NTNU

Norwegian University of
Science and Technology

Heuristic solution approach to simultaneous optimization of heat and work integration

A two-level optimization model using genetic algorithms to establish the energy target for maximum heat and work recovery

Kari Sofie Hall Borge

Industrial Economics and Technology Management
Submission date: June 2017
Supervisor: Asgeir Tomasgard, IØT
Co-supervisor: Bjørn Nygreen, IØT
Truls Gundersen, EPT
Matias Vikse, EPT

Norwegian University of Science and Technology
Department of Industrial Economics and Technology Management

Description of the problem as stated in the thesis contract

The purpose of this thesis is to develop and implement an optimization method in order to establish the energy target for optimal heat and work integration of multiple pressure changing process streams. Since two forms of energy (heat and work) are under consideration, the objective is to minimize the exergy consumption by locating the optimal stream split arrangement for the pressure changing streams.

Background:
Extensive efforts have been made to optimize heat recovery networks. However, few articles have been published describing how pressure energy of process streams can be integrated in the heat exchanger network to enhance energy efficiency. The objective of this work is to study the influence of integrating compressors and expanders into the heat exchanger network, and identify inlet temperatures to the pressure changing units so to minimize the exergy consumption. Methods have been developed for heat integration of one compressing and one expanding stream. This work extends the problem to include multiple compressing and expanding streams.

Main tasks:

- Formulate a two-level optimization model
- Implement the optimization model for evaluation of candidate solutions
- Implement the search algorithm
- Design new test cases involving multiple pressure changing streams
- Test the model on the new test cases
- Discuss the value of integrating compressors and expanders into the heat exchanger network

Abstract

Heat integration is of significant importance to the process industry in the efforts of reducing energy consumption and total annual costs. The heat integration problem attempts finding the optimal interconnections of processing equipment in order to reduce the energy requirements of the process. A complete model of the heat integration problem involves determining the optimal performance of the heat integrated system and the optimal design of the heat exchanger network. The performance of the system can be identified prior to the network design. This work is concerned with identifying the performance of a system in terms of an energy target for optimal heat integration.

Heat integration has long been a central topic in the process industry, however, work integration is a topic of recent interest. Both heat and work are forms of energy that are often available in industrial processes, and utilization of this energy should therefore be attempted instead of resulting in waste. For small problems, optimization of heat and work integration can be modeled with mixed-integer nonlinear programming. Introducing more than one compressing and one expanding stream, no exact solution method has currently been able to identify the energy target for optimal heat and work integration.

In order to solve the problem of simultaneous heat and work integration, a two-level optimization model using genetic algorithms has been developed. Results from solving a set of test cases with one compressing and one expanding stream, in which exact methods can prove optimality, show that the two-level optimization model is able to find the optimal or a near-optimal solution. Four new test cases are presented involving multiple compressing and expanding streams, in which the model provides high quality solutions. The model cannot guarantee a global optimal solution. However, in comparison with common practice in the industry, the two-level optimization model generates significantly better solutions.

Two genetic algorithms have been implemented and tested in this research; a basic genetic algorithm and a genetic algorithm with crowding. The algorithm with crowding was implemented in an effort to improve the global search, reducing the likelihood of getting stuck in a local optimum. Results from both algorithms are presented in comparison with common practice in the industry.

Sammendrag

Grunnet klimaendringer og store utgifter forbundet med høyt energibruk er det en økende interesse for energieffektive løsninger i produksjonsanlegg. Miljøpåvirkningene og produksjonskostnadene kan reduseres betraktelig. Varmegjenvinning er viktig for å redusere energiforbruk og består hovedsakelig i å designe et varmevekselnettverk som minimaliserer drifts -og produksjonskostnader. Maksimal varmegjenvinning kan identifiseres før utforming av selve nettverket i form av et minstekrav til energiforbruk. Denne oppgaven omhandler lokalisering av et minstekrav til energiforbruk i industrielle prosesser.

Varmeintegrering har vært gjenstand for betydelig forskning i lengre tid. I et varmevekselnettverk skal varme strømmer kjøles ned og kalde strømmer varmes opp. Uten trykkforandring kan disse avkjøles og varmes opp kontinuerlig. Det er derimot ofte et behov for trykkendring av slike strømmer. Trykkendring fører imidlertid til en diskontinuerlig endring i temperatur. Varmeintegrering av få strømmer med trykkendring kan modelleres med blandet heltall -og kontinuerlig ikke-lineær programmering. For prosesser med mer enn en komprimerende og en ekspanderende strøm finnes det ingen eksakt løsningsmetode som kan lokalisere et minstekrav til energiforbruk ved optimal varmeintegrasjon.

For å kunne finne gode løsninger for varmeintegrasjon av strømmer med trykkendring er det utviklet en optimeringsmodell med to nivå som benytter genetiske algoritmer. Modellen er testet på problem som ikke involverer mer enn en komprimerende og en exspanderende strøm. I disse tilfellene er optimal løsning kjent og modellen fant optimal eller tilnærmet optimal løsning. Modellen kan også vise til gode resultater for fire nye test problemer av større størrelse. De nye testtilfellene er laget for å studere fordelaktig integrering av et større antall strømmer med trykkendring enn tidligere. To-nivå modellen er en heuristisk metode og kan ikke garantere optimal løsning. Men, i sammenligning med vanlig praksis i industrien genererer modellen betydelig mer energieffektive løsninger.

To genetiske algoritmer er implementert og testet i denne oppgaven; en grunnleggende genetisk algoritme og en genetisk algoritme med crowding. Algoritmen med crowding ble implementert i et forsøk på å forbedre den globale søkeevnen, ved å redusere risikoen for å gå seg fast i et lokalt optimum. Resultater fra begge algoritmer er sammenlignet med vanlig praksis i industrien.

Preface

This master's thesis is written as part of my MSc. in Industrial Economics and Technology Management at the Norwegian University of Science and Technology, Department of Industrial Economics and Technology Management. The thesis is a continuation of the work done in my specialization project during the fall of 2016.

My technological specialization is in process engineering and my economical specialization is in optimization. I have chosen a project in which I can use my knowledge from both fields. Combining the two disciplines have been rewarding, as large parts of my education have come to use. Also, it has been inspiring to experience how expertise in the filed of optimization may contribute to new developments in the field of process engineering.

Acknowledgments

First and foremost, I would like to thank my supervisor, Asgeir Tomasgard at the Department of Industrial Economics and Technology Management, for facilitating the project and making time for meetings and guidance. I am also thankful to Truls Gundersen at the Department of Energy and Process Engineering for inspiring me to take on the project.

I would then like to thank Björn Nygreen at the Department of Industrial Economics and Technology Management for helpful conversations on difficult matters. A great thank you is also required to Fu Chao from Sintef Energy Research for valuable feedback on results and for sharing his knowledge in process synthesis. Furthermore, I am thankful to Matias Vikse, PhD candidate at the Department of Energy and Process Engineering, for always keeping the door open for any matters.

Lastly, my greatest appreciation to Tom Kåre Borge from Kongsberg Defence and Aerospace for devoting considerably time to understanding my project. I am grateful for the help given in developing the solution method and for providing assistance in using genetic algorithms. Most of all I am thankful for encouragement and for giving valuable feedback when I have been in need of discussion of an issue.

Trondheim, 30th of June 2017
Kari Sofie Hall Borge

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

DP Disjunctive programming

GA Genetic algorithm

GCC Grand composite curve

GDP General disjunctive programming

HEN Heat exchanger network

IEA International energy agency

LNG Liquefied natural gas

LP Linear programming

MINLP Mixed-integer nonlinear programming

MIP Mixed-integer programming

NLP Nonlinear programming

PA Pinch analysis

PTA Problem table algorithm

SA Simulating annealing

Nomenclature

Symbol	Unit	Description
Q	[kW]	heat transfer (thermal energy)
W	[kW]	work
T	[K]	temperature
T_0	[K]	ambient temperature
ΔT	[K]	temperature difference
m	[kg/s]	flow rate
c_p	[kJ/kg K]	specific heat
Ex	[kW]	exergy

Subscripts and superscripts

Symbol	Description
s	stream
b	stream branch
z	stream segment
S	supply
T	target

Symbol	Description
H	hot
C	cold
HU	hot utility
CU	cold utility
P	pressure change
CP	constant pressure
in	inlet
out	outlet
k	heat interval
p	pinch
lb	lower bound
ub	upper bound

List of Symbols

This section lists mathematical symbols and operators in the report.

Symbol	Description
S	upper case letters denote sets
s	lower case letters denote variables
$\forall$	"for all"
$\in$	"element of"
$\mathbb{R}$	set of real numbers
$\vert$	conditional event
$\setminus$	denote a set difference

Chapter 1

Introduction

Climate change has become an inevitable reality and a global concern. New regulations and environmental laws are being developed on an international level. According to the International Energy Agency (IEA), energy efficiency must account for more than 50% of the actions against global warming, highlighting the need for more energy efficient solutions. As illustrated in Figure 1.1, unrealized energy efficiency potentials are still high. The process industry[1] typically includes the largest industrial consumers of energy, such as petroleum refineries and chemical industries [68].

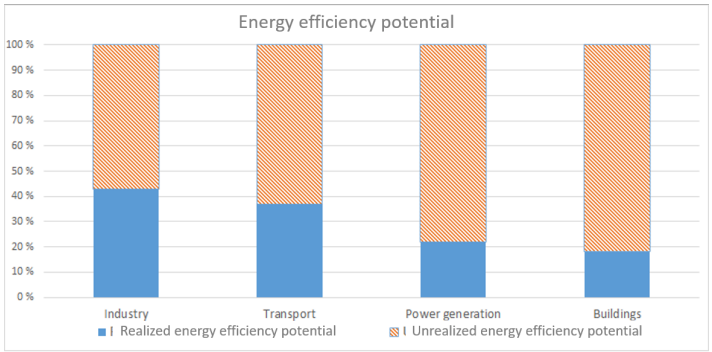

Figure 1.1: Energy efficiency potential by sector (numbers taken from [61]).

[1] Process industries are comprised of manufacturing businesses that produce end products through chemical or mechanical means on a continual basis, with raw materials or feed-stock being converted in batch mode or in a continuous flow stream. Process industries include chemicals, petroleum refining, coal liquefaction, gas to liquids, petrochemicals (plastics, synthetic fibers, and synthetic rubbers produced from petroleum), food and drink, pulp and paper, among others [69].

Lately, there has been a great interest for heat integration in the process industry in order to reduce energy consumption and total annual costs. In industrial processes there are streams that need heating and streams that need cooling. One way to achieve this is to use hot steam and cooling water, as illustrated in Figure 1.2. However, production of hot steam is costly and energy demanding. By using hot streams to heat cold streams and cold streams to cool hot streams, as illustrated in Figure 1.3, it is possible to reduce the need for external heating and cooling considerably.

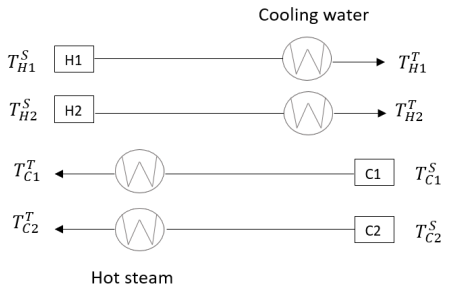

Figure 1.2: Non-integrated process.

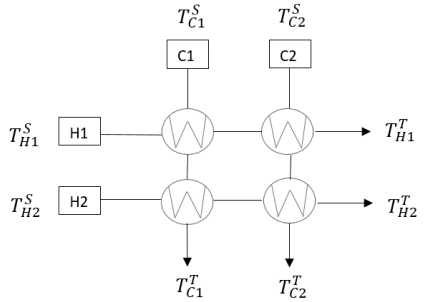

Figure 1.3: Heat integrated process.

