condition is met. The pseudocode for EAs is given in Fig. 2.1. The algorithm starts by randomly initializing the candidate solutions that go through the evolutionary process. Several techniques have been proposed in the literature for each of these processes. Some common selection process of parents can be based on simple roulette wheel or tournament selection. Reproduction process is generally carried out by crossover operators where selected parents produce progenies which go through mutation process. Crossover is explorative operator that makes a big jump to an area somewhere "in between" two parents while the mutation is exploitative that creates random small diversions to the progenies, thereby staying near the parents. Lastly in the survival of the fittest stage, only the foremost promising solutions survive to go to the next generation (Coello Coello, 2002).

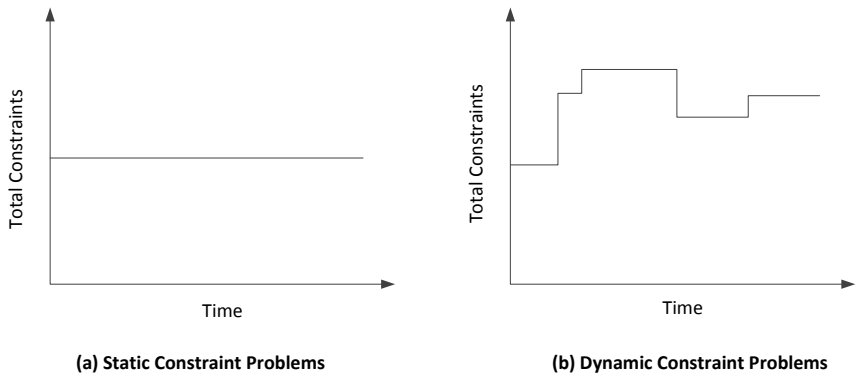


Fig. 2.2. Types of Constraint Problems

CPs can be divided into two general types: static and dynamic. In static CP constraints do not change at any stage but in dynamic CP constraints can change due to change in time, environment or user's requirement as shown in Fig. 2.2. Constraints can be added, updated or removed at any time in dynamic CPs.

2.2. Evolutionary algorithms for static constraint problems

Formally a static constraint problem (SCP) is defined by an input vector $\vec{x} = \{x_1, x_2, \ldots x_n\}$ and a set of constraints $\{c_1, c_2, \ldots c_m\}$. Each variable x_i has a nonempty domain D_i of possible values. Each constraint c_j involves some subset of the variables and specifies the allowable combinations of values for that subset. As introduced in Chapter 1, CSP is the most basic form of SCP. It can be said that the mathematical form of a CSP is a pair $\langle S, g \rangle$ where S is the search space containing each of variable x_i and its corresponding domain D_i, and g is a Boolean evaluation function to determine the feasibility of an input vector. A solution of CSP is $\vec{x} \in S$ with $g(\vec{x}) = true$. Depending on the problem the outcome can be either, one solution, some solutions or the confirmation that no solution exists. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints. If a CSP also requires a solution to optimize an objective function is called constrained optimizing problem (COP) that requires another evaluation function $f(\vec{x})$ to measure fitness (Craenen et al., 2003; Russell and Norvig, 2003). Details of these evaluation functions are given in Chapter 4.

Characteristically, the CPs solved by EAs are penalty based functions. A penalty function updates the fitness of chromosomes in EA. A penalty term is used in general for reward and punishment for satisfying and/or violating the constraints (Coello Coello, 2002). Use of penalty functions has been commonly reported in the literature for use in constrained optimization; however, it has its own advantages and disadvantages that have led to the development of different strategies. CPs are NP hard problems which can make the given problems even more complex if constraints are not handled gracefully. Still there has not been enough research focus to the development of new efficient techniques compared to the techniques proposed and advanced for unconstrained problems (Kramer, 2010). The common approaches applied in constraint handling for EAs are summarized below (Michalewicz and

Janikow, 1991; Michalewicz and Schoenauer, 1996; Coello Coello, 2002, 2012; Michalewicz and Fogel, 2004a).

2.2.1. Penalty Functions

As discussed above penalty functions have been the most commonly used approach to deal with constraint problems. Penalty functions were originally proposed by Courant in the 1940s (Courant, 1943). A penalty function updates the fitness of chromosomes in EA. A penalty term is used in general for reward and punishment for satisfying and/or violating the constraints (Coello Coello, 2002). Its aim is to decrease (*punish*) the fitness of infeasible solutions as to favor (*reward*) feasible individuals in the selection and survival processes. Its main advantage is its simplicity and compatibility for EA's objective functions. The major shortfall of the penalty function is that most of them are problem dependent that requires a careful fine-tuning of parameter to obtain competitive results (Liu et al., 2010). The penalty factors, which determine the severity of the punishment, must be set by the user and their values are problem dependent (Mezura-montes and Coello, 2006). In (Tessema and Yen, 2006), an adaptive penalty function based genetic algorithm is proposed that penalizes individuals based on ratio of total feasible and infeasible individuals present in the population. There are two types of penalty functions – *exterior* and *interior*:

1. Exterior – the EA randomly generates candidate solution in a search space which has a goal to reach towards feasible space and in case of optimization problem search for optimum solution using evolutionary operators.
2. Interior – here the initial population should be feasible solutions in case of optimization problems. The penalty value is inversely proportional to the distance of a point from the feasible-infeasible boundary. So the penalty value approaches to infinity as it moves closer to the boundary. Since this method requires initial feasible population to be generated and maintained which is computationally expensive and makes it difficult to find the schemata that will drive the population toward the optimum.

There are various forms of penalty exist in the literature, like static penalty, dynamic penalty, adaptive penalty, annealing penalty and death penalty. These penalties are defined in evaluation function of EAs that manages both feasible and infeasible functions. Death penalty is a simple and easy to use penalty function that totally ignores the infeasible solutions which generally lead to poor solutions (Michalewicz and Fogel, 2004a). Another commonly used

penalty function is adaptive penalty where penalty values are adapted to cross the feasibility boundary back and forth. This is generally done to search for solution in the boundary between the feasible and infeasible region known as strategic oscillation (Glover and Kochenberger, 1996). Normally, these are problem dependent, or require complex calculation to perform crossover and mutation to produce feasible progeny from feasible parents. Moreover, finding an initial feasible solution is itself an NP-hard problem (Coello Coello, 2002). Details of all forms for penalty functions can be found in (Michalewicz and Schoenauer, 1996; Coello Coello, 2002).

2.2.2. Repair Algorithm

It is very important to design an evaluation function that distinguishes between feasible and infeasible solutions. As far as EA is concerned it regularly produces both feasible and infeasible solutions because of its random nature. Its operators are 'blind' towards constraints i.e. knowledge from constraints are not utilized. The feasible parents not necessarily produce feasible progenies if the search space is non-convex. Hence some guidance from external source is required. Some infeasible solution may be very promising like a close neighbor to the optimal solution. Outright rejection of any infeasible solution may have drastic effect on the performance of the algorithm as it happens in "death penalty" where infeasible solutions are not tolerated at all. To avoid losing such important information repair functions are defined that is also a popular choice for EAs (Michalewicz and Fogel, 2004a). Repair function does the local search and tries to repair the infeasible solutions to make them feasible solutions. However its major weakness lies in its problem dependency and in many cases repairing an infeasible solution is itself a complex problem (Coello Coello, 2002). Some successful repair algorithms are repairGA (Nguyen and Yao, 2009) and GENOCOP III (Michalewicz and Nazhiyath, 1995).

2.2.3. Multi-Objective Optimization (MOO)

In MOO, multiple constraints are transformed into multiple objectives. Pareto-based selection approaches are currently the most popular multi-objective evolutionary algorithm (MOEA) solution technique. In a typical MOO problem there exists a set of solutions which are superior to the rest of the solution in the search space when all objectives are considered but are inferior to other solutions in the space in one or more objectives. These solutions are known as *pareto-optimal* solutions or non-dominated solutions (Srinivas and Deb, 1994). Non-dominance of a vector of multi-objective values compared to other vectors in *objective*

space is determined when each element of a vector is better or equal to other vectors. (Full definition of *pareto* concepts can be found in (Van Veldhuizen and Lamont, 2000)). There are many established algorithms like MOGA (Fonseca and Fleming, 1993), VEGA (Schaffer, 1985), NSGA and NSGAII (Deb et al., 2002) that efficiently solve the constraint problems that can be transformed into multi objective optimization problems. MOO transforms multiple constraints into multiple objective functions that are optimized simultaneously by continuously updating the generated *pareto front*.

2.2.4. Hyper-heuristics with evolutionary algorithms

Hyper-heuristics are relatively new approach proposed by (Burke, Kendall, et al., 2003). These are general systems that are able to handle a wide range of problem domains with respect to meta-heuristic technology which tends to be customized to a particular problem or a narrow class of problems (Burke, Kendall, et al., 2003). Canonical EAs generally require parameter tuning and a tailor made objective function which poses some difficulties in terms of easily applying them to newly class of problems, or even new instances of similar problems. A hyper-heuristic could be thought of as a (meta)-heuristic which operates on lower level (meta)-heuristics. A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems (Burke, Hyde, et al., 2010). Hyper-heuristics only require sequence of heuristics of the class of problem (for low level heuristic) rather than attempting to solve the problem directly. This approach is quite generic that automatically produces an adequate combination of the low-level heuristic to effectively solve the given problems. Some hyper-heuristics are also intelligent learning algorithms as they get the feedback from the search process. High level heuristic determines how to apply the low level heuristic using problem dependent local properties. (Burke et al., to appear). Hyper-heuristic with higher level as Genetic algorithm (GA) and lower level as variable neighborhood search (VNS) has been very successful for timetabling problem which is a discrete COP (Burke, Eckersley, et al., 2010).

2.2.5. Constraint guided search

Information discovery from constraints can guide the evolutionary search to improve the performance of EAs as they are ‘blind’ to constraints. Many EAs move away from problem dependent coefficient based penalty functions to generic distance function given in Eq. (4.8). However it is simply one form of a penalty function that is based on the distance of a solution from the feasible region (Michalewicz and Schoenauer, 1996; Coello Coello, 2002). The

advantage of this evaluation function is that it is generic and promotes feasible solution over infeasible one. If the problem is COP then another evaluation function is required to optimize the given problem. The former evaluation function is used for feasibility check and later one for fitness function optimization. Cultural algorithms use two subpopulations – population space and the belief space. (Ricardo Landa Becerra and Carlos A. Coello Coello, 2006) proposed cultured differential evolution (CDE) that uses differential evolution (DE) as the population space and belief space as the information repository to store experiences of individuals for other individuals to learn. (Amirjanov, 2006) proposed changing domain range based genetic algorithm (CRGA) that adaptively shifts and shrinks the size of search space of the feasible region by employing feasible and infeasible solution in the population to reach the global optimum. (Mezura-Montes and Coello, 2005) proposed simple multi-membered evolution strategy (SMES) that uses a simple diversity mechanism by allowing infeasible solutions to remain in the population. A simple feasibility-based comparison mechanism is used to guide the process toward the feasible region of the search space. The idea is to allow the individual with the lowest amount of constraint violation and the best value of the objective function to be selected for the next population. PSO-DE proposed by (Liu et al., 2010) is another algorithm based on a distance function that hybridizes particle swarm optimization (PSO) and DE to solve real-valued COPs efficiently.

2.2.6. Other Methods

(Paredis, 1994) proposed co-evolution strategies which unlike the evaluation function of penalty or repair functions, handles constraints and objective separately. It utilizes *predator-prey* model to keep two populations – one population represents solutions that satisfies many constraints while other population represents those individuals whose constraint(s) is violated by lots of individuals in the first population. Here fitness calculation is expensive as it requires historical record to compute the fitness.

Another strategy is based on special representations and operators using *decoders* technique that maps genotypes to phenotypes (Koziel and Michalewicz, 1998). Each decoder links feasible solution and a decoded solution. The idea of this method is to transform a constrained-optimization problem into an unconstrained one by adding (or subtracting) a certain value to/from the objective function based on the amount of constraint violation present in a certain solution. This strategy requires extra computational effort to find the intersection of a line with the boundary of the feasible region.

2.3. Inadequacies of Existing Approaches

Many techniques have been proposed in the literature to exploit the evolutionary search for CPs as constraints carry lots of information about the structure of the problem. Some of the techniques are problem dependent or too expensive for practical use. Constraints can reduce the search space that can make the heuristic search more efficient by searching in feasible search space only. The search space can be far too big or it may have very complex shaped feasible solutions space. It is common then to resort to some kind of heuristic approach of obtaining at least a certain level of solution quality (Burke, Kendall, et al., 2003). If the search space is non-convex then the existing evolutionary operators are merely evaluator of feasible and infeasible solutions. They do not reduce the search space but only keep the promising solutions in the pool of chromosomes. Most of the work in the literature is focused on optimizing COPs rather than making an intelligent operator to locate the feasible regions; which is quite essential if the feasible search space is very complex or negligible ($\rho \approx 0$) relative to the whole search space, where ρ is constraint strength described in Chapter 5. Even the benchmark problems mostly reported in the literature have big feasible search space shown in (Michalewicz and Schoenauer, 1996), so the difficulty of finding a CSP solution is not a challenging task. The proposed technique ICHEA tries to utilize as much information as possible from the constraints to locate the feasible regions (Sharma and Sharma, 2012a). ICHEA can also work on non-convex search space. The details of ICHEA operators are given in Chapter 3 and Chapter 6.

2.4. Evolutionary algorithm for Dynamic Constraint Problems

A DCP is a sequence of static CPs where each increment differs from the previous one by the addition or removal of some constraints. It is indeed easy to see that all the possible changes to a CP (constraint or domain modifications, variable additions or removals) can be expressed in terms of constraint updates (Verfaillie and Jussien, 2005). To solve such a sequence of CPs, it is always possible to solve each one from scratch as it has been done for the first one but this naive method, which remembers nothing from the previous reasoning, has two important drawbacks (Verfaillie and Schiex, 1994):

Inefficiency: which may be unacceptable in the framework of real time applications like planning or scheduling, where the time allowed for re-planning is limited.

Instability: of the successive solutions, which may be unpleasant in the framework of an interactive design or a planning activity, if some work has been started on the basis of the previous solution.

A DCP is a sequence of static CPs that is formed by constraint changes. The notion of DCP has been introduced to represent such situations by (Dechter, 1992). It is still need to be investigated that whether current EAs that solve SCPs efficiently can be enhanced to solve DCPs without restarting the algorithm on introduction of a new constraint. Some attempt has been made to solve DCP using EA like in (El Rhalibi and Kelleher, 2003) uses MOO to transform changes in constraints as a new objective function with changes in so called *disruption* function. This function is used to estimate the effect of changing an initial constraint to a new one. The changes are reflected in *pareto set* and program runs again to get the new *pareto optimal set* guided by previous *pareto front*.

A local search is another approach to reuse previous solutions for DCP. The previous solution can simply be used as a starting assignment for the new local search repair-based algorithm. DCP features employing the previous related CSP to find a minimal change solution to the current CSP but it can be computationally very challenging (Verfaillie and Jussien, 2005; Li et al., 2007). (Ghedira, 1994) used multi-agent approach with simulated re-annealing for DCPs. The aim was to successively solve each of these CPs with both efficiency (i.e. reuse the reasoning done before rather than starting from scratch) and stability (i.e. modify the previous solution as little as possible). The mechanisms proposed does not memorize anything except the previous partial solution (or the previous solution if any) that will be repaired. The above outlined procedure, called revision, is very difficult and computationally expensive (Coello Coello, 2002) which is still the subject of intense research in the frame of the centralized approaches.

Normally backtracking algorithm or problem dependent heuristic algorithm is used to solve DCPs like vehicle routing problem with time window (El Rhalibi and Kelleher, 2003) and flow shop rescheduling (Li et al., 2007). Backtracking algorithm tries to make a DCP more efficient by reusing the previous experience to guide the back track search. It builds the *nogoods* recording during backtrack search (Schiex and Verfaillie, 1993) but on some instances, reusing previous solutions or *nogoods* may be less efficient than solving the new problem from scratch (Verfaillie and Jussien, 2005). There have been several attempts to reuse the previous solutions for DCSP. However the revision of previous solutions is very

difficult and computationally expensive (Coello Coello, 2002) which is still the subject of intense research.

2.5. Research challenges

Since DCP can be considered as sequence of SCPs changes over time, it becomes quite essential to investigate how constraints are handled in SCPs. If there is provision in EAs to

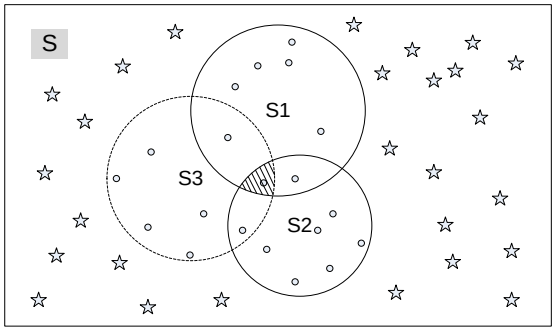


Fig. 2.3. Incremental constraint handling

incrementally take constraints without restarting the system then it becomes easier for a given CP/DCP to allow what-if analysis, what-if modeling and revision to solutions. The *incrementality* has an implicit advantage of solving dynamic CPs where constraints can be added/removed anytime by using the solutions developed to the point in time and not requiring regeneration of solutions from the scratch each time constraint(s) change. Fig. 2.3 shows constraint satisfaction set S_3 from constraint c_3 is added in a search space S that already has two static constraint satisfaction sets S_1 from constraint c_1 and S_2 from constraint c_2. The overlapping set $\{S_1 \cap S_2\}$ represents the feasible regions produced through combination of constraints c_1 and c_2. All existing solutions $\{\forall \vec{x} \in S_1 \cap S_2\}$ can be used as partial solution to locate new feasible region introduced through constraint c_3 which is $\{S_1 \cap S_2 \cap S_3\}$. In this manner the search space S is reduced after addition of a new constraint into the search space where previously found partial solutions are fully utilized without regenerating the whole solution. The reason behind this is to produce an efficient solution by keeping the previous solutions that can incorporate changes or addition to the existing constraints without enforcing EA to rebuild the solution. The system should allow what-if analysis on the current solution (without a restart) i.e. it should be intelligent enough to indicate whether the addition or update of a new constraint is resulting in a valid solution or not.

Incrementality also helps in dealing with the constraints of different strengths and preferences. These constraints can be represented in hierarchical form from the most preferred constraint to least preferred constraint. In another words it shows the "hardness" or "softness" of the constraints. There are two types of constraints: *hard* and *soft*. The hard constraints cannot be violated in under any circumstances however the soft constraints have an allowable degree of constraint violation to certain extent (Burke et al., 2007). The top levels are for hard constrains followed by soft constraints. Fig. 2.4 shows an example of constraint structure where level1 constraints has hard constraints indicated as c_{11} and c_{12}, soft constraints $c_{21}\ c_{22}\ c_{23}$ have second preference and $c_{31}c_{32}c_{33}$ are least preferred constraints. ICHEA solves the CP by incrementally taking constraints for consideration into EA in the given order of preferences. It does not incorporate constraints of lower preferences while resolving the constraints of the current preference. If a constraint of a certain level does not produce any

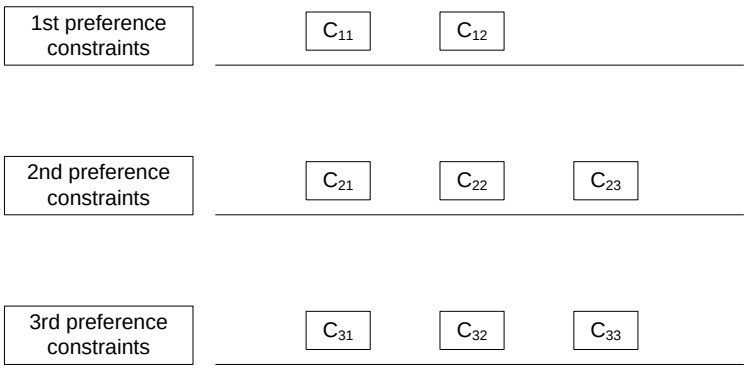


Fig. 2.4. Hierarchical constraints: downward preferences

partial solution then it can be concluded that the given constraint is not acceptable for the CP and the decision maker can get the feedback to readjust or remove this particular constraint to get the solution for the given problem. This incremental approach of creating partial solutions in every level of the hierarchy of constraints can also be used to solve DCSPs.

2.6. Summary

This chapter has briefly described commonly used constraint handling techniques with EAs. The main objective of the thesis is to utilize EAs to solve CSPs by reducing its inherent problems to handle constraints and the first major gap identified is that EAs largely ignore the hidden information from constraints. We have attempted to enhance the search operators of EA that are guided by the information from constraints to produce more efficient and robust solutions that also have relatively higher success rate in finding solutions (Sharma and

Sharma, 2012a). There are four different types of CPs as shown in Fig. 2.5. The most basic one is CSP where the objective is to find at least one feasible solution. COP and DCSP are supersets of CSP as for COP the optimum solution is to be found from that set of CSP solutions, and DCSP is a sequence of many CSPs. Finally, DCOP is the outermost superset of all other CPs as it deals with constraint satisfaction and optimization in a dynamic environment. Constraint handling by EAs for COPs has been mainly reported in the literature.

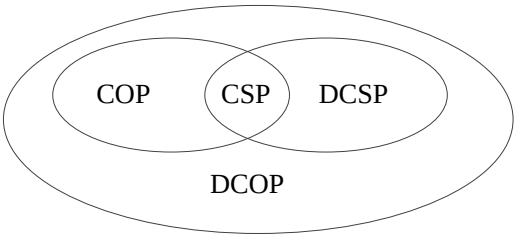


Fig. 2.5. Types of constraint problems

There is a rich set of benchmark problems for continuous COP at (Liang et al., 2006), however these benchmark problems are not ideal for continuous CSP problems as the solution for CSP can be achieved in less than a second with many EAs as discussed in later chapters. The feasible search space for the benchmark problems is supposed to be complex or negligible relative to the whole search space. Hence we used different benchmark problems from COCONUT benchmark collections available in (Shcherbina, 2011) for continuous CSPs. For discrete CSPs we only used N-Queen problem and benchmark timetabling problem for discrete COPs for the experiments. Nguyen and Yao have recently introduced some benchmark problems for continuous DCOP in (Nguyen and Yao, 2009, 2012), but again these are not ideal for continuous DCSP so we have used the same benchmark problems we had used for CSP and modified them by including a dynamic component. For discrete DCOP/DCSP we used same timetabling benchmark problems by including incrementality that mimics dynamic behavior of the problem.

Chapter 3

ICHEA for CSPs in Continuous Domain

3.1. Introduction

CSPs underpin many science and engineering applications. The CSPs are a well-known satisfiability problem that is NP-complete because of their complexity to solve the problem (Rossi et al., 1990; Craenen et al., 2003; Craenen, 2005). CSPs are the most basic form of CPs. It is an utmost necessity to provide an efficient model for constraint handling for these problems as other forms of CPs are its extension. Fig. 2.5 demonstrates a CSP's vicinity with respect to other types of CPs. As described in Chapter 1, traditional EAs are 'blind' to constraint as they do not extract enough information from the constraints but search the solution through random heuristic greedy approach (Craenen et al., 2000, 2003). This causes the search engine to spent extra computational effort in searching for the solution into the wider search space without only concentrating in the restricted smaller feasible search space. Constraints can reduce the search space and intuitively it can make the heuristic search more efficient by extracting information from constraint to guide the search engine, search in feasible search space only. The proposed model ICHEA attempts to solve CSP by utilizing as much information as possible that can be derived from constraints to guide the evolutionary search without using penalty functions and making it problem independent. It is a proposed variation of EA that does not disregard the information from constraints to produce more efficient results. This algorithm also does not require initial feasible solutions and it is also not restricted to produce feasible progenies from feasible parents. The work presented here is from our papers (Sharma and Sharma, 2012a) and (Sharma and Sharma, 2012b).

Generally, violation count is used as a fitness function for any CSP. Depending on the strengths of constraints, individual weights can be assigned to constraints in a penalty function to calculate the fitness value. To avoid problem dependent penalty functions and to utilize some knowledge from constraints to guide the evolutionary search many heuristic algorithms do not use violation count but use a distance function to indicate how far an individual is from the feasible regions (Michalewicz and Schoenauer, 1996). It transforms

constraint functions to a fitness function to rank individual chromosomes. This fitness function tries to progressively bring the chromosomes closer to the feasible space using the following function:

$$fitness_i(X) = \begin{cases} g_i(\vec{x}), & if(g_i(\vec{x}) < 0) \\ 0, & if(g_i(\vec{x}) \geq 0) \end{cases} \quad (3.1)$$

$$e = \sum_{i=1}^{m} |fitness_i(\vec{x})| \quad (3.2)$$

where $\vec{x}$ is an input vector, m is the total number of constraints and g_i is i^{th} constraint function in the form of $g_i(\vec{x}) \geq 0$ explained in Section 3.2 at Eq. (3.6) - (3.8). The fitness function $fitness_i$ is a measurement of *euclidean* distance of vector $\vec{x}$ from the nearest point of the feasible region where constraint g_i is satisfied. The error function e is the summation of all the fitness functions. The objective is to minimize the error value e. The solution to CSP is found when $e = 0$ where at least one solution is acceptable. Fitness value based on the distance from constraint satisfaction regions given in Eq. (3.1) and Eq. (3.2) produce good results and are independent of problems; however, the major drawback of such fitness functions is that they are limited to continuous domains only. They cannot be used for discrete CSPs where fitness functions typically depend on violation counts and penalty functions.

3.2. ICHEA for continuous search space

CSP is defined by an input vector $\vec{x} = \{x_1, x_2, \ldots x_n\}$ in a finite search space S where each variable x_i has a finite domain D_i. A set of constraints $\{c_1, c_2, \ldots c_m\}$ are defined in the form of functions:

$$c_i(x_1, x_2, \ldots x_n) = \begin{cases} 1, & if\ satisfied \\ 0, & if\ violated \end{cases} \quad (3.3)$$

Constraint satisfaction sets (feasible regions for each constraint) $\{S_1, S_2, .. S_m\}$ can also be defined where:

$$S_i = \{\vec{x} \in S \mid c_i(\vec{x}) = 1, i \in \{1, \ldots, m\}\} \quad (3.4)$$

A solution of a CSP is $s \in S$ when all the constraints c_i are satisfied, which can be given as:

$$\sum_{i=1}^{m} c_i(s) = m \tag{3.5}$$

For continuous CSPs numerical constraints can be given in two forms – equality and inequality (Deb et al., 2002; Tessema and Yen, 2006; Liu et al., 2010).

$$g_i(\vec{x}) \geq 0 \qquad i = 1, \ldots, k \tag{3.6}$$

$$h_j(\vec{x}) = 0 \qquad j = k+1, \ldots, m \tag{3.7}$$

The equality constraints cannot be solved directly using EAs so it is converted into inequality constraint by introducing a positive tolerance value δ.

$$g_j(\vec{x}) = \delta - \left|h_j(\vec{x})\right| \geq 0 \tag{3.8}$$

To utilize constraint satisfaction sets in an EA, a **new crossover** operator is defined in ICHEA that uses knowledge from constraints rather than blindly search for the solution. The idea of *intermarriage* is incorporated into the proposed crossover operator where both parents belong to different subpopulation sets i.e. satisfaction sets in the context of CSPs. In other words the parent vectors from different sets S_i and S_j are taken for crossover where $i \neq j$. It is also possible that a parent does not belong to any of the constraint satisfaction set i.e. $\{S - S_1 \cup S_2 \cup \ldots \cup S_m\}$. The generated offspring contains genes from both parents. The purpose is to make a "generic" offspring that tries to satisfy more than one constraint because its parents are from two different constraint satisfaction sets. The algorithm favors those offspring which satisfy more constraints by utilizing Deb's ranking scheme based on feasibility (Deb et al., 2002) to rank the population.

3.2.1. Intermarriage crossover

In *intermarriage* crossover, two parents generate two offspring. This is a dual process where both parents move closer to each other one at a time and their new positions are considered as two new offspring. An offspring from two parents through *intermarriage* is defined in a search space as a constant multiple of difference of two parent vectors as shown in Eq. (3.9). Initially offspring O_1 is placed at position $r(P_2 - P_1)$ which is then iteratively moves closer to parent P_1 until it also satisfies the constraint(s) that P_1 satisfies and similarly offspring O_2 is

designated. r is a coefficient in the range $(0, 1)$ which is generally 0.5 that gives binary traversal for convergence. The iterative move can be captured as:

$$O_1 = r^i(P_2 - P_1) \tag{3.9}$$

Variable i gets incremented from 1 to a threshold value T in the sequence $\langle 1,2,\ldots,T\rangle$. The *intermarriage* crossover process is shown in the Fig. 3.1 where ✓ mark indicates possible placement for an offspring and × mark indicate the offspring vector is unacceptable in that particular position. So using the Eq. (3.9) the next i value is used until the offspring finds an acceptable place or a threshold value T is reached.

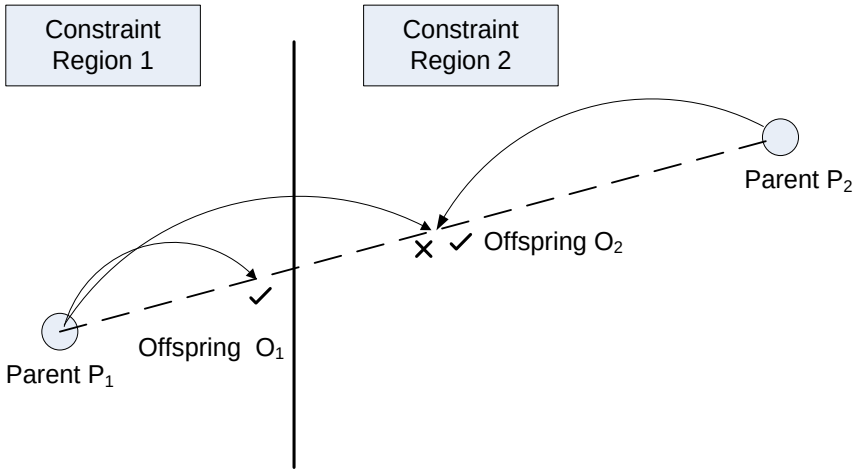


Fig. 3.1. Intermarriage Crossover

The *intermarriage* crossover would cause two selected vectors (as parents) of different constraint satisfaction sets to come closer (as offspring) towards constraint boundary because the solution space lies in the overlapping boundary region. Favoring points for *intermarriage* that satisfy more constraints results in finding solution space quickly. As *intermarriage* crossover looks for feasible solution using binary search in each side of the parents, its worst time complexity is $2\log_2 T$ which represents total number of function calls (NFC) in a worst case of an *intermarriage* crossover. We empirically determined T = 5 to 10 gives satisfactory results for our experiments. We used T = 10. Larger values like 20, 50 or 100 slow down the overall process and tend to get stuck into local optimal solution by depleting the diversity. The smaller values like 1, 2 or 3 reduce the capability of intermarriage crossover to locate overlapping regions.

It is also possible that no overlapping region exists between two parents as shown in Fig. 3.2. In this case the crossover process does not carry on to look for overlapping region instead the offspring that lies in the *hole* (O_1) is accepted. This case commonly arises during the *intermarriage* crossover. Firstly, if these holes are not accepted then no offspring will be produced. Secondly, these holes are near the boundary of feasible regions and are considered promising values as many solutions are found near boundary regions (Schoenauer and Michalewicz, 1996). It is important to keep individuals with lower fitness value of promising regions in the population to maintain the diversity (Liu et al., 2010).

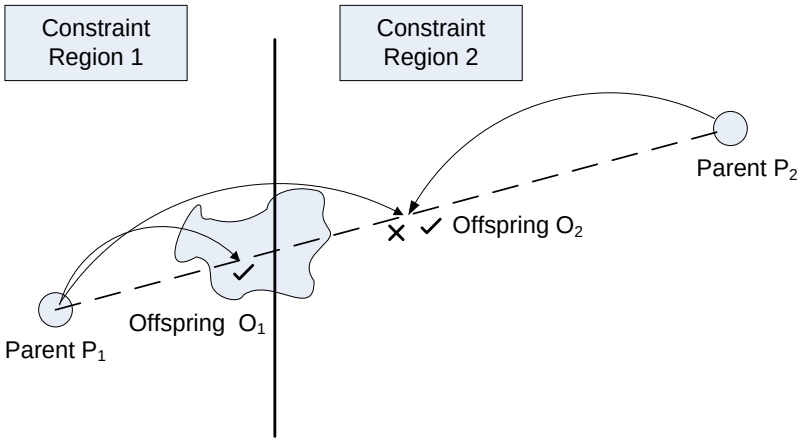


Fig. 3.2. Intermarriage Crossover with holes

This *intermarriage* crossover may converge quickly resulting in low diversity of the population. To avoid this early conversion, the concept of multi-parent crossover has been incorporated. However, rather than picking desirable parents from the population, new parents are generate in the vertexes of a hyper rectangle that encloses a parent. This hyper rectangle is dynamically created from the locations of two chosen parents P_1 and P_2 for crossover. Fig. 3.3 shows 2-dimensional search space where parents P_1 and P_2 are selected for intermarriage crossover. To make a hyper rectangle around each parent the following steps are being followed:

1. Determine the distance ΔP_{21} from P_2 to P_1, which is then multiplied by $dimMatrix$. $dimMatrix$ is a square diagonal matrix of size n which is the total dimensions of the search space. The diagonal entries are only ± 1 as shown below. $dimMatrix$ produces 2^n possible combinations of matrices where only maximum of up to 10 is chosen

randomly. An instance i of $dimMatrix$ namely $dimMatrix_i$ is chosen to create a parent P_i. Matrix multiplication of $dimMatrix_i$ and ΔP_{21} gives the distance from new parent P_i to P_1denoted by ΔP_i.

$$dimMatrix = \begin{bmatrix} \pm 1 & 0 & \cdots & 0 & 0 \\ 0 & \pm 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & \pm 1 & 0 \\ 0 & 0 & \cdots & 0 & \pm 1 \end{bmatrix}$$

$$\Delta P_{21} = (P_2 - P_1) \text{ and } \Delta P_i = \Delta P_{21} \times dimMatrix_i.$$

2. Add vector P_1 to the distance vector ΔP_i to get parent P_i:

$$P_i = P_1 + \Delta P_i$$

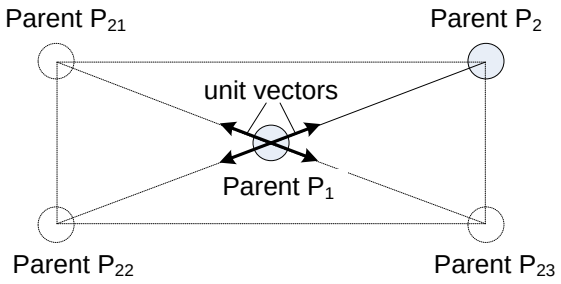


Fig. 3.3. Multi-Parents Generated from one Parent P_1

Fig. 3.3 shows parent P_1 is placed in the center of hyper rectangle. Parents P_{22} and P_{23} are generated from $dimMatrix$ $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ respectively. Parent P_1 goes through intermarriage crossover with each of these parents and then only the best offspring is selected to go into the offspring pool. This same process is repeated for the other parent P_2.

3.2.2. Arranged marriage for crossover

Intermarriage crossover requires two parents to be selected randomly from parent pool as randomness is necessary in EAs to maintain diversity. However, because of randomness of the selection process of parents many times the desired pair does not get selected or "arranged" for crossover. To encourage the selection of good mating pair that satisfies some different constraints, a repertoire τ of constraint satisfaction set of the size of the population of chromosomes is created. As soon as an individual satisfies a constraint it is added into this repertoire. τ works as a *queue* – once it is full oldest ones are removed to make room for new entries. The probability for arranged marriage is defined by a new parameter α. Some parents undergo arranged *intermarriage* crossover at the rate α with the "arranged" parents from the

repertoire τ that must satisfy at least one different constraint. We empirically determined α = 0.5 for all the benchmark problems experimented in this thesis.

3.2.3. Shrink virtual feasible region

For some problems, compared to the size of the search space the satisfaction set is relatively very small and search for a solution is difficult. So ICHEA starts with pre-defined extended virtual feasible search space of the satisfaction sets that gradually decreases to the actual size so that at least some individuals can occupy repertoire τ. It helps in locating the feasible region by narrowing down the search. In the test problems search space can be extended or reduced by changing the value of δ given in Eq. (3.8). ICHEA starts with value of δ = 10^{maxN} that gradually decreases to δ = 10^{minN}. Initially repertoire τ is divided into $(|maxN - minN| + 1)$ sections to hold constraint satisfaction sets with of δ = 10^{minN} to δ = 10^{maxN}. As soon as a section is filled with chromosomes with solutions δ = 10^{maxN} $maxN$ is decremented by 1. Eventually δ reaches to 10^{minN} when τ is completely filled. The older ones with higher value of δ are now removed to make room for the new entries that have lowest value of δ. Fig. 3.4 shows how constraint satisfaction set for a given constraint shrinks

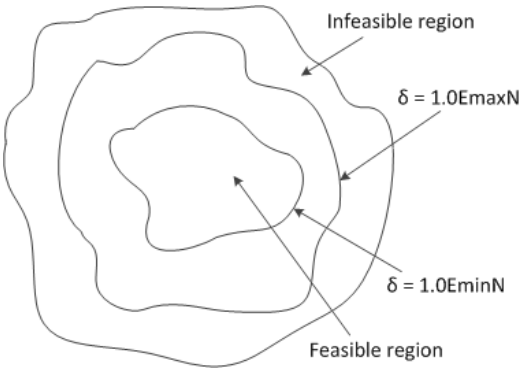


Fig. 3.4. Shrinking of the virtual feasible region of a given constraint from δ = 10^{maxN} to δ = 10^{minN} for CSPs

from δ = 10^{maxN} to δ = 10^{minN}. $\delta = 10^{maxN}$ can go beyond the given search space as δ eventually falls into the search space. ICHEA starts with random initialization of population that uses heuristic operators to move towards the feasible region by letting the closest individuals survive. Virtual feasible region for δ = 10^{maxN} provides additional tool to easily locate the promising regions that can gradually guide the evolutionary search towards the actual feasible region (δ = 10^{minN}) specified by the user. $maxN$ and $minN$ are usually 2 and -3 respectively. $maxN$ < 2 is not very effective and $maxN \geq 2$ is able to locate the feasible

regions quickly. Any value above 2 can also be chosen but it will be decremented to 2 quickly after few generations as extended virtual feasible regions are located.

3.3. ICHEA Algorithm

ICHEA is a variation of EA that adds constraint handling features to the canonical GAs. The pseudocode of ICHEA is given below in Fig. 3.5 followed by a description.

```
chromosomes = initializeChromosomes();
for each generation
  parents = NoveltyTournamentSelection();
  offspring = interMarriageCrossover(parents);
  Mutation(offspring);
  chromosomes = chromosomes + offspring;
  SortAndReplace();
  CheckTerminationCriteria();
End for loop;
```

Fig. 3.5. A Pseudocode for ICHEA

InitializeChromosomes: Chromosomes are randomly generated in a search space initially where 50% is boundary points specified by lower and upper bound of each dimension. The length of a chromosome is same as the dimension of input variables of a CSP.

NoveltyTournamentSelection: Novelty search is a new approach to heuristic search inspired by natural evolution's open-ended propensity to perpetually discover novelty. The first paper on novelty was published in 2008 by (Lehman and Stanley, 2008). It is normally used when the objective function is deceitful for example a maze problem. The novelty selection has been given higher priority than fitness value in ICHEA to increase the diversity of the search space. Novelty of an individual at a point in the search space is a measure of sparseness (σ) that is an average distance to the l nearest neighbors of that point:

$$\sigma(\vec{x}) = \frac{1}{l}\sum_{i=1}^{l} dist(\mu_i, \vec{x}) \tag{3.10}$$

where l is a fixed parameter that is determined empirically. We used $l = 4$ for our experiments. μ_i is the i^{th} nearest neighbor of $\vec{x}$ and $dist$ function determines the *euclidean* distance between μ_i and $\vec{x}$.

The tournament selection of size 2 has been used where the winner of the two chromosomes is based on the following preferences:

1. Its novelty in the search space
2. If novelty is same then higher fitness value
3. If fitness is same then a chromosome is picked randomly

InterMarriageCrossover: The crossover techniques have been described in Section 3.2.1.

Mutation: ICHEA uses polynomial mutation as described in (Deb et al., 2002) for real valued data.

SortAndReplace: The idea of Deb's ranking scheme based on feasibility (Deb et al., 2002) has been incorporated here. At the end of each generation, chromosomes are sorted based on the following order of preferences:

1. Number of satisfied constraints.
2. Fitness value which is the distance function from Eq. (3.2).

Once the chromosomes are sorted a *random death* concept is used defined in (Sharma, 2010) to delete some predefined number of chromosomes randomly from the population. The certain top percentage of the population is spared to control the search direction. If a deleted chromosome satisfies at least one constraint then it is archived into the queue called *suspension* queue that releases them back into the population after certain generations (10 generations used in the experiments) to replace other newly deleted individuals. The size of the suspension queue is same as the population of chromosomes (Sharma, 2010). This idea is similar to short term memory of *tabu* search algorithm that keeps a *tabu* list of already visited states (Glover, 1990); however, *suspension* queue does not work as an on-line memory list but only maintains previous good solutions in the population to maintain diversity.

CheckTerminationCriteria: The iteration is stopped when:

1. maximum number of generation is reached or

2. CSP solution is found or
3. process is stalled by no improvement in the solution for some generations.

3.4. Experiments

ICHEA is evaluated for continuous CSP with canonical genetic algorithm (GA), non-dominated sorting genetic algorithm (NSGA II) (Deb et al., 2002) and hybridization of particle swarm optimization with differential evolution (PSO-DE) (Liu et al., 2010) to compare its robustness and efficiency. GA has been used as-it-is without any modification or specialization to handle constraints. NSGA II solves CSPs through multi-objective optimization and it has been shown in (Deb et al., 2002) that it has an edge over other multi objective optimization algorithms to solve COPs. PSO-DE is one of the latest proposed algorithms to solve numerical COPs very efficiently. It is able to reach to optimum solutions easily for many benchmark problems (Liu et al., 2010). The code for GA is taken from genetic algorithms toolbox revision: 1.1.4.2, Matlab 7.0.1, NSGA II is taken from its author's website in (Deb, 2011) developed in C language and PSO-DE is developed in C++ language which has also been taken from its author's website in (Wang, 2012). ICHEA has been developed in Java language. All algorithms have been tested on Windows XP machine with Pentium (R) i5 CPU 2.52 GHz and 3.24 GB RAM. No parallel processing or distributed environment is used for the experiments. All of these algorithms use distance function shown in Eq. (3.2) as a fitness function without any additional penalty functions.

An average of 10 successive runs is taken into account to test the algorithms based on success rate (SR) and generation count to reach to the solution. SR is the rate of successful trials for each problem i.e. $SR = successful\ trials\ /total\ trials$. The common efficiency measurement – the number of function calls (NFC) is not used in the experiments as ICHEA is local search intensive while other algorithms are not. Table 3.1 shows the parameter

Table 3.1. Parameter Settings

Parameters	ICHEA	GA	PSO-DE	NSGA
Population Size	25	100	100	100
Crossover rate	0.8	0.8	[0.95 1.0]	0.8
Mutation rate	0.1	0.1	-	0.1
Max generations	10,000	50,000	200,000	50, 000

settings for all the algorithms. Different maximum generations have been taken as the computation speed of each algorithm varies. PSO-DE uses an additional parameter – amplification faction F that is randomly generated within $[0.9\ 1.0]$.

Seven benchmark problems from CSP domain (Shcherbina, 2011) and COP domain (Michalewicz and Schoenauer, 1996; Liu et al., 2010) have been resourced for the comparative test. Benchmark COP problems have been widely reported in the literature hence only the details of benchmark CSP problems have been given in Appendix B. Table 3.2 to Table 3.6 shows all the test results with best, median and worst solutions for each problem in terms of SR and efficiency. Columns are left blank with "−" if either it is not applicable or no good results have been obtained.

3.4.1. Benchmark problem H77 (trigonometry)

This trigonometric problem is taken from (Shcherbina, 2011). This is a hard problem where finding a CSP solution is expensive for all the algorithms except ICHEA as shown in Table 3.2. For high value of δ all algorithms except GA perform equally well, however, for low value of δ only ICHEA is able to produce good solutions. ICHEA has found a CSP solution $\vec{x} = \{1.195386, 1.229610, 1.379109, 1.496249, 0.861684\}$ for $\delta = 10^{-1}$ and $\vec{x} = \{1.182863, 1.196184, 1.395304, 1.472380, 0.597622\}$ for $\delta = 10^{-3}$.

Table 3.2. Benchmark Trigonometric Problem – H77

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	1.00	1.00	1.00	1.00
	Best	6951 gens at 69.2s	17 gens at 0.03s	136 gens at 2.4s	8 gens at 0.3s
	Median	7428 gens at 83.3s	45gens at 0.1s	136 gens at 2.4s	22 gens at 0.64s
	Worst	9532 gens at 92.2s	221gens at 0.4s	136 gens at 2.4s	48 gens at 1.53s
10^{-3}	SR	0.00 (1/5 constraint violated)	0.20	0.00 (1/5 constraint violated)	1.00

3.4.2. Benchmark problem Chem (Quadratic)

This quadratic problem has also been taken from (Shcherbina, 2011). The results obtained shown in Table 3.3 clearly shows its complexity. Again only ICHEA is able to produce some decent solutions for low value of δ. A CSP solution found by ICHEA when $\delta = 10^{-1}$ is $\vec{x} =$

{4.1853401E-9, 345.670635, 0.018339, 0.839276, 0.033066} and $\vec{x} = \{0.003262, 32.855192, 0.066828, 0.859189, 0.036943\}$ when $\delta = 10^{-3}$.

Table 3.3. Benchmark Quadratic Problem – Chem.

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	0.00 (2/5 constraint violated)	0.00 (5/5 constraint violated)	0.00 (1/5 constraint violated)	1.00
	Best	50,000 gens at 515.2s	200,000 gens at 461.0s	50,000 gens at 784.1s	54 gens at 0.83s
	Median	-	-	-	238 gens at 4.66
	Worst	-	-	-	559 gens at 11.1s
10^{-3}	SR	0.00 (5/5 constraint violated)	0.00 (5/5 constraint violated)	0.00 (4/5 constraints violated)	0.30
	Best	50,000 gens at 443.2s	200,000 gens at 508.2s	50,000 gens at 825.4s	5900 gens at 196.4s
	Median	-	-	-	-
	Worst	-	-	-	-

3.4.3. Benchmark problem Broyden10 (Polynomial)

This polynomial problem also falls from (Shcherbina, 2011). It is very hard to locate feasible solutions in this problem where none of the algorithm is able to find decent solutions for low value of δ so only the results for the high value of δ is shown in Table 3.4. ICHEA manages to find some solutions with SR of 0.80. A CSP solution by ICHEA is $\vec{x} = \{$-0.469390, -0.467320, -0.517875, -0.571916, -0.579391, -0.613264, -0.614568, -0.628205, -0.609854, -0.597433$\}$ for $\delta = 10^{-1}$.

Table 3.4. Benchmark Polynomial Problem – Broyden10

δ		**GA**	**PSO-DE**	**NSGA II**	**ICHEA**
10^{-1}	SR	0.00 (10/10 constraints violated)	0.40	0.00 (10/10 constraints violated)	0.80
	Best	50,000 gens at 419.0s	20,187 gens at 59.8s	20,000 gens at 524.0s	116 gens at 189.1s
	Median	-	40,205 gens at 114.1s	-	248 gens at 235.1s
	Worst	-	-	-	-

3.4.4. Benchmark problem HS109 (trigonometry)

This is another hard CSP problem taken from (Shcherbina, 2011) where no good solutions are observed for the original problem. Originally this problem belongs to COP domain that has been converted to CSP by introducing error value eps which is a deflection from best known optimum solution. We used the eps value of 10% rather than usual 1% to increase the size of feasibility region. The results are shown in Table 3.5 where ICHEA is able to find some solutions with SR of 0.70. ICHEA finds a solution at $\vec{x}$ = {880.638886, 913.162107, 0.013351,-0.423890, 251.978946, 249.595357, 208.083786, 364.873013, 323.539328}. Since constraint satisfaction is a subset of constraint optimization, all algorithms have been tested

Table 3.5. Benchmark Trigonometric Problem – HS109

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	0.00 (6/11 constraints violated)	0.00 (6/11 constraints violated)	0.00 (4/11 constraints violated)	0.70
	Best	50,000 gens at 371.9s	200,000 gens at 728.8s	50,000 gens at 1903.0s	53 gens at 71.0s
	Median	-	-	-	70 gens at 239.3s
	Worst	-	-	-	-

on COP benchmark problems (Michalewicz and Schoenauer, 1996; Liu et al., 2010) as well. The objective is changed to find at least one CSP solution rather than search for an optimum solution. In many of these problems finding a feasible solution or CSP solution is not a challenge. For example problem solutions for G01, … , G10 are obtained in less than 1 second except for problem G05. Hence test results of only few benchmark COP problems have been discussed in the following section.

3.4.5. Benchmark problems from COP domain

Table 3.6 shows the experimental results of three COP benchmark problems where the objective is limited to find any CSP solution. All algorithms find a feasible solution for problems G01 and G02 very quickly but for problem G05 GA and NSGA II struggle to get good solutions especially if the positive tolerance value for equality constraints (δ) is very small. ICHEA and PSO-DE both find the CSP solution very quickly. CSP solutions obtained by ICHEA for G01 and G02 are $\vec{x}$ = {0.011489,0.933194, 0.207289, 0.946219, 0.085079, 0.971863, 0.129976,0.916992, 0.702793, 0.08101, 0.195612, 0.012448, 0.193993} and $\vec{x}$ = {3.195913, 3.851266, 9.98212, 5.575662, 8.290663, 6.438110, 5.518242, 0.454351,

9.670131, 9.109863, 0.614877, 9, 9.912918, 9.925446, 9, 9, 0.184588, 1.928690, 5.703132, 0.966780} respectively. G05 solutions for $\delta = 10^{-1}$, $\delta = 10^{-3}$ and $\delta = 10^{-5}$ are $\vec{x}$ = {800.4835239, 900.4204775, 0.03422915, -0.4372399}, $\vec{x}$ = {624.191861, 1086.318629, 0.158952, -0.377384} and $\vec{x}$ = {819.310134, 881.169111, 0.021207, -0.443612} respectively.

Table 3.6. COP Benchmark Test Problems Results

Prob	δ		GA	PSO-DE	NSGA II	ICHEA
G01	-	SR	1.00	1.00	1.00	1.00
		Median	2 gens at 0.03s	5 gens at 0.016s	13 gens at 0.32s	1 gen at 0.02s
G02	-	SR	1.00	1.00	1.00	1.00
		Median	2 gens at 0.01s	1 gens at 0.03s	2 gens at 0.01s	1 gen at 0.01s
G05	10^{-3}	SR	0.00 (1/5 constraint violated)	1.00	0.00 (1/5 constraint violated)	1.00
		Best	50,000 gens at 370.8s	211 gens at 0.45s	50,000 gens at 899.1s	16 gens at 0.36s
		Median	-	343 gens at 0.77s	-	17 gens at 0.37s
		Worst	-	662 gens at 1.4s	-	17 gens at 0.37s
	10^{-5}	SR	0.00 (3/5 constraints violated)	1.00	0.00 (3/5 constraints violated)	1.00
		Best	50,000 gens at 318.6s	376 gens at 0.76s	50,000 gens at 910.2	18 gens 0.40s
		Median	-	1818 gens at 3.44s	-	19 gens 0.41s
		Worst	-	1958 gens at 4.4s	-	21 gens at 0.46s

3.5. Discussion

The experimental results for continuous CSPs are based on SR and efficiency that indicate canonical GA is generally not able to handle hard CSPs when only distance value is used as a fitness function. It must resort to some penalty functions. NSGA II also struggles in solving hard CSPs. PSO-DE generally solves a problem very quickly compared to other algorithms but only if the feasible regions are larger with $= 10^{-1}$. For hard CSPs with $\delta = 10^{-3}$ (which is generally an acceptable value) PSO-DE has low SR. Comparative results of all algorithms shows the competitiveness of ICHEA where it is able to find feasible solutions with relatively higher success rate on hard CSPs as it makes use of knowledge from constraints more effectively using a novel search operator – *intermarriage* crossover.

ICHEA does not need to have initial feasible solutions; however, it is able to maintain feasible solutions through *intermarriage* crossover in an efficient way without using any kind of repair functions. The selection of parents is based on their novelty in a search space to enforce diversity (Lehman and Stanley, 2008). The test results show that ICHEA provides a generic EA that can solve numeric CSPs efficiently without making use of problem dependent penalty functions.

3.6. Summary

This chapter has described the proposed variation of EA named ICHEA to handle CSPs. The main objective for this work is to provide some additional guidance to the EAs that are generally 'blind' towards constraints. It has been shown through benchmark problems that ICHEA outperforms standard GA, NSGA II and PSO-DE in terms of higher SR and efficiency for continuous CSPs. ICHEA can also be modeled to handle discrete CSPs. The search mechanism of ICHEA is guided by constraints where it concentrates on the feasible regions of constraint satisfaction sets to get the solution without putting extra computational effort in searching through the whole search space. The search mechanism of ICHEA is equipped with novel operators guided by constraints which concentrate on the feasible regions of constraint satisfaction sets to get a solution without putting extra computational effort in searching through the whole search space. ICHEA is intended to provide a generalized tool to solve any type of CSP. ICHEA has also been applied to other types of CPs in Chapters 4, 5 and 6.

Chapter 4

ICHEA for Constraint Optimization Problems

4.1. Introduction

It has been shown in Chapter 3 that ICHEA is able to solve real-valued CSPs. ICHEA has ability to extract and exploit information from constraints that guides its evolutionary search operators in contrast to traditional EAs that are 'blind' to constraints. Even its efficacy to solve CSPs it is important to model it to handle constraint optimization problems (COPs) as many real world CPs are in the form of COPs. This chapter proposes an enhancement to ICHEA to solve real-valued COPs. The presented approach demonstrates very competitive results with other state-of-the-art approaches in terms of quality of solutions on well-known benchmark test problems. The work presented here is mainly from our paper (Sharma and Sharma, 2012c)

4.2. Formalization of CSPs and COPs

As described in Chapter 1, constraint problems can be divided into two general classes: Constrained Optimizing Problems (COPs) and constraint satisfaction problems (CSPs). The difference between these classes is that in the first an optimal solution that satisfies all the constraints should be found, while in the second class any solution as long as all the constraints are satisfied is acceptable (Eiben, 2001a). Chapter 3 shows that ICHEA is very effective in solving real-valued CSPs; however, its ability to solve COPs was not investigated. This chapter discusses an enhancement of ICHEA to solve real-valued COPs.

A solution to real-valued COP has two folds – search for an optimum solution and satisfy all the constraints. Real-valued COP can be formulated as:

$$\text{optimize } f(\vec{x}) \tag{4.1}$$

where COP's objective function $f(\vec{x})$ has an n-dimensional input vector $\vec{x} = \{x_1, x_2, \ldots x_n\}$ that is defined in a search space S. More specifically, $\vec{x} \in \mathcal{F} \subseteq S$, where $\mathcal{F}$ being the feasible

region on the search space $S \subseteq \mathbb{R}^n$. Usually, the search space S is defined as a n-dimensional rectangle in $\mathbb{R}^n$. The domain of variables is defined by their lower bounds l_i and upper bounds u_i:

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n \tag{4.2}$$

The feasible region $\mathcal{F}$ with bounds on each dimension is further restricted by a set of m additional constraints that can be given in two relational forms – equality and inequality (Deb et al., 2002; Tessema and Yen, 2006; Liu et al., 2010).

$$g_i(\vec{x}) \geq 0 \qquad i = 1, \ldots, k \tag{4.3}$$

$$h_j(\vec{x}) = 0 \quad j = k+1, \ldots, m \tag{4.4}$$

The equality constraints $h_j(\vec{x})$ cannot be solved directly using EAs so it is converted into inequality constraints by introducing a positive tolerance value δ.

$$g_j(\vec{x}) = \delta - \left|h_j(\vec{x})\right| \geq 0 \tag{4.5}$$

A set of individual feasible regions $\{\mathcal{F}_1, \mathcal{F}_2, .. \mathcal{F}_m\}$ for each constraint can also be defined as:

$$\mathcal{F}_i = \left\{\vec{x} \in \mathcal{F} \mid g_i(\vec{x}) \geq 0,\ i \in \{1, \ldots, m\}\right\} \tag{4.6}$$

Many EAs uses a distance function as their fitness function to rank individuals. The distance function indicates how far a chromosome is from the feasible regions (Michalewicz and Schoenauer, 1996). This fitness function tries to bring the chromosomes closer to the feasible region using the following function for $\forall i : \{1 \leq i \leq m\}$:

$$fitness_i(\vec{x}) = \begin{cases} g_i(\vec{x}), & if\ g_i(\vec{x}) < 0 \\ 0, & if\ g_i(\vec{x}) \geq 0 \end{cases} \tag{4.7}$$

$$e = \sum_{i=1}^{m} |fitness_i(\vec{x})| \tag{4.8}$$

The fitness function fitness_i is a measurement of *euclidean* distance of a vector $\vec{\text{x}}$ from a feasible region $\mathcal{F}_\text{i}$. The error function e is the summation of all the fitness functions.

Minimizing the error value e leads toward a CSP solution where the objective function $f(\vec{x})$ is not needed. A solution to CSP is found when $e = 0$. To get a COP solution, CSP solutions are further processed to get optimum value of $\vec{x}$ that optimizes the objective function $f(\vec{x})$.

4.3. ICHEA for COP

ICHEA introduced in Chapter 3 is limited to works for CSPs only. We have enhanced the algorithm as below to work for COPs as well.

4.3.1. Parallel processing for CSP and COP

The foundation of ICHEA lies in acknowledging the information from the set of feasible regions $\mathcal{F}$ that guides its evolutionary search to solve CSPs effectively. To enhance its capability in solving COPs a formative approach is taken where ICHEA runs two processes in parallel – one to solve CSP and another to optimize CSP solutions. The parallel process starts by dividing the whole population pop into 2 parts. First part pop_{COP} keeps the CSP solutions that are required for optimization and the second part pop_{CSP} keeps the *good* infeasible solutions that are processed to get CSP solutions. The ratio of $pop_{COP}: pop_{CSP}$ is 1:4 for our experiments determined empirically.

pop_{CSP} is divided into equal sized m slots where slot i is allocated for individuals that violate i constraints. If there are no individuals with i violations then its allocated space is evenly distributed to other slots. This is done to keep diverse population of partially feasible solutions as (Liu et al., 2010) have observed that only keeping individuals with lower degree of constraint violations might cause the population to be trapped in a local optimum. Let pop_{CSP_i} indicate the population of individuals that violate i constraints so the total population pop_{CSP} is:

$$pop_{CSP} = \sum_{i=1}^{m} pop_{CSP_i}$$

Then pop_{CSP_i} is sorted according to the fitness and the best $|pop_{CSP}|/m$ is selected for subpopulation pop_{CSP_i}.

$$\therefore \max(|pop_{CSP_i}|) = |pop_{CSP}|/m$$

If after allocation, k slots have $|pop_{CSP_i}| < |pop_{CSP}|/m$, then unallocated population of individuals $pop_{unalloc}$ can be computed as given in Eq. (4.9).

$$pop_{unalloc} = \sum_{i=1}^{m} \begin{cases} |pop_{CSP}|/m - |pop_{CSP_i}|, & if\ |pop_{CSP_i}| < |pop_{CSP}|/m \\ 0, & otherwise \end{cases} \quad (4.9)$$

This unallocated population $pop_{unalloc}$ needs to be allocated evenly in the slots that have $|pop_{CSP_i}| > |pop_{CSP}|/m$.

4.3.2. Search focus towards best so far individual

Intermarriage crossover guides the evolutionary search to focus on feasible regions. In addition to normal *intermarriage* crossover the same parents undergo *intermarriage* crossover with a neighbor of incumbent solution to guide the search focus towards best so far individual. This is similar to PSO approach (Eberhart and Kennedy, 1995) where all swarm particles tend to move towards better positions nearby the best position that leads to optimum solution (Eberhart and Kennedy, 1995; Onwubolu and Sharma, 2004). This helps in exploring promising solution in a nearby region of the current best solution. If the *intermarriage* crossover operator is denoted by $\otimes$ then the *intermarriage* crossover initiated by parents P_i and P_j involves the following steps:

1. $P_i \otimes P_j$
2. $P_i \otimes P_{dimMatrix_i}$ where $\{P_{dimMatrix_i} \in P_{dimMatrix} | i \in randSet \wedge |randSet| = 2\}$ where $randSet = \{\exists i: 1 \leq i \leq |P_{dimMatrix}|\}$
3. $P_{neighbor_j} = \sigma(P_j + P_{best})$ where $\sigma \in (0.0,1.0)$
4. $P_i \otimes P_{neighbor_j}$

The step (1) is just a normal *intermarriage* crossover between P_i and P_j followed by step (2) that is an *intermarriage* crossover between a parent P_i and aforementioned newly created parents on the vertices of the hyper-rectangle $\forall P_{dimMatrix_i}$ (see Section 3.2.1) so that exploration is not limited to the selected population only. Step (3) determines the common neighbor $P_{neighbor_j}$of parent P_j and the current best chromosome P_{best} using a randomly generated coefficient σ in the range of (0.0, 1.0). Finally intermarriage crossover happens between P_i and $P_{neighbor_j}$ in step (4) which is inspired from PSO to search near by the current best solution. These four steps are specifically used to find the optimal solution for a COP.

4.4. Experiments

To validate the efficacy of ICHEA, 11 benchmark problems from COP domain (Michalewicz and Schoenauer, 1996; Liang et al., 2006; Liu et al., 2010) have been selected. All test problems are mathematical functions of various types like quadratic, linear, nonlinear and trigonometric. ICHEA has been compared against five highly regarded approaches as referred

Table 4.1. Parameter Settings

Parameters	ICHEA
Population size	100
Maximum generations	1.0E3
Maximum evaluations	1.0E6
Mutation rate	0.1
Crossover rate	1.0

in the Section 2.2.5: CRGA (Amirjanov, 2006), SAPF (Tessema and Yen, 2006), PSO-DE (Liu et al., 2010), CDE (Ricardo Landa Becerra and Carlos A. Coello Coello, 2006) and SMES (Mezura-Montes and Coello, 2005). No parallel processing or distributed environment is used for the experiments. An average of 10 successive runs for ICHEA is taken into account to demonstrate its solution quality against published results of other algorithms mentioned above. Table 4.1 shows the parameter settings used for all test problems. Generally, ICHEA is able to find a solution close to optimal solution in a few generations but it is allowed to run full 1.0E3 generations to try to obtain best possible solutions. For example best solutions for problem G12, G08, G11 and G01 are obtained in 10, 12, 28, 234 generations with 9.1E3, 1.1E4, 2.4E4, 2.4E5 evaluations respectively. The positive tolerance value δ for problem G03 and G11 is 1.0E-3 and 1.0E-5 respectively.

Table 4.2 shows the statistical summary of the results for all the tested problems showing best, median, mean and worst solutions obtained with their corresponding standard deviations (SDs). Table 4.3 – Table 4.5 show the same results compared with other algorithms based on best, mean and worst solutions respectively. The results in bold indicate the optimum solutions or one of the best amongst all the algorithms. ICHEA is able to reach global optimum for problems G01, G04, G06, G08, G11 and G12 while problems solutions for G02, G03, G09 is very close to optimum solutions. For problems G10 very good solutions are not observed within the limited generations. This demonstrates the competitiveness of ICHEA with other algorithms. The best results obtained by each algorithm in Table 4.3 are also

represented graphically in Fig. 4.1. The functional values are linearly transformed to error values. The error values are in scale [0 10] to measure the performance of an algorithm relative to other algorithms in one figure. The lower the error values the better the algorithm.

Table 4.2. Experimental results of ICHEA on 11 benchmark functions

Functions	Best	Median	Mean	Worst	SD
G01	**-15.00000**	-15.00000	**-15.00000**	**-15.00000**	5.4E-07
G02	-0.803036	-0.784636	**-0.768525**	-0.743884	2.3E-02
G03	-1.00497	-1.00483	**-1.00476**	**-1.00483**	1.1E-04
G04	**-30665.539**	-30665.539	-30665.537	-30665.530	3.2E-03
G06	**-6961.814**	-6961.813	**-6961.814**	**-6961.814**	1.85E-05
G07	24.6149	24.9502	25.7139	27.2705	1.0E+00
G08	**-0.095825**	-0.095825	**-0.095825**	**-0.095825**	2.3E-07
G09	680.645	680.742	680.774	680.995	1.1E-01
G10	7128.097	7165.736	7196.508	7297.964	5.8E+01
G11	**0.7500**	0.7500	**0.7500**	**0.7500**	3.2E-05
G12	**-1.00000**	-1.00000	**-1.00000**	**-1.00000**	1.2E-06

We have also taken the count of final results that are ranked in top half, achieved by all the algorithms. The last rows of Table 4.3 – Table 4.5 show the count of top ranked final results where PSO-DE, SMES and ICHEA are found to be best 3 out of 6 algorithms for getting good mean and worst solutions and PSO-DE, SMES, CDE and ICHEA are best 4 out of 6

Table 4.3. Comparison of best solutions of ICHEA with five other state-of-the-art algorithms

Functions	ICHEA	CRGA	SAPF	PSO-DE	CDE	SMES
G01	**-15.00000**	-14.9977	-15.000	-15.000000	-15.000000	-15.000
G02	-0.803036	-0.802959	-0.803202	-0.8036145	-0.803619	-0.803601
G03	-1.00497	-0.9997	-1.000	-1.0050100	-0.995413	-1.000
G04	**-30665.539**	-30665.520	-30665.401	-30665.539	-30665.539	-30665.539
G06	**-6961.814**	-6956.251	-6961.046	-6961.8139	-6961.8139	-6961.814
G07	24.6149	24.882	24.838	24.30621	24.30621	24.327
G08	**-0.095825**	-0.095825	-0.095825	-0.095826	-0.095825	-0.095825
G09	680.645	680.726	680.773	680.6301	680.6301	680.632
G10	7128.097	7114.743	7069.981	7049.248	7049.248	7051.903
G11	**0.7500**	0.750	0.749	0.749999	0.7499	0.75
G12	**-1.00000**	-1.000000	-1.000000	-1.000000	-1.000000	-1.000
Top ranked	7/11	3/11	4/11	**11/11**	**9/11**	**11/11**

algorithms for reaching towards optimum solution; however, according to "no-free-lunch" theorem no algorithm is the best for all classes of problems (Wolpert and Macready, 1997). PSO-DE is able to demonstrate very impressive results for benchmark COPs but it is not able to perform well for CSPs as demonstrated in Chapter 3 where ICHEA outperforms it in terms of success rate and efficiency.

Table 4.4. Comparison of mean solutions of ICHEA with five other state-of-the-art algorithms

Functions	ICHEA	CRGA	SAPF	PSO-DE	CDE	SMES
G01	**-15.00000**	-14.9850	-14.552	-15.000000	-14.999996	-15.000
G02	**-0.768525**	-0.764494	-0.755798	-0.756678	-0.724886	-0.785238
G03	**-1.00476**	-0.9972	-0.964	-1.0050100	-0.788635	-1.000
G04	-30665.537	-30664.398	-30665.922	-30665.539	-30665.539	-30665.539
G06	**-6961.814**	-6740.288	-6953.061	-6961.8139	-6961.8139	-6961.284
G07	25.7139	25.746	27.328	24.30621	24.30621	24.475
G08	**-0.095825**	-0.095819	-0.095635	-0.0958259	-0.095825	-0.095825
G09	680.774	681.347	681.246	680.6301	680.6301	680.643
G10	7196.508	8785.149	7238.964	7049.248	7049.248	7253.047
G11	**0.7500**	0.752	0.751	0.749999	0.757995	0.75
G12	**-1.00000**	-1.000000	-0.99994	-1.000000	-1.000000	-1.000
Top ranked	**8/11**	2/11	1/11	**10/11**	6/11	**9/11**

Table 4.5. Comparison of worst solutions of ICHEA with five other state-of-the-art algorithms

Functions	ICHEA	CRGA	SAPF	PSO-DE	CDE	SMES
G01	**-15.00000**	-14.9467	-13.097	-15.000000	-14.999993	-15.000
G02	-0.743884	-0.722109	-0.745712	-0.6367995	-0.590908	-0.751322
G03	**-1.00483**	-0.9931	-0.887	-1.0050100	-0.639920	-1.000
G04	-30665.530	-30660.313	-30656.471	-30665.539	-30665.539	-30665.539
G06	**-6961.814**	-6077.123	-6943.304	-6961.8139	-6961.8139	-6952.482
G07	27.2705	27.381	33.095	24.3062	24.3062	24.843
G08	**-0.095825**	-0.095808	-0.092697	-0.0958259	-0.095825	-0.095825
G09	680.995	682.965	682.081	680.6301	680.6301	680.719
G10	7297.964	10826.09	7489.406	7049.248	7049.249	7638.366
G11	**0.7500**	0.757	0.757	0.750001	0.796455	0.75
G12	**-1.00000**	-1.000000	-0.999548	-1.000000	-1.000000	-1.000
Top ranked	**8/11**	1/11	1/11	**10/11**	7/11	**9/11**

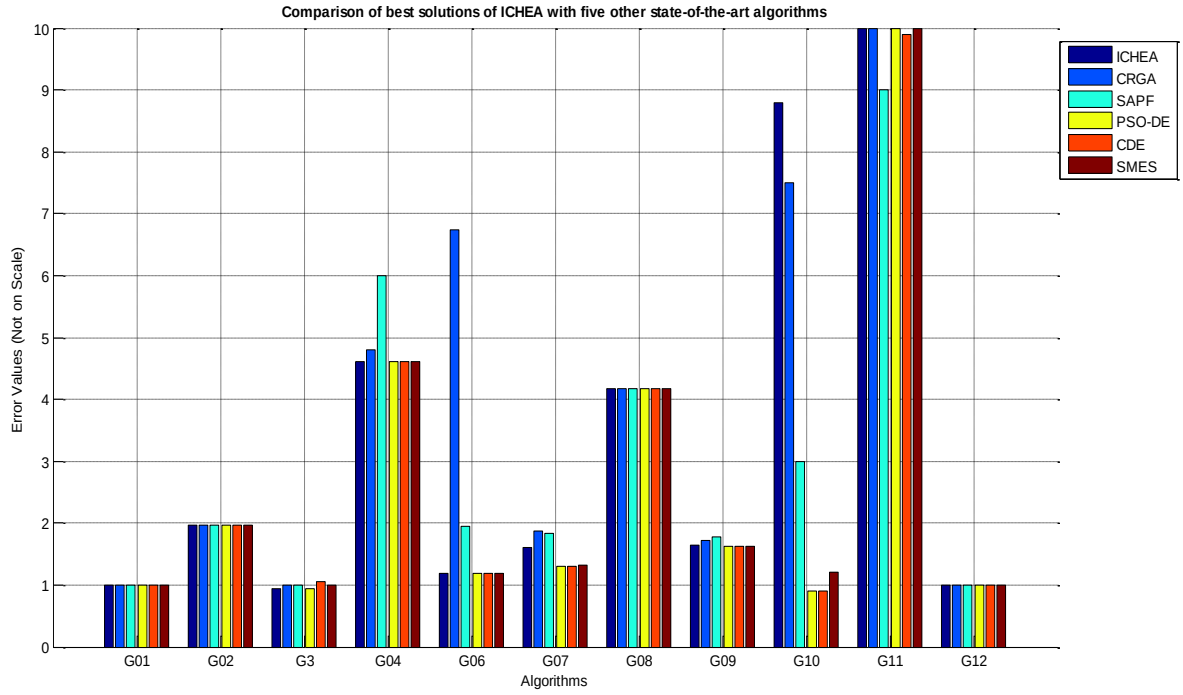


Fig. 4.1. Comparison of error values of best solutions of ICHEA with five other state-of-the-art algorithms

4.5. Discussion

ICHEA was introduced in Chapter 3. It solved real-valued CSP solutions only where it was able to outperform many other EAs in terms of success rate and efficiency. In this chapter ICHEA has been enhanced to solve COPs as well. The comparative test results on benchmark COPs are very promising and competitive with other state-of-the-art algorithms. ICHEA is a problem independent formulation where consistent results have been observed for all the test problems using common parameters. Chapter 3 also demonstrated that extracting knowledge from constraints can produce very good solutions efficiently. Hence the basic structure of ICHEA has been kept intact while enhancing it to employ constraint optimization tasks. The current form of ICHEA is still problem independent where addition of parallel processing simultaneously deals with constraint satisfaction and optimization tasks. Intermarriage crossover has been adjusted to search for an optimum solution that still utilizes information from the constraints.

4.6. Summary

ICHEA introduced in Chapter 3 has been demonstrated to outperform many well-known EAs including PSO-DE to solve benchmark CSPs. ICHEA has been enhanced in this chapter

without losing its integrity to solve real-valued COPs which has shown a competitive results. This new ICHEA runs in two parallel processes – one for CSP and another for COP. The CSP process searches feasible regions to make a population of feasible solutions while COP process tries to optimize the solutions using the whole population. The main idea remains the information extraction from constraints that reduces the search space to promising regions only. However, this form of ICHEA is restricted to solve only real-valued CSP and COP but it has been extended to work for discrete constraints problems in Chapter 6 as the basic idea remains extracting information from constraints. Chapter 5 involves enhancement of ICHEA for dynamic CSPs and COPs in continuous domain.

Chapter 5

Incrementality in ICHEA

5.1. Introduction

Many real-world constrained problems have a set of predefined static constraints that can be solved by evolutionary algorithms (EAs) whereas some problems have dynamic constraints that may change over time or may be received by the problem solver at run time so that a solution found so far is not valid later. Some techniques are focused on repairing the original constraints while others are focused on finding solutions that are able to absorb these changes (Barták and Salido, 2011). Recently there has been some interest in academic research for solving continuous dynamic constraint optimization problems (DCOPs) where some new benchmark problems have been proposed (Nguyen and Yao, 2009). ICHEA has been demonstrated in Chapter 3 and Chapter 4 to be a versatile and constraints guided EA for continuous CPs which efficiently solves static CSPs and COPs. This chapter shows the improvements in efficiency and SR of ICHEA by an enhancement of the algorithm to deal with local optima obstacles and diversity management. The enhancement also includes a capability to handle dynamically introduced constraints without restarting the whole algorithm that uses the knowledge from already solved constraints using an incremental approach. Experiments on benchmark CSPs adapted as dynamic CSPs have shown very promising results. We also investigate efficiency of ICHEA in solving benchmark DCOPs and compare and contrast its performance with other well-known EAs. This chapter focuses on handling continuous DCSPs and continuous DCOPs using ICHEA with incrementality that does not require the solutions to be regenerated when a new constraint or set of constraints are added, updated or removed. Most of the work presented here is from our papers (Sharma and Sharma, 2012d, 2012e)

CSPs are at the core of many real-world applications, including control, and diagnosis of physical systems such as car, planes, and factories. It is also used in modern robotic systems such as control of modular, hyper-redundant robots, which are robots with many more degrees of freedom than required for typical tasks. Sometimes the environment of the CSPs changes

along with time as in obstacle avoidance, vehicle routing and reusing previously generated university timetable. The strength of ICHEA is that it makes use of knowledge from constraints rather than blindly search in the solution space as done by traditional EAs. ICHEA has been demonstrated in Chapter 3 to outperform other well regarded EAs like NSGA II (Deb et al., 2002) and PSO-DE (Liu et al., 2010). However it also exhibited drawbacks in solving hard CSPs where it was computationally expensive as it generally requires more than 200 seconds of CPU time to solve benchmark CSPs on a standard standalone machine. Its SR was also very low in the range of 0.0-0.6 for hard problems. Hence we have introduced some new strategies to improve ICHEA to address these drawbacks. Moreover, ICHEA described so far is only able to handle static CSPs in its current state so we propose an enhancement to the ICHEA to realize *incrementality* in constraint solving. Using this approach the dynamic behavior of CSPs can be handled efficiently as it is quintessential for real-time DCSPs where only little research has been reported using EAs. Furthermore, there are not any established benchmark problems for them. Some benchmark problems are compiled in (Nguyen and Yao, 2012) and (Karimi et al., 2012) but they are used for DCOPs and dynamic optimization problems respectively. The difference between static/dynamic constraint optimization and constraint satisfaction is that in first an optimal solution that satisfies all the constraints available at that particular time should be found, while in second any solution as long as all the constraints available at the given time are satisfied is acceptable (Eiben, 2001b). Because of the unavailability of benchmark DCSPs we have transformed some existing benchmarks CSPs from (Shcherbina, 2011) to DCSPs. The performance of existing ICHEA is enhanced to solve CSPs now called ICHEA+ (ICHEA–plus) to distinguish from previously proposed ICHEA in Chapter 3. Solving real-valued DCSPs using an incremental approach is called IICHEA (incremental ICHEA).

CPs are generally in the form of COPs that drives the research focus towards developing the models to handle COPs. A major question raises here is whether all the constraint handling approaches for static COPs are applicable for DCOPs as well (Nguyen and Yao, 2009). This question has not been addressed extensively in the literature especially for real-valued DCOPs where benchmark problems were also unavailable until Nguyen and Yao has introduced some problems in (Nguyen and Yao, 2009, 2012) together with a penalty function based novel algorithm repair genetic algorithm (RepairGA) to solve these problems efficiently. (Richter, 2010) proposed memory design to solve DCOPs. Memory design is traditionally used to solve unconstrained dynamic optimization problems where the usual practice is to set aside a

memory space to hold some promising individuals from the population that replaces other poor performing individuals when the change in environment is detected (Richter, 2010). There are also two canonical EAs namely hyper-mutation genetic algorithm (hyperM) and random-immigrant genetic algorithm (RIGA) that are based on "introduce diversity" and "maintain diversity" strategies respectively to solve DCOPs. We have used the same benchmark problems to test the performance of ICHEA against RepairGA, hyperM, RIGA and canonical genetic algorithm (GA). We used the same ICHEA customized for COPs in Chapter 4 with added features of improved ICHEA i.e. ICHEA+ elaborated in this chapter.

5.2. Formalization of real valued DCSPs

Generally violation count is used as a fitness function for any CSPs. Depending on the strengths of constraints, individual weights is assigned to constraints in a *penalty* function to calculate the fitness value. Many EAs do not use violation count but use a distance function to indicate how far an individual is from the feasible regions, to utilize some knowledge from constraints to guide the evolutionary search rather than using problem dependent *penalty* functions (Michalewicz and Schoenauer, 1996). The inequality constraint functions can be transformed into a fitness function to rank individual members in the population generated by ICHEA. This fitness function given in Eq. (3.1) and Eq. (3.2) tries to bring the individuals closer to the feasible space. For DCSPs the total number of constraints m is not know a priori and the solution has to be produced based on constraints that come to hand. A DCSP can be defined as a sequence of static CSPs where each one differs from the previous one by the addition or removal of some constraints. It is indeed easy to see that all the possible changes to a CSP (constraint or domain modifications, variable additions or removals) can be expressed in terms of constraint additions or removals (Verfaillie and Jussien, 2005). The same fitness function given in Eq. (3.1) and Eq. (3.2) are used for DCSPs. To solve such a sequence of CSPs, it is always possible to solve each constraint from scratch as it has been done for the first one but this naive method, which remembers nothing from the previous reasoning, is inefficient and instable. Its inefficiency in terms of handling constraints for real time applications (like planning or scheduling), where the time allowed for re-planning is limited. The solution can also be regarded as instable for the framework of an interactive design or a planning activity, if some work has been started on the basis of the previous solution which is desirable but cannot be reused (Verfaillie and Jussien, 2005).

5.3. Formalization of real-valued DCOPs

A solution to real-valued static COP has two folds – search for an optimum solution that also must satisfy all the constraints. The fitness function of COPs has been described in Eq. (4.7) and Eq. (4.8). For DCOPs the total number of constraints m is not know a priori and the solution has to be produced based on constraints that come to hand where the constraints and objective functions can change over time. Hence the fitness functions for static COP given in Eq. (4.7) and Eq. (4.8) has to be transformed into dynamic COP by making it time dependent by introducing a parameter for time *t* as given in Eq. (5.1) and Eq. (5.2) below.

$$fitness_i(\vec{x}, t) = \begin{cases} g_i(\vec{x}, t), & if\ g_i(\vec{x}, t) < 0 \\ 0, & if\ g_i(\vec{x}, t) \geq 0 \end{cases} \tag{5.1}$$

$$e(t) = \sum_{i=1}^{m(t)} |fitness_i(\vec{x}, t)| \tag{5.2}$$

where $g_i(\vec{x}, t)$ is inequality constraint function at time t that delivers fitness $fitness_i(\vec{x}, t)$. Its dynamic CSP solution with total number of constraints $m(t)$ at time t is given by $e(t)$.

To determine the performance of an algorithm at time t is usually measured by an offline performance measure described in (Branke and Schmeck, 2003; Wang et al., 2007) which is an average fitness error between the optimal fitness of the current environment and the best-of-generation fitness at each generation. The average fitness error ($error(t)$) at time t can be calculated as:

$$error(t) = \frac{1}{t} \sum_{i=1}^{t} \left| f_{optimal} - f_{best}(t) \right| \tag{5.3}$$

where $f_{optimal}$ is the known optimal fitness and $f_{best}(t)$ is the fitness of the best solution achieved at generation t. (Nguyen and Yao, 2012) has modified this offline performance that demonstrates the performance based on "good feasible solution" rather than any good solution that may be infeasible. This measure is always greater than or equal to zero. If in any generation there is no feasible solution, the worst possible value that a feasible solution can have is taken for $f_{best}(t)$; however, this can be incomputable for some hard problems (for example problems discussed in Chapter 3) where any feasible solution may not be found for the entire generations. Fortunately, all of these benchmark problems in (Nguyen and Yao, 2012) are able to find at least one feasible solution easily before the change in environment. We used both of these offline errors for our experiments and to differentiate these

measurements the first one is called *traditional performance measure* and later one is called *feasibility performance measure*. Feasibility performance measure is more applicable for CPs as it takes feasibility into account because infeasible solutions are unacceptable.

5.4. Benchmark problems

As mentioned above there is little research reported on real-valued DCSPs nor there is any benchmark problems available for it. There are some benchmark problems for dynamic optimization problems in (Li et al., 2008) and (Karimi et al., 2012) that are without constraints. Some recently developed EAs have performed well on these benchmark problems like self-adaptive differential evolution algorithm (jDE) (Brest et al., 2009), dynamic hybrid particle swarm optimization (DHPSO) (Karimi et al., 2012) and triggered memory based particle swarm optimization (TMPSO) (Wang et al., 2007). Nguen and Yao in (Nguyen and Yao, 2012) has introduced some benchmark problems for real-valued DCOPs and a novel algorithm repair genetic algorithm (RepairGA) to solve these problems efficiently that are used in the experiments. However, none of benchmarks are for DCSPs so we took some benchmark CSPs from coconut benchmark (Shcherbina, 2011) and converted them to DCSP by taking one constraint at a time that is solved as a sequence of static constraints. Only addition of constraints is considered in making a dynamic environment for DCSPs. Update of constraints or redefinition of feasible regions are simply removal and re-addition of constraints. A new constraint is added into the environment in every 100 generations or else if all the current constraints are satisfied – whichever is earlier.

5.5. Enhancement to ICHEA for CSPs

ICHEA is a variation of EA that uses its own crossover operator namely *intermarriage* crossover described in Chapter 3, selects two parents from different *constraint satisfaction sets* S_i to make them come closer iteratively towards their corresponding feasible boundary because the CSP solutions lie in the overlapping boundary region of feasible regions that satisfy different constraints. Favoring individuals that satisfy higher number of constraints and the use of feasible regions for *intermarriage* crossover guide the evolutionary search in finding the solution space quickly. This guiding process has helped ICHEA to outperform other well-known EAs to solve CSPs where constraint strengths are very high i.e. the feasible regions are very small compared to the whole search space. Calculation for constraint strengths has been shown in Section 5.6.

As mentioned in Section 5.1, even after ICHEA's success in outperforming other well-known EAs to solve CSP, it is still computationally expensive as its solutions for some benchmark problems generally require more than 200 seconds of a CPU time on a standard machine to produce a solution. Its SR is also very low in the range of 0.0-0.6 for some hard benchmark problems. Hence the following enhancement is proposed that improves its performance in terms of efficiency and SR. ICHEA+ has shown an improvement as high as 68 times over the previous ICHEA on benchmark problems. ICHEA+ is even able to produce efficient solutions with high SR of up to 1.00 on low positive tolerance value ($\delta = 10^{-3}$) on hard CSP problems where previous ICHEA had low success with SR = 0.00.

5.5.1. Diversity Management

As mentioned in Chapter 4 the lower the individuals' degree of constraint violation, the higher the probability that it clusters together around the current best solution and individuals with lower degrees of constraint violations are very difficult to jump out of current best solution's adjacent region. This may cause the current best solution to stay on the same position for a long time leading to loss of diversity in the population. To avoid this scenario ICHEA+ keeps the fair share of all degrees of constraint violating individuals in the population as it is implemented for COPs in Chapter 4. The whole population pop is divided into equal sized m slots where slot i is allocated to subpopulation pop_i that contains individuals violating i constraints. After allocating k slots if $|pop_i| < |pop|/m$, then unallocated population of individuals $pop_{unalloc}$ is:

$$pop_{unalloc} = \sum_{i=1}^{m} \begin{cases} |pop|/m - |pop_i|, & if\ |pop_i| < |pop|/m \\ 0, & otherwise \end{cases} \tag{5.4}$$

This unallocated population $pop_{unalloc}$ needs to be allocated evenly in the slots that have $|pop_i| > |pop|/m$.

5.5.2. Stalled local optimal solutions management

The above diversity management is not sufficient to avoid the population getting stuck into local optimal solution for hard CSPs. This is a common problem for EAs when the whole population gets stuck around local optimal solution and lose its diversity. We introduce the concept of *forced constraint violations* to tackle this issue. This works like *tabu* search algorithm where the individuals try to move away from the forcibly introduced new infeasible regions (*tabu* regions). If the population is stagnant for certain number of generations then the

current best solution is considered as local optimal solution where some region around it is marked as a new infeasible region to move the population away from it. This region is defined as a hyper-sphere whose centre is the location of the current best (local optimal) solution with the radius defined as distance from the location of current best individual with the location of the worst individual that has the same degree of violations as the current best individual. If the current best individual belongs to a subpopulation pop_i which is sorted according to the fitness from best to worst where an individual can be described as $\vec{x}_j \in \{pop_i | 1 \leq j \leq |pop_i|\}$ has best individual $\vec{x}_1$ and worst individual $\vec{x}_{|pop_i|}$. The radius R of the hyper-sphere can be

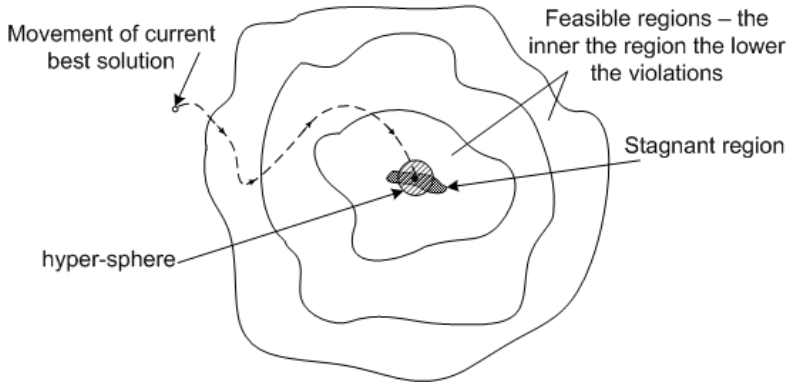


Fig. 5.1. Making hyper-sphere around stalled local optimal solution

computed as: $R = \left|\vec{x}_1 - \vec{x}_{|pop_i|}\right|$ and hence the new forced dynamic constraint is: $g_{m+1}(\vec{x}_j) = \sum_{i=1}^{n}(x_i - \mu_i)^2 > R^2$ where $\mu_i \in \vec{x}_1$ and $x_i \in \vec{x}_j$. Fig. 5.1 demonstrates the movement of the current best individual that starts from high violation regions towards low violation regions until it is trapped in a stagnant region which is then referred as stalled local optimal solution.

5.6. Experiments on benchmark DCSPs

ICHEA is a problem independent tool to solve any given n dimensional CSP so we use the following parameters to solve all the problems:

Stall threshold = 12 generations, crossover rate = 1.0, mutation rate = 0.1, maximum generation = 1000 and $|pop| = \begin{cases} 25 & , n > 6 \\ 100, & 1 \leq n \leq 6 \end{cases}$

As one constraint is considered at a time for DCSP, we would also like to see if the constraint strengths of individual constraint matters in finding an efficient solution. Hence two different sequences of static CSPs are used where each constraint is added into the environment – from lowest to highest strength and vice versa. As described in (Liu et al., 2010) constraint strength (ρ) is computed *offline* by using the formula $\rho = |\Omega|/|pop|$, where $|pop|$ is the number of solutions randomly generated from pop, $|\Omega|$ is the number of feasible solutions out of these $|pop|$ solutions. In the experimental setup, $|pop|$=10,000 and ρ value is computed as the average of five successive runs.

It has been demonstrated in Chapter 3 that ICHEA outperforms all other tested EAs where other tested EAs have very low SR. Hence we are only providing the results of ICHEA+ with previously introduced ICHEA. ICHEA has been developed in Java language and the tests have been carried out on the same Windows XP machine with Pentium (R) i5 CPU 2.52 GHz and 3.24 GB RAM. No parallel processing or distributed environment is used for the experiment. An average of 10 successive runs is taken into account to test the algorithms based on SR and generation count to reach to the solution. SR is the rate of successful trials for each problem i.e. $SR = successful\ trials\ /total\ trials$. Nine test cases have been created using the benchmark problems from CSP domain (Shcherbina, 2011) and COP domain (Michalewicz and Schoenauer, 1996; Liu et al., 2010). Table 5.2 and Table 5.6 show constraint strengths for problems *H77* and *HS109* respectively. Other problems have same constraint strengths for all the constraints with $\rho = 0$. The ρ values are sorted in both ascending and descending order for separate tests where constraints are incrementally added to the search space in that order. As described earlier DCSPs are basically sequence of static constraints that are incrementally added to the search space. Table 5.1, Table 5.3, Table 5.4, Table 5.5 and Table 5.7 show CSP test results for problems H77, Chem, Broyden10, HS109 and COP – G05 respectively. Each table results show test results of ICHEA+ with previous ICHEA to compare their performances on different benchmark problems. The results of IICHEA (both ascending(↑) and descending(↓) order of ρ) have also been shown on the same tables to compare the results of ICHEA solving both static CSPs and dynamic CSPs as it is important to show whether information from already solved constraints has been utilized or not. The test results are shown with best, median and worst solutions for each problem in terms of SR and efficiency. Columns are left blank with "–" if either it is not applicable or no good results have been obtained. The last column named *imp* shows the improvement of the ICHEA+ over ICHEA; the *imp* values for best, median and worst solutions indicate how

many times the ICHEA+ is better than ICHEA and the *imp* values for SR indicate the increase in SRs from ICHEA to ICHEA+. Each benchmark problem has been tested on two different δ values $\{10^{-1}, 10^{-3}\}$ except for problem G5 which only uses $\delta = 10^{-5}$.

Fig. 5.2 – Fig. 5.5 depict the average of all test runs to compare the performances of ICHEA+ and IICHEA. The y-axis shows the error value given in Eq. (3.2) and x-axis shows the number of generations. The graphical image shows the progress of ICHEA in solving CSPs and DCSPs. The spikes in the graphs for IICHEA indicate that a new constraint has been introduced into the search space and spikes for ICHEA+ indicate a virtual feasible region has been shrunk which results in violation of additional constraints as described in Section 3.2.3. This causes the error value of current best individual to increase as ICHEA+ favors individuals with less constraint violations. Detailed experimental analysis for each test problem has been given next.

5.6.1. H77 (Trigonometry)

The test result for *H77* benchmark problem (Shcherbina, 2011) shown in Table 5.1 has been elaborated below:

Table 5.1. Benchmark trigonometry problem H77

δ		ICHEA+	IICHEA ($\rho \uparrow$)	IICHEA ($\rho \downarrow$)	ICHEA	imp
10^{-1}	SR	1.00	1.00	1.00	1.00	0.0
	Best	5 gens at 0.6s	6 gens at 0.7s	9 gens at 1.1s	8 gens at 0.3s	0.5
	Median	8 gens at 1.2s	6 gens at 0.7s	11 gens at 1.5s	22 gens at 0.64s	0.5
	Worst	13 gens at 2.0s	8 gens at 0.9s	13 gens at 2.0s	48 gens at 1.53s	0.8
10^{-3}	SR	1.00	1.00	1.00	1.00	0.0
	Best	7 gens at 0.91s	8 gens at 1.0s	20 gens at 3.4s	447 gens at 19.0s	20.9
	Median	16 gens at 2.3s	28 gens at 4.4s	41 gens at 6.8s	3250 gens at 113.7s	49.4
	Worst	21 gens at 3.1s	37 gens at 5.8s	66 gens at 11.3s	6297 gens at 211.4s	68.2

Improvement for CSPs (ICHEA+ vs ICHEA): The improvement column indicate there is not much improvement for $\delta = 10^{-1}$ for ICHEA+ over ICHEA as it is already an easy problem to solve with SR of 1.00 that takes less than 2.0 sec of computational time. Massive improvement can be seen for $\delta = 10^{-3}$ where median solution for ICHEA+ is close to 50 times better than ICHEA.

Handling DCSP (ICHEA+ vs IICHEA): The constraint strengths for all 3 constraints of this problem has been shown in Table 5.2, and Table 5.1 shows the increasing order of ρ is twice as good as its decreasing order for $\delta = 10^{-3}$. Results for *ICHEA+* and IICHEA are more or less same as Fig. 5.2 shows that solutions for both static and DCSPs converge nearly at the same time.

Table 5.2. Constraint Strengths for H77

Constraints	ρ
1	6.33E-01
2	6.14E-01
3	6.33E-04

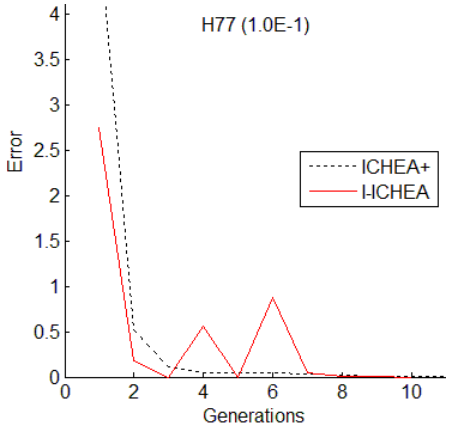


Fig. 5.2. IICHEA and ICHEA+ comparison for H77 ($\delta = 10^{-1}$)

5.6.2. Chem (Quadratic)

The test result of *Chem* benchmark problem (Shcherbina, 2011) in Table 5.3 has been explained below:

Table 5.3. Benchmark Quadratic Problem Chem

δ		ICHEA+	IICHEA	ICHEA	imp
10^{-1}	SR	1.00	1.00	1.00	0.0
	Best	11 gens at 1.8s	19 gens at 2.7s	54 gens at 0.83s	0.5
	Median	26 gens at 4.03s	25 gens at 3.5s	238 gens at 4.66s	1.2
	Worst	37 gens at 6.7s	33 gens at 4.8s	559 gens at 11.1s	1.7
10^{-3}	SR	1.00	1.00	0.30	0.7
	Best	75 gens at 7.3s	34 gens at 5.4s	5900 gens at 196.4s	26.9
	Median	621 gens at 108.3s	267 gens at 45.7s	-	3.1
	Worst	740 gens at 122.9s	291 gens at 49.6s	-	2.7

Improvement for CSPs (ICHEA+ vs ICHEA): The improvement column indicates there is not much improvement for $\delta = 10^{-1}$ as both ICHEA+ and ICHEA have shown similar results; however, major improvement can be seen for $\delta = 10^{-3}$ where SR has been increased by 70%.

Handling DCSP (ICHEA+ vs IICHEA): The constraint strengths for all the constraints are 0 for this problem. Fig. 5.3 shows IICHEA starts converging after 50 generations where ICHEA+ is still converging slowly after 100 generations.

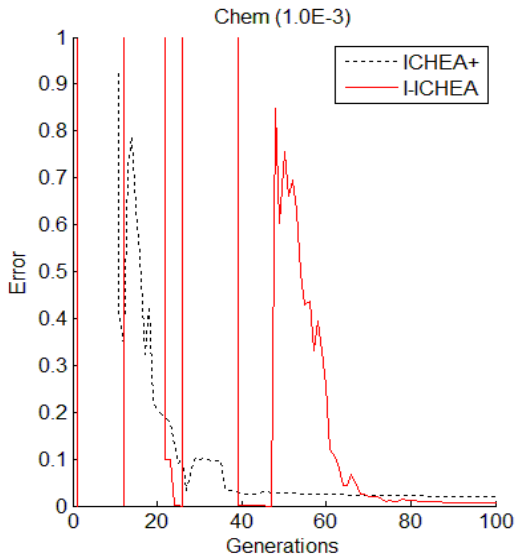


Fig. 5.3. IICHEA and ICHEA+ comparison for Chem ($\delta = 10^{-3}$)

5.6.3. Broyden10 (Polynomial)

Table 5.4 shows the test result of *Broyden10* benchmark problem (Shcherbina, 2011) that has been explained below:

Table 5.4. Benchmark Polynomial Problem Broyden10

δ		ICHEA+	IICHEA	ICHEA	imp
10^{-1}	SR	1.00	1.00	0.80	0.2
	Best	22 gens at 39.6s	31 gens at 45.0s	116 gens at 189.1s	4.8
	Median	29 gens at 55.8s	49 gens at 81.7	248 gens at 235.1s	4.2
	Worst	53 gens at 182.1s	158 gens at 300.0s	-	5.5
10^{-3}	SR	1.00	1.00	-	1.0
	Best	28 gens at 51.8s	36 gens at 59.4s	-	19.3
	Median	42 gens at 85.3s	38 gens at 74.0s	-	11.7
	Worst	79 gens at 269.3s	174 gens at 385.3s	-	3.7

Improvement for CSPs (ICHEA+ vs ICHEA): The improvement column indicates that the SR has been increased by 20% and efficiency by 4-5 times for $\delta = 10^{-1}$. Massive improvement can be seen for $\delta = 10^{-3}$ where SR is increased by 100%.

Handling DCSP (ICHEA+ vs IICHEA): The constraint strengths for all the constraints are 0 for this problem. Fig. 5.4 shows that IICHEA stalls at error value 2.0 after 45 generation to solve the last constraint while ICHEA+ converges without stalling.

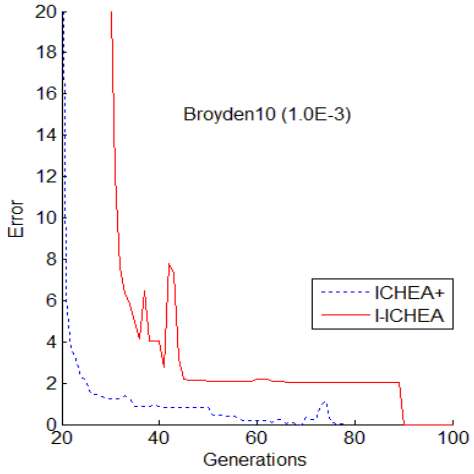


Fig. 5.4. IICHEA and ICHEA+ comparison for Broyden ($\delta = 10^{-3}$)

5.6.4. HS109 (trigonometry)

HS109 (Shcherbina, 2011) is a hard benchmark problem. Table 5.5 shows its test results that

Table 5.5. Benchmark Trigonometric Problem HS109

δ		ICHEA+	IICHEA ($\rho\uparrow$)	IICHEA ($\rho\downarrow$)	ICHEA	imp
10^{-1}	SR	0.70	0.70	0.70	0.70	0.0
	Best	54 gens at 81.1s	57 gens at 87.7s	59 gens at 79.8s	53 gens at 71.0s	0.9
	Median	131 gens at 205.0s	113 gens at 183.0s	66 gens at 92.1s	70 gens at 239s	1.2
	Worst	192 gens at 361.3s	186 gens at 283.6s	150 gens at 208.6s	-	2.8
10^{-3}	SR	0.10	0.80	0.80	0.0	0.8
	Best	133 gens at 204.7s	100 gens at 155.0s	89 gens at 128.4s	-	4.9
	Median	-	122 gens at 186.0s	125 gens at 183.2s	-	0.0
	Worst	-	156 gens at 250.5s	151 gens at 230.6s	-	0.0

Table 5.6. Constraint Strengths for HS109

Constraints	ρ
1	0
2	0.87536
3	0.87662
4	0
5	0
6	0
7	0.00004
8	0.9241
9	0.44
10	0.41442
11	0.41874

have been explained below:

Improvement for CSPs (ICHEA+ vs ICHEA): The improvement column indicates there is not much improvement for $\delta = 10^{-1}$, however, massive improvement can be seen for $\delta = 10^{-3}$ if IICHEA is compared with ICHEA instead of ICHEA+ where SR has been increased by 80%.

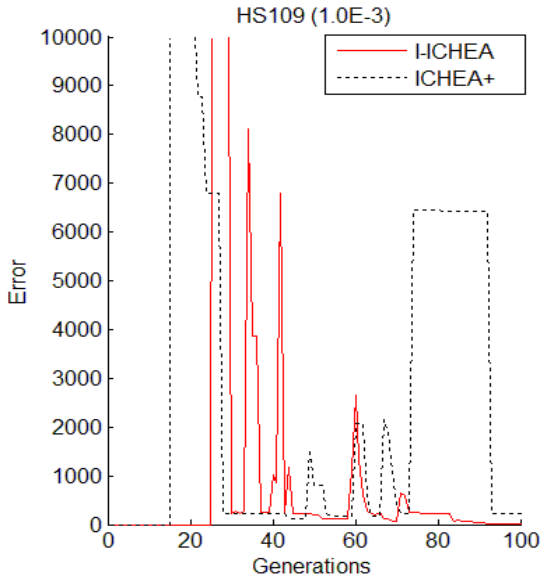


Fig. 5.5. IICHEA and ICHEA+ comparison for HS109 ($\delta = 10^{-3}$)

Handling DCSP (ICHEA+ vs IICHEA): The constraint strength for all 11 constraints of this problem has been shown in Table 5.6, and Table 5.5 shows the decreasing order of ρ has produced slightly better results than increasing order in terms of efficiency. Fig. 5.5 shows IICHEA starts converging after 75 generations where ICHEA+ still stalls after 90 generations.

5.6.5. G05 (COP benchmark)

Since CSPs are a subset of COPs we used some benchmark problems from COP domain (Michalewicz and Schoenauer, 1996; Liu et al., 2010) as well where the objective is changed to find at least one CSP solution rather than search for an optimum solution. In many of these problems finding a feasible solution or CSP solution is not a challenge. For example problem solutions for G01, … , G10 are obtained in less than 1 second. Hence we only picked G05 for our experiment. The test results are shown in Table 5.7 where no improvement has been observed as this problem is very easy for CSP domain that has SR of 100% with less than 1.0 second to reach to the solution.

Table 5.7. COP Benchmark problem G05

δ		ICHEA+	I-CHEA	ICHEA	imp
10^{-5}	SR	1.00	1.00	1.00	0.0
	Best	26 gens at 0.52s	29 gens at 0.53s	18 gens 0.40s	0.77
	Median	30 gens at 0.57s	34 gens at 0.62s	19 gens 0.41s	0.72
	Worst	39 gens at 0.72s	36 gens at 0.64s	21 gens at 0.46s	0.64

5.7. Experiments on benchmark DCOPs

As mentioned in Section 5.4, Nguyen and Yao have proposed some new benchmark problems in (Nguyen and Yao, 2009) which we will be using to compare the performance of ICHEA with aforementioned dynamic constraint handling algorithms RepairGA, RIGA, hyperM and

Table 5.8. Properties of Benchmark Problems (Nguyen and Yao, 2009)

Problem	objFunc	Constr	DFR	SwGO	GIB	NAO
G24_0	Cyclic	No	-	-	Yes	No
G24_1	Cyclic	Fixed	2	Yes	Yes	No
G24_3	Fixed	Linear	1-3	Yes	Yes	Yes
G24_4	Cyclic	Linear	1-3	Yes	Yes	No
DFR	Number of disconnected feasible regions					
SwGO	Global opt. switches among disconnected regions					
NAO	Newly appearing optima without changing existing optima					
GIB	Global optimum is in the boundary of feasible area					

GA. The description of test problems is given in Table 5.8 where some problems have been omitted because of their common properties with the listed ones and unavailability of published results. The paramter settings used for all the benchmark problems are same as in the published results in (Nguyen and Yao, 2009) except for problem G24_3 which has a typographical error for variable $S_2(t)$ which should be $S_2(t) = 2 - t\frac{x_2max - x_2min}{S}$. Parameter settings for RepairGA, RIGA, hyperM and GA are also same as in (Nguyen and Yao, 2009). ICHEA customized for COPs in Chapter 4 has been incorporated with additional enhancements proposed for ICHEA+ in Section 5.5, is used for the experiments. ICHEA is a problem independent tool and it does not require any penalty function. We used the following parameters to solve all the benchark problems:

Population = 100, crossover rate = 1.0, muation rate = 0.1, $|P_{dimMatrix}|$ = 2.

Table 5.9. Traditional Performance Measure

Algorithms	Error	StDev	vsGA	Error	StDev	vsGA
	G24_0 (dynF + noC)			G24_1 (dynF + fixC)		
ICHEA	0.0051	0.004	88.44	0.0333	0.005	24.35
RepairGA	0.2531	0.026	1.77	0.0448	0.009	18.13
RIGA	0.2854	0.043	1.57	0.5734	0.076	1.42
HyperM	0.2660	0.012	1.69	0.6472	0.271	1.25
GA	0.4488	0.049	-	0.8117	0.077	-
	G24_3 (fixF + dynC)			G24_4 (dynF+dynC)		
ICHEA	0.0187	0.003	52.24	0.0799	0.006	11.07
RepairGA	0.0148	0.002	66.11	0.0695	0.009	12.72
RIGA	0.6664	0.063	1.46	0.5937	0.054	1.49
HyperM	1.1079	0.482	0.88	2.3370	1.942	0.38
GA	0.9760	0.127	-	0.8842	0.081	-

Table 5.10. Feasibility Performance Measure

Algorithms	Error	StDev	Error	StDev
	G24_0 (dynF + noC)		G24_1 (dynF + fixC)	
ICHEA	0.005	0.004	0.033	0.005
RepairGA	0.468	0.059	0.226	0.024
RIGA	0.131	0.034	0.401	0.046
HyperM	0.173	0.042	0.450	0.094
GA	0.214	0.037	0.587	0.085
	G24_3 (fixF + dynC)		G24_4 (dynF+dynC)	
ICHEA	0.019	0.003	0.066	0.009
RepairGA	0.116	0.008	0.211	0.015
RIGA	0.340	0.045	0.492	0.071
HyperM	0.461	0.104	0.494	0.039
GA	0.384	0.092	0.627	0.045

An average of 10 successive runs for ICHEA is taken into account to demonstrate its solution quality against published results of above mentioned algorithms in (Nguyen and Yao, 2009). The problem environment changes after every 1000 evaluations as in (Nguyen and Yao, 2009) which is approximately 40 generations. To compare the performance of ICHEA with other algorithms we used both performance measures given in Eq. (5.3). The experimental result of the traditional performance measure is given in Table 5.9 and feasibility performance measure is given in Table 5.10. Both tables show the error value as their performance value

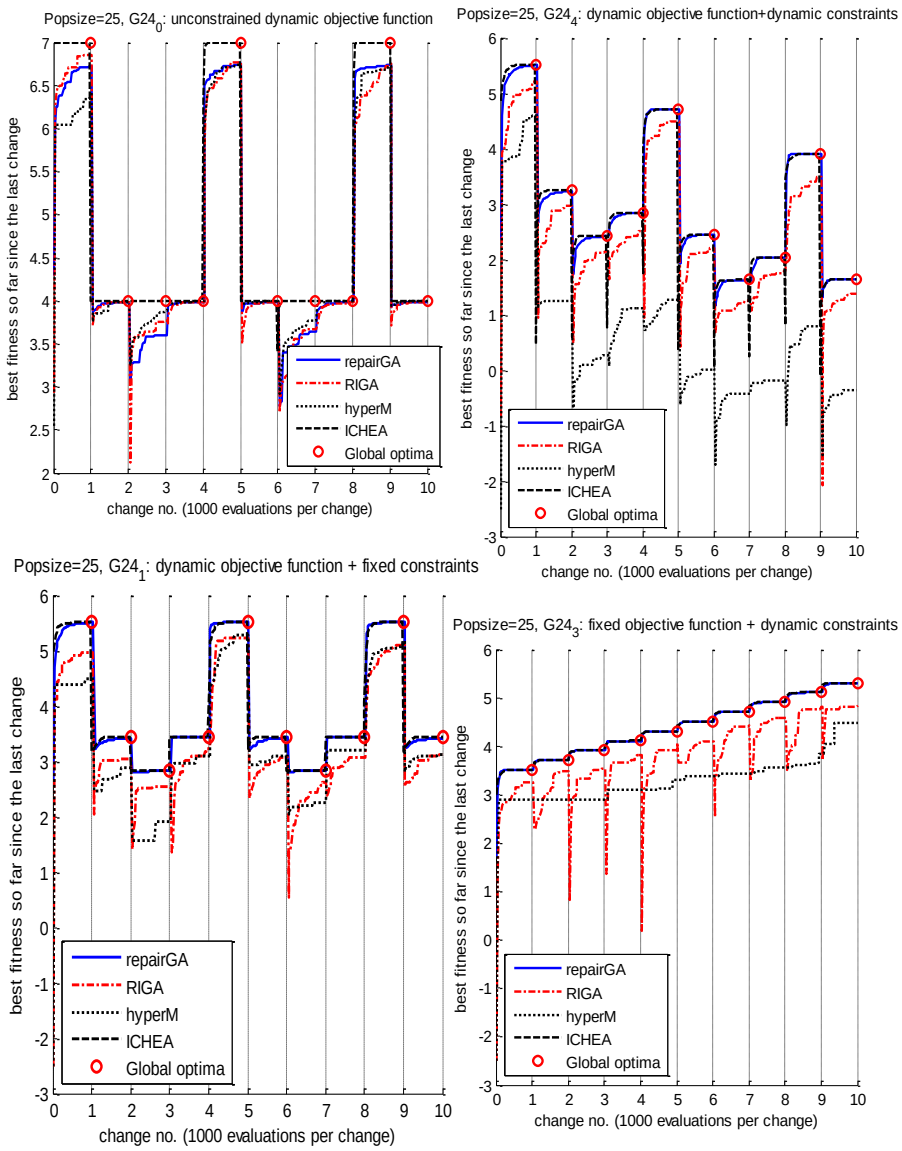


Fig. 5.6. Plots of current best solution on each generation for repairGA, RIGA, hyperM and ICHEA

with standard deviation ($StDev$) and comparison measurement with GA ($vsGA$) that indicates how many times the tested algorithms is better than GA. Table 5.9 shows ICHEA has outperformed other algorithms on test problem G24_0 and G24_1, and showed similar performance with RepairGA for G24_3 and G24_4 based on traditional performance measurements. Table 5.10 shows the feasibility performance measure which is more applicable measurement in the context of constrained problems as infeasible solutions are not taken into account where ICHEA has completely outperformed all other algorithms on all the tested problems. Plots of the current best individuals for every generation for all the algorithms are shown in Fig. 5.6.

5.8. Discussion

The experimental setup in Section 5.6 has dual objectives. Firstly, it demonstrates the comparative study of previously reported ICHEA in Chapter 3 with an upgraded ICHEA that applies some new strategies proposed in Section 5.5.1 and Section 5.5.2. Secondly, it establishes that ICHEA is able to handle dynamic constraints in an incremental manner by reusing knowledge from previous increments. The experimental results show that the addition of diversity management and stalled local optimal solutions management has improved the performance of ICHEA to solve CSPs. Previously introduced ICHEA in Chapter 3 has very low SR for many benchmark problems when δ is 10^{-3} because of local optimal solutions that makes the whole population become stagnant that has been massively improved for problems – *HS109*, *Broyden10* and *Chem*. ICHEA's efficiency has also been improved massively at different rates for all the hard problems except *G05* which is a simple problem in the perspective of CSPs. The second objective of the experiment is to show if IICHEA can perform similar to ICHEA+ where the experimental results show that IICHEA has not only performed similar to ICHEA+ but has outperformed it for problems – *HS109* and *Chem*. This demonstrates that ICHEA makes full use of constraints solved in previous increments that enables it to handle dynamic constraints as well. Dynamically added constraints to ICHEA give the solution with same efficiency and success rate as of solving all the constraints concurrently. The experimental results on the order of constraint strength did not produce any conclusive results about the performance of ICHEA as shown in problems – *H77* and *HS109* where results with mixed success have been observed.

The experimental outcome for ICHEA on benchmark DCOPs is also very promising. All the tested benchmark problems are unique in nature in the context of DCOPs. G24_0 objective function changes over time that does not have any constraint, G24_1 also has dynamic objective function but fixed constraints. The constraints of G24_3 and G24_4 change over time but G24_3's objective function is fixed while G24_4's objective function is dynamic. The performance comparison based on traditional performance measure shows ICHEA has outperformed all other algorithms where problems have no constraints or constraints are fixed. For dynamic constraints ICHEA has produced similar performance as of RepairGA; however, this traditional measurement does not reflect the quality of solutions in terms of their feasibility. Hence feasibility performance measure is more applicable when there are constraints in the problem. The test results based on feasibility performance measure shows

that ICHEA has clearly outperformed all other algorithms because ICHEA's main strength is that it makes use of knowledge from constraints rather than blindly search in the solution space as traditional EAs do (Sharma and Sharma, 2012a, 2012c, 2012d). Fig. 5.6 and Table 5.10 show ICHEA is not only able to immediately recover from change in environment but also produces good quality feasible solutions. As demonstrated in Chapter 4, ICHEA for COPs runs two parallel processes internally – one to keep looking for feasible solutions and another to optimize the existing feasible solutions. Any disruption in the search space invokes ICHEA to prioritize search for feasible solutions through its *intermarriage* crossover that results in high quality feasible solutions. Another advantage of ICHEA is that it does not use problem dependent or problem class dependent penalty functions as RepairGA uses penalty value of 2.5 for these benchmark problems.

5.9. Summary

ICHEA is a versatile EA to solve various types of real-valued constraint problems. It has been demonstrated in Chapter 3 and Chapter 4 that it performs well for benchmark CSPs and COPs. This chapter has demonstrated the capabilities of ICHEA to handle dynamic constraints for DCSPs and DCOPs. The shortcoming of ICHEA introduced in Chapter 3 has been addressed and some improvements have been proposed to cater for local optimal solution and maintain the diversity. We also proposed the provision of handling the constraints incrementally which enables solving DCSPs effectively. It has been shown through benchmarks problems that the new strategies applied to ICHEA helps in maintaining the diversity of the populations and dealing with local optimal solutions by dynamically creating new constraints. This has helped massively in getting higher SR for most of the test problems. ICHEA has also been tested to handle DCSPs on benchmark CSPs that have been transformed to DCSPs. It has been shown that constraints can be added dynamically to ICHEA without restarting the algorithm and it can still give the solution with similar efficiency and SR as of solving all the constraints concurrently because ICHEA utilizes the knowledge from constraints of previous increments. The experimental results on the order of constraint strengths have been inconclusive in finding a CSP solution in an incremental approach. For future work efficiency of ICHEA can be tested on dynamic constraints where previous constraints can be removed or updated that distort the previous feasible regions.

The second half of the experiments evaluates the performance of ICHEA on benchmark DCOPs where it has outperformed other state-of-the-art EAs on the scale of feasibility

performance measurement; however, there is need to diversify the benchmark problems with high dimensional problems as current problems are limited to two dimensional only. An advantage of ICHEA over other EAs is that it is a problem independent constraint handling EA that utilizes knowledge from constraints without using any repair or penalty functions. It extracts and exploits knowledge from constraints for its evolutionary search and is independent of the characteristics of the problems.

Chapter 6

Constraint Handling for Discrete Search Space

6.1. Introduction

Many real world CSPs have constraints defined as qualitative data. The qualitative data are generally represented in a discrete form. These CSPs have discrete search space S made up of individual points. For example a particular course is scheduled on Monday at a particular room for a timetabling problem. The search space consists of all the possibilities of feasible solutions as discrete points with nothing in between them. Searching for solution requires "hopping" from one point to the other unlike continuous search space where the locality of a point, its neighbors and the boundary for feasible regions can be identified with mathematical formulations as given in Eq. (3.1) (Onwubolu, 2002). For this reason generally the fitness function for discrete CSP is weighted penalty function based on violation counts as shown below in Eq. (6.1).

$$f(\vec{x}) = \sum_{i=1}^{m} w_i c_i(\vec{x}) \tag{6.1}$$

where $w_i \geq 0$ are the weighted coefficients representing the relative importance of the constraints. Input vector $\vec{x}$ and the *boolean evaluation function* $c_i(\vec{x})$ have been defined in Section 3.2. Its main weakness is the difficulty to determine the appropriate weights when there is not enough information about the problem (Coello, 1998). The solution of a CSP is $\vec{x} \in S$ when all the constraints are satisfied as given in Eq. (3.5). We used N-Queen problem for the experiments as this is one of the simplest form of discrete CSP where $\forall w_i = 1$. Some of the contents of this chapter are from our papers (Sharma and Sharma, 2012b, 2013).

6.2. Intermarriage crossover for discrete CSPs

Intermarriage crossover defined in Section 3.2.1 for continuous search space is not applicable for discrete search space as its formulation is based on real numbers for continuous domain. The concept of *intermarriage* crossover is to fuse feasible solutions from two different constraint satisfaction sets together that makes the offspring "generic" that satisfy more

constraints because its parents are from two different constraint satisfaction sets. If the fusion of two discrete feasible solutions is represented by $\oplus$ then the *intermarriage* crossover of two parents for discrete CSP transformed from Eq. (3.9) can be given as:

$$O_i = r^i\left(P_j \oplus P_i\right) = (P_j \oplus P_i) \tag{6.2}$$

For discrete *intermarriage* crossover value of r and i is 1 because fusion is non-iterative as shown in Fig. 6.1 where offspring are accepted if fusion results in better chromosome(s). ICHEA uses variable length chromosomes (partial solutions) to accomplish discrete valued *intermarriage* crossover where genotype is used as phenotypes. Variable length chromosome

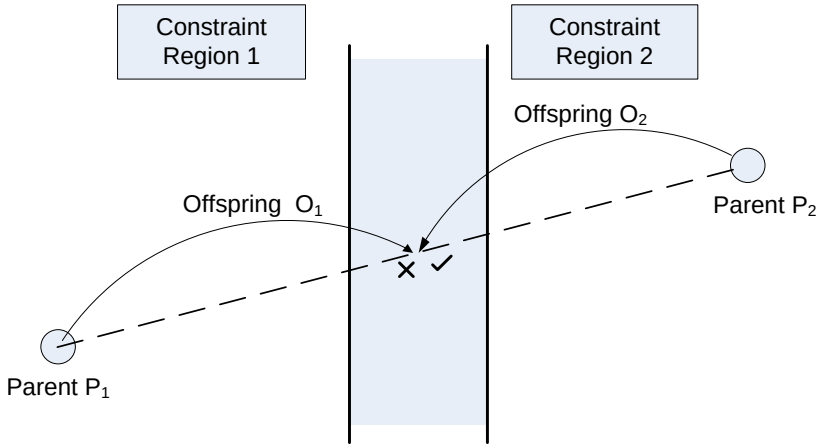


Fig. 6.1. Intermarriage crossover for discrete CSPs

has been used in many applications like (Bandyopadhyay and Pal, 2001; De Weck and Kim, 2004; Sharma and Omlin, 2009). Partial solutions are feasible solutions when only a subset of constraints has been considered. A CSP in n dimensional search space has input vectors $\vec{x} = \{x_1, x_2, \ldots x_n\}$ with m constraints. If only k constraints have been satisfied or required to be satisfied then the partial solution $p = \{\overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_k}\}$ represents a set of satisfied constraints where $|\overrightarrow{c_i}| = n_i$ for $\forall i \in \{1, \ldots, k\}$ with $n_i \leq n$ and $k \leq m$. Each constraint can have different input variables $\{x_a, \ldots, x_b\}$ where $a, b \leq n$. Its cost and fitness function for a CSP can be given as:

$$cost(p) = n - |p| \tag{6.3}$$

$$fitness(p) = n - |p| \tag{6.4}$$

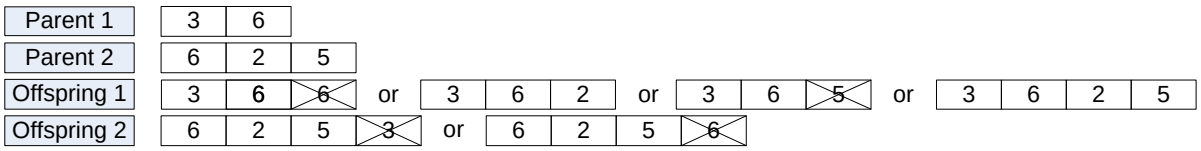


Fig. 6.2. Variable length intermarriage crossover

A partial solution p_i takes part in *intermarriage* crossover by attempting to allocate constraints $\{\overrightarrow{c_a}, \ldots, \overrightarrow{c_b}\}$ from another partial solution p_j into its existing feasible space (pattern of data points). To elaborate more we use an example of an N-Queen problem. N-Queen problem is a classic CSP that can be expressed as placing N queens on N x N chessboard such that no queen should be attacked by one another (Letavec and Ruggiero, 2002). An N-Queen problem has one dimensional constraints $c_i = x_1$ for $\forall i \in \{1, \ldots, m\}$ where i represents the column and the value of x_1 represents the row of the chess board. Suppose N-Queen problem of size 6 has two parents P_1 and P_2 with partial solutions $\langle 3, 6\rangle$ and $\langle 6, 2, 5\rangle$ respectively. The values represent constraints in the given order like parent P_1 satisfies two constraints where first value 3 represents first queen is at column 3 and first value 6 represents second queen is at column 6. The *intermarriage* crossover only tries to append/fuse the allele values of one parent into another one at a time until no more allele values is left. All the allele values that violate the constraints are dropped so the offspring are also feasible chromosomes. The generated offspring from these parents either satisfy equal or more constraints as shown in Fig. 6.2 where offspring $O_1{:}\langle 3, 6, 6\rangle$ has two conflicting queens in row 6, offspring $O_1{:}\langle 3, 6, 5\rangle$ has again two conflicting queens in row 6 and row 5 attacking each other diagonally. Offspring $O_1{:}\langle 3, 6, 2, 5\rangle$ has no conflicts with the fusion. Each offspring has traits from both parents. An advantage of using variable length chromosome in this manner is reduction in computational time.

Intermarriage crossover avoids recalculation of objective function because it only requires allele values to be appended. For example an N-Queen problem on chess board of size N requires $N(N+1)/2$ operations on a single function call and for one complete crossover it requires $N(N+1)$ operations. Its time complexity of *Big-O* order is $O(N^2)$ on average. On the other hand, the *intermarriage* crossover only checks the violation of appended allele value with all other existing feasible values that requires $(l_1 + l_2 + N) + N(l_1' + l_2')$ operations

where l_1 and l_2 are the lengths of partial solutions of the parents and l'_1 and l'_2 are length of their respective non-duplicate allele values. The formulation of the time complexity is given in Appendix A. The first expression of time complexity $(l_1 + l_2 + N)$ indicates number of operations required to find the duplicate values. The second expression $N(l'_1 + l'_2)$ indicates the operations required to append the non-duplicate allele values to each other parents. The best time complexity is $(1 + 1 + N) + N(1 + 1) = 3N + 2$ operations when lengths of both parents are 1 and the worst time complexity is $(N/2 + N/2 + N) + N(N/2 + N/2) = N^2 + 2N$. As the evolutionary search progresses the length of partial solution increases towards maximum and length of non-duplicate allele values decreases to minimum. The generic time complexity can be written as $\big((N - \gamma) + (N - \gamma) + N\big) + N(d)$ where γ is an integer value between $[1, N - 1]$ that decreases from $N - 1$ to 1, and d is an integer value between $[2, N]$ that decreases from maximum to minimum non-duplicate value, as the evolutionary search progresses. ICHEA struggles and spent most of its computational time as the length of partial solutions increases which causes the constraints to become more intense. So the average computational activity occurs when the length of partial solutions increases that in turn decreases the length of non-duplicated allele values. In other words average time complexity is computed when $\gamma \to 1$ and $d \to 2$. The generic time complexity can be rewritten as $N(d - 2\gamma + 1)$ so $\lim_{d\to 2} \lim_{\gamma\to 1} N(d - 2\gamma + 1) = N$ as d and γ are both small integer values approaching towards respective minimum. Hence the average time complexity has the *Big-O* order of *O(N)*.

6.3. Solving discrete COP

So far we have seen constraints as a form of a mathematical function, which are required to be satisfied under any circumstances to have an acceptable solution. Such constraints are known as "hard constraints". Solutions, which satisfy all the hard constraints, are often called "feasible" solutions. In addition to the hard constraints there are usually various constraints that are considered to be desirable but not essential. These are often called "soft constraints" (Burke, Bykov, et al., 2003). Soft constraints can have some degree of satisfaction or order of preferences for a particular problem as shown in Fig. 2.4. Soft constraints can be represented by penalty functions for COPs where higher weights demonstrate lower preferences and vice versa for higher preferences. However the common problem of a penalty function as described in Section 2.2.1 is its dependency on the problem and difficulty in finding good weight factors like choosing or fine tuning number of parameters. Usually cost based weights

are used in penalty functions which are problem dependent to cater for preferences of the constraints. For example, in a university timetabling problem, Prof. X might prefer teaching in the morning whereas Prof. Y prefers teaching in the afternoon (Russell and Norvig, 2003). The most basic form of a fitness function shown in Eq. (6.5) is given in terms of violation count where constraint strengths or degree of violation has not been considered (Eiben et al., 1998; Eiben, 2001a):

$$f(v) = \sum_{p=0}^{D} v_p \tag{6.5}$$

where function $f(v)$ is the fitness value for $v = \langle v_0, \dots, v_D \rangle$ satisfied constraints. D is the lowest preference defined (highest numeric value for preference) which is one less than total number of preferences and v_p is total number of satisfied constraints with p_{th} preference. If a cost function is desired instead of fitness function then violation count $(L - v_p)$ can be used instead of v_p where L is the total constraints of a given problem. This is the most basic form of fitness function that only sums up the number of different constraint but does not rank them according to preferences. Eq. (6.5) can be modified to take preferences/strengths into account in the following function:

$$f(v) = \sum_{p=0}^{D} w_p v_p \tag{6.6}$$

where a problem dependent weight factor w_p is used to give higher strengths to more preferred constraints.

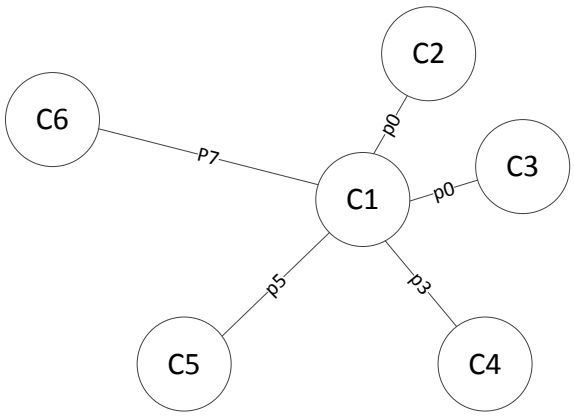


Fig. 6.3. Constraint satisfaction with preferences in respect to other constraints

In order to make a generic fitness function, the constraint strengths (preferences) and their relation with other constraints need to be considered. A constraint in a search space can have multiple degrees of constraint satisfaction relative to other constraints. A feasible region in a continuous search space is simply an overlapping region between constraints; however the overlapping region in discrete search space can represent the degree of satisfaction from first preference to last preference. Fig. 6.3 shows constraint *C1* has different degrees of satisfaction with other constraints. The *intersection* of *C1* with *C2* and *C3* results in constraint satisfaction with preference *p0*, constraint *C4* with preference *p3* and so on. If a generic fitness function is desired to maximize higher preferences then Eq. (6.7) ensures that constraints of higher preference are always given priority where l_p is total constraints of p_{th} degree of satisfaction and β is a constant value more than the maximum possible value that any l_p can hold. Fig. 6.3 has $l_0 = 2, l_1 = 0, l_2 = 0, l_3 = 1$ and so on. A generic value for β is L^2, however any smaller problem dependent maximum value can also be used as exponential component in the function limits the number of preferences as long as maximum possible number is not reached for a computer program. This equation also supports incrementality by reusing partial solutions that take consideration of higher preferences initially to build solutions that is followed by accommodation of remaining constraints of lower preferences. This equation gives higher fitness to the solution where constraints are solved with higher preferences.

$$f(l) = \sum_{p=0}^{D} l_p(\beta + 2)^{D-p} \tag{6.7}$$

This equation encourages in maximizing l_p for higher preferences, however, if the problem requires minimization of l_p for lower preferences then the fitness function would be:

$$f(l) = \sum_{p=0}^{D} (L^2 - l_p)(L^2 + 2)^p \tag{6.8}$$

where $\beta = L^2$ is used. The proofs of Eq. (6.7) and Eq. (6.8) are given in Appendix A. Note that these fitness function are only applicable for single objective COPs and not for multi objective COPs. MOO uses *pareto* front to achieve the same results where decision makers can pick the results from their choice, not necessarily the one with the least number of constraint satisfactions with lowest preferences (Quan et al., 2007; Dong et al., 2012).

So far couple of fitness functions are defined for discrete COPs but how these functions can be utilized in EA/ICHEA is not discussed. Local search and hyper-heuristics are frequently

used to solved these kinds of problems (Russell and Norvig, 2003; Burke, Eckersley, et al., 2010; Demeester et al., 2012). *Intermarriage* crossover described for discrete search space in Section 6.2 is only applicable for CSPs. For COPS some optimization techniques have to be incorporated. Commonly used crossover operators are generally not applicable as two feasible parents not necessarily produce feasible offspring and repairing infeasible offspring can be very expensive (Craenen et al., 2003). So the only possible choice remains is to use mutation strategies that look for neighbourhood solutions of feasible solutions in a population to search for the optimum solution.

For dynamic COP same equations Eq. (6.5) – Eq. (6.7) can be used with inclusion of parameter t representing time or increment, consequently changing variable l_p with function $l_p(t)$. Many *big* static COPs can also be solved incrementally by treating them as dynamic COP as discussed in Chapter 5 where new constraints or a subset of constraints can be introduced into the system after every n generations. This *divide and conquer* technique for solving a *big* COPs has shown better results than solving it otherwise in the experiments. A new fitness function given in Eq. (6.9) is derived to show higher fitness for later increments i.e. where more constraints have been satisfied. The higher number of satisfied constraints should have higher fitness value. The proof of the Eq. (6.9) is given in Appendix A.

$$f(l) = (L^2+2)^{D+1}\sum_{p=0}^{D} l_p + \sum_{p=0}^{D}(L^2 - l_p)(L^2+2)^p \tag{6.9}$$

6.3.1. Algorithms for discrete COPs

Section 4.3 describes optimization techniques implemented in ICHEA for continuous COPs. This section explains *intermarriage* crossover together with additional mutation techniques commonly used for discrete COPs. The *intermarriage* crossover for discrete COP is influenced from our work in (Sharma, 2010) using *influence* operator with Particle Swarm Optimization approach (Eberhart and Kennedy, 1995) where all swarm particles tend to move towards better positions nearby the best position that leads to the optimum solution (Eberhart and Kennedy, 1995; Onwubolu and Sharma, 2004). This helps in exploring promising solutions in a nearby region of the current best solution. If the *influence* operator is denoted by $\otimes$ then crossover between feasible solutions P_i and P_j involves the following steps:

1. $P_i = P_i \otimes P_j$

2. $P_j = P_j \otimes P_i$
3. $P_i = P_i \otimes P_{best}$
4. $P_j = P_j \otimes P_{best}$

The *influence* operator simply tries to influence chromosome A with predefined number of allele value(s) of chromosome B (called degree of influence) as shown in Fig. 6.4. Suppose chromosome $A = \{4, 2, 5, 1, 3\}$ is influenced with second allele value of chromosome $B = \{3, 1, 4, 2, 5\}$ that moves the influenced value 1 at second position from current fourth position in chromosome A. If a move is not valid then it can be repaired, rejected or kept separately in the repertoire of infeasible solutions which can be persisted with influence operator until a feasible solution is obtained. Influence operator brings one chromosome closer to another.

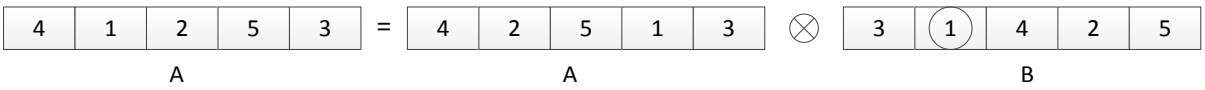


Fig. 6.4. Influence Operator: Chromosome *A* is influenced with the second allele value of Chromosome *B*

In addition to the crossover some mutation strategies can also be applied like swapping two allele values or group of allele values. Remove some allele values and place them in a different position. Many times these operators are problem dependent to define their feasibility after the move. For example in exam timetabling problems experimented in this chapter we defined following operators to search for an optimum solution.

1. *Mutation by traditional Kempe chain*: the Kempe chain has been primarily used in graph coloring problem but it has been proven successful in timetabling problems as well (Tuga et al., 2007; Burke, Eckersley, et al., 2010; Demeester et al., 2012). It starts with moving $1 - 5$ elements randomly picked from timeslot *I* to *J* but this may cause conflicts in timeslot *J* so all conflicting elements from *J* is moved to *I* but again the inclusion of new elements from *J* may cause conflict in *I*. So all conflicting elements are moved back and forth from *I* to *J* until no conflicts remain.
2. *Mutation by boundary Kempe chain*: we also propose a variant of traditional Kempe chain for exam timetabling problem called boundary Kempe chain where selection of timeslot *I* is based on locations of most conflicting elements of the timetable and timeslot *J* is selected from far ends of the timetable as it is likely to have less conflict

on the ends because of less conflicting timeslots at one end. Timeslot I is randomly picked from the locations of top 10% of the most conflicting elements (exams) and timeslot J is selected from the farthest or next to farthest timeslots either from beginning or end of the timetable.

3. *Swap two elements*: randomly swap two elements (exams) which may result in an infeasible solution. These infeasible solutions are kept in a separate set which is later repaired using traditional Kempe chain by removing the infeasible element from timeslot I to J (Burke, Eckersley, et al., 2010).
4. *Swap the whole group (time slot)*: Either swap two timeslots or move a timeslot from one place to another randomly.
5. *Mutation by removal*: remove an element (exam) randomly then try to insert it into a different timeslot. Reject the solution if infeasible.
6. *Mutation cluster*: randomly pick and add an exam from a cluster of possible exams that a timeslot can have for a feasible solution which is not already in that timeslot. Locate this exam that is elsewhere in the timetable and delete it.
7. *Crossover using influence operator* (Sharma, 2010): This can be used between feasible and infeasible chromosomes described earlier. A predefined number of infeasible solutions neighboring a feasible solution called a *community* move closer towards a feasible solution that can transform infeasible solutions to a feasible solution. Community members do not move to other communities while they are infeasible, however, only the best feasible member within a community influences other members. The major drawback here is infeasible solution can become a duplicate of the feasible solution.
8. *Crossover using Kempe chain*: As described earlier the *intermarriage* crossover between two feasible solutions can be performed using influence operator; however, influence operator might produce an infeasible solution because of its *move*. The *move* of Kempe chain can be used that produces feasible solutions, instead of the influence operator's *move* for exam timetabling problem. This is done by first locating the timeslot carrying the picked allele value of the influencer chromosome B to influence the other chromosome A. We record the position of this timeslot in chromosome B. The same position of a timeslot in chromosome A works as timeslot I for chromosome A. Then we locate the timeslot carrying the same allele value in chromosome A which

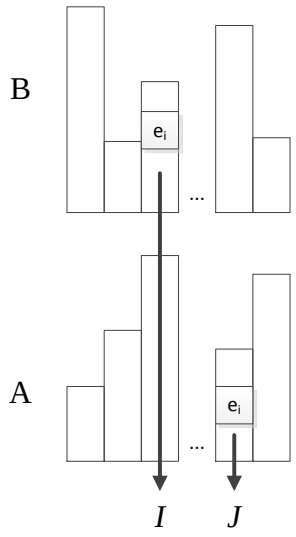


Fig. 6.5. Crossover using kemp chain for influence operator

will be designated as timeslot *J*. Now we apply traditional kemp chain on chromosome *A*. This would influence the chromosome A with chromosome *B* and at the same time keeps the feasibility intact. The process of selecting *I* and *J* for kempe chain *move* has been also shown in Fig. 6 where allele value e_i from chromosome *B* influences chromosome *A*. Timeslot *I* and timeslot *J* is determined by the positions of the column containing e_i in chromosome *B* and chromosome *A* respectively. Once timeslot *I* and timeslot *J* are found kempe chain mutation is applied on chromosome A.

Each of these operators runs in a sequence in ICHEA where next operator is selected when there is no improvement for the best so far solution for *s* generations. Stagnant generations *s* is 5 in our experiment as higher value higher value slows down the process and getting another operator is likely to resolve the stagnant state. In hyper-heuristic domain these operators can be used as lower level heuristics where sequence of operators is chosen heuristically. This is a cyclic process that runs until the termination condition is met.

Local search using the above mentioned operators can be useful to maintain the feasible solutions and search for optimal solution through neighborhood search. Hill-climbing algorithm is canonical form of a local search, however, the basic hill-climbing algorithm generally provides locally optimum values which depends on the selection of the starting point. If lucky then some of these initial locations will have a path that leads to the global optimum solution (Michalewicz and Fogel, 2004b). Hill-climbing is an iterative process where each candidate solution searches in its nearby region for better solutions. This becomes

even more difficult when the search space is discontinuous with constraints. More sophisticated local search techniques are Simulated Annealing (probabilistic hill-climbing) and Tabu Search where they apply some mechanism to escape from local optimal solution (Michalewicz and Fogel, 2004b; Burke, Eckersley, et al., 2010).

```
i ← 0
Each vector V_i of the population of feasible solutions pop_COP
has its corresponding stack H_i to store previous changes to
the solutions and corresponding queue T_i to hold tabu list.
For each vector V_i ∈ pop_COP
  {C} = clone(V_i)
  k ← 0
  For each clone of V_i
    C_k ← mutation(C_k)
    If f(C_k) > f(V_i) Then
      Push the vector V_i in its corresponding fixed sized
      stack H_i ← Push(H_i, V_i)
      V_i ← C_k
    Else
      If V_i is not progressive Then
        Set V_i as tabu by enqueuing it into its corresponding
        fixed sized queue T_i ← Enqueue(T_i, f(V_i))
        Revert V_i to its previous solution
        V_i ← Pop(H_i)
        Terminate For loop
      End If
    End If
    k ← k + 1
  End For loop
  i ← i + 1
End For loop
```

Fig. 6.6. Pseudocode for revertible clonal hill-climbing

We propose a new local search strategy which is called *revertible clonal hill-climbing* (RCHC). The closest resemblance of RCHC is hill-climbing with *tabu* search. However it differs in management of *tabu* list and inclusion of backtracking mechanism. Firstly feasible solutions pop_{COP} are retrieved through intermarriage crossover describe in Section 6.2 where each feasible solution of the population will carry a set of *tabu* list and a set of previous solutions. The procedure begins by creating number of clones for each individuals inspired by CLONAX (de Castro and Von Zuben, 2002; Sharma and Sharma, 2011) algorithm where each clone goes through allocated mutation process. The size of the clones is determined by a formula $N_c = min\left(5, \sum_{i=1}^{|pop_{COP}|} \frac{\omega |pop|}{i}\right)$ similarly used in (de Castro and Von Zuben, 2002) for CLONALG algorithm where N_c represents size of the clones for a partial solution i sorted from best to worst. ω is 1 in our experiments. This formula creates more clones for better solutions. A mutant clone immediately replaces the incumbent solution if it has better fitness value. If an individual has not been improved in t generations then it is reverted to its previous state and the current state is marked *tabu*. The algorithm keeps the history of H changes which is generally in the range of $[0, 5]$. This is similar to the *backtrack* algorithm when a solution does not improve for δ generations then this solution is reverted to its parent branch and the new search is started so that poor solution can be eliminated with the new solutions to promote diversity. However this algorithm has limited number of backtracks for efficiency purpose. The pseudocode of RCHC is given in Fig. 6.6. The size of *tabu* list and previous states are constant. If the lists are full then the new record replaces the oldest record. If all previous solutions are retrieved then *revert* process removes the individual from the

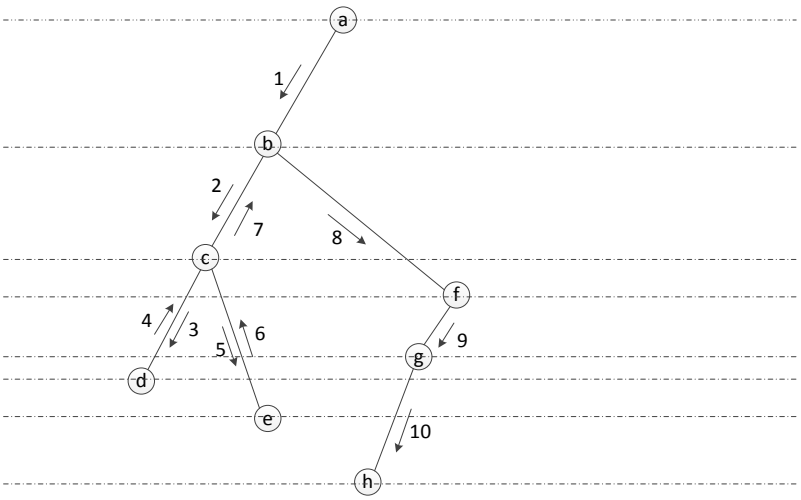


Fig. 6.7. Revertible hill-climbing for a feasible solution

population and new feasible solution is added into the population of feasible solutions pop_{COP}. Fig. 6.7 shows backtracking steps of a feasible solutions starting at position a. When the solution is improved it traverses through positions $a \rightarrow b \rightarrow c \rightarrow d$ iteratively. When position d does not progress it reverts to position c which is then improved through mutation and finds a new position e. After some time position e becomes stagnant and similarly it traverses through $e \rightarrow c \rightarrow b \rightarrow f \rightarrow g \rightarrow h$ where h is its latest position. Iterative steps are numbered in sequence in the figure. RCHC is part of ICHEA which facilitates backtracking for evolutionary search where *mutation* is any one of 1–6 operators described earlier. In case of crossover operators 7 and 8 ICHEA's selection process provides two vectors V_i and V_j.

6.3.2. Search space analysis

Population based meta-heuristic like an EA uses greedy approach by keeping the good solutions and discarding solutions with low fitness values, however, these low fitness individuals can be promising depending on its locality in the search space. We define some basic types of search space for local search and their characteristics with depictions shown in Fig. 6.8. Diagram 1 shows a small and steep hill, diagram 2 shows a steep hill with larger spread, diagram 3 has small elevations (local optimum), diagram 4 has gradual elevation on a bigger space, diagram 5 has little elevation followed by a big plateau which elevates again and diagram 6 is same as diagram 5 but with a hole (constraint) in the plateau. More possibilities of the search space are possible but the objective here is to demonstrate that a point with lower fitness value in a search space can lead to the global optimum solution. If the population of EA is not managed properly then greedy approach tends to discard promising points in early generations. For example if diagram 4 leads to global optimum and EA starts with points that have higher fitness values located near the local optimum regions of diagram 1 or 2 together with some point in the region of diagram 4 towards the lower elevation. As the algorithm progresses the points of diagram 4 may not survive because of their lower fitness value. Once these points are deleted local search through neighborhood would fail to locate the global optimal solution.

ICHEA is a population based EA that uses RCHC for discrete COPs. Given the scenario above ICHEA does not filter out solutions based on fitness value alone. If a particular point has low fitness value but its fitness value gradually increases then it is considered promising without comparing it with other points of higher fitness value. When a solution becomes stagnant then it is reverted or deleted as described in RCHC algorithm.

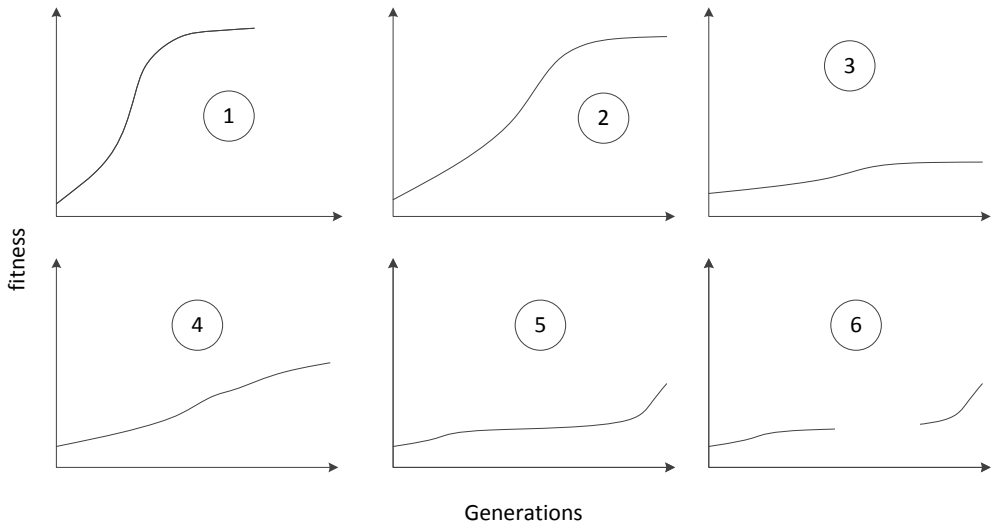


Fig. 6.8. Types of promising regions in a search space

6.3.3. Stalled local optimal solutions management

It has been noted through experiments that ICHEA still gets stalled in local optimal solution. We applied a similar strategy of *tabu* search algorithm based technique described for continuous COPs in Section 5.5.2 which is further enhanced to suit the needs of discrete search space. We keep the history of t previous best solution achieved so far to determine the *tabu* region of the search space using the following formulation:

$$S_{tabu_1} = S_{curBest} \cap S_{prevBest_1}$$

$$S_{tabu_{i+1}} = S_{tabu_i} \cap S_{prevBest_i} \; for \; \forall \; i \; \in \{2, 3, .. t\} \tag{6.10}$$

Eq. (6.10) shows the *tabu* region in the search space by S_{tabu_i} where $i \in \{1, 2, 3, .. t\}$ for a predefined constant value t which is generally 5. For discrete search space S_{tabu_i} can be just a sequence/pattern of allele values. $S_{curBest}$ is the best solution of the current generation and $S_{prevBest_i}$ is the previous i^{th} best solution. The intersection of S_{tabu_i} and $S_{prevBest_i}$ retrieves the common or unchanged allele value from either current best to previous best or current *tabu* region to previous best. This is done to track the traversal of the best solution and try to divert the whole population that is stalled in the local optimal region. If S_{tabu_i} for $i = 1$ does not improve the best solution then i is incremented to get the next *tabu* region. 0 shows the incremental shrinking of the *tabu* region. Part (A) shows the traversal of the best solution, Part (B) and Part (C) show the reduction of the size of *tabu* region for next increments based

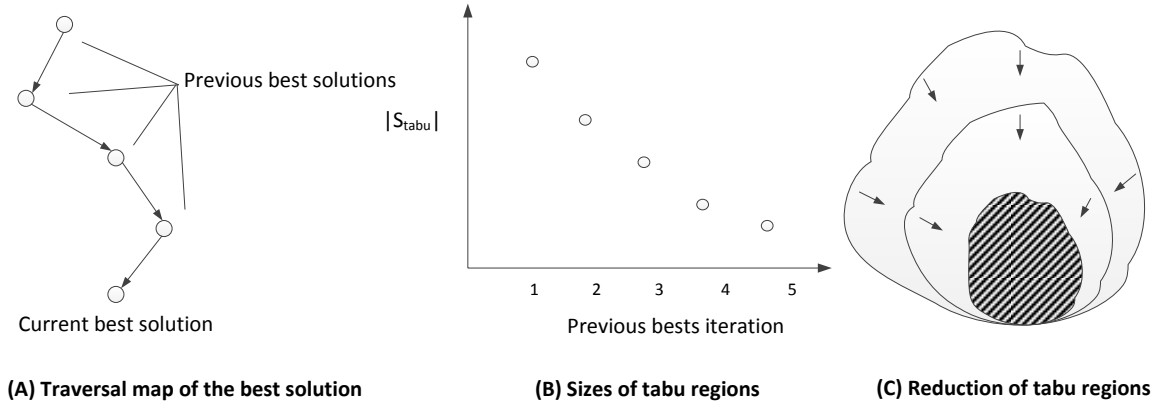


Fig. 6.9. Shrinking of tabu region for stalled ICHEA

on Eq. (6.10). As the *tabu* region decreases in size it is more likely to find chromosomes that have similar pattern to the *tabu* region. In other words when a *tabu* region becomes a subset of a chromosome in terms of pattern matching of allele values then the incumbent chromosome is said to be *tabu* which is shown in Eq. (6.11) where $\omega(S_i)$ is a function denotes if a chromosome S_i is tabu or not. For example a pattern of *tabu* region can be denoted as $S_{tabu} = \langle 2, 5, \#, 1, 3, \# \rangle$ where # indicates a chromosome in question can have any value at that position. A chromosome $S = \langle 2, 5, 4, 1, 3, 6 \rangle$ is a subset of S_{tabu} but $S = \langle 2, 5, 1, 4, 3, 6 \rangle$ is not.

$$\omega(S_i) = \begin{cases} 0, & if\ S_i \subseteq S_{tabu_i} = \emptyset \\ 1, & if\ S_i \subseteq S_{tabu_i} \neq \emptyset \end{cases} \tag{6.11}$$

6.4. ICHEA algorithm on discrete search space

Some adjustments are required in the previously introduced ICHEA for continuous domain. As discussed in Chapter 5 solving CPs incrementally has many advantages. Some *big* CPs like exam timetabling problems can be divided into several components (subsets of constraints) then each component can be solved incrementally. This divide and conquer approach solves a CP by taking a component to get feasible solutions before taking the next component. In Chapter 5 each component consists of one constraint only which were sorted decreasingly based on their constraint strengths ρ. In the literature, exam timetabling problems sort the constraints according to the largest degree (LD), saturation degree (SD),

largest weighted degree (LWD), largest penalty (LP) or random Order (RO) (Caramia et al., 2001; Burke et al., 2005, 2012). LD and SD are commonly used sorting order. In LD exams are ordered decreasingly according to the number of conflicts each exam has with others, and in SD the exams are ordered increasingly according to the number of remaining timeslots available to assign them without causing conflicts. The definition of other sorting orders can be found in (Burke et al., 2012). ICHEA uses LD to sort all the exams based on clashes with other exams. It takes only 5% of the sorted exams in every increment and once a feasible solution is obtained the optimization operators are applied for G generations before taking next increment of exams. The value of G is 500 in our experiments.

Incrementality in solving CPs also comes handy when a new constraint is added or an existing constraint is changed. It is also useful in doing what-if analysis on constraint strengths or inclusion/exclusion of constraints. The by-product of incrementality is a set of partial solutions for each increment that can be stored separately and later reused, where a new constraint can be added or an existing constraint can be changed without making too much distortion to the current solution. Suppose there are n sets of partial solutions $\{P_1, \ldots, P_n\}$ where each set of partial solution P_i satisfy L_i constraints and carries N_i feasible solutions. Fig. 6.11 Part 1 and Part 2 show the algorithm for reusing partial solutions to cater for any changes to the constraints. The algorithm describes the addition of a new set of k constraints C of same strengths that are verified against partial solutions to get feasible solutions. If a current partial solution is unable to accommodate constraints C then these constraints are tested with previous partial solutions iteratively until all the partial solutions are exhausted. If constraints are structured according to their respective strengths then the algorithm is biased towards retaining solutions that solves more constraints of higher strengths rather than

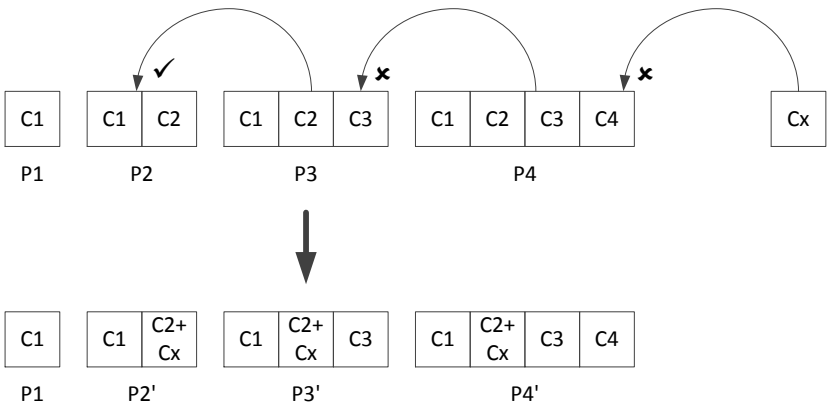


Fig. 6.10. Incremental process shown diagrammatically

keeping the existing partial solutions. We used the following notations for the algorithm:

1. $\boldsymbol{X}_{\boldsymbol{i}}^{\boldsymbol{R_i \times C_i}}$: Matrix $\boldsymbol{X}$ on i^{th} increment has R_i rows and C_i columns.
2. P_i: Set P_i on i^{th} increment.
3. P_i^j: Vector j of set P_i on i^{th} increment.
4. $\otimes$: Intermarriage crossover operator
5. %: Additional comments afterwards

The graphical interpretation of the algorithm can be seen in Fig. 6.10 where a constraint C_x is

$i \leftarrow n$

While $(i \geq 1)$

% the structure of $P_i^{N_i \times L_i}$ can be kept intact if no distortion of the solution is allowed.

Run ICHEA on existing partial solutions $P_i^{N_i \times L_i}$ and add a set of k constraints $C^{k \times 1}$ for the feasibility check for a given number of generations.

% Define parents pool size d which is generally $T/2$

$T = \{P_i^{N_i \times L_i} \cup C^{k \times 1}\}$ % T is total population of size $N_i + k$

For Each Generation

$X \leftarrow$ Selection(T) where $|X| = d$;

$P'_i = \{p \in X \,|X \subset T\}$

% Intermarriage crossover using operator $\otimes$.

$P''^{j}_i \leftarrow p1 \otimes p2$ where $p1, p2 \in P'_i$ and $p1 \neq p2$

$P''_i = \{\forall P''^{j}_i \,|\, j \in \{1, \ldots, N_i\}\}$.

Mutation(P''_i);

$X = \{P''_i \cup \mathrm{T}\}$ % Collect all chromosomes

$T \leftarrow$ SortAndReplace(X);

CheckTerminationCriteria();

End for loop;

...

Fig. 6.11. (Part 1): Reusing partial solutions in ICHEA for new addition of constraints

...

Solution is found when $\left\{\exists T^j \in T \mid \left|T^j\right| = L_i + k,\ j \in \{1, \ldots, |T|\}\right\}$ however, feasible solutions can be stored for optimization purpose in T'.

$T' = \left\{\forall T^j \in T \mid \left|T^j\right| = L_i + k,\ j \in \{1, \ldots, |T|\}\right\}$.

If $(T' \neq \emptyset)$ **Then**

Print("Constraints C have been assimilated into partial solution P_i. Input T' into ICHEA to get full solution");

Terminate While loop

Else

If P_i has same or higher strength ($\succcurlyeq$) than C then we need to keep all constraints of P_i.

If $(P_i \succcurlyeq C)$ **Then**

$F = \left\{\forall c \in \left\{T^j - P_i^j\right\} \mid P_i^j \subset T^j \wedge j \in \{1, \ldots, N_i\}\right\}$

Else

$F = \left\{\forall c \in \left\{T^j - C\right\} \mid C \subset T^j \wedge j \in \{1, \ldots, N_i\}\right\}$

End If

Print("List of combinations of constraints not solved:")

Print(F);

Print("Moving to next partial solution");

% Stop if user decides.

$i \leftarrow i - 1$

End if

End While loop

Fig. 6.11. (part 2): Reusing partial solutions in ICHEA for new addition of constraints

presented to partial solutions. It is assumed that constraint C_x has the lowest strength. Partial solutions P_4 and P_3 are unable to accommodate C_x but P_2 gives feasible solutions with C_x. Now the previous partial constraints are no longer valid which are replaced by P'_2, P'_3 and P'_4 recpectively. All these new partial solutions satisfy the given constraint C_x.

This algorithm is also useful in doing what-if analysis or updating constraints by revisiting previously solved partial solutions. If the current solution is to be kept intact then the allele

values in the chromosomes can be locked to prevent any changes through evolutionary operators. However, the search space becomes more constrained when the problem is required to be optimized. Any changes in an existing constraint can also be backtracked to the increment where it was resolved. For real time DCPs this algorithm is also very helpful as it does not require restarting the whole process unless none of the partial solutions are able to accommodate the changes in constraints.

```
chromosomes = initializeChromosomes();
for each generation
  parents = Selection();
  offspring = interMarriageCrossover(parents);
  Mutation(offspring);
  chromosomes = chromosomes + offspring;
  SortAndReplace();
  CheckTerminationCriteria();
End for loop;
```

Fig. 6.12. Pseudocode for ICHEA

As discussed in Section 4.3.1 ICHEA runs two processes in parallel – one to solve CSP and another to optimize CSP solutions. The parallel process starts by dividing the whole population pop into 2 parts. First part pop_{COP} keeps the feasible solutions that are required for optimization and the second part pop_{CSP} keeps the *good* infeasible solutions that are processed to get CSP solutions. The ratio of $pop_{COP}: pop_{CSP}$ is 1:1 for our experiments. The pseudocode of ICHEA is given in Fig. 6.12. followed by the description. Initialization of chromosomes and operators for reproduction has been modified but the structure of ICHEA is still same as any other EA.

InitializeChromosomes: ICHEA promotes incrementality and it can solve a given CP in an incremental manner. To solve a discrete CP incrementally, only a set of constraint can be considered at a time. For example if there are total of m constraints for a static CP then only r proportion can be considered at a time for certain number of generations. A population of chromosomes for the first increment can be randomly generated using sequence of integer values $\langle 1, 2, \ldots rm \rangle$ where each number represents a constraint. For next $(i + 1)$ increment the new chromosomes have values in the range $\langle irm + 1, irm + 2, \ldots (i + 1)rm \rangle$. If $(i + 1)rm > m$ then it is turned into m.

InterMarriageCrossover: The crossover techniques have been described in Section 6.2 works well for discrete CSPs but it is not applicable for COPs. COPs depend on mutation strategies only as general crossover techniques mostly produce infeasible solutions which would require a separate repair technique to transform all the infeasible offspring to feasible offspring which is normally computationally expensive (Coello Coello, 2002).

Mutation: Many mutation strategies can also produce infeasible solutions but it can be less expensive to repair compared to offspring produced through common crossover operators. Additionally it suits well for local search techniques. As mentioned in Section 6.3.1 ICHEA uses RCHC to optimize a CP. Once a chromosome solves all irm constraint for i^{th} increment, it is considered a partial solution for COP which then qualifies to apply mutation strategies to optimize the exiting partial/full solutions. We have also incorporated aforementioned *influence* operator. This is particular useful for CP as feasible solutions *influence* infeasible solutions. This makes infeasible solutions feasible by *moving* towards feasible solutions. Same *influence* operator is also applied to the *tabu* regions. Initially a *tabu* region goes through other mutation operations to change the course of the evolutionary search which later influences infeasible solutions towards it in an attempt to make new feasible solutions outside the *tabu* regions.

SortAndReplace: sorting is not required for feasible solutions (pop_{COP}) as RCHC only keeps the good solutions and poor solution are either discarded or stored as *tabu* solutions temporarily. Hence the population of feasible solutions remains intact. Population of infeasible solutions (pop_{CSP}) are increased by k additional solutions so an intelligent sorting is required that keeps the good solutions as well as maintains the diversity; because as the generation progresses EAs tend to preserve better chromosomes and poor chromosomes die away. If only best ones are kept then diversity is lost (Goldberg, 1989). We first sort the whole pop_{CSP} then pick the solutions using the following exponential function in Eq. (6.12) that keeps the good solutions as well as maintains the diversity. The size of the whole population remains intact. The details of the Eq. (6.12) is given in Appendix A.3.

$$f''(i) = \begin{cases} f'(i-1) + 1, \ if\ f'(i) \leq f'(i-1) \\ f'(i), \ otherwise \end{cases} \tag{6.12}$$

where $f'(i) = \left\lfloor (|POP_{CSP}| + k) \frac{f(i) - f(1)}{1 - f(1)} \right\rfloor$ and $f(i) = e^{-\rho\left(\frac{|POP_{CSP}| - i}{|POP_{CSP}|}\right)}$ with value of $\rho = 5$.

Other functionalities of ICHEA are same as defined in Section 3.3 for continuous search space.

6.5. Experiments for discrete CSPs

We used a toy problem for the experiment namely N-Queen problem described in Section 6.2 that serves as a classic discrete CSP. To demonstrate the importance of information from constraints the experiments are conducted in two phases. The first part of the experiment tries to solve N-Queen problem without using any sort of information from the problem. The second part of the experiment does the preprocessing of the chromosomes to work on unique allele values only. The idea is to provide as much information as possible to the evolutionary search.

6.5.1. No exploitation of information from the problem

For this test case we compared ICHEA with Differential Evolution (DE), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), standard Genetic Algorithms (GAs) and Non-dominated Sorting GA-II (NSGA-II) (Deb et al., 2002) that do not use any sort of information from the problem domain. GA is taken from Genetic Algorithms toolbox Revision: 1.1.4.2, 2004 available in Matlab 7.0.1 and NSGA II written in C language is taken from (Deb, 2011). ICHEA has been developed in Java language. The test results for DE and CMA-ES have been taken from (Rahnamayan and Dieras, 2008) where 20 trials for each problem have been taken into account. The test results are based on number of function calls (NFC) and success rate (SR). If the NFC $\geq 2x10^6$ then it is considered that the solution is not found. The experimental set up is discussed below.

- **Fitness Function:** the fitness function is the total violation count and the chromosomes are ranked based on this fitness function. The solution for CSP is to find at least one chromosome with no violation.
- **Allele Values:** DE, CMA-ES and GA generate real numbers for allele values but in case of N-queen problem the real numbers are converted into integer values by taking the round off value to calculate the fitness. NSGA II uses binary string representation and ICHEA uses integer values. Candidate solutions can have duplicate allele values.
- **Efficiency Measures:** NFC and SR are used to compare the performance of different algorithms. NFC is simply the total count of objective function invoked by the

algorithm. SR is the rate of successful trials for each problem i.e. $SR = successful\ trials/total\ trials$.

- **Parameters:** for all the algorithms population size of 100 is used. Scattered crossover is used for GAs. Mutation rate of 0.1 is used for ICHEA and GAs. All default parametric values are used for NSGA-II.

Table 6.1 shows the comparative results based on NFC and SR to solve N-queen problem. N denotes the size of the chessboard. It can be observed as the problem size increases the solution quality decreases for all the algorithms except ICHEA. The outcome of the test

Table 6.1. Comparative test results on no problem specific information extraction

N	CMA-ES (Rahnamayan and Dieras, 2008)	DE (Rahnamayan and Dieras, 2008)	GA	NSGA II	ICHEA
4	456 NFC (SR = 1.00)	134 NFC (SR = 1.00)	367 NFC (SR = 1.00)	93 NFC (SR= 1.00)	39 NFC (SR = 1.00)
5	656 NFC (SR = 1.00)	254 NFC (SR = 1.00)	750 NFC (SR = 1.00)	217 NFC (SR = 1.00)	37 NFC (SR = 1.00)
6	22,013 NFC (SR = 1.00)	1,11,136 NFC (SR = 0.65)	30,086 NFC (SR = 0.75)	694 NFC (SR = 1.00)	51 NFC (SR = 1.00)
7	9,964 NFC (SR = 1.00)	24,338 NFC (SR = 0.95)	1,400 NFC (SR = 1.00)	2631 NFC (SR = 1.00)	34 NFC (SR = 1.00)
8	84,962 NFC (SR = 1.00)	7,576 NFC (SR = 0.75)	3,786 NFC (SR = 0.80)	1273 NFC (SR = 1.00)	41 NFC (SR = 1.00)
9	133,628 NFC (SR = 1.00)	19,296 NFC (SR = 0.50)	18,333 NFC (SR = 0.80)	27,852 NFC (SR = 1.00)	72 NFC (SR = 1.00)
10	263,572 NFC (SR = 0.95)	286,208 NFC (SR = 0.30)	3,300 NFC (SR = 0.30)	1,737 NFC (SR = 1.00)	83 NFC (SR = 1.00)
11	284,382 NFC (SR = 0.95)	68,255 NFC (SR = 0.10)	15,550 NFC (SR = 0.40)	SR = 0.00	132 NFC (SR = 1.00)
12	295,740 NFC (SR = 0.75)	99,120 NFC (SR = 0.25)	23,000 NFC (SR = 0.70)	SR = 0.00	122 NFC (SR = 1.00)
13	376,631 NFC (SR = 0.85)	95,485 NFC (SR = 0.15)	3,400 NFC (SR = 0.10)	SR = 0.00	293 NFC (SR = 1.00)
14	450,654 NFC (SR = 0.85)	160,475 NFC (SR = 0.10)	47,350 NFC (SR = 0.40)	SR = 0.00	308 NFC (SR = 1.00)
15	627,391 NFC (SR = 0.50)	223,425 NFC (SR = 0.10)	95,625 NFC (SR = 0.40)	SR = 0.00	381 NFC (SR = 1.00)

results clearly shows that ICHEA produces consistent results and dominates other EAs. ICHEA is the most efficient algorithm by getting the lowest NFC and highest success rate (SR = 1.00) for all the problems. GA shows unpredictable results where it usually finds the solution in very few evaluations but if it is stuck in local minima then it is generally not able to find the optimum solution.

6.5.2. Information extraction and exploitation

The second test case involves utilization of information extraction and exploitation from the N-Queen problem in evolutionary search. Here problem specific chromosomes have been used where only unique integer values are taken into account for chromosomes' allele values. Unique integers ensure that queens are at least in different rows which satisfy a part of constraint for this problem. All the parameter remains same as of the previous experiment. We used Simulated Annealing (SA), Tabu Search (TS), Particle Swarm Optimization (PSO) and GA along with ICHEA for the experiment. The test results of SA, TS and GA is taken from (Martinjak and Golub, 2007) and test results for PSO is taken from (Hu et al., 2003). ICHEA does not need to be modified as it has problem independent formulation for its *intermarriage* crossover. Appended allele values are not necessarily unique.

Table 6.2 shows the comparative test results based on NFC only when some problem specific information has been extracted from the problem. The best, median and mean results for ICHEA have also been shown. No changes are made to ICHEA but it still performs best for most of the problems (as shown in bold). The results are also represented graphically in Fig.

Table 6.2. Comparative test results on after information extraction from the problem

N	SA (Martinjak and Golub, 2007)	TS (Martinjak and Golub, 2007)	GA (Martinjak and Golub, 2007)	GA (Eastridge and Schmidt, 2008)	PSO (Hu et al., 2003)	ICHEA (best)	ICHEA (median)	ICHEA (mean)
8	493	182	400	**100**	-	84	119	115
10	948	472	4910	266	-	97	162	**176**
20	-	-	-	2000	5669.7	279	698	**898**
30	2160	4655	91790	2300	-	301	538	**970**
50	2849	22663	1759230	5660	14991.4	729	1190	**2257**
75	6091	81030	571170	6300	-	380	2568	**3393**
100	7873	206910	887770	15600	36199.4	1977	3702	**7595**
200	21708	2399940	2287960	460475	934399	7360	14533	**15489**
300	**24636**	9382620	2774820	-	-	6767	34043	37730

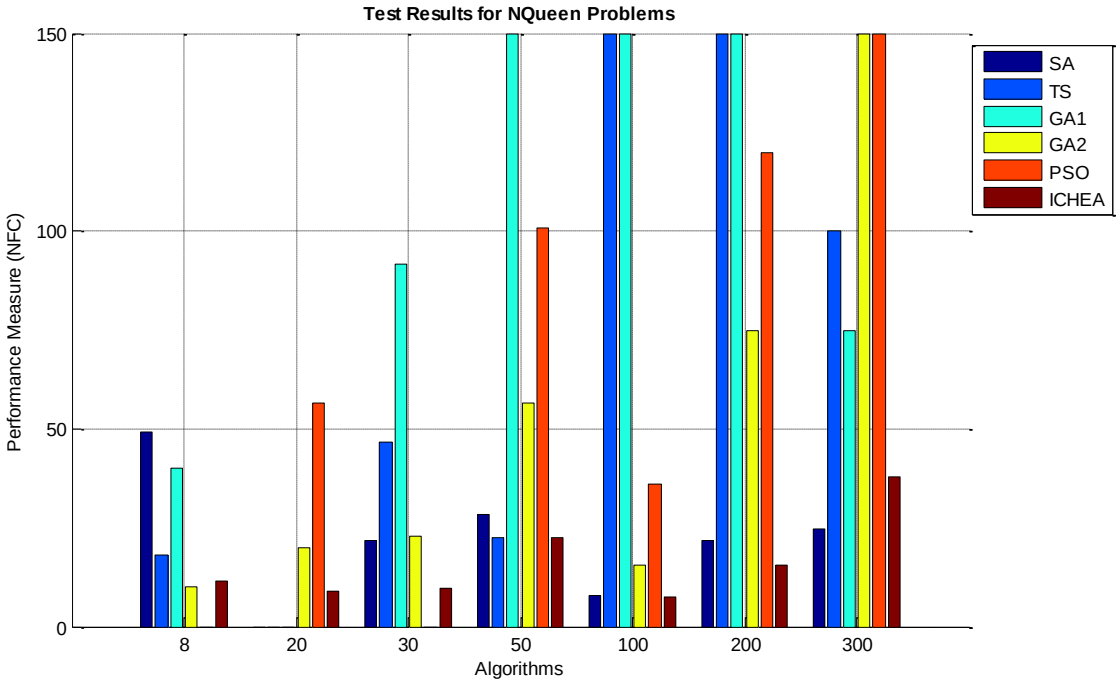


Fig. 6.13. Performance comparison of tested algorithms on NQueen problems

6.13 to compare the performances of each algorithm. The problem solutions for some problems are given in Appendix B. The test results obtained by (Eastridge and Schmidt, 2008) is also impressive where the authors use partially matched crossover (PMX) and an unusual selection process where only top two candidates are selected for mating in each generation and rest of the population is replaced by making duplicates of this pair.

6.6. Experiments for discrete COPs

A good example of a discrete COP is a university exam time tabling problem that has been attracting the attention of the scientific research community across Artificial Intelligence and Operational Research for more than 40 years (Burke et al., 2005; Qu et al., 2008). A timetable in general is a placement of a set of meetings in time. A meeting is a combination of resources (like rooms, people and items of equipment). An instance of a timetabling problem is university exam timetabling where exams are required to be spread out sufficiently for all the students to give them break so they are better prepared for next exams. Exams must be scheduled so that no student has more than one exam at a time (Burke et al., 1997). Exam timetabling problem can be expressed as allocating a set of exams $E = \{e_1, e_2, \dots, e_e\}$ into a limited number of ordered timeslots (time periods) $T = \{t_1, t_2, \dots, t_t\}$ and rooms of certain capacity in each timeslot $R = \{r_1, r_2, \dots, r_t\}$ subject to a set of constraints. Benchmark exam timetabling problems from ("Benchmark Exam Timetabling Datasets," 2013) used in our

experiments do not take into account room capacities. Exam timetabling problems have two dimensional constraints $c_i = \{x_1, x_2\}$ for $\forall i \in \{1, \dots, e\}$ where x_1 and x_2 represent an exam e_{j} allocated to timeslot t_{k} for $\forall j \in \{1, \dots, e\}$ and $\forall k \in \{1, \dots, t\}$. There are total of e constraints and one exam cannot be allocated to more than one timeslots (Qu et al., 2008). Suppose two parents P_1 and P_2 have variable length partial solutions $\langle\{3,1\},\{6,1\}\rangle$ and $\langle\{6,2\},\{2,3\},\{5,4\}\rangle$ respectively. The *intermarriage* crossover of ICHEA only tries to add the exams of one parent into another. Exams that violate the *hard* constraints are dropped so the offspring are also feasible chromosomes. The generated offspring from these parents either satisfy equal or more constraints. One possible offspring can be $\langle\{3,1\},\{6,1\},\{2,9\},\{5,1\}\rangle$ that has been produced by adding exams from P_2 to P_1 depends on the availability of timeslots. Exams 2 and 5 are allocated to the available timeslots 9 and 1 respectively. Each offspring has traits from both parents. An advantage of using variable length chromosome in this manner is generality of the crossover operator that only tries to fuse components from one parent into another. It does not need to define any problem specific algorithm to get feasible solutions as many other approaches like (Burke et al., 1996; Abdullah et al., 2006; Demeester et al., 2012) use bespoke algorithms or graph-coloring heuristics to get the feasible solutions.

A common focus in the literature has been mainly to produce optimum solution with lowest cost function indicating high spread of exams for each student (Burke et al., 1997; Burke, Eckersley, et al., 2010; McCollum et al., 2010; Demeester et al., 2012). Efficiency is an important aspect of optimization problems, however conforming to a time limit is not an important constraint in real world timetabling (Abdullah et al., 2006; Demeester et al., 2012). An important but generally over-sighted issue with timetabling problem is regeneration of a timetable using previously prepared timetables. The traditional techniques in any university require lots of user input where a lot of work from the technicians is required every time a new time table is generated. The drawback of EAs of not imitating real human behavior of learning from past can be labeled as an “unintelligent artificial intelligence” solution. A simple solution to this problem is to provide a set of partial solutions that are solved incrementally according to their preferences and which can later be combined to get the final solution. Some partial solutions can also be reused, updated or removed and then merged again to get the final solution more efficiently without regenerating the whole solution.

As discussed in Section 6.3 there are two ways to provide fitness function for a CP: a problem dependent weight based penalty function and a generic penalty function. We used University

of Toronto benchmark exam timetabling problems (version I) given in (Qu et al., 2008; "Benchmark Exam Timetabling Datasets," 2013) where the given weights based on the spread of exams for each student is:

$$W_d = S_d 2^{4-d} \tag{6.13}$$

where d is the distance between two timeslots in the range [0 4], S_d is total corresponding students and W_d is the total corresponding weight. The cost function is the average weight corresponds to each student given as:

$$f = \frac{1}{S} \sum_{d=0}^{4} S_p 2^{4-d} \tag{6.14}$$

However, Eq. (6.9) can be used for generic penalty function where l_p indicates total exams violating constraints of p^{th} preference. We mostly focused on standard weights based fitness function for benchmark exam timetabling problem as there are lots of published results to compare with. At the end we only had limited experimental results for generic fitness function since there are no known results to be compared from our knowledge. Currently, its major drawback is that it is not able to handle overlapping constraints like maximally spread the exams for each student as well as minimize the students affected for constraints of low preferences as computed through weights in Eq. (6.13).

6.6.1. Problem dependent weights based fitness function

Hyper-heuristics have been frequently used to solve benchmark exam timetabling problems which show promising results. The same hyper-heuristics can be applied to same class of problems like graph coloring problem and exam timetabling problems (Burke, Eckersley, et al., 2010; Demeester et al., 2012). Hyper-heuristics heuristically selects these mutation strategies that best suits a given problem. ICHEA is a meta-heuristic algorithm that uses multiple mutation strategies to optimize a CP as described in Section 6.3.1. All the benchmark problems have been experimented on a Windows 7 machine with Pentium (R) i5 CPU 2.52 GHz and 3.24 GB RAM except the problem Pur93 which was run on a server machine (Intel Xeon CPU 2.90GHz and 128 GB RAM) because of its size and memory requirements. No parallel processing or distributed environment has been used for the experiments. We ran all the problems over-night because of their size and complexity. Additionally, real world timetabling problem does not required to be solved within minutes or hours (Abdullah et al.,

2006; Demeester et al., 2012). Even though smaller sized problems like Hec92 and Sta83 can be solved within an hour or two; however problem Pur93 had to be run for almost 24 hours because of its huge size. All the experimental results have been verified through the standard evaluator program available in the dedicated website for research on benchmark exam timetabling problems ("Benchmark Exam Timetabling Datasets," 2013). We executed each problem for 10 trials to get SRs and establish statistical evaluations.

As discussed in Chapter 5 ICHEA is able to incrementally solve a dynamic CP. An incremental approach to solve a complex static CP gives better results than solving entire constraints altogether. We used both approaches in the experiments to demonstrate supremacy of one approach over another. We observed that this incremental approach also helps in quickly providing feasible partial solutions and eventually feasible solutions at the SR of 100% for all the benchmark problems; whereas SRs of non-incremental ICHEA are very low for bigger problems like Car91 and Uta92 have only 0%-10% of SR, and 30%-70% for other problems of medium size. Non-incremental ICHEA also takes much longer duration to get the first feasible solution. The unpromising outcome from non-incremental ICHEA has led us to do the experiments with incremental ICHEA only. We first sort the constraints (exams clashes) according to LD then remove first 5% of the total exams as input for each increment in ICHEA. *Intermarriage* crossover constructs the new partial feasible solutions which are then optimized using mutation strategies for feasible partial solutions. We used two instances of ICHEA for the experiments to demonstrate the credibility of incrementality. The first and second instances of ICHEA optimize the partial solutions for 0 and 500 generations respectively. The only difference between these two instances is the first one does not apply

Table 6.3. Parameter settings for ICHEA and IICHEA for benchmark exam timetabling problems

Parameter	Value
Population size	50
Optimizing partial solution for G generations in each increment	500
Degree of influence	3
Community size	4
Total communities	10
Stagnant generations	5
A constant β to generate clones	1
Set of H changes for a solution in RCHC	3
Stagnant for δ generations to start backtrack	5
History of t previous best solution for a tabu set	5

Table 6.4. Statistical summary of results from IICHEA and ICHEA

Instance	Best	Median	Worst	SD
Car91	4.91 (5.1)	5.04 (5.3)	5.16 (5.46)	0.01 (0.15)
Car92	4.08 (4.3)	4.1 (4.45)	4.2 (4.54)	0.05 (0.10)
Ear83	33.24 (33.6)	34.02 (34.69)	34.7 (37.39)	0.57 (1.24)
Hec92	10.13 (10.17)	10.33 (10.45)	10.61 (11.15)	0.15 (0.37)
Kfu93	13.58 (13.8)	13.8 (14.1)	14.21 (15.09)	0.20 (0.37)
Lse91	10.37 (10.95)	10.51 (11.34)	10.67 (11.8)	0.11 (0.27)
Pur93	4.67 (5.2)	4.78 (5.43)	4.99 (5.81)	0.12 (0.21)
Rye92	8.63 (9.07)	8.76 (9.4)	8.85 (9.7)	0.08 (0.19)
Sta83	157.03 (157.03)	157.03 (157.03)	157.03 (157.03)	0.0 (0.0)
Tre92	8.33 (8.8)	8.5 (9.3)	8.8 (9.6)	0.16 (0.28)
Uta92	3.28 (3.48)	3.41 (3.60)	3.57 (3.64)	0.07 (0.07)
Ute92	24.85(24.9)	24.9 (25.7)	25.1 (27.0)	0.10 (0.87)
Yor83	36.24 (36.45)	36.6 (37.04)	38.8 (39.69)	0.71 (1.15)

optimization strategies to partial solutions while the other optimizes the partial solution for 500 generations. However, both instances get the feasible solutions incrementally. To distinguish the two instances the first one is called ICHEA and second one is called incremental ICHEA (IICHEA) as it fully exploits the notion of incrementality. These partial solutions consist of exams from current and all previous increments being allocated in the timetable. The total available timeslots of the exam timetables are always fixed to the given value. Constraint optimization happens in a parallel process of optimizing feasible partial solutions and finding feasible partial solutions from infeasible partial solutions as discussed in Section 6.4. This has been realized previously with experimental results on continuous domain in Chapter 5. More importantly ICHEA does not have to define any problem specific algorithm to get feasible solutions as many other approaches like (Burke et al., 1996; Abdullah et al., 2006; Demeester et al., 2012) use bespoke algorithms or SD graph-coloring heuristics to get the feasible solutions.

The parameter settings for IICHEA and ICHEA to solve the benchmark exam timetabling problems are given in Table 6.3. The statistical results of IICHEA and ICHEA on all the problems from University of Toronto benchmark exam timetabling problems (version I) from (Qu et al., 2008; "Benchmark Exam Timetabling Datasets," 2013) are shown in Table 6.4. We only used version I because it has been mostly reported in the literature. ICHEA results are in the brackets. The best solutions from IICHEA for all the tested problems are given in Appendix B. We also compared our best solutions with other published results cited

frequently in the literature in Table 6.5 which is also represented graphically in Fig. 6.14. The functional values have been linearly scaled to [0 20] to show error values relative to each other algorithm in one figure. The lower the error value the better the algorithm.

Table 6.5. Best results from the literature compared with IICHEA

Algorithms	Car91	Car92	Ear83	Hec92	Kfu93	Lse91	Pu93	Rye92	Sta83	Tre92	Uta92	Ute92	Yor83
IICHEA	4.9	4.1	33.2	10.1	13.6	10.4	4.7	8.6	157.0	8.3	3.3	24.8	36.2
(Carter et al., 1996)	7.1	6.2	36.4	10.8	14.0	10.5	3.9	7.3	161.5	9.6	3.5	25.8	41.7
(Merlot et al., 2003)	5.1	4.3	35.1	10.6	13.5	10.5	-	8.4	157.3	8.4	3.5	25.1	37.4
(Casey and Thompson, 2003)	5.4	4.4	34.8	10.8	14.1	14.7	-	-	134.9 ?	8.7	-	25.4	37.5
(Yang and Petrovic, 2005)	4.5	3.9	33.7	10.8	13.8	10.4	-	8.5	158.4	7.9	3.1	25.4	36.4
(Abdullah et al., 2006)	5.2	4.4	34.9	10.3	13.5	10.2	-	8.7	159.2	8.4	3.6	26.0	36.2
(Eley, 2007)	5.2	4.3	36.8	11.1	14.5	11.3	-	9.8	157.3	8.6	3.5	26.4	39.4
(Burke and Bykov, 2008)	4.6	3.8	32.7	10.1	12.8	9.9	4.3	7.9	157.0	7.7	3.2	27.8	34.8
(Burke, Eckersley, et al., 2010)	4.9	4.1	33.2	10.3	13.2	10.4	-	-	156.9	8.3	3.3	24.9	36.3
(Demeester et al., 2012)	4.5	3.8	32.5	10.0	12.9	10.0	5.7	8.1	157.0	7.7	3.1	24.8	34.6

Table 6.6. Best results of some benchmark exam timetabling problems from IICHEA

Instance	l_0	l_1	l_2	l_3	l_4
Car91	1522	1604	1610	1641	882
Ear83	316	372	352	363	200
Hec92	124	133	140	140	90
Kfu93	534	511	515	505	232
Lse91	409	422	434	464	180
Sta83	123	147	123	137	116
Tre92	418	453	507	470	272
Ute92	184	192	184	250	76
Yor83	320	438	352	464	212

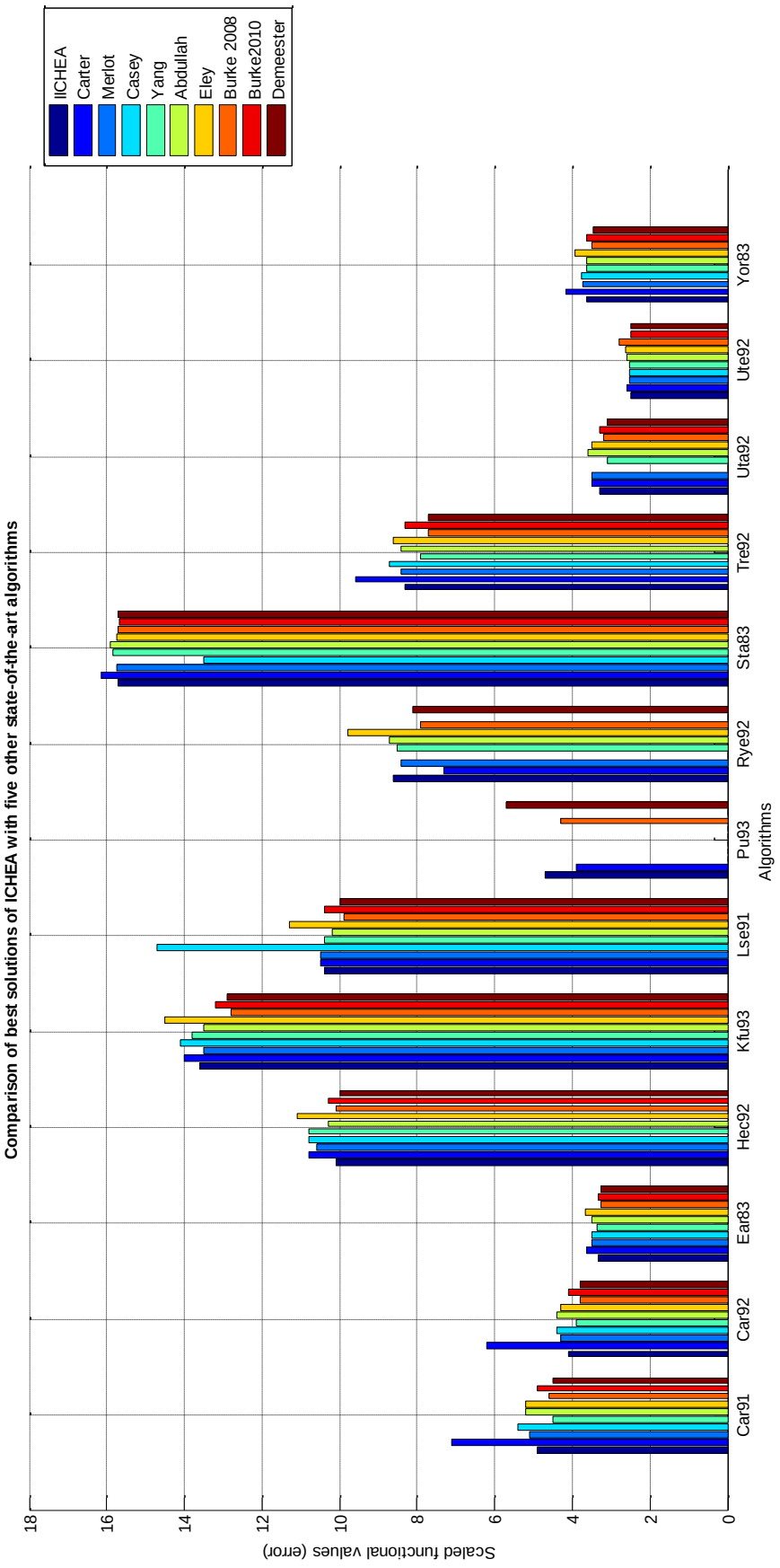


Fig. 6.14. Comparison of error values of tested algorithms on timetabling problems

6.6.2. Generic fitness function

Sometimes constraints have hierarchical structure where a constraint is preferred over others. We used all same benchmark timetabling problems with same parameter settings for IICHEA to be solved with a fitness function from Eq. (6.8). This fitness function is generic and does not have problem dependent weights. The purpose is not to replace weight based penalty function but to show how a generic fitness function can be applied to any constraint problem. The currently form of the equation is for mutually exclusive constraints. Hence we have considered the constraints as only the distance between exams and not the combination of distance and students involved in the exams. We do not have any published results to compare with our results as only problem dependent weights have been used for benchmark exam timetabling problem. We only show the best results for some of the benchmark problems in 0.

6.7. Discussion

The test results for discrete CSPs show that EAs can be significantly improved if the chromosomes are designed to be a problem specific. The experiment in Section 6.5.2 shows if only the unique integer value is taken into account for allele values then the solution is converged much earlier for N-Queen problem. This kind of chromosomes shows considerable improvement in GA. The objective here is not to get the best results for N-queen problem but to show how intelligent an algorithm is. The results in Section 6.5.1 shows that tested optimization algorithm (DE, CMA-ES, GA and NSGA II) blindly searches for the optimum solution through greedy heuristic search manner without extracting the information from constraints while ICHEA utilizes the information from constraints through its *intermarriage* crossover operator and gets the best results. Test results in Section 6.5.2 again favor ICHEA. The advantage of ICHEA is that its formulation is problem independent which still extracts enough information from the constraint to solve the problems efficiently. It can be argued that ICHEA also uses problem dependent integer values for N-Queen problem. In principle the novel formulation of ICHEA does not require the generated integer values to be unique. ICHEA works with allele coupling only. So only the definition of constraints and the rules for coupling of two allele values to partially satisfy the constraints need to be provided. It only maintains the population of feasible solutions that drastically reduces the size of the search space

Experimental results for benchmark exam timetabling problems for COPs are very promising. Results for problems Ear83, Hec92, Sta83, Tre92, Ute92 are in top three and other results are also in the upper half of the best results. It is noted that IICHEA has been giving consistent results for all the problems. It is noted that exam timetabling problems show good results with hyper-heuristics. Using the incrementality technique of IICHEA on these hyper-heuristics can produce even better results as shown in the comparative results between ICHEA and IICHEA. Incrementality in ICHEA produces better results than without incrementality. Consequently, IICHEA can also be used for real time COPs. IICHEA also does not require having a separate problem specific algorithm to get feasible solutions as a preprocessor for constraint optimization. It has found feasible solutions for all the problems at the SR of 100%. The current version of IICHEA seems to have many parameters, however most of the parameters are logical whose values are selected intuitively that should give similar results to other class of problems as well. We also took the initiative to use generic fitness function for evolutionary search for single objective COPs which can become very handy when confronting a new problem to get initial results without much effort. The current generic fitness functions are quite naïve that caters for mutually exclusive constraints only. The future work is to enhance the functions for other types of constraints as well.

6.8. Summary

This chapter focuses on incorporating ICHEA for solving discrete CPs. ICHEA has been designed as a generic framework for evolutionary search that extracts and exploits information from constraints. ICHEA has shown promising results experimented on CSPs and COPs. We proposed another version of *intermarriage* crossover operator for discrete CSPs to get the feasible solutions. The experimental results on a CSP – N-Queen shows exploitation of information from constraints leads to better results. Constraint optimization requires additional optimization techniques that are not all generic in its current form. ICHEA uses many problem specific mutation strategies to optimize exam timetabling problems. Future work is to provide some generic reproduction operators. We also proposed a generic fitness function for single objective COPs that is inspired from developing a *pareto-front* using multi objective COPs. However the status of this fitness function is still in its infancy as it only solves mutually exclusive constraints. A major experimental observation was realizing the efficacy of incrementality in evolutionary search. Incrementality helps in getting feasible solutions with SR of 100% that also produces solutions of better quality. Incremental ICHEA

can also be used for real time dynamic COPs in discrete domain. The competitive results from ICHEA shows its potential in making a generic evolutionary computational model that discovers information from constraints.

Chapter 7

Conclusion and Future Work

The original contribution to knowledge of this thesis is realizing the importance of information from constraints to get efficient and quality results for CPs using EAs. Traditional EAs are 'blind' to constraints as they do not extract and exploit information from the constraints. The search operators of EAs like crossover and mutation not necessarily produce feasible *offspring* from feasible *parents*. This causes the search engine to spent extra computational effort in searching for the solution into the wider search space without only concentrating in the restricted smaller feasible search space. Constraints reduce the search space and it can make the heuristic search more efficient by extracting information from constraints to guide the search engine, search in promising search space only where degree of feasibility is high. We equipped EA with novel operators like *intermarriage* and *arrange-marriage* crossover that extract and exploit information from constraints to better 'inform' search for the feasible solutions. These operators are generic that try to produces more promising progenies from parents without using *repair* functions which slows down the process. Most importantly our evolutionary operators avoid using problem dependent *penalty* functions. We also proposed a strategy to locate feasible regions which are hard to found due to the complexity of constraints and their small size relative to the size of the search space. This strategy involves expansion of constraints regions initially that iterative shrinks to easily locate the feasible regions. The experimental results on benchmark CSPs show ICHEA is able to outperform other meta-heuristic algorithms using the proposed strategies to locate feasible regions efficiently.

We have defined all forms of CPs reported in the literature where each type of the CP has been analyzed and experimented with the benchmark datasets. We also identified lack of research for CSPs compared to COPs where only benchmark problems for COPs are popular and frequently reported. The experimented results on benchmark CSPs demonstrate that many good algorithms for COPs have not been able to perform equally well for CSPs while ICHEA has been consistent with both types of CPs. ICHEA has also performed competitively with

benchmark COPs. ICHEA runs two parallel processes one each for constraint optimization and constraint satisfaction. Once a feasible solution is obtained through one process then it is optimized in another process. Our optimization technique has been inspired from traditional techniques of optimizing a problem. Like movement of swarm particles towards better positions, cloning and mutation of chromosomes and influencing an inferior solution towards more promising solution.

ICHEA has proved to be a versatile EA as it also performs very well for dynamic CPs based on our experimental results for benchmark dynamic CSPs and dynamic COPs. We employed the notion of incrementality for evolutionary search. The basic idea is to use the concept of divide and conquer. It has been established through experimental results that solution quality and SR of ICHEA through incrementality are much better than solving the CPs without incrementality. For incremental solutions we only take a subset of constraints in each increment to find the partial feasible solutions. These partial feasible solutions go through optimization process for certain generations before next increment of the constraints are added into the search space. We also applied incrementality to dynamic CPs where a constraint or subset of constraint can be added and updated at a given time. Particularly the experimental results on benchmark numerical DCOPs clearly show the efficacy of ICHEA using incrementality.

Evolutionary search works well when the population is diverse. We have also designed some strategies to maintain the diversity of the population by not only keeping the best solutions in the population (Goldberg, 1989). The survival of the solutions is based on an exponential function that ensures solutions with very high fitness are always picked together with the solutions in different regions of the search space. Novelty selection from (Lehman and Stanley, 2008) also adds diversity to the population. Our proposed local search algorithm RHCH has been designed on the basis of keeping only the promising solutions in the population no matter if the fitness values are low at a given point in time. However, solutions with high fitness values are always encouraged. A major problem of any EAs is the population being trapped into local optimal solution. We proposed some guiding principles for EAs to move out of the local optimal solution like using *forced constraint violations* that makes hyper-sphere around stalled local optimal solution whose inner region is defined as a new infeasible region (constraint). This new forced constraint is treated as *tabu* for some generations to move the population away from the stalled local optimal solutions. Our

algorithm RCHC also addresses the issue of the population getting stagnant by focusing on promising solutions from all over the search space irrespective of their fitness value. Once the population has become stagnant then it is backtracked to previous good solutions to divert the search into other regions of the search space by making the path *tabu* that leads to stagnant solutions.

We have also defined the generic fitness functions for single objective discrete COPs that work for well-structured constraints with different levels of strengths. Currently these fitness functions only work for mutually exclusive constraints which will be enhanced in future to model complex structured constraints. The main objective here is avoid using commonly used problem dependent penalty factors which is sometimes hard to determined that also requires a careful fine-tuning of parameter to obtain competitive results (Mezura-montes and Coello, 2006; Liu et al., 2010). This can also become very useful when confronting a new COP to get initial results without much effort of using penalty functions.

In summary, most of the research questions and objectives established for the research (as identified in Chapter 1) were addressed. We developed a variation of EA called ICHEA that successfully extracts the information from constraints to guide the evolutionary search which is evident from the experimental results. Our new operator *intermarriage crossover* is able to produce equal or more promising progenies from its parents. This helps in solving CSPs efficiently. ICHEA works well in both qualitative and quantitative search space. We have also identified the characteristics of CPs and divided them into four groups as described in Chapter 2. Each of these types of CPs has been modeled to work in ICHEA. The experimental results have been promising and competitive with other state-of-the-art EAs. The other major objective was the realization of the importance of incrementality in solving CPs. The partial solutions or previously solved solutions are utilized to solve many benchmark dynamic CPs and large static CPs successfully and efficiently. The change in constraints does not force ICHEA to regenerate the whole solution. Theoretically, incrementality can also help in doing what-if analysis for CPs as described in Chapter 6; however, its practicality and efficiency on a real world problem has been left for future work.

Several research ideas evolved from the investigation and are proposed for possible future extension of the research outcomes. It has been observed that hyper-heuristics work well for exam timetabling problem. We would like to make hyper-heuristic version of ICHEA that will take the incremental approach in solving the timetabling problems. We still need to work

for improving generalization of the optimization techniques which currently is specific to the class of problems being solved. The future work also involves deploying ICHEA in multi-agent environment where each constraint or a subset of constraints is handled by autonomous agent (Barták and Salido, 2011). Any update, change, conflicts in constraints would be handled by its owner agent who relays the changes to other agents sanction to derive the final solution.

Appendix A

Derivation of Formulas

A.1. Time complexity of *intermarriage* crossover for N-Queen problem

Intermarriage crossover takes two parents randomly and produces two progenies. For N-Queen problem firstly the non-duplicate values in each of the parents has to be found. The corresponding non-duplicate values are appended to each other chromosomes. Number of operations required to find the non-duplicate values in two chromosomes are computed below.

A zero vector $\boldsymbol{A} = \langle a_1, a_2, \cdots, a_N \rangle$ of cardinality N is created where N is the size of the N-Queen problem i.e. $A = \mathbf{0}$ and $|A| = N$. Two chromosomes can be represented by $\boldsymbol{C_1} = \langle c_1, c_2, \cdots, c_{l_1} \rangle$ and $\boldsymbol{C_2} = \langle c_1, c_2, \cdots, c_{l_2} \rangle$ where l_1 and l_2 are lengths of chromosomes $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ respectively.

	0	1	2	3	4
	0	0	0	0	0
C_1 = {2,3}	0	0	1	1	0
C_2 = {0,2}	10	0	11	1	0

Duplicate value(s) are {2} and non-duplicate values for C_1 is {3} and C_2 is{0}

Fig. A.1. Finding non-duplicate values in 2 chromosomes

Set $\forall a_{c_i} = a_{c_i} + 1$ where $\forall c_i \in \boldsymbol{C_1}$ and $\forall a_{c_i} = a_{c_i} + 10$ where $\forall c_i \in \boldsymbol{C_2}$. Fig. A.1 shows an example of finding non-duplicate values for two chromosomes $\boldsymbol{C_1} = \{2, 3\}$ and $\boldsymbol{C_2} = \{0, 2\}$. Firstly value 1 is added at location 2 and 3 on vector $\boldsymbol{A}$ for $\boldsymbol{C_1}$ then value 10 is added on the same vector A at location 0 and 2 for $\boldsymbol{C_2}$. This makes $\boldsymbol{A} = \langle 10, 0, 11, 1, 0 \rangle$. Iterating through

vector $\boldsymbol{A}$ gives duplicate and non-duplicate values. Locations of value 11 in the vector $\boldsymbol{A}$ gives duplicate values and locations of 1 and 10 gives the non-duplicate values for chromosomes $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ respectively. In this case duplicate value is $\{2\}$ and non-duplicate values are $\{3, 0\}$.

The set of non-duplicate values for $\boldsymbol{C_1}$ is: $\boldsymbol{D_1} = \{x \in \boldsymbol{A} | x = 1\}$ and set of non-duplicate values for $\boldsymbol{C_2}$ is $\boldsymbol{D_2} = \{x \in \boldsymbol{A} | x = 10\}$. The length of these non-duplicate sets can be given as $l'_1 = |\boldsymbol{D_1}|$ and $l'_2 = |\boldsymbol{D_2}|$. The computational requirement for finding these non-duplicate values can be given as:

l_1 iterations are required to add 1 to vector A for the corresponding c_i values of $\boldsymbol{C_1}$ and l_2 iterations are required to add 10 to the same vector A for the corresponding c_i values of $\boldsymbol{C_2}$. Then N iterations are required to locate value 11 in the vector $\boldsymbol{A}$. Hence the total operations T_1 is:

$$T_1 = l_1 + l_2 + N \tag{A.1}$$

The operations required to append the non-duplicate allele values to each other parents. Operations required to append l'_1 values to $\boldsymbol{C_2}$ and l'_2 values to $\boldsymbol{C_1}$ is given in Eq. (A.2) and Eq. (A.3).

$$l_2 + (l_2 + 1) + (l_2 + 2) + \cdots + (l_2 + l'_1) \tag{A.2}$$

$$l_1 + (l_1 + 1) + (l_1 + 2) + \cdots + (l_1 + l'_2) \tag{A.3}$$

Eq. (A.2) and Eq. (A.3) can be resolved as Eq. (A.4) and Eq. (A.5) respectively.

$$l_2 l'_1 + \frac{(l'_1 - 1)l'_1}{2} \tag{A.4}$$

$$l_1 l'_2 + \frac{(l'_2 - 1)l'_2}{2} \tag{A.5}$$

Now total operations T_2 required for appending non-duplicate valued to each other is given in Eq. (A.6).

$$T_2 = l_2 l'_1 + \frac{(l'_1 - 1)l'_1}{2} + l_1 l'_2 + \frac{(l'_2 - 1)l'_2}{2}$$

$$\leq l_2 l_1' + {l_1'}^2 + l_1 l_2' + {l_2'}^2$$

$$= l_1'(l_2 + l_1') + l_2'(l_1 + l_2')$$

$$\leq l_1'(N) + l_2'(N) \tag{A.6}$$

Hence the time complexity T after including both terms from Eq. (A.1) and Eq. (A.6) is given in (A.7):

$$T = T_1 + T_2$$

$$T = (l_1 + l_2 + N) + N(l_1' + l_2') \tag{A.7}$$

A.2. Preference based penalty function

When constraints are to be solved according to the preferences then Eq. (6.7) can be used to solve the given COP. This equation gives higher fitness to the solution where constraints are solved with higher preferences. For example if two solutions $\langle 5, 3, 0, 2\rangle$ and $\langle 5, 4, 0, 1\rangle$ have total of 10 constraints where each element indicates the number of solved constraints with the order of preferences from highest to lowest. For the first solution 5 constraints are solved with the highest preference and 3 constraints are solved with second highest preference and so on. The second solution has better fitness as constraints solved with first order preference is same but the second solution solves more constraints with second order of preference. Eq. (6.7) can be solved easily with proof by contradiction. Eq. (6.7) can be restated as:

$$f(l) = \sum_{p=0}^{D} l_p(\mu)^{D-p} \tag{A.8}$$

where $f(l) \Rightarrow$ fitness function for $l = \langle l_0, \ldots, l_D\rangle$ satisfied constraints

$L \Rightarrow$ Total constraints where $L > 0$

$l_p \Rightarrow$ Total degree of satisfaction with p^{th} preference.

$\mu \Rightarrow$ A positive constant.

$D \Rightarrow$ Lowest preference defined (highest numeric value for preference) where $D > 0$

$p \Rightarrow$ Preference for constraints in domain $[0\ D]$

To proof by contradiction we use a proposition $H1: f(l) > f(l')$ for $\forall l_k = l'_k$ where $k = \{0,1,2, \dots, i-1\}$ and $l_i > l'_i$. The idea is to give higher fitness to the solutions that satisfy more constraints with higher preferences.

Let us say proposition $H1$ does not hold for $l_i > l'_i$ i.e $\neg H1 = f(l) \leq f(l')$

$$f(l) \leq f(l')$$

$$\mu^{D-0}(l_0) + \mu^{D-1}(l_1) + \dots + \mu^{D-i}(l_i) + \dots + \mu^{D-D}(l_D) \leq$$
$$\mu^{D-0}(l'_0) + \mu^{D-1}(l'_1) + \dots + \mu^{D-i}(l'_i) + \dots + \mu^{D-D}(l'_D)$$

Since $l_k = l'_k \ where\ k = \{0,1,2, \dots, i-1\}$ we have:

$$\mu^{D-i}(l_i - l'_i) + \cdots + \mu^{D-D}(l_D - l'_D) \leq 0$$

$$\mu^{D-i}(l_i - l'_i) \leq \mu^{D-D}(l'_D - l_D) + \cdots + \mu^{D-(i+1)}\left(l'_{(i+1)} - l_{(i+1)}\right)$$

So the upper bound of the RHS can be transformed into a geometric series for $\forall l'_k = L^2$ and $l_k = 0$ where $(i+1) < k \leq D$. L^2 is the maximum possible value that any l_k or l'_k can have determined through the summation of constraint satisfaction for k^{th} preference $(L-1) + (L-2) + \cdots + 1 = \frac{L(L-1)}{2} < L^2$ of all the constraints. First constraint can make maximum of $(L-1)$ satisfaction with k^{th} preference, second constraint can make maximum of $(L-2)$ satisfaction with k^{th} preference, and so on.

$$\therefore \mu^{D-i}(l_i - l'_i) \leq \mu^0(L^2) + \cdots + \mu^{D-(i+1)}(L^2) = L^2 \frac{(\mu^{D-(i+1)+1} - 1)}{\mu - 1}$$

$$\mu^{D-i}(l_i - l'_i) \leq \mu^{D-i}(\frac{L^2}{\mu - 1})$$

$$(l_i - l'_i) \leq \frac{L^2}{\mu - 1}$$

Since the expression $(l_i - l'_i)$ contradicts if it is < 1, $\therefore \frac{L^2}{\mu-1}$ should also be < 1.

$$\Rightarrow \mu > L^2 + 1$$

We can choose any value for $\mu > L^2 + 1$. One possible option is $L^2 + 2$. Hence:

$$(l_i - l'_i) \leq \frac{L^2}{\mu - 1} < \frac{L^2}{L + 1} < 1$$

Since $l_i > l'_i$ and both are whole numbers so the above statement is contradictory. Let say $l_i = l'_i + x$ where $x \geq 1$ then the above equation can be rewritten as:

$$l'_i + x - l'_i < 1$$

$$x < 1 \text{ (Contradiction)}$$

This demonstrates proposition $\neg H1$ does not hold. Hence our hypothesis $H1: f(l) > f(l')$ is true for $\forall l_k = l'_k$ where $k = \{0,1,2, \dots, i-1\}$ and $l_i > l'_i$. ■

The Proof of Eq. (6.8) for minimization of l_p for lower preferences is similar to above. If two solutions $\langle 3, 5, 2, 2 \rangle$ and $\langle 5, 1, 3, 2 \rangle$ have total of 10 constraints where each element indicates the number of solved constraints with the order of preferences from highest to lowest. One constraint can overlap with other constraints with different preferences. The first solution has better fitness as constraints solved with last order of preference is same but the first solution has less constraints of unwanted (high order) preferences. We use a proposition $H1: f(l) > f(l')$ for $\forall l_k = l'_k$ where $k = \{i+1, \dots, D\}$ and $l_i < l'_i$. The idea is to give higher fitness to the solutions that minimizes the constraint satisfaction with high order preferences. The generic form of Eq. (6.8) is:

$$f(l) = \sum_{p=0}^{D} (L^2 - l_p)(\mu)^p \tag{A.9}$$

Let us say proposition $H1$ does not hold for $l_i < l'_i$ i.e $\neg H1 = f(l) \leq f(l')$

$$f(l) \leq f(l')$$

$$\mu^0(L^2 - l_0) + \mu^1(L^2 - l_1) + \dots + \mu^i(L^2 - l_i) + \dots + \mu^D(L^2 - l_D) \leq$$
$$\mu^0(L^2 - l'_0) + \mu^D(L^2 - l'_1) + \dots + \mu^i(L^2 - l'_i) + \dots + \mu^D(L^2 - l'_D)$$

Since $l_k = l'_k \; where \; k = \{i+1, \dots, D\}$ we have:

$$\mu^0(l'_0 - l_0) + \cdots + \mu^i(l'_i - l_i) \leq 0$$

$$\mu^0(l'_0 - l_0) + \cdots + \mu^{(i-1)}\left(l'_{(i-1)} - l_{(i-1)}\right) \leq \mu^i(l_i - l'_i)$$

So the lower bound of the LHS can be similarly transformed into geometric series for $\forall l'_k = 0$ and $l_k = L^2$ where $0 < k \leq i - 1$.

$$\mu^0(-L^2) + \cdots + \mu^{(i-1)}(-L^2) \leq \mu^0(l'_0 - l_0) + \cdots + \mu^{(i-1)}\left(l'_{(i-1)} - l_{(i-1)}\right)$$

$$\therefore \mu^i(l_i - l'_i) \geq (-L^2)\frac{\mu^i - 1}{\mu - 1} = \frac{L^2 - L^2\mu^i}{\mu - 1} \geq \frac{-L^2\mu^i}{\mu - 1}$$

$$\frac{-L^2\mu^i}{\mu - 1} \leq \mu^i(l_i - l'_i)$$

$$(l_i - l'_i) \geq \frac{L^2}{1 - \mu}$$

Since the expression $(l_i - l'_i)$ should be ≤ -1, it contradicts if it is > -1, $\therefore \frac{L^2}{1-\mu}$ should also be > -1. Note that μ is a positive integer that makes the expression $(1 - \mu)$ a negative integer.

$$\Rightarrow \mu > 1 + L^2$$

We can choose any value for $\mu > 1 + L^2$. One possible option is $L^2 + 2$. Hence:

$$(l_i - l'_i) \geq \frac{L^2}{1 - \mu} = \frac{L^2}{-1 - L^2} = -\left(\frac{L^2}{L^2 + 1}\right) > -1$$

Since $l_i < l'_i$ and both are whole numbers so the above statement is contradictory. Let say $l_i = l'_i - x$ where $x \geq 1$ then the above equation can be rewritten as:

$$l'_i - x - l'_i = -x > -1 \text{ or } x < 1 \text{ (Contradiction)}$$

This demonstrates proposition $\neg H1$ does not hold. Hence our hypothesis $H1: f(l) > f(l')$ is true for $\forall l_k = l'_k$ where $k = \{i + 1, \ldots, D\}$ and $l_i < l'_i$. ■

The proof of Eq. (6.9) needs to demonstrate higher functional value for higher value of $l = \sum_{p=0}^{D} l_p$. The maximum value of the expression $(L^2 - l_p)$ in Eq. (6.8) is L^2 and the maximum possible value for Eq. (6.8) using sum of geometric series is:

$$f(l) \leq L^2 \frac{(L^2+2)^{D+1}-1}{(L^2+2)-1} < \frac{L^2((L^2+2)^{D+1}-1)}{L^2} < (L^2+2)^{D+1}$$

Using the maximum value derived above Eq. (6.8) can be modified as:

$$f(l) = l(L^2+2)^{D+1} + \sum_{p=0}^{D}(L^2 - l_p)(L^2+2)^p$$

which can be easily proven by showing proof by contradiction.

A.3. Selecting infeasible solutions

If the population size of infeasible solutions towards the end of a generation is $|pop_{CSP}| + k$ where k is added infeasible solutions in a given generation then we need to pick the solutions that have good fitness and that also represent the diverse population. Firstly, the population is sorted decreasingly according to the fitness value. The purpose is to retain good solutions with high fitness values to keep the search focus towards better solutions while maintaining the diversity by keeping solutions from different regions in the search space even with low fitness values. Mathematically, the selection of solutions is heavy towards the initial indices and lighter towards the higher index of the population set. It can be formulated using the following exponential function in Eq. (A.10):

$$f(i) = e^{-\rho\left(\frac{|POP_{CSP}|-i}{|POP_{CSP}|}\right)} \tag{A.10}$$

Function $f(i)$ gives the desired distribution of the solutions to be selected from the population where $i \in \{1, \ldots, |POP_{CSP}|\}$ and ρ defines the level of curve whether extreme, normal or least. Fig. A.2 shows the effect of different values of ρ on the exponential function. Continuous values of function $f(i)$ need to be mapped to discrete indices of the population of sorted solutions which can be done using the following transformation function $f'(i)$ in Eq. (A.11):

$$f'(i) = \left\lfloor (|POP_{CSP}| + k)\frac{f(i)-f(1)}{1-f(1)} \right\rfloor \tag{A.11}$$

The mapping function $f'(i)$ in Eq. (A.11) gives duplicate indices which are again transformed to the nearest indices using function $f''(i)$ in Eq. (A.12) that ensure all unique solutions to be selected from the population.

$$f''(i) = \begin{cases} f'(i-1) + 1, & if\ f'(i) \leq f'(i-1) \\ f'(i), & otherwise \end{cases} \quad \text{(A.12)}$$

To see a numerical example using Eq. (A.10) – Eq. (A.12), we choose $\rho = 5$, $|POP_{CSP}| = 10$ and $k = 40$. The result from Eq. (A.10) is:

$$f = \{0.0067, 0.0117, 0.0205, 0.0357, 0.0622, 0.1084, 0.1889, 0.3292, 0.5738, 1.0000\}$$

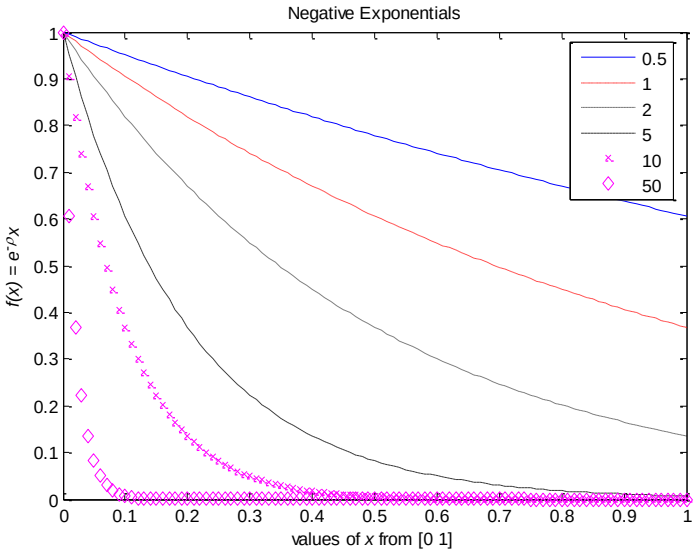


Fig. A.2. Mapping of population for selection

These continuous values of function f are transformed into function f' using Eq. (A.11) that gives the indices of the population. It can be noted that the selection of solutions is heavy towards the initial indices of the sorted population.

$$f' = \{0, 0, 0, 1, 2, 5, 8, 15, 27, 49\}$$

The output of function f' has some duplicate values which are not desired so these duplicate values are transformed to the nearest unique values using Eq. (A.12) as shown below:

$$f'' = \{0, 1, 2, 3, 4, 5, 8, 15, 27, 49\}$$

Here the index values start from 0 and ends with $|POP_{CSP}| + k - 1$ which are the first and last elements of the sorted set of infeasible solutions respectively.

Appendix B

Benchmark Problems and Solutions

B.1. Benchmark CSP dataset from COCONUT (Shcherbina, 2011)

We used following benchmark CSPs for the experiments. Each test problem describes some parameters and/or variables that are subject to the given constraints.

B.1.1. Trigonometric problem *H77*

Given the values of parameters $best\_val$ and eps three constraints c_1, c_2, c_{13} need to be satisfied.

$best\_val = 0.2415051288$ (known best value), $eps = 1$, $\delta = [0.001, 0.1]$

$$c_1 \Rightarrow est\_val + eps - (2x_1(x_1 - x_2 - 1) + 1 + x_2^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6) \geq 0$$

$$c_2 \Rightarrow -\delta \leq x_1^2 x_4 + \sin(x_4 - x_5) - 2 * S_2 \leq \delta$$

$$c_3 \Rightarrow -\delta \leq x_2 + x_3^4 x_4^2 - 8 - S_2 \leq \delta$$

B.1.2. Quadratic problem *Chem*

Given the following parameters and variables $c_1, \ldots c_5$ equality constraints need to be satisfied.

$r = 10,\ r_5 = 0.193,\ r_6 = \frac{0.002597}{\sqrt{40}},\ r_7 = \frac{0.003448}{\sqrt{40}},\ r_8 = \frac{0.00001799}{40},\ r_9 = \frac{0.0002155}{\sqrt{40}},$
$r_{10} = \frac{0.00003846}{40}$

$0 \leq y[i] \leq 1 \times 10^8$ for $i \in \{1, \ldots, 5\}$

$$c_1 \Rightarrow 3 * y[5] - y[1](y[2] + 1) = 0$$

$$c_2 \Rightarrow y[2](2y[1] + y[3]^2 + r8 + 2r_{10}y[2] + r_7 y[3] + r_9 y[4]) + y[1] - ry[5] = 0$$

$$c_3 \Rightarrow y[3] * (2 * y[2] * y[3] + 2 * r5 * y[3] + r6 + r7 * y[2]) - 8 * y[5] = 0$$

$c_4 \Rightarrow y[4](r_9 y[2] + 2y[4]) - 4ry[5] = 0$

$c_5 \Rightarrow y[2](y[1] + r_{10} y[2] + y[3]^2 + r8 + r_7 y[3] + r_9 y[4]) + \mathrm{y}[1] + r_5 \mathrm{y}[3]^2 + \mathrm{y}[4]^2 - 1 + r_6 \mathrm{y}[3]$

B.1.3. Polynomial problem *Broyden10*

The domain of variables $x[i]$ has been given subject to 10 constraints $c_1, \ldots c_{10}$.

$-1.0 \times 10^8 \leq x[i] \leq 1.0 \times 10^8$ for $i \in \{1, \ldots, 10\}$

$c_1 \Rightarrow x[1](2 + 5x[1]^2) + 1 - \big(x[1](1 + x[1]) + x[2](1 + x[2])\big) = 0$

$c_2 \Rightarrow x[2](2 + 5x[2]^2) + 1 - (x[1](1 + x[1]) + x[3](1 + x[3])) = 0$

$c_3 \Rightarrow x[3](2 + 5x[3]^2) + 1 - \big(x[1](1 + x[1]) + x[2](1 + x[2]) + x[4](1 + x[4])\big) = 0$

$c_4 \Rightarrow x[4](2 + 5x[4]^2) + 1 - (x[1](1 + x[1]) + x[2](1 + x[2]) + x[3](1 + x[3]) + x[5](1 + x[5])) = 0$

$c_5 \Rightarrow x[5](2 + 5x[5]^2) + 1 - (x[1](1 + x[1]) + x[2](1 + x[2]) + x[3](1 + x[3]) + x[4](1 + x[4]) + x[6](1 + x[6])) = 0;$

$c_6 \Rightarrow x[6](2 + 5x[6]^2) + 1 - (x[1](1 + x[1]) + x[2](1 + x[2]) + x[3](1 + x[3]) + x[4](1 + x[4]) + x[5](1 + x[5]) + x[7](1 + x[7])) = 0$

$c_7 \Rightarrow x[7](2 + 5x[7]^2) + 1 - (x[2](1 + x[2]) + x[3](1 + x[3]) + x[4](1 + x[4]) + x[5](1 + x[5]) + x[6](1 + x[6]) + x[8](1 + x[8])) = 0$

$c_8 \Rightarrow x[8](2 + 5x[8]^2) + 1 - (x[3](1 + x[3]) + x[4](1 + x[4]) + x[5](1 + x[5]) + x[6](1 + x[6]) + x[7](1 + x[7]) + x[9](1 + x[9])) = 0$

$c_9 \Rightarrow x[9](2 + 5x[9]^2) + 1 - (x[4](1 + x[4]) + x[5](1 + x[5]) + x[6](1 + x[6]) + x[7](1 + x[7]) + x[8](1 + x[8]) + x[10](1 + x[10])) = 0$

$c_{10} \Rightarrow: x[10](2 + 5x[10]^2) + 1 - (x[5](1 + x[5]) + x[6](1 + x[6]) + x[7](1 + x[7]) + x[8](1 + x[8]) + x[9](1 + x[9])) = 0$

B.1.4. Trigonometric problem *HS109*

With the following parameters and variables solve 15 constraints $c_1, \dots c_{15}$

$a = 50.176,\ \mathrm{b1} = 0.25,\ \mathrm{b} = \sin(\mathrm{b1}),\ \mathrm{c} = \cos(\mathrm{b1}),$
$best\_val = 5327.541669,\ eps = 53.27541669$ (1% of $best\_val$)

$-1.0 \times 10^8 \leq x[i] \leq 1.0 \times 10^8$ for $i \in \{1, \dots, 10\}$

$c_1 \Rightarrow 3\mathrm{x}[1] + 1 \times 10^{-6}\mathrm{x}[1]^3 + 2\mathrm{x}[2] + 0.522074 \times 10^{-6}\mathrm{x}[2]^3 \leq best\_val + eps$

$c_2 \Rightarrow x[4] - x[3] + 0.55 >= 0$

$c_3 \Rightarrow x[3] - x[4] + 0.55 >= 0;$

$c_4 \Rightarrow 2250000 - x[1]^2 - x[8]^2 >= 0$

$c_5 \Rightarrow 2250000 - x[2]^2 - x[9]^2 >= 0$

$c_6 \Rightarrow x[5]x[6]sin(-x[3] - 0.25) + x[5]x[7]sin(-x[4] - 0.25) + 2\,b\,x[5]^2 - a\,x[1] + 400a = 0$

$c_7 \Rightarrow x[5]x[6]sin(x[3] - 0.25) + x[6]x[7]sin(x[3] - x[4] - 0.25) + 2bx[6]^2 - ax[2] + 400a = 0$

$c_8 \Rightarrow x[5]x[7]sin(x[4] - 0.25) + x[6]x[7]sin(x[4] - x[3 - 0.25) + 2\,b\,x[7]^2 + 881.779a = 0$

$c_9 \Rightarrow ax[8] + x[5]x[6]cos(-x[3] - 0.25) + x[5]x[7]cos(-x[4] - 0.25) - 200a - 2cx[5]^2 + 0.7533 \times 10^{-3}ax[5]^2 = 0$

$c_{10} \Rightarrow ax[9] + x[5]x[6]cos(x[3] - 0.25) + x[6]x[7]cos(x[3] - x[4] - 0.25) - 2cx[6]^2 + 0.7533 \times 10^{-3}ax[6]^2 - 200a = 0$

$c_{11} \Rightarrow x[5]x[7]cos(x[4] - 0.25) + x[6]x[7]cos(x[4] - x[3] - 0.25) - 2cx[7]^2 + 22.938a + 0.7533 \times 10^{-3}ax[7]\ \text{^}2 = 0$

$c_{12} \Rightarrow x[i] \geq 0$ for $i \in \{1,2\}$

$c_{13} \Rightarrow -0.55 \leq x[i] \leq 0.55$ for $i \in \{3,4\}$

$c_{14} \Rightarrow 196 \leq x[i] \leq 252 \; for \; i \in \{5, \ldots, 7\}$

$c_{15} \Rightarrow -400 \leq x[i] \leq 800 \; for \; i \in \{8,9\}$

B.2. Best Solutions for Benchmark Timetabling Problems by ICHEA

We used well known benchmark problems from Version I of ("Benchmark Exam Timetabling Datasets," 2013) for exam timetabling problems reported in several articles. The best solutions achieved by incremental ICHEA are shown below where comma separated values represent slot number starting from 0 to one less than maximum available slots. We used almost same format for the solutions as required by the evaluator program in ("Benchmark Exam Timetabling Datasets," 2013) where slots are sorted according to the exam numbers. For example the solution for *Car9i* describes that exam 1 is placed in slot 27, exam 2 is placed on slot 27, exam 3 is placed on slot 22 and so on. We did not use exam numbers in the solution to save space.

B.2.1. Problem Car91

1. Exams: 682
2. Students: 16925
3. Enrollments: 56877
4. Timeslots: 35
5. Solution:
   27,27,22,1,13,32,8,19,2,15,29,4,34,18,6,14,24,4,5,7,10,13,34,18,4,9,24,11,14,7,1,0,5,3,
   23,27,0,5,12,16,10,25,1,27,26,29,24,29,18,18,18,23,27,18,23,17,22,1,1,6,24,13,29,25,9
   ,3,26,29,0,16,0,32,17,17,15,11,34,18,25,10,27,20,29,32,7,23,8,16,4,18,0,20,31,9,9,11,0
   ,25,8,33,20,17,22,15,11,18,7,22,3,17,16,25,18,31,5,8,34,8,12,12,4,0,6,22,2,31,9,28,34,
   16,12,9,8,12,23,31,18,26,7,0,17,2,34,8,0,34,15,29,2,2,2,31,31,12,12,11,16,30,10,31,34,
   21,1,29,3,14,24,5,33,18,10,33,19,13,23,1,20,28,2,16,6,25,10,4,22,34,18,13,20,26,34,31
   ,32,27,31,30,1,34,34,18,30,5,18,12,7,30,24,14,11,6,30,21,3,28,7,28,10,23,3,12,16,34,2
   5,11,21,8,33,10,34,21,1,16,28,30,12,4,26,21,27,3,2,6,15,25,18,3,10,9,34,15,2,27,23,3,5
   ,22,0,6,34,14,0,24,16,17,16,0,2,19,33,21,22,28,26,9,1,5,5,29,33,2,0,22,18,17,30,13,13,
   29,6,7,20,34,11,10,34,28,16,4,28,28,31,7,3,28,23,23,23,25,24,27,24,24,13,13,2,11,20,5
   ,32,29,6,31,32,34,21,20,8,14,14,25,13,6,22,8,2,17,22,0,5,11,10,4,15,20,0,24,10,34,27,2
   0,20,20,20,20,20,20,20,20,3,3,8,26,3,15,13,9,12,33,25,25,25,19,19,17,27,14,9,29,34,5,

0,6,2,13,31,23,9,8,1,22,33,34,34,34,34,34,34,34,20,20,25,20,20,32,32,4,4,2,4,22,22,24,18,15,15,31,27,27,11,9,9,13,33,0,6,24,14,29,29,22,34,12,25,33,6,3,8,17,34,21,5,34,0,0,0,0,0,28,34,8,1,23,23,18,32,7,23,18,18,18,30,5,12,21,22,10,15,31,0,20,7,3,18,8,11,0,19,14,24,26,20,19,14,14,14,15,14,25,32,26,26,2,28,5,2,12,6,16,22,17,34,7,27,10,20,6,26,24,9,30,23,31,33,5,12,15,34,34,10,30,16,25,0,20,33,9,24,20,12,30,30,0,15,17,27,6,17,29,14,13,13,25,25,22,29,13,24,2,16,0,20,33,25,19,8,18,0,34,3,12,31,32,4,19,12,19,10,15,0,34,27,32,18,34,26,2,3,30,13,13,0,34,21,1,2,9,17,15,27,7,22,34,8,23,14,18,18,9,6,6,4,29,15,7,28,11,31,17,25,32,8,16,34,7,18,28,13,12,18,26,15,7,19,13,26,7,25,0,34,4,7,11,15,17,22,24,17,18,23,9,21,32,33,30,15,26,32,24,28,16,22,4,0,15,34,0,1,24,21,1,9,8,24,13,21,5,0,28,10,3,6,25,13,19,5,29,6,34,15,20,10,25.

6. Cost: 4.907

B.2.2. Problem Car92

1. Exams: 543
2. Students: 18419
3. Enrollments: 55522
4. Timeslots: 32
5. Solution:
   9,27,31,2,18,20,16,25,8,29,20,19,25,29,5,30,30,2,22,12,8,0,10,24,22,25,20,22,20,20,22,20,22,22,22,5,13,21,23,1,6,24,23,21,19,23,29,27,12,28,9,2,3,18,24,7,21,31,0,16,30,10,22,4,31,31,4,10,15,14,4,6,30,24,27,10,14,15,3,31,26,7,4,22,19,8,18,13,0,23,0,2,21,15,28,27,5,17,28,24,16,30,13,6,30,22,8,23,26,3,15,16,2,18,31,11,25,1,26,30,4,14,3,19,13,6,6,29,2,11,23,5,15,15,3,12,23,18,6,26,15,20,22,0,0,26,9,5,29,15,3,19,20,9,16,2,31,22,13,16,18,31,25,28,13,22,27,17,0,24,31,23,24,25,25,10,6,4,11,28,31,14,15,6,24,9,2,22,0,0,10,17,18,25,7,15,4,31,31,13,12,6,7,18,18,19,19,19,7,7,8,2,8,31,11,8,22,26,16,14,10,28,0,24,3,2,16,13,29,20,7,30,3,13,23,14,23,4,10,24,1,17,15,30,0,4,31,25,21,1,7,20,31,5,10,11,11,11,26,11,10,25,21,21,9,27,9,2,18,11,3,12,15,21,24,29,3,24,0,14,10,17,13,31,7,28,0,0,0,0,0,21,8,21,21,26,29,10,5,30,15,25,1,1,13,13,18,11,26,7,0,30,2,16,7,24,0,24,10,16,4,7,14,19,1,25,15,0,31,31,31,31,17,19,2,26,20,10,27,6,28,5,4,0,26,31,29,31,15,11,0,9,17,23,7,7,5,7,7,6,4,16,3,22,23,19,14,25,8,26,13,30,1,26,11,7,20,14,9,8,15,27,3,16,7,23,28,20,23,29,17,11,7,17,0,1,28,27,11,11,7,28,19,14,7,2,5,11,30,12,24,1,6,24,0,25,31,30,12,16,15,30,29,2,1,13,13,17,15,31,22,9,30,5,10,14,20,29,9,1,26,17,11,3,31,15,4,1,12,

31,25,21,21,20,25,22,0,6,25,21,1,21,0,31,0,21,8,14,31,9,25,31,15,15,15,9,23,0,14,29,5,28,30,25,0,31,15,19,6,25,2,21,1,7,25,14,31,30,6,10,31,2,16,19,19,13,0,0,25,30,15,0,8,30,17,23,0,14,9,26,9,27,13,6,0,5,23,28,7,16,19,30,23,30,13,6,12,24,11,26,1,21,9

6. Cost: 4.079

B.2.3. Problem Ear83

1. Exams: 190
2. Students: 1125
3. Enrollments: 8109
4. Timeslots: 24
5. Solution:
   15,8,15,15,8,4,14,12,12,12,6,19,11,11,4,9,6,0,0,0,0,0,1,0,0,0,1,0,2,0,10,20,16,19,10,20,20,18,17,20,13,18,13,3,19,17,16,20,20,16,17,16,17,20,10,23,18,7,18,20,3,3,14,3,3,3,3,3,3,4,4,13,10,11,5,20,9,23,23,23,23,23,23,23,23,23,23,23,22,22,23,7,7,6,13,10,7,13,19,14,14,14,14,5,12,6,13,5,6,20,11,5,10,12,8,9,9,9,9,9,9,9,9,4,17,21,21,14,14,7,7,6,6,17,17,20,20,16,8,7,19,17,17,2,11,13,11,19,5,13,13,5,6,7,5,7,5,9,8,8,14,23,14,5,9,13,16,4,17,16,10,9,14,14,8,15,21,15,14,4,11,14,11,11,19,19,6,10,10,11
6. Cost: 33.244

B.2.4. Problem Hec92

1. Exams: 81
2. Students: 2823
3. Enrollments: 10632
4. Timeslots: 18
5. Solution:
   5,4,12,13,0,0,0,2,0,16,17,8,9,2,1,13,7,7,6,12,11,14,3,1,0,0,16,15,15,10,17,9,4,17,10,16,4,0,17,2,15,6,0,14,17,12,2,1,4,10,6,17,9,5,13,7,15,12,3,0,17,9,3,16,6,0,12,11,9,8,1,3,5,13,1,17,11,14,7,8,13
6. Cost: 10.133

B.2.5. Problem Kfu93

1. Exams: 461
2. Students: 5349
3. Enrollments: 25113
4. Timeslots: 20
5. Solution:
   10,10,6,12,15,19,0,1,15,9,6,0,12,0,13,0,15,19,0,17,13,6,2,11,0,19,10,5,0,9,15,2,5,11,19,12,0,6,9,1,17,2,2,0,11,18,0,16,0,10,14,13,12,18,18,19,16,0,2,19,15,0,19,6,13,6,19,19,0,0,9,0,15,9,10,13,0,19,7,19,13,0,10,0,17,0,0,18,14,15,0,11,19,5,2,18,2,17,0,11,10,14,18,3,19,12,0,6,13,19,0,0,5,4,14,14,5,0,2,0,16,0,12,16,3,0,9,19,0,10,18,19,12,6,3,12,18,0,19,0,14,0,15,19,15,18,10,11,17,6,14,0,2,19,0,12,6,9,0,13,12,0,6,0,0,14,4,6,18,3,15,4,5,16,19,16,1,0,4,5,14,0,15,4,13,0,0,19,13,1,7,17,15,16,6,10,16,0,4,19,10,19,4,15,19,9,0,0,9,3,3,3,0,9,11,11,9,16,12,6,14,2,18,4,19,10,15,9,0,3,13,6,0,9,12,12,6,11,14,5,12,16,1,9,0,15,6,8,0,7,7,8,7,7,8,7,8,10,1,2,10,15,17,1,5,19,18,4,1,2,12,5,18,0,17,16,13,19,6,19,0,13,12,6,4,18,19,18,19,17,9,19,19,17,11,19,17,14,5,19,3,0,10,19,6,19,8,13,0,9,17,15,12,6,5,5,2,11,11,15,0,1,1,10,18,18,14,16,11,9,5,8,1,11,15,3,17,9,2,13,0,12,15,19,2,13,6,19,0,0,0,16,18,18,0,13,3,17,19,18,0,0,10,0,19,19,14,9,13,11,19,19,7,13,19,6,0,14,1,17,0,5,11,19,5,6,2,7,6,16,13,13,13,13,13,12,12,12,12,0,5,5,6,9,0,4,15,16,10,2,19,16,16,16,16,9,15,19,0,3,16,10,0,6,19,12,0,6,19,15,2,11,19,17,10,18,0,1,1,19,4,15,0,5,14,0,11,10,1,3,16,19,7,6,13,19,0,0,4,7,9
6. Cost: 13.58

B.2.6. Problem Lse91

1. Exams: 381
2. Students: 2726
3. Enrollments: 10918
4. Timeslots: 18
5. Solution:
   4,15,0,6,17,14,17,14,9,7,1,17,0,7,17,15,11,13,4,17,0,1,0,0,11,11,1,1,1,17,9,12,14,7,5,0,6,15,3,10,13,7,5,2,17,17,12,4,0,17,6,17,0,6,4,8,2,5,10,16,8,16,8,17,13,3,10,12,6,10,13,0,15,15,6,8,9,7,6,12,12,0,13,15,16,16,4,0,5,16,4,0,0,14,6,3,2,17,9,0,9,17,17,10,2,16,8,1

4,17,13,12,4,11,17,5,11,16,1,6,17,5,0,2,13,13,0,7,17,1,0,8,7,8,5,13,1,5,10,10,4,17,9,7,0,12,4,2,8,17,0,7,0,13,9,14,2,0,14,0,6,17,4,9,1,0,3,7,0,0,15,14,7,10,4,12,10,17,10,17,0,4,6,12,15,9,14,4,17,10,15,10,4,17,7,7,0,17,17,1,11,17,7,13,5,7,0,7,0,3,17,2,10,17,17,17,17,6,17,13,16,0,11,6,9,17,2,13,5,14,11,17,8,10,1,14,6,4,13,14,12,12,4,11,11,13,0,7,5,7,13,14,5,0,6,14,0,6,3,3,13,17,1,4,2,4,14,3,5,3,8,2,2,15,13,0,7,12,0,17,1,11,5,12,7,0,17,3,9,10,0,16,12,16,6,6,9,0,17,15,15,3,13,17,3,0,11,0,13,6,17,6,0,0,6,12,11,10,3,13,11,6,12,11,0,17,13,17,10,17,15,5,4,9,13,16,16,16,0,15,17,16,10,0,11,16,0,7,3,14,13,7,16,7,10,17,1,7,10,13,13,8,8,5,0,6,3,10,12,17,2,4,8,2,8,0,6,0,2,0,0,0

6. Cost: 10.371

B.2.7. Problem Pur93

1. Exams: 2419
2. Students: 30029
3. Enrollments: 120681
4. Timeslots: 42
5. Solution:
   41,39,20,16,34,41,20,6,0,28,41,8,24,11,37,4,10,29,26,17,34,0,16,36,6,39,3,2,20,26,34,10,15,36,20,22,30,14,41,26,2,40,41,8,38,32,41,35,27,37,33,37,31,38,36,12,15,34,41,4,5,11,20,0,23,27,19,27,13,22,19,7,10,34,25,19,9,41,12,0,39,9,15,32,31,12,24,21,8,27,40,31,39,7,14,21,4,6,19,39,16,5,15,25,32,36,26,33,0,27,16,41,14,32,17,41,16,0,19,16,8,21,23,4,30,35,3,13,27,33,40,37,35,25,18,36,19,25,26,21,31,12,11,27,32,20,34,34,14,10,26,8,30,41,8,25,21,18,3,32,1,10,37,31,12,0,25,26,31,20,41,21,26,36,19,37,7,4,13,8,18,41,27,21,22,3,19,29,17,39,32,11,34,20,41,0,7,3,32,24,37,15,1,39,27,20,24,5,8,10,8,18,7,41,41,21,0,27,5,12,4,32,38,19,36,33,24,7,16,10,33,26,15,41,0,35,37,10,37,5,1,13,22,5,16,6,19,29,30,25,36,3,0,9,18,25,40,30,10,6,38,22,24,32,7,41,13,33,30,30,41,6,39,2,34,14,37,0,15,41,31,27,22,13,0,30,23,20,38,33,15,16,28,17,6,37,12,18,0,27,34,34,36,3,29,23,3,37,8,24,11,35,39,0,33,33,8,14,14,1,10,16,29,21,21,0,13,7,41,29,24,10,26,24,36,30,2,4,20,0,31,7,11,34,40,24,41,25,27,38,12,7,38,12,31,14,8,0,29,27,15,2,28,33,38,36,7,12,0,10,21,5,34,41,15,30,39,38,22,0,12,0,0,12,40,39,40,12,23,20,41,34,6,14,37,24,30,17,28,35,24,17,36,26,39,18,23,41,21,13,6,21,27,32,25,29,26,22,2,21,26,36,41,34,21,30,1,9,4,26,19,41,12,30,13,7,34,23,40,41,17,38,6,3,1,40,0,9,18,2,26,16,34,29,20,32,8,35,29,41,23,9,16,7,0,23,15,4,41,1,23,6,1,12,40,19,26,23,6,0,38,20,13,6,30,41,33,31,15,30

,23,19,36,7,41,25,41,1,41,41,2,38,2,9,34,37,35,20,31,36,1,20,34,27,0,28,20,18,5,3,12,1
4,33,21,33,14,37,0,1,30,10,8,11,41,0,17,32,6,24,34,4,39,0,26,20,0,29,14,7,12,37,1,19,1
9,25,39,3,14,17,25,7,2,41,37,36,19,4,12,23,37,27,29,29,29,28,27,26,9,27,1,20,28,28,5,
9,28,21,34,0,30,17,29,3,29,2,4,31,23,35,32,26,40,40,8,11,18,0,25,25,20,20,5,12,1,32,2
7,1,40,9,28,32,4,29,41,13,41,20,30,7,26,0,33,1,21,8,20,0,0,1,1,10,24,0,27,37,32,23,13,
38,41,0,13,19,1,41,31,20,20,1,0,20,27,34,35,18,24,37,0,37,23,37,38,23,21,0,25,5,21,33
,10,0,3,27,6,32,25,4,31,41,28,3,0,18,41,20,4,23,19,36,24,11,31,0,10,41,35,5,29,24,37,8
,7,26,2,23,24,23,36,0,25,20,16,34,41,12,19,17,0,28,29,15,11,22,7,41,21,18,12,6,20,3,3
0,41,35,10,31,31,39,35,23,40,30,41,19,39,2,6,34,10,41,2,22,30,36,14,41,13,16,32,0,21,
40,33,34,37,8,11,6,35,30,33,21,23,31,24,14,17,18,17,40,0,17,0,38,41,14,0,16,16,41,35,
25,26,27,18,21,31,2,39,21,34,0,16,9,41,24,3,25,29,26,33,41,32,20,6,13,37,41,17,1,38,9
,24,20,38,7,26,9,17,18,32,7,12,41,33,16,0,15,32,38,5,12,25,20,41,2,24,27,21,34,38,39,
41,28,8,18,1,10,35,9,16,6,11,9,19,11,17,5,24,32,1,24,23,23,23,40,5,37,13,17,12,32,3,2
6,3,39,24,26,32,18,34,8,18,29,17,40,0,1,8,3,16,14,38,36,21,37,0,30,10,38,0,35,9,21,15,
0,20,4,23,12,19,2,16,2,37,0,32,33,13,27,26,10,16,5,40,23,34,6,41,3,38,17,16,41,20,4,3
4,35,23,41,19,2,33,28,21,37,41,41,0,7,41,10,31,17,16,11,9,2,13,18,17,27,0,24,35,24,7,
31,5,35,11,3,31,18,34,8,33,27,30,1,9,18,24,35,6,5,3,6,10,22,2,4,1,3,1,2,9,1,3,10,11,9,3,
2,12,12,11,3,2,11,23,2,8,4,10,0,7,9,33,3,2,34,34,9,24,35,5,4,23,33,4,2,5,4,34,4,34,1,3,3
4,14,36,21,2,11,2,20,33,33,33,33,20,3,41,4,1,20,26,3,31,26,11,3,10,38,14,1,34,33,33,2
6,0,26,3,6,5,25,21,1,33,32,21,2,41,33,0,21,33,0,33,2,30,35,1,34,2,11,35,12,34,11,26,6,
3,11,0,35,1,35,34,0,37,35,1,21,33,2,1,9,36,1,20,2,3,0,10,2,3,1,33,28,19,22,26,0,30,30,1
2,11,35,13,24,33,10,38,26,2,12,36,15,11,26,14,38,9,11,26,25,3,38,4,37,26,16,4,4,4,2,3
2,26,10,0,0,34,31,41,10,36,36,22,40,11,29,31,20,22,30,5,41,29,13,28,2,24,14,19,11,0,0
,21,17,16,29,10,4,21,30,3,23,26,1,39,14,28,35,6,10,1,19,32,20,29,23,5,34,21,5,5,41,21,
24,0,22,21,18,8,5,41,24,3,29,0,9,11,32,41,0,22,33,12,18,10,36,3,32,20,20,41,6,2,26,5,2
5,40,0,10,32,41,24,30,3,13,0,5,9,41,15,16,34,25,23,38,29,0,41,11,39,16,25,41,23,25,38
,8,41,32,35,4,8,41,1,11,25,19,28,33,41,38,1,14,29,19,20,1,41,17,15,5,28,14,16,38,6,9,8
,5,12,0,13,24,6,29,8,5,31,9,25,4,7,7,40,3,0,5,10,33,39,13,1,0,17,31,2,35,26,10,7,2,0,41,
23,5,24,12,20,30,1,22,8,1,0,13,40,25,31,38,23,22,39,31,7,5,24,41,35,4,19,36,23,9,26,2
7,29,30,18,4,17,2,2,9,6,8,7,7,7,41,11,5,5,39,13,31,6,36,4,24,18,25,38,39,6,19,15,27,5,3
9,19,27,41,37,32,2,32,4,25,15,28,21,30,4,13,13,12,12,13,13,13,24,12,12,12,6,32,10,26,
22,41,10,5,29,38,4,38,0,11,27,14,1,3,41,17,16,5,32,12,35,2,0,41,39,22,6,29,39,0,24,41,
39,6,2,15,21,0,41,7,12,20,18,0,23,26,34,0,37,8,29,14,1,34,32,30,26,16,40,3,26,12,0,32,

39,41,30,14,36,6,24,9,12,26,16,8,41,36,0,6,18,2,8,31,20,30,32,31,20,10,3,28,24,4,9,12,35,24,23,6,10,21,2,6,34,17,28,32,5,11,28,30,40,0,16,11,40,41,0,16,40,14,22,32,7,7,30,21,7,8,3,7,0,13,14,0,7,32,26,23,27,33,21,33,17,7,11,27,28,8,21,33,35,35,23,29,14,30,29,7,24,23,20,13,41,18,2,18,41,11,32,8,19,27,3,35,41,14,16,5,19,30,41,14,41,33,25,1,18,16,18,0,17,39,28,28,17,18,25,17,0,16,17,16,16,17,17,17,21,16,17,16,15,15,10,10,0,11,12,16,32,9,37,16,41,35,27,7,15,16,25,31,20,29,20,12,10,2,23,3,41,14,17,9,23,9,23,14,25,37,1,34,25,6,28,20,20,1,38,14,20,25,32,12,30,32,7,16,41,0,34,9,14,41,0,14,0,1,17,33,32,2,24,35,38,12,12,6,5,26,0,0,17,41,33,7,20,12,8,23,41,35,33,40,11,27,38,0,15,36,23,22,12,25,41,21,2,28,27,35,6,40,18,17,27,32,35,41,1,19,7,37,20,35,20,6,31,16,33,1,6,12,41,16,27,32,19,41,31,36,41,24,21,38,34,12,32,6,18,37,11,0,35,34,34,18,32,15,9,35,25,22,31,34,19,8,32,1,15,31,30,31,32,8,4,9,2,38,38,34,9,40,0,10,13,2,28,37,20,3,13,8,31,0,32,41,25,41,27,7,27,16,33,32,7,0,14,2,20,28,41,7,40,10,14,11,10,38,2,14,26,41,7,30,5,2,40,31,0,11,18,35,6,29,18,1,18,27,23,6,9,24,41,39,8,22,28,38,7,35,38,16,22,28,21,12,4,40,26,19,5,21,31,10,21,38,11,10,9,26,40,40,33,7,33,20,26,1,8,8,26,11,0,38,17,25,31,7,40,22,35,17,41,41,41,41,41,41,33,41,2,23,26,23,41,3,34,26,22,23,20,33,31,16,41,1,4,11,34,41,2,2,17,33,32,27,14,27,15,8,1,21,20,38,8,13,38,39,25,38,22,31,5,26,29,29,0,39,1,2,6,25,4,18,36,10,39,4,6,8,19,14,2,17,32,23,27,27,9,32,0,36,33,6,12,1,41,17,21,41,36,5,32,36,36,36,34,4,22,30,30,15,0,3,18,11,20,37,9,2,30,10,22,35,9,29,7,40,34,27,33,11,21,19,1,38,30,0,40,36,36,27,40,24,36,3,25,13,37,5,29,0,31,32,6,31,41,12,1,20,19,16,7,31,18,9,20,35,33,31,6,40,39,40,9,0,29,21,1,32,10,38,1,16,12,27,41,18,36,38,22,16,19,1,34,22,12,8,37,26,22,31,35,38,3,29,34,32,17,14,20,17,0,7,41,20,12,35,1,39,27,3,0,4,7,6,32,9,14,11,22,38,0,33,19,37,31,33,3,5,26,25,10,14,20,40,31,30,39,4,4,37,5,10,32,20,18,6,35,25,25,40,10,16,38,27,7,4,23,2,29,39,17,8,22,14,25,41,10,8,0,7,8,8,8,4,31,22,9,12,33,4,40,19,31,8,0,33,31,4,33,3,13,8,11,9,19,24,18,33,21,22,22,4,26,26,38,21,3,35,41,1,22,36,4,0,1,13,34,20,35,4,20,12,6,11,23,16,24,3,27,26,41,26,23,32,7,15,22,13,28,14,2,14,14,24,4,39,0,31,1,10,0,31,20,4,39,25,32,39,18,3,3,41,2,30,38,12,37,27,28,25,25,32,30,16,41,9,26,36,33,29,40,19,19,34,0,12,11,8,2,2,18,12,27,27,9,3,38,27,12,35,32,36,34,14,30,22,12,41,38,35,23,12,27,18,30,24,35,9,41,0,29,41,23,12,18,0,41,40,34,40,28,23,0,4,1,33,6,16,31,13,15,41

6. Cost: 4.674

B.2.8. Problem Rye92

1. Exams: 482

2. Students: 11483
3. Enrollments: 45051
4. Timeslots: 23
5. Solution:
   0,0,1,0,22,1,1,19,6,6,15,10,0,8,15,10,7,1,12,0,6,14,21,16,1,0,8,19,4,12,13,1,14,7,3,5,0,22,10,21,0,11,22,15,0,19,4,21,16,4,10,19,0,8,13,22,4,4,10,13,0,6,15,22,11,11,2,14,21,14,22,0,15,10,0,22,7,15,4,11,19,19,16,0,22,19,5,12,22,22,20,1,21,0,4,4,22,15,4,10,0,7,17,22,13,1,22,16,8,8,8,16,22,8,11,18,10,4,8,0,3,11,8,7,19,5,5,16,1,10,1,22,9,4,21,8,4,1,16,6,1,22,22,3,17,16,14,16,19,12,3,22,6,19,7,6,18,5,14,18,8,16,3,12,13,14,7,0,1,4,18,17,17,1,21,17,11,0,13,22,3,12,22,22,16,4,19,22,11,5,16,0,22,0,0,0,11,5,14,14,11,22,14,12,15,10,19,17,16,22,13,7,22,21,9,12,14,18,0,8,21,18,0,15,14,12,20,12,11,17,14,9,16,20,2,0,22,3,12,22,0,3,13,1,15,20,10,22,19,0,2,19,10,16,0,19,11,4,15,22,13,1,17,7,22,4,1,6,16,7,10,18,14,22,22,18,22,0,4,6,11,9,21,5,11,21,3,17,12,14,8,6,15,10,4,12,3,6,9,16,14,1,0,22,1,8,8,17,0,0,0,22,4,4,15,17,18,4,17,3,14,21,0,4,4,7,1,15,2,16,22,0,22,0,12,22,22,5,10,13,20,20,13,4,22,7,13,13,16,3,11,8,17,0,22,2,4,15,10,0,11,10,0,4,4,1,22,18,12,13,21,10,21,19,22,6,6,17,0,10,16,22,22,6,0,7,22,22,0,14,6,16,16,16,17,10,0,15,4,19,7,22,4,19,14,19,8,4,0,0,17,0,22,1,5,2,5,5,8,9,10,20,0,15,6,6,8,7,10,6,8,19,8,4,22,0,19,10,19,13,15,16,21,22,7,22,19,0,8,0,22,13,12,0,22,6,22,6,0,16,1,0,12,7,22,0,17,0,11,18,8,4,7,6,18,7,18,22,0,22,16,11,1,0,22,2
6. Cost: 8.625

B.2.9. Problem Sta83

1. Exams: 139
2. Students: 611
3. Enrollments: 5751
4. Timeslots: 13
5. Solution:
   6,6,11,4,0,5,5,5,12,12,0,12,12,12,12,12,5,9,9,1,0,9,1,9,9,11,10,6,0,6,9,9,9,0,3,9,9,11,1,7,0,1,11,9,7,7,7,10,0,3,3,3,3,3,9,3,6,0,0,0,11,11,0,11,11,11,9,11,11,11,5,3,3,3,3,3,9,0,12,6,9,6,6,3,9,7,11,9,9,7,0,0,0,0,7,7,2,5,5,7,5,5,1,1,1,1,12,12,12,12,12,0,6,6,0,6,0,0,0,1,11,1,7,7,1,7,1,1,1,10,5,9,6,8,10,8,4,7,2

6. Cost: 157.033

B.2.10. Problem Tre92

1. Exams: 261
2. Students: 4360
3. Enrollments: 14901
4. Timeslots: 23
5. Solution:
   1,15,11,12,22,3,17,20,14,6,22,9,15,19,0,22,3,12,4,15,7,6,17,22,20,22,4,7,4,11,15,17,21,7,11,7,19,18,2,19,7,7,5,21,4,0,10,0,15,17,14,9,21,14,17,10,18,14,5,22,0,12,1,22,1,19,6,16,20,2,3,16,20,5,14,1,0,6,3,6,20,3,9,21,15,8,14,5,21,2,11,18,0,14,13,13,16,20,14,5,8,22,18,0,22,11,9,12,2,5,11,10,1,22,2,8,2,21,6,18,20,16,17,9,19,1,22,15,8,5,7,10,16,1,13,22,0,7,15,22,1,13,22,14,1,7,3,16,6,9,19,11,21,4,5,1,21,21,6,14,16,9,17,8,19,9,13,11,22,12,3,11,12,12,22,10,10,22,4,3,21,13,22,12,6,21,10,6,4,20,15,19,19,7,11,15,22,2,18,0,10,0,8,19,22,10,7,15,4,13,9,2,19,0,0,0,15,12,17,7,21,6,20,16,1,10,8,20,2,18,14,13,6,5,18,5,19,6,0,9,0,18,14,19,8,9,12,22,3,14,1,19,5,16,4,3,3,3,22,20,22
6. Cost: 8.330

B.2.11. Problem Uta92

1. Exams: 622
2. Students: 21266
3. Enrollments: 58979
4. Timeslots: 35
5. Solution:
   26,0,27,13,2,34,18,0,29,19,6,3,27,8,0,0,4,15,27,34,32,0,1,29,26,18,19,13,9,6,24,11,4,19,23,7,10,25,34,1,9,29,16,22,16,32,0,4,31,14,5,28,25,23,2,13,3,17,34,7,5,21,14,25,12,19,8,15,15,28,27,8,17,13,2,31,7,21,8,34,15,22,34,8,10,17,32,18,17,8,5,17,29,25,15,12,23,28,28,30,7,24,1,22,28,4,7,16,19,27,20,21,13,5,0,14,24,32,8,26,11,3,34,30,29,6,21,15,0,13,7,31,11,17,22,16,26,19,34,2,22,11,20,18,9,12,8,24,6,4,9,33,17,11,0,20,22,29,15,30,4,34,21,8,4,0,23,1,10,2,34,22,32,27,17,5,6,1,33,20,24,20,20,6,25,20,25,12,34,28,26,25,10,22,25,18,14,21,33,14,6,31,10,9,34,17,11,19,26,7,30,20,0,8,2,0,28,30,12,31,15,26,

0,27,20,14,13,26,7,34,12,34,7,0,34,12,34,32,14,15,32,32,4,9,24,0,12,32,32,5,0,14,20,11,18,18,28,28,9,15,0,20,32,9,25,9,25,20,32,30,6,0,33,22,11,1,34,28,17,14,22,6,4,8,14,33,34,3,31,16,0,12,18,0,25,21,26,24,16,29,5,29,11,28,1,16,30,14,27,3,30,11,15,16,15,33,21,0,7,18,17,14,12,34,32,24,0,21,33,9,2,30,12,0,25,17,34,34,5,14,16,28,0,28,2,20,32,11,29,34,30,34,34,34,2,1,33,33,33,33,6,7,20,0,24,28,20,12,5,26,14,26,18,23,33,6,23,0,3,25,29,18,24,16,1,0,10,1,31,10,20,18,24,28,12,33,16,22,6,3,32,25,4,0,30,15,20,12,27,26,10,24,34,0,25,2,30,27,24,16,6,34,5,19,12,30,26,20,1,17,34,12,30,9,21,14,27,4,12,33,12,33,3,31,28,2,6,25,11,25,13,11,34,13,15,0,28,8,23,32,16,19,21,17,17,22,28,16,22,31,27,14,34,17,0,34,28,6,2,12,23,8,5,10,26,20,1,2,3,5,7,9,34,2,32,8,23,17,11,26,2,30,13,17,30,27,33,14,22,22,25,34,23,23,4,29,33,14,26,27,21,10,0,20,34,6,17,7,24,0,12,22,21,23,21,1,20,24,13,12,28,34,7,2,7,14,11,5,27,13,22,6,17,19,8,34,21,34,22,28,28,13,34,13,29,2,16,30,6,5,0,17,10,3,3,23,21,27,33,34,18,25,7,34,4,13,13,8,4,4,22,34,16,23,30,10,29,9,22,18,19,1,2,34,26,18,8,13,30,2,15,11,7,29,5,33,13,0,0,32,8,17

6. Cost: 3.279

B.2.12. Problem Ute92

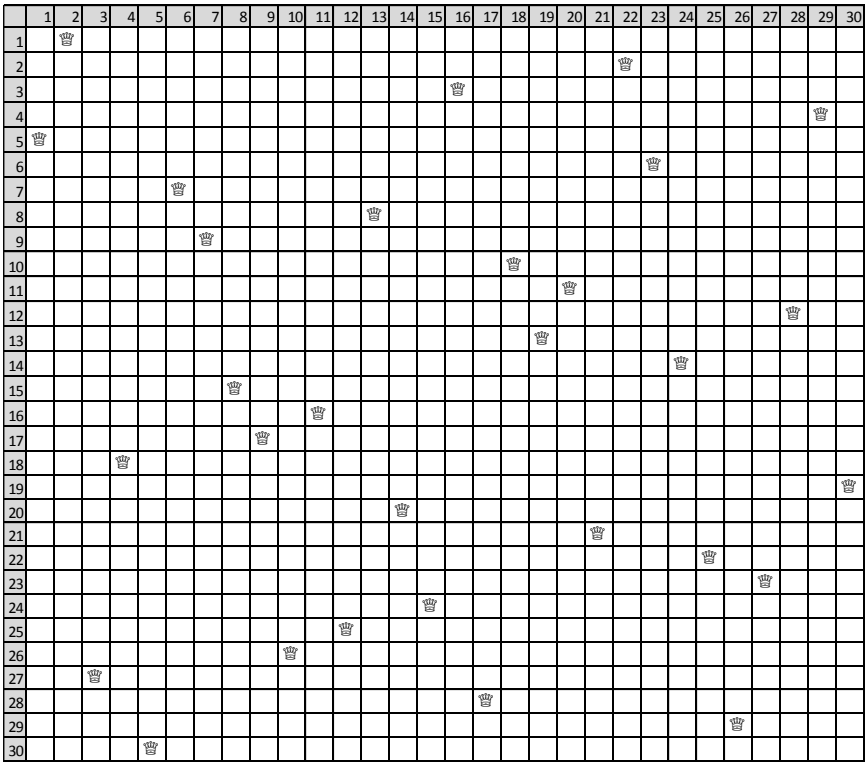
1. Exams: 184
2. Students: 2749 (incorrectly shows 2750 in the website for benchmark problems)
3. Enrollments: 11793
4. Timeslots: 10
5. Solution:
   9,3,6,0,3,9,6,6,7,4,4,4,5,5,5,5,9,0,8,2,6,7,9,2,2,4,3,0,2,7,6,5,0,0,7,4,9,3,2,9,0,7,9,3,7,9,9,3,0,7,9,5,0,2,7,5,6,8,3,1,9,0,6,8,0,0,0,2,2,6,0,9,4,9,9,4,3,0,6,9,8,7,9,3,4,7,3,2,7,1,9,4,9,6,5,0,0,7,8,5,5,5,5,5,5,9,9,5,5,9,3,7,0,9,3,0,5,8,7,2,0,9,6,8,5,5,8,9,2,9,7,3,9,7,1,2,0,0,2,4,2,0,1,9,9,2,8,0,0,7,4,7,6,0,9,9,3,3,0,6,6,9,4,1,1,4,9,0,3,9,9,9,2,6,5,3,7,0,3,5,1,4,5,3
6. Cost: 24.847

B.2.13. Problem Yor83

1. Exams: 181
2. Students: 941
3. Enrollments: 6034

4. Timeslots: 21
5. Solution:
   14,11,8,12,12,8,2,17,5,4,1,1,7,5,9,5,14,3,15,10,7,14,11,6,2,2,11,20,20,20,20,20,20,20,20,20,20,19,20,20,19,18,18,7,20,1,1,12,17,10,6,6,3,1,3,5,15,14,14,11,9,13,2,15,8,14,9,11,14,14,15,8,14,15,14,13,9,9,10,10,2,5,5,4,4,7,11,13,11,7,7,8,3,3,3,0,4,6,0,2,17,16,7,13,11,4,4,4,4,4,0,13,0,0,0,5,0,16,12,12,7,12,13,10,13,6,12,12,0,0,1,0,9,9,11,11,13,3,15,5,1,15,15,13,5,10,9,6,9,10,18,17,17,17,18,14,3,6,11,16,5,8,3,2,6,2,7,14,5,1,12,8,12,7,13,3,2,19,8,10
6. Cost: 36.235

B.3. A solution for 30X30 N-Queen Problems by ICHEA



References

Abdullah, S., Ahmadi, S., Burke, E.K. and Dror, M.,2006. *Investigating Ahuja–Orlin's large neighbourhood search approach for examination timetabling*. OR Spectrum. Vol. 29, 351–372.

Amirjanov, A.,2006. *The development of a changing range genetic algorithm*. Computer Methods in Applied Mechanics and Engineering. Vol. 195, 2495–2508.

Bandyopadhyay, S. and Pal, S.K.,2001. *Pixel classification using variable string genetic algorithms with chromosome differentiation*. IEEE Transactions on Geoscience and Remote Sensing. Vol. 39, 303–308.

Barták, R. and Salido, M.A.,2011. *Constraint satisfaction for planning and scheduling problems*. Constraints. Vol. 16, 223–227.

Bliek, C., Spellucci, P., Vicente, L.N. and Neumaier, A.,2001. *Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of the Art*, Report of the European Community funded project COCONUT. Mathematisches Institut der Universitat Wien.

Brailsford, S.,1999. *Constraint satisfaction problems: Algorithms and applications*. European Journal of Operational Research. Vol. 119, 557–581.

Branke, J. and Schmeck, H.,2003. *Designing Evolutionary Algorithms for Dynamic Optimization Problems*, in: Ghosh, D.A., Tsutsui, P.D.S. (Eds.), Advances in Evolutionary Computing, Natural Computing Series. Springer Berlin Heidelberg, pp. 239–262.

Brest, J., Zamuda, A., Boskovic, B., Maucec, M.S. and Zumer, V.,2009. *Dynamic optimization using Self-Adaptive Differential Evolution*, in: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 415–422.

Burke, E. and Bykov, Y.,2008. *A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems*, in: PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling.

Burke, E., Bykov, Y., Newall, J. and Petrovic, S.,2003. *A Time-Predefined Local Search Approach to Exam Timetabling Problems*. Iie Transactions. Vol. 1153, 76–90.

Burke, E., Dror, M., Petrovic, S. and Qu, R.,2005. *Hybrid Graph Heuristics within a Hyper-Heuristic Approach to Exam Timetabling Problems*, in: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Next Wave in Computing, Optimization, and Decision Technologies, Operations Research/Computer Science Interfaces Series. Springer US, pp. 79–91.

Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. and Qu, R.,to appear. *Hyper-heuristics: A survey of the state of the art*. Journal of the Operational Research Society.

Burke, E., Jackson, K., Kingston, J.H. and Weare, R.,1997. *Automated University Timetabling: The State of the Art*. The Computer Journal. Vol. 40, 565–571.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S.,2003. *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*, in: Glover, F.,

Kochenberger, G.A. (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science. Springer US, pp. 457–474.

Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S. and Qu, R.,2010. *Hybrid variable neighbourhood approaches to university exam timetabling*. European Journal of Operational Research. Vol. 206, 46–53.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J.R.,2010. *A Classification of Hyper-heuristic Approaches*, in: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science. Springer US, Vol. 146, pp. 449–468.

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R.,2007. *A graph-based hyper-heuristic for educational timetabling problems*. European Journal of Operational Research. Vol. 176, 177–192.

Burke, E.K., Newall, J.P. and Weare, R.F.,1996. *A memetic algorithm for university exam timetabling*, in: Burke, E., Ross, P. (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 241–250.

Burke, E.K., Qu, R. and Soghier, A.,2012. *Adaptive selection of heuristics for improving exam timetables*. Annals of Operations Research. 1–17.

Caramia, M., Dell'Olmo, P. and Italiano, G.F.,2001. *New Algorithms for Examination Timetabling*, in: Näher, S., Wagner, D. (Eds.), Algorithm Engineering, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 230–241.

Carter, M.W., Laporte, G. and Lee, S.Y.,1996. *Examination Timetabling: Algorithmic Strategies and Applications*. The Journal of the Operational Research Society. Vol. 47, 373–383.

Casey, S. and Thompson, J.,2003. *GRASPing the Examination Scheduling Problem*, in: Burke, E., Causmaecker, P.D. (Eds.), Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 232–244.

Coello, C.A.C.,1998. *A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques*. Knowledge and Information Systems. Vol. 1, 269–308.

Coello Coello, C.A.,2002. *Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art*. Computer Methods in Applied Mechanics and Engineering. Vol. 191, 1245–1287.

Coello Coello, C.A.,2012. *Constraint-handling techniques used with evolutionary algorithms*, in: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion, GECCO Companion '12. ACM, New York, NY, USA, pp. 849–872.

Courant, R.,1943. *Variational methods for the solution of problems of equilibrium and vibrations*. Bulletin of the American Mathematical Society. 1–23.

Craenen, B.G.W.,2005. *Solving constraint satisfaction problems with evolutionary algorithms (Phd Dissertation)*. Vrije Universiteit, Amsterdam.

Craenen, B.G.W., Eiben, A.E. and Marchiori, E.,2000. *Solving constraint satisfaction problems with heuristic-based evolutionary algorithms*, in: Proceedings of the 2000 Congress on Evolutionary Computation, 2000. IEEE, Vol. 2, pp. 1571–1577.

Craenen, B.G.W., Eiben, A.E. and van Hemert, J.I.,2003. *Comparing evolutionary algorithms on binary constraint satisfaction problems*. IEEE Transactions on Evolutionary Computation. Vol. 7, 424–444.

Culberson, J.C.,1998. *On the futility of blind search: An algorithmic view of "no free lunch"*. IEEE Transactions on Evolutionary Computation. Vol. 6, 109–127.

De Castro, L.N. and Von Zuben, F.J.,2002. *Learning and optimization using the clonal selection principle*. Evolutionary Computation, IEEE Transactions on. Vol. 6, 239–251.

De Weck, O. and Kim, I.Y.,2004. *Variable Chromosome Length Genetic Algorithm for Structural Topology Design Optimization*. Strain. Vol. AIAA-2004-1911, 1–12.

Deb, K.,*Kanpur Genetic Algorithms Laboratory [WWW Document]*. URL http://www.iitk.ac.in/kangal/codes.shtml [Accessed 29 Sep. 2011].

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.,2002. *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation. Vol. 6, 182–197.

Dechter, R.,1992. *Constraint networks*, in: Encyclopedia of Artificial Intelligence, In Shapiro, S., Ed. John Wiley & Sons, Ltd, New York, USA, pp. 276–285.

Demeester, P., Bilgin, B., Causmaecker, P.D. and Berghe, G.V.,2012. *A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice*. Journal of Scheduling. Vol. 15, 83–103.

Dong, N., Wei, F. and Wang, Y.,2012. *Preference Based Multiobjective Evolutionary Algorithm for Constrained Optimization Problems*, in: 2012 Eighth International Conference on Computational Intelligence and Security (CIS). pp. 65–70.

Eastridge, R. and Schmidt, C.,2008. *Solving n-queens with a genetic algorithm and its usefulness in a computational intelligence course*. J. Comput. Sci. Coll. Vol. 23, 223–230.

Eberhart, R. and Kennedy, J.,1995. *A new optimizer using particle swarm theory*, in: Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on. pp. 39 –43.

Eiben, A.E.,2001a. *Evolutionary algorithms and constraint satisfaction: definitions, survey, methodology, and research directions*. Springer-Verlag, pp. 13–30.

Eiben, A.E.,2001b. *Evolutionary Algorithms and Constraint Satisfaction: Definitions, Survey, Methodology, and Research Directions*, in: Theoretical Aspects of Evolutionary Computing. Springer-Verlag, pp. 13–58.

Eiben, A.E., Hemert, J.I. van, Marchiori, E. and Steenbeek, A.G.,1998. *Solving Binary Constraint Satisfaction Problems Using Evolutionary Algorithms with an Adaptive Fitness Function*, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, PPSN V. Springer-Verlag, London, UK, UK, pp. 201–210.

Eiben, A.E. and Smith, J.E.,2003. *Introduction to Evolutionary Computation, 1st ed*. Springer-Verlag.

El Rhalibi, A. and Kelleher, G.,2003. *An approach to dynamic vehicle routing, rescheduling and disruption metrics*, in: IEEE International Conference on Systems, Man and Cybernetics, 2003. IEEE, Vol. 4, pp. 3613–3618.

Eley, M.,2007. *Ant algorithms for the exam timetabling problem*, in: Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling VI, PATAT'06. Springer-Verlag, Berlin, Heidelberg, pp. 364–382.

Fonseca, C.M. and Fleming, P.J.,1993. *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization*, in: Proceedings of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 416–423.

Ghedira, K.,1994. *Distributed simulated re-annealing for dynamic constraint satisfaction problems*, in: Sixth International Conference on Tools with Artificial Intelligence. IEEE, pp. 601–607.

Glover, F.,1990. *Tabu Search—Part II*. ORSA Journal on Computing. Vol. 2, 4–32.

Glover, F. and Kochenberger, G.A.,1996. *Critical Event Tabu Search for Multidimensional Knapsack Problems*, in: Osman, I.H., Kelly, J.P. (Eds.), Meta-Heuristics. Springer US, pp. 407–427.

Goldberg, D.,2002. *The Design of Innovation*. Kluwer Academic Publishers, New York.

Goldberg, D.E.,1989. *Genetic algorithms in search, optimization, and machine learning, 1st ed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hu, X., Eberhart, R.C. and Shi, Y.,2003. *Swarm intelligence for permutation optimization: a case study of n-queens problem*, in: Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE. pp. 243 – 246.

Karimi, J., Nobahari, H. and Pourtakdoust, S.H.,2012. *A new hybrid approach for dynamic continuous optimization problems*. Applied Soft Computing. Vol. 12, 1158–1167.

Koziel, S. and Michalewicz, Z.,1998. *A Decoder-Based Evolutionary Algorithm for Constrained Parameter Optimization Problems*, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, PPSN V. Springer-Verlag, London, UK, pp. 231–240.

Kramer, O.,2010. *A Review of Constraint-Handling Techniques for Evolution Strategies*. Applied Computational Intelligence and Soft Computing. Vol. 2010, 1–11.

Krasnogor, N. and Smith, J.,2005. *A tutorial for competent memetic algorithms: model, taxonomy, and design issues*. IEEE Transactions on Evolutionary Computation. Vol. 9, 474–488.

Lehman, J. and Stanley, K.,2008. *Exploiting Open-Endedness to Solve Problems Through the Search for Novelty*, in: Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI). MIT Press.

Letavec and Ruggiero,2002. *The Queens Problem - Delta*. INFORMS Transactions on Education. Vol. 2, 101–103.

Li, C., Yang, S., Nguyen, T.T., Yu, E.L., Yao, X., Jin, Y., Beyer, H.G. and Suganthan, P.N.,2008. *Benchmark Generator for CEC'2009 Competition on Dynamic Optimization*. IEEE Congress on Evolutionary Computation.

Li, T., Xiao, Y. and Wang, H.,2007. *Dynamic Constraint Satisfaction Approach to Hybrid Flowshop Rescheduling*, in: 2007 IEEE International Conference on Automation and Logistics. Jinan, China, pp. 818–823.

Liang, J.J., Runarsson, T.P., Mezura-Montes, E., Clerc, M., Suganthan, P.N., Coello, C.A.C. and Deb, K.,2006. *Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-parameter Optimization (Technical Report)*. Nanyang Technological University, Singapore.

Liu, H., Cai, Z. and Wang, Y.,2010. *Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization*. Applied Soft Computing. Vol. 10, 629–640.

Maciej Norberciak,2006. *Universal Method for Solving Timetabling Problems Based on Evolutionary Approach*, in: Proceedings of the International Multiconference on Computer Science and Information Technology. pp. 149–157.

Martinjak, I. and Golub, M.,2007. *Comparison of Heuristic Algorithms for the N-Queen Problem*, in: Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on. pp. 759 –764.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R. and Burke, E.K.,2010. *Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition*. INFORMS J. on Computing. Vol. 22, 120–130.

Merlot, L.T.G., Boland, N., Hughes, B.D. and Stuckey, P.J.,2003. *A Hybrid Algorithm for the Examination Timetabling Problem*, in: Burke, E., Causmaecker, P.D. (Eds.), Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 207–231.

Mezura-montes, E. and Coello, C.A.C.,2006. *A Survey of Constraint-Handling Techniques Based on Evolutionary Multiobjective Optimization (No. EVOCINV-04-2006)*. Departamento de Computación, Evolutionary Computation Group at CINVESTAV.

Mezura-Montes, E. and Coello, C.A.C.,2005. *A simple multimembered evolution strategy to solve constrained optimization problems*. IEEE Transactions on Evolutionary Computation. Vol. 9, 1 – 17.

Michalewicz, Z. and Fogel, D.B.,2004a. *Constraint-Handling Techniques*, in: How to Solve It: Modern Heuristics. Springer, pp. 231–270.

Michalewicz, Z. and Fogel, D.B.,2004b. *How to Solve It: Modern Heuristics, 2nd ed*. Springer.

Michalewicz, Z. and Janikow, C.Z.,1991. *Handling Constraints in Genetic Algorithms*, in: ICGA. San Diego, CA, USA, pp. 151–157.

Michalewicz, Z. and Nazhiyath, G.,1995. *Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints*, in: IEEE International Conference on Evolutionary Computation. Vol. 2, pp. 647–651.

Michalewicz, Z. and Schoenauer, M.,1996. *Evolutionary algorithms for constrained parameter optimization problems*. Evolutionary Computation. Vol. 4, 1–32.

Müller, T.,2005. *Constraint-based Timetabling (PhD Dissertation)*. Charles University, Prague.

Nguyen, T. and Yao, X.,2012. *Continuous Dynamic Constrained Optimisation - The Challenges*. IEEE Transactions on Evolutionary Computation. Vol. 16, 769–786.

Nguyen, T.T. and Yao, X.,2009. *Benchmarking and solving dynamic constrained problems*, in: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 690 –697.

Ólafsson, S.,2006. *Chapter 21 Metaheuristics*, in: Shane G. Henderson and Barry L. Nelson (Ed.), Handbooks in Operations Research and Management Science. Elsevier, Vol. 13, pp. 633–654.

Onwubolu, G.C.,2002. *The dilemma of hill climbing*, in: Emerging Optimization Techniques in Production Planning and Control. Imperial College Press, London, UK.

Onwubolu, G.C. and Sharma, A.,2004. *Particle Swarm Optimization for the assignment of facilities to locations*, in: New Optimization Techniques in Engineering. Springer-Verlag.

Paredis, J.,1994. *Co-evolutionary Constraint Satisfaction*, in: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, PPSN III. Springer-Verlag, pp. 46–55.

Pemberton, J.C. and Galiber, I.I.I.,2001. *A constraint-based approach to satellite scheduling*, in: DIMACS Workshop on on Constraint Programming and Large Scale Discrete Optimization. American Mathematical Society, Rutgers Univ., Piscataway, New Jersey, United States, pp. 101–114.

Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G. and Lee, S.Y.,2008. *A survey of search methodologies and automated system development for examination timetabling*. Journal of Scheduling. Vol. 12, 55–89.

Quan, G., Greenwood, G.W., Liu, D. and Hu, S.,2007. *Searching for multiobjective preventive maintenance schedules: Combining preferences with evolutionary algorithms*. European Journal of Operational Research. Vol. 177, 1969–1984.

Rahnamayan, S. and Dieras, P.,2008. *Efficiency competition on N-queen problem: DE vs. CMA-ES*, in: Canadian Conference on Electrical and Computer Engineering, CCECE 2008. IEEE Xplore, Niagara Falls, US, pp. 000033–000036.

Ricardo Landa Becerra and Carlos A. Coello Coello,2006. *Cultured differential evolution for constrained optimization*. Computer Methods in Applied Mechanics and Engineering. Vol. 195, 4303–4322.

Richter, H.,2010. *Memory Design for Constrained Dynamic Optimization Problems*, in: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A., Goh, C.-K., Merelo, J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G. (Eds.), Applications of Evolutionary Computation, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Vol. 6024, pp. 552–561.

Rossi, F., Petrie, C. and Dhar, V.,1990. *On the Equivalence of Constraint Satisfaction Problems*. in proceedings of the 9th European conference on Artificial Intelligence. 550–556.

Russell, S.J. and Norvig, P.,2003. *Artificial Intelligence - a modern approach, 2nd ed*. Pearson Education Inc, New Jersey.

Sabin, M., Freuder, E.C. and Wallace, R.J.,2003. *Greater Efficiency for Conditional Constraint Satisfaction.*, in: CP'03. pp. 649–663.

Salcedo-Sanz, S.,2009. *A survey of repair methods used as constraint handling techniques in evolutionary algorithms*. Computer Science Review. Vol. 3, 175–192.

Schaffer, J.D.,1985. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*, in: Proceedings of the 1st International Conference on Genetic Algorithms. L. Erlbaum Associates Inc., pp. 93–100.

Schiex, T. and Verfaillie, G.,1993. *Nogood Recording for static and dynamic constraint satisfaction problems*, in: , Fifth International Conference on Tools with Artificial Intelligence, 1993. TAI ’93. Proceedings. IEEE, pp. 48–55.

Schoenauer, M. and Michalewicz, Z.,1996. *Evolutionary Computation at the Edge of Feasibility*, in: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, PPSN IV. Springer-Verlag, London, UK, pp. 245–254.

Shang, Y. and Fromherz, M.P.J.,2003. *Experimental complexity analysis of continuous constraint satisfaction problems*. Information Sciences. Vol. 153, 1–36.

Sharma, A.,2010. *A new optimizing algorithm using reincarnation concept*, in: 2010 11th IEEE International Symposium on Computational Intelligence and Informatics (CINTI). Budapest, Hungary, pp. 281–288.

Sharma, A. and Omlin, C.W.,2009. *Performance Comparison of Particle Swarm Optimization with Traditional Clustering Algorithms used in Self-Organizing Map.* International Journal of Computational Intelligence. Vol. 5, 1–12.

Sharma, A. and Sharma, D.,2011. *Clonal Selection Algorithm for Classification*, in: Liò, P., Nicosia, G., Stibor, T. (Eds.), Artificial Immune Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, Vol. 6825, pp. 361–370.

Sharma, A. and Sharma, D.,2012a. *ICHEA – A Constraint Guided Search for Improving Evolutionary Algorithms*, in: In: Huang, T., Zeng, Z., Li, C., Leung, C.S. (eds.) ICONIP 2012, Part I. LNCS. Springer, Heidelberg, Doha, Qatar, Vol. 7663, pp. 269–279.

Sharma, A. and Sharma, D.,2012b. *ICHEA for Discrete Constraint Satisfaction Problems*, in: Thielscher, M., Zhang, D. (Eds.), AI 2012: Advances in Artificial Intelligence, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 242–253.

Sharma, A. and Sharma, D.,2012c. *Real-Valued Constraint Optimization with ICHEA*, in: Huang, T., Zeng, Z., Li, C., Leung, C. (Eds.), Neural Information Processing, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Vol. 7665, pp. 406–416.

Sharma, A. and Sharma, D.,2012d. *An Incremental Approach to Solving Dynamic Constraint Satisfaction Problems*, in: Huang, T., Zeng, Z., Li, C., Leung, C. (Eds.), Neural Information Processing, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Vol. 7665, pp. 445–455.

Sharma, A. and Sharma, D.,2012e. *Solving Dynamic Constraint Optimization Problems Using ICHEA*, in: Huang, T., Zeng, Z., Li, C., Leung, C. (Eds.), Neural Information Processing, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Vol. 7665, pp. 434–444.

Sharma, A. and Sharma, D.,2013. *Constraint Optimization for Timetabling Problems using a Constraint Driven Solution Model*, in: 26th Australasian Joint Conference on Artificial Intelligence. Springer Berlin / Heidelberg, Dunedin, New Zealand.

Shcherbina, O.,*The COCONUT Benchmark [WWW Document]*. A benchmark for global optimization and constraint satisfaction. URL http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html [Accessed 22 Sep. 2011].

Srinivas, N. and Deb, K.,1994. *Muiltiobjective optimization using nondominated sorting in genetic algorithms*. Evolutionary Computation. Vol. 2, 221–248.

Tessema, B. and Yen, G.G.,2006. *A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization*, in: IEEE Congress on Evolutionary Computation, 2006. CEC 2006. IEEE, pp. 246–253.

Tsang, E.,1993. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.

Tuga, M., Berretta, R. and Mendes, A.,2007. *A Hybrid Simulated Annealing with Kempe Chain Neighborhood for the University Timetabling Problem*. IEEE, pp. 400–405.

Van Veldhuizen, D.A. and Lamont, G.B.,2000. *Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art*. Evolutionary Computation. Vol. 8, 125–147.

Verfaillie, G. and Jussien, N.,2005. *Constraint Solving in Uncertain and Dynamic Environments: A Survey.* Constraints. 253–281.

Verfaillie, G. and Schiex, T.,1994. *Solution reuse in dynamic constraint satisfaction problems*, in: Proceedings of the Twelfth National Conference on Artificial Intelligence (vol. 1), AAAI '94. American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 307–312.

Vivekanandan, P., Rajalakshmi, M. and Nedunchezhian, R.,2013. *An Intelligent Genetic Algorithm for Mining Classification Rules in Large Datasets*. Computing and Informatics. Vol. 32, 1–22.

Wallace, R.J., Grimes, D. and Freuder, E.C.,2009. *Solving Dynamic Constraint Satisfaction Problems by Identifying Stable Features.*, in: IJCAI'09. pp. 621–627.

Wang, H., Wang, D. and Yang, S.,2007. *Triggered Memory-Based Swarm Optimization in Dynamic Environments*, in: Applications of Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg, pp. 637–646.

Wang, Y.,*Yong Wang CV [WWW Document]*. URL http://deptauto.csu.edu.cn/staffmember/YongWang.htm [Accessed 28 Mar. 2012].

Wolpert, D.H. and Macready, W.G.,1997. *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation. Vol. 1, 67–82.

Xing, L., Chen, Y. and Cai, H.,2006. *An intelligent genetic algorithm designed for global optimization of multi-minima functions*. Applied Mathematics and Computation. Vol. 178, 355–371.

Yang, Y. and Petrovic, S.,2005. *A Novel Similarity Measure for Heuristic Selection in Examination Timetabling*, in: Burke, E., Trick, M. (Eds.), Practice and Theory of Automated Timetabling V. Springer Berlin Heidelberg, Berlin, Heidelberg, Vol. 3616, pp. 247–269.

CTVR: Home [WWW Document]. URL http://4c.ucc.ie/ctvr/ [Accessed 14 Aug. 2011].

List of References on Constraint-Handling Techniques used with Evolutionary Algorithms [WWW Document]. URL http://www.cs.cinvestav.mx/~constraint/ [Accessed 11 Mar. 2013].

Benchmark Exam Timetabling Datasets [WWW Document]. URL http://www.cs.nott.ac.uk/~rxq/data.htm [Accessed 7 Jun. 2013].