Multiobjective Optimization of Building Design Using Artificial Neural Network and Multiobjective Evolutionary Algorithms

Laurent Magnier

A Thesis

in

The Department

of

Building, Civil and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Building Engineering) at

Concordia University

Montréal, Québec, Canada

February, 2008

Your file Votre référence
ISBN: 978-0-494-63229-1
Our file Notre référence
ISBN: 978-0-494-63229-1

ABSTRACT

Multiobjective Optimization of Building Design Using Artificial Neural Network and Multiobjective Evolutionary Algorithms

Laurent Magnier

Building design is a very complex task, involving many parameters and conflicting objectives. In order to maximise the comfort and minimize the environmental impact, multiobjective optimization should be used. While some tools such as Genetic Algorithms (GA) exist, they are very seldom used in the industry, due to the large computational time they require.

This thesis focuses on a specific approach called GAINN (Genetic Algorithm Integrating Neural Network), which combines the rapidity of evaluation of Artificial Neural Networks (ANN) with the optimization power of GAs. The thesis concentrates on a better handling of multiple objectives, in order to efficiently exploit the methodology and increase its accessibility for the non-expert. First, a Multiobjective Evolutionary Algorithm (MOEA), NSGA-II, has been selected and programmed in MATLAB. Then, two new MOEAs were developed, specifically designed to take advantage of GAINN fast evaluations. These two MOEAs have proven to be more efficient than NSGA-II on benchmark test functions, for a comparison based on a maximum runtime.

In a second part of this thesis, developed MOEAs were used inside GAINN methodology to optimize the energy consumption and the thermal comfort in a residential

building. This optimization was successful, and enabled significant improvements in terms of energy consumption and thermal comfort. It also enabled to illustrate very clearly the relation between these two objectives. This optimization however highlighted two limitations regarding the ANN, the number of training cases and the accuracy in the vicinity of optimal solutions. Finally, the developed algorithms were applied on a past optimization study, in order to highlight the improvements added to GAINN methodology by the use of MOEA.

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Laurent Magnier

Entitled: Multiobjective optimization of building design using Artificial Neural Network and Multiobjective Evolutionary Algorithms

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Building Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair

_____ Examiner

_____ Examiner

_____ Supervisor

Approved by _____

Chair of Department or Graduate Program Director

_____ 20___ _____

Dean of Faculty

ACKNOWLEDGEMENTS

The author first wants to acknowledge the constant support of his supervisor, Dr. Haghighat. Thank you for your guidance, for the research atmosphere you created, and for the autonomy you gave me, while providing essential advices in very crucial points of my thesis. You did not only make me do research, you taught me what research is, and you made me *become* a researcher. All the research skills I have now come from your guidance; and without you, this thesis could have never been accomplished.

I also want to thank several persons from École Nationale des Travaux Publics de l'État, who helped me make this Master feasible. Thanks to Dr. Michel and Dr Guarracino for their scientific advices, and thanks to Mr Royer for his constant and generous help in solving administrative problems.

I want to acknowledge as well Liang Zhou and Jérôme Conraud for their valuable help in my research. I also want to acknowledge the support of NRC-IRC-Twin House project research group for the data and information provided. Special thanks to Marianne Manning whose advices made the building simulation feasible.

Many thanks go to my colleagues and friends in Dr Haghighat's team, and to Alexandre Hugo and to Sina Mesghali, who made my Master studies a great learning in so many more aspects than only science. Thanks for your support and for the atmosphere you created that made my studies a real pleasure. I also want to thank my two families, the one in France and the one here, for their constant support and help.

Finalement, merci à Gabrielle Bergeron-Brûlé, sans qui je ne serais pas là aujourd'hui.

TABLE OF CONTENTS

LISTE OF FIGURES

LISTE OF TABLES

ACRONYMS

ANN	Artificial Neural Network
CS	Conraud's solution
CFM	Cubic Feet by Minute
DTLZ	Series of MOEA test functions (initials of their creators Deb, Thiele, Laumanns, and Zitzler)
IPS	Ideal Pareto Set
FS1 (FS2)	Family Sorting 1 (respectively 2)
GA	Genetic Algorithm
GAINN	Genetic Algorithm Integrating Neural Network
GHG	Green House Gas
LEED	Leadership in Energy and Environmental Design
LHS	Latin Hypercube Sampling
MOEA	Multiobjective Evolutionary Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm
NSGAINN	Non-dominated Sorting Genetic Algorithm for Integrated Neural Network
PLAGUE	Poly-objective Looped Algorithm using Genetics and an Uncompleted Extinction

PMV	Predicted Mean Vote
PPD	Percent People Dissatisfied
PSS	PLAGUE's Solution Set
RSA	Response Surface Approximation
SSE	Sum of Squared Errors
SSW	Sum of Squared Weights
TRNSYS	Transient Energy System Simulation Tool
ZDT	Series of MOEA test functions (initials of their creators Deb, Thiele, and Zitzler)

NOMENCLATURE

CSP	Cooling set point	*(°C)*
E_{fan}	Fan energy consumption	*(kWh)*
E_{cool}	Cooling and dehumidifying energy consumption	*(kWh)*
E_{heat}	Heating (furnace) energy consumption	*(kWh)*
HSP	Heating set point	*(°C)*
FDMID	Thermostat delay before end of occupancy in middle season	*(minutes)*
FDS	Thermostat delay before end of occupancy in summer	*(minutes)*
FDW	Thermostat delay before end of occupancy in winter	*(minutes)*
N_{Dis}	Annual cumulative time with $\|PMV\|_{avg} > 0.5$	*(hours)*
$\|PMV\|_{avg}$	Annual average of the absolute PMV in the house	
RHMID	Relative humidity set point in middle season	
RHS	Relative humidity set point in summer	
RHW	Relative humidity set point in winter	
SDMID	Thermostat delay before occupancy in middle season	*(minutes)*
SDS	Thermostat delay before occupancy in summer	*(minutes)*
SDW	Thermostat delay before occupancy in winter	*(minutes)*
TCK	Thickness of concrete in interior floors	*(centimeters)*
VRR	Ventilation rate in recirculation mode	*(m^3/s)*
VRC	Ventilation rate in cooling mode	*(m^3/s)*
VRH	Ventilation rate in heating mode	*(m^3/s)*

Y	Metric used to study the convergence of a solution set	
WF1N	First floor north window size	(m^2)
WF1S	First floor south window size	(m^2)
WF2N	Second floor north window size	(m^2)
WF2S2	Second floor south west window size	(m^2)
WF2S1	Second floor south east window size	(m^2)
Δ	Metric used to study the spreading of a solution set	

CHAPTER 1

INTRODUCTION

1.1 Background

Global warming is likely to become the most important phenomenon of the 21st century, from an environmental as well as economical and social point of view. Most studies agree on an average increase of temperature of several Celsius degrees, on flux of millions of people, on rise of food prices, and on increase of numbers and frequencies of environmental disasters such as storms or floods. According to the great majority of experts, this global warming is caused by green house gases (GHG) such as CO_2, whose emissions are caused to a great extent by human activity (IPCC, 2007).

In Canada, buildings use 30% of the total energy consumption and are responsible for as much as 20% of GHG emissions (NRCan, 2005). The building industry has therefore a significant impact on global warming and is a priority in reducing the overall energy consumption. Accordingly, new constructions practices are rising to handle the energy consumption problems and many energy ratings (the most important being LEED (2007)) have been developed to assess the environmental impact of a building. Some governmental programs and regulations are also created to incite building owners to reduce their environmental print. The effect of these strategies and initiatives is however too little to yet have a significant impact on GHG emission.

One salient aspect of the fight against global warming is the relative inertia of the population, based on the fact that people are generally willing to reduce their environmental impact but still want to maintain their habits and comfort. Building science is primarily concerned about this aspect since no one would live in a very green but very uncomfortable house. Looking in more details at the whole concept of sustainable building, we realize that a building which pretends to be sustainable has to be *at the same time* environmental-friendly, comfortable, and affordable. These three aspects are essential and none of them can be ignored without compromising the concept of sustainability. The combination of these three very conflicting objectives however makes building design a highly complex task. A technology, strategy or design concept must be studied as a multiobjective problem. On the other hand, it also has to be extremely efficient in order to comply with objectives and policies, and justify the time spent on a project.

1.2 Problem Statement

Modern building design is a complex task, involving many different approaches, parameters, and conflictive objectives. This complexity is furthermore combined with a growing demand from both users and standards to have low energy consumptions, good thermal comfort, indoor air quality, visual comfort, etc. Thermal comfort, particularly, is a priority in residential buildings and many studies have proved that it can have a significant impact on productivity in working areas (Fanger, 2000). On the regulations side, standards are more and more detailed in terms of energy consumption or

construction requirements, and some of them involve the use of new technologies such as photovoltaic panels or solar water heater (RT2005 in France for instance (2005)). Finally, the large amount of money and time spent for new buildings gender very high expectations regarding the final building quality.

Despite all this, most designers continue to use traditional design techniques for building design. Practices are generally based on rules of thumb, on simulations of a limited number of cases, or sometimes on variable-by-variable parametric runs. Although widely used, these methods have many drawbacks. First, a rather limited range of possibilities is covered, studied, and finally proposed to the decision maker. The design is also limited by engineer assumptions, which is furthermore harmful since most rules of thumb come from times when the environmental impact was not an issue, and when green building science was not fully developed. More importantly, the designs coming from that kind of approach are very unlikely to be optimal. The design process is strongly limited by the small number of cases simulated (if any), which cannot handle the complexity of interactions between parameters and objectives of modern constructions. The resulting design may be relatively good, but has almost no chance to be the best possible one, and therefore has little chance to have a significant impact on GHG reductions.

Indeed, the use of a real optimization tool is the only way to ensure optimal designs. Tools such as Genetic Algorithms have proven their efficiency in many academic studies, but are almost never used in the industry. This is further regrettable since, when simulations are used, the relatively long time spent to create the simulation model is not fully exploited. The hours and sometimes days dedicated to create a decent model are

wasted by the lack of a true optimization. Meanwhile, some optimization tools already exist, and are sometimes readily available for building simulation (GenOpt for instance (Wetter, 2001)).

Many reasons may explain why optimization techniques are so rarely used in the building industry. The most likely reason is the very long computational time required by most optimization tools. Genetic Algorithms for instance require thousands of evaluations to reach optimal solutions. Simulation softwares, in turn, can provide accurate information regarding the building behaviour, but require a significant time to run (up to several hours). Therefore, a direct combination of GAs and time-expensive simulation tools makes optimization a very time-consuming and unattractive process. This issue is an extremely limiting aspect of building optimization, which needs to be overcome before someone could expect to see optimization tools used regularly in the building industry.

A last problem encountered in building design is how to handle multiple conflictive objectives. As described previously, a building study cannot be limited to one single issue but needs to take into account many of them, the most common ones being the cost, the energy consumption, and the indoor environment. The management of these different issues is generally undertaken by engineers at the design stage, most of the time taking all objectives except one as a constraint. Once again, this approach is not likely to find optimal trade-offs, and cannot even be called multiobjective handling. Even in rare cases where optimization algorithms are used, this problem remains, because the most commonly used method to handle different objective is to aggregate them in a weighted-sum and optimize it as a single objective. Once again, this cannot be called multiobjective

optimization, and this method suffers from assuming the importance of each objective *a priori*. The weights used in the weighted-sum, set at the beginning of the study by the engineer, give no final choice to building owner but rather assume what he *would* think would be important, and propose him one single solution, argued to be optimal.

From this discussion, we can draw the following. First, building design is more and more complex and plays a significant role in GHG emissions. In order to decrease the environmental impact of buildings, while maintaining a good indoor environment, multiobjective optimization tools should be used. Tools such as genetic algorithms are available and are extremely efficient, but suffer from the high number of evaluations/simulations required, which leads to very unattractive time costs in building application. There is therefore a need for a multiobjective, rapid, efficient, and accessible optimization tool, so optimization can start being used more widely in the building industry. This would enable better designs, better handling of multiple objectives and constraints, and hopefully energy reductions at a bigger scale than what currently happens.

1.3 Research Objectives

The objective of this thesis is to improve the current optimization methodologies, in order to make the optimization process more applicable for building design. The work will be based on a methodology which combines artificial neural networks and genetic algorithms to enable fast and accurate optimization. This methodology will be improved by a better handling of multiple objectives, by the use of multiobjective genetic algorithms. The developed methodology will then be tested on two case-studies and results will be discussed. In details, this thesis will be based on the following steps:

- Study the optimization techniques currently available through a comprehensive literature review, pointing out their respective assets and limitations. *(Chapter 2)*
- Focus on GAINN methodology, describe it, and study its assets and its current limitations. Based on this study, explain why multiobjective optimization should be added through the present thesis. *(Chapter 2)*
- Review multiobjective optimization techniques, analyse them, and find which algorithm is the best one. Program this algorithm in MATLAB, and test it to verify its reliability compared to the original algorithm. *(Chapter 3)*
- Create two multiobjective genetic algorithms specifically designed to take advantage of GAINN fast evaluations, program them, and compare them with base algorithm. *(Chapter 4)*

- Apply the whole methodology and developed algorithms to a first case study, for the optimization of thermal comfort and energy consumption in a residential building. *(Chapter 5)*
- Apply the optimization algorithms to a second case-study, in order to study the improvement added to GAINN methodology by the use of multiobjective optimization algorithms, comparing the results to the classical weighted-sum technique. *(Chapter 6)*
- Discuss the limitations of the current work, and propose future work to overcome them and/or improve the methodology. *(Chapter 7)*

CHAPTER 2

LITERATURE REVIEW

This chapter presents a comprehensive review of the optimization techniques currently available. The chapter is divided into two parts. The first part details and discusses the optimization concepts and available major algorithms. The second part focuses on a specific approach called GAINN (Genetic Algorithm Integrating Neural Network), studies its current limitations, and justifies the development chosen for this thesis.

2.1 Literature Review on Optimization

2.1.1 Foreword

From a mathematical standpoint, optimization is the process of maximizing (or minimizing) a function *f(X)*, possibly subject to several constraints, for a given number and ranges of variables $x_n \epsilon X$ (Deb, 2001). In more practical terms, optimization refers to finding the best possible configuration for a given problem. In building design, it may be for instance finding the design that meets the regulations requirements and budget, while offering the lowest energy consumption and providing the highest thermal comfort.

The first use of a real optimization tool in building engineering was investigated in 1968 by G. Neil Harper (Deb, 2001). Since then, optimization has gained interest and is now

frequently used, mostly in academic studies. One should however remain careful not to confuse the term *optimization*, with a simple *improvement*. For instance, a sensitivity analysis cannot be considered as an optimization. It may be able to find a *better* solution regarding the objective studied but there is no guarantee and indeed little chance to find the *best* solution. Indeed, the whole concept of *best* solution is not as straightforward as it looks. As long as only one objective is involved, optimization can be defined as a simple maximization or minimization. Dealing with multiple objectives at the same time makes however the optimization definition much more complex, as will be described in the next sections of this thesis.

A wide variety of optimization algorithms have been created and studied throughout the last centuries. The first optimization techniques, like the Gauss steepest descent developed in the 18^{th} century, were based on pure mathematics. More complex techniques have been later developed, and the first modern technique referred as optimization, Dantzig's linear programming, appeared in the 1940's (Dantzig, 1949) and was used at that time by the US military. Since then, a rising interest in optimization has led to the development of dozens different algorithms used in a wide range of applications. The major approaches will be discussed in the next paragraphs.

2.1.2 Gradient-based and Gradient-free Algorithms

Optimization algorithms are generally divided into two main categories: conventional gradient-based and gradient-free methods (Deb, 2001). Gradient-based approaches directly use mathematical tools to find optimal solutions. Some examples of gradient-

based algorithms are the Sequential Quadratic Programming (SQP) (Fletcher, 1979) and the Hookes-Jeeves algorithms (Hooke and Jeeves, 1960). The working principle is that from an initial value, the local gradient information is used to establish a direction of search at each iteration, until an optimum is reached. This kind of algorithms only work with objective functions which are twice differentiable or that can be approximated by terminated first order or second order Taylor series expansion around the initial guessed value (Deb, 2001). While this type of approach has been used in past studies such as the optimization of heating system (House and Smith, 1995), or more recently for the optimization of a cooling plant control scheme (Sun and Reddy, 2005), it suffers from two major limitations.

First, gradient-based methods are prone to local extrema. Depending on the starting value, they are likely to get trapped in the nearest local optimal value, missing the actual optimum. Taking several different initial values could eventually be seen as a solution to overcome this problem but it would provide little more guarantees, and may become a pure random search (Wang and Jin, 2000). The second major limitation of gradient-based approaches is that, as stated above, they only work with differentiable or at least relatively smooth functions. As far as building phenomena are concerned, functions are very often non-linear problems. Moreover, both discrete and continuous variables are involved, which may lead to discontinuous outputs (Wetter and Wright, 2003; Lu et al., 2005). Gradient-based methods are thus not suitable for most building applications. Since Artificial Neural Network (ANN) outputs are generally highly non-linear functions, gradient-based methods cannot either be combined with ANN.

The second and more modern school of optimization techniques, referred to as gradient-free, relies on stochastic techniques rather than derivatives to determine the search direction. This behavior allows the exploration of the whole search space, focussing only on regions of interest. Unlike the techniques previously described, gradient-free approaches can easily avoid local extrema and have proven their efficiency on optimization problems where classical methods fail (Goldberg, 1989). Several different algorithms from this school of optimization have been developed. A review of the predominant ones used for building applications is detailed by Wetter and Wright (2004). Between all gradient-based techniques, population-based techniques and more precisely Genetic Algorithms are predominant and have proven their efficiencies in hundreds of cases; they will therefore be discussed in more details.

2.1.3 Genetic Algorithms

Genetic Algorithm (GA) is an optimization technique developed by Holland (1975) in the 1970s and is based on Darwin's theory of evolution. GA's principle is simple, although unusual. In a nutshell, each solution is referred as an individual, which may further produce children, and on which an evolution mechanism is applied. GA has been used in a wide range of studies, from medicine (Lahanas et al., 2003) to transportation engineering (Syarif and Gen, 2003). Regarding building applications, GA are frequently used, for the optimization of building thermal system design (Wright et al., 2002), the optimization of HVAC controls (Huang and Lam, 1997; Lu et al., 2005), and the minimization of a chiller energy costs (Chow et al., 2002).

The pseudo algorithm of GA is displayed in Figure 1, and can be described with the following steps:

- First, a random population is created, where each individual represents a candidate solution. Individuals are modelled as a set of parameters.
- At each generation, couples of individuals (referred as parents) produce new solutions by gene-crossover and mutation (these new individuals are referred as offspring)
- At the end of each generation, the candidate solutions are evaluated using a so-called evaluation function (or objective function), representative of the objective studied. For building applications, this function can typically be the energy consumption.
- The last two steps operate until the termination criterion is reached (generally based on the number of generations, or on the stagnancy of population fitness)

```
BEGIN
     INITIALIZE population with random candidate
     solutions;
     EVALUATE each candidate;
     REPEAT UNTIL (TERMINATION CONDITION is satisfied)
     DO
          1 SELECT parents;
          2 RECOMBINE pair of parents;
          3 MUTATE the resulting offspring;
          4 EVALUATE new candidates;
          5 SELECT individuals for the next generation
     END DO
END
```

Figure 1: Basic Genetic Algorithm pseudo-code

As a gradient-free method, GA is able to deal with nonlinear functions, and to find global optima without being trapped in local ones. Furthermore, it can handle real, discrete, or even discontinuous variables, and be applied on noisy objective functions (Wright et al., 2002; Huang and Lam, 1997). Regarding the efficiency, GA is recognized to enable very detailed optimization and is capable of finding optimal or near optimal solutions using less computational time than other algorithms (Sakamoto et al., 1999, Wetter and Wright 2003). Another quality of GA is that it can be used for true multiobjective optimization. GA has been able to successfully handle multiple objectives, where other evolutionary algorithms such as particle swarm optimization have failed (Srinivasan and Seow, 2003). One last quality of GA is that it can perform very well when associated with response surface approximation methods (Chow et al., 2002; Lu et al., 2005).

A main drawback of GA is the high number of calls to evaluation function. In building applications, these evaluations are generally estimated by an external simulation program such as CFD or other simulation softwares. If accurate results are required, each evaluation can be time consuming, and thus the complete computational process becomes extremely unattractive. For instance, for the two-objective optimization of building floor shape, Wang et al. (2006) used an evaluation tool where each evaluation took 24 seconds (CPU-time). In that case, the total optimization time, which is mainly due to evaluations, was 68 hours. Based on a simple rule of three, one can expect that, using a simulation software where each evaluation would take thirty minutes, a similar optimization would result in a total optimization time of more than 6 months. Despite all its qualities, the use

of genetic algorithm is therefore strongly limited by the high number of evaluations it requires. This shortcoming should be overcome before being able to take full advantage of this technique.

2.1.4 Artificial Neural Network

Artificial Neural Network (ANN) is not an optimization method by itself. However, it is a very efficient approximation method that can be used inside an optimization. ANN is a Response Surface Approximation (RSA) technique, with an architecture based on the human brain. It was first studied in the late 1940s and later developed during the 1980s. An ANN is aimed to provide a fast and accurate approximation of a given system, based on a set of inputs and outputs. It can be applied to any kind of systems, and is argued to be able accurately simulate it, as long as training is sufficient. ANNs have been widely and successfully used in a number of engineering studies, including building applications (Yang et al., 2005; Pala et al., 2008).

The ANN architecture is based on the human brain neural network (Figure 2). The input information passes through several layers of neurons, in which signal is processed, in order to deliver the final output. More precisely, an ANN is composed of a layer of input nodes (representing the system variables), a layer of output nodes (approximated results), and at least one hidden layer connecting the input and output layers. Each node of a hidden layer is connected to those of the previous and following layers, and computes a specific output as a reaction of its inputs.

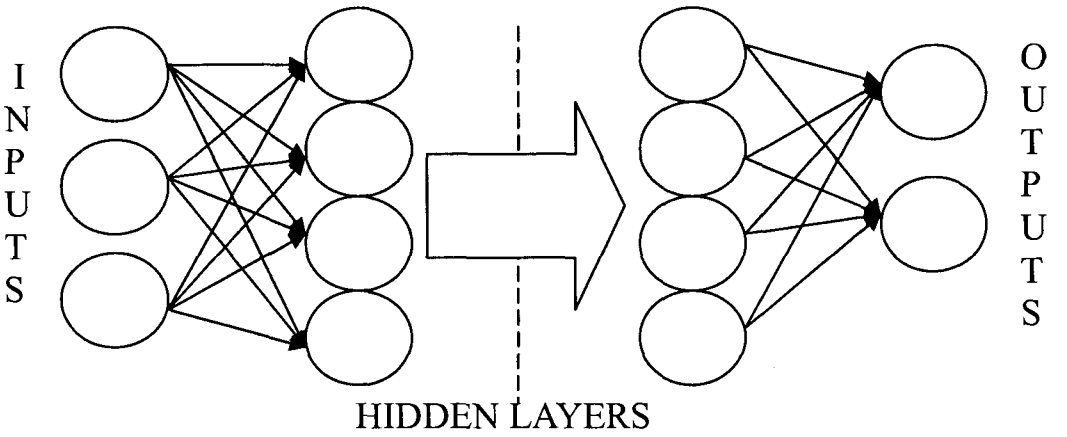
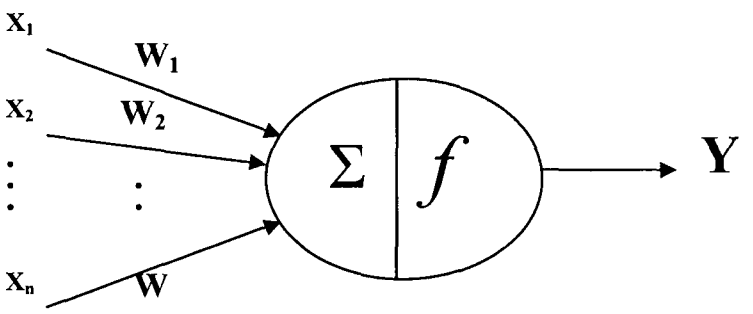


Figure 2: Artificial neural network schematic view

Each node of an ANN is a processing neuron, as depicted in Figure 3. The neuron receives a signal (x_i) from each neuron in the previous layers, and applies a specific weight (w_i) on this signal. All signals are then added together and a transfer function is applied to the weighted sum to generate the neuron's output (Y).



Where ;
x_i: is the neuron input
W_i is the weight associated with the input x_i
Σ is the weighted-sum of inputs
f: is the transfer function
Y is the neuron output

Figure 3: One neuron in ANN

There is theoretically no limitation for the numbers of inputs and outputs, nor for the number of neurons. In particular, several outputs can be simulated and ANNs are therefore suitable for multiobjective purpose. Network complexity and especially the number of neurons in hidden layers have a significant influence on the accuracy and computational time. According to MATLAB instructions (2006), a two-layer neural network using sigmoid transfer functions for the first layer and linear functions for the second layer is able to approximate any function having a finite number of discontinuities.

The underlying concept of ANNs is learning. Once parameters such as the number of layers, the number of neurons and activation functions are chosen, the network will learn by itself how to approximate a given system by calculating the proper weights. This process is called training, and requires a set of data containing inputs and related outputs. This set of data has to be precise (often taken from experiments and/or validated simulation programs), in sufficient quantity, and representative of the range of parameters the ANN is supposed to approximate. Once a neural network is trained, it has to be tested with a new set of data, different from the previous one. Data is processed by the ANN and the resulting outputs should be as close as possible to the validated outputs. If the differences between the ANN outputs and the base program/experiment's outputs are lower than 5%, the ANN is validated.

Various methods can be used for training, with various efficiencies. The most commonly used method is the *back propagation method*, which can be enhanced by using Levenberg-Marquardt and Bayesian algorithms (MATLAB, 2006). The quality of the training dataset is also crucial for the ANN's accuracy. Use of Latin Hypercube Sampling or orthogonal sampling is generally recommended to generate a small but very representative case sample (Lee et al., 2006)

Although the ANN is very widely used for approximation, it is not the only RSA algorithm available. Other major techniques exist such as Kernel Recursive Least Squares (Engel et al., 2004), or algorithms based on regression trees (Li et al., 2000). There is no general agreement on which RSA technique is the most efficient. While several comparison studies have been published (Simpson et al., 1998; Jin, 2005), there are no strong conclusions on assets or drawbacks of each method since performances depend on problems studied. Nonetheless, Artificial Neural Network appears to be amongst the most reliable approximation models, both in term of efficiency and range of problems that can be modeled. Moreover, due to its good acceptance in the scientific community, the ANN is readily available in several computer programs, like MATLAB for instance.

2.1.5 Summary Regarding Building Applications

In building engineering, the optimization process can be applied either to the design of the building (Conraud, 2008), for the settings of the HVAC system (Nassif et al. 2003), or for ongoing optimization (Coffey, 2008). In any case, the optimization algorithm should be fast, efficient, and reliable. Studying gradient-based algorithms, those display several weaknesses, such as being limited to differentiable functions and being prompt to local optimum (Deb, 2001). Though gradient-based algorithms are still used in some studies (Sun and Reddy, 2005), gradient-free algorithms such as GAs are now often preferred (Amirjanov and Sobolev, 2006). GAs have proven to be extremely efficient in terms of optimization, and can handle multiple objectives (Wang et al. 2006). The shortcoming of GAs is the very high number of evaluations they require, especially for multiobjective optimization (Deb, 2001). Therefore, building optimizations using GAs either use very small populations and numbers of generation (Caldas and Norford, 2002; Wetter, 2004), or are based on very simplified models instead of using a complete simulation software (Peippo et al. 1998). In both situations, the optimization can be significantly affected. According to Conraud (2008), it is crucial to decrease the computational time associated with GAs, in order to see optimizations more widely used by building designers.

2.2 GAINN Methodology

2.2.1 Description of GAINN

As explained above, GAs are very efficient tools for optimization. They are gradient-free (thus having less chance to fall on local extrema), able to deal with non-linear objectives functions and provide optimized results. The main drawback of this technique is the computational time for reaching optimal solutions. This drawback becomes a serious limitation for building applications since computer simulation programs such as TRNSYS, ESP-r, or EnergyPlus can be time consuming to operate. Consequently, it is necessary to find a way to reduce the evaluation time in order to take full advantage of GA capabilities while keeping a reasonable optimization time.

GAINN stands for Genetic Algorithm Integrating Neural Network, and is an interesting though greatly unexploited approach to reduce optimization time while using GA. The main idea of GAINN is to benefit from the rapidity of evaluation provided by the ANN as well as the optimization power of the GA. The procedure is to first use an ANN to approximate the system studied, and then use this ANN inside the GA as the objective function. The outcome is a drastic reduction of the simulation time, while keeping an acceptable quality and reliability in solutions.

The complete workflow of GAINN is illustrated in Figure 4, and is divided in three steps. First, a base software or experimental set-up is used to generate a database of cases. Once the database is created, it can be used to train and validate the artificial neural network. The ANN is then integrated into the genetic algorithm as the evaluation function, so the

GA can run with almost instantaneous evaluation of individuals. The GA optimization finally provides the optimal solution set, which can further be checked for accuracy using the original simulation software.

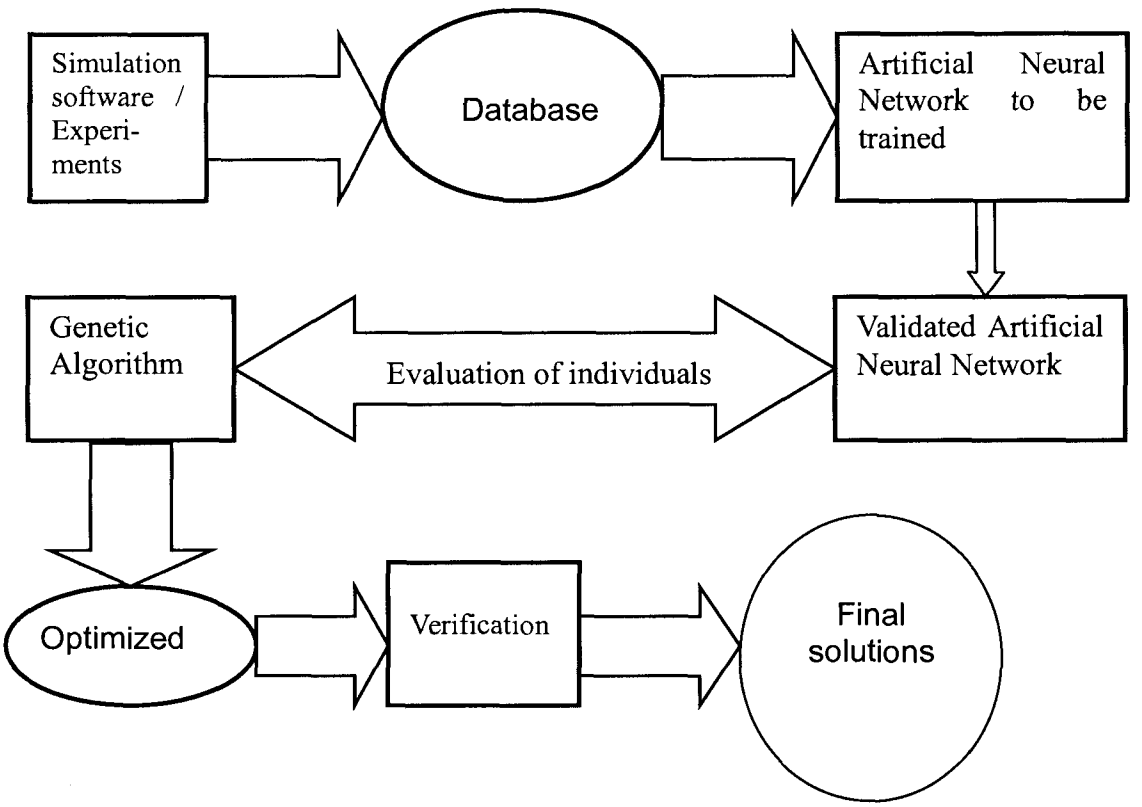


Figure 4: Workflow of GAINN methodology

2.2.2 Literature Review on GAINN

The integration of GA and ANN is not a new idea. Such integration can be found as early as 1993, applied for the optimization of plant growth (Morimoto, 1993). GAINN has later been applied in various domains such as in chemical engineering (Nandi et al., 2002), or

for engine optimization (Kesgin, 2003). Focusing on building applications, GAINN has been rarely documented. The first use of GAINN in building engineering was in 2002, for the optimization of chillers control (Chow et al. 2002). This study introduced the methodology to the building field, and proved its efficiency in terms of accuracy and reduction of the total optimization time. Later, GAINN has been successfully applied in other studies, such as Zhou (2007), combined with Computational Fluids Dynamics, and Conraud (2008), combined with ESP-r.

According to the previous studies, GAINN methodology can be very efficient for building applications. Due to the ANN evaluations inside the GA, a significant amount of time can be saved, while keeping the optimization reliable. In Zhou (2007) for instance, the total optimization time using GAINN was found 17 times lower than the optimization time expected if CFD was directly used for GA evaluations. Regarding the accuracy of the method, results can be trusted as long as the ANN is validated correctly. The use of Latin Hypercube Sampling is recommended to create a small and representative database for ANN training (Lee et al., 2006). Nonetheless, the approximation of some complex issues such as visual comfort may be difficult to achieve. Conraud (2008) for instance, had to use 25 times more training cases than what is generally recommended by the LHS method to accurately train his ANN. Indeed, one main limitation of GAINN is that the optimization relies on the ANN accuracy. If the ANN is not 100% accurate in the vicinity of the optimal solution, results could be affected, and optimal solutions could be missed.

Another major drawback regarding how GAINN methodology has been applied so far is the handling of multiple objectives. In the great majority of previous studies (with the

notable exception of Amanifard et al., 2008), multiple objectives were handled by using aggregative weighted-sums. This method suffers from many limitations, such as being dependent on stated assumptions and on the initial situation. It also provides no guarantee to reach optimal solutions from a multiobjective point of view (Jain et al., 2005).

Finally, very theoretical improvements of GAINN methodology have been studied (Jin, 2005). These improvements belong to computer engineering and are beyond the scope of this thesis; they will thus not be discussed in details. In a nutshell, one promising mechanism is to include ANN training inside the optimization, with training data based on the GA's current population (Nain and Deb, 2005). Though interesting, such approach used for building applications would require a continuous linking between ANN, GA, and the base simulation software, which is unpractical.

2.2.3 Conclusion

The GAINN methodology is a very promising approach for building optimization, and provides equilibrium between accuracy and efficiency. To the author's opinion, the methodology nonetheless requires more studies, and has been underexploited. The main development to be added to GAINN methodology is a better handling of multiple objectives, by the implementation of a true multiobjective genetic algorithm. Regarding the efficiency, a true multiobjective optimization would no longer be dependent on stated assumptions, nor on the initial situation. Another drawback of optimizations based on weighted sum is that each run provides a single so-called "optimal" solution. A true multiobjective optimization provides a curve or a surface of solutions, and can therefore

enable a better understanding of the problem, and give more flexibility to the decision maker. Moreover, since much more solutions are provided, the methodology would more efficiently exploit the time spent for training, compared to previous studies where one single optimal solution was provided after days of calculation.

This thesis will therefore focus on the use of multi-objective optimization algorithms inside GAINN. This development is expected not only to improve the optimization efficiency but also to make the methodology closer to real-world scenarios and sustainable issues. Finally, multiobjective optimization algorithms would more efficiently exploit the assets of the methodology and therefore give a stronger justification to the time spent for training. In a nutshell, it will make the methodology closer to industrial needs and more attractive for potential users.

CHAPTER 3

BASE MULTIOBJECTIVE EVOLUTIONARY ALGORITHM

3.1 Multi-Objective Evolutionary Algorithms

This chapter reports a comprehensive review of multiobjective optimization. First, the notion of Pareto-optimality will be presented. Then, the major multiobjective optimization algorithms will be briefly discussed and a suitable base algorithm will be chosen for the current study. This algorithm will be programmed and tested to validate its efficiency, compared to the original version.

3.1.1 Pareto Optimality

The concept of Pareto optimality or non-dominance is the basis of multiobjective optimization. This notion, originally proposed by F.Y. Edgeworth (1881) and later generalized by V. Pareto (1896), can be described as follows:

For a multiobjective optimization problem of the form:

$$Minimize\ [f_1(x), f_2(x), \ldots, f_k(x)]$$

Where $x \in F$ is a vector of decision variables (subject to several constraints), and;

f_i are the objectives functions

A vector x is Pareto optimal if there does not exist any other vector y such that $f_i(y) \leq f_i(x)$ for all i and $f_j(y) < f_j(y)$ for at least one j.

In other words, a vector is said to be Pareto optimal, or non-dominated, if there does not exist any vector which could decrease some of its objectives, without increasing at the same time at least one other objective.

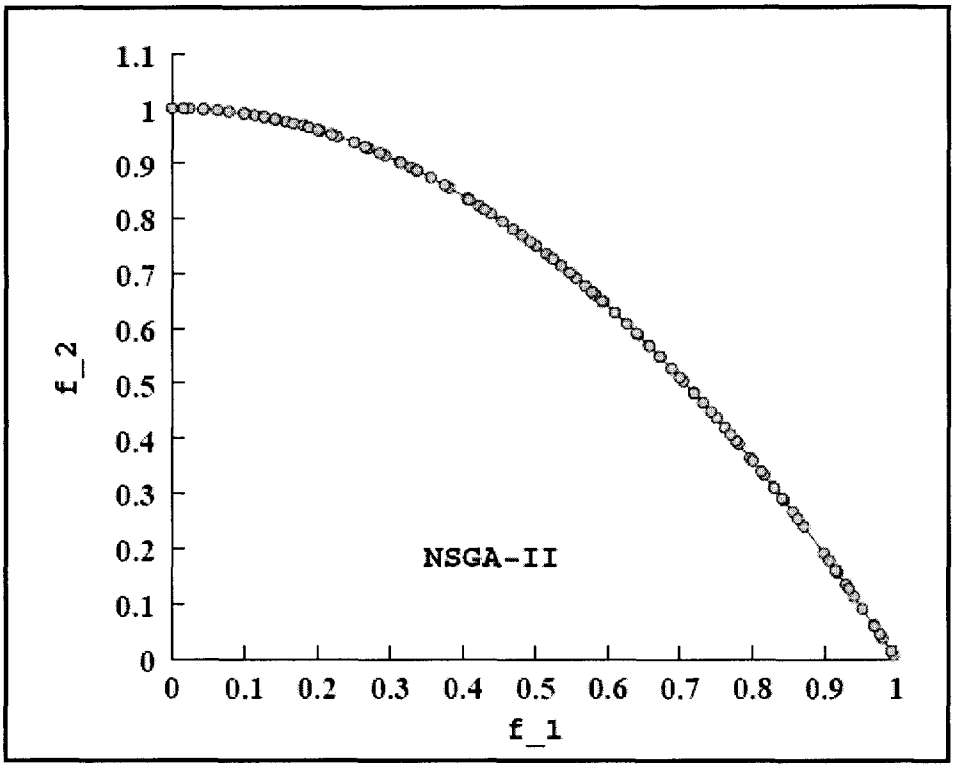


Figure 5: Example of a Pareto Front (Deb, 2002)

The notion of dominance is crucial in multiobjective optimization. Two individuals do not necessarily dominate each other, and a non-dominated individual should always be regarded as the best of its kind. Therefore, Pareto-based optimization cannot lead to a single solution but to a set of solution, named Pareto-optimal set, where all solutions are

Pareto-optimal. The frontier of the solution set is called Pareto Front. It can typically be illustrated as a line for two-objective problems (Figure 5) and a surface for three-objective ones.

The quality of a solution set can be assessed using two parameters. The first one is the convergence, representing how close each point is to the true Pareto front, i.e. how optimized solutions are. The second quality of a solution set is the spreading of solutions over the Pareto front. Solutions should be widely spread to cover the whole range of possibility. Convergence and spreading are predominant in multiobjective optimization study. They are illustrated in Figure 6.

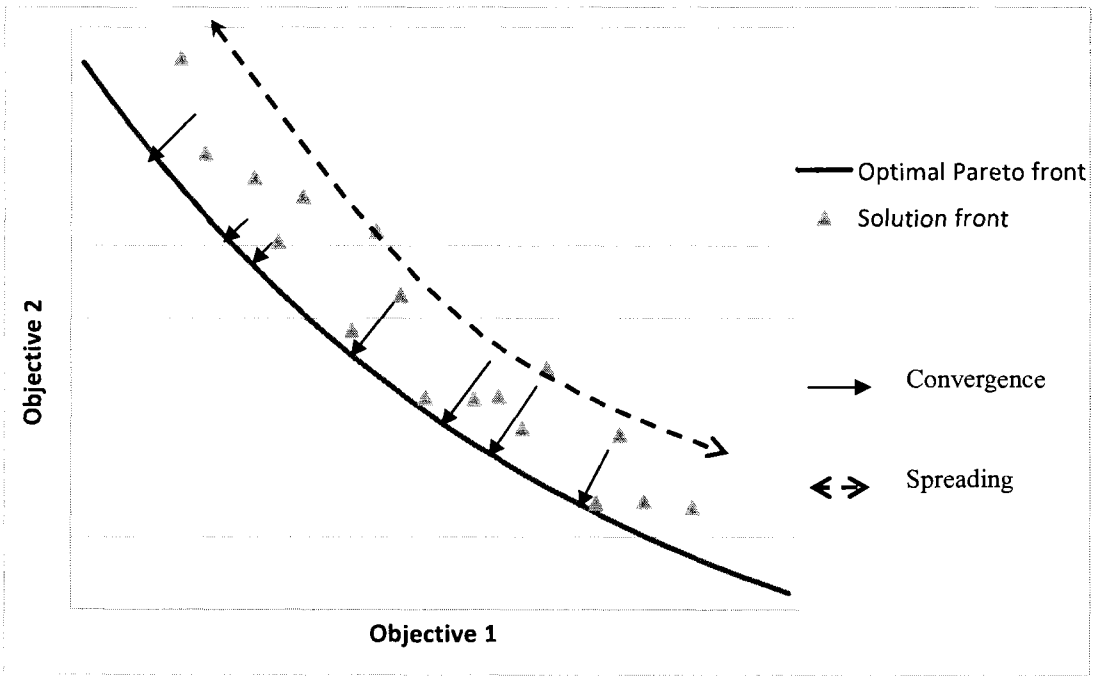


Figure 6: Convergence and spreading of a solution set

3.1.2 Multiobjective Evolutionary Algorithms

The Pareto-based optimization approach was initiated by Goldberg at the end of the 1980s. Unlike the aggregative/weighted-sum method, or the population-based approach, the Pareto-based approach enables a true multiobjective optimization, where all objectives are optimized simultaneously. This approach does not require any weights to be set (unlike weighted-sum) and is independent of the initial situation. Since all objectives are optimized at the same time, any trade-off is considered as a solution. The optimization is also generally more efficient. Finally, the solution set provided is composed of dozens of solutions, and therefore offers a wide variety of choices to the decision maker.

Algorithms based on the Pareto concept are referred Multiobjective Evolutionary Algorithms (MOEAs). These algorithms are in most cases similar to classical GAs, and keep the same assets (efficiency, gradient-free, etc.) and drawbacks (number of evaluations mostly). The main difference between MOEAs and one-objective GA occurs in the selection process. In this selection, the concept of non-dominance is introduced, using various methods, to evaluate each candidate solution.

Many different MOEAs have been developed over the last decades, and have been carefully studied by the author. In a nutshell, differences between MOEAs lie in the method they use to handle both convergence and spreading. Spatial considerations especially are subject to discussion, and may be handled using various techniques, generally based on a division of the search space. Another issue carefully studied is how

to maintain a sufficient diversity inside the population, in order to avoid local extrema and to expand the Pareto front. An important notion called elitism is also introduced in "second-generation" MOEAs, to ensure that valuable solutions cannot be lost and that fitness of the population can only increase.

For this thesis, the base MOEA will be the Non-dominated Sorting Genetic Algorithm II (NSGA-II), developed by Deb et al. (2001). According to most reviewers (Zitzler et al. 2000, Jain et al. 2005), this algorithm is one of the most efficient MOEA in terms of both convergence and spreading of the solution set. It is based on a simple structure and requires less calculation time than most MOEAs need. Moreover, NSGA-II requires very few parameters to be set, which makes it accessible to non-experts (unlike SPEA2 for instance). NSGA-II has been intensively used over the last years in various domains and is recognized for its reliability (Majumdar et al. 2005, Fu et al. 2008). It is also relatively simple to program and to customize.

3.2 NSGA-II

3.2.1 General Description of NSGA-II and Pseudo-code

The pseudo-code of NSGA-II is shown in Figure 7. NSGA-II follows the same steps as classical GAs. First, it initializes a random population of N individuals, then it produces children/offspring by recombination and mutation, evaluates the individuals, and finally selects the fittest ones. Several aspects of NSGA-II are however very specific to this algorithm:

- The parental population is chosen through a tournament selection. This selection process enables to select a parent based on both convergence and spreading, while maintaining a reasonable diversity amongst the population.

- The genetic operators used inside NSGA-II are generally (although not necessarily) the Simulated Binary Crossover, and the Polynomial mutation. These operators use a stochastic approach to determine children genes, based on the genes of their parents (a more detailed description can be found in Appendix B). They are extremely efficient when real variables are used.

- The selection process is computed at each generation on an intermediate population combining both parents and offspring. Therefore, no valuable solution can be lost, which makes NSGA-II elitist.

- For the selection, NSGA-II uses a non-dominated-and-crowding sorting and selection.

```
BEGIN
I) INITIALIZE population with random candidate solutions;
II) EVALUATE each candidate;
III) REPEAT UNTIL (TERMINATION
CONDITION is satisfied) DO :
        1 SELECTION of parents by tournament selection;
        2 RECOMBINATION of pair of parents to produce offspring;
        3 MUTATION;
        4 NON-DOMINATED-AND-CROWDING-SORTING of parents and children
        5 SELECTION of individuals for the next generation based first on rank
        and further on crowding distance
IV) END DO
END
```

Figure 7: Pseudo-code of NSGA-II

3.2.2 Non-dominated-and-crowding Sorting and Selection

The non-dominated-and-crowding sorting and selection is the key mechanism from which NSGA-II takes its efficiency. This process enables to focus on the convergence of the population while maintaining a very good spreading of the population. It is based on the two following parameters

The first parameter used is the *rank* of an individual. The notion of rank is closely related to dominance. In a population, non-dominated individuals have a rank of one, they

belong to the first front. Individuals which are dominated only by solutions from the first front belong to the second front, and are assigned a rank of two. More generally, all individuals having a same rank do not dominate each other, but they dominate individuals with a higher rank, and they are dominated by individuals with a lower rank. Therefore, the notion of rank enables to compare an individual with the whole population regarding convergence.

The second parameter is the *crowding distance* of an individual. As its name says, the crowding distance represents how crowded the space around the individual is. A small crowding distance implies that individuals are close to each other, and thus diversity is low. In order to increase diversity and expand the Pareto front, individuals with the highest crowding distances should be preferred. In NSGA-II, the crowding distance of an individual is calculated as follows. First, the population is sorted in descending order regarding a specific objective. Then the crowding distance of extrema are set equal to infinite; for all other individuals, the following calculation is computed:

$$dist(i,obj) = \frac{Value(i-1,obj) - Value(i+1,obj)}{MaxValue(obj) - MinValue(obj)}$$

Where: _ i is the individuals studied, and;

_ (i-1) and (i+1) are the two individuals respectively following and preceding i in the sorted population

Finally, for each individual, the distances associated with all objectives are summed. The result is the total crowding distance of the individual.

Once crowding distances and ranks are calculated, the selection process can be computed. The process is illustrated in Figure 8 and can be described with the following steps. Fronts are taken successively according to their ranks, then for each front:

- If the size of the front plus size of already selected population is inferior to N (population size), all individuals of this front are selected.
- If this size is superior to N, individuals with the highest crowding distance are selected, until a size of N is reached.

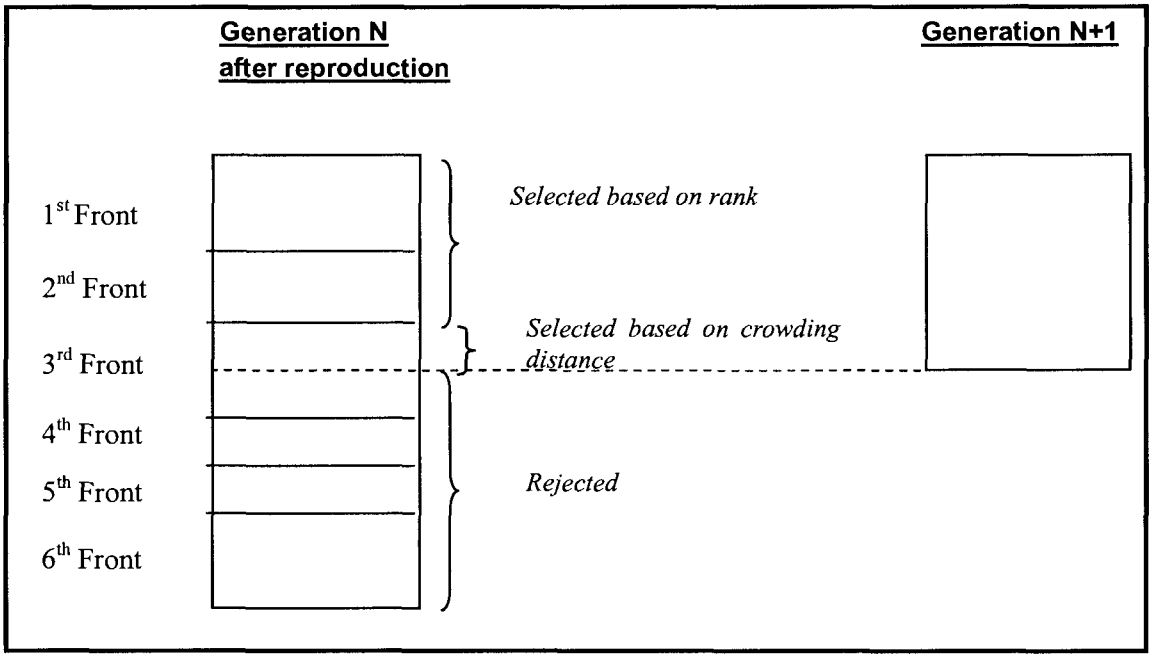


Figure 8: Non-dominated-and-crowding selection

3.2.3 NSGA-II Programming

The base calculation program for this thesis is MATLAB. MATLAB was chosen because it is a calculation program widely recognized and used in the industry, and because it already integrates a very user-friendly Neural Network Toolbox. NSGA-II algorithm was not readily available in MATLAB toolboxes but a very basic version of NSGA-II, developed by A. Seshadri was available in MATLAB user's community[1].

Based on this rather limited code, the author developed a complete NSGA-II program. The developed code is very robust, and user-friendly. All MOEA parameters can be changed from a single input file, and both discrete and real variables are supported. A special care was taken to make the program as fast and reliable as possible. In particular, the complex mechanism used for non-dominated-and crowding-sorting can be computed using various methods; the method chosen for this thesis is the fastest one described in Deb's book "Multi-objective optimization using evolutionary algorithms" (2001). The complete code of the program can be found in Appendix A.

[1] http://www.mathworks.com/matlabcentral/fileexchange/10429, accessed on March 2007

3.3 Validation of the NSGA-II programmed

3.3.1 Testing Methodology

NSGA-II is the base-case MOEA with which the algorithms further developed will be compared. Therefore, it is crucial to make sure that the programmed version of NSGA-II is as efficient as the original. Although the author programmed the algorithm very carefully, some tests were performed to ensure that the algorithm is working properly. The testing methodology was based on Deb et al. work (Deb, 2002). This paper was selected because of the amount of documentation it provides about NSGA-II programming, metrics used for tests, and results. For a purpose of comparison, the programmed version of NSGA-II was tested on the same test functions and with the same parameters as in Deb's study (Table 1). As in the base study, each function was tested 10 times, and the average results were studied.

Population size	Crossover type	Mutation type	Crossover probability	Mutation probability	Termination criterion :
100	Simulated Binary Crossover	Polynomial mutation	0.9	1/(number of variables)	250 generations

Table 1: Parameters for NSGA-II tests (Deb, 2002)

In order to quantify MOEA's efficiency, the two metrics of Deb's study were used: Y and Δ. These metrics base their calculations on a 500 individuals Ideal Pareto Set (referred as IPS) where all solutions are optimal and equally spaced. In this thesis, IPS came from

either mathematical formulation, from specific MOEA solutions websites[2], or from calculated solution sets. (In the later case, NSGA-II was run with a population of 500 individuals for at least 1000 generations, which is expected to produce perfect or almost perfect solution sets.) The two metric used are the following ones:

Y metric: The Y metric is used to quantify the convergence of the solution front. It is calculated as the average of distances between each solution found and the closet IPS solution (Figure 9). The lower Y is, the closer results are from optimal solutions.

Δ metric: The Δ metric quantifies the spreading of the solution front (Figure 10). The lower the metric is, the better is the spreading. It is defined as:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \overline{d} \right|}{d_f + d_l + (N-1) \times \overline{d}}$$

Where:

- d_f and d_l are the Euclidean distances between the extrema solutions found and the extrema solutions of IPS;
- d_i is the Euclidean distance between two consecutives solutions;
- and d is the average all d_i.

[2] http://delta.cs.cinvestav.mx/~ccoello/EMOO/testfuncs/

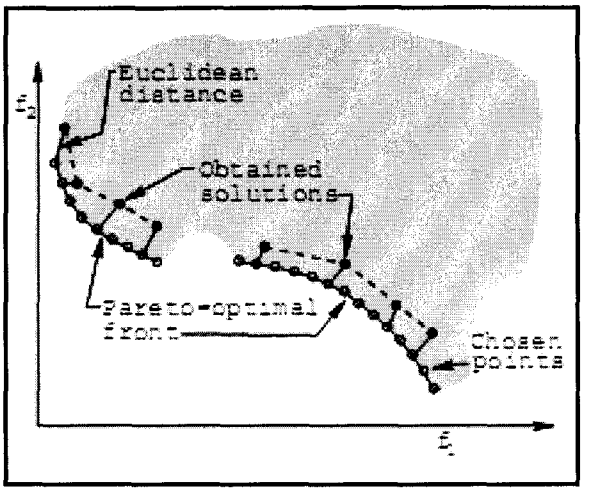


Figure 9: Illustration of the Y metric (from Deb, 2002)

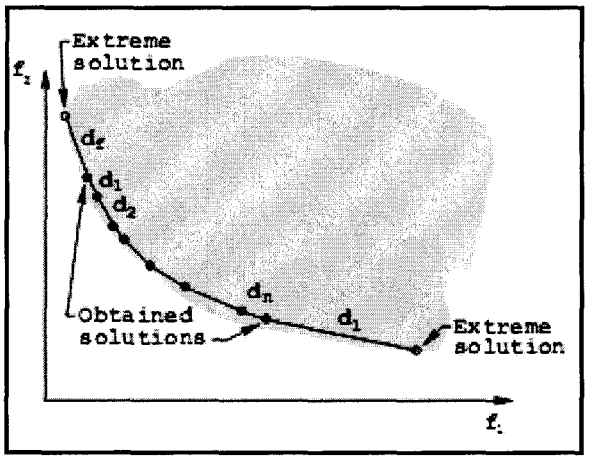


Figure 10: Illustration of the Δ metric (from Deb, 2002)

3.3.2 Results

Results are summarized in Table 2 and in Figures 11 and 12. Some solutions sets are illustrated in Figures 14a, b, c, and d. Regarding convergence to Pareto front (Figure 11), we can see a good agreement between results from programmed version of NSGA-II and results from the original program for the first three functions. For latter functions,

surprisingly, the programmed NSGA-II performed better than the original one. Except for FON function with a 14% error increase, all results were as good as or better than original ones.

Test function		SCH	FON	POL	KUR	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Y mean	*Current thesis*	0.0034	0.0022	0.0134	0.0123	0.0015	0.0010	0.0045	0.0039	0.0008
	Deb (2002)	0.0034	0.0019	0.0156	0.0290	0.0335	0.0724	0.1145	0.5131	0.2966
	% Difference	0%	14%	-14%	-58%	-96%	-99%	-96%	-99%	-100%
Δ mean	*Current thesis*	0.282	0.403	0.954	0.500	0.410	0.426	0.679	0.383	0.620
	Deb (2002)	0.478	0.378	0.452	0.412	0.390	0.431	0.739	0.703	0.668
	% Difference	-41%	7%	111%	21%	5%	-1%	-8%	-45%	-7%

Table 2: Comparison of convergence and spreading metrics for programmed and original NSGA-II

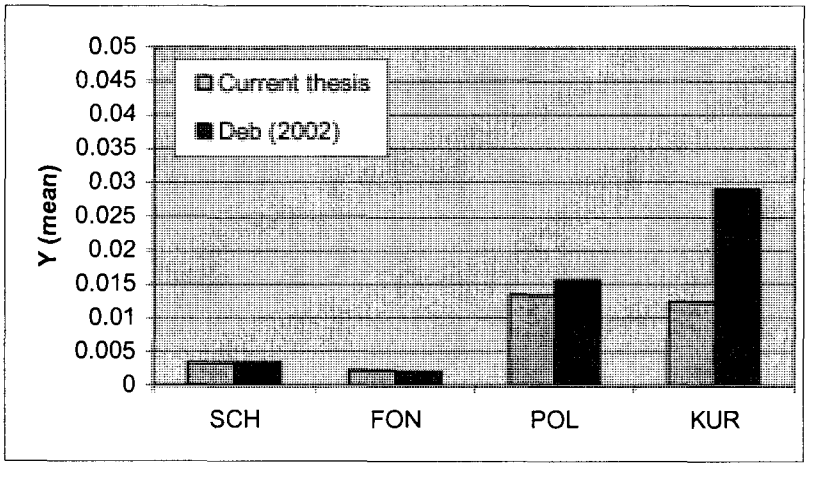


Figure 11: Mean of convergence metric Y for programmed and original NSGA-II

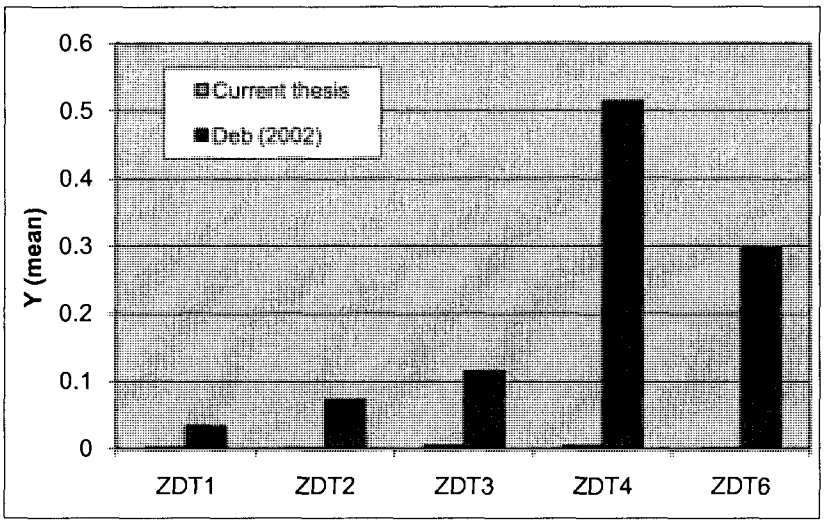


Figure 12: Mean of convergence metric Y for programmed and original NSGA-II

Regarding spreading of solutions (Figure 13), results from programmed version of NSGA-II were similar to Deb's results, except for POL and ZDT4 function. The average difference between programmed and original NSGA-II was 8.73%. Regarding POL function, the Δ metric was found two times higher than that of the original study.

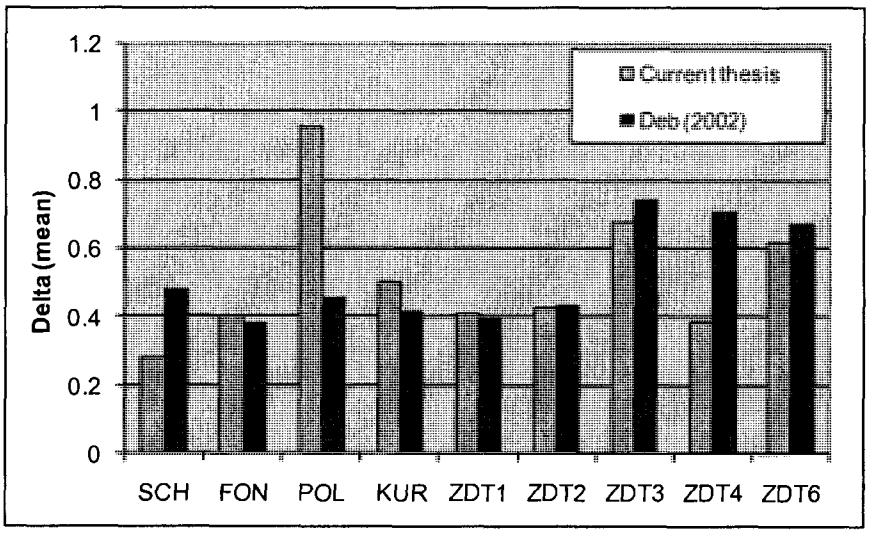


Figure 13: Mean of diversity metric Δ for the programmed and the original NSGA-II

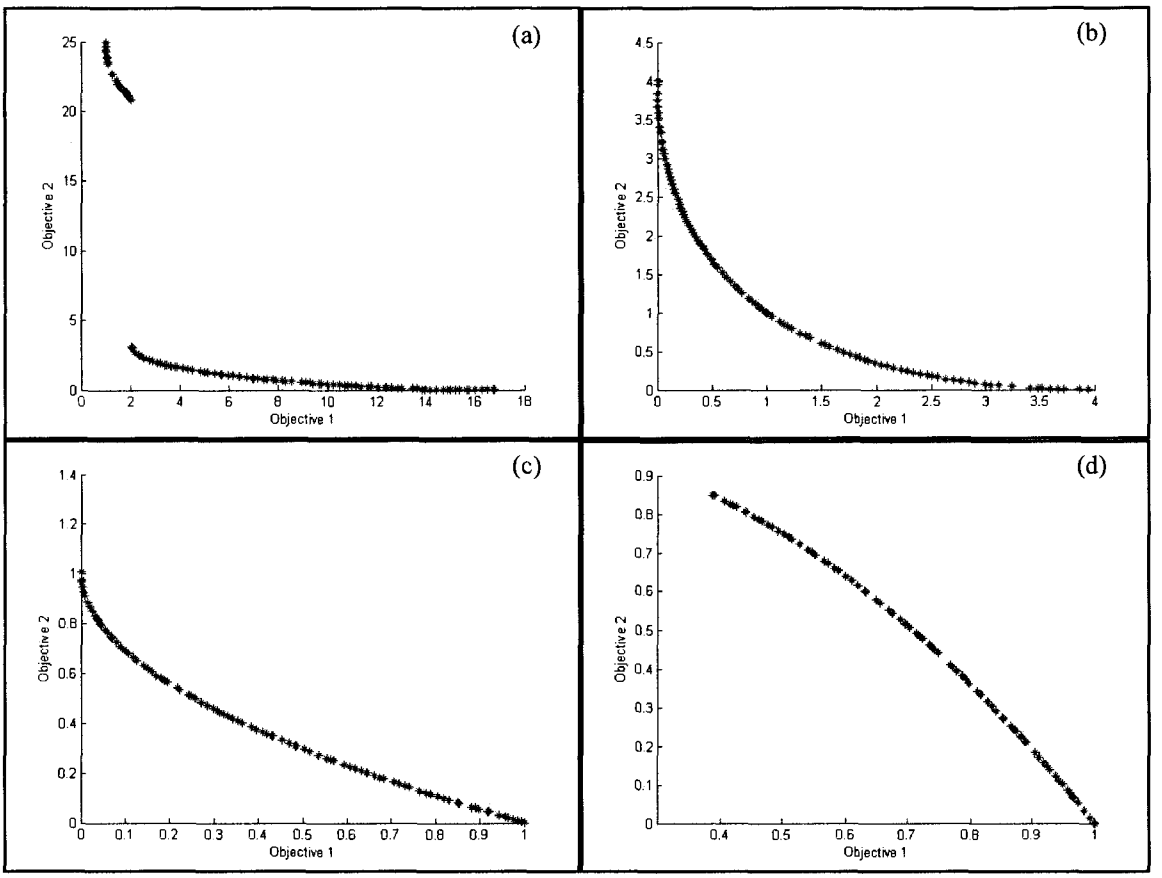


Figure 14: Solutions sets for POL, SCH, ZDT4, and ZDT6 Functions respectively

3.3.3 Discussion

Major differences appeared in the Y metric between the programmed and the original version of NSGA-II for the ZDT suite. This difference can be caused by several calculation parameters, not specified in the original study. In particular, the SBX operator used for reproduction exists in two forms (for bounded or for unbounded variable) and may use different behaviours when handling multiple variables (50% genes changed, all genes changed, etc.). Population sizes used for parental tournament selection may also influence the results. Then, the Ideal Pareto Solution sets used in this chapter may be

different from the ones used in the original study. This can furthermore gender discrepancies. Nonetheless, the results of the programmed version of NSGA-II were in all cases (except for the FON function) better or similar to the results of the original study.

Regarding the Δ metric, we can generally see a good agreement between results from the programmed version of NSGA-II and results from the original version. The high discrepancy appearing for the POL function was most likely due to the discontinuity of its Pareto front (Figure 14a). The original paper is unclear about how calculation is performed for discontinuous front and especially if a different Δ value should be calculated for each continuous part of the front. In the present work, Δ was only calculated one time for the whole front. Regarding ZDT4, a relatively large difference appeared (45.47%) but was in favour of the programmed version of NSGA-II.

As a conclusion, in almost all cases, the convergence of solutions coming from the programmed version of NSGA-II was better or similar to the convergence of the results of the original study. Meanwhile, the spreading of solutions was similar. In other words, the programmed algorithm is as efficient as the original algorithm, and is sometimes even more efficient. The process of comparing further developed MOEAs with the programmed version of NSGA-II is thus expected to be conservative. Therefore, the author decided to validate the programmed version of NSGA-II as a base case for further comparison.

CHAPTER 4

DEVELOPEMENT OF MULTIOBJECTIVE GENETIC ALGORITHMS

As discussed in Chapter 2, the number of evaluations is generally a very limiting parameter when using a MOEA. Since fitness evaluation may be a time-consuming process, the number of evaluations should generally be kept to a minimum. Accordingly, the behaviour of many MOEA is to create at each generation a given number of individuals, include all of them in an intermediate population, and then apply a sorting over this population. Such behaviour emphasizes more on the sorting than on the creation of new individuals to improve the overall fitness of the population. The basic assumption leading to this is that evaluations may be time consuming, and should therefore be limited.

In the particular case of GAINN methodology, the situation is radically different. Since evaluations are performed by an ANN, the time associated with each evaluation is negligible. Sorting of the population, in turn, becomes the time-expensive step. In order to take full advantage of GAINN methodology, a specific MOEA should therefore be used, with a behaviour that does not take into account the number of evaluations, but on the contrary maximizes the use of evaluations before proceeding to the complete sorting of the population. In the literature, the great majority of MOEAs, including NSGA-II, limit themselves in terms of number of evaluations, and are thus not suitable for the current case. Some interesting MOEAs such a SEAMO (Valenzuela 2002) or the Micro

Genetic Algorithm (Coello and Pulido, 2001) have been investigated as potential solutions. Unfortunately, the efficiency of these MOEAs has only been proven in problems with hundred of variables and objectives, very unlikely to appear in real situations.

The author finally decided to develop a new version of NSGA-II, adapted to the current methodology. An other and more particular MOEA was also developed, designed specifically to take advantage of GAINN fast evaluations. These two MOEAs will be presented and tested in this chapter. To the author's knowledge, these two MOEAs are original, and cannot be found in the literature.

4.1 Non-dominated Sorting Genetic Algorithm for Integrated Neural Network (NSGAINN)

4.1.1 General Description and Pseudo-code

The first algorithm developed is the Non-dominated Sorting Genetic Algorithm for Integrated Neural Network (NSGAINN). This algorithm is a variation of NSGA-II based on the idea of intelligently maximizing call for objective function before proceeding to sorting. The pseudo-code of NSGAINN is described in Figure 15. As can be seen, NSGAINN uses the same behaviour than NSGA-II in the first 80% of the run. In the last 20% of the run, a special procedure is used for reproduction, and a selection process is introduced to study which offspring should be included in the current population.

```
BEGIN

I) INITIALIZE population with N random candidate solutions;

II) EVALUATE each candidate;

III) REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO

    If time<80%*Maximum time

    NSGA-II behaviour:

        1 SELECTION of parents by tournament selection;

        2 RECOMBINATION of pair of parents to produce a total
        of N offspring;

        3 MUTATION;

        4 NON-DOMINATED-AND-CROWDING-SORTING of parents and
        children;

        5 SELECTION of individuals for the next generation
        based first on rank and further on crowding distance
        (NSGA-II selection);

    If time>80%*Maximum time

    NSGAINN behaviour:

        1 SELECTION of parents by tournament selection;

        While offspring population's size < N

            2 RECOMBINATION of pair of parents to produce 4
            offspring per mate;

            3 MUTATION;

            4 NSGAINN family selection;

        End Loop

        5 NON-DOMINATED-AND-CROWDING-SORTING of parents and
        children;

        6 SELECTION of individuals for the next generation
        based first on rank and further on crowding distance
        (NSGA-II selection);

    End

IV) END DO
```

Figure 15: Pseudo-code of NSGAINN

4.1.2 Salient Modifications Compared to NSGA-II

NSGAINN Family Sorting

In NSGA-II, all produced children are included in the population, regardless of their qualities. In the last 20% of NSGAINN run, a non-dominated sorting is applied over the family (meaning parents and all children) after each mating and only fit-enough children are kept. In details, two selection processes may occur, depending on advancements of the optimization.

- Family Selection 1 (FS1): Each child is compared with its parents. An offspring is included in the population only if it is non-dominated.
- Family Selection 2 (FS2): Children are compared with the whole family. An offspring is included in the population only if it is non-dominated and if it dominates at least one parent, or if it improves one objective's minimum so far.

Probabilities of occurrence are used for each of these selections processes, in order to gradually increase fitness pressure. A trial-and-error process has led to the following probabilities for each selection process (illustrated in Figure 16):

$$p(FS1) = 1 - \left(5 * \frac{current\_time}{Maximum\_time} - 4 \right)^{0.5}$$

$$p(FS2) = \left(5 * \frac{current\_time}{Maximum\_time} - 4 \right)^{0.5}$$

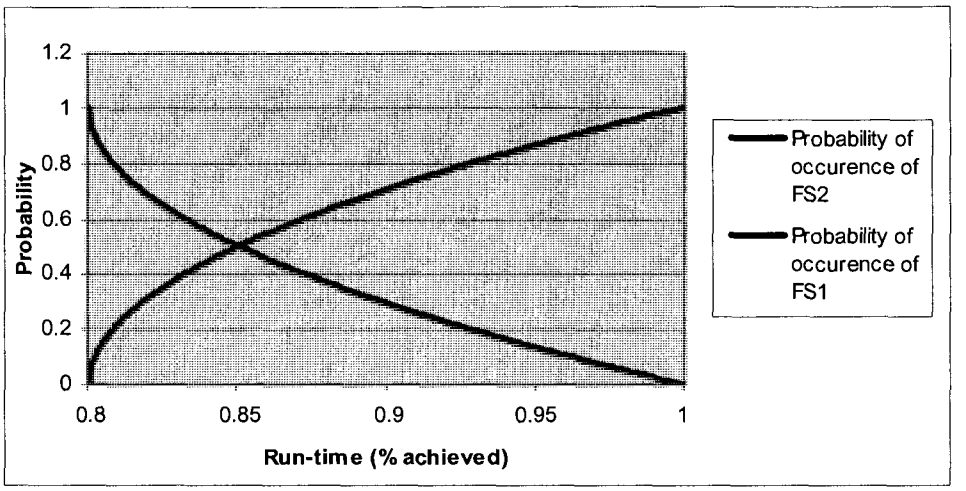


Figure 16: Probabilities for family sorting

Number of Offspring

The other important change of NSGAINN compared to NSGA-II is that in the last 20% of the run, each couple of parents produces 4 children at each mating (instead of 2 classically). This is made possible by the fact that for the multi-variable Simulated Binary Crossover, each variable has a 50% probability to be altered. In NSGAINN, the same 50% probability is kept, but if a variable is kept unchanged for one brotherhood, SBX is applied to the other brotherhood.

4.1.3 Discussion

In the first 80% of the run of a GA, the non-dominance of all individuals is not required, because the main purpose at this phase is to explore decision space, and to test different opportunities. Forcing the non-dominance at this phase would actually harm the diversity, which is well known to increase risks of getting trapped in local minima and produce narrow solution fronts. Also, since NSGAINN is based on the already efficient NSGA-II, gain would be unlikely during this phase.

However, in the last 20% of the run of a GA, population is expected to be relatively close to the Pareto front. The main goal is not to explore the solution space anymore but to increase closeness to optimal solutions and spreading of the solution set. Since the parent population is already close to optimal solutions, the probability to produce fitter offspring is low. If no selection mechanism is used and all offspring are introduced in the population, the time-consuming sorting of the whole population is computed to finally get a very little improvement in the overall fitness of the population. In NSGAINN, the two selection processes introduced ensures that a significant improvement is added to the population before proceeding to the sorting of the complete population.

In details, FS1, which keeps offspring only if they are not dominated by their parents, ensures spreading of the solution set, while keeping a moderate fitness pressure. FS2 is more aggressive and selects individuals only if they dominate their parents or if they extend the solution front. This behaviour forces the convergence of solutions or the extension of the solution front. Regarding probabilities of the two selections processes, the fitness pressure increases with time, as population is expected to get closer to optimal

solutions. At the very end of the run, where solutions are assumed to be almost optimal, the probability of FS2 is very high, in order to further force an improvement in the results.

Regarding the number of offspring, producing more than two children for each couple is generally not recommended, because it could harm diversity by focusing on a small quantity of couples. In NSGAINN however, due to family selection processes, offspring have relatively little chances to be included in the population. Over the four individuals created by each couple, it is likely that less than two will be kept. On the other hand, producing four children per mate enables to explore more widely the opportunity of each couple. As far as family selection processes are concerned, it also enables to compare an offspring with five individuals (for FS2), instead of three if only two children were produced. This thus provides a fairly more representative idea of the non-dominance and makes the family sorting more efficient.

On the whole, NSGAINN is expected to present an improvement compared to NSGA-II by applying a stronger fitness pressure in the last part of the run. Thanks to the production of four children and to family selection processes, only fit-enough individuals are selected for the complete sorting of the population, and no time is wasted sorting dominated individuals. The shortcoming of this behaviour is that a lot more evaluations are required; this is however not a problem when GAINNN methodology is used.

4.2 Poly-objective Looped Algorithm using Genetics and an Uncompleted Extinction (PLAGUE)

4.2.1 General Description and Pseudo-code

A second and less classical algorithm, named PLAGUE, has been created for the current thesis and is specifically designed to be used with the current optimization methodology. PLAGUE stands for Poly-objective Looped Algorithm using Genetics and an Uncompleted Extinction. This algorithm takes its roots in many different MOEAs such as NSGA-II, SEAMO (Valenzuela 2002), or Micro-GA (Coello and Pulido 2001). PLAGUE is founded on the idea of cycles of evolution between geological eons, with progressive increases of the size and fitness of the population, followed by sudden drastic reductions of the population size. This reduction is here referred as "plague", named as the well-known disease.

The general behaviour of PLAGUE can be described with the pseudo-code described in Figure 17. It is divided into two phases: the expansion phase, and the « plague » selection.

BEGIN

I) INITIALIZE population with N random candidate solutions;

II) EVALUATE each candidate;

III) REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO

_ **Expansion phase:** While population size is inferior to five times the initial size (N)

- Generate offspring by recombination and mutation;
- Include offspring only if they are not dominated by their parents;
- Remove children-dominated parents from population.

End Loop

_ **Plague selection:** When population size is superior or equal to 5 times N, proceed to "Plague"

- Sort the population based on rank and crowding distance, select the 85%•N fittest individuals, and include them in the next population (85% for 2 objectives, or 80%•N for 3 objectives);
- For each objective, select the 5%•N best individuals regarding this specific objective over the entire population, no matter its rank, and include them in the next population;
- Take 5%•N individuals, randomly over the entire population, regardless of their rank, and include them in the next population.

End loop

IV) END DO

Figure 17: Pseudo-code of NSGAINN

4.2.2 Main Aspects of PLAGUE Behaviour

Expansion Phase

The expansion phase is dedicated to let the population grow until five times its initial size. It is important to note that PLAGUE uses an initial population size of 80 individuals, which is smaller than what most MOEAs use (typically 100 individuals). During that phase, the whole population is selected as parents (there is no tournament selection) and two offspring are produced at each mate. A family selection is then applied, based on the following rules:

- A child is included in the population only if it is not dominated by any member of its family.
- Any child included in the population can further be used as a parent for reproduction.
- If a parent is dominated by any member of the family, then it is removed from the population.

Plague

This second step, called *"plague"* is dedicated to reduce the population size by 80%. Individuals of the next generation are selected in a way that ensures fitness improvement, spreading of solutions, and diversity. The selection process produces a new population of size N (initial size) and is computed as follows for a two-objective problem[3]:

- Eighty five percent of next population is selected based on dominance and on crowding distance. This selection is based on the same non-dominated-and-crowding sorting than NSGA-II.
- For each objective, the 5% x N individuals presenting the lowest values for this specific objective are selected, regardless of the dominance or of the crowding distance.
- The last five percent of the population is randomly selected over the entire population, regardless of the rank or of the crowding-distance.

For the very last generation, instead of *plague* selection, a non-dominated sorting is applied over the entire population, and all non-dominated individuals are kept. This genders a final solution set of generally greater size than the initial population size, which compensates for the small size of PLAGUE's initial population compared to other MOEAs.

[3] For a three-objective problem, the first step selects 0.8xN individuals and the second step genders 0.15xN individuals.

4.2.3 Discussion

Although relatively simple, PLAGUE behaviour is expected to be very efficient. The family selection process applies a limited yet significant fitness pressure regarding the offspring production. The deletion of child-dominated parents, in turn, enables to improve efficiency by removing individuals which are close to, but worse than, another. This process is similar to the parental replacement used in SEAMO, claimed to be elitist (Valenzuela 2002). The population size does not necessarily increase (and may even decrease) after reproduction. The purpose is not to expand the population quickly, but to let the population improve significantly before proceeding to the next time-consuming sorting. The population at the end of the expansion phase may not be composed of only non-dominated individuals, but the overall fitness is likely to have improved and several search directions have been studied. At this step, the population is assumed to have reached a state where major improvements are unlikely, and only then the time-consuming sorting can be valuable.

Plague selection process is divided in three parts handling three different issues. The first portion is selected to improve the overall population fitness, the second portion is aimed at extending the solution front, and the last portion is dedicated to introduce diversity. Indeed, this portion of random individuals helps the spreading and the convergence, without deteriorating the overall efficiency of the algorithm. In the worst case, these random individuals will be worthless, and be very soon deleted by their children. Due to PLAGUE working principle, this will waste a minimum amount of time. In the best case,

these individuals will create a good diversity and produce individuals which, though dominated, will lead to a better convergence by adding new search directions. If there is a gap between a local and a global minimum, it is possible for this gap to be filled during the expansion phase by successive offspring. At the very end of the generation, only fit-enough will be selected, while transition-individuals will be discarded.

Regarding the number of evaluations, it is obvious that PLAGUE requires much more evaluations per generation than NSGA-II. Once again, PLAGUE has been created to be associated with GAINN methodology where evaluations are fast, and is not expected to be efficient in other situations.

4.3 Comparisons between NSGA-II, NSGAINN, and PLAGUE

4.3.1 Testing Methodology

Parameters

The comparison between the two developed algorithms and NSGA-II were based on a maximum run-time criterion. While the number of generations is much more frequently used as termination criterion, it was not possible to use it in this study, since it would have favored too much PLAGUE compared to the two other algorithms. For the purpose of comparison only, the maximum time is an acceptable criterion since the three MOEAs are based on a mostly similar code, and since all tests will be performed on a same

computer. Regarding algorithms parameters, NSGA-II parameters were kept as default for the three algorithms studied (except regarding the children-by-mate number in NSGAINN, and the population size in PLAGUE). These parameters are summarized in Table 3.

	NSGA-II	NSGAINN	PLAGUE
Population size	100		80
Crossover type	Simulated Binary Crossover		
Crossover probability	0.9		
Distributions indice for crossover	20		
Mutation type	Polynomial mutation		
Mutation probability	1/(number of variables)		
Distributions indice for mutation	20		
Termination criterion	Maximum time		

Table 3: Parameters used for comparison of NSGA-II, NSGAINN, and PLAGUE

Test functions

Test functions used for comparison are summarized in Table 4. They are taken from the two-objective ZDT suite, and the three-objective DTLZ suite. These functions are very commonly used in MOEA testing and are specifically designed to challenge MOEAs regarding global optimum and spreading of the solution front. The run-time associated with each function was designed to be long enough to enable for at least one MOEA to reach the Pareto front, while remaining short enough to display differences between MOEAs results. Numbers of variables and run-time associated to each function are also summarized in Table 4.

Function	Number of variables	Run-time
ZDT1, ZDT2, ZDT3	30	15 seconds
ZDT4	10	15 seconds
ZDT6	10	30 seconds
DTLZ1 to DTLZ7	10	120 seconds

Table 4: Summary of test functions and variables

The CPU runtime was calculated by MATLAB, with all tests performed on a same computer, in the absence of any other major activity. The computer used was equipped with a Genuine Intel(R) CPU T2300 @1.66GHz, 1GB of RAM, and Windows XP (SP2); MATLAB version used is 7.0 . Each test was performed five times for each function and for each algorithm studied.

Metric Used for Comparison

There is no common agreement regarding the metric(s) to use to assess the quality of a solution front. The metric used in this study was the dominated space (Zitzler, 1999), as illustrated in Figure 18. The main asset of this metric is that it is able to assess both convergence and spreading simultaneously. It is also able to deal with discontinuous fronts, and to compare two solutions sets even if their sizes are different. It is important to note that this metric is problem dependent, with a maximum reachable value function of the Pareto front's shape and of the space chosen to study dominance.

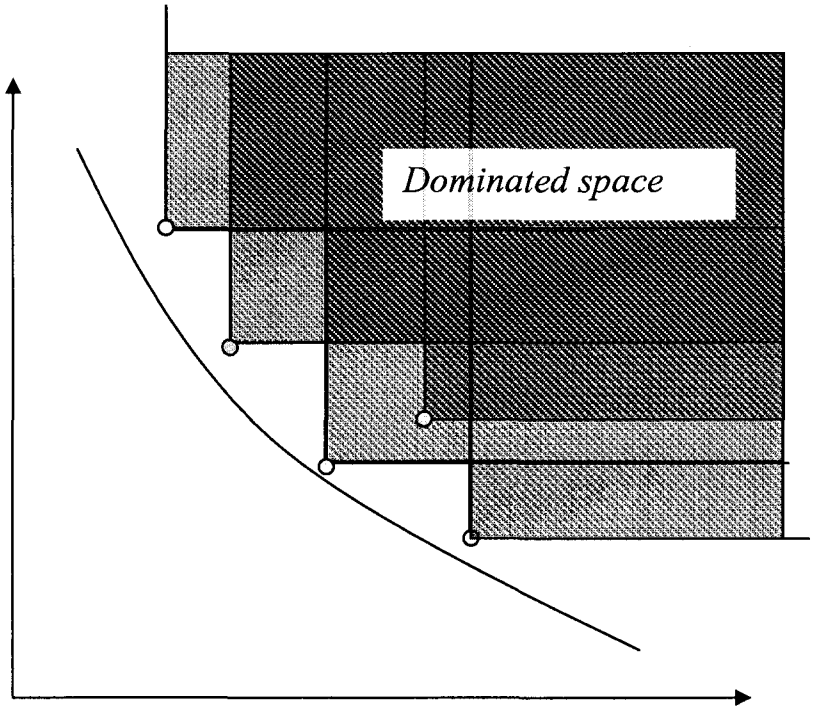


Figure 18: Illustration of a dominated space

In this thesis, the dominated space was calculated statistically, as proposed by Fieldsend et al (2003). The calculation can be described as follows: first a square (for two-objective problems) or a cube (for three-objective ones) is created, with bounds equal to the true Pareto front bounds. Then, a sample of 1 000 000 points is randomly taken inside this square (respectively cube), and each of these points is tested to see if it is dominated by the solution set. The dominated space is equal to the ratio of dominated individuals in the sample.

This method was tested by the author for reliability by calculating the dominated-space 50 times for a same population. Tests were done using the solution set of NSGA-II on ZDT6 (30 variables, 120 seconds); and the solution set of NSGA-II on DTLZ2 (30 variables, 240 seconds). The average errors were respectively 0.00123 (0.41%) for the two-objective problem and 0.0011 (0.29%) for the three-objective problem; variances were respectively 1,77E-07 and 1,9573E-07. This calculation method can therefore be considered as reliable.

4.3.2 Results

Average Dominated Space

Some solution sets, representative of the situations encountered, are illustrated in Figures 19 and 20. The average dominated-spaces over the five runs are summarized for each algorithm and for each function in Table 5. These results are also illustrated in Figure 21 (for two objective problems) and Figure 22 (for three-objective problems). Since the dominated space is, to some extent, dependent on the size of the solution set, dominated space is calculated two times for PLAGUE, once using the complete solution set, and then using only the best 100 individuals (noted PLAGUE(100)).

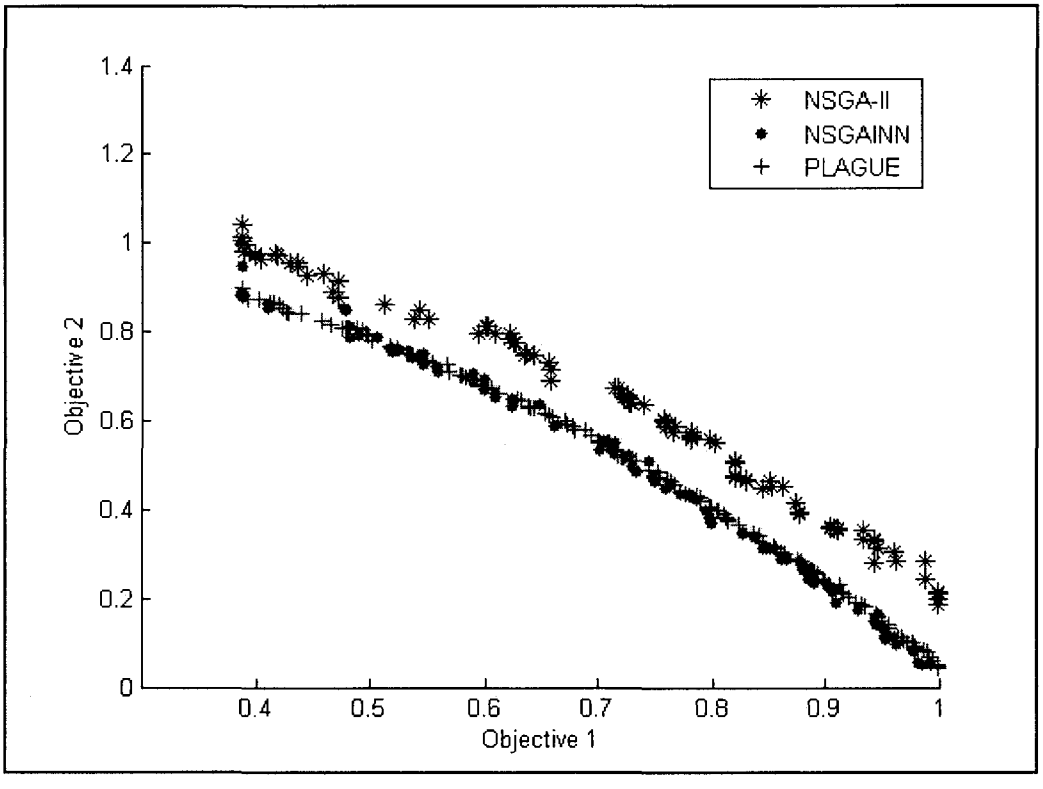


Figure 19: Solutions sets for ZDT6 function

	NSGA-II	NSGAINN	PLAGUE	PLAGUE (100)
ZDT1	0,626	0,615	0,656	0,653
ZDT2	0,000	0,066	0,193	0,192
ZDT3	0,477	0,485	0,511	0,510
ZDT4	0,000	0,135	0,343	0,342
ZDT6	0,219	0,305	0,291	0,290
DTLZ1	0,279	0,581	0,775	0,751
DTLZ2	0,372	0,376	0,410	0,373
DTLZ3	0,065	0,289	0,334	0,310
DTLZ4	0,303	0,382	0,418	0,383
DTLZ5	0,092	0,093	0,093	0,091
DTLZ6	0,195	0,219	0,196	0,193
DTLZ7	0,249	0,255	0,327	0,322

Table 5: Comparison of average dominated space

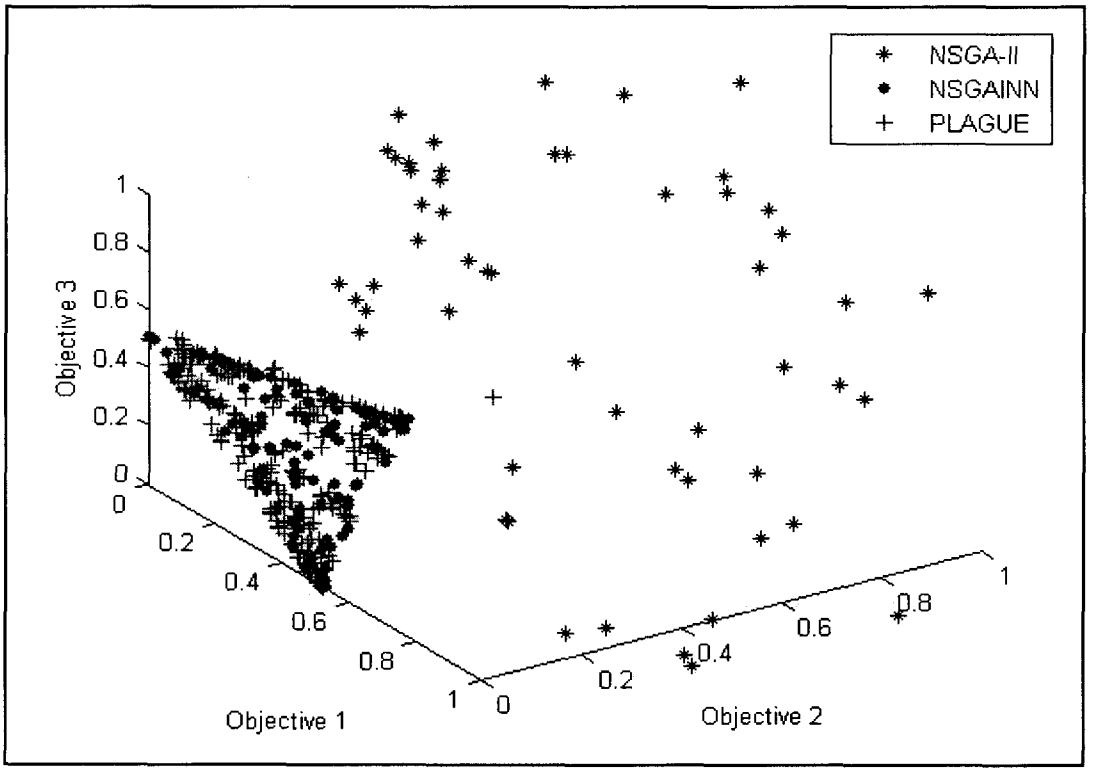


Figure 20: Solution sets for DTLZ1 function

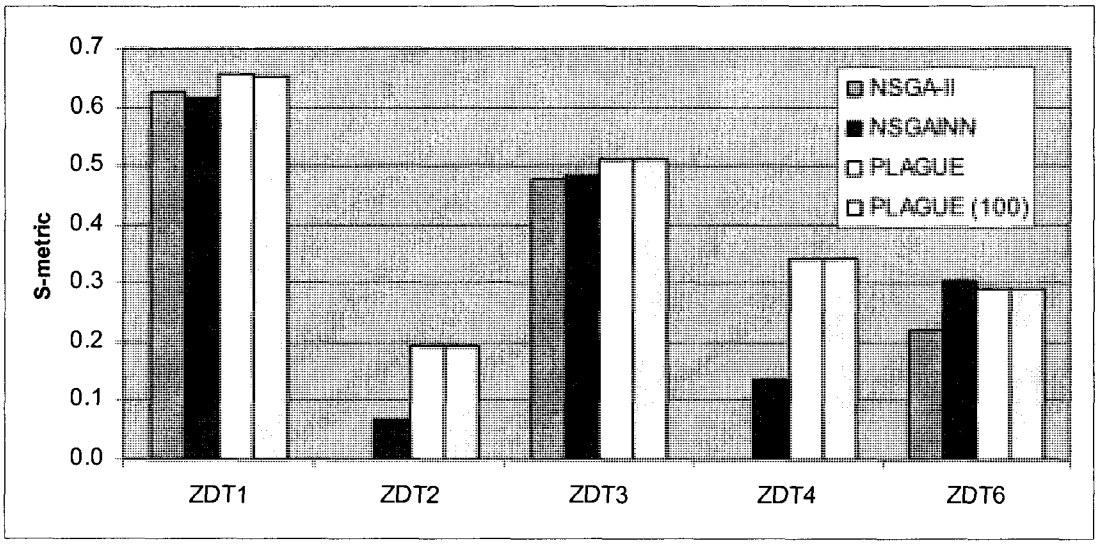


Figure 21: Comparison of average dominated space for ZDT functions

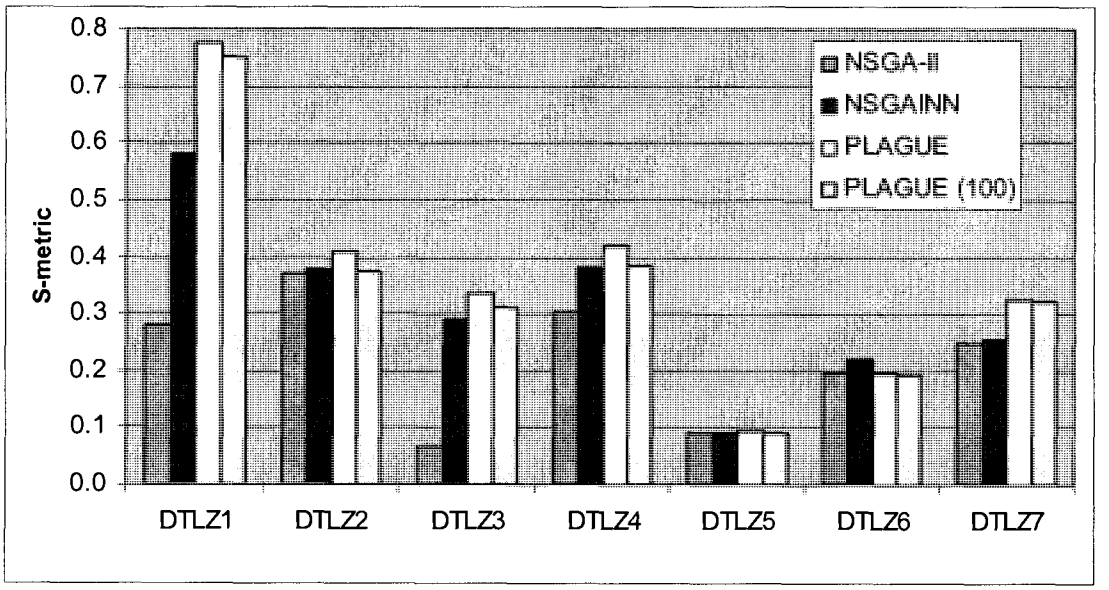


Figure 22: Comparison of average dominated space for DTLZ functions

As can be seen, both PLAGUE and GAINN performed better than NSGA-II for all test functions, except NSGAINN for ZDT1 function. The improvement ranged from 2% to more than 394% for NSGAINN, and from 0% to 416% for PLAGUE (the improvement was virtually infinite in cases NSGA-II dominated space is zero). PLAGUE was generally the most efficient algorithm, although NSGAINN got better results than PLAGUE for ZDT6 and DTLZ6. Regarding PLAGUE reduced to 100 individuals, it performed better than NSGA-II for all functions except for DTLZ4 and DTLZ5 (2% decrease).

Maximum Dominated Space

The maximum dominated spaces reached over the five runs are summarized for each algorithm and for each function in Table 6. Results are also illustrated in Figure 23 (for two-objective problems) and Figure 24 (for three-objective problems).

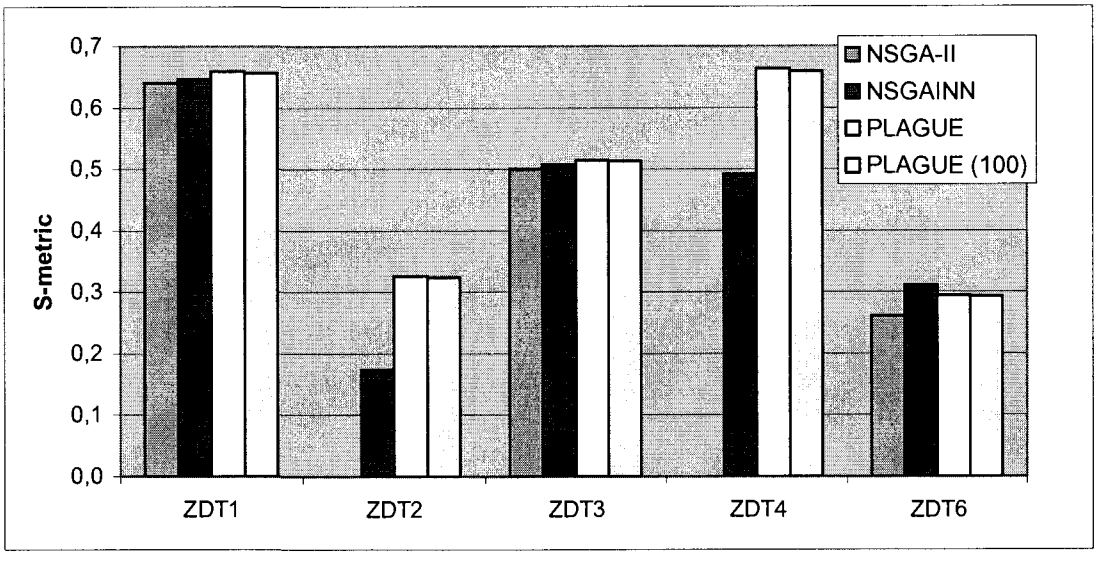


Figure 23: Comparison of maximum dominated space for ZDT series

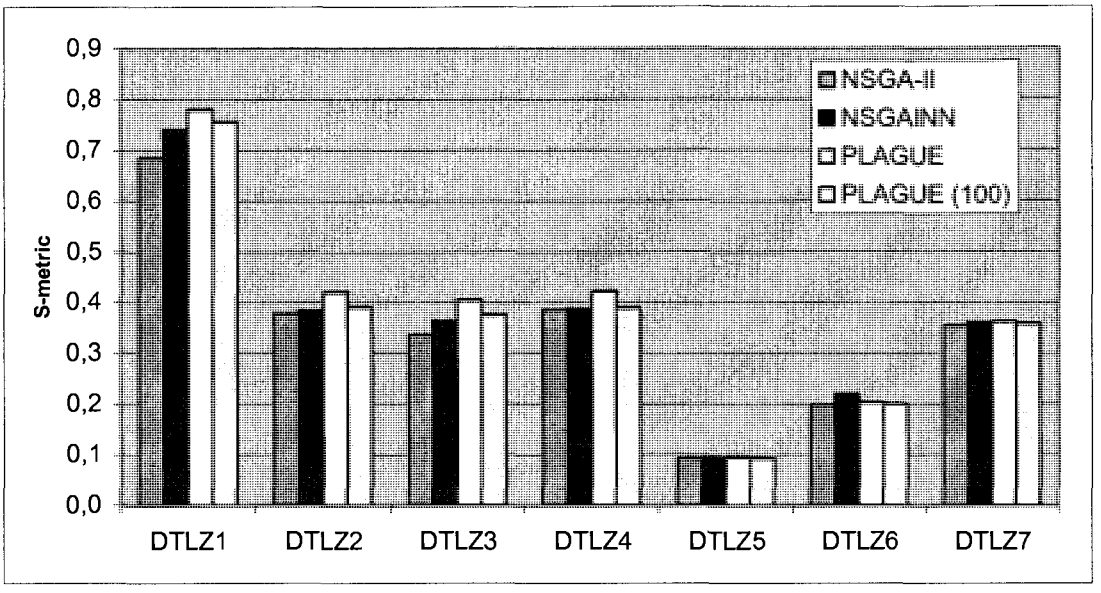


Figure 24: Comparison of maximum dominated space for DTLZ series

	NSGA-II	NSGAINN	PLAGUE	PLAGUE (100)
ZDT1	0,641	0,647	0,660	0,657
ZDT2	0,000	0,174	0,326	0,324
ZDT3	0,500	0,507	0,515	0,513
ZDT4	0,000	0,491	0,663	0,659
ZDT6	0,261	0,311	0,294	0,293
DTLZ1	0,685	0,739	0,780	0,755
DTLZ2	0,378	0,383	0,420	0,391
DTLZ3	0,335	0,363	0,404	0,375
DTLZ4	0,384	0,386	0,422	0,390
DTLZ5	0,093	0,093	0,094	0,092
DTLZ6	0,198	0,220	0,203	0,200
DTLZ7	0,354	0,360	0,362	0,358

Table 6: Maximum dominated space result for the three MOEAs

PLAGUE and NSGAINN got better results than NSGA-II for all functions except DTLZ2 for NSGAINN. PLAGUE was better than NSGAINN for most functions. Improvement in maximal results compared to NGSA-II ranged respectively from 1% to 19% for NSGAINN, and from 1% to 21% for PLAGUE. Once again, NSGAINN got better results than PLAGUE for ZDT6 and DTLZ6.

Number of Evaluations

Average numbers of evaluation for each algorithm and for each function are summarized in Table 7 (average of 5 runs), and are illustrated in Figure 25 (for two-objective functions) and Figure 26 (for three-objective functions).

	NSGA-II	NSGAINN	PLAGUE
ZTL1	7600	6796	14870
ZTL2	4406	4451	27306
ZTL3	7830	7222	15830
ZTL4	5520	6546	21097
ZTL6	13175	18160	35036
DTLZ1	58082	79344	163760
DTLZ2	81026	105550	100640
DTLZ3	60935	80654	172070
DTLZ4	71156	104120	99269
DTLZ5	79281	103610	123590
DTLZ6	79737	112760	107530
DTLZ7	71780	98702	100279

Table 7: Number of evaluations for the three MOEAs

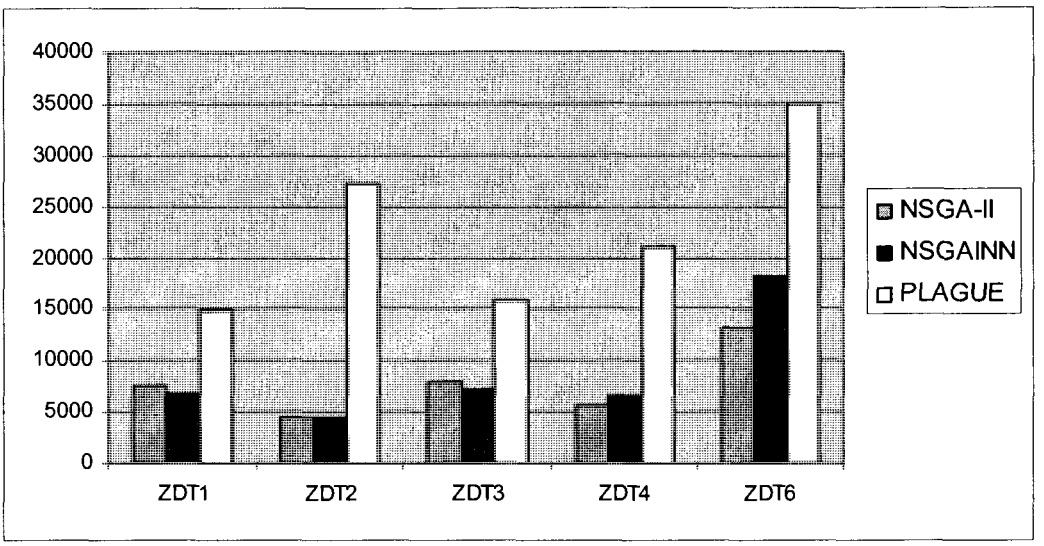


Figure 25: Comparison of number of evaluation for ZDT suite

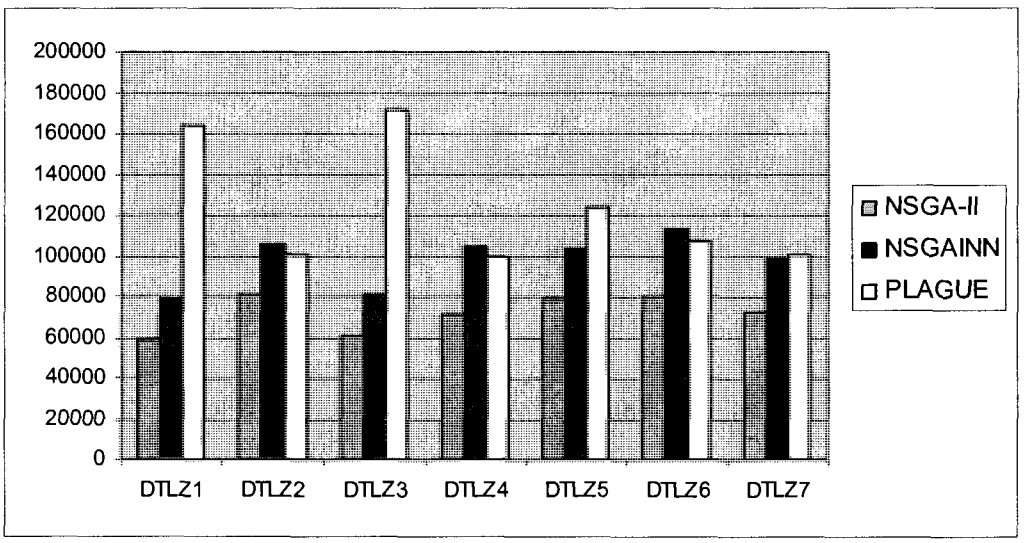


Figure 26: Comparison of number of evaluation for DTLZ suite

As expected, both PLAGUE and NSGAINN generally required more evaluations than NSGA-II. PLAGUE required an average of 143% more evaluations than NSGA-II, and up to 520% more for ZDT2. NSGAINN required an average of 25% more, and surprisingly sometimes required fewer evaluations than NSGA-II (11% less for ZDT1 and 8% less for ZDT3).

4.3.3 Discussion

The conclusion we can draw from these results is first that both developed algorithms are generally more efficient than NSGA-II, regarding both average and maximum results. The improvement in result is in most cases significantly greater than the uncertainty of the metric and the author is therefore confident in his results. Comparing the two developed algorithms, PLAGUE is generally the most efficient, although NSGAINN sometimes provides better average and maximum results (for ZDT6 and DTLZ6).

There is little difference in results between PLAGUE and PLAGUE(100) for two-objective functions. The difference can however become significant for the three-objective ones (up to 8% difference). PLAGUE solution set was reduced to 100 to be compared fairly to other algorithms, but the point is PLAGUE actually provides more solutions, which is one of its assets. Therefore, to author's opinion, the dominated space of PLAGUE's whole solution set can be considered as representative, when studying the overall efficiency of the MOEA. Nevertheless, even reduced to 100 individuals, PLAGUE remains more efficient than NSGA-II in almost all cases, and is the best of the

three MOEAs for two-objective functions. For three-objective function, considering PLAGUE population reduced to 100 individuals, NSGAINN become the most efficient algorithm for half of the functions tested.

The fact that developed algorithms require more evaluations than NGSA-II is both logical and expected. For GAINN methodology, due to ANN fast calculations, the number of evaluations is not a limiting parameter. If one uses PLAGUE outside of GAINN however, directly combined with TRNSYS for instance, the time requirement could double or even be multiplied by six compared to NSGA-II. In such cases, PLAGUE could become less useful.

Regarding NSGAINN, there is also an increase in number of evaluation compared to NSGA-II, due to the large amount of children evaluated and rejected through family sorting. This increase, 25%, is significant, but may be compensated by the better results it provides. Even if it was created for GAINN, NSGAINN could therefore be efficient in some problems where the computational time associated with evaluation is relatively short (milliseconds or lower).

For ZDT1 and ZDT3 functions, NSGAINN surprisingly requires fewer evaluations than NSGA-II. This can be explained by the fact that NSGA-II does not require family sorting and therefore runs more generation than NSGAINN. Since NGSAINN finishes with fewer generations, fewer evaluations were required. It is interesting to note that in those cases, the family sorting was not efficient, and NSGAINN did not provide better results than NSGA-II (and was even worse for ZDT1).

4.4 Conclusion

According to the tests performed, both developed MOEAs are more efficient than NSGA-II, as long as they are used inside GAINN methodology or for problems where the evaluation function is extremely fast. In cases where evaluations are time consuming, both NSGAINN and PLAGUE are expected to perform relatively bad; for evaluation-times in the order of milliseconds, NSGAINN may keep a good performance though. As long as used inside GAINN, the two developed algorithms are a success, noting the following.

All tests have been performed on a runtime basis. Although to author's opinion it is the fairest methodology in this case, this termination criterion is rarely used, and it suffers from being greatly influenced by the quality of the programming and by the language chosen; maximal generation is often preferred. Actually, in all tests performed, both NSGAINN and PLAGUE had fewer generations than NSGA-II. Therefore, the improvement shown in this thesis would be even larger if maximal generation was used as termination criterion. The methodology chosen is conservative, and improvement does not rise from criterion used.

Another crucial factor, as in any MOEA testing, is the parameters chosen, which greatly influences results. Different reproduction parameters especially, including crossover and mutation types, probabilities, and distributions indices, would have led to different results. As explained before, the author kept the same parameters as in NSGA-II original

paper, and expects to be conservative in his results. In that sense, use of other reproduction parameters, more specifically chosen for NSGAINN or PLAGUE, could lead to an even better improvement compared to NSGA-II. This should be studied in future works using developed MOEAs.

Finally, and more generally, one could wonder how practical the improvement is. Indeed, NSGA-II can almost always reach the optimal Pareto front, providing a sufficiently long runtime. One could wonder how few seconds can have an impact on the overall methodology. Indeed, the impact of the developed MOEAs may be little in cases where GAINN is used in combination with building simulation software, since the total computation time will be in the order of hours or days. However, if one uses GAINN for online optimization (which is a promising yet unexploited application of the methodology), improvement can become significant, enabling faster optimizations, and therefore faster reactions of the system.

CHAPTER 5:

MAIN CASE STUDY: OPTIMIZATION OF A RESIDENTIAL BUILDING

5.1 Description of the Optimization Problem

5.1.1 Overview of the Optimization

As written in the introduction, a sustainable building should have the smallest environmental impact, while remaining comfortable for the occupants. The energy consumption and thermal comfort should specifically be studied. In many situations, the whole problem is to find a *trade-off* between comfort and energy consumption. In residential buildings, this trade-off is very dependent on how much the occupant is willing to maintain or lower his thermal comfort in order to save energy. This decision is very personal and cannot be predicted or imposed; however, a tool should be provided to enable a global view of the range of possibilities. In that sense, a multiobjective optimization, optimizing *both* thermal comfort and the energy consumption, and based on no assumption regarding the occupant/building owner environmental awareness, becomes a very interesting tool.

The optimization undertaken in this study is aimed at exploring the trade-offs between thermal comfort and the energy consumption for a typical residential building. The optimization will be based on a simulated occupancy schedule, temperature set points and heating and cooling schedule throughout the day. While some multiobjective optimizations of HVAC set points have already been studied (Wright and Loosemore, 2002; Nassif et al. 2003), the current study will include additional variables related to passive solar design such as windows sizes and thermal mass.

The base building of this study is one of the two (identical) residential houses of the NRC Twin House Project. This building was chosen because of the quality of its simulated occupancy, and because of its continuous monitoring of interior as well as exterior conditions. The house is a typical 2-storey wood-frame house, with 210 m^2 of living area. It is built to meet the R-2000 standard with a wall RSI-value of 3.5, and a roof RSI-value of 8.6. Windows are composed of low-e Argon filled double glazing units, and air tightness of the house has been tested at 1.5 air change per hour at 50 Pa. Regarding the HVAC system, the house is equipped with a high efficiency condensing gas furnace (80% steady-state efficiency and a rated output of 67,500 Btu/h,), a 12 SEER air-conditioner with a 2 ton capacity, and a high efficiency (84%) heat recovery ventilator. Plans of the house are shown in Figure 27 and a more detailed description of the house characteristics and testing can be found in Swinton (2003).

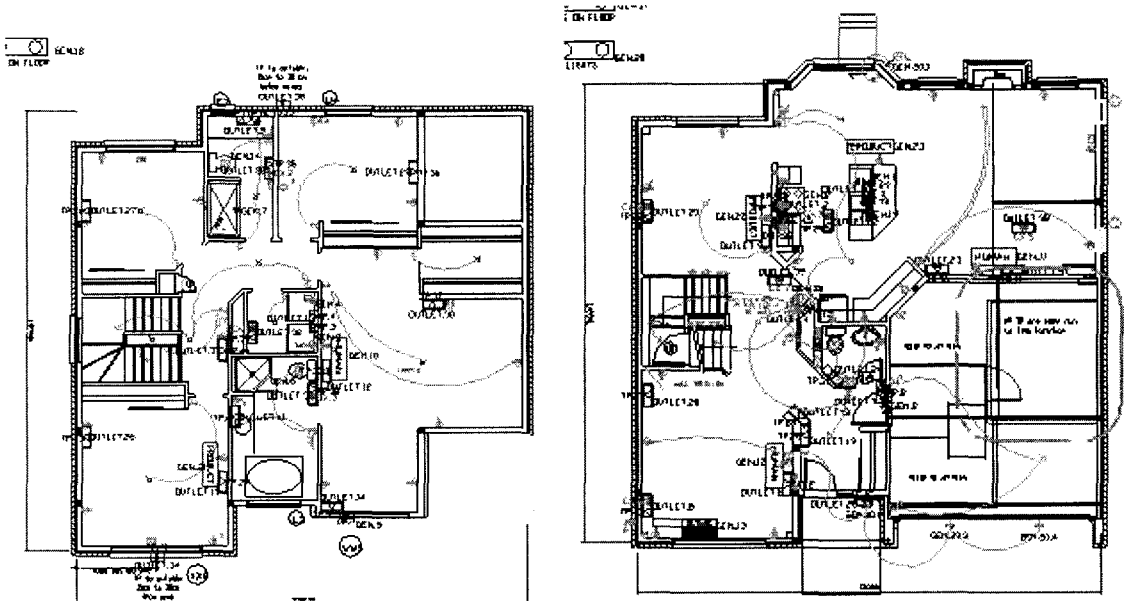


Figure 27: Layouts of the first and second floors of the building studied

5.1.2 Optimization Methodology

The optimization involves complex phenomena and complex relations between the variables studied. Moreover, the simulation of the thermostat programme requires a very small simulation time step. In that situation, the use of GAINN methodology is justified, in order to provide an efficient optimization of the building while maintaining a reasonable computational time. The methodology is described in the following Figure:

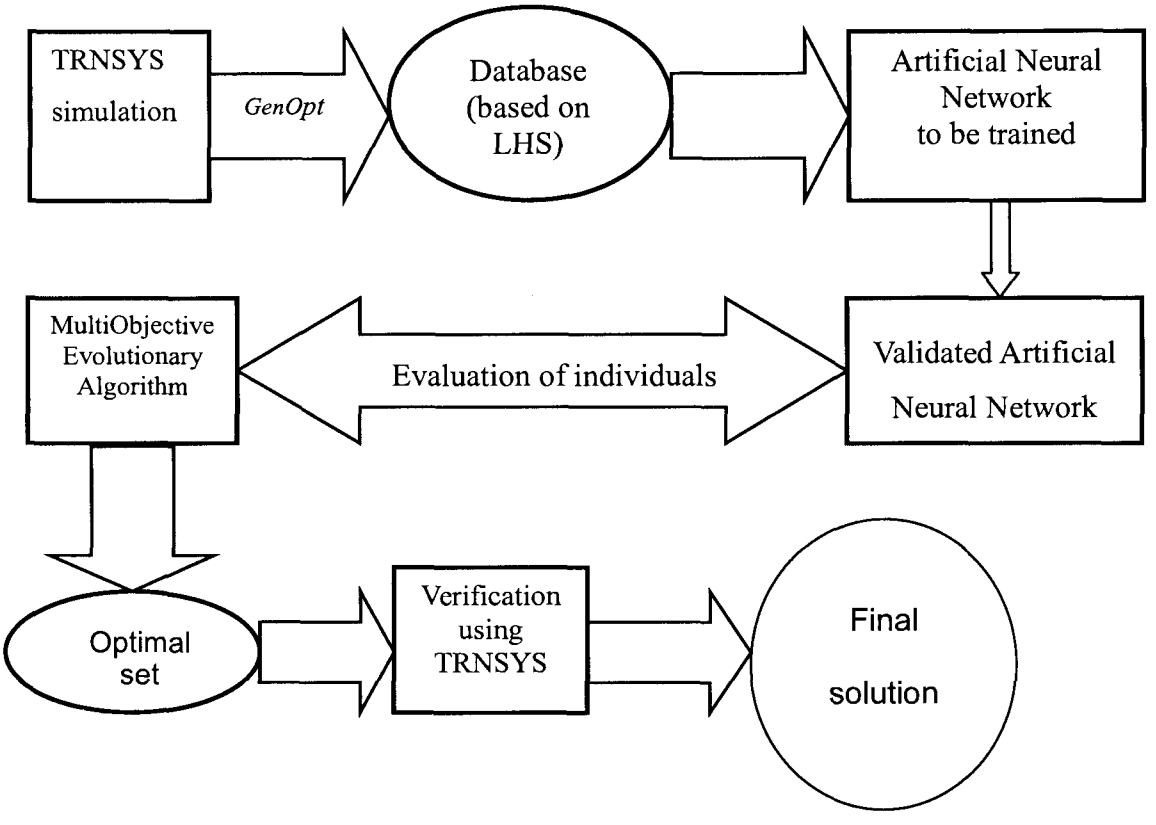


Figure 28: Workflow of GAINN methodology

1. Building simulation

The building will be simulated using TRNSYS software. TRNSYS is a TRaNsient SYstems Simulation program that includes a modular structure, a graphical interface, and a large library of components. TRNSYS has been chosen for this study because it is recognized as one of the best building simulation software, along with EnergyPlus and ESP-r (DOE, 2008).

2. *Parametric runs*

In order to create a database for ANN training, parametric runs have to be computed. In order to automate TRNSYS runs, GenOpt (Wetter, 2001) will be used. When associated with TRNSYS, GenOpt can automatically generate building (.bui) and deck (.dck) files based on chosen templates, run TRNSYS with those files, save results, and restart again. Thanks to GenOpt automation, there is no need to write all deck and building files by hand, and therefore a significant amount of time is saved. More importantly, the risk of mistakes while writing the files is significantly lowered.

GenOpt is provided with several parametric and (single-objective) optimization algorithms. In this study, TRNSYS parametric simulations are to be run with parameters taken from columns of a matrix. Quite surprisingly, this behaviour was not readily available in GenOpt algorithms, and had to be programmed. The algorithm used is a slightly modified version of GenOpt's "parametric run" algorithm. The complete code of this algorithm, written in java, can be found in Appendix C.

3. *Design of experiment*

In order to reduce the size of the training database while keeping the sample representative, Latin Hypercube Sampling (LHS) is used. LHS is one of the most common methods used to generate a small and representative sample of a population, for a specified number and ranges of variables. Studies have shown that using LHS, a

number of cases greater than twice the number of parameters is sufficient to correctly sample the search space (Mackay, 1988).

The principle of LHS is simple and can be illustrated as in Figure 29. For a 2-variable problem with a search space conceptualized as a square, the LHS method takes one and only one point per each column and per each row. The complete sample is therefore relatively little but remains representative of the whole search space. In this study, LHS will be computed in MATLAB, using the Model-Based-Calibration Toolbox.

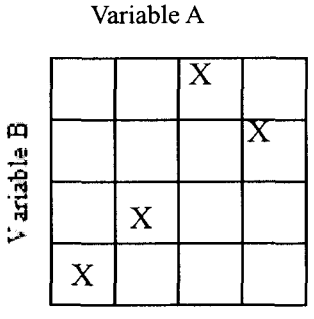


Figure 29: Illustration of LHS for a 2-variable problem

4. Artificial Neural Network

The Artificial Neural Network will be computed using built-in ANN-toolbox of MATLAB. It will be trained using a first sample from LHS, and checked for validation using a second and smaller sample.

5. Multiobjective optimization

Once trained and validated, the ANN will be used as the evaluation function inside the three MOEAs described previously. Details about the optimization process will be described in a next section.

6. Verification of the results

Finally, once optimization completed, some solutions will be checked using TRNSYS, to ensure that the ANN is accurate enough in the vicinity of the optimal solutions. This step, sometimes ignored in previous studies (Conraud, 2008), is crucial to assess the overall efficiency of the methodology.

5.1.3 Metrics used for Objective Evaluations

Energy consumption

The energy consumption of the building will be directly assessed by TRNSYS. The total energy consumption, E_{tot} is composed of:

- Heating/furnace consumption E_{heat};
- Cooling consumption (including dehumidifier consumption) E_{cool};
- Fan consumption E_{fan} ;

Energy consumptions for domestic hot water and appliances were not included because they are not affected by the parameters studied. Lighting consumption is somewhat influenced by windows sizes and could have been included. The author however makes the assumption that variations in lighting consumption can be neglected, especially if compact fluorescent lamps are used. Finally, the energy consumption for humidifying the air was assumed to be negligible.

Thermal comfort

The metric used to assess thermal comfort is the Predicted Mean Vote (PMV), based on Fanger's model (Fanger, 2000). PMV is representative of what a large population would think of a thermal environment, and is used to assess thermal comfort in standards such as ASHRAE 55 (ASHRAE, 2004). It ranges from -3 (too cold) to +3 (too warm), and a PMV value of zero is expected to provide the lowest Percent of People Dissatisfied (PPD) among a population. An absolute PMV value of 0.5 is generally recognized as the limit of the comfort zone.

In order to optimize thermal comfort, two parameters are used in this study:

- Average absolute PMV $|PMV_{house}|_{avg}$: averaged over the whole year and over all occupied zones.
- Number of hours with $|PMV_{house}| > 0.5$ N_{dis}: representing the cumulative time with discomfort over the whole year. This metric will be used as a penalty term inside optimization.

In this study, PMV values are calculated by TRNSYS for each zone, using a constant metabolic rate of 1 met, a constant air velocity of 0.1 m/s, and a clothing factor equal to 0.5 clo in summer, 0.9 clo in winter, and 0.8 during the rest of the year. The PMV values will only be taken into account if at least one occupant is present inside the house. For night periods in the heating season, thermal comfort is ignored and a constant set point of 18°C is maintained. In the cooling season, night thermal comfort is also ignored; the night cooling set point is however kept equal to the one used during occupancy, which is expected to provide an acceptable comfort. Local discomfort such as draft or temperature asymmetry are not included in this study, because they are either not expected to appear (the house is airtight and well-insulated), or too complicated to assess in a residential environment.

5.1.4 Study Parameters

The study parameters are the ones expected to most significantly influence both the thermal comfort and the energy consumption. Since the base building is already well insulated, and since the insulation affects the energy consumption but has little impact on thermal comfort, the author decided to exclude insulation from study parameters. Regarding window type, the windows installed in the base building (low-e Argon filled double glazing) are generally recommended for passive solar design. Windows type has thus also been excluded from the present study. Considering the base building and the optimization problem studied, only two parameters related to the building envelope have been kept to carry out the optimization: windows sizes, and thermal mass.

Regarding the HVAC system, the focus has been the program and schedule, with variables influencing both the thermal comfort and the energy consumptions. Variables are closely related to the occupancy schedule. This schedule, based on the Twin House Project occupancy schedule, can be summarized as follows. Apart from the night, the house is considered to be occupied between 6:45AM and 8:00AM, and between 5:30PM and 11:00PM, 5 days per week (see Figure 30). During the week-end, it is occupied from to 6:45AM to 11:00PM. Four parameters have been selected for the HVAC system: the temperature set points, the relative humidity set points, the ventilation rates, and the system delays.

Windows sizes

Sizes of the windows are predominant in passive solar design, since it is the main parameter influencing the amount of solar radiation entering the house. Sizes should be carefully thought to enable a high amount of incoming solar radiation, but to avoid overheating the house in summer and to keep thermal losses to a minimum. Structural, privacy and aesthetics issues should also been taken into account. Based on these considerations, the window sizes are varied between 20% and 60% of the corresponding wall areas. For architectural reasons, only north and south windows is studied, for a total of 5 windows.

Thermal mass:

In passive solar design, the thermal mass of a building is a predominant parameter because it influences the amount of heat a building can store. In this study, the thermal mass of the house is changed by changing the thickness of the concrete slab in the interior floors. The thickness of the slab varies from 5 centimetres to 25 centimetres. Structural issues are beyond the scope of this study.

Temperature set points:

Heating and cooling temperature set points are varied according to ASHRAE guidelines (2004). The heating set point temperature is varied between 20°C to 25°C, and the cooling set point temperature is varied between 23 °C to 27 °C. It should be noted that at night or when the house is not occupied, the heating set point is fixed at 18°C. Also, cooling is turned off if no one is present inside the house.

Relative humidity set points:

Relative humidity has a significant influence on the thermal comfort. In this study, three humidity set points are optimized for respectively the summer, the winter, and the mild seasons. The winter season is assumed to be between November 1st and March 1st, the summer season between July 1st and September 1st, and the mild season the rest of the year. These seasons are based on the outside temperature profile in Ottawa (WeatherOffice 2008). Based ASHRAE guidelines, relative humidity set points are varied between 30% to 60% (ASHRAE, 2004).

Thermostat delays

A delay takes place between the moment heating is set on, and when the house reaches the set point temperature. A delay also exists between the moment heating is turned off and when thermal discomfort appears. In order to optimize the energy consumption based on the thermal mass of the house, starting and stopping delays are introduced in the thermostat schedule, as illustrated in Figure 30. The starting delay is defined as the delay between the time the set point is switched from setback to regular set point temperature and the time occupants enter the house, before occupancy. It varies between 0 and 30 minutes. The stopping delay is the delay between the time the set point is switched from regular set point to setback temperature and the time occupants leave the house or go to bed. It varies between 0 and 60 minutes. Different delays are studied for the three seasons previously described.

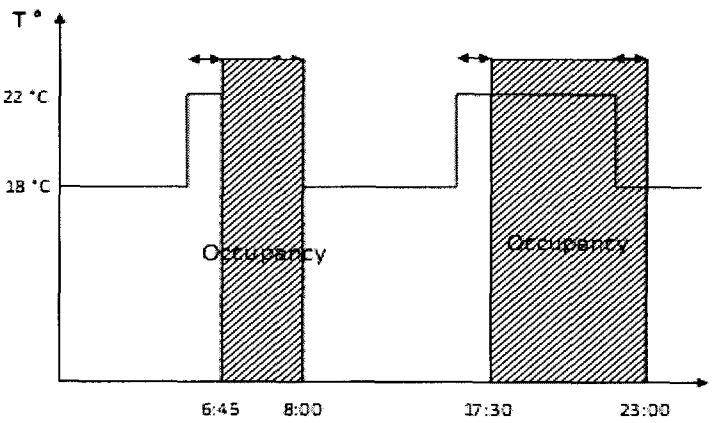


Figure 30: Illustration of the HVAC schedule

Ventilation rates

Finally, ventilation rates are studied, conjointly with the delays previously described. The amount of outdoor air brought in by the heat recovery ventilator (0.0307 m^3/s (65 CFM)) remains constant, maintaining an acceptable indoor air quality inside the house. The three ventilation rates corresponding to respectively continuous recirculation, heating mode, and cooling mode, are varied from 0.118 m^3/s to 0.708 m^3/s (250 CFM to 1500 CFM). These values correspond respectively to a 20%/80% ratio of outdoor/indoor air, and to the maximum ventilation rate in the base building.

In summary, the following table shows the ranges of the optimization variables:

Variable	Name	Lower bound	Upper bound	Unit
1st Floor north window	WF1N	4.76	14.30	[m2]
1st Floor south window	WF1S	2.20	6.60	[m2]
2nd Floor north window	WF2N	4.06	12.18	[m2]
2nd Floor southwest window	WF2S2	1.38	4.14	[m2]
2nd Floor southeast window	WF2S1	2.08	6.25	[m2]
Thickness of concrete in interior	TCK	5	25	[cm]
Heating set point	HSP	20	25	[°C]
Cooling set point	CSP	23	27	[°C]
RH set point (x3)	RH*	30	60	[%RH]
Starting delay (x3)	SD*	0	30	[min]
Stopping delay (x3)	FD*	0	60	[min]
Ventilation rate (x3)	VR**	0.118	0.0708	[m^3/s]

* : *S*, *W*, or *Mid* is added to the name of the variable if it corresponds to respectively summer, winter, or middle season.

** : VRH, VRC, and VRR for the ventilation rates in respectively heating, cooling and recirculation modes.

Table 8: Ranges of variables used for optimization

5.2 Building Simulation

5.2.1 TRNSYS Simulation

Overview of the model

The building model was developed in TRNSYS, and special care was taken for wall insulations, occupancies and appliances schedules, and HVAC equipment. A schematic view of the model is shown in Figure 31; Table 9 presents a list of the components used in the model.

Type	Description	Name
2	Differential controller	Heating Control
2	Differential controller	Cooling Control
9	Data Reader For Generic Data Files	Temperature and humidity
9	Data Reader For Generic Data Files	Irradiance
14	Internal gains schedule	Various
16	Sun radiation estimator	Type 16a
25	Printer to output file	System printer
33	Psychometrics	Psychometrics
56	Multi-zone building	Twin House
65	Online graphical plotter	Results
69	Sky temperature estimator	Sky temp
121	Furnace	Furnace
501	Soil Temperature Profile	Ground temperature
648	Air Mixing Valve	Flow mixer in/out
696b	Simplified Air Conditioning Device	Cooling system
760	Sensible Air to Air Heat Recovery	Heat exchanger

Table 9: List of components used in the TRNSYS model

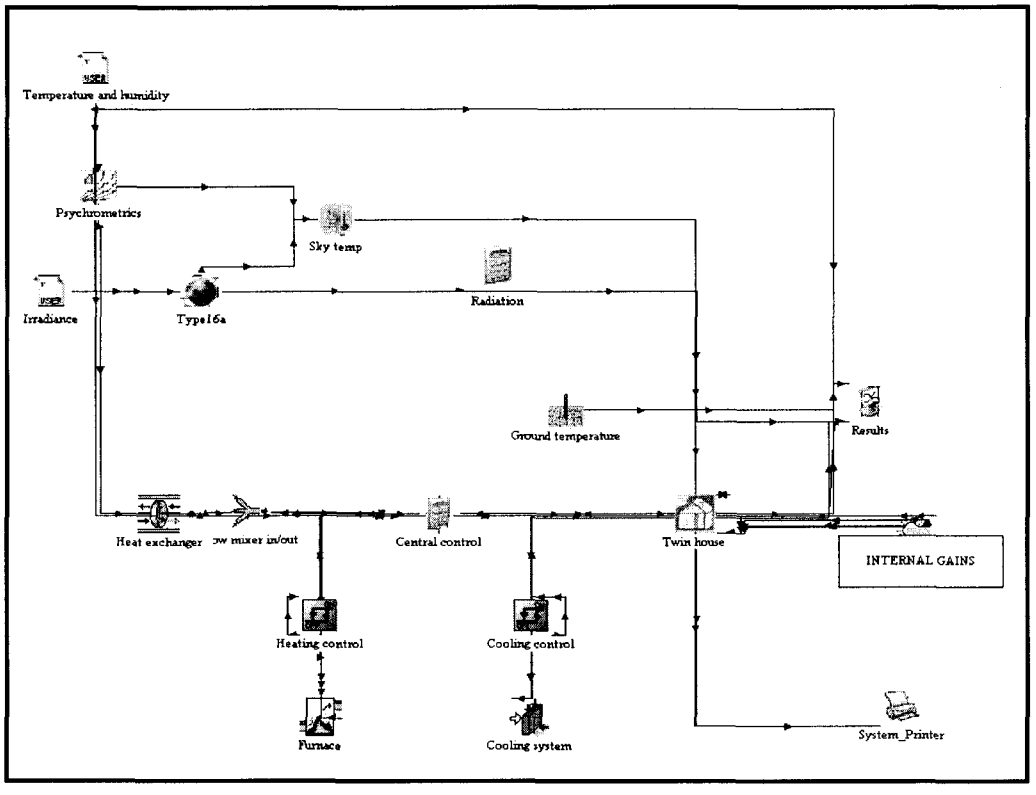


Figure 31: TRNSYS model view

The type-56 multi-zone building is a reproduction of the reference house (Figure 32). The building model is divided into 7 zones: first floor (open space), second floor north, second floor master bedroom, second floor south bedroom, garage, basement, and attic (Table 10). The first four zones are heated and cooled. The basement is partially heated and cooled (simulated with a low ventilation rate), since it is not supposed to be occupied. In the actual building, the so-called garage is in reality a control room, with its own ventilation system; in TRNSYS simulation, garage is not heated nor cooled, but its temperature is artificially set to 21°C all year long. Apart from HVAC continuous recirculation, no air change has been simulated between adjacent rooms.

Zone	Description	Floor area	Volume
First floor	Living room, kitchen, dining room	97.3	269.5
Second floor north	Various rooms	54.3	143.6
Second floor master bedroom	Master bedroom with bathroom	43	113.7
Second floor south bedroom	Bedroom	18.2	48.2
Basement	Non-occupied basement	97.3	228.7
Garage	Control room, not studied	47	95.2
Attic	Attic	n. a.	40

Table 10: Description of zones in type 56

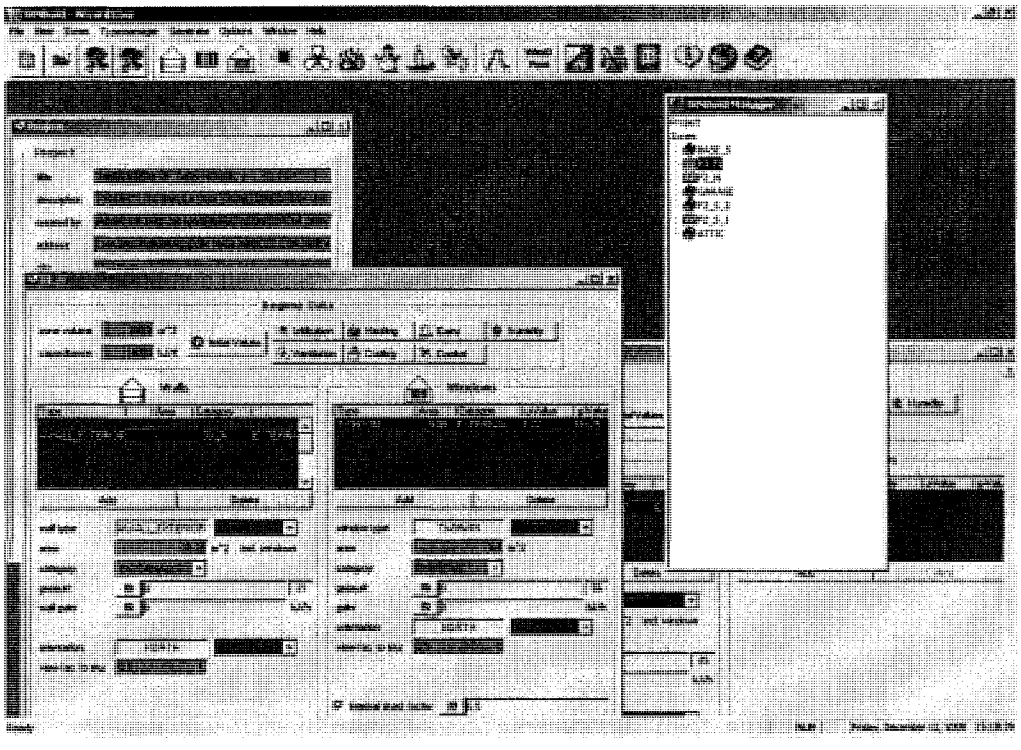


Figure 32: Screenshot of the type 56 model

Finally, a control strategy has been implemented for the venitian blinds, based on the following rules:

- Blinds are always closed at night.
- If at least one occupant is present inside the house, a blind is closed if the solar radiation on the corresponding window is superior to 432kJ/hr.m^2 (default value used for blind control in TRNSYS).
- If no one is present inside the house, blinds are always closed in cooling season, and are always open the rest of the year.

This simple strategy is expected to provide an energy efficient and realistic control of blind opening and closing.

5.2.2 Validation of the Model

In order to validate the TRNSYS model, simulation results have been compared with measured data. The TRNSYS model was run using the base building parameters described in Swinton (2001), with a 2 minutes time step, and using exterior measured data as an input. The measured data, provided by NRC, included exterior and interior temperatures and humidity, solar radiation, and energy consumptions, for the months of January and August 2003.

Heating consumption

Figure 33 displays the daily heating consumption for the month of January. As can be seen, the simulated results are reasonably close to measured data. The average absolute relative error between simulated and measured heating daily consumptions is 12.2%. Regarding the monthly total heating consumption for the month of January, the difference is 3.4% with respectively 4,329 kWh for measured consumption and 4,172kWh for simulated consumption.

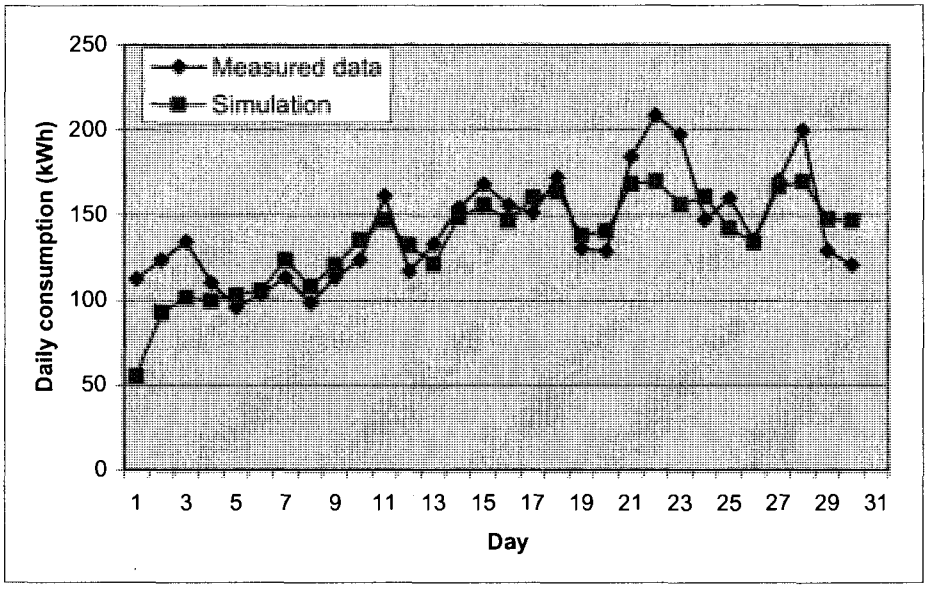


Figure 33: Simulated and measured daily heating consumption

Cooling consumption

Figure 34 displays the daily cooling consumption for the month of August. Due to technical problems, no data was measured between day 15 and day 18. These days have therefore been ignored. Once again, we can see a relatively good agreement between the

simulated and the measured data. Ignoring days where no data was available, the average absolute relative error in daily cooling consumption is 30%, which is significant. The relative error regarding the monthly cooling consumption is however 3.4%, which is acceptable (respectively 407 kWh for them measured consumption and 420 kWh for simulated consumption).

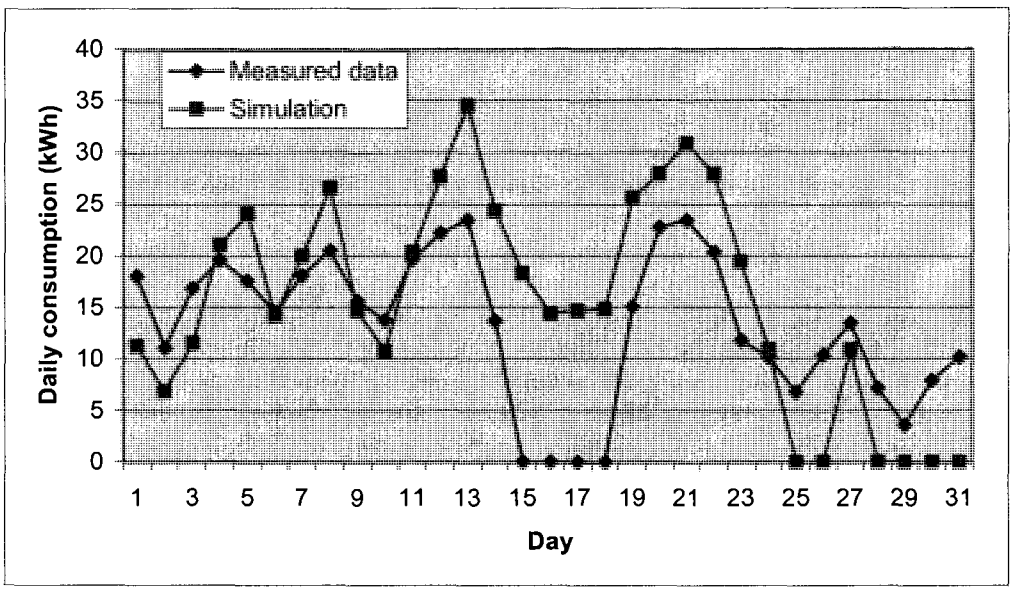


Figure 34: Simulated and measured daily cooling consumption

Fan consumption

Finally, simulated fan consumptions are in good agreement with the measured data. The simulated fan consumption for the month of January was 4.5% lower than the measured consumption (315 kWh and 330 kWh respectively). The simulated fan consumption for the month of August was 10% lower than the measured consumption (250 kWh and 278 kWh respectively).

5.2.3 Discussion

There is an acceptable agreement between measured and simulated heating and cooling consumptions. Several points should nonetheless be noted. First, the thermostat temperature used in the simulation is the average between the temperatures of the first floor and second floor North zone. In the actual building, the thermostat is situated in the corridor of the north of second floor, but close to the stairway. The temperature at this central position is influenced by first floor temperature and possibly by sun radiation, and variations are therefore likely to appear. Setting the monitored temperature as the average of the two-zones was found to give the best results overall. The second point to be noted is the relative uncertainty regarding basement temperature and ventilation distribution throughout the house. Air distribution was designed to keep a similar temperature in each zone all year long, and a basement temperature of around 18-19 °C in winter. Assumptions regarding air distribution were found by trial-and-error. On the whole, the TRNSYS simulation can be validated since it shows reasonably small differences between measured and simulated energy consumptions.

5.3 Artificial Neural Network Approximation

5.3.1 Parametric Runs

A sample of 450 cases was used for ANN training. This sample was created by Latin Hypercube Sampling, based on the variables and ranges previously described (Table 8). All the cases have then been simulated with TRNSYS, thanks to GenOpt automation.

Simulations were performed with a 2-minutes time step, and all simulations were preceded by a pre-simulation of 30 days. Due to the very small time step, the total simulation time of the 450 cases took around three weeks (using the same computer as the one described in Chapter 4).

5.3.2 Artificial Neural Network Training

The artificial neural network was composed of one input layer representing the 20 variables described before, one hidden layer composed of 20 neurons, and one output layer composed of the three energy consumptions and the two thermal-comfort variables. The number of neurons in the hidden layer was found by trial-and-error (see discussion below). Transfer functions used are hyperbolic tangent sigmoid functions in the initial and hidden layers, and linear functions in the output layer. The method used for the ANN training is the back-propagation, associated with Levenberg-Marquardt and Bayesian regularization algorithms. All inputs and outputs were scaled to the [-1, 1] range prior to training, as recommended in MATLAB (2006), to enable a better efficiency.

The ANN was trained with 450 cases. The training was considered to have reached convergence if both the sum of squared error (SSE) and the sum of squared weights (SSW) stabilized over certain iterations (as shown in Figure 35). The ANN training reached this goal after 516 epochs, with a final SSE of 1.16. Regression correlation coefficients between the network outputs and the corresponding TRNSYS simulation outputs were found very close to 1 for the five outputs studied, demonstrating a very

good correlation between outputs and target values. Figure 36 illustrates the regression for cooling consumption.

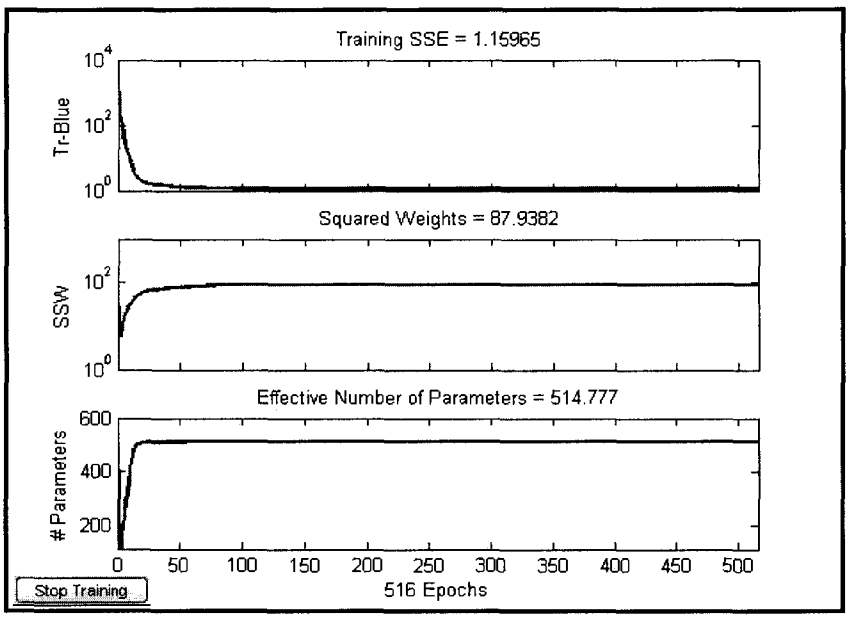


Figure 35: Convergence history of ANN training with Bayesian regularization

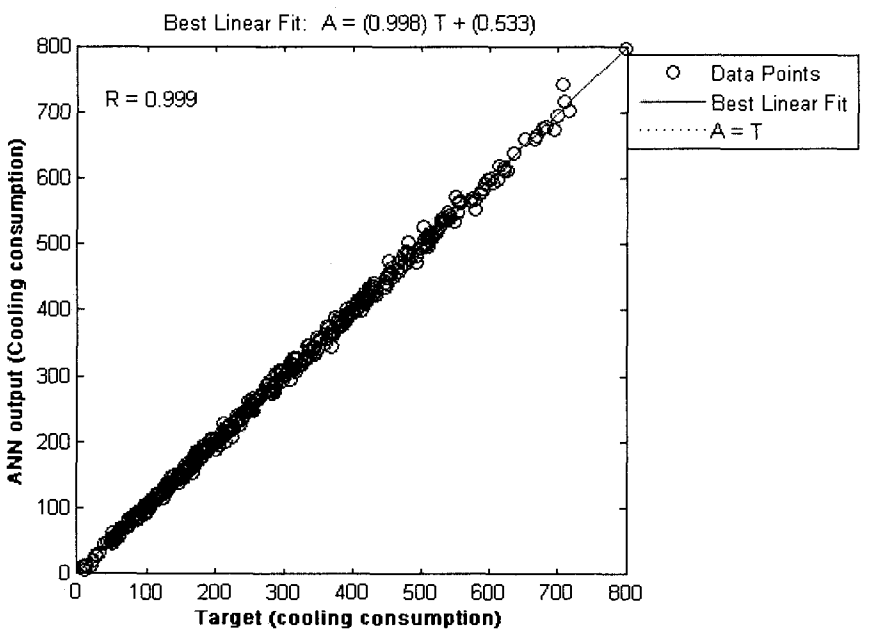


Figure 36: Linear regression of ANN predicted cooling consumption on targets

5.3.2 Artificial Neural Network Validation

A sample of 45 cases, different from the previous ones, was used for ANN validation. Figure 37 illustrates the relative error between ANN and TRNSYS outputs for the five outputs. These relative errors are summarized in Table 11. As can be seen, the average relative errors regarding energy consumption outputs are good, with 0.4% for heating, 2.6% for cooling, and 0.95% for fan consumptions. This leads to an average relative error around 0.5% for the total energy consumption. Regarding the two thermal comfort outputs, the average relative errors are a bit higher but still acceptable, with respectively 3.9% for the average absolute PMV and 5.2% for the cumulative time with discomfort.

Relative error		<1%	<2.5%	<5%	<10%	<25%	Average
Percentage of cases when error falls into the range	E_{heat}	93%	100%	100%	100%	100%	0.4%
	E_{cool}	33%	60%	89%	98%	100%	2.6%
	E_{fan}	58%	96%	100%	100%	100%	0.9%
	$\|PMV\|$	18%	40%	78%	96%	100%	3.9%
	N_{dis}	13%	38%	64%	84%	100%	5.2%

Table 11: Statistical repartition of relative errors in ANN validation

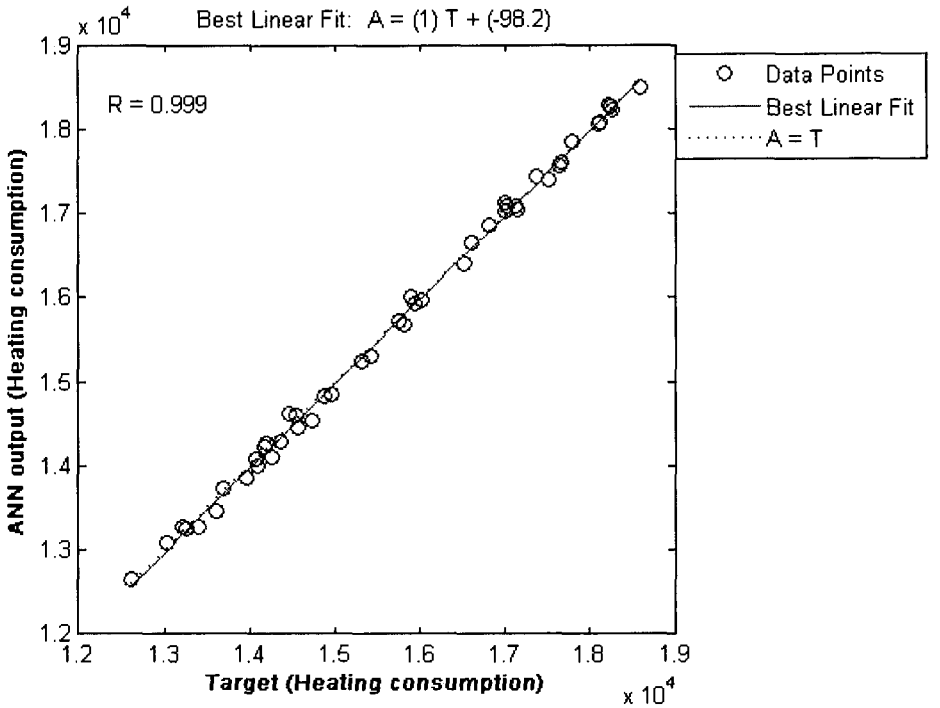


Figure 37a: Linear regression of ANN predicted heating consumption on targets

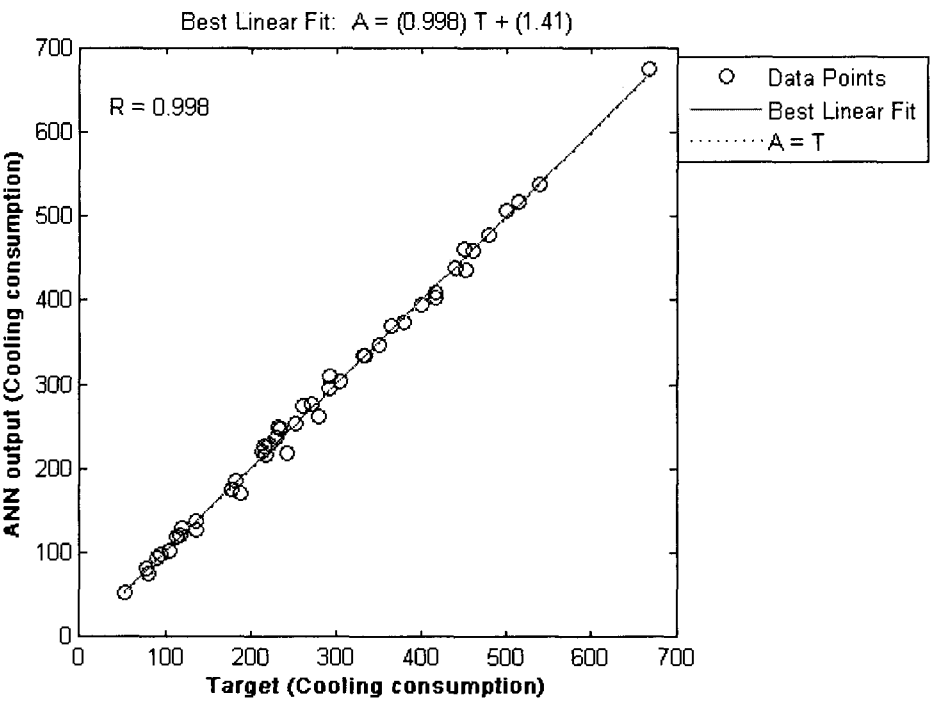


Figure 37b: Linear regression of ANN predicted cooling consumption on targets

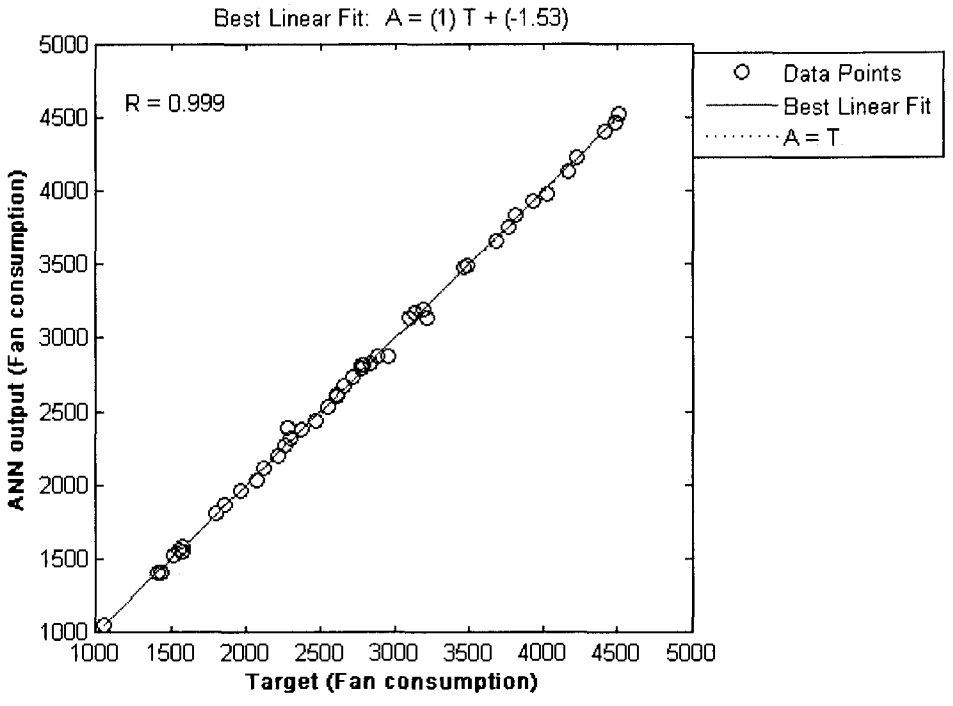


Figure 37c: Linear regression of ANN predicted fan consumption on targets

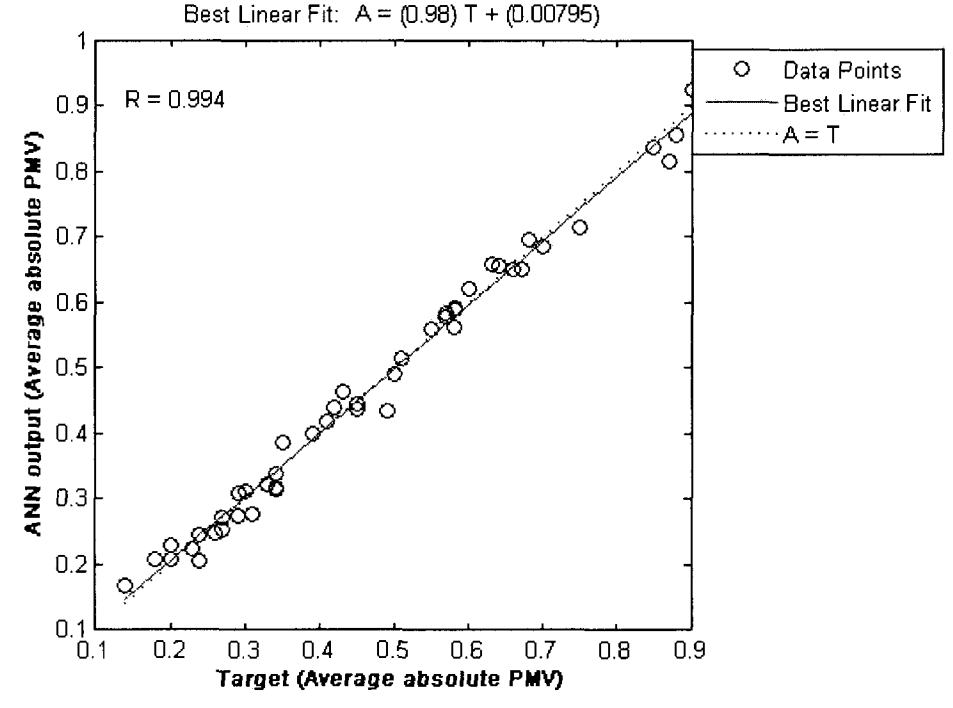


Figure 37d: Linear regression of ANN predicted average absolute PMV on targets

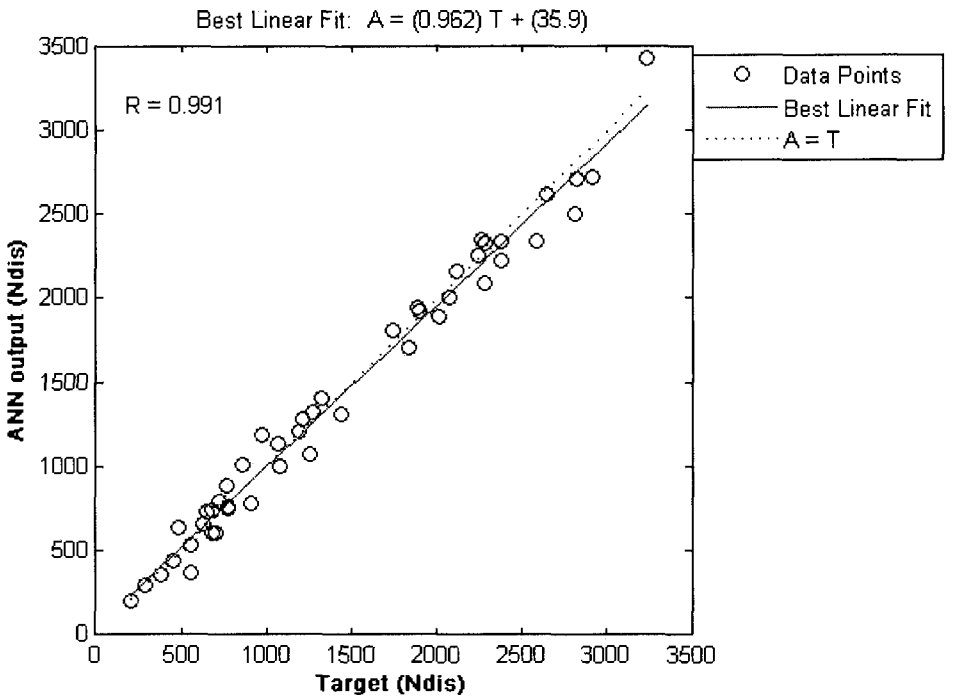


Figure 37e: Linear regression of ANN predicted N_{dis} on targets

5.3.3 Discussion

Reaching an acceptable ANN accuracy was difficult for the current case study. Using LHS, the general rule of thumbs states that a number of cases superior to 2.5xN, where N is the number of variables, is sufficient for ANN training. In the current case, this rule did not apply, and as much as 22.5xN cases were required to enable an acceptable ANN accuracy. An even higher number of training cases would have probably led to a better accuracy, but would have furthermore increase the computational time.

The number of neurons to set in the hidden layers was also difficult to determine. In the current case, increasing the number of neurons in the hidden layers could on average

improve ANN predictions, but it also increased the maximal errors. This result is caused by a phenomenon called overfitting, in which the ANN uses a high number of parameters to have a very high accuracy regarding the training data, at the cost of great variations between each training point. This behaviour is very dangerous in the current study since those great variations could lead to false optimum in further optimization. Based on the general idea that it is less dangerous for optimization to have small and frequent errors rather than rare but important ones, the author decided to keep 20 neurons in the hidden layer.

On the whole, the author considers the ANN accuracy as acceptable, since the relative errors for energy consumptions are very low, and since the relative error of 3.9% for PMV results in very little variations in the PMV value. The relative error for the cumulative time with discomfort is a little bit more problematic, but is still acceptable since this output will only be considered as a constraint.

5.4 MOEA Optimization

5.4.1 Optimization Set-up

The optimization problem was considered as a two-objective problem; the total energy consumption and the thermal comfort as objectives. Since thermal discomfort should never occur inside the house, the cumulative number of hours with discomfort (N_{dis}) has been handled as a constraint. The optimization problem can be summarized as the minimization of:

$$\begin{cases} F_1 = (E_{heat} + E_{cool} + E_{fan}) \times (1+PT) \\ F_2 = |PMV| \times (1+PT) \end{cases}$$

Where:

_ E_{heat}, E_{cool}, E_{fan} are the energy consumptions for respectively heating, cooling, and fan operation;

_ |PMV| is the average absolute PMV over the whole year, and;

_ PT is a penalty term, equal to the annual cumulative time where the |PMV| is higher than 0.5 (N_{dis}), divided by 100. (Dividing N_{dis} by 100 was found sufficient to ensure a good constraint handling, while maintaining an acceptable convergence rate).

All the parameters used in the evaluation function are calculated by the neural network. The code required to implement ANN calculations inside the MOEA is straightforward. In order to significantly reduce the evaluation time, the following method should however be used. Instead of using the MATLAB default procedure, ANN calculations should be reconstructed manually, based on the weights and bias matrices (see Appendix D). Doing so, calls to MATLAB complex routines regarding ANN architecture are not required, and the evaluation time is significantly reduced. According to tests performed by the author, this method enables to perform 10,000 evaluations in 1.44 seconds, while MATLAB default calculation takes as much as 38.8 seconds for the same work. The former method was thus used in the remaining of the thesis.

5.4.2 First Optimization

In the first optimization set-up, the 20 parameters described in section 5.1.4 (Table 8) were used as variables. NSGA-II, NSGAINN, and PLAGUE were run with the same parameters as in Chapter 4, with a termination criterion set to a maximal runtime of 600 seconds. The optimization results are illustrated in Figure 38.

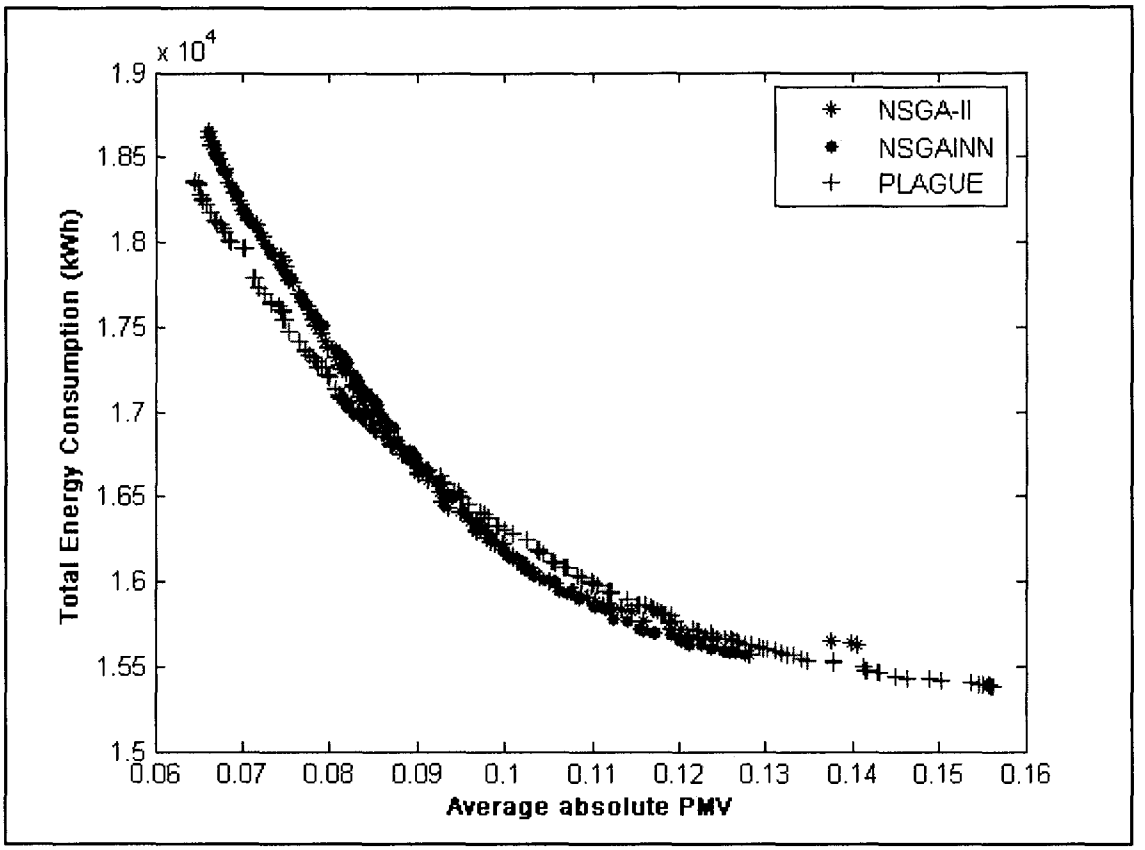


Figure 38: Results of the first optimization

Optimal solutions range from an absolute average PMV of 0.064 for an annual energy consumption of 18,360 kWh, to an annual energy consumption of 15,380 kWh for an absolute average PMV of 0.156. The spreading of solutions between these two extrema was good, and drew an almost continuous curve. Also, in all cases, the constraint was properly handled with a penalty term – and therefore a cumulative time with discomfort – equal to zero.

The three different MOEAs display similar results, regarding convergence and spreading of solutions. PLAGUE appears to provide the best spreading, while NSGAINN and NSGA-II provide slightly better results for |PMV| values between 0.09 and 0.13. For the purpose of comparison, optimization results were compared with the base case configuration, with four manually constructed designs expected to provide good results, and with five random designs (see Appendix E). Results are illustrated in Figure 39 (the penalty term was not considered for these evaluations). As can be seen, MOEAs solutions sets are better than the base and the random cases in terms of both comfort and energy consumption. Three of the manually constructed designs enable lower energy consumptions than the optimization results. However, these three cases gender more than 1000 hours per year with an average PMV higher than 0.5, and should therefore be rejected.

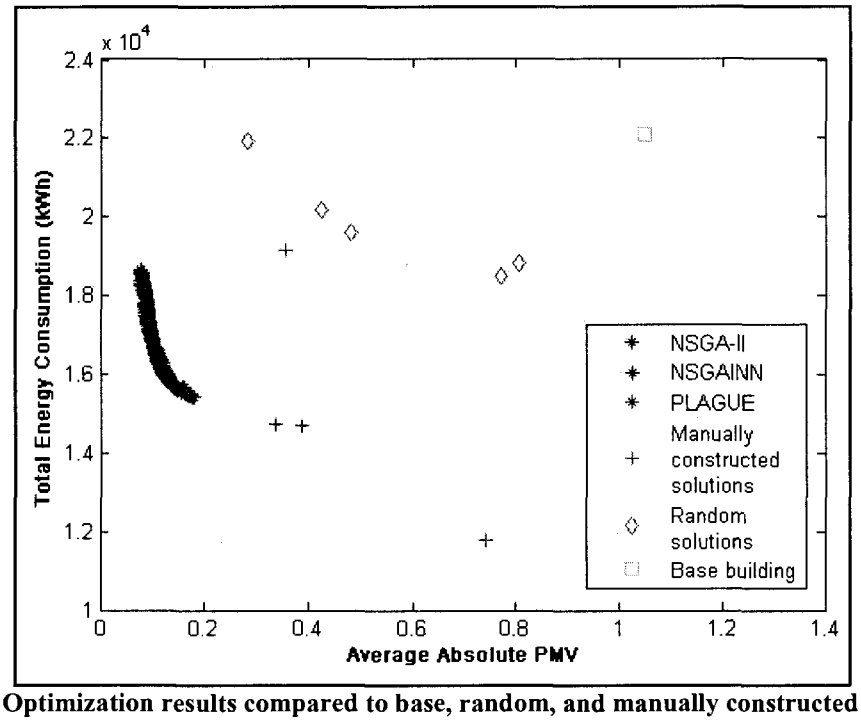


Figure 39: Optimization results compared to base, random, and manually constructed cases

Variables		Range	Base case	Optimal solutions	
				Lower value	Upper value
Temperature Setpoints	HSP	[20,25]	21	22.3	23.5
	CSP	[23,27]	21	24.6	24.9
Starting Delays	SDHW	[0,30]	Not applicable: constant temperature set points all year long	0	29.9
	SDMID	[0,30]		29.7	30
	SDS	[0,30]		19.7	30
Stopping Delays	FDMID	[0,60]		16.8	60
	FDS	[0,60]		0.1	60
	FDW	[0,60]		0	24.8
Ventilation Rates	VRR	[0.118,0.708]	0.448	0.118	0.118
	VRC	[0.118,0.708]	0.680	0.118	0.120
	VRH	[0.118,0.708]	0.618	0.469	0.708
Relative Humidity	RHW	[30,60]	none	59.9	60
	RHMID	[30,60]	none	30	60
	RHS	[30,60]	none	59.1	60
Windows Sizes	WF1N	[4.76, 14.30]	9.4	4.77	14.30
	WF1S	[2.20, 6.60]	5.29	2.41	6.61
	WF2N	[4.06, 12.18]	3.69	4.06	12.19
	WF2S2	[1.38, 4.14]	3.87	1.38	3.96
	WF2S1	[2.08, 6.25]	4.02	2.79	6.25
Thermal mass	TCK	[0.05,0.25]	0.05	0.25	0.25

Table 12: Variables ranges in optimal and base designs

Table 12 summarizes the range of values taken by optimal solutions' variables. The most salient fact of the optimization is the thickness of concrete in the interiors floors, set to the maximal values in all cases. This additional thermal mass enables to store more heat from sun radiation, and to smooth the temperature variation, especially in summer. Heating and cooling set points vary respectively between 22.3°C and 23.5°C, and between 24.6°C and 24.9°C, which is relevant with ASHRAE comfort range. Recirculation and heating ventilation rates are always set to the minimal value, in order to reduce fan consumption. Finally, relative humidity set points are set to 60% in winter and summer to respectively improve comfort and decrease dehumidification consumption.

These relative humidity values are high, but are still in accordance with ASHRAE comfort zone. It may however cause problems such as condensation or mold growth in winter. The other variables such as windows areas or thermostat delays take a wide range of values, in order to find all optimal trade-offs between energy consumption and thermal comfort.

In a nutshell, this first optimization was a success. The results provided by the MOEAs were significantly better than the base case and the manually constructed designs, regarding the two objectives studied. The spreading of the solutions was also very good, with an almost continuous curve. Finally, the ranges of variables in the optimal solutions appear to be relevant, and most variables effectively vary along the optimal front. Nevertheless, the thermal mass in optimal solutions is extremely high for a residential building. A second optimization was therefore set-up with a constant thermal mass equal to that of the base building.

5.4.3 Second Optimization

In the first optimization's results, the thermal mass of the building was in all cases set to the maximal value. Since the thermal mass of the building was very high, some parameters such as thermostat delays may not have been accurately studied. Therefore, a second optimization study was performed, letting all the parameters except the thermal mass vary. In this optimization, 19 variables were hence used, with a constant thickness of 5 centimetres of concrete. Thanks to GAINN particular approach, this second optimization did not require any additional TRNSYS simulations. The already trained

ANN was directly used and only the MOEA optimization part had to be redone. The three MOEAs were run with the same parameters as before. Results of optimization are shown in Figure 40, and variables ranges are summarized in Table 13.

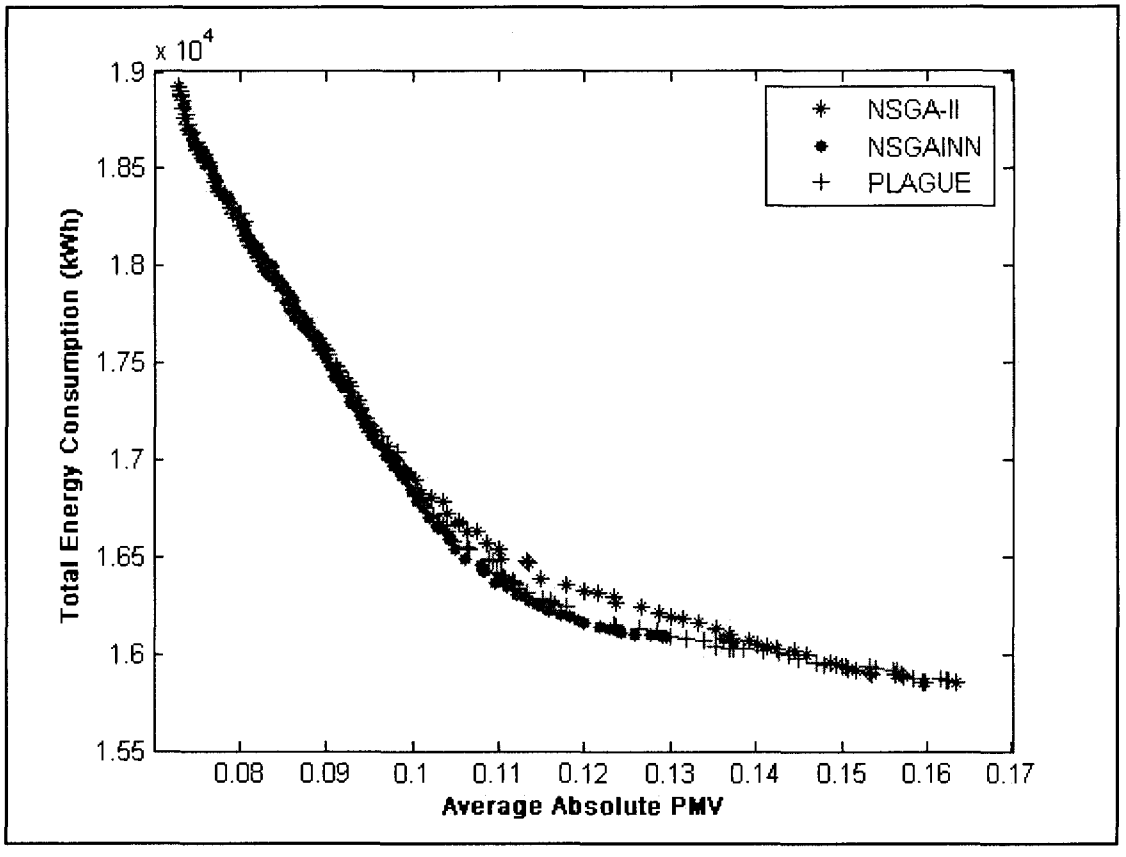


Figure 40: Results of the second optimization

Once again, the optimization was very efficient in terms of both convergence and spreading of the solutions, with the optimal front being an almost continuous curve. Optimal solutions range from an absolute average PMV of 0.073 for an annual energy consumption of 18,912 kWh, to an annual energy consumption of 15,960 kWh for an absolute average PMV of 0.159. As expected, the energy consumptions of optimal

solutions were higher in this second optimization than in the first optimization. Regarding the variable changes between the two optimizations, we can see in Table 13 that some variables such as ventilation rates or relative humidity set points are similar in both solution sets. The most salient changes between the two optimizations results are the window sizes and the delays used in the thermostat program. Particularly, in the second optimization, sizes of south windows are generally set to very low values. This can be explained by the fact that the thermal mass of the house is smaller, and therefore overheating is more likely to occur than in the first optimization set-up.

Variables		Range	Optimal solutions in second optimization			Average in the first optimization
			Lower value	Upper value	Average	
Temperature Setpoints	HSP	[20,25]	22.6	23.6	23.2	23.0
	CSP	[23,27]	24.7	25.1	24.9	24.8
Starting Delays	SDHW	[0,30]	21	30	29	13
	SDMID	[0,30]	26.4	30	29.5	30.0
	SDS	[0,30]	29.8	30	30.0	29.4
Stopping Delays	FDMID	[0,60]	24.0	60	52.0	38.8
	FDS	[0,60]	59.6	60	59.9	48.1
	FDW	[0,60]	0	60	17	1
Ventilation Rates	VRR	[0.118,0.708]	0.118	0.118	0.118	0.118
	VRS	[0.118,0.708]	0.118	0.118	0.118	0.118
	VRH	[0.118,0.708]	0.529	0.708	0.645	0.591
Relative Humidity	RHW	[30,60]	60	60	60.0	60.0
	RHMID	[30,60]	30	60	43	45
	RHC	[30,60]	60	60	60.0	60.0
Windows Sizes	WF1N	[4.76, 14.30]	4.77	4.80	4.77	5.55
	WF1S	[2.20, 6.60]	3.87	5.88	4.98	5.34
	WF2N	[4.06, 12.18]	4.06	12.19	6.60	5.70
	WF2S2	[1.38, 4.14]	1.38	3.70	1.60	1.66
	WF2S1	[2.08, 6.25]	2.08	4.51	2.24	4.93

Table 13: Variable ranges in optimal solutions

It is interesting to note that the relation between the average PMV and the energy consumption depicted in Figure 40 is not a straight line. The curve seems to be composed of two lines of different slopes. In terms of design, we can see that in the first part of the curve (below |PMV| = 0.11), small decreases of thermal comfort can lead to relatively large reductions in energy consumption. In the second part of the curve, the inverse situation occurs, and small increases of energy consumption can lead to significant increases of thermal comfort. This case highlights the major advantage of a true multiobjective optimization, which is to provide a complete understanding of the situation, and to bring to light the potentiality of each investment. In the current case, the occupants of the house could be easily convinced to lower the average PMV from 0.08 to 0.11, in order to reduce energy consumption by up to 11%.

5.4.4 Verification of Results using TRNSYS

Even if the ANN was properly trained, some differences may appear between ANN outputs and simulation results. Using GAINN methodology, it is important to check optimized solutions using the base simulation software, to ensure that the ANN predictions were correct. Due to the large number of solutions in the current study, only a small percentage of results were verified. Results may therefore not be representative of the accuracy of the ANN over the complete solution sets. Nonetheless, according to the test performed on 8 random optimal solutions, it appears that the ANN is very accurate in terms of energy consumptions. The average relative errors between ANN predictions and TRNSYS results were respectively 1%, 2.3%, and 3.3% for heating, cooling, and fan

consumptions. This leads to a very good average relative error in the total energy consumption of approximately 1%. However, regarding the average absolute PMV, some significant errors appear. The PMV values coming from ANN calculations were underestimated by an average of 0.02 over the 8 cases studied. In one case, the error in the PMV value was even as high as 0.05. Such errors mean that, even if the ANN was properly trained with an acceptable average relative error, it may still be relatively inaccurate in the vicinity of optimal solutions.

Due to time considerations, the author could not run more simulations to try to make the ANN more accurate. Moreover, it appears that PMV values are just shifted by 0.02, so we may assume that optimization results are still reliable, except in that they overestimate the thermal comfort. The improvement compared to the base design is also notably higher than ANN errors. Nonetheless, the issue of the accuracy of the ANN in the optimal region should be more carefully studied in the future.

5.5 Conclusion

GAINN methodology and the developed algorithms were successfully applied to this case study. Although it required a significant amount of training data, the ANN was able to accurately approximate the base building simulation software. Thanks to this ANN, two optimizations were undertaken with a computational time as low as 5 minutes in both cases. The total computational time associated with the whole optimization (i.e. including

ANN training and validation) is approximately three weeks. Based on the number of evaluations of NSGA-II (around 140,000 for a 5 minutes run), each optimization would have taken more than 10 years if ANN approximations were not used. In other words, this optimization would have never been possible without using GAINN methodology.

Regarding the optimization results, the developed algorithms (NSGAINN and PLAGUE) as well as NSGA-II performed very well in these optimizations, regarding convergence and spreading. Optimal solutions display significant improvement in terms of both thermal comfort and energy consumption when compared to the base case, or to manually constructed cases. Thanks to the multiobjective algorithms, a wide range of solution was covered, drawing an almost continuous curve. From a design standpoint, it means that the optimization results are valuable in any situation, no matter the priorities of the decision maker regarding thermal comfort or energy consumption. Moreover, thanks to the curve drawn by the optimal front, one can see the impact on thermal comfort of any reduction or increase in the energy consumption. The final decision can therefore be based on a real understanding of the situation, and of the impact of energy consumption on thermal comfort.

The second optimization highlighted a very useful ability of GAINN methodology, which is to be able to change the optimization set up without requiring any additional simulation. Once the ANN is trained, any optimization can be performed using a lower number or smaller ranges of variables, and take only few minutes to compute. This enables to focus

on some parameters, or, as in the current thesis, to ignore one variable in order to get more realistic results.

Finally, one strong limitation was highlighted by the current case study. Even if the ANN is accurate regarding validation data, some significant errors may appear in the vicinity of optimal solutions. While this does not invalidate the complete methodology, it could seriously harm the relevancy of the optimization results. This point need to be more carefully studied in future work, and some strategies should be developed to ensure the accuracy of the ANN in the optimal region.

CHAPTER 6

SECOND CASE STUDY: THREE-OBJECTIVE OPTIMIZATION OF A SCHOOL

This last chapter describes a second optimization study, based on the work of Conraud (2008). Through this study, the developed algorithms will be used with an ANN validated by Conraud, in order to optimize the energy consumption, and the thermal and visual comforts of a school. This study has two purposes. First, it will expose the capability of developed MOEAs on a three-objective problem. Then, it will highlight the improvement added to GAINN by the use of MOEAs, compared to Conraud's study where weighted-sum optimization was used. It is well known from the literature that MOEAs generally display several advantages over classical GAs using weighted-sum. This chapter will deeply study the differences between each optimization approach, in the particular case of an association with GAINN methodology.

6.1 Description of the Design Problem

All the work described in this section has been set up, developed, and studied by J. Conraud for his Master Thesis at Concordia University (2008). The description provided here is a summary of the optimization problem studied and the reader is referred to the original thesis for further details.

6.1.1 Presentation of the Building

The building studied is a school, located in Grong (Norway), which is specifically designed to take advantage of natural daylighting and natural ventilation. The school was built with a concern to reduce the heating and ventilation energy consumption and to provide good indoor air quality conditions as well as good visual and thermal comforts for pupils. The main feature of the building is an underground system used to preheat and precool the air, and distribute it to the classrooms. The air is then collected to an extract chamber where it can be exhausted (Figure 41). The model used to simulate the building is based on the ESP-r simulation developed by Wachenfeldt (2003). A bird view of this model is illustrated in Figure 42.

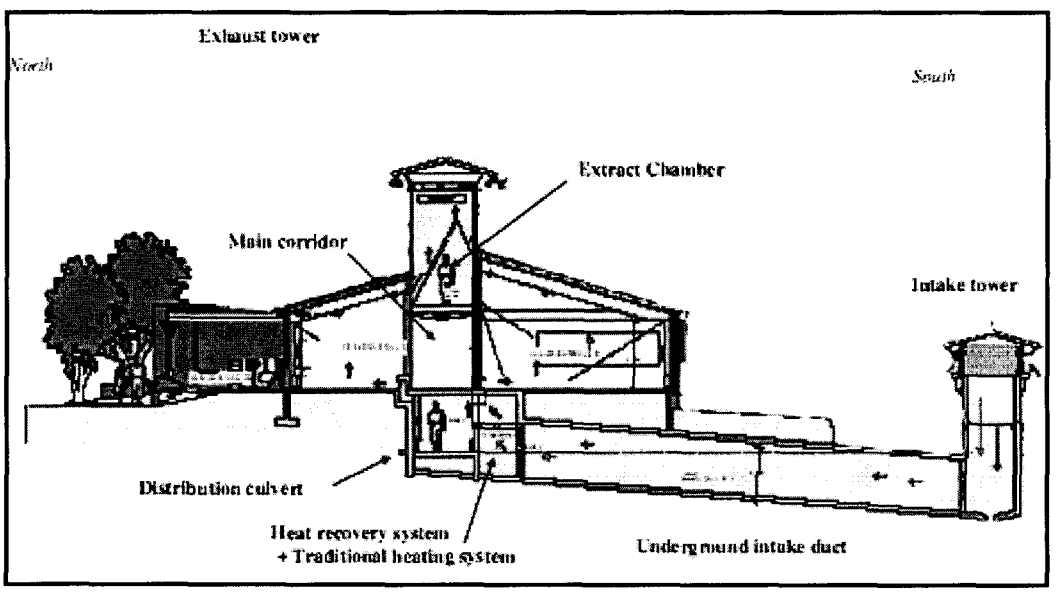


Figure 41: Cross-sectional view of the ventilation system (from Conraud 2008)

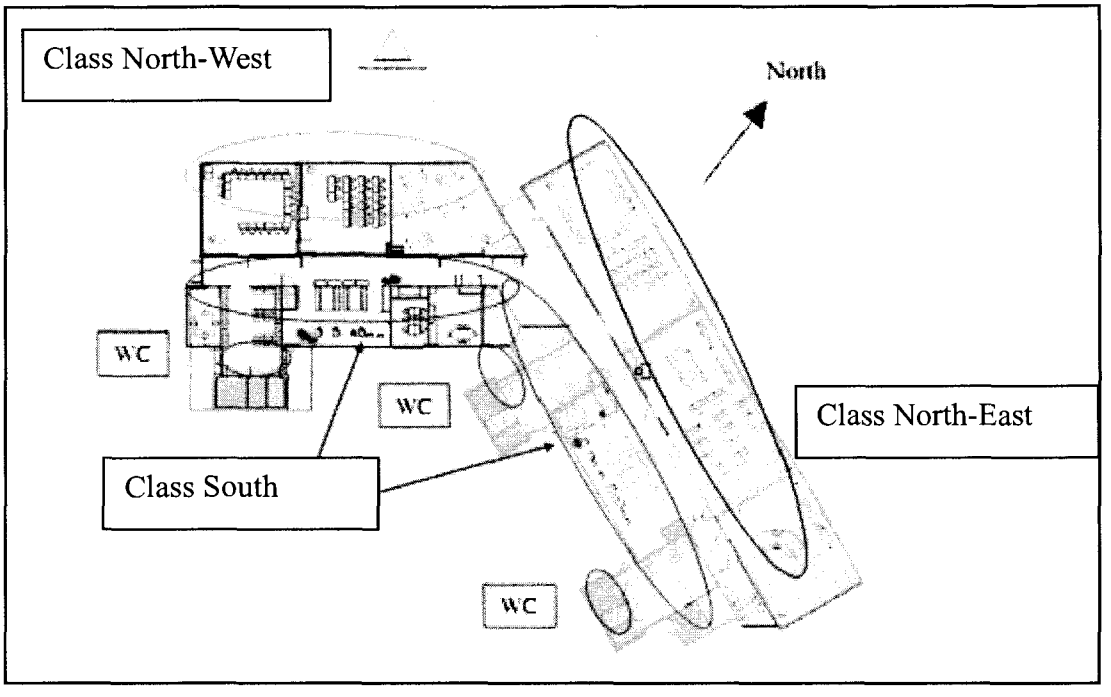


Figure 42: Bird view of the model (adapted from Conraud 2008)

6.1.2 Optimization Objectives

Three objectives, divided into 8 outputs, are studied as part of the optimization:

Total energy consumption (2 outputs)

The energy performance of the school is assessed by the annual heating and cooling loads. These values were calculated using ESP-r.

Thermal comfort (3 outputs)

The thermal comfort is represented by the cumulative frequency of time for which the Predicted Percent Dissatisfied (PPD) in the classroom is lower than 20%. Clothing levels were set at 1.0 clo in winter, 0.75 clo in autumn and spring, and 0.5 in summer, and metabolic rates were 100 W/m² to account for children's activity.

Visual comfort (3 outputs)

The visual comfort is assessed by daylight factors. A daylight factor is defined as the interior horizontal daylight illuminance expressed as a percentage of the horizontal daylight illuminance available to an unobstructed site. It was calculated for each classroom by ESP-r. The overall visual comfort is represented by the average of several daylight factors estimated at a series of points located halfway through the room from the windows, and one meter away from the side walls.

6.1.3 Optimization Variables

This case study investigated 24 parameters. Table 14 summarizes these parameters as well as their lower and upper bounds. They can be divided into four categories:

- Length and width of the windows (for the classrooms and the extract chamber);
- Thickness of insulation in the classrooms floors;
- Cooling capacity (for the classrooms, the corridor, and the distribution duct), and ;
- Temperature setpoints (for the classrooms, the corridor, and the distribution duct).

Zone	Variables	Upper Bound	Lower Bound	Conditions
Extract Chamber	Window Southwest Height	1.7 m	50% (0.85 m)	Continuous
	Window South Height	1.7 m	50% (0.85 m)	
Exhaust Tower	Height	6 m	2 m	
Class Northeast	Window – Length	5 m	50% (2.5 m)	
	Window – Height	1.7 m	50% (0.85 m)	
Class Northwest	Window – Length	17 m	50% (8.5 m)	
	Window – Height	1.7 m	50% (0.85 m)	
Class South	Window SE – Length	12.2 m	50% (6.1 m)	
	Window SE – Height	1.7 m	50% (0.85 m)	
	Window SW – Length	18.78 m	50% (9.39 m)	
	Window SW– Height	1.7 m	50% (0.85 m)	
Class Northeast	Insulation Thickness	15 cm	5 cm	{5,10,15}
Class Northwest	Insulation Thickness	15 cm	5 cm	
Class South	Insulation Thickness	15 cm	5 cm	
Class Northeast	Cooling Capacity	3,000 kW	0.0 kW	Continuous
	Temperature Setpoint	30°C	25°C	
Class Northwest	Cooling Capacity	3,000 kW	0.0 kW	
	Temperature Setpoint	30°C	25°C	
Class South	Cooling Capacity	5,000 kW	0.0 kW	
	Temperature Setpoint	30°C	25°C	
Corridor	Cooling Capacity	1,000 kW	0.0 kW	
	Temperature Setpoint	30°C	25°C	
Distribution Duct	Cooling Capacity	8,000 kW	0.0 kW	
	Temperature Setpoint	30°C	25°C	

Table 14: Study parameters and their upper and lower bounds (from Conraud 2008)

6.1.4 ANN Training and Validation

A Perl program developed by Conraud was used to automate 1,500 simulations for ANN training and validation. The neural network was composed of 24 inputs and 8 outputs previously described, and 15 neurons in the hidden layer, and was programmed in MATLAB. Once trained, the ANN performed relatively well regarding heating demand,

thermal comfort, and daylight factors (except south daylight factor) with an average relative error below 2% (Table 15). ANN predictions were less accurate regarding cooling demand and south daylight factors. For these outputs, the relative errors were respectively below 15% for 90% of the cases, and below 10% in 90% of the cases. Moreover, the maximal relative error regarding the cooling demand was as high as 82%. This very high maximal relative error may be extremely dangerous for optimization, since it may lead to false optimum. Nonetheless, Conraud decided to validate his ANN for further optimizations. This ANN was therefore also used by the author in the remaining of this chapter.

	Heating demand	Cooling demand	TC NWest	TC NEast	TC South	VC NWest	VC NEast	VC South
Max	0.60%	**81.98%**	5.06%	1.33%	1.24%	3.61%	8.02%	**21.10%**
Min	0.00%	0.01%	0.00%	0.00%	0.01%	0.00%	0.00%	0.01%
Average	0.13%	**6.45%**	0.44%	0.22%	0.29%	0.72%	1.91%	**2.50%**
Deviation	0.11%	9.40%	0.45%	0.19%	0.22%	0.60%	1.54%	3.18%

Table 15: Relative errors between building simulations and ANN predictions (Conraud 2008)

6.2. Optimization Search

6.2.1 Methodology

Original optimization results of Conraud thesis were kept unchanged for the current comparison. These results are based on an aggregative approach regarding multiple objectives, using the weighted-sum described below as a global performance metric:

$$\min L_2(x) = \min\left[\sum_{i=1}^{8} w_i \left|\frac{f_i(x) - \min f_i(x)}{\max f_i(x) - \min f_i(x)}\right|^2\right]^{1/2}$$

Where:

- x is an input vector belonging to the search space;

- f_1 to f_3 are the thermal comfort factors for the classrooms facing northwest, northeast, and south respectively;

- f_4 and f_5 are the total heating energy demand and total cooling energy demand;

- f_6 to f_8 are the average daylight factors for the classrooms facing northwest, northeast, and south respectively;

_ w_i are the weights associated with each function.

All objectives were assigned an equal importance by setting an absolute weight of 1 for each function. (A weight of -1 was used for functions for f_1 to f_3 and f_6 to f_8 since these functions are to be maximized).

In this study, the design problem was optimized as a three-objective minimization problem with objectives being respectively thermal comfort, energy consumption, and visual comfort. Using the same nomenclature as before, the design problem can be summarized as the minimization of:

Thermal comfort index $F_1(x) = -(f_1(x) + f_2(x) + f_3(x))$

Energy consumption $F_2(x) = f_4(x) + f_5(x)$

Visual comfort index $F_3(x) = -(f_6(x) + f_7(x) + f_8(x))$

MOEAs were run with the same parameters as in Chapter 4. Since there was no sufficient information available regarding the runtime of the original optimization, the author decided to run MOEAs with a time limit set to 60 seconds. This is assumed to be fair since this time is reasonably short, and since the original optimization was run until it reached a steady state optimum.

Early tests showed that, with a 60-seconds runtime, all three MOEAs gave very similar results. In the following, only PLAGUE solution set is kept to illustrate the comparison between MOEAs and Conraud's results.

6.2.2 Comparisons between PLAGUE's Solution Set and Conraud's Solution

The two solution sets coming respectively from Conraud study and from PLAGUE optimization are illustrated in Figures 43 to 46. The solution set coming from Conraud study is indeed limited to one single point, which is the optimal value of the weighted-sum studied. The solution space is three-dimensional; each solution is associated with three output values, representing the three objectives studied. Figure 43 shows a 3D view of the solution sets. Projections of the solution sets in two dimensional spaces are illustrated in Figure 44 to 46.

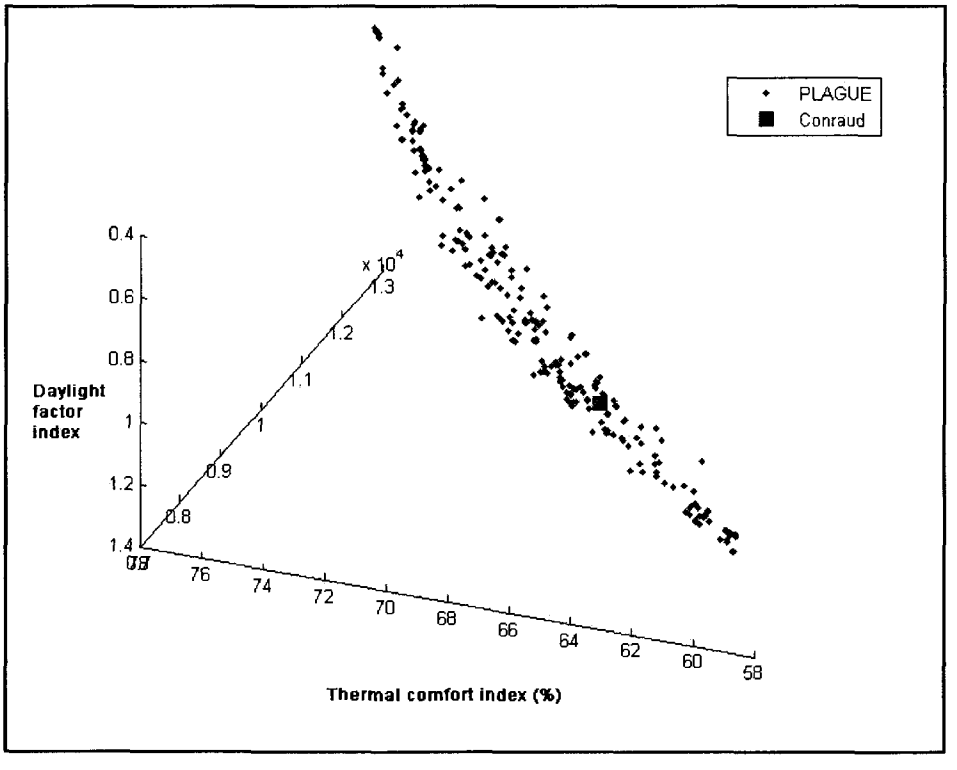


Figure 43: 3-D view of the solution sets

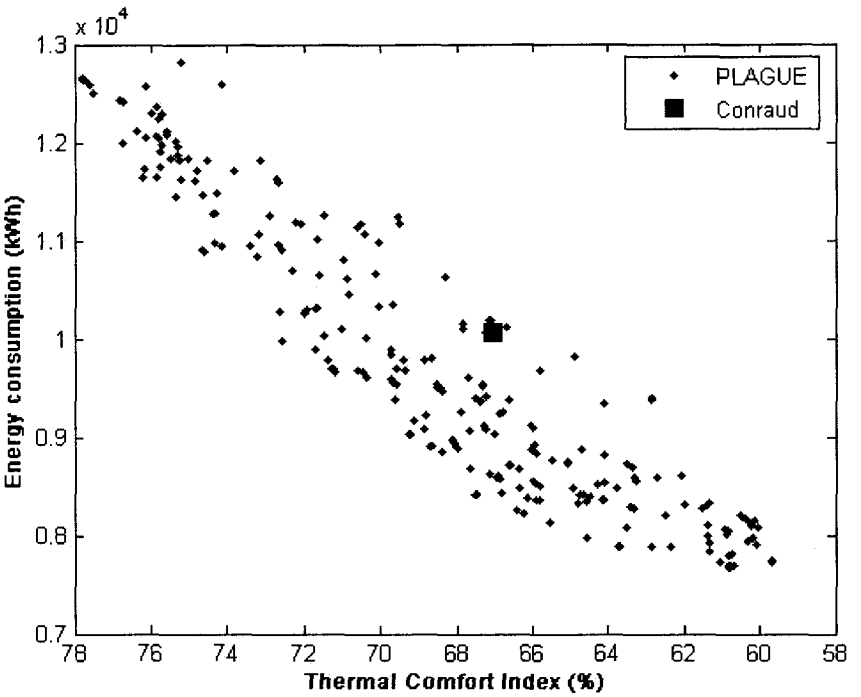


Figure 44: Thermal comfort index Vs energy consumption

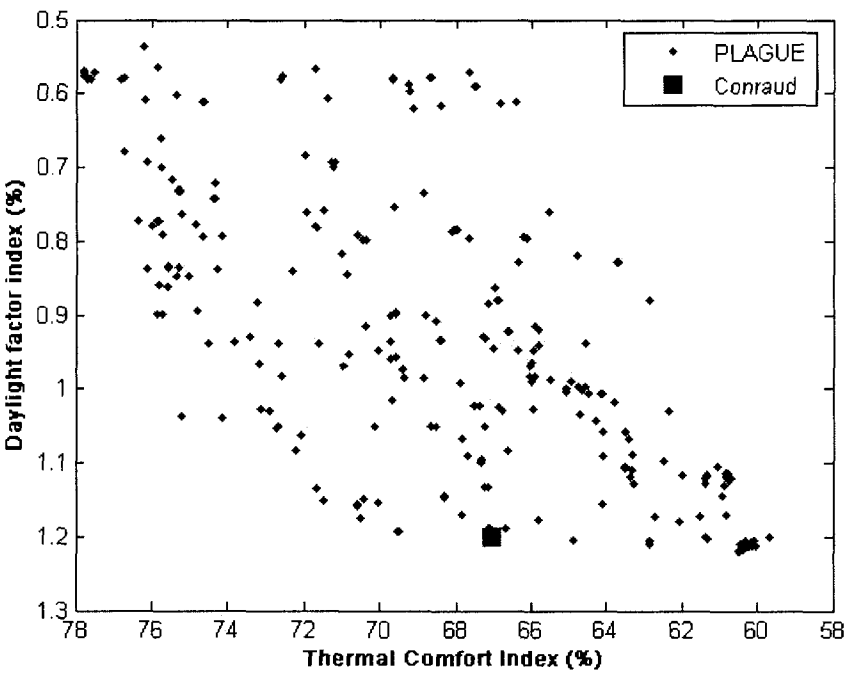


Figure 45: Thermal comfort index Vs daylight factor index

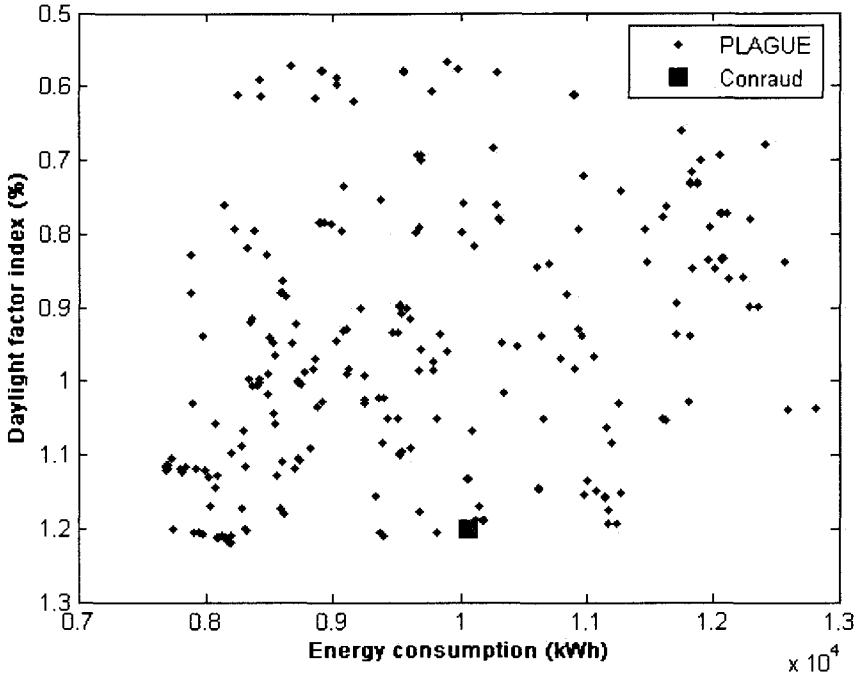


Figure 46: Daylight factor index Vs energy consumption

6.3 Discussion

6.3.1 In Terms of Optimization

In Figures 44 to 46, the points the closest to the lower-left corner are the best solutions regarding this two-objective space. As can be seen in Figure 45, no solution is better than Conraud's solution in terms of both thermal comfort and daylight factor index. Therefore, PLAGUE's Solution Set (PSS) does not dominate Conraud's Solution (CS). Further calculations showed that PPS is neither dominated by CS. In other words, individual results from PPS and from CS are equally good from a multiobjective point of view.

Consequently, one cannot argue that PLAGUE provides better individuals results than the aggregative method in that case. The methodology is not improved by the use of MOEA regarding the quality of each solution. This lack of improvement is probably due to the efficient non-linear weighted-sum used by Conraud. Even though there is no improvement from an *individual* standpoint, there is nonetheless a major improvement considering the overall population, as will be described in the next sections.

6.3.2 In Terms of Range of Solutions and Understanding of the Problem

As written previously, a solution set cannot be judged solely based on the convergence of each solution, but must also be studied in terms of spreading of these solutions. From Figures 43 to 46, the most striking improvement coming from using MOEAs is the wide range of solutions provided. Instead of a single point, dozens of Pareto-optimal solutions are found. The improvement is not actually the quantity of solutions, but the choices and of the understanding it enables. One hundred solutions mean one hundred suggestions to propose to a client, and one hundred opportunities for him to choose. In terms of understanding of the situation, the quantity of solutions enables to bring to light the relations between the objectives. In Figure 44 for instance, the relation between the energy consumption and the thermal comfort is obvious.

In addition, in the present case study, the base building was already very efficient in terms of comfort and energy performance. It was thus very difficult to assume, before

optimization, what improvements can be expected and what to focus on. Indeed, Conraud's final solution resulted in an increase of thermal comfort, but in a decrease of visual comfort and energy consumption. As can be seen on Figure 44, various other opportunities of improvement exist. From this figure, one can see that Conraud's solution, which is very good in terms of visual comfort, is relatively weak in terms of energy consumption and thermal comfort. Thanks to the quantity of optimal solutions the MOEA provides, the decision maker can have a global understanding of the potentialities of the building. Only then he can decide which modifications are the best ones, in his opinion, for the building.

6.3.3 In Terms of Accessibility

In terms of accessibility of the methodology, an important improvement added by MOEAs is that there is no longer need to select weights. The selection of weights is a rather complicated process, where importance of each objective has to be assumed. A bad set of weights would lead to an irrelevant and useless optimization. Therefore, a person wanting to use GAINN methodology had to either spend a significant amount of time and carefully think about the weight selection, or select one specific set of weights and produce a biased optimization (as in the current case). Thanks to MOEAs, the process of selecting weights is not longer required, and the methodology is therefore simpler and more accessible.

6.3.4 In Terms of Reliability

A last change in GAINN methodology coming from the use of MOEA relates to the ANN reliability and degree of precision. Obviously, the optimization algorithm cannot improve the accuracy of the ANN. However, since a multiobjective optimization provides a large set of solutions, the impact of little inaccuracy can be reduced. In the current case, ANN simulations were inaccurate regarding cooling load and south daylight factors in 10% of the cases. In one case, the relative error in the cooling load was even as high as 82%. As mentioned previously, this major inaccuracy may lead to false optimum and erroneous results. In Conraud optimization, since only one solution is provided, and since this solution was not verified using the base simulation software, there is a 10% probability that the optimization result is inaccurate. In other words, there is a chance that the *only outcome* of the case-study is erroneous.

Using a MOEA, and with a similar ANN inaccuracy in 10% of the cases, erroneous results may still appear. However, since more than one hundred solutions are provided, odds are that a significant number of them will be accurate. Even if there is no guarantee of accuracy for every single solution, a majority of results are still likely to be valuable. As a conclusion, the use of a MOEA inside GAINN methodology is safer, in that it compensates the possible inaccuracy of the ANN by providing more solutions, and since it does not limit the optimization outcome to a single and possibly erroneous point.

6.3.5 In Terms of Attractiveness

Finally, looking at the potential use of optimization in the industry, the implementation of a MOEA improves the attractiveness of GAINN methodology. In Conraud study, after almost one week of calculation, one single solution was provided. With a similar computational time, the use of MOEA provided one hundred optimization opportunities, as well as a better understanding of the design problem. The time invested in the overall process is therefore more efficiently exploited by MOEAs, and the optimization output is more attractive. Consequently, thanks to the use of MOEA inside GAINN, building optimization is more likely to be used in the industry, which could further enable significant improvements in terms of comfort, energy consumptions, and green house gases emissions.

CHAPTER 7

CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

7.1 Concluding Remarks on the Present Work

With a growing environmental awareness and demand for a comfortable environment, the process of building design should more than ever be optimized. Optimization tools such as Genetic Algorithms, or even Multiobjective Evolutionary Algorithms, exist and are known to be efficient but they rarely have been used in the industry at the design stage. The main reason of this lack of interest is the high number of evaluations and therefore the time investment they require. This thesis focused on the development of a new optimization approach called GAINN, and used this approach by combining it with MOEAs.

First, a very efficient existing MOEA, called NSGA-II, was chosen for the current study, programmed in MATLAB, and validated. Then, the author explored the opportunity of creating MOEAs specifically designed to take advantage of GAINN fast calculations. Two new MOEAs were developed: NSGAINN, and PLAGUE. These MOEAs both show a significant improvement compared to NSGA-II on several benchmark test functions. PLAGUE was found to be the most efficient MOEA for two-objective functions and for half of the three-objective functions. In the other half of the three-objective functions, NSGAINN was the most efficient MOEA.

In Chapter 5, the methodology and the developed MOEAs were used for the optimization of the energy consumption and the thermal comfort in a residential building. According to the validation data, the ANN was able to accurately predict the studied metrics with an average relative error of 0.5% for the total energy consumptions and an average relative error of less than 4% for the average absolute PMV. The limitations highlighted in Conraud (2008) regarding the increased need for training data also applied in this case, since a number of cases equal to 25 times the number of parameters was required for ANN training. A second important limitation was also highlighted regarding the accuracy of the ANN in the vicinity of optimal solutions. Even if the ANN was validated with an acceptable accuracy, some significant errors appeared in the final solution set. Nonetheless, the two optimizations undertaken were successful, and provided significantly better results than the base case and the manually constructed solutions.

Regarding the accessibility of the methodology, this case study proved that optimization can be performed relatively easy, without requiring strong programming skills. Providing sufficient instructions, non-experts could apply the methodology easily, thanks to the combination of TRNSYS, GenOpt automation, user-friendly MATLAB toolboxes for Latin Hypercube Sampling and ANN, and the programmed MOEAs.

In a second case study, the methodology was tested on a three-objective problem, and MOEA optimization was compared with the classical weighted-sum approach. Significant gains were found in terms of spreading of the solutions, accessibility of the methodology, and choice given to the decision maker. Moreover, thanks to the use of MOEAs, some processes such as the selection of weights for the aggregative sum are no

longer required, and the effects of ANN inaccuracy can be somehow lowered. On the whole, the use of a MOEA significantly improved GAINN methodology by exploiting more efficiently the time spent for training. The methodology became therefore more attractive and more likely to be used in the industry.

7.2 Limitations and Future Work

7.2.1 Regarding the Main Case Study

The optimization set-up of the main case study is limited in several aspects. Indeed, the purpose of this case study was mainly to propose an application of the methodology and of the developed algorithms. Many interesting studies have previously been performed regarding thermostat programming, including additional parameters such as set-back temperatures or condensation on interior surfaces (Maheshwari et al. 2000; Manning et al. 2007). The current case study was not designed to be as complete as those studies.

A major limitation of this case study is that the occupancy is based on a fixed schedule, which is very unlikely to be the case in actual situations. An optimal solution of this study is likely to produce discomfort if an occupant changes its schedule. One future work could be to include some randomness in the occupancy schedule, and study the optimization of the system regarding that specific situation. Such approach would make the whole study more complex but also more realistic. Another future work would be to take advantage of the multi-zonal model. In the current study, only the average of the

PMV over the four occupied zone was studied. A future work could exploit the multi-zonal aspect of the model, and optimize for instance the air distribution inside each zone according to thermal comfort.

Another limitation of the first optimization of this case-study is the influence of thermal mass. The TRNSYS model developed for this thesis has not been validated regarding thermal mass variations, and the model may not be accurate for cases with high thermal masses and solar radiations. Another simulation software could be tested to verify the results of optimization. Regarding the thermal mass, adding a 25 centimetres thick concrete slab may not be feasible for structural reasons. In this thesis, the concrete slab has been used as a convenient way to increase the thermal mass in TRNSYS, but this issue should be more carefully studied.

7.2.2 Regarding Developed Algorithms

The two MOEAs developed in this thesis are very promising, but may require more study. First, other programming languages and codes should be used to confirm the conclusions of this thesis. While the author is fairly confident in his conclusions, some programming issues could change the time spent on each process and therefore significantly change the comparison results (since tests were performed on a maximal runtime basis). Also, some constraint handling techniques should be implemented, so developed MOEAs can be tested on constrained problems. Finally, the influence of several parameters such as reproduction parameters should be studied in more details to enable the best efficiency of

each algorithm, based on its specific behaviour. The amount of time where NSGAINN behaves as NSGA-II (80% of the run so far) should also be optimized possibly based on the evolution of the population's fitness.

Finally, the two MOEAs developed in this thesis are expected to be efficient only with extremely quick evaluations functions such as an ANN function. Further research is needed to properly evaluate the performance and usefulness of NSGAINN and PLAGUE for other applications. Also, the time improvement added by developed algorithms is likely that it will have a very little impact on the overall optimization if GAINN is combined with time-expensive simulation softwares. In such cases, the reduction in the optimization time would be negligible compared to the time spent for ANN training. In order to take full advantage of the developed MOEAs, they should be used in problems where the optimization time is very limited, such as for online optimization (discussed hereafter).

7.2.3 Regarding GAINN Methodology

The main limitation of GAINN methodology concerns the ANN training and validation. In the main case study, the rule of thumbs stating that, using LHS, a number of cases greater than twice the number of parameters is sufficient for ANN training did not apply. This conclusion is in agreement with Conraud (2008). This additional need for training data multiplies the computational time. It should therefore be taken into account in future works. While the approach remains valuable in terms of time saving, further studies should be performed regarding the number of cases to use for ANN training in order to make sure that the ANN would be accurate in all situations. The opportunity of using other sampling methods (instead of LHS) and other training method (instead of back propagation) should also be studied.

A more problematic point concerns the accuracy of the ANN in the vicinity of optimal solutions. In the main case study of this thesis, even if the ANN displayed an acceptable accuracy regarding the validation data, significant errors appeared when optimal solutions were tested. Such inaccuracy of the ANN may lead to major optimization errors, and should be studied in details before GAINN could be validated as a reliable optimization methodology.

In order to increase ANN accuracy in the optimal region, it would be interesting in future studies to include some chosen cases in the training dataset, in addition to the random cases. For instance, some manually constructed cases, designed to be efficient, could be included for training. Although these cases are not expected to be optimal, they may

increase the accuracy of the ANN in the optimal region, and lead to more accurate results. The integration of ANN training according to GA's current population, as proposed by Nain and Deb (2005), could also be very useful if a tool is developed to link ANN, GA, and the simulation software. A simpler solution could also be to use a two-step procedure, with a first optimization based on a low-fidelity ANN (trained with a small number of cases), and a second optimization based on an ANN trained on cases coming from the first optimization.

Regarding the accessibility of the method, it has been improved by the use of MOEAs, due to the fact that weights selection and sensitivity analysis are no longer required. The use of GenOpt also enables to run hundred of simulations automatically, without requiring any action from the user during the parametric runs. Nonetheless, many aspects still affect the accessibility of the method. While the creation, training, and validation of the ANN are very easy in MATLAB, the selection of the number of cases for training remains difficult. The ANN construction and especially the number of neurons to be included in the hidden layer is not obvious either. Number of hidden neurons and number of cases for training should be more carefully studied for building applications, and guidelines should be proposed.

Finally, a very promising application of GAINN would be to use it for ongoing optimization. Ongoing optimization (also referred as online optimization) is a method in which the controls of a system are optimized in real time. This enables to obtain, at each moment, the best possible configuration, by adapting controls to weather or occupancy changes. The current main limitation of this technique is the difficulty of predicting the

building reaction to the changes of controlled variables (Coffey, 2008). The GAINN approach could perfectly overcome this drawback by using the ANN to provide fast predictions of building behaviour, and then optimize the control variables thanks to the GA. The need for training data for the ANN training would not be an issue in this case, since online optimization generally involves continuous monitoring of the building. Data could therefore be continuously stored, so the ANN training could become more efficient each day, making the GAINN methodology more accurate. The application of GAINN for ongoing optimization would be a remarkable future work, taking full advantage of the methodology to solve the very complex issue of building reactivity.

References

Amirjanov, A., and Sobolev, K. (2006). *Genetic algorithm for cost optimization of modified multi-component binders*. Building and Environment, *41*(2), 195-203.

ASHRAE (2004). ANSI/ASHRAE Standard 55-2004: Thermal Environmental Conditions for Human Occupancy, Atlanta, Georgia, USA.

Caldas, L. G. and Norford, L. K. (2002). *A design optimization tool based on a genetic algorithm*. Automation in Construction, 11(2), 173-184.

Chow, T. T., Zhang, G. Q., Lin, Z., and Song, C. L. (2002). *Global optimization of absorption chiller system by genetic algorithm and neural network*. Energy and Buildings, *34*(1), 103-109.

Coello Coello, C.A. and Toscano Pulido, G. (2001). *Multiobjective Optimization Using A Micro-Genetic Algorithm*, Proceedings of the Genetic and Evolutionary Computation. 274–282.

Coffey, B. (2008). *A Development and Testing Framework for Simulation-Based Supervisory Control With Application to Optimal Zone Temperature Ramping Demand Response Using a Modified Genetic Algorithm*. Master Thesis, Concordia University (Canada), Canada.

Conraud, J. (2008). *A Methodology for the Optimization of Building Energy, Thermal, and Visual Performance*. Master Thesis, Concordia University (Canada), Canada.

Dantzig, G. B. (1949). *Programming in a linear structure.* Econometrica, 17, 73-74.

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

Deb, K. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* 6 (2):182–197.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002). *Scalable multi-objective optimization test problems* Proceedings of the Congress on Evolutionary Computation 1, 825–830.

DOE (2008). Website. U.S. Department of Energy.http://www.eere.energy.gov/ (Last accessed December 5, 2008).

Edgeworth, F.Y. (1881). *Mathematical Physics.* New York, A.M. Kelley Publishers. As cited in Jain et al. (2005).

Engel, Y., Mannor, S., and Meir, R. (2004). *The kernel recursive least-squares algorithm. Signal Processing, IEEE Transactions on, 52*(8), 2275-2285.

Fanger, P. (2000). Provide Good Air Quality on Air Distribution in Rooms, Reading, United-Kingdom.

Fieldsend, J.E., Everson, R.M., and Singh, S. (2002). *Using unconstrained elite archives for multi-objective optimisation*, IEEE Transactions on Evolutionary Computation, 7 (3): 305–323.

Fletcher, R. (1979). *Practical methods of optimization*, Wiley, Chichester, New York.

Fu, G., Butler, D., and Khu, S. (2008). *Multiple objective optimal control of integrated urban wastewater systems*. Environmental Modelling & Software, *23*(2), 225-234.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning,* Kluwer Academic Publishers, Boston, MA.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, USA.

Hooke, R., and Jeeves, T. A. (1960). *Direct search solution of numerical and statistical problems*, Journal of the Association for Computing Machinery Journal, 8 212-229.

House, J.M., and Smith, T.P. (1995). *A System Approach to Optimal Control for HVAC and Building System*, ASHRAE Transactions, 101(2): 647-660.

Huang, W., and Lam, H. N. (1997). *Using genetic algorithms to optimize controller parameters for HVAC systems*. Energy and Buildings, *26*(3), 277-282.

IPCC (2007). Intergovernmental panel on climate change. website. http://www.ipcc.ch/(Last accessed December 5, 2007).

Jain, L. C., Goldberg, R., and Ajith, A. (2005). *Evolutionary multiobjective optimization: theoretical advances and applications*, Springer, New York.

Jin, Y. (2005). *A Comprehensive Survey Of Fitness Approximation In Evolutionary Computation*, Soft Computation, 9 (1) 3–12.

Lahanas M., Schreibmann E., Milickovic N., and Baltas D. (2003). *Intensity modulated beam radiation therapy dose optimization with multiobjective evolutionary algorithms*. Proccedings of the Second International Conference on Evolutionary Multi--Criterion Optimization (EMO 2003), 648-661.

Lee, J.H., Ko, Y.D., Han, K., and Yun, I. (2006). *Comparison of Latin Hypercube Sampling and Simple Random Sampling Applied to Neural Network Modeling of HfO2 Thin Film Fabrication*, Transactions on Electrical and Electronic Materials, 7-4, 210-214.

(The) Leadership in Energy and Environmental Design (LEED) Green Building Rating System. (2007), Available from: http://www.usgbc.org/DisplayPage.aspx?CMSPageID=222.

Li, K.C., Lue, H.H., and Chen, C.H. (2000). *Interactive tree-structured regression via principal Hessian directions*. Journal of the American Statistical Association 95, 547-560.

Lu, L., Cai, W.J., Xie, L.H., Li, S.J., and Soh, Y.C. (2005). *HVAC System Optimization in Building Section*, Energy and Buildings, 37, 11-22.

Maheshwari, G.P, Al-Taqi, H., Al-Murad, R., and Suri, R.K. (2000). *Programmable thermostat for energy saving*. Energy and Buildings 33, 667-72.

Majumdar, S., Mitra, K., and Raha, S. (2005). *Optimized species growth in epoxy polymerization with real-coded NSGA-II*. Polymer, *46*(25), 11858-11869.

Manning, M., Swinton, M.C., Szadkowski, F., Gusdorf, J., Ruest, K. (2007). *The effects of thermostat setback and setup on seasonal energy consumption, surface temperatures, and recovery times at the CCHT twin house research facility*. ASHRAE Transactions 113(1), 1-12.

Matlab (2006), Documentation, Version 71, MathWorks Inc.

McKay, M.D. (1988), *Sensitivity Arid Uncertainty Analysis Using a Statistical Sample of Input Values*, in Uncertainty Analysis, Y. Ronen, ed., CRC Press, 145-186.

Morimoto, T., Takeuchi, T., and Hashimoto, Y. (1993). *Growth Optimization Of Plant By Means Of The Hybrid System Of Genetic Algorithm And Neural Network.* Proceedings of 1993 International Joint Conference on Neural Networks, 2979-2982.

Nain, P. K. S. and Deb, K. (2005). *A multi-objective optimization procedure with successive approximate models*. KanGAL Report No. 2005002. Kanpur, India.

Nandi, S., Mukherjee, P., Tambe, S., Kumar, R., and Kulkarni, B. (2002). *Reaction modeling and optimization using neural networks and genetic algorithms: Case study involving ts-1-catalyzed hydroxylation of benzene*, Industrial & Engineering Chemistry Research, 41 (9), 2159–2169.

Nassif, N., Kajl, S., and Sabourin, R. (2003) *Two-objective online optimization of supervisory control strategy*, Proceedings of the Eighth Building Simulation Conference (IBPSA'03), 1, 927-934, Eindhoven, Netherlands.

Natural Resources Canada (2005). Energy Use Data Handbook Tables (Canada): Table 2 & 3, available at http://oee.nrcan.gc.ca/corporate/statistics/neu d/dpa/handbook_totalsectors_ca.cfm?attr=0, (Last accessed November 4, 2008).

Pala, M., Caglar, N., Elmas, M., Cevik, A., and Saribiyik, M. (2008). *Dynamic soil–structure interaction analysis of buildings by neural networks*. Construction and Building Materials, *22*(3), 330-342.

Pareto, V. (1896). *Cours D'économie politique*, volume I and II. Lausanne, Rouge. As cited in Jain et al. (2005).

Peippo, K., Lund, P. D., and Vartiainen, E. (1999). *Multivariate optimization of design trade-offs for solar low energy buildings*. Energy and Buildings, 29(2), 189-205.

Réglementation Thermique 2005 (2006). *Arrêté du 24 mai 2006 relatif aux caractéristiques thermiques des bâtiments nouveaux et des parties nouvelles des bâtiments*. Journal Officiel de la République Française no 121 of 25 May 2006 (in French).

Sakamoto, Y., Nagaiwa, A., Kobayasi, S., and Shinozaki, T. (1999), *An Optimization Method of District Heating and Cooling Plant Operation Based on Genetic Algorithm*, ASHRAE Transactions, 105(2): 104-115.

Srinivasan, D., and Seow, T. (2005). *Particle Swarm Inspired Evolutionary Algorithm (PS-EA) for Multi-Criteria Optimization Problems*, Evolutionary Multiobjective Optimization, 147-165.

Sun, J., and Reddy, A. (2005). *Optimal control of building HVAC&R systems using complete simulation-based sequential quadratic programming (CSB-SQP)*. Building and Environment, 40(5), 657-669.

Swinton, M.C., Entchev, E., Szadkowski, F., Marchand, R.G. (2003) *Benchmarking twin houses and assessment of the energy performance of two gas combo heating systems*, Proceedings of the Ninth Canadian Conference on Building Science and Technology, Vancouver, BC, NRCC-38459, 365–381.

Syarif, A., and Gen, M. (2003). *Solving exclusionary side constrained transportation problem by using a hybrid spanning tree-based genetic algorithm*. Journal of Intelligent Manufacturing, 14(3/4): 389-399.

Simpson, T., Mauery, T., Korte, J., and Mistree, F. (1998). *Comparison of response surface and Kriging models for multidiscilinary design optimization*. 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 98-4755.

Valenzuela C. (2002). *A Simple Evolutionary Algorithm for Multi-Objective Optimization (SEAMO),* Congress on Evolutionary Computation (CEC'2002), IEEE press, 1, 727-733

Wachenfeldt, B.J. (2003), *Natural Ventilation in Buildings. Detailed Prediction of Energy Performance*, Ph.D. Thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Wang, S.W., and Jin, X.Q. (2000). *Model-based Optimal Control of VAV Air-conditioning System Using Genetic Algorithm*, Building and Environment, 35, 471-478.

Wang, W., Rivard, H., and Zmeureanu, R. (2006). *Floor shape optimization for green building design*. Advanced Engineering Informatics, 20(4), 363-378.

Weather Office (2008). Website. Environment Canada,. http://www.weatheroffice.gc.ca/canada_e.html (Last accessed December 5, 2008).

Wetter, M. (2004). *Simulation-Based Building Energy Optimization*. Ph.D. dissertation, Berkeley University. California.

Wetter, M. (2001), *GenOpt(r) - A Genetic Optimization Program*, Proceedings of the 7th International IBPSA Conference, Rio de Janeiro, Brazil.

Wetter, M. And Wright, J. (2003), *Comparison of a Generalized Pattern Search and a Genetic Algorithm Optimization Method*, Proceedings of the 8th International IBPSA Conference, Eidhoven, the Netherlands.

Wetter, M. and Wright, J. (2004). *A Comparison of Deterministic and Probabilistic Optimization Algorithms for Non-smooth Simulation-based Optimization*, Building and Environment, 39(8), 989-999.

Wright, J. A., Loosemore, H. A., & Farmani, R. (2002). *Optimization of building thermal design and control by multi-criterion genetic algorithm*. Energy and Buildings, 34(9), 959-972.

Yang, J., Rivard, H., and Zmeureanu, R. (2005). *Building Energy Prediction with Adaptive Artificial Neural Networks*, Proceedings of the 9th International IBPSA Conference, Montreal, Quebec, Canada.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD Thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Zitzler, E., Deb, K., Thieler, L. (2000). *Comparison of multiobjective evolutionary algorithms: empirical results.* IEEE Transactions on Evolutionary Computation 8: 173-195.

Zitzler, E. and Thiele, L. (1999). *Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach*. IEEE Transactions on Evolutionary Computation, 3(4): 257-271.

Zhou, L. (2007). *Optimization of ventilation system design and operation in office environment*. Ph.D. dissertation, Concordia University (Canada), Canada.

APPENDIX A:

MOEA CODES

NSGA_II.m

```
clc
tic

%% Initialize constants and main parameters
constants

CPUTIME=0;
%% Initialize the population
population                                                                    =
initialize_population(N,M,V,UB,LB,Step,flw,slw,bi,mint,maxt,CPUTIME,time_limit);


%% Sort the initialized population based on rank and crowding distance
population = non_domination_and_crowding_sort(population,N,M,V);


CPUTIME=toc;

%% Start the evolution process

while (generation<=max_generation)&&(CPUTIME<time_limit)
   tic

   %% Select the parents (based on tournament selection)
   parent_population = tournament_selection(population,M,V,pool_size);


   %% Generate offspring by crossover and mutation
   offspring_population                                                       =
genetic_operator(parent_population,probability_crossover,probability_mutation,crossover
_type,mutation_type,UB,LB,Step,N,M,V,mu,mum,flw,slw,bi,mint,maxt,CPUTIME,time_limi
t);


   %% Combine current population and offspring in so called intermediate
   %% population

   intermediate_population = [offspring_population(:,1:M+V); population(:,1:M+V)];

   %% Sort the intermediate population
   sorted_intermediate_population = ...
```

```
    non_domination_and_crowding_sort(intermediate_population,N,M,V);

  %% Selection of individuals for next generation based on rank and
  %% crowding distance
  population = replace_chromosome(sorted_intermediate_population,N,M,V);

  %% Display algorithm progress


  generation=generation+1;
  CPUTIME=CPUTIME+toc;
end

Total_time_for_optimization = CPUTIME;



%% Results

% Display computation time
fprintf('Total time for optimization is  %g seconds \n',Total_time_for_optimization);

% Termination reason

if generation==max_generation+1
  disp('Optimization stopped because maximum generation was reached.');
  reason=1;
elseif CPUTIME>=time_limit
  disp('Optimization stopped because maximum CPU time was reached.');
  reason=2;
end

%% Visualization

% Elite population plot

if want_plot
  if M==2
    plot(population(:,V + 1),population(:,V + 2),'*');
    title('Multi-objective optimization: final population');
    xlabel('Objective 1');
    ylabel('Objective 2');
  elseif M==3
    figure1=plot(population(:,V + 1),population(:,V + 2),'*');
    title('Multi-objective optimization: final population');
    xlabel('Objective 1');
    ylabel('Objective 2');
    menu('Click here to proceed to next figure','ok');
    plot(population(:,V + 1),population(:,V + 3),'*');
    title('Multi-objective optimization: final population');
    xlabel('Objective 1');
    ylabel('Objective 3');
    menu('Click here to proceed to next figure','ok');
```

```
        plot(population(:,V + 2),population(:,V + 3),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 2');
        ylabel('Objective 3');
        menu('Click here to proceed to next figure','ok');
        plot3(population(:,V + 1),population(:,V + 2),population(:,V + 3),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 1');
        ylabel('Objective 2');
        zlabel('Objective 3');
    end
end
```

NSGAINN.m

```
close
clc
tic

%% Initialize constants and main parameters
constants

%% Initialize the population
population                                                                    =
initialize_population(N,M,V,UB,LB,Step,flw,slw,bi,mint,maxt,CPUTIME,time_limit);

%% Sort the initialized population based on rank and crowding distance
population = non_domination_and_crowding_sort(population,N,M,V);


CPUTIME=toc;

%% Start the evolution process

while (generation<=max_generation)&&(CPUTIME<time_limit)
   tic

   %% Select the parents (based on tournament selection)
   parent_population = tournament_selection(population,M,V,pool_size);

   %% Generate offspring by crossover and mutation
   if CPUTIME/time_limit<0.8
      offspring_population                                                    =
genetic_operator(parent_population,probability_crossover,probability_mutation,crossover
_type,mutation_type,UB,LB,Step,N,M,V,mu,mum,flw,slw,bi,mint,maxt,CPUTIME,time_limi
t);
   else
      CPUTIME=CPUTIME+toc;
      crossover_type=7;
      Min_=min(population(:,V+1:V+M));
      offspring_population                                                    =
genetic_operator2(parent_population,probability_crossover,probability_mutation,crossove
r_type,mutation_type,UB,LB,Step,N,M,V,mu,mum,CPUTIME,time_limit,Min_,flw,slw,bi,mi
nt,maxt);
      CPUTIME=offspring_population(length(offspring_population(:,1)),1);
      tic
      offspring_population(length(offspring_population(:,1)),:)=[];
   end

   %% Combine current population and offspring in so called intermediate
   %% population
   intermediate_population = [population(:,1:M+V);offspring_population(:,1:M+V)];

   %% Sort the intermediate population

   intermediate_population                                                    =
non_domination_and_crowding_sort(intermediate_population,N,M,V);
```

```
    %% Selection of individuals for next generation based on rank and
    %% crowding distance
    population = replace_chromosome(intermediate_population,N,M,V);

    %% Display algorithm progress

    generation=generation+1;
    CPUTIME=CPUTIME+toc;
end

Total_time_for_optimization = CPUTIME;

%% Results

% Display computation time
fprintf('Total time for optimization is  %g seconds \n',Total_time_for_optimization);

% Termination reason

if generation==max_generation+1
    disp('Optimization stopped because maximum generation was reached.');
    reason=1;
elseif CPUTIME>=time_limit
    disp('Optimization stopped because maximum CPU time was reached.');
    reason=2;
end

%% Visualization

% Elite population plot

if want_plot
    if M==2
        plot(population(:,V + 1),population(:,V + 2),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 1');
        ylabel('Objective 2');
    elseif M==3
        figure1=plot(population(:,V + 1),population(:,V + 2),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 1');
        ylabel('Objective 2');
        menu('Click here to proceed to next figure','ok');
        plot(population(:,V + 1),population(:,V + 3),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 1');
        ylabel('Objective 3');
        menu('Click here to proceed to next figure','ok');
        plot(population(:,V + 2),population(:,V + 3),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 2');
        ylabel('Objective 3');
        menu('Click here to proceed to next figure','ok');
```

```
        plot3(population(:,V + 1),population(:,V + 2),population(:,V + 3),'*');
        title('Multi-objective optimization: final population');
        xlabel('Objective 1');
        ylabel('Objective 2');
        zlabel('Objective 3');
    end
end
```

PLAGUE.m

```
tic
clc

%% Initialize constants and main parameters
PLAGUE_constants
Min_=zeros(M,1);

%% Initialize the population
population                                                                    =
initialize_population(N,M,V,UB,LB,Step,flw,slw,bi,mint,maxt,CPUTIME,time_limit);


%% Sort the initialized population based on rank and crowding distance
population = non_domination_and_crowding_sort(population,N,M,V);


CPUTIME=toc;

%% Start the evolution process

while (generation<=max_generation)&&(CPUTIME<time_limit)
   tic

      population=population(:,1:M+V);

population=PLAGUE_genetic_operator2(population,probability_crossover,probability_mut
ation,crossover_type,mutation_type,UB,LB,Step,4*N,M,V,mu,mum,CPUTIME,time_limit,fl
w,slw,bi,mint,maxt);

      CPUTIME=CPUTIME+toc;
      tic

      %% Sort the intermediate population
      PLAGUENOW


   %% Display algorithm progress
   generation=generation+1;
   CPUTIME=CPUTIME+toc;
end

if CPUTIME>10000
   CPUTIME=CPUTIME-10000;
end
Total_time_for_optimization = CPUTIME;



%% Results
```

```
% Display computation time
fprintf('Total time for optimization is  %g seconds \n',Total_time_for_optimization);

% Termination reason

if generation==max_generation+1
   disp('Optimization stopped because maximum generation was reached.');
   reason=1;
elseif CPUTIME>=time_limit-1
   disp('Optimization stopped because maximum CPU time was reached.');
   reason=2;
end

%% Visualization

% Elite population plot

if want_plot
   if M==2
      plot(population(:,V + 1),population(:,V + 2),'*');
      title('Multi-objective optimization: final population');
      xlabel('Objective 1');
      ylabel('Objective 2');
   elseif M==3
      figure1=plot(population(:,V + 1),population(:,V + 2),'*');
      title('Multi-objective optimization: final population');
      xlabel('Objective 1');
      ylabel('Objective 2');
      menu('Click here to proceed to next figure','ok');
      plot(population(:,V + 1),population(:,V + 3),'*');
      title('Multi-objective optimization: final population');
      xlabel('Objective 1');
      ylabel('Objective 3');
      menu('Click here to proceed to next figure','ok');
      plot(population(:,V + 2),population(:,V + 3),'*');
      title('Multi-objective optimization: final population');
      xlabel('Objective 2');
      ylabel('Objective 3');
      menu('Click here to proceed to next figure','ok');
      plot3(population(:,V + 1),population(:,V + 2),population(:,V + 3),'*');
      title('Multi-objective optimization: final population');
      xlabel('Objective 1');
      ylabel('Objective 2');
      zlabel('Objective 3');
   end
end
```

Constants.m

```
%% Population parameters

N = 100;          % Population size
M = 2;           % Number of objectives
V = 20;           % Number of decision variables

%% Variables bounds
UB=[];
LB=[];
Step=[];
for i=1:V
LB=[LB,-1.0];       %Lower bounds
UB=[UB,1.0];        %Upper bounds
Step=[Step,0];
end

%% Genetic operation parameters

crossover_type=1;            %Crossover type
probability_crossover=0.9;     %Probability of crossover
mu = 20;       %The distribution indices for crossover operator

mutation_type=12;             %Mutation type
probability_mutation=1/V;      %Probability of mutation
mum =20;      %The distribution indices for mutation operators


%% Genetic algorithm parameters

max_generation = 2500;                  %Maximum generation
pool_size = round(N);    %Pool size for tournament selection


%% Initialization

generation=1;


%% Termination criterion (please be careful changing this part)

termination_by_generation=0;
termination_by_cputime=0;
CPUTIME=0;
time_limit=60;

%% Others
want_plot=0;
```

Constants2.m

```
%% Initialize the variables
```

```
%% Population parameters

N = 100;          % Population size
M = 3;           % Number of objectives
V = 20;           % Number of decision variables


%% Variables bounds
UB=[];
LB=[];
Step=[];
for i=1:V
LB=[LB,0];       %Lower bounds
UB=[UB,1];         %Upper bounds
Step=[Step,0];
end


%% Genetic operation parameters

crossover_type=1;            %Crossover type
probability_crossover=0.9;     %Probability of crossover
mu = 20;       %The distribution indices for crossover operator

mutation_type=12;             %Mutation type
probability_mutation=1/V;      %Probability of mutation
mum =20;      %The distribution indices for mutation operators


%% Genetic algorithm parameters

max_generation = 250000;                  %Maximum generation
pool_size = round(N);     %Pool size for tournament selection


%% Initialization

generation=1;

%% Termination criterion (please be careful changing this part)

termination_by_generation=0;
termination_by_cputime=1;
CPUTIME=0;
time_limit=60);

%% Others
want_plot=0;
```

---

crossover.m

```
%% Crossover
```

```
%% Crossover type is chosen giving user choice

%% One point crossover

if crossover_type==2
   crossover_point=max(round((rand(1)*V)),1);
   for vars=1:crossover_point
      child_1(vars)=parent_1(vars);
      child_2(vars)=parent_2(vars);
   end
   for vars=crossover_point:V
      child_1(vars)=parent_2(vars);
      child_2(vars)=parent_1(vars);
   end


   %% Two points crossover

elseif crossover_type==3
   crossover_point1=min(max(round((rand(1,2)*V)),1));
   crossover_point2=max(max(round((rand(1,2)*V)),1));
   for vars=1:crossover_point1
      child_1(vars)=parent_1(vars);
      child_2(vars)=parent_2(vars);
   end
   for vars=crossover_point1:crossover_point2
      child_1(vars)=parent_2(vars);
      child_2(vars)=parent_1(vars);
   end
   for vars=crossover_point2:V
      child_1(vars)=parent_1(vars);
      child_2(vars)=parent_2(vars);
   end

   %% Scattered/random crossover

elseif crossover_type==4
   for vars=1:V
      if rand(1)<0.5
         child_1(vars)=parent_1(vars);
         child_2(vars)=parent_2(vars);
      else
         child_1(vars)=parent_2(vars);
         child_2(vars)=parent_1(vars);
      end
   end

   %% SBX (Simulated Binary Crossover)

elseif crossover_type==1
   u = rand(1,V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<0.5
```

```
        diff=abs(parent_1(vars)-parent_2(vars));
        if diff==0
           alpha=2;
        else
           beta=1+2*min([UB(vars)-parent_1(vars);parent_1(vars)-
LB(vars);parent_2(vars)-LB(vars);UB(vars)-parent_2(vars)])/diff;
           alpha=2-beta^(-mu-1);
        end
        if u(vars) <= (1/alpha)
           bq = (alpha*u(vars))^(1/(mu+1));
        else
           bq = (1/(2- alpha* u(vars)))^(1/(mu+1));
        end
        child_1(vars) = 0.5*(parent_1(vars)+parent_2(vars) - bq*abs(parent_1(vars)-
parent_2(vars)));
        child_2(vars) = 0.5*(parent_1(vars)+parent_2(vars) + bq*abs(parent_1(vars)-
parent_2(vars)));
        if Step(vars)~=0
           child_1(vars)=Step(vars)*round(child_1(vars)/Step(vars));
           child_2(vars)=Step(vars)*round(child_2(vars)/Step(vars));
        end
     end
  end
elseif crossover_type==5
  mu=10+round(10*rand(1));
  u = rand(1,V);
  random1=rand(1,V);
  for vars = 1 : V
     if random1(vars)<0.5
        % Generate a random number
        u = rand(1);
        if u <= 0.5
           bq = (2*u)^(1/(mu+1));
        else
           bq = (1/(2*(1 - u)))^(1/(mu+1));
        end
        child_1(vars) = 0.5*(parent_1(vars)+parent_2(vars) - bq*abs(parent_1(vars)-
parent_2(vars)));
        child_2(vars) = 0.5*(parent_1(vars)+parent_2(vars) + bq*abs(parent_1(vars)-
parent_2(vars)));
        if Step(vars)~=0
           child_1(vars)=Step(vars)*round(child_1(vars)/Step(vars));
           child_2(vars)=Step(vars)*round(child_2(vars)/Step(vars));
        end
        if child_1(vars) > UB(vars)
           child_1(vars) = UB(vars);
        elseif child_1(vars) < LB(vars)
           child_1(vars) = LB(vars);
        end
        if child_2(vars) > UB(vars)
           child_2(vars) = UB(vars);
        elseif child_2(vars) < LB(vars)
           child_2(vars) = LB(vars);
        end
```

```
      end
   end
elseif crossover_type==6
   u = rand(1,V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<0.5
         % Generate a random number
         u = rand(1);
         if u <= 0.5
            bq = (2*u)^(1/(mu+1));
         else
            bq = (1/(2*(1 - u)))^(1/(mu+1));
         end
         child_1(vars) = 0.5*(parent_1(vars)+parent_2(vars) - bq*abs(parent_1(vars)-
parent_2(vars)));
         child_2(vars) = 0.5*(parent_1(vars)+parent_2(vars) + bq*abs(parent_1(vars)-
parent_2(vars)));
         if Step(vars)~=0
            child_1(vars)=Step(vars)*round(child_1(vars)/Step(vars));
            child_2(vars)=Step(vars)*round(child_2(vars)/Step(vars));
         end
         if child_1(vars) > UB(vars)
            child_1(vars) = UB(vars);
         elseif child_1(vars) < LB(vars)
            child_1(vars) = LB(vars);
         end
         if child_2(vars) > UB(vars)
            child_2(vars) = UB(vars);
         elseif child_2(vars) < LB(vars)
            child_2(vars) = LB(vars);
         end
      end
   end
elseif crossover_type==7
   u = rand(1,V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<0.5
         % Generate a random number
         u = rand(1,V);
         random1=rand(1,V);
         for vars = 1 : V
            if random1(vars)<0.5
               diff=abs(parent_1(vars)-parent_2(vars));
               if diff==0
                  alpha=2;
               else
                  beta=1+2*min([UB(vars)-parent_1(vars);parent_1(vars)-
LB(vars);parent_2(vars)-LB(vars);UB(vars)-parent_2(vars)])/diff;
                  alpha=2-beta^(-mu-1);
               end
               if u(vars) <= (1/alpha)
                  bq = (alpha*u(vars))^(1/(mu+1));
```

```
            else
               bq = (1/(2- alpha* u(vars)))^(1/(mu+1));
            end
            child_1(vars) = 0.5*(parent_1(vars)+parent_2(vars) - bq*abs(parent_1(vars)-
parent_2(vars)));
            child_2(vars) = 0.5*(parent_1(vars)+parent_2(vars) + bq*abs(parent_1(vars)-
parent_2(vars)));
            if Step(vars)~=0
               child_1(vars)=Step(vars)*round(child_1(vars)/Step(vars));
               child_2(vars)=Step(vars)*round(child_2(vars)/Step(vars));
            end
         end
      end
   else
      % Generate a random number
      u = rand(1,V);
      random1=rand(1,V);
      for vars = 1 : V
         if random1(vars)<0.5
            diff=abs(parent_1(vars)-parent_2(vars));
            if diff==0
               alpha=2;
            else
               beta=1+2*min([UB(vars)-parent_1(vars);parent_1(vars)-
LB(vars);parent_2(vars)-LB(vars);UB(vars)-parent_2(vars)])/diff;
               alpha=2-beta^(-mu-1);
            end
            if u(vars) <= (1/alpha)
               bq = (alpha*u(vars))^(1/(mu+1));
            else
               bq = (1/(2- alpha* u(vars)))^(1/(mu+1));
            end
            child_1b(vars)          =          0.5*(parent_1(vars)+parent_2(vars))          -
bq*abs(parent_1(vars)-parent_2(vars)));
            child_2b(vars)          =          0.5*(parent_1(vars)+parent_2(vars))          +
bq*abs(parent_1(vars)-parent_2(vars)));
            if Step(vars)~=0
               child_1b(vars)=Step(vars)*round(child_1(vars)/Step(vars));
               child_2b(vars)=Step(vars)*round(child_2(vars)/Step(vars));
            end
         end
      end
   end
   child_1b((V+1):(V+M))                                                                    =
evaluate_objective(child_1b,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
   child_2b((V+1):(V+M))                                                                    =
evaluate_objective(child_2b,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
end


child_1((V+1):(V+M))                                                                        =
evaluate_objective(child_1,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
child_2((V+1):(V+M))                                                                        =
```

```
evaluate_objective(child_2,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
```

---

crossover2.m

```
%%Crossover sorting

if rand2(1,pp)<(5*CPUTIME/time_limit-4)^0.5
   crossover2_1;
else
   crossover2_2;
end
```

---

crossover2_1

```
%%Crossover replacement SEAMO

infants=[child_1;child_2;child_1b;child_2b];
family=[parent_1(1:M+V);parent_2(1:M+V);infants];
family=non_domination_sort(family,M,V);



sumO=sum(family(3:6,V+M+1));
sumP=(sum(family(1:2,V+M+1)));
if min(infants(:,V+1:V+M))>Min_
```

```
   if (sumP>=3)
      if family(1,M+V+1)==2
         if family(3,M+V+1)==1
            family(1,:)=non_domination_partial_sort(family(1,:),family(3,:),M,V);
            if family(1,M+V+1)==2
               successful_crossover(1)=1;
            end
         end
         if family(4,M+V+1)==1
            family(1,:)=non_domination_partial_sort(family(1,:),family(4,:),M,V);
            if family(1,M+V+1)==2
               successful_crossover(2)=1;
            end
         end
         if family(5,M+V+1)==1
            family(1,:)=non_domination_partial_sort(family(1,:),family(5,:),M,V);
            if family(1,M+V+1)==2
               successful_crossover(3)=1;
            end
         end
         if family(6,M+V+1)==1
            family(1,:)=non_domination_partial_sort(family(1,:),family(6,:),M,V);
            if family(1,M+V+1)==2
               successful_crossover(4)=1;
            end
         end
```

```
        end
        if family(2,M+V+1)==2
            if family(3,M+V+1)==1
                family(2,:)=non_domination_partial_sort(family(2,:),family(3,:),M,V);
                if family(2,M+V+1)==2
                    successful_crossover(1)=1;
                end
            end
            if family(4,M+V+1)==1
                family(2,:)=non_domination_partial_sort(family(2,:),family(4,:),M,V);
                if family(2,M+V+1)==2
                    successful_crossover(2)=1;
                end
            end
            if family(5,M+V+1)==1
                family(2,:)=non_domination_partial_sort(family(2,:),family(5,:),M,V);
                if family(2,M+V+1)==2
                    successful_crossover(3)=1;
                end
            end
            if family(6,M+V+1)==1
                family(2,:)=non_domination_partial_sort(family(2,:),family(6,:),M,V);
                if family(2,M+V+1)==2
                    successful_crossover(4)=1;
                end
            end
        end
    end
else
    if (child_1(V+1:V+M)<Min_)~=0
        successful_crossover(1)=1;
        Min_=min([Min_;child_1(V+1:V+M)]);
    end
    if (child_2(V+1:V+M)<Min_)~=0
        successful_crossover(2)=1;
        Min_=min([Min_;child_2(V+1:V+M)]);
    end
    if (child_1b(V+1:V+M)<Min_)~=0
        successful_crossover(3)=1;
        Min_=min([Min_;child_1b(V+1:V+M)]);
    end
    if (child_2b(V+1:V+M)<Min_)~=0
        successful_crossover(4)=1;
        Min_=min([Min_;child_2b(V+1:V+M)]);
    end
end
```

---

crossover2_2

```
%% Crossover2_2

infants=[child_1;child_2;child_1b;child_2b];
family=[parent_1(1:M+V);parent_2(1:M+V);infants];
```

```
infants=family_sort(family(3:6,:),family(1:2,:),M,V);

successful_crossover=infants(1:4,M+V+1);
```

---

evaluate_objective.m

```
function f = evaluate_objective(x,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit)

% Function to evaluate the objective functions for the given input vector
% x. x has the decision variables
if V==19
 X=x(1:19)';
 X(20)=-1;
else
X=x(1:20)';
end
f = [];
%% Objective function one

a(1)=-1+2/(1+exp(-2*(flw(1,:)*X+bi(1))));
a(2)=-1+2/(1+exp(-2*(flw(2,:)*X+bi(2))));
a(3)=-1+2/(1+exp(-2*(flw(3,:)*X+bi(3))));
a(4)=-1+2/(1+exp(-2*(flw(4,:)*X+bi(4))));
a(5)=-1+2/(1+exp(-2*(flw(5,:)*X+bi(5))));
a(6)=-1+2/(1+exp(-2*(flw(6,:)*X+bi(6))));
a(7)=-1+2/(1+exp(-2*(flw(7,:)*X+bi(7))));
a(8)=-1+2/(1+exp(-2*(flw(8,:)*X+bi(8))));
a(9)=-1+2/(1+exp(-2*(flw(9,:)*X+bi(9))));
a(10)=-1+2/(1+exp(-2*(flw(10,:)*X+bi(10))));
a(11)=-1+2/(1+exp(-2*(flw(11,:)*X+bi(11))));
a(12)=-1+2/(1+exp(-2*(flw(12,:)*X+bi(12))));
a(13)=-1+2/(1+exp(-2*(flw(13,:)*X+bi(13))));
a(14)=-1+2/(1+exp(-2*(flw(14,:)*X+bi(14))));
a(15)=-1+2/(1+exp(-2*(flw(15,:)*X+bi(15))));
a(16)=-1+2/(1+exp(-2*(flw(16,:)*X+bi(16))));
a(17)=-1+2/(1+exp(-2*(flw(17,:)*X+bi(17))));
a(18)=-1+2/(1+exp(-2*(flw(18,:)*X+bi(18))));
a(19)=-1+2/(1+exp(-2*(flw(19,:)*X+bi(19))));
a(20)=-1+2/(1+exp(-2*(flw(20,:)*X+bi(20))));


A(1)=slw(1,:)*a'+bi(21);
A(2)=slw(2,:)*a'+bi(22);
A(3)=slw(3,:)*a'+bi(23);
A(4)=slw(4,:)*a'+bi(24);
A(5)=slw(5,:)*a'+bi(25);

for i=1:5
   Ascaled(i) = 0.5*(A(i)+1)*(maxt(i)-mint(i)) + mint(i);
end

f(1)=abs(Ascaled(4))*(1+max(Ascaled(5)/100,0));
f(2)=sum(Ascaled(1:3))*(1+max(Ascaled(5)/100,0));
```

```
count=getappdata(0,'count');
count=count+1;
setappdata(0,'count',count);
```

---

family_sort.m

```
function f = family_sort(x,y,M,V)

for i = 1 : 4
   j=1;
   % Number of individuals that dominate this individual
   dominated = 0;
   while (dominated == 0)&(j<=2)
      dom_less = 0;
      dom_equal = 0;
      dom_more = 0;
      for k = 1 : M
         if (x(i,V + k) < y(j,V + k))
            dom_less = dom_less + 1;
         elseif (x(i,V + k) == y(j,V + k))
            dom_equal = dom_equal + 1;
         else
            dom_more = dom_more + 1;
         end
      end
      if dom_less == 0 && dom_equal ~= M
         dominated = 1;
      end
      j=j+1;
   end
   if dominated == 0
      x(i,M + V + 1) = 1;
   else
      x(i,M+V+1)=2;
   end
end
f=x();
```

---

genetic_operator.m

```
function f =
genetic_operator(parent_population,probability_crossover,probability_mutation,crossover
_type,mutation_type,UB,LB,Step,N,M,V,mu,mum,flw,slw,bi,mint,maxt,CPUTIME,time_limi
t);

NP = length(parent_population(:,1));

p = 1;
pp=1;
randgen=ceil(NP*rand(2,ceil(N/2)));
while p < N+1
```

```
   num_parent_1 = randgen(1,pp);
   parent_1 = parent_population(num_parent_1,:);
   child_1=parent_1;
   num_parent_2 = randgen(2,pp);
   while isequal(num_parent_1,num_parent_2)
      num_parent_2 = ceil(NP*rand(1));
   end
   parent_2 = parent_population(num_parent_2,:);
   child_2=parent_2;
   if rand(1) < (probability_crossover)
      crossover
   end

   parent_3=child_1;
   mmutation
   child_1 = child_3;

   parent_3=child_2;
   mmutation
   child_2 = child_3;

   child(p,:) = child_1(1:M+V);
   child(p+1,:) = child_2(1:M+V);
   p = p + 2;
   pp=pp+1;
end

f = child;
```

---

genetic_operator_2.m

```
function f = genetic_operator2(parent_population,probability_crossover,probability_mutation,crossover_type,mutation_type,UB,LB,Step,N,M,V,mu,mum,CPUTIME,time_limit,Min_,flw,slw,bi,mint,maxt);

child=[];
NP = length(parent_population(:,1));
p = 1;
randgen=ceil(NP*rand(2,2*N));
rand2=rand(1,2*N);
pp=1;
count=0;
tic

while (p < N+1)&&(count<100*N*M)&&(CPUTIME<(time_limit-1))
   successful_crossover(1:4)=0;
   num_parent_1 = randgen(1,pp);
   parent_1 = parent_population(num_parent_1,:);
   child_1=parent_1;
   child_1b=parent_1;
   num_parent_2 = randgen(2,pp);
   while isequal(num_parent_1,num_parent_2)
```

```
      num_parent_2 = ceil(NP*rand(1));
    end
    parent_2 = parent_population(num_parent_2,:);
    child_2=parent_2;
    child_2b=parent_2;
    if rand(1) < (probability_crossover)
      crossover
    end

    parent_3=child_1;
    mmutation
    child_1 = child_3;

    parent_3=child_2;
    mmutation
    child_2 = child_3;
    parent_3=child_1b;
    mmutation
    child_1b = child_3;

    parent_3=child_2b;
    mmutation
    child_2b = child_3;

    crossover2
    if sum(successful_crossover)>0
      if successful_crossover(1)==1
        child(p,1:M+V) = child_1;
        p=p+1;
      elseif successful_crossover(2)==1;
        child(p,1:M+V) = child_2;
        p=p+1;
      elseif successful_crossover(3)==1;
        child(p,1:M+V) = child_1b;
        p=p+1;
      elseif successful_crossover(4)==1;
        child(p,1:M+V) = child_2b;
        p=p+1;
      end
    end

    pp=pp+1;
    if (mod(pp,2*N)==0)
      CPUTIME=CPUTIME+toc;
      tic;
      randgen(2,:)=ceil(NP*rand(1,2*N));
      rand2=rand(1,2*N);
      pp=1;
    end
    count=count+1;
end

CPUTIME=CPUTIME+toc;
```

```
cputime=ones(1,M+V)*CPUTIME;
f = [child;cputime];
```

---

initialize_population.m

```
function                                                    f                                                    =
initialize_population(N,M,V,UB,LB,Step,flw,slw,bi,mint,maxt,CPUTIME,time_limit)

%% This function initializes the population with N individuals and each
%% individual having M decision variables based on the selected problem.


%% Initialize the decision variables

RAND=rand(N,V);
for vars = 1 : V
   f(:,vars) = LB(vars)+(UB(vars)-LB(vars))*RAND(:,vars);
   if Step(vars)~=0
      f(:,vars)=Step(vars)*round(f(:,vars)/Step(vars));
   end
end

%% Evaluate the objective function
for i = 1 : N
   f(i,(V+1):(V+M))                                                                                    =
evaluate_objective(f(i,:),M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
end
```

---

mmutation.m

```
%% Mutation process


%% Mutation

if mutation_type==2
   for vars=1:V
      child_3(vars)=(1+(1-2*rand(1))*mum(vars))*parent_3(vars);
      if Step(vars)~=0
         child_3(vars)=Step(vars)*round(child_3(vars)/Step(vars));
      end
      if child_3(vars) > UB(vars)
         child_3(vars) = UB(vars);
      elseif child_3(vars) < LB(vars)
         child_3(vars) = LB(vars);
      end
   end

elseif mutation_type==1
   child_3 = parent_3(1:V);
   random1=rand(1,V);
   for vars = 1 : V
```

```
      if random1(vars)<probability_mutation
         delta=min([parent_3(vars)-LB(vars);UB(vars)-parent_3(vars)])/(UB(vars)-LB(vars));
         r = rand(1);
         if r<=0.5
            deltaq = ((2*r)^(1/(mum+1)) - 1)*(parent_3(vars)-LB(vars))/(UB(vars)-LB(vars));
         else
            deltaq = (1 - (2*(1 - r))^(1/(mum+1)))*(UB(vars)-parent_3(vars))/(UB(vars)-
LB(vars));
         end
         child_3(vars) = child_3(vars) + (UB(vars)-LB(vars))*deltaq;
         if Step(vars)~=0
            child_3(vars)=Step(vars)*round(child_3(vars)/Step(vars));
         end
      end
   end
elseif mutation_type==12
   child_3 = parent_3(1:V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<probability_mutation
         delta=min([parent_3(vars)-LB(vars);UB(vars)-parent_3(vars)])/(UB(vars)-LB(vars));
         r = rand(1);
         if r<0.5
            deltaq = (2*r+(1-2*r)*(1-delta)^(mum+1))^(1/(mum+1)) - 1;
         else
            deltaq = 1 - (2*(1 - r)+2*(r-0.5)*(1-delta)^(mum+1))^(1/(mum+1));
         end
         child_3(vars) = parent_3(vars) + (UB(vars)-LB(vars))*deltaq;
         if child_3(vars) > UB(vars)
            child_3(vars) = UB(vars);
         elseif child_3(vars) < LB(vars)
            child_3(vars) = LB(vars);
         end
         if Step(vars)~=0
            child_3(vars)=Step(vars)*round(child_3(vars)/Step(vars));
         end
      end
   end
elseif mutation_type==3
   mum=10+round(10*rand(1));
   child_3 = parent_3(1:V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<probability_mutation
         r=rand(1);
         if r < 0.5
            delta(vars) = (UB(vars)-LB(vars))*((2*r)^(1/(mum+1)) - 1);
         else
            delta(vars) = (UB(vars)-LB(vars))*(1 - (2*(1 - r))^(1/(mum+1)));
         end
         child_3(vars) = child_3(vars) + delta(vars);
         if Step(vars)~=0
            child_3(vars)=Step(vars)*round(child_3(vars)/Step(vars));
         end
```

```
      end
   end
elseif mutation_type==4
   child_3 = parent_3(1:V);
   random1=rand(1,V);
   for vars = 1 : V
      if random1(vars)<probability_mutation
         r=rand(1);
         if r < 0.5
            delta(vars) = (UB(vars)-LB(vars))*((2*r)^(1/(mum+1)) - 1);
         else
            delta(vars) = (UB(vars)-LB(vars))*(1 - (2*(1 - r))^(1/(mum+1)));
         end
         child_3(vars) = child_3(vars) + delta(vars);
         if Step(vars)~=0
            child_3(vars)=Step(vars)*round(child_3(vars)/Step(vars));
         end
         if child_3(vars) > UB(vars)
            child_3(vars) = UB(vars);
         elseif child_3(vars) < LB(vars)
            child_3(vars) = LB(vars);
         end
      end
   end
end


child_3((V+1):(V+M))                                                    =
evaluate_objective(child_3,M,V,flw,slw,bi,mint,maxt,CPUTIME,time_limit);
```

---

non_domination_and_crowding_sort.m

```
function f = non_domination_and_crowding_sort(x,N,M,V)


ND = length(x(:,1));
x=x(:,1:M+V);
x(:,M+V+1)=0;
front = 1;

F(front).f = [];
individuall = [];
for i = 1 : ND
   % Number of individuals that dominate this individual
   individuall(i).dominating = 0;
   % Individuals which this individual dominates
   individuall(i).dominated = [];
   for j = 1 : ND
      Sbetter=0;
      better = 0;
      equal = 0;
      worse = 0;
```

```
        for obj = 1 : M
            if (x(i,V + obj) > x(j,V + obj))
                worse = worse + 1;
            elseif (x(i,V + obj)== x(j,V + obj))
                equal = equal + 1;
            elseif (x(i,V + obj) < x(j,V + obj))
                better=better+1;
            end
        end
        if better == 0 && equal ~= M
            individuall(i).dominating = individuall(i).dominating + 1;
        elseif worse == 0 && better>0
            individuall(i).dominated = [individuall(i).dominated j];
        end
    end
    if individuall(i).dominating == 0
        x(i,M + V + 1) = 1;
        F(front).f = [F(front).f i];
    end
end


% Find the rank of each individual
size_=0;
while (~isempty(F(front).f))&&((size_)<N)
    Q = [];
    for i = 1 : length(F(front).f)
        if ~isempty(individuall(F(front).f(i)).dominated)
            for j = 1 : length(individuall(F(front).f(i)).dominated)
                individuall(individuall(F(front).f(i)).dominated(j)).dominating = ...
                    individuall(individuall(F(front).f(i)).dominated(j)).dominating - 1;
                if individuall(individuall(F(front).f(i)).dominated(j)).dominating == 0
                    x(individuall(F(front).f(i)).dominated(j),M + V + 1)=front + 1;
                    Q = [Q individuall(F(front).f(i)).dominated(j)];
                end
            end
        end
    end
    size_=size_+length(F(front).f);
    front =  front + 1;
    F(front).f = Q;
end

for i=1:ND
    if x(i,M+V+1)==0
        x(i,M+V+1)=front;
    end
end


last_front=front-1;


[temp,index_of_fronts] = sort(x(:,M + V + 1));
```

```
sorted_based_on_front = x(index_of_fronts,:);


%% Find the crowding distance for each individuall in each front
current_index = 0;

%For each front up to 5
for front = 1 : min(last_front,500)
   y = [];
   sorted_based_on_objective = [];
   previous_index = current_index;

   %For each individual in this front
   length_front=(length(F(front).f));
   y = sorted_based_on_front(current_index+1:current_index+length_front,:);
   current_index = current_index + length_front;


   % Sort each individuall based on the objective

   for obj = 1 : M
      [temp, index_of_objectives] = sort(y(:,V + obj));
      sorted_based_on_objective = y(index_of_objectives,:);
      f_max= sorted_based_on_objective(length_front, V + obj);
      f_min = sorted_based_on_objective(1, V + obj);
      y(index_of_objectives(length_front),M + V + 1 + obj)= Inf;
      y(index_of_objectives(1),M + V + 1 + obj) = Inf;

      for j = 2 : length(index_of_objectives) - 1
         next_obj  = sorted_based_on_objective(j + 1,V + obj);
         previous_obj  = sorted_based_on_objective(j - 1,V + obj);
         if (f_max - f_min == 0)
            y(index_of_objectives(j),M + V + 1 + obj) = Inf;
         else
            y(index_of_objectives(j),M + V + 1 + obj) = (next_obj - previous_obj)/(f_max-
f_min);
         end
      end
   end
   distance= zeros(length_front,1);
   for obj = 1 : M
      distance(:) = distance(:) + y(:,M + V + 1 + obj);
   end
   y(:,M + V + 2) = distance;
   y = y(:,1 : M + V + 2);
   z(previous_index+1:current_index,:) = y;
end

f = z();
```

non_domination_partial_sort.m

```
function f = non_domination_partial_sort(x,y,M,V)


NT = length(x(:,1));
ND = length(y(:,1));

for i = 1 : NT
   j=1;
   % Number of individuals that dominate this individual
   dominated = 0;
   while (dominated == 0)&(j<=ND)
      dom_less = 0;
      dom_equal = 0;
      dom_more = 0;
      for k = 1 : M
         if (x(i,V + k) < y(j,V + k))
            dom_less = dom_less + 1;
         elseif (x(i,V + k) == y(j,V + k))
            dom_equal = dom_equal + 1;
         else
            dom_more = dom_more + 1;
         end
      end
      if dom_less == 0 && dom_equal ~= M
         dominated = 1;
      end
      j=j+1;
   end
   if dominated == 0
      x(i,M + V + 1) = 1;
   else
      x(i,M+V+1)=2;
   end
end
f=x();
```

---

non_domination_sort.m

```
function f = non_domination_sort(x,M,V)

ND = length(x(:,1));

for i = 1 : ND
   j=1;
   % Number of individuals that dominate this individual
   dominated = 0;
   while (dominated == 0)&(j<=ND)
      dom_less = 0;
      dom_equal = 0;
      dom_more = 0;
      for k = 1 : M
         if (x(i,V + k) < x(j,V + k))
            dom_less = dom_less + 1;
```

```
            elseif (x(i,V + k) == x(j,V + k))
               dom_equal = dom_equal + 1;
            else
               dom_more = dom_more + 1;
            end
         end
         if dom_less == 0 && dom_equal ~= M
            dominated = 1;
         end
         j=j+1;
      end
      if dominated == 0
         x(i,M + V + 1) = 1;
      else
         x(i,M+V+1)=2;
      end
   end
   f=x();
```

tournament_selection.m

```
function f = tournament_selection(population,M,V,pool_size)

NT = length(population(:,1));

randtr=ceil(NT*rand(2,pool_size));
for i = 1 : pool_size
   %% Generate (tour_size) different individuals
   candidate=randtr(:,i);
   while candidate(2)==candidate(1)
      candidate(2) = ceil(NT*rand(1));
   end

   c_obj_rank= population(candidate(:),M+V+1);
   c_obj_distance = population(candidate(:),M+V+2);


   %% Find fittest individual

   %Find minimal rank individual(s)
   min_candidate = find(c_obj_rank == min(c_obj_rank));

   if length(min_candidate) == 1
      % If there is only one minimal rank individual, it is chosen
      f(i,:) = population(candidate(min_candidate(1)),:);

   elseif length(min_candidate) ~= 1
      % If several individuals have minimal rank, the one with highest
      % crowding distance is chosen
      max_candidate = ...
         find(c_obj_distance(min_candidate) == max(c_obj_distance(min_candidate)));
      if length(max_candidate) ~= 1
         % If again several individuals have same distances, individuals
         % is randomly chosen between those
         max_candidate = max_candidate(max(round(rand(1)*length(max_candidate)),1));
      end
      f(i,:) = population(candidate(min_candidate(max_candidate)),:);
   end
end
```

PLAGUE_constants.m

```
%% Initialize the variables

%% Population parameters

N = 80;          % Population size
M = 2;           % Number of objectives
V = 20;           % Number of decision variables
```

```
%% Variables bounds
UB=[];
LB=[];
Step=[];
for i=1:V
LB=[LB,-1.0];        %Lower bounds
UB=[UB,1.0];          %Upper bounds
Step=[Step,0];
end


%% Genetic operation parameters

crossover_type=1;              %Crossover type
probability_crossover=0.9;     %Probability of crossover
mu = 20;        %The distribution indices for crossover operator

mutation_type=12;              %Mutation type
probability_mutation=1/V;      %Probability of mutation
mum =20;       %The distribution indices for mutation operators


%% Genetic algorithm parameters

max_generation = 250000;                    %Maximum generation
pool_size = round(N);     %Pool size for tournament selection


%% Initialization

generation=1;


%% Termination criterion (please be careful changing this part)

termination_by_generation=0;
termination_by_cputime=1;
CPUTIME=0;
time_limit=getappdata(0,'time');
last_population=[];
%% Others
want_plot=0;
```

PLAGUE_family_sorting.m

```
%%Crossover replacement

infants=[child_1;child_2];
family=[parent_1(1:M+V);parent_2(1:M+V);child_1;child_2];
family=non_domination_sort(family,M,V);
```

```
if p>0
   if family(1,M+V+1)==2
      population(num_parent_1,:)=[];
      p=p-1;
      if family(2,M+V+1)==2
         if num_parent_2>num_parent_1
            population(num_parent_2-1,:)=[];
            p=p-1;
         else
            population(num_parent_2,:)=[];
            p=p-1;
         end
      end
   elseif family(2,M+V+1)==2
      population(num_parent_2,:)=[];
      p=p-1;
   end
end
if family(3,M+V+1)==1
   population=[population; child_1];
   p=p+1;
end
if family(4,M+V+1)==1
   population=[population; child_2];
   p=p+1;
end
```

PLAGUE_genetic_operator2.m

```
function f = PLAGUE_genetic_operator2
(parent_population,probability_crossover,probability_mutation,crossover_type,mutation_t
ype,UB,LB,Step,wished_size,M,V,mu,mum,CPUTIME,time_limit,flw,slw,bi,mint,maxt);


NP = length(parent_population(:,1));
population=parent_population;
p = 1;
randgen=ceil(NP*rand(2,40));
pp=1;

while (p < wished_size+1)
   successful_crossover(1:4)=0;
   num_parent_1 = randgen(1,pp);
   if num_parent_1>NP
      num_parent_1 = ceil(NP*rand(1));
   end
   parent_1 = population(num_parent_1,:);
   child_1=parent_1;
   num_parent_2 = randgen(2,pp);
   if num_parent_2>NP
      num_parent_2 = ceil(NP*rand(1));
   end
   while isequal(num_parent_1,num_parent_2)
```

```
      num_parent_2 = ceil(NP*rand(1));
   end
   parent_2 = population(num_parent_2,:);
   child_2=parent_2;
   if rand(1) < (probability_crossover)
      crossover
   end

   parent_3=child_1;
   mmutation
   child_1 = child_3;

   parent_3=child_2;
   mmutation
   child_2 = child_3;

   PLAGUE_family_sorting;
   NP=length(population(:,1));
   pp=pp+1;
   if (mod(pp,40)==0)
      NP=length(population(:,1));
      randgen=ceil(NP*rand(2,40));
      pp=1;
   end
end

f = population;
```

PLAGUE_non_domination_and_crowding_sort.m

```
function f = PLAGUE_non_domination_and_crowding_sort(x,N,M,V)

ND = length(x(:,1));
x=x(:,1:M+V);
x(:,M+V+1)=0;
front = 1;

F(front).f = [];
individuall = [];
for i = 1 : ND
   % Number of individuals that dominate this individual
   individuall(i).dominating = 0;
   % Individuals which this individual dominates
   individuall(i).dominated = [];
   for j = 1 : ND
      Sbetter=0;
      better = 0;
      equal = 0;
      worse = 0;
      for obj = 1 : M
         if (x(i,V + obj) > x(j,V + obj))
            worse = worse + 1;
```

```
            elseif (x(i,V + obj)== x(j,V + obj))
                equal = equal + 1;
            elseif (x(i,V + obj) < x(j,V + obj))
                better=better+1;
            end
        end
        if better == 0 && equal ~= M
            individuall(i).dominating = individuall(i).dominating + 1;
        elseif worse == 0 && better>0
            individuall(i).dominated = [individuall(i).dominated j];
        end
    end
    if individuall(i).dominating == 0
        x(i,M + V + 1) = 1;
        F(front).f = [F(front).f i];
    end
end


% Find the rank of each individual
size_=0;
while (~isempty(F(front).f))&&((size_)<N)
    Q = [];
    for i = 1 : length(F(front).f)
        if ~isempty(individuall(F(front).f(i)).dominated)
            for j = 1 : length(individuall(F(front).f(i)).dominated)
                individuall(individuall(F(front).f(i)).dominated(j)).dominating = ...
                    individuall(individuall(F(front).f(i)).dominated(j)).dominating - 1;
                if individuall(individuall(F(front).f(i)).dominated(j)).dominating == 0
                    x(individuall(F(front).f(i)).dominated(j),M + V + 1)=front + 1;
                    Q = [Q individuall(F(front).f(i)).dominated(j)];
                end
            end
        end
    end
    size_=size_+length(F(front).f);
    front =  front + 1;
    F(front).f = Q;
end

for i=1:ND
    if x(i,M+V+1)==0
        x(i,M+V+1)=front;
    end
end


last_front=front-1;


[temp,index_of_fronts] = sort(x(:,M + V + 1));
sorted_based_on_front = x(index_of_fronts,:);
```

```
%% Find the crowding distance for each individuall in each front
current_index = 0;

%For each front up to 5
for front = 1 : min(last_front,5)
   y = [];
   sorted_based_on_objective = [];
   previous_index = current_index;

   %For each individual in this front
   length_front=(length(F(front).f));
   y = sorted_based_on_front(current_index+1:current_index+length_front,:);
   current_index = current_index + length_front;


   % Sort each individuall based on the objective

   for obj = 1 : M
      [temp, index_of_objectives] = sort(y(:,V + obj));
      sorted_based_on_objective = y(index_of_objectives,:);
      f_max= sorted_based_on_objective(length_front, V + obj);
      f_min = sorted_based_on_objective(1, V + obj);
      y(index_of_objectives(length_front),M + V + 1 + obj)= Inf;
      y(index_of_objectives(1),M + V + 1 + obj) = Inf;

      for j = 2 : length(index_of_objectives) - 1
         next_obj  = sorted_based_on_objective(j + 1,V + obj);
         previous_obj  = sorted_based_on_objective(j - 1,V + obj);
         if (f_max - f_min == 0)
            y(index_of_objectives(j),M + V + 1 + obj) = Inf;
         else
            y(index_of_objectives(j),M + V + 1 + obj) = (next_obj - previous_obj)/(f_max-
f_min);
         end
      end
   end
   distance= zeros(length_front,1);
   for obj = 1 : M
      distance(:) = distance(:) + y(:,M + V + 1 + obj);
   end
   y(:,M + V + 2) = distance;
   y = y(:,1 : M + V + 2);
   z(previous_index+1:current_index,:) = y;
end

%For front higher than 5
if last_front>5
   y = [];
   sorted_based_on_objective = [];
   previous_index = current_index;

   %For each individual in this front
   length_front=ND-current_index;
```

```
    y = sorted_based_on_front(current_index+1:current_index+length_front,:);
    current_index = current_index + length_front;


    % Sort each individuall based on the objective

    for obj = 1 : M
        [temp, index_of_objectives] = sort(y(:,V + obj));
        sorted_based_on_objective = y(index_of_objectives,:);
        f_max= sorted_based_on_objective(length_front, V + obj);
        f_min = sorted_based_on_objective(1, V + obj);
        y(index_of_objectives(length_front),M + V + 1 + obj)= Inf;
        y(index_of_objectives(1),M + V + 1 + obj) = Inf;

        for j = 2 : length(index_of_objectives) - 1
            next_obj  = sorted_based_on_objective(j + 1,V + obj);
            previous_obj  = sorted_based_on_objective(j - 1,V + obj);
            if (f_max - f_min == 0)
                y(index_of_objectives(j),M + V + 1 + obj) = Inf;
            else
                y(index_of_objectives(j),M + V + 1 + obj) = (next_obj - previous_obj)/(f_max-
f_min);
            end
        end
    end
    distance= zeros(length_front,1);
    for obj = 1 : M
        distance(:) = distance(:) + y(:,M + V + 1 + obj);
    end
    y(:,M + V + 2) = distance;
    y = y(:,1 : M + V + 2);
    z(previous_index+1:current_index,:) = y;
end
f = z();
```

PLAGUENOW.m

```
%%PLAGUE

if CPUTIME<time_limit-1.5
    last_population=unique(population,'rows');

    %% Main population

sorted_population=PLAGUE_non_domination_and_crowding_sort(last_population,ceil(0.
85*N),M,V);
    main_population= replace_chromosome(sorted_population,ceil((1-0.05*M)*N),M,V);

    %% Specialist population
    survivors=[];
    for obj=1:M
        [temp,index_of_fronts] = sort(sorted_population(:,V+obj));
        sorted_based_on_objective = sorted_population(index_of_fronts(:),:);
        survivors=[survivors;sorted_based_on_objective(1:round(0.05*N),:)];
```

```
    end

    %% Diversity population
    h=ceil(length(population(:,1))*rand(1,round(N/20)));
    Hazard=population(h,:);

    population=[survivors(:,1:M+V);main_population(:,1:M+V);Hazard];
    CPUTIME=CPUTIME+toc;
    tic
end
if CPUTIME>time_limit-1.5
    if isempty(last_population)
        last_population=population;
    end

sorted_population=non_domination_and_crowding_sort(last_population,length(last_popu
lation(:,1)),M,V);
    population=[];
    for indv=1:length(sorted_population(:,1))
        if sorted_population(indv,M+V+1)==1
            population=[population;sorted_population(indv,:)];
        end
    end
    CPUTIME=CPUTIME+10000;
End
```

---

replace_chromosome.m

```
function f  = replace_chromosome(intermediate_population,N,M,V)

NR=length(intermediate_population(:,1));
f=[];
%% Get the index for the population sort based on the rank
[temp,index] = sort(intermediate_population(:,M + V + 1));

%% Now sort the individuals based on the index
sorted_population = intermediate_population(index,:);


%% Find the maximum rank in the current population
max_rank = sorted_population(NR,M+V+1);

%% Start adding each front based on rank and crowing distance until the
%% whole population is filled.

previous_index = 0;
for i = 1 : max_rank
    current_index = find(sorted_population(:,M + V + 1) == i,1,'last');
    if current_index < N
        % All individuals of current rank are taken
        f(previous_index + 1 : current_index, :) = ...
            sorted_population(previous_index + 1 : current_index, :);
```

```
    elseif current_index==N
        % Same as before, but whole new population is filled so loop ends
        f(previous_index + 1 : current_index, :) = ...
            sorted_population(previous_index + 1 : current_index, :);
        return
    elseif current_index > N
        % Individuals of current rank are chosen based on theri crowding
        % distance

        remaining = N - previous_index;
        temp_pop= sorted_population(previous_index + 1 : current_index, :);
        [temp,temp_sort_index]=sort(temp_pop(:, M + V + 2),'descend');
        f(previous_index + 1:previous_index+remaining,:) =
temp_pop(temp_sort_index(1:remaining),:);
        return
    end
    previous_index = current_index;
end
```

---

tournament_selection.m

```
function f = tournament_selection(population,M,V,pool_size)

NT = length(population(:,1));

randtr=ceil(NT*rand(2,pool_size));
for i = 1 : pool_size
    %% Generate (tour_size) different individuals
    candidate=randtr(:,i);
    while candidate(2)==candidate(1)
        candidate(2) = ceil(NT*rand(1));
    end

    c_obj_rank= population(candidate(:),M+V+1);
    c_obj_distance = population(candidate(:),M+V+2);


    %% Find fittest individual

    %Find minimal rank individual(s)
    min_candidate = find(c_obj_rank == min(c_obj_rank));

    if length(min_candidate) == 1
        % If there is only one minimal rank individual, it is chosen
        f(i,:) = population(candidate(min_candidate(1)),:);

    elseif length(min_candidate) ~= 1
        % If several individuals have minimal rank, the one with highest
        % crowding distance is chosen
        max_candidate = ...
            find(c_obj_distance(min_candidate) == max(c_obj_distance(min_candidate)));
        if length(max_candidate) ~= 1
```

```
            % If again several individuals have same distances, individuals
            % is randomly chosen between those
            max_candidate = max_candidate(max(round(rand(1)*length(max_candidate)),1));
        end
        f(i,:) = population(candidate(min_candidate(max_candidate)),:);
    end
end
```

APPENDIX B:

SIMULATED BINARY CROSSOVER AND POLYNOMIAL MUTATION

Possible genetic operators are numerous, such as single-point or two-point crossovers, or mutation based on a probability distribution. For building applications though, the use of real parameters is recommended; in that case, the most efficient genetic operators are the Simulated Binary Crossover and the Polynomial mutation. The main asset of these genetics operators is that offspring chromosomes can take any value between parents' variables, according on a distribution probability. They are often associated with NSGA-II and are one reason of its efficiency. They re computed as follows:

Simulated Binary Crossover (SBX)

Generate a random number u $\in$ [0,1]

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c+1}} \\ \left(\frac{1}{2\times(1-u)}\right)^{\frac{1}{\eta_c+1}} \end{cases} \quad \text{If } u < 0.5$$

$c_1 = 0.5 \times ((P_1 + P_2) - \beta \mid P_1 + P_2 \mid)$

$c_2 = 0.5 \times ((P_1 + P_2) + \beta \mid P_1 + P_2 \mid)$

Polynomial crossover

Generate a random number r $\in$ [0,1]

$$\delta = \begin{cases} (2u)^{\frac{1}{\eta_m+1}} - 1 & \text{If } r < 0.5 \\ 1 - \left(\frac{1}{2\times(1-u)}\right)^{\frac{1}{\eta_m+1}} & \end{cases}$$

$c_1 = P_1 + \delta_x (V_{max} - V_{min})$

where c_i and P_i are respectively offspring and parents variables,

ηc and ηm are respectively crossover and mutation distribution indices, and

V_{max} and V_{min} are variable upper and lower bounds.

For multiple variables problems, SBX is applied on each variable with a probability of 0.5, and Polynomial mutation is applied on each variable according to mutation probability. Closeness between parents and offspring is controlled by distribution indices ηc and ηm, with a low number giving a high probability to create far children. Different distribution indices values can be found in the literature; in this thesis, both crossover and mutation distribution indices have been chosen as 20, as in the Deb (2000), which produces offspring relatively close to their parents.

APPENDIX C:

CODE USED FOR GENOPT RUNS

```
package genopt.algorithm;

import genopt.GenOpt;
import genopt.io.InputFormatException;
import genopt.lang.OptimizerException;
import genopt.algorithm.util.math.Point;
import genopt.algorithm.util.math.Fun;
import genopt.simulation.SimulationInputException;
import java.io.IOException;
import java.util.TreeMap;

/** Class for doing a parametric run where one parameter
  * is perturbed at a time while the others are fixed.
  * Linear and logarithmic spacing can be selected for each
  * parameter independently.<BR>
  *
  * <P><I>This project was carried out at:</I>
  * <UL><LI><A HREF="http://www.lbl.gov">
  * Lawrence Berkeley National Laboratory (LBNL)</A>,
  * <A HREF="http://simulationresearch.lbl.gov">
  * Simulation Research Group</A>,</UL></LI>
  * <I>and supported by</I><UL>
  * <LI>the <A HREF="http://www.energy.gov">
  * U.S. Department of Energy (DOE)</A>,
  * <LI>the <A HREF="http://www.satw.ch">
  * Swiss Academy of Engineering Sciences (SATW)</A>,
  * <LI>the Swiss National Energy Fund (NEFF), and
  * <LI>the <A HREF="http://www.snf.ch">
  * Swiss National Science Foundation (SNSF)</A></UL></LI><P>
  *
  * Copyright (c) 1998-2003 The Regents of the University of California
 * (through Lawrence Berkeley National Laboratory),
```

```
 * subject to receipt of any required approvals from U.S. Department of Energy.
 *
 * @author <A HREF="mailto:MWetter@lbl.gov">Michael Wetter</A>
 *
 * @version GenOpt(R) 2.0.0 (Jan. 5, 2004)<P>
 */

/* Redistribution not allowed.
   Product and company names mentioned herein may be the trademarks of their
   respective owners. Any rights not expressly granted herein are reserved.

   NOTICE: The Government is granted for itself and others acting on its
   behalf a paid-up, nonexclusive, irrevocable, worldwide license in this
   data to reproduce, prepare derivative works, and perform publicly and
   display publicly. Beginning five (5) years after the date permission
   to assert copyright is obtained from the U.S. Department of Energy,
   and subject to any subsequent five (5) year renewals, the Government is
   granted for itself and others acting on its behalf a paid-up, nonexclusive,
   irrevocable, worldwide license in this data to reproduce, prepare
   derivative works, distribute copies to the public, perform publicly
   and display publicly, and to permit others to do so.
   NEITHER THE UNITED STATES NOR THE UNITED STATES DEPARTMENT OF ENERGY,
   NOR ANY OF THEIR EMPLOYEES, MAKES ANY WARRANTY, EXPRESS OR IMPLIED,
   OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY,
   COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT,
   OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE
   PRIVATELY OWNED RIGHTS
 */

public class Parametric extends Optimizer
{
   /** Constructor
    * @param genOptData a reference to the GenOpt object.<BR>
    * <B>Note:</B> the object is used as a reference.
    *          Hence, the datas of GenOpt are modified
    *          by this Class.
```

```
 * @exception OptimizerException
 *@exception IOException if an I/O exception occurs
 * @exception Exception
 * @exception InputFormatException
 */
public Parametric(GenOpt genOptData)
      throws OptimizerException, IOException, Exception, InputFormatException
{
      super(genOptData, 0);
      dimCon = getDimensionContinuous();
      dimDis = getDimensionDiscrete();

      dimF = getDimensionF();
      String em = "";

      // get additional input
      stopAtError = getInputValueBoolean("StopAtError");

      // check input for errors
      // check whether all lower and upper bounds are set
      for (int i = 0; i < dimCon; i++){
         if (getKindOfConstraint(i) != 3)
               em += "Parameter '" + getVariableNameContinuous(i) +
                  "' does not have lower and upper bounds specified.";
      }
      if (em.length() > 0)
         throw new OptimizerException(em);

      for (int i = 0; i < dimCon; i++){
         // check that all values are positive if
         // logarithmic spacing is required
         if (getDx(i) < 0){ // have logarithmic scale
               if(getL(i) <= 0)
                  em += "Parameter '" + getVariableNameContinuous(i) +
                        "' has logarithmic scale and lower bound '" + getL(i) + "'." + LS;
               if(getU(i) <= 0)
```

```
                em += "Parameter '" + getVariableNameContinuous(i) +
                    "' has logarithmic scale and upper bound '" + getU(i) + "'." + LS;
        }
        // check that l != u if step != 0
        if (getDx(i) != 0 && getL(i) == getU(i))
            em += "Parameter '" + getVariableNameContinuous(i) +
              "' has step size unequal 0 but its lower bound equal to its upper bound." + LS;
        // check that step is an integer value
        if ( Math.rint(getDx(i)) != getDx(i) )
            em += "Parameter '" + getVariableNameContinuous(i) +
              "' has a step size equal to '" + getDx(i) + "'. Require an integer value." + LS;

    }

    if (em.length() > 0)
      throw new OptimizerException(em);

    // all input is OK
    // initialize list with evaluated points
    evaPoi = new TreeMap();
}

/** Runs the evaluation
 * @return <CODE>+4</CODE> the only possible return value
 * @exception Exception
 * @exception OptimizerException
 */
public int run() throws OptimizerException, Exception {
    Point poi = new Point(dimCon, dimDis, dimF);
    // initialize points with current settings
    poi.setXIndex( getX(), getIndex() );
    poi.setStepNumber(0);
    final Point defPoi = (Point)poi.clone();
    // vary continuous parameters
    for(int iC = 0; iC < dimCon; iC++){
      // reset point to default values, so all coordinates are at their inital values
```

```
            poi = (Point)defPoi.clone();

            int nStep = Math.round( (float)getDx(iC) );
            if ( nStep != 0 ){
                  // set up spacing
                  double[] xSp;
                  xSp = null;
                  xSp = Fun.getSpacing(nStep, getL(iC), getU(iC));

                  for(int iS = 0; iS < xSp.length; iS++){
                     poi.setX(iC, xSp[iS]);
                     this.getF(poi);
                  }
            }
         }
         // vary discrete parameters


   int len = getLengthDiscrete(1);
   if ( len != 1 ){
                  for (int ind = 0; ind < len; ind++){
            poi = (Point)defPoi.clone();
            for(int iD = 0; iD < dimDis; iD++){
            // reset point to default values, so all coordinates are at their initial values
                  poi.setIndex(iD, ind);
                  }
         this.getF(poi);
            }
         }
         return 4;
}

/** Evaluates a simulation and reports result
 *@param pt point to be evaluated
 *@return a clone of the point with the new function values stored
 *@exception OptimizerException if an OptimizerException occurs or
```

```
     *          if the user required to stop GenOpt
     *@exception SimulationInputException if an error in writing the
     *          simulation input file occurs
     *@exception Exception if an I/O error in the simulation input file occurs
     */
    public Point getF(final Point pt)
            throws SimulationInputException, OptimizerException, Exception
    {
            Point r = roundCoordinates( pt );
            r.setStepNumber(1);


            if(evaPoi.containsKey(r)){          // point already evaluated
              println("Point already evaluated. Take function value from database.");
              Double[] fD = (Double[])(evaPoi.get(r));
              double[] f = new double[fD.length];
              for (int i = 0; i < fD.length; i++)
                    f[i] = fD[i].doubleValue();
              r.setF(f);
              r.setComment("Point already evaluated.");
            }
            else{    // point not yet evaluated
              try{
                    r = super.getF(r);
                    r.setComment("Function evaluation successful.");
              }
              catch(SimulationInputException e){
                    // must throw such an exception
                    // since input is wrong
                    throw e;
              }
              catch(Exception e){
                    if(stopAtError || mustStopOptimization())
                      throw e;
                    else{
                      String em = "Exception in evaluating x = ( ";
```

```
                for (int i=0; i < dimCon-1; i++)
                    em += r.getX(i) + ", ";
                if (dimDis == 0)
                        em += r.getX(dimCon-1) + ")." + LS;
                else{
                        em += r.getX(dimCon-1) + "; ";
                        for (int i=0; i < dimDis-1; i++)
                            em += r.getIndex(i) + ", ";
                        em += r.getIndex(dimDis-1) + ")." + LS;
                }
                setWarning( em + e.getMessage() );
                double[] f = new double[dimF];
                for(int i=0; i<dimF; i++)
                        f[i] = 0;
                r.setF(f);
                r.setComment("Error during function evaluation. See log file.");
            }
            // proceed as usual
        }

        Double[] fD = new Double[r.getDimensionF()];
        for (int i = 0; i < fD.length; i++)
            fD[i] = new Double(r.getF(i));
        // we must clone the object that we put into the TreeMap
        // Otherwise, it's coordinates get changed since the map
        // contains only a reference to the instance.
        evaPoi.put(r.clone(), fD);
    }
    report(r, SUBITERATION);
    report(r, MAINITERATION);
    return r;
}

/** number of independent continuous variables */
protected int dimCon;
/** number of independent discrete variables */
```

```
    protected int dimDis;
    /** number of function values */
    protected int dimF;
    /** flag whether run should stop or proceed if a simulation error occurs */
    protected boolean stopAtError;
    /** list with evaluated points and its function values */
    protected TreeMap evaPoi;


}
```

APPENDIX D:

ANN MANUAL RECONSTRUCTION CODE

```
function f = evaluate_objective(x,M,V,flw,slw,bi,mint,maxt)

% Function to evaluate the objective functions for the given input vector
% x. x has the decision variables

%variable x
X=x(1:20)';

%flw = first layer weights
%slw = second layer weights
%bi = bias

f = [];

%Hidden neurons outputs
a(1)=-1+2/(1+exp(-2*(flw(1,:)*X+bi(1))));
a(2)=-1+2/(1+exp(-2*(flw(2,:)*X+bi(2))));
a(3)=-1+2/(1+exp(-2*(flw(3,:)*X+bi(3))));
a(4)=-1+2/(1+exp(-2*(flw(4,:)*X+bi(4))));
a(5)=-1+2/(1+exp(-2*(flw(5,:)*X+bi(5))));
a(6)=-1+2/(1+exp(-2*(flw(6,:)*X+bi(6))));
a(7)=-1+2/(1+exp(-2*(flw(7,:)*X+bi(7))));
a(8)=-1+2/(1+exp(-2*(flw(8,:)*X+bi(8))));
a(9)=-1+2/(1+exp(-2*(flw(9,:)*X+bi(9))));
a(10)=-1+2/(1+exp(-2*(flw(10,:)*X+bi(10))));
a(11)=-1+2/(1+exp(-2*(flw(11,:)*X+bi(11))));
a(12)=-1+2/(1+exp(-2*(flw(12,:)*X+bi(12))));
a(13)=-1+2/(1+exp(-2*(flw(13,:)*X+bi(13))));
a(14)=-1+2/(1+exp(-2*(flw(14,:)*X+bi(14))));
a(15)=-1+2/(1+exp(-2*(flw(15,:)*X+bi(15))));
a(16)=-1+2/(1+exp(-2*(flw(16,:)*X+bi(16))));
a(17)=-1+2/(1+exp(-2*(flw(17,:)*X+bi(17))));
```

```
a(18)=-1+2/(1+exp(-2*(flw(18,:)*X+bi(18))));
a(19)=-1+2/(1+exp(-2*(flw(19,:)*X+bi(19))));
a(20)=-1+2/(1+exp(-2*(flw(20,:)*X+bi(20))));

Final output
A(1)=slw(1,:)*a'+bi(21);
A(2)=slw(2,:)*a'+bi(22);
A(3)=slw(3,:)*a'+bi(23);
A(4)=slw(4,:)*a'+bi(24);
A(5)=slw(5,:)*a'+bi(25);

%Rescaling

for i=1:5
   Ascaled(i) = 0.5*(A(i)+1)*(maxt(i)-mint(i)) + mint(i);
end

f(1)=abs(Ascaled(4))*(1+max(Ascaled(5)/100,0));
                    f(2)=sum(Ascaled(1:3))*(1+max(Ascaled(5)/100,0));
```

APPENDIX E:
MANUALLY CONSTRUCTED AND RANDOM DESIGNS

Varia-bles	Range	Manually constructed solutions				Random solutions				
HSP	[20,25]	22	22	20.5	23	22.2	20.6	20.2	22.3	24.3
CSP	[23,27]	24.5	26	26.5	25	26.7	26.9	25.7	26.5	23.0
SDHW	[0,30]	30	30	5	30	4	10	5	5	6
SDMID	[0,30]	30	30	5	30	13	14	12	27	0
SDC	[0,30]	30	30	5	30	9	12	6	19	22
FDMID	[0,60]	60	60	60	0	23	51	22	37	44
FDC	[0,60]	60	60	60	0	12	56	20	39	24
FDW	[0,60]	60	60	60	0	38	22	26	36	34
VRR	[0.118,0.708]	0.118	0.118	0.118	0.708	0.541	0.698	0.594	0.533	0.404
VRC	[0.118,0.708]	0.472	0.472	0.118	0.708	0.185	0.515	0.707	0.685	0.153
VRH	[0.118,0.708]	0.472	0.472	0.118	0.236	0.330	0.531	0.686	0.561	0.555
RHW	[30,60]	50	50	60	60	43	48	38	56	45
RHMID	[30,60]	50	50	60	40	52	56	40	50	32
RHC	[30,60]	50	50	60	30	41	57	53	54	54
WF1N	[4.76, 14.30]	4.7	4.7	4.7	4.7	11.2	5.6	7.4	8.7	9.3
WF1S	[2.20, 6.60]	6.6	6.6	6.6	6.6	6.2	5.9	6.4	4.8	2.3
WF2N	[4.06, 12.18]	4.06	4.06	4.06	4.06	10.7	5.4	10.8	10.9	7.7
WF2S2	[1.38, 4.14]	4.14	2	2	1.38	4.0	4.0	3.1	2.1	2.4
WF2S1	[2.08, 6.25]	6.25	4	4	2.084	2.9	3.4	4.6	2.8	4.7
TCK	[0.05,0.25]	0.1	0.1	0.05	0.25	0.10	0.12	0.13	0.11	0.13