Heat integration addresses the problem of maximizing heat recovery among process streams. The problem involves determining optimal performance of the heat integrated system and the optimal design of the heat exchanger network (HEN). Locating the optimal network design is a complex task involving a difficult combinatorial problem. For a fixed number of streams there are a great number of possible interconnections of process equipment. However, the number of network configurations that satisfies the optimal performance of the system is much smaller. This work is concerned with identifying the performance of heat integrated systems, which may serve as a guideline in the design of the network. The performance of the system is defined by minimum energy consumption for optimal heat integration.

Despite the large interest in optimization of heat recovery there are limited sources in the literature regarding pressure recovery in order to enhance the heat integration even further. Handling pressure requires significant energy consumption and is especially important in oil refineries and cryogenic processes[2]. In order to improve energy efficiency, it is of great interest to study the interaction between heat and work related to pressure change of process streams. Research on heat and work integration has hitherto focused on integration of one compressing and one expanding stream. This work expands the problem involving multiple compressing and expanding streams.

[2]Cryogenic processes take place at very low temperatures, such as production of liquefied natural gas (LNG). Commonly, a gas is referred to as cryogenic if it can be liquefied at temperatures below 150 °C (123.15 K)

An effective way to study the impact of heat and work integration is mathematical programming. A mathematical formulation of heat and work integration requires a good way to model the heat distribution among the process streams in which ensures thermodynamically feasibility and efficient algorithms for finding the best solutions. The heat and work integration problem frequently involve nonlinear behaviour. Unlike linear programming (LP) problems, conventional solution methods for nonlinear programming (NLP) problems are complex and not very efficient. For multiple compressing and expanding streams, the complexity of the heat and work integration problem increases significantly. Genetic algorithms (GA) have emerged as a powerful stochastic search and optimization technique founded on the principle of natural evolution. GAs do not guarantee a global optimal solution, but often locate very good solutions within reasonable time for problems that are otherwise difficult to solve.

This work presents a two-level optimization model using GAs. The objective is to identify the energy target for heat and work integration of multiple pressure changing streams. The two-level optimization model decouples the optimization problem into two loops: The outer loop uses a GA to fix the inlet temperatures to the pressure changing units. Each evaluation of the heuristic optimizer in the outer loop requires solving an inner loop problem by highly efficient linear programming. For increasing problem sizes, heuristic search methods are more likely to outperform exact solution methods. The purpose of the two-level optimization model is to combine the two methods in an effort to reduce the number of decision variables that must be handled by the search algorithm. The new contribution of this model is the ability to handle multiple compressing and expanding streams.

Literature regarding heat integration is reviewed in Chapter 2. A detailed description of the problem is given in Chapter 3. A brief introduction to previous methods and acquired knowledge about heat and work integration is presented in Chapter 4. The model utilizes GAs in the search process. In order to implement an efficient algorithm for the problem under consideration, a thorough study of GAs has been carried out. An introduction to this type of metaheuristic procedure is given in Chapter 5. The two-level optimization model is presented in Chapter 6. In Chapter 7, implementation details are stated. A thorough computational study of four new test cases is presented in Chapter 8. Chapter 9 concludes the thesis and Chapter 10 presents suggestions for future research.

Chapter 2

Literature review

This chapter provides a brief review of published work regarding heat integration. Section 2.1 summarizes previous studies of heat integration and different methods that has been developed in order to solve the problem of optimal heat integration. Section 2.2 presents previous work regarding heat and work integration, whilst Section 2.3 gives a brief overview of the use of GAs in the field of heat integration.

2.1 Heat integration

Heat integration in industrial processes has been subject to a significant amount of research over the past 40 years, in order to increase energy efficiency and reduce annual costs. Optimal heat integration consists of integrating hot and cold process streams in a network of heat exchangers. A typical industrial process contains 30 to 80 streams leading to a large number of possible network configurations. However, Hohmann [33] and Linnhoff and Flower [45] recognized that the minimum energy consumption for optimal heat integration can be established prior to the actual network design, simplifying the problem considerably; the number of configurations satisfying the energy target of minimum energy consumption is much less than the total number of configurations.

Pinch analysis (PA) is a well-established methodology that provides a systematic procedure of minimizing the energy consumption in industrial processes. PA was developed in the late 1970s by the identification of the heat recovery pinch point, presented by Linnhoff and Flower [45]. A similar concept was applied by Umeda et al. [66]. PA uses thermodynamic concepts to determine the energy target for optimal heat integration and provides methods to achieve this target in the network design. Linnhoff and Flower [45] developed the problem table algorithm (PTA) in order to determine the energy target

more efficiently. Thermodynamic insights and targets, developed by the authors mentioned above among others, have led to significant improvements in energy efficiency and motivated algorithmic approaches to optimization of heat integration.

Cerda et al. [8] formulated an LP model based on the transportation problem, in order to determine the energy target for optimal heat integration. Papoulias and Grossmann [60] used a similar approach. They proposed various formulations of the transshipment model: An LP problem was used to predict the energy target and thus the minimum utility costs. A mixed-integer programming (MIP) model was developed to determine the matches and the heat loads that would have to take place in the network in order to satisfy the cost target. The MIP model provides information for deriving the network structure, but the structure design had to be generated manually. Floudas et a. [15] extended this work to automatically generate the network structure by combining the transshipment model with a nonlinear procedure.

Later developments concern design of the HEN, in which satisfies the energy target for optimal heat integration. Duran and Grossmann [12] presented an NLP model for simultaneous process optimization and heat integration. The NLP problem corresponds to a nondifferentiable optimization problem, requiring smooth approximations. Grossmann et al. [30] developed a disjunctive optimization method[1] that uses binary variables and avoid using smooth approximations. A mixed-integer nonlinear programming (MINLP) model was developed, in which is reduced to an MIP problem when only isothermal streams are present.

For further reading, an early review of process synthesis and heat integration was presented by Nishida et al. [56]. Following, two thorough reviews on the topic of heat exchanger networks were contributed by Gundersen and Naess [31] and Ježowski [35]. A complementary review was later presented by Furman and Sahinidis [20].

2.2 Heat and work integration

Optimal integration of heat and work may yield significant energy savings and thus reduce annual costs. Aspelund [1] presented a heuristic graphical method for correct integration of compressors and expanders. Wechsung et al. [71] used these heuristic rules and proposed an MINLP model for optimization of HENs, wherein selected streams are subject to pressure manipulation. They demonstrated that a favorable sequence for compression and expansion can significantly reduce the energy consumption. The sequence is illustrated in Figure 2.1. The hot stream is compressed, expanded and compressed again, whereas the cold stream is expanded, compressed and then expanded. When expanded, the hot stream

[1] Disjunctive programming (DP) is optimization over disjunctive sets. A disjunctive set can be defined as inequalities connected by the logical operations *and* and *or*. A excellent introduction to DP is found in Bjørnqvists doctorial dissertation [5].

temporarily behaves as a cold stream, while a compressed cold stream behaves as a hot stream. Letting the streams go through multiple pressure manipulation stages, the streams may temporarily act as utility streams.

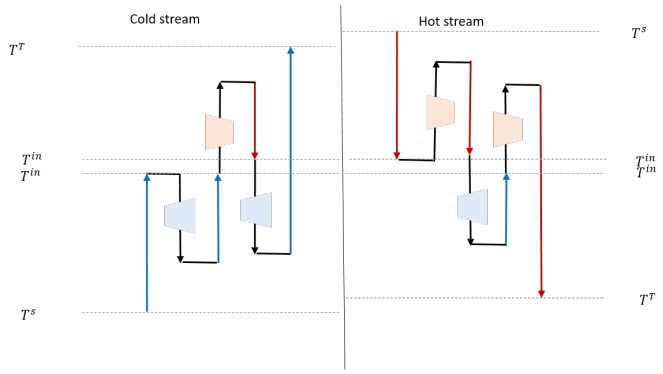

Figure 2.1: Favorable route for compression and expansion for a total of three pressure manipulation stages.

Assuming that supply and target temperatures and supply and target pressures of the streams in Figure 2.1 are fixed, the intermediate inlet and outlet temperatures to the pressure changing units and the intermediate pressures are subject to optimization.

Fu and Gundersen [16] derived a systematic graphical design procedure for integration of a compressing stream with the HEN. A similar procedure was carried out for integration of an expanding stream [17]. It was proved that the heuristic rules proposed by Aspelund [1] do not always hold and a set of theorems for appropriate placement of the pressure changing units was proposed. Because both heat and work are involved, the objective is to minimize the exergy consumption. Fu and Gundersen [18] also derived a theorem for integration of both a compressing and an expanding stream. Integration of multiple compressing and expanding streams have not yet been considered. For multiple pressure changing streams, a complicating factor is the generation of new pinch points and which pressure changing unit should be considered first.

Onishi et al. [57] presented a superstructure for optimization of HENs, considering the adjustment of pressure levels of process streams to enhance heat integration. The model was formulated using generalized disjunctive programming (GDP) and re-formulated as an MINLP model. The model utilizes the sequence structure developed by Wechsung et al. [71], illustrated in Figure 2.1. The models allow for coupling between turbines and compressors, and selection of turbines and valves to minimize the total annual costs. The authors demonstrated that integration of heat and work can reduce the amount of necessary utilities, lowering the costs involved in the process; however, the nonconvex MINLP problem is difficult to solve for other than small problems.

Dowling [11] explored mathematical programming-based equations of the rules for above ambient compression developed by Fu and Gundersen [16]. An LP model for an optimal compression strategy was presented, which minimizes a weighted combination of work and hot utility. The major limitation regarding this model is the shortfall of not considering the outlet temperatures from the compressors as potential pinch candidates. Dowling utilized a superstructure representation for splitting of pressure changing streams. Maurstad [52] extended this structure to include potential pinch point candidates generated by the outlet temperatures from the pressure changing units. Maurstad [52] also developed an MINLP model, in which do not rely on the rules developed by Fu and Gundersen [18]. This model considered, at the most, one compressing and one expanding stream.

2.3 Utilization of genetic algorithms in heat integration problems

GAs were mostly developed in the 1970s as a stochastic search technique inspired by natural evolution. GAs aim to improve the performance by sampling promising regions in the search space, i.e. regions with high probability of good solutions. The first main work is attributed to Holland [34]. A thoroughly review of the history of GAs and other evolutionary algorithms is given by Bäck et al. [3]. The interest and utilization of GAs in the field of heat integration is relatively recent. Heuristic search algorithms are in general computationally expensive. The increased interest in GAs is facilitated by the increased availability of high performance computers and improved guidelines for the specification of the GA parameters.

Lewin et al. [41] developed an approach to optimization of HENs based on the use of GAs. The approach consists of a two-level algorithmic structure; the upper level GA, which generates HEN structures by applying genetic operators on the solution population and the lower level optimization algorithm, which carries out parameter optimization for a given HEN structure. Lewin [40] presented a modification of the algorithm in which stream splitting is supported. Another application of GAs was presented by Wang et al. [70] for optimization of separation sequences in distillation systems and their HENs. Ravagnani et al. [63] used GAs for optimization of HENs based on a previous optimization of ΔT_{min} and the energy target. In this work, ΔT_{min} is optimized using GAs together with PA techniques. When ΔT_{min} is identified, the optimal HENs above and below pinch are obtained separately using GAs. Pettersson and Soderman [62] developed a method for optimization of HENs, in which accounts for uncertainties in operating conditions. In this method, a GA is combined with NLP.

Later developments in the application of GAs have utilized hybrid algorithms, in which genetic operators are combined with other search techniques. Yu et al. [72] presented a GA in combination with simulated annealing (SA) to search for the optimal HEN in large scale

systems. The method was used to solve a problem involving 167 streams. Also Luo et al. [48] combined genetic operators with SA and other strategies so that the structural search ability of the algorithm is greatly improved. The algorithm was developed for design of large scale HENs. Maehara and Shimoda [49] presented a hybrid method of GAs and the Nelder-Mead algorithm in the search for the optimum chiller configuration for a heat source plant. The Nelder-Mead algorithm was implemented in order to reduce the number of calculations required to determine the optimum chiller configuration. The performance of the combined algorithm improved the ability to find the optimal configuration.

For a more thoroughly analysis of the utilization of GAs in the field of heat integration, Gosselin et al. [27] has contributed with an extensive review of when and how GAs have been used in heat integration problems. This review concerns articles published throughout the 1990s and 2000s. An overview of the algorithmic methods in the publications presented in this section is listed in Table 2.1. The table shows the problem objectives and significant GA parameter values. An interesting observation from Table 2.1 is the significantly large mutation rates, in comparison with what is regarded as common values (0.001-0.05).

Article		**Problem**	**GA**				
Year	**Ref.**	**Objective**	**Bin/real**	$\mathbf{P_{cross}}$	$\mathbf{P_{mut}}$	**Pop size**	**Max. iter**
2013	[49]	Min. cost	-	0.1	0.05	100	1320
2008	[48]	Min. cost	R	0.8	0.1	100	400
2007	[62]	Min. cost	R	-	0.01	20	100
2004	[63]	Min. cost	R	0.8	0.1	5-57	50
2000	[72]	Min cost	-	0.3	0.4	80	400
1998	[70]	Min. cost	R	0.2	0.5	1000	160
1998	[40]	Max. heat recovery	R	0.6	0.01	-	-
1998	[41]	Max. heat recovery	R	0.6	0.1-0.5	40-80	100

Table 2.1: Significant GA parameters implemented for GAs in optimization of heat integration in industrial processes.

The articles presented above concern heat integration of constant pressure process streams. To the author's knowledge there has been no efforts in solving the heat and work integration problem using genetic search algorithm.

More recently, Li et al. [42] presented an efficient hierarchical optimization framework for optimization of water distribution. There are two loops in this model: the outer loop uses heuristic algorithms and the inner loop uses efficient LP. The choice of heuristic algorithm in the outer loop can vary. Li et al. [42] used GAs, particle swarm optimization and SA, among others. The purpose of such a model is to reduce the computational effort by decreasing the number of variables that must be analyzed by the heuristic algorithm, and formulate the specific problem so that the inner loop, with many decision variables, can be

solved with highly efficient LP. A similar approach to heat and work integration has been developed in this thesis.

Figure 2.2: Timeline for developments within heat integration in process industries.

Chapter 3

Problem description

This master's thesis addresses energy targeting for optimal heat and work integration. Work related to pressure changing streams is an energy form that are often available in process plants. Pressure based energy can effectively be transformed to heating and cooling duty and may significantly reduce the utility requirements. The problem can be described as follows:

> A set of hot streams are to be heated and a set of cold streams are to be cooled. All streams have given supply and target temperatures, flow rates and fixed heat capacities. The pressure changing streams have fixed supply and target pressures. The problem consists in identifying maximum thermal integration of the system through efficient placement of pressure changing units, in which the total energy consumption is minimal.

Placement of compressors and expanders are defined by the inlet temperatures to these units. Common practice in the industry is to let the pressure changing streams be compressed at ambient temperature and expanded at hot utility temperature, in order to minimize compression work and maximize expansion work. However, compressors add heat to the system and the cooling demand may increase if compressors operate at low temperatures. Similarly, the heating required may increase if expanders are operating at high temperatures. Thus, there is a trade-off between work consumption and demand for external heating and cooling. One can achieve higher energy efficiency by letting compression and expansion take place at a higher and a lower temperature, respectively. The required heating and cooling may be significantly reduced in the expense of less expansion work and more compressor work. Additionally, the pressure changing streams may be split into branches so that parts of the stream enter separate units at different temperatures. This may enhance the heat integration even further.

This work is concerned with identifying the inlet temperatures to the pressure changing units in order to obtain the energy target for maximum heat and work recovery. Addition-

ally, the number of branches necessary to obtain maximum recovery must be identified as well as the optimal distribution of the flow rates through the compressors and expanders on each branch.

Energy targeting is beneficial when the energy is the dominant cost item in the process. Optimization of energy efficiency can lead to highly efficient processes, but on the other hand be economically impractical. Stream splitting may lead to significant reduction in energy consumption, but also implies additional pressure changing units, in which engender considerable capital costs. Introducing pressure changing streams will therefore require an appropriate trade-off between energy efficiency and investment costs. In order to gain insight into energy efficient processes without unrealistic cost conditions, the maximum number of branches is restricted to three. The objective of this work is to develop a model to identify the minimum energy consumption for optimal heat and work integration and consideration of costs are thus beyond the scope. Nevertheless, a brief discussion about the investment cost implications for the generated solutions are provided.

Chapter 4

Heat and work integration

PA has laid the foundation for process optimization and heat integration. A short introduction is given in Section 4.1, highlighting the most prominent features. Section 4.2 presents the heat cascade, which is a tool to calculate the overall heating and cooling requirements in HENs. The optimization model developed in this thesis optimizes heat and work integration using mathematical programming together with PA techniques.

Industrial processes involve multiple equipment units. A fundamental concept in PA is correct integration of process equipment in order to enhance energy savings. Integration of compressors and expanders are of recent interest and is the focus of this thesis. Section 4.3 introduces current knowledge about beneficial integration of compressors and expanders. When integrating compressors and expanders into the HEN, two forms of energy of different quality are under consideration, mainly heat and work. Exergy provides a unit of quality measurement for different forms of energy and is a suitable measure of the process performance. Section 4.4 provides a brief description of the exergy term.

4.1 Pinch nnalysis

PA is a well established methodology for optimizing heat integration in industrial processes. The method aim to maximize process-to-process heat recovery, while reducing the need of external heating and cooling. The fundamental principle in PA is to match streams requiring heat with streams rejecting heat. According to the first law of thermodynamics [55], the amount of heat available for exchange associated with stream s is expressed with Equation 4.1 and 4.2, for hot and cold streams, respectively.

$$Q_s = (mc_p)_s \left(T_s^S - T_s^T\right), \qquad \forall s \in S^H \tag{4.1}$$

$$Q_s = (mc_p)_s \left(T_s^T - T_s^S\right), \qquad \forall s \in S^C \tag{4.2}$$

where T^S and T^T are the supply and target temperatures, m is the mass flow rate and c_p is the heat capacity of the fluid. Heat recovery is restricted by the shape of the stream composite curves and the fact that heat can only be transferred from a higher temperature to a lower temperature, defined by the second law of thermodynamics [55]. Composite curves are graphical representations of enthalpy change of streams. The curves illustrate how much heat is available and how much heat is required within certain temperature intervals. The composite curves for a four-stream example presented in Table 4.1 is illustrated in Figure 4.1.

Stream	$\boldsymbol{T^S}$ [° C]	$\boldsymbol{T^T}$ [° C]	$\boldsymbol{mc_p}$ [kW/° C]
H1	170	60	3
H2	150	30	1.5
C1	20	135	2
C2	80	140	4

Table 4.1: Stream data for a four-stream example.

The heat recovery is also restricted by the minimum temperature difference, ΔT_{min}, which is an economic parameter featuring a near-optimal trade-off between investment costs and operating costs. A large ΔT_{min} implies high energy consumption, whilst a small ΔT_{min} will require a large heat exchanger. Parallel composite curves allows for a high level of heat recovery. The point of smallest vertical distance (ΔT_{min}) between the composite curves represents a bottleneck for maximum heat integration and is referred to as the pinch point. A key finding from PA is that for maximum heat integration, no heat is transferred across the pinch temperature [46]. Thus, the pinch point divides the process into two distinct regions; for heat exchange at temperatures above pinch there will be a deficit of heat, whilst below pinch there will be an excess of heat [46].

Consider the four-stream example in Table 4.1. The composite curves in Figure 4.1 are shifted horizontally until the pinch point is located. Further alignment of the composite curves violates the ΔT_{min} constraint. The minimum utility requirements are determined by reading off the horizontal difference between the curves at the end points on the graph. Maximum heat recovery within the process can thus be identified prior to the design of the network. The utility requirement is a lower bound for the given process and may serve as a guideline for what is achievable in the actual design of the HEN [71].

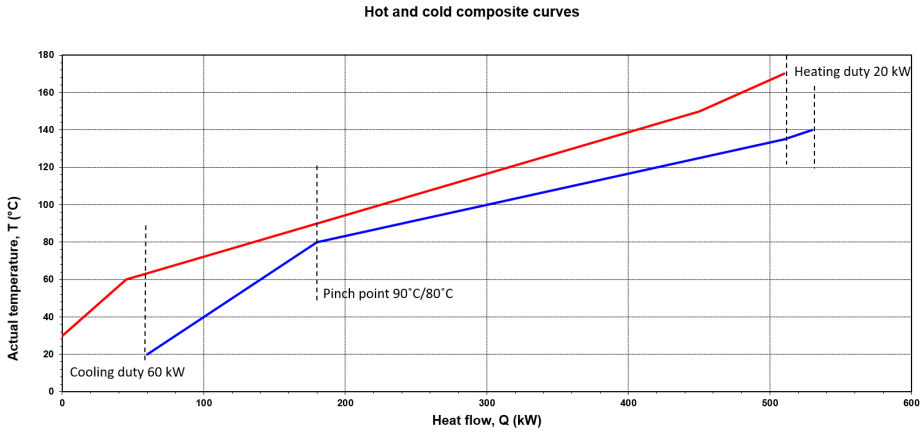

Figure 4.1: Composite curves illustrating the pinch point and the utility requirements.

4.2 The heat cascade method

The heat cascade is a representation of the heating and cooling demands within a process. The heat cascade requires a partitioning of the continuous temperature range into $|K|$ successive intervals. The temperature intervals are established based on stream supply temperatures[1]. By employing the minimum temperature driving force ΔT_{min}, each interval will have two corresponding temperatures; T_k^H at the hot side and T_k^C at the cold side. Therefore, within each interval it is thermodynamically feasible to transfer heat from hot streams to cold streams, and to subsequent lower intervals. A heat balance is required for each interval k, illustrated in Figure 4.2.

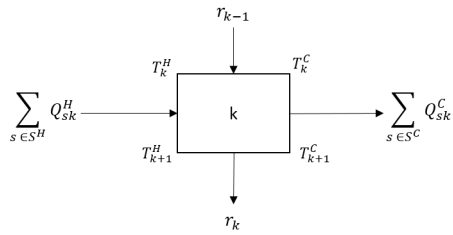

Figure 4.2: Heat balance for temperature interval k.

[1] Based on investigation of the composite curves, Grimes et al. [29] discovered that the supply temperatures of the process streams correspond to possible pinch points. In order to obtain a suitable partitioning so that no pinch point is located inside a temperature interval, each supply temperature T^{S} gives rise to a temperature interval, $T_{\mathrm{k}}^{\mathrm{H}}$, $T_{\mathrm{k}}^{\mathrm{C}}$

Heat residual r_{k-1} is the excessive heat from the previous interval, whilst r_k is the heat cascaded to the next interval. Q^H_{sk} and Q^C_{sk} are heat available by the hot streams and heat required by the cold streams in interval k. Thus, the heat balance for an interval k is calculated with Equation 4.3.

$$r_k = r_{k-1} + \sum_{s \in S^H} Q^H_{sk} - \sum_{s \in S^C} Q^C_{sk} \tag{4.3}$$

Constant pressure streams are continuously heated and cooled from the respective supply temperatures to the target temperatures. Q^H_{sk} and Q^C_{sk} for each interval k can therefore be calculated with Equation 4.4 and 4.5.

$$Q^H_{sk} = (mc_p)_s \left(T^H_k - max\left[T^H_{k+1}, T^T_s\right]\right), \forall s \in S^H \tag{4.4}$$

$$Q^C_{sk} = (mc_p)_s \left(min\left[T^C_k, T^T_s\right] - T^C_{k+1}\right), \forall s \in S^C \tag{4.5}$$

Further elaboration of the heat cascade method is illustrated in Example 4.1 below.

Example 4.1. *Consider the stream data in Table 4.1. The heat cascade requires two iterations. First, the heat load entering the first interval equals zero. Subsequently, all the residuals are calculated using Equation 4.3 as represented in the second column in Table 4.2. To avoid the negative residuals a heat load equal to the most negative residual must be added to the first interval. The revised calculated residuals are shown in the column for the second iteration. Hot utility equals the heat load entering the first interval and the cold utility equals the heat load leaving the last interval. The pinch location is designated by a zero-heat residual, in this example r_3, with corresponding pinch temperatures $90^{\circ}C/80^{\circ}C$. The heat cascade together with the grand composite curve (GCC) for the revised heat loads are shown in Figure 4.3.*

Heat residual	**First iteration [kW]**	**Second iteration [kW]**
$r_1(Q^{HU})$	0	20
r_2	60	80
r_3	-20	0
$r_4(Q^{CU})$	40	60

Table 4.2: Heat residuals cascaded in Figure 4.3.

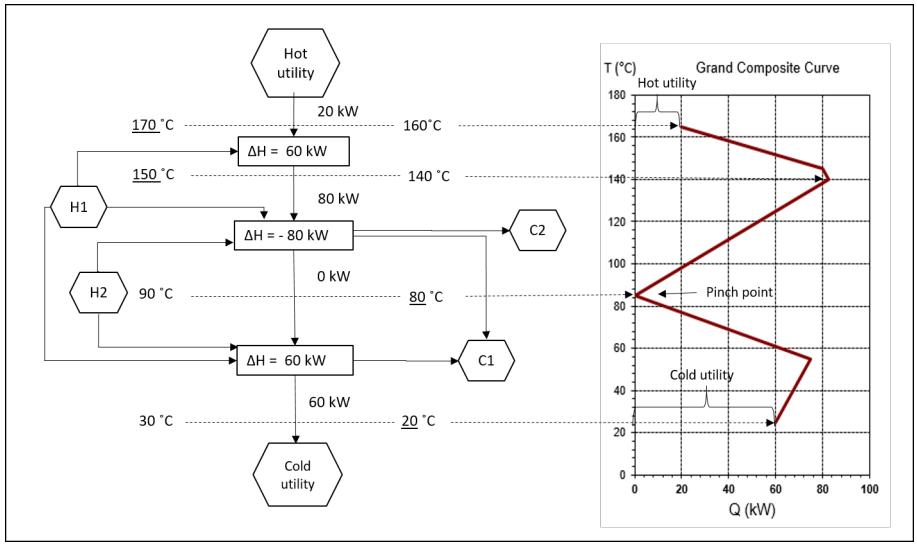

Figure 4.3: The heat cascade and the GCC for the stream data in Table 4.1.

The GCC illustrates the variation of heat supply and heat demand within a process. The GCC clearly illustrates the pinch point and the minimum required heating and cooling. A more thoroughly study of the GCC and the development of these curves can be found in [39].

4.3 Correct integration of compressors and expanders

From PA it is known that for heat exchange above pinch there will be a deficit of heat, whilst below pinch there will be an excess of heat. This implies that, for temperatures above pinch, one should increase the amount of heat available from the hot streams and reduce the amount of heat required by the cold streams. Likewise, for temperatures below pinch, the heat provided by the hot streams should be reduced and the heat required by the cold streams should be increased. This is commonly referred to as the Plus/Minus-principle [2]. With regard to the Plus/Minus-principle, a compressor adds heat to the system and should preferably be placed above pinch [1]. Expansion provides cooling and should preferably be placed below pinch [1].

[2]The Plus/Minus-principle is a consequence of having two distinct separated regions of heat deficit and heat surplus. Linnhoff et al. [47] among others [44][67] have discussed the application of this outcome in various ways.

In order to reduce work consumption, compressors should operate at low temperatures. Common practice in the industry is for compressors to operate at ambient temperature in order to save energy. Compressors add heat to the system and the cooling demand may increase if compressors operate at low temperatures. Similarly, expanders produce more work at higher temperatures. Expansion provides cooling to the system and may increase the need of heating if expanders operate at high temperatures. Thus, there is a trade-off between work consumption and demand for external heating and cooling. Wechsung et al. [71] argued for beneficial compressing and expanding at pinch temperature. This implies that expanders operate at the highest temperature in the region with an excess of heat, and compressors operate at the lowest temperature in the region of heat deficit. New insight have further clarified that if a pressure changing stream requires heat, the input temperature should be at the cold pinch. Equivalently, if a stream supplies heat the input temperature should be at the hot pinch [19].

Pinch compression and expansion is, however, only beneficial under certain conditions. Fu and Gundersen [16] proposed a set of theorems for favorable compression and expansion. The theorems require stream splitting and capital costs should therefore be taken into consideration. The theorems are presented in their entirety by Fu and Gundersen [16] [17]. In short, compression should take place at pinch temperature, at an intermediate temperature below pinch, at ambient temperature or at a new pinch temperature. Similarly, expansion should take place at pinch temperature, at an intermediate temperature above pinch, at hot utility temperature or at a new pinch temperature. This is illustrated in Figure 4.4 and Figure 4.5. The challenge is to determine how much of the stream mass flow should be compressed or expanded at each temperature.

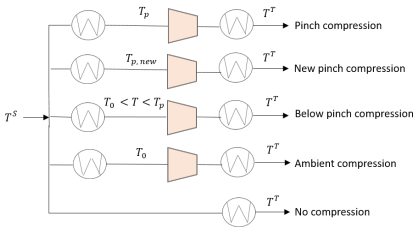

Figure 4.4: Favourable compression.

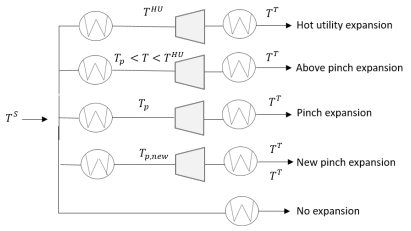

Figure 4.5: Favourable expansion.

The stream split arrangements in Figure 4.4 and Figure 4.5 include a branch for compression and expansion at new pinch points. The original pinch point is removed when the generated compressor heat is more than the heat required. Similarly, a new pinch point is created if the cooling effect from expansion is more than the cooling required by the process. In some cases it will be optimal to place pressure changing units starting at these new pinch points.

4.4 Exergy

When introducing pressure manipulation to enhance energy efficiency the objective of heat and work integration becomes twofold: In addition to minimize the utilities one would aim to minimize the total energy consumption, i.e. minimize the work required by the compressors and maximize the work produced by the turbines. Heat and work are two forms of energy of different quality. Mechanical work is of high quality and can effectively be converted to other forms of energy. Heat, however, is of low quality. A one-to-one substitution of work to save heat is therefore undesirable.

Exergy is a measure of energy quality in the sense that different energy forms have different capabilities to generate work [32]. Exergy is the useful energy and may be destroyed by irreversibilities opposed to energy, which is always conserved. Thus, one would aim to minimize the exergy destruction in a system. The exergy of heat at temperature T is the maximum amount of work that can be extracted when the system is brought to equilibrium with the surroundings. Equation 4.6 and Equation 4.7 express the exergy of an amount of heat Q at temperature T, for T above and below surrounding temperature T_0, respectively. The exergy content of an amount of heat is the heat load multiplied with the Carnot-factor.

$$Ex = Q\left(1 - \frac{T_0}{T}\right), \qquad T \geq T_0 \tag{4.6}$$

$$Ex = Q\left(\frac{T_0}{T} - 1\right), \qquad T < T_0 \tag{4.7}$$

Exergy represents the ability to produce work. Electricity and mechanical energy can be completely converted into work, neglecting minor losses that will always be present in practice. Throughout this report, the work consumed by compressors and the work produced by turbines are assumed to be 100% exergy.

Exergy captures all forms of potential to convert energy into work. By introducing exergy to the objective function the optimization of heat integration is no longer limited to only thermal sources of energy. Minimizing exergy consumption ensures maximum energy efficiency.

Chapter 5

Genetic algorithms

This chapter is an introduction to genetic algorithms (GA). The purpose is to provide the basic understanding of the general principles behind this type of search technique. This chapter is mainly based on the books by Gen and Cheng [23] and Sivanandam and Deepa [64], if not otherwise are being stated. The first section introduces the principle structure of GAs. Encoding of variables is a primary concern in the design of these algorithms and will be discussed in Section 5.2. Fitness evaluation is a fundamental component in GAs and the following section shortly introduces the main tasks of the fitness evaluation process. This step is highly problem dependent and a thorough study of the fitness environment for the algorithms developed in this thesis is given i Chapter 6. GAs require a set of initial feasible solutions, hence, Section 5.4 presents common initialization procedures. The next four sections concern the generation of new and better solutions. These steps are selection, crossover, mutation and replacement. Different search termination criteria is listed in the following Section 5.9. In order to increase the robustness against convergence to local optima, one of the algorithms developed in this thesis has implemented crowding. Section 5.10 therefore provides the essentials of crowding techniques. The quality of performance of GAs depend on the balance between exploration of the search space and exploitation of favourable areas of the search space. Schema theory provides insight into how GAs exploit the search space and is presented briefly in Section 5.11. Finally, GAs are powerful search techniques for problems that are otherwise difficult to solve. However, there are inherent limitations to such heuristic search methods. Advantages and limitations are briefly discussed in Section 5.12.

5.1 Principle structure of genetic algorithms

GAs are powerful stochastic search techniques inspired by evolution and natural selection. GAs are gradient-free methods, which makes them suitable for problems that are difficult to solve for gradient-based methods; such as problems involving discontinuous functions, discrete and mixed discrete-continuous design variables and multimodal problems.

GAs maintain a population of individuals $P(t)$. Each individual represents a candidate solution to the problem under consideration. Figure 5.1 illustrates a flow sheet representation of the principle steps in GAs. Two major processes are essential: the evaluation process and the process of generating new populations. The value of an objective function $f(x)$ is termed the fitness and the evaluation process is referring to the process of evaluating the objective function for different solutions. The evaluation process distinguishes bad solutions from better ones in order to favor good solutions in the generation of new populations.

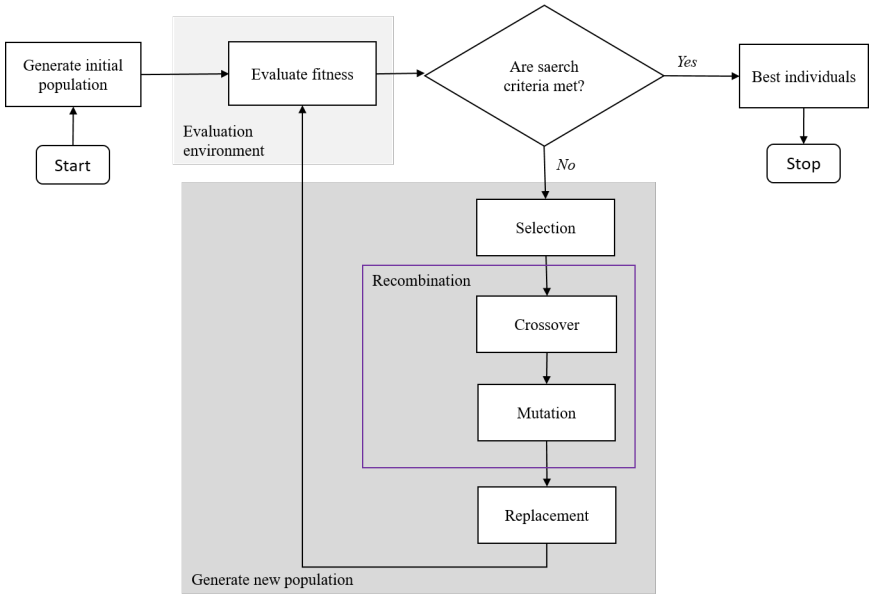

Figure 5.1: Principle structure of GAs.

The process of generating a new population of fitter individuals consists of four operations: selection, crossover, mutation and replacement. The selection process withdrawal a subgroup from the population, which possesses a higher average fitness than the current population. The subgroup is often referred to as the mating pool. The individuals in the mating pool undergo a stochastic recombination process through crossover and mutation. Crossover combine parts from two individuals to form new ones and mutation creates new

individuals from making random changes to a single individual. These steps are guided by the fitness of the individuals. The new individuals are called the population of offsprings, $C(t)$. Individuals in the parent population $P(t)$ are replaced by fitter individuals from the population of offsprings in order to create a new population, $P(t+1)$. The process continues until a predefined stopping criteria is reached.

GAs are a class of search methods combining directed and stochastic search techniques. An appropriate balance between exploration of the search space and exploiting accumulated information is essential in developing an efficient algorithm. Exploitation means that, during the search, the algorithm uses information obtained in the past (about previously visited points in the search space) in order to determine smaller regions that are promising for future search. Exploration is the procedure that obtains new information; the algorithm visits new regions in the search space in order to find promising subregions. In GAs, the selection mechanism exploit the accumulated information by directing the search towards promising regions. New regions are explored through the recombination processes. Other importances while considering GAs are how the fitness of the individuals are being measured and how the individuals are represented. Whilst the evaluation process and the selection process are performed on real values, the crossover and mutation processes are referred to as genetic operators as they perform on a gene representation of the solutions. The following sections examine the basic components of GAs.

5.2 Encoding of variables

The hereditary information of an organism is called its genotype. The observable properties of the genotype is referred to as phenotype. The genetic information in your genes, genotype, decides the colour of your eyes, phenotype. Similarly, in a GA the genes are the encoded version of each set of design variables defining a candidate solution and the encoding of the entire solution is termed genotype. The genotype representation of a solution is often referred to as a chromosome. The real number representation of a solution is called the phenotype, which is the solution set to the candidate solution. An illustration of the terminology presented above is illustrated in Figure 5.2. Each chromosome is a collection of genes which must be decoded into phenotype for a real value representation.

The fitness of an individual is expressed as a function of the phenotype, whilst the genetic operators perform on genotypes. A key issue when using GAs is how solutions to a problem are encoded into genotype. Traditionally, encoding is carried out using binary strings. Binary representation is the encoding strategy used for both algorithms developed in this thesis. In practice, encoding of variables means a discretization of the search space, illustrated in Figure 5.3. Figure 5.3 shows the decoding process from binary representation into real value representation and thus on to fitness evaluation.

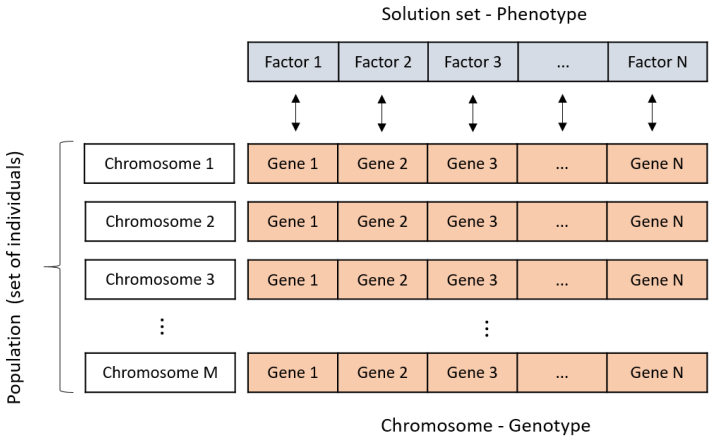

Figure 5.2: Terminology used in the field of GAs.

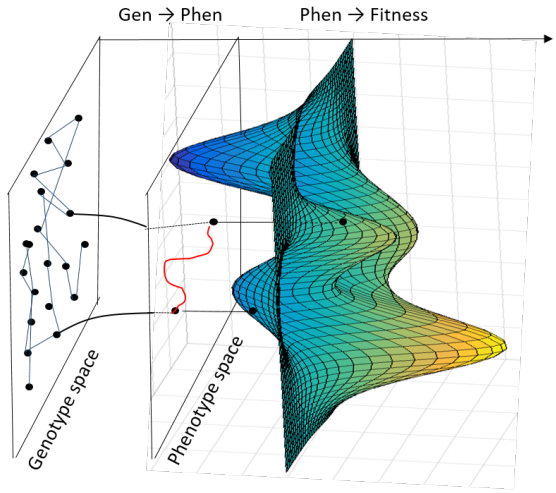

Figure 5.3: The fitness is a function of real values in phenotype space. The real values are decoded from the binary representation in genotype space.

An illustration of binary encoding is given in the following example.

Example 5.1. *Consider maximizing the objective function below (Sivanandam [64]),*

$$f(x) = x^2, \qquad x \in \{0, 31\}.$$

$2^5 = 32$*, hence* 5 *bits are required in order to represent the* 32 *possible solutions in binary strings. Figure 5.4 illustrates eight random generated bit strings of size five. Each gene represents a real number between* 0 *and* 31*. The real number representation is the phenotype, illustrated in the blue column.*

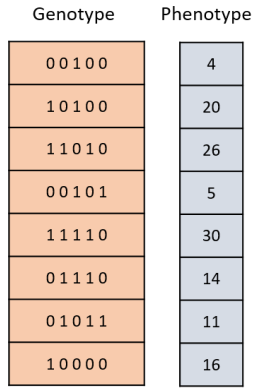

Figure 5.4: Genotype and phenotype representation of eight possible values of $x \in \{0, 31\}$.

In Example 5.1, the phenotype is the real number representation of the bit strings. In general, one is required to define a decoding function that maps the chromosomes consisting of genes into solutions represented by decision variables. The number of bits needed in order to represent each variable must be decided according to a required precision. Example 5.2 illustrates a more general application of binary encoding.

Example 5.2. *Consider Rastrigin's function*[1]*,*

$$R(x) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2), \quad x \in [-5.12, 5.12] \ \ \forall i = 1, 2$$

Suppose that a precision of two decimal places for each variable is required. The resolution, $\Delta x = 10^-2$*, depends on the upper and lower bounds on the decision variables and the number of bits needed, nbits, such that*

$$\Delta x \leq \frac{x_{ub} - x_{lb}}{2^{nbits} - 1} \tag{5.1}$$

The number of bits required can then be calculated as follows

[1]Rastrigin's function is a non-convex function often used to test the performance of genetic algorithms. It is a typical example of non-linear multimodal function

$$nbits \geq \frac{ln(\frac{x_{ub}-x_{lb}}{\Delta x} - 1)}{ln2}. \tag{5.2}$$

That is, 11 bits are required to represent x_1 and x_2 and the total length of the chromosome equals bit size $11 + 11 = 22$. The string $< 10010110111001001110 11 >$ would be a chromosome representing one possible solution $< x_1, x_2 >$. The value of x_1 is determined by decoding the first gene (the first 11 bits) and x_2 the second gene (the last 11 bits). The real values are decoded from the expression below,

$$x_i = b_i \Delta x + x_{lb}, \quad \forall i = 1, 2 \tag{5.3}$$

where b_i is the real value of the binary string representing variable x_i. Hence, $x_1 = 0.92$ and $x_2 = -3.54$

Binary encoding has been the traditional way to map characters from genotype to phenotype space. However, binary encoding encompasses several drawbacks. One is the discretization of the variable range. High precision of variable representation can be obtained by increasing the number of bits. However, an increased number of bits will increase the size of the search space. Another issue is the hamming cliff [23]. Two parameters may have a large hamming distance[2] while belonging to points in phenospace that are very close. In Example 5.2, $< 01111111111 >$ and $< 10000000000 >$ are neighbouring points in phenotype space, but they have maximum Hamming distance in genotype space. For the pair to be equal in genotype space, all bits have to change simultaneously, which is very unlikely to occur. Hence, binary encoding will not preserve the locality of points in phenotype space. Other representation strategies have been suggested in the literature, such as real number-encoding and data structure encoding, but will not be discussed here. For further information about other encoding systems the reader are referred to the book by Gen and Cheng [23].

5.3 Fitness evaluation

GAs are population based search techniques where all individuals within a population are evaluated by the fitness function. If a GA is applied to Example 5.1, the fitness is obtained directly by substituting the decoded variables into the objective function. The fitness values of the population in Example 5.1 are displayed in Figure 5.5.

The fitness evaluation must be more sensitive than just detecting what is a good solution as opposed to a bad solution. A quality measure is required to accurately score the

[2]The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. In other words, it measures the minimum number of substitutions required to change one string into the other.

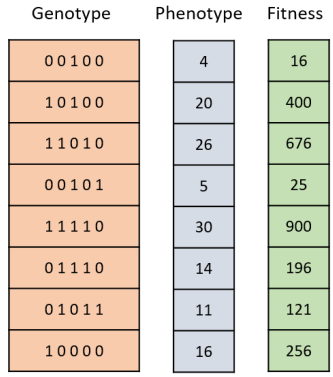

Figure 5.5: Fitness values of the individuals in Figure 5.4.

solutions based on fitness values so that a complete solution can be distinguished from a more complete solution. In complex optimization problems the evaluation process is not a simple step. Often, the problem at hand must be adapted to a genetic algorithmic approach. In this thesis, an LP model is defining the fitness function. In order to identify optimal heat and work integration of process streams, the LP model requires information about how pressure change affects the heat distribution in the process. This information is delivered by the GA. A thorough review of the fitness evaluation in the algorithms developed in this thesis is given in Chapter 6.

Because GAs are a stochastic search technique, many iterations are required. In each iteration, all new individuals in the population are going through a fitness evaluation. Thus, high computational speed is an important property of a fitness function.

5.4 Initialization of population

GAs require an initial population of individuals. The initial population should ideally converge towards the best solution. Therefore, it is important to choose individuals in the first population so that the algorithm will find the best final result, in terms of solution quality and computational time. There are several ways of selecting the starting point depending on the problem characteristics. This section briefly describes three general approaches that are commonly used: random initialization, uniform initialization and biased initialization.

The three approaches are illustrated in Figure 5.6. With random initialization a random value is assigned to each gene. The value is chosen within the range of feasible values. By choosing the values randomly, the whole solution space is represented. Additionally, random initialization is simple to use and easy to implement. For more careful coverage of

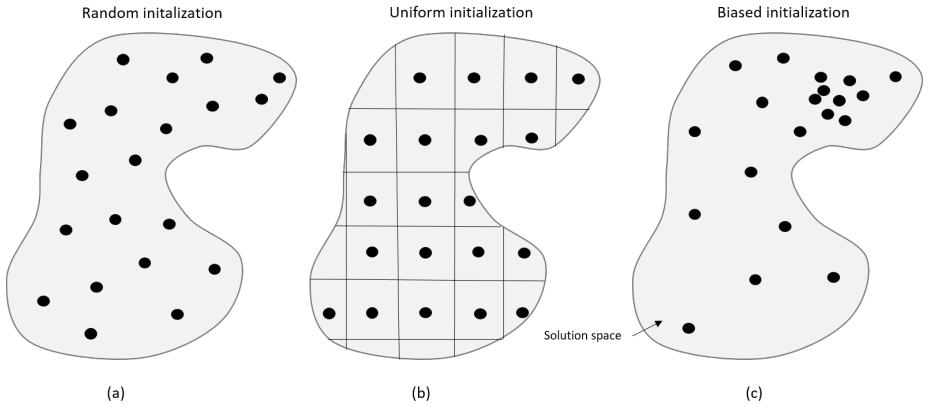

Figure 5.6: Graphical illustration of three approaches to population initialization: (a) random initialization, (b) uniform initialization and (c) biased initialization.

the search space, one can implement a procedure to ensure that the population is uniformly distributed, i.e. dividing the search space into subregions and make sure that at least one sample is drawn from each subregion. If there exist prior knowledge about a favourable region in the solution space, heuristics can be applied in order to bias the initial population towards the promising region. The most beneficial combination of individuals in the initial population depends on the characteristics of the search space and the problem under consideration.

5.5 Selection

The balance between exploration of the search space and exploitation of the accumulated information can be adjusted by the selective pressure [6]. The selection process is the driving force of the exploitation of the accumulated information and directs the genetic search towards promising regions in the search space. With too much force, the search will terminate prematurely; with too little force, the progress will be slower than necessary. It is therefore of great interest to know the distribution of the selection probabilities for different selection methods.

In practice, selection is the process of choosing two parents from the current population to create new individuals. Selection improves the quality of the population by assigning a higher probability of selection to the individuals of high quality opposed to the individuals of low quality. Many methods on how to perform the selection have been proposed. Common types are as follows:

- Roulette wheel selection
- Elitist selection
- Tournament selection
- Linear rank selection

Roulette wheel selection, often referred to as proportional selection, is one of the traditional selection strategies proposed by Holland [34]. The basic idea is to determine a selection probability P_i for each individual i proportional to the fitness value,

$$P_i = \frac{f(c_i)}{\sum_{i=1}^{N} f(c_i)}, \tag{5.4}$$

where $f(c_i)$ is the fitness value of chromosome c_i and N is the number of chromosomes in the population. A roulette wheel model displays these probabilities. The selection process is based on spinning the roulette wheel a number of times equal to the population size, each time selecting a single chromosome to the new population. A slice of the roulette wheel is assigned to each individual. The size of the slice, representing the selection probability, is proportional to the fitness value of the respective individual. An individual with twice as good fitness as another individual will have twice the likelihood to be selected. The selection probabilities for the eight individuals in Figure 5.5 are illustrated in Table 5.1.

Individual	1	2	3	4	5	6	7	8
Fitness value	16	400	676	25	900	196	121	256
P_i	0.01	0.15	0.26	0.01	0.35	0.07	0.05	0.10

Table 5.1: Roulette wheel selection probabilities of individuals in Example 5.1.

Elitist selection is a deterministic selection procedure. The elitist operation preserves the best solution(s) obtained so far at any stage and pass that on to the next generation in order to ensure that the best achievement so far does not get lost during genetic operations. Elitist selection is generally used in combination with other selection procedures. Both algorithms developed in this thesis make use of elitist selection.

Other types of selection procedures contain random and deterministic features simultaneously. A typical example is tournament selection. Tournament selection from the population in Figure 5.5 is illustrated in Figure 5.7. This method randomly chooses a set of chromosomes from the population and keep the best chromosome for recombination. The number of chromosomes chosen from the population is called the tournament size. The most common value of the tournament size is 2, called binary tournament. Pairs of chromosomes are drawn at random from the population. From the selected pair, the chromosome with highest fitness is the tournament winner and is inserted in the mating pool. This process continues until the mating pool is full. The mating pool will further be subject to crossover and mutation in order to generate a new population.

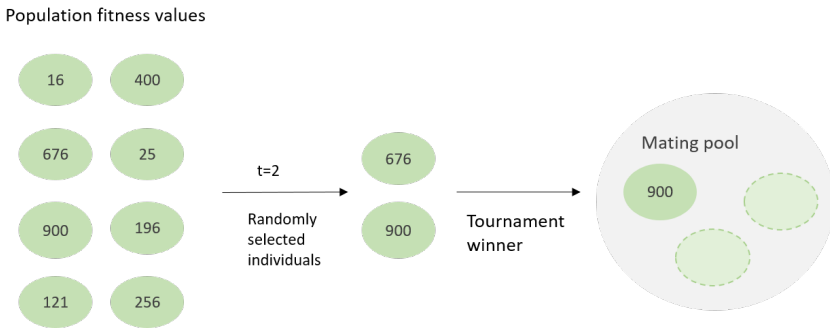

Figure 5.7: Tournament selection.

The selection pressure can be derived assuming the individuals are ordered according to their fitness value, such that $f(c_1) \leq f(c_2) \leq ... \leq f(c_N)$. For a population of size N and tournament size t, Bäck [2] derived the selection probability of individual i to be

$$P_i = N^{-t}((N - i + 1)^t - (N - i)^t). \tag{5.5}$$

For the fitness values in Figure 5.5, Table 5.2 displays the ranking according to fitness value and the corresponding selection probabilities, calculated using Equation 5.5. In this case $N = 8$ and the tournament size equals 2.

Individual	1	2	3	4	5	6	7	8
Fitness value	16	400	676	25	900	196	121	256
Rank	8	3	2	7	1	5	6	4
P_i	0.02	0.17	0.20	0.05	0.23	0.11	0.08	0.14

Table 5.2: Binary tournament selection probabilities.

In comparison to the Roulette wheel selection procedure, the selection probabilities are more evenly distributed in binary tournament selection, which means a more moderate trade-off between exploration of the search space and exploitation of accumulated information. The less fit individuals have a higher chance of getting selected. Increasing the tournament size will reduce the chances of the less fit individuals to be selected to the mating pool, leading to a decrease in diversity. The first algorithm developed in this thesis utilizes a tournament selection process.

In the roulette wheel selection procedure the selection pressure is proportional to the chromosome fitness. This exhibit undesirable properties when the fitness values among

individuals in a population differ very much or very little. For example, in early generations one may experience a tendency of very fit individuals to dominate the selection process. This can be observed in Table 5.1. The two most fit individuals are assigned very high selection probabilities. The other chromosomes will have too few chances of getting selected, resulting in rapid convergence. Linear ranking selection was introduced by Baker [4] to mitigate this problem. In Linear ranking the individuals are stored and ranked according to fitness values. Rank 1 is assigned to the best individual and rank N to the worst individual. The selection probabilities are linearly assigned to each individual according to their rank, given by

$$P_i = \frac{1}{N}\Big(\eta^+ - (\eta^+ - \eta^-)\frac{i-1}{N-1}\Big), \qquad i \in \{1, ..., N\}. \tag{5.6}$$

Here, i is the respective rank and the constants η^+ and η^- determine the slope of the linear function. $\eta^- = 2 - \eta^+$ and $\eta^+ = c$, for $1 < c \geq 2$, thus c controls the selection bias. For c approaching 1 the selection probability becomes more uniformly distributed among the individuals. The bias towards higher selection probabilities for individuals with higher fitness increases with higher values of c. The value assigned to c depends on the desirable degree of exploitation of favourable areas in the search space. Calculations of selection probabilities for the eight individuals in Figure 5.5 are listed in Table 5.3 for $c = 1.5$.

Individual	1	2	3	4	5	6	7	8
Fitness value	16	400	676	25	900	196	121	256
Rank	8	3	2	7	1	5	6	4
P_i	0.06	0.15	0.17	0.08	0.19	0.12	0.10	0.13

Table 5.3: Selection probabilities according to a linear ranking of fitness values.

From Table 5.3 one can observe even more evenly distributed selection probabilities among the individuals than for tournament selection. The weak individuals have a better chance of being selected, leading to a higher diversity in the population.

5.6 Crossover

Crossover is the process of recombining the genetic material in parent chromosomes to produce offsprings that share characteristics with the parents. The crossover operator will have a dual effect on the search process. First, characteristics from the parents passes on to the offsprings in an effort to preserve favourable characteristics. Second, the recombination of gene material will direct the search into new regions. Crossover is illustrated in Figure 5.8 for three commonly used methods: single point crossover, two-point crossover and uniform crossover.

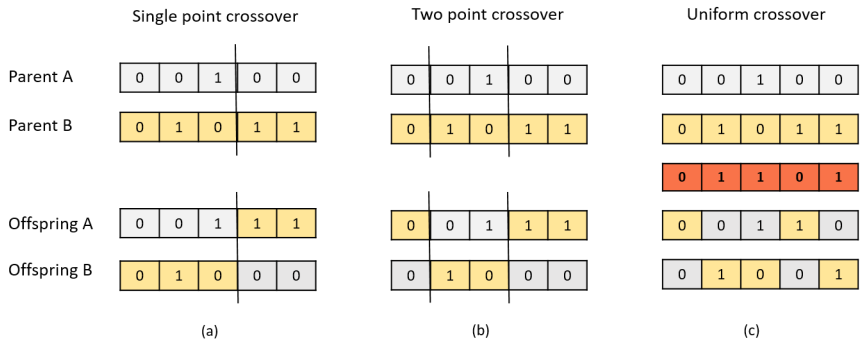

Figure 5.8: Three common crossover methods: (a) single point crossover, (b) two point crossover and (c) uniform crossover.

Single point crossover uses a randomly generated crossover point in order to perform recombination. The two mating chromosomes, parent A and parent B, are cut once at corresponding points. The sections after the cuts are exchanged, creating two new offsprings. Apart from single point crossover, many different crossover algorithms have been devised. Other methods often involve multiple crossover points. A two-point crossover procedure is illustrated in Figure 5.8(b).

Uniform crossover is another crossover type, featuring multiple crossover points. In uniform crossover, the recombination is performed according to a random generated binary mask of the same length as the chromosomes. For each bit position there will be a fifty percent probability that the value in that position will be drawn from each parent, as shown in Figure 5.8(c). Uniform crossover may potentially add a large number of crossover points. Adding additional crossover points often reduces the performance of the algorithm because gene patterns are more likely to be disrupted. This is more thoroughly explained in Section 5.11. However, multiple crossover points lead to a more exploratory search of the solution space, which in some cases may be beneficial. The crossover rate controls the number of new individuals created through crossover and is usually kept high in order for the algorithm to evolve towards better solutions.

5.7 Mutation

The main objective of mutation is to prevent the algorithm from being trapped in local optima. Where crossover exploit the accumulated information in the population to create better individuals, mutation enhance the exploration of the search space. In biology, mutation is a random change in a DNA sequence that builds up a gene due to mistakes when the DNA is copied. Similarly, in GAs, a mutation is a random alteration of bits changing

a chromosome into a complete different candidate solution. The process simply consists of flipping the bits, as illustrated in Figure 5.9. Traditionally, mutation is performed on genotypes, but a number of algorithms have adopted crossover and mutation methods of good performance operating on real values.

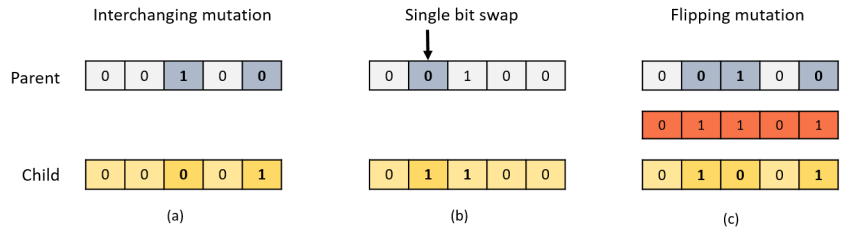

Figure 5.9: Three common mutation methods: (a) interchanging mutation, (b) single bit swap mutation and (c) flipping mutation.

There are many ways to perform mutation on chromosomes. Figure 5.9 illustrates three common mutation methods: interchanging mutation, single bit swap mutation and flipping mutation. In interchanging mutation two bit positions in the string are chosen randomly. The bits corresponding to these positions are interchanged. In single bit swap one bit position is randomly chosen. The corresponding bit is then swapped. In flipping mutation the mutation is based on a random generated mutation mask of the same length as the chromosome. If a bit in the mutation mask takes on value 1, the corresponding bit in the parent chromosome is flipped (0 to 1, or 1 to 0), producing an offspring.

The mutation rate is usually kept low. New individuals are produced from parents of high quality. If the mutation rate is high and heavily disrupt the gene material, the offsprings will not inherent the advantageous features of its parents. The search will therefore seek towards a higher exploration rate in the expense of exploitation of the accumulated information.

5.8 Replacement

During crossover and mutation new offsprings are produced. However, not all parents and offsprings can return to the population of the next generation. Once offsprings are produced, a method must determine which of the current members of the population should be replaced by the new solutions. This process is called replacement. Common replacement strategies are listed below:

- *Random replacement* replaces randomly chosen parent individuals with the offsprings. This strategy provides high exploration rate as all individuals have equal probability of surviving the next generation

- *Weak parent replacement* replaces a weaker parent with a fitter offspring. Newly generated offsprings will replace the weaker of its two parents if it has a better fitness than the parents. This process improves the overall fitness if combined with a selection strategy that selects both weak and fit parents for crossing, if not, the opportunity to replace the weak will never occur.
- *Both parents replacement* replaces both parents in each generation with the offsprings. E.g. two newly generated offsprings will replace both parents. This strategy works well if not combined with a selection process that strongly favours fit individuals. In such cases highly fit individuals may be lost in future generations.

5.9 Search termination

If GAs are ran indefinitely, the global optimum will eventually be found. However, in practice, the algorithm perform a search within a finite range of time. Various stopping conditions are being used depending on the specific problem and solution requirements. A selection of various stopping criteria is listed below:

- *Maximum generation* is the most commonly used criteria to stop the algorithm. The algorithm stops when a specified number of generations have evolved.
- *Elapsed time* terminates the search when a specific time have elapsed.
- *Stall generation* stops the algorithm if there is no improvement in the objective function for a predefined sequence of consecutive generations.
- A *best individual* stopping criteria terminates the search once the fittest individual(s) reaches a predefined convergence value. This criteria brings the algorithm to a fast conclusion, guaranteeing at least one good solution.
- *Worst individual* stopping criteria is similar to the previous one. The search terminates when the least fit individual in the population is not worse than a predefined convergence criteria. This guarantees the final population to possess a certain standard, even though the best individual may not be significant better than the worst.
- A *sum of fitness* termination strategy will terminate the search when the fitness of the entire population reach a convergence value. This strategy guarantees that all individuals in the population will be within a particular fitness range.

5.10 Crowding techniques

In the development of GAs, a major challenge is an appropriate balance between exploration of the search space and exploitation of accumulated information. Too much emphasis on exploration may cause a waist of valuable effort on solutions that are less likely to be good. On the other hand, increasing the exploitation rate may lead to premature convergence to a local optimum. Premature convergence to local optima is a difficulty that commonly arises when GAs are applied to complex problems.

Premature convergence occurs when individuals with higher fitness values attain dominance in the population. In general this is due to the loss of diversity within the population caused by unfavourable selection pressure, poor schema distribution and poor evolution parameter settings. Premature convergence occurs in situations where all individuals in a population occupies only a subspace of the search space, illustrated in Figure 5.10. If the global optimum lies outside the subspace, GAs are likely to converge towards the local optimum within the subspace.

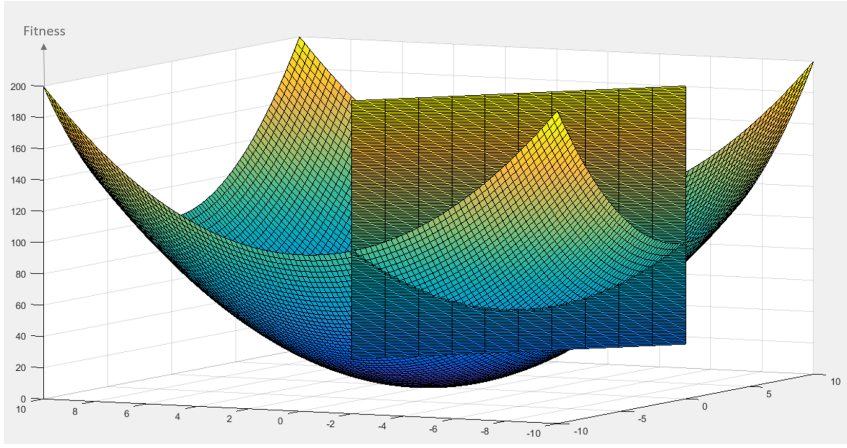

Figure 5.10: Premature convergence is highly likely when e.g. in three dimensions all individuals in the population occupies a two dimensional place.

To avoid premature convergence, one has to maintain the diversity within the population during the evolution process. In practice, it is common to control certain parameters [10]. The selection pressure can be adjusted to increase the exploitation of accumulated information, whilst higher mutation rate increases the exploration of the search space. A wide range of methods to avoid premature convergence is thoroughly studied in the literature [59][59][22][53]. Crowding is a common technique preserving diversity within a population and is applied to one of the algorithms developed in this thesis.

The initial crowding method was developed by De Jong [36] and consists of pairing offsprings with similar parent individuals. For each pair a decision is made as to which of them will survive the next generation. The latter is commonly referred to as the replacement phase, whilst the former is the pairing phase. Comparing of individuals is computationally demanding. Mahfoud [50][51] realized that the offsprings are likely to be similar to their parents. Thus, offsprings competing for survival with their most similar parent can be used to efficiently preserve the diversity in a population, requiring less computational effort. An outline of this crowding method is illustrated in Algorithm 1.

Algorithm 1 Crowding algorithm

1: parent individuals are randomly paired
2: with probability P_c, the parents in each pair (p_1, p_2) are recombined. The two resulting offsprings (c_1, c_2) are mutated with probability P_m
Each offspring competes with one of its two parents for survival. Let $d(p_i, c_i)$ denote the hamming distance between a parent and a child
3: **if** $d(p_1, c_1) + d(p_2, c_2) < d(p_1, c_2) + d(p_2, c_1)$ **then**
4: p_1 competes with c_1
5: p_2 competes with c_2
6: **else**
7: p_1 competes with c_2
8: p_2 competes with c_1
9: **end if**

Depending on how the replacement phase is carried out, there are two main types of crowding: deterministic and probabilistic [53]. Deterministic crowding selects the fittest individual for the next generation. Probabilistic crowding selects the surviving individual from a probabilistic formula based on relative fitness value. Deterministic crowding develops an exploitative strategy which highly favors fit individuals. Let P_r denote the probability that an offspring c replaces parent p in the population. In deterministic replacement P_r can be expressed as follows:

$$P_{\mathrm{r}} = \begin{cases} 1 & \text{if} \quad f(c) > f(p) \\ 0.5 & \text{if} \quad f(c) = f(p) \\ 0 & \text{if} \quad f(c) < f(p) \end{cases} \tag{5.7}$$

The fitter individuals will always survive, leading to a loss of the genetic material in the less fit individuals. This may be a disadvantage if it leads to premature convergence. Probabilistic crowding promotes higher exploration of alternative solutions, with a selection probability depending on the fitness values such that,

$$P_r = \frac{f(c)}{f(c) + f(p)}. \tag{5.8}$$

For a more controlled trade-off between exploration of the search space and exploitation of accumulated information, generalized crowding was introduced by Galan and Mengshoel [22]. In generalized crowding the degree of exploration can be controlled by means of a parameter ϕ, named the scaling factor. The scaling factor allows for a wide range of selective pressures to be applied: For higher values of ϕ it is more likely that areas of less good solutions in thesolution space are included in the search. Lower ϕ increases the degree of exploitation of accumulated information. Galan and Mengshoel [22] established the survivor between parent p and offspring c as follows,

$$P_{\mathrm{r}} = \begin{cases} \frac{f(c)}{f(c)+\phi \times f(p)} & \text{if} \quad \text{f(c) > f(p)} \\ 0.5 & \text{if} \quad f(c) = f(p) \\ \frac{\phi \times f(c)}{\phi \times f(c)+f(p)} & \text{if} \quad f(c) < f(p) \end{cases}, \tag{5.9}$$

where f is the fitness function and $\phi \in \{\mathbb{R} \mid \phi \geq 0\}$ denotes the scaling factor. The major advantage with generalized crowding is that ϕ can easily be adjusted, allowing for a wide range of replacement rules. The optimal value of ϕ are problem dependent. A GA with crowding is developed in this thesis and presented in Chapter 6.

The crowding technique aim to preserve population diversity and prevent premature convergence by eliminating the most similar individual whenever a new one enters the population. The method has been widely used and found effective for problems of all levels of difficulty [59]. Using crowding, one has been able to solve much harder problems than those solvable with traditional hill-climbing techniques [59]. Maintaining bit wise diversity is, however, a difficult issue. Ideally, the method should encourage the population to arrive at a stable mixture of different solutions. However, crowding is a method which strives to maintain the diversity of the pre-existing mixture in the population, which may not be sufficient [50] [59].

5.11 Schema theory

Although *survival of the fittest* has seemed to work well in the real world there is still a question about how the concept of GAs works in a computer. Various methods are proposed to gain insight into the behaviour of GAs, such as schema theory, Markov chain theory and dimensional analysis. The following section is considering schema theory, since this is considered the fundamental theory. A study of other theories is provided by Pandey et al. [59].

Schema theory was developed by Holland [34] in order to explain why GAs have the tendency to converge towards optimal solutions. Schema representation is a useful notation for detecting similarities among individuals in a population. Holland [34] formulated

the schema theorem for canonical[3] GAs to demonstrate how schema evolve through successive generations. The algorithms developed in this thesis are not canonical; however, knowledge about schema theory may be found useful in order to gain a better understanding of the underlying mechanisms of GAs. Holland [34] defined a schema as describing a subset of genotype chromosomes with similarities at certain string positions. The strings below

$$< 100 >$$
$$< 101 >$$

are similar in the sense that they are identical at which the last position is ignored. Regarding $*$ as a symbol for a position that may take value 1 or 0, the strings above can be represented by $[1\ 0\ *]$. $H = [1\ 0\ *]$ is said to be a schema, identifying the two strings above. For further illustration, a schema can be viewed as a hyperplane[4] in a ℓ-dimensional space, where ℓ is the length of the chromosome string. The schema mentioned above is in the three-dimensional space and is illustrated in Figure 5.11.

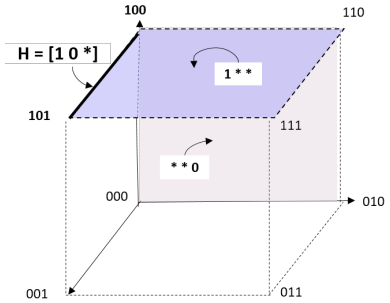

Figure 5.11: Schema in a three dimensional search space.

The schema theorem, proposed by Holland [34], provides a model for the expectation of schema survival to the next generation. Following is a consideration of the effect of selection, crossover and mutation on a schema, leading to the schema theorem.

The effect of selection on a schema

$H = [1\ 0\ *]$ is said to be a schema identifying the two strings $< 1\ 0\ 0 >$ and $< 1\ 0\ 1 >$. Similarly, a string belonging to schema H is said to be an instance of H.

[3] Canonical GAs are algorithms with proportional selection, single point crossover and bit-wise mutation.

[4] A hyperplane is a subspace of one dimension less than its ambient space. In a 3-dimensional space the hyperplanes are of 2 dimensions.

Definition 5.1. *Let x denote a decision variable vector. Then x is said to be an instance of schema* $[1\ 0\ *]$ *if and only if x matches* $[1\ 0\ *]$ *at every position not containing* $*$.

For example, string $[1\ 0\ 0]$ is said to be an instance of schema $[10*]$ and $[**0]$, among others. The number of instances of schema H in generation t, is commonly noted as $m(H,t)$. The number of instances of a specific schema depends on the order of the schema.

Definition 5.2. *Schema order* $o(H)$ *is the number of fixed positions in a gene. For example* $o([1**]) = 1$.

The order of a schema reveals information about how many instances of a schema exists within a population. A schema of order o represents 2^{l-o} different strings of length l. Possible schema orders and the equivalent number of instances for a three dimensional space are listed in Table 5.4. Hence, schema of higher order represents less strings than one of lower order. This will be shown to have an impact on the ability of a schema to survive the next generation.

$o(H)$	No. of instances of schema H
0	8
1	4
2	2
3	1

Table 5.4: Orders of schema in a three dimensional space and the corresponding number of instances of the schema.

The average fitness of schema H at iteration t is denoted $f(H,t)$ and can be calculated from the sum of fitness values $f(x)$ of the individuals being instances of schema H divided by the number of instances of schema H in generation t, $m(H,t)$ such that,

$$f(H,t) = \frac{\sum_{x \in H} f(x)}{m(H,t)}. \tag{5.10}$$

The degree of which a schema survives the selection process is measured as the expected number of instances of schema H in the population at time $t+1$,

$$E[m(H,t+1] = m(H,t)\frac{f(H,t)}{\bar{f}} \tag{5.11}$$

where $\bar{f}$ is the average population fitness at time t. Equation 5.11 reveals that schema with higher average fitness than the average population fitness are more likely to appear in the next generation.

The effect of crossover on a schema

The defining length of a schema provides information about the structure of the schema. A schema with fixed positions located close to one another are more likely to survive to the next generation during crossover.

Definition 5.3. *Schema defining length $d(H)$ is the distance between the two furthest fixed bits in a schema. For example* $d([1\ *\ *\ *\ 1\ *\ 0]) = 6$.

Consider two parents, P_1 and P_2, where P_1 is an instances of schema $H = [*\ 1\ 0\ *\ *]$ and P_2 is not. The two parents are subject to single point crossover, illustrated in Table 5.5. A schema survives a crossover operation if one of the parents is an instance of the schema and one of the offsprings is an instance of the schema. The schema will be broken by the location of the crossover point, unless the second parent is able to repair the disrupt gene.

H = [* 1 0 * *]			
$P_1 = [1\ 1\ \underline{0}\ \ 1\ 0] \in H$	$\rightarrow$	$C_1 = [1\ 1\ 0\ 1\ 1] \in H$	H survived
$P_2 = [1\ 0\ \underline{1}\ \ 1\ 1] \notin H$		$C_2 = [1\ 0\ 1\ 1\ 0] \notin H$	
$P_1 = [1\ \underline{1}\ 0\ \ 1\ 1] \in H$	$\rightarrow$	$C_1 = [1\ 1\ 1\ 1\ 1] \notin H$	H destroyed
$P_2 = [1\ \underline{0}\ 1\ \ 1\ 1] \notin H$		$C_2 = [1\ 0\ 0\ 1\ 1] \notin H$	

Table 5.5: Survival of schema H after crossover.

The crossover point is selected randomly among $\ell - 1$ possible positions, where ℓ is the length of the chromosome string. The probability that the crossover point occurs within the defining length, disrupting the schema, is therefore

$$\frac{d(H)}{\ell - 1} \tag{5.12}$$

A schema may be conserved if both parents happen to contain parts of the schema at the correct places. Thus, the upper bound of the probability that schema H is being destroyed is then,

$$D_c(H) \leq P_c\ \frac{d(H)}{\ell - 1} \tag{5.13}$$

where P_c is the crossover probability. The probability of the schema surviving the crossover then becomes,

$$S_{\mathrm{c}}(H) = 1 - D(H)_{\mathrm{c}} \geq 1 - P_{\mathrm{c}} \ \frac{d(H)}{\ell - 1} \tag{5.14}$$

Studying Equation 5.14, schema of lower order are more likely to survive a single point crossover operation. This means that schema with fixed bit positions closely located are less likely to be disrupted and will survive the next generation.

The effect of mutation on a schema

Mutation is applied bit-wise, which means that in order for a schema to survive, all fixed bits in the schema must remain unchanged. The probability of a bit not changing is,

$$(1 - P_m). \tag{5.15}$$

The probability that all fixed bits in the schema do not change and the schema survives is then,

$$S_m(H) = (1 - P_m)^{o(H)}. \tag{5.16}$$

Hence, schema of lower order are more likely to survive a bit-wise mutation process, meaning that schema with less fixed bit positions are more likely to survive.

The schema theorem

The combining effect of selection, crossover and mutation on a schema H constitute the schema theorem developed by Holland [34],

$$E[m(H, t+1)] \geq m(H, t) \frac{f(H, t)}{\bar{f}} \left(1 - P_c \frac{d(H)}{1 - \ell} \right) (1 - P_m)^{o(H)} \tag{5.17}$$

in which states that schema with low defining length, low order and above average population fitness will be more likely to survive the next generation. Exploration will dominate early stages in the search process, and over time the GA increasingly converge towards what it has detected as the most fit schema. The fact that the GA can identify the fittest part of the search space very quickly is a powerful property, however, since the GA always operates on populations of finite size, there is inherently sampling errors in the search, and

in some cases the GA can magnify a small sampling error, causing premature convergence. Schema theory is not a fundamental theory of the behaviour of GAs, but rather an insight into the mechanisms that brings the search towards promising regions in the search space.

5.12 Advantages and limitations of genetic algorithms

GAs are population based search techniques that do not require computations of gradients. They can cover a large range of the search space, being less likely of getting stuck in local optima than gradient-based methods. GAs are suitable when the search space is large, complex and poorly understood. These methods can handle discrete, mixed discrete-continuous variables and discontinuous functions. This make GAs particular applicable to heat and work integration problems, which has proven to result in combinatorial, non-differential and non-convex problems. GAs are also suitable for multiobjective optimization.

Although GAs perform a more systematic search than completely random methods, they still require a large number of function evaluations. An important property of GAs is that they are easily parallelized. The objective function of several individuals in a population could be calculated simultaneously on different processors. The algorithms developed in this thesis do not utilize parallelization, but this is an inherent property that is suggested to be taken advantage of in future work.

The main limitation using GAs, which also applies to other metaheuristic approaches, is the inability to guarantee global optimum. For a minimization problem, a GA provides an upper bound. One cannot prove optimality of the bound unless there exists a good lower bound that matches the solution found. The quality of the solution, and whether or not the algorithm will get stuck in a local optimum, are highly dependent on the control parameter settings. The literature provides reasonable values to various parameters such as the population size, selection pressure, crossover and mutation rate, but in practice these values are chosen by trial and error.

Chapter 6

Methodology

This chapter presents a two-level optimization model for energy targeting of optimal heat and work integration. The two-level model decouples the optimization problem into two loops: The outer loop uses a heuristic search algorithm to fix the inlet temperatures to the pressure changing units. Each evaluation of the heuristic optimizer in the outer loop requires solving the inner loop problem by highly efficient linear programming. Two genetic algorithms are developed for the outer loop; a basic GA and a GA with crowding.

The simultaneous heat and work integration problem involving multiple pressure changing streams is difficult to solve efficiently with one-level optimization. The challenge, however, with a two-level optimization model is to formulate the problem into levels of optimization that will be more efficient than one-level optimization. Section 6.1 presents common characteristics of the heat and work integration problem. The following Section 6.2 discusses why a two-level optimization model is a suitable approach to the heat and work integration problem. Section 6.3 presents the general structure of the model. The two search algorithms developed for the outer loop are presented in the last Section 6.4. The complete implementation of the algorithms are attached in Appendix D

6.1 Characteristics of heat and work integration

The objective of heat and work integration is twofold: In addition to minimize utility consumption, one would aim to minimize the total energy consumption, i.e. minimize the work required by the compressors and maximize the work produced by the turbines. As discussed in Section 4.4, heat and work are two forms of energy of different quality; a one-to-one substitution of work to save heat is undesirable. Therefore, regarding heat

and work integration, optimization with respect to minimum exergy consumption is more representative than optimization with respect to energy.

Introducing pressure changing streams entail two complicating factors: (1) pressure manipulation causes a sudden change in temperature of the fluid dividing the stream into two segments; one segment before and one segment after the unit. The segments change temperature continuously, but the transition between the two segments is discontinuous [58], (2) the inlet and outlet temperatures to the pressure changing units are not restricted to lie within the range of the supply and target temperatures. This implies that a cold stream may act as a hot stream, and a hot stream may act as a cold stream [52].

In order to increase the energy efficiency, pressure changing streams may be split into branches, allowing for pressure change at multiple temperatures. Each branch will generate two new stream segments. This stream split arrangement was presented by Dowling [11], illustrated in Figure 6.1 for a pressure changing stream s.

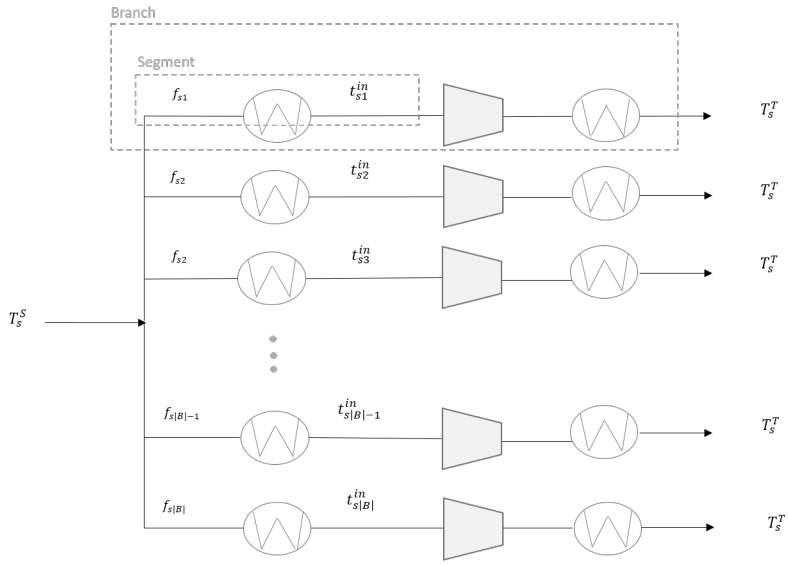

Figure 6.1: Stream split arrangement for pressure changing streams.

Each branch b is assigned a fraction f_{sb} of the mass flow rate of stream s. The inlet temperatures to the pressure changing units become target temperatures of the new stream segments before the unit. Equivalently, the outlet temperatures become supply temperatures of the new stream segments after the unit. Consequently, the pressure changing streams can be modeled as a set of constant pressure stream segments. The identity of the segments is determined by the supply and target temperatures, which is defined by the inlet and outlet temperatures to the pressure changing units.

For constant pressure streams the heat cascade method, in which ensures thermodynamically feasibility, can be applied to locate minimum heating and cooling demands for heat integration of a process. An introduction to the heat cascade was presented in Section 4.2. The pressure changing streams can be modeled as a set of stream segments with continuous temperature change, thus, the heat available from the hot stream segments and the heat required by the cold stream segments in each temperature interval k, can be calculated with Equation 6.1 and 6.2.

$$Q^H_{skbz} = f_{sb}(mc_p)_s \left(T^H_k - max\left[T^H_{k+1}, T^T_s\right]\right) \tag{6.1}$$

$$Q^C_{skbz} = f_{sb}(mc_p)_s \left(min\left[T^C_k, T^T_s\right] - T^C_{k+1}\right) \tag{6.2}$$

Each segment z belongs to a branch b, which is derived from a pressure changing stream s. The fraction f_{sb} determines the amount that flows through the pressure changing unit on branch b. The heat balances for each temperature interval, for both constant pressure streams and pressure changing streams, can then be calculated with Equation 6.3.

$$r_k = r_{k-1} + \sum_{s\in S^H} Q^H_{sk} + \sum_{s\in S^H}\sum_{b\in B}\sum_{z\in Z} Q^H_{skbz} - \sum_{s\in S^C} Q^C_{sk} - \sum_{s\in S^C}\sum_{b\in B}\sum_{z\in Z} Q^C_{skbz} \tag{6.3}$$

Equation 6.3 remains linear if the temperatures defining the intervals and the end states of the stream segments are fixed.

6.2 Design thought of the two-level optimization model

Considering the heat cascade method to identify the energy target for optimal heat and work integration: For variable unit inlet temperatures, the heat balance constraints become bilinear. Also, additional heat balance constraints are required to ensure thermodynamically feasibility. However, the problem remains linear when the unit inlet temperatures are fixed. Being able to separate the placement of the pressure changing units from the heat integration problem would considerably simplify the problem.

The motivation for this thesis is to identify the energy target for optimal heat and work integration involving multiple pressure changing streams. However, for increased practical value, future research should take costs into consideration. The objective of minimizing exergy consumption and the objective of reducing the total costs contradict each other. Considering the two objectives simultaneously may be resolved with multi-objective optimization. Multi-objective optimization can support the decision making with locating a reasonable energy target for the least number of operating units.

With the purpose of including pressure changing streams to enhance heat integration and facilitate an expansion of the problem to include costs, a two-level optimization model is developed. The model decouples the optimization problem of heat and work integration into two loops: The inner loop decision variables define a linear optimization problem that can be efficiently solved with linear programming when the outer loop decision variables are fixed. The inner loop locates the minimum exergy consumption for optimal heat and work integration for fixed unit inlet temperatures. The outer loop searches for the optimal values of inlet temperatures to the pressure changing units. The outer loop requires heuristic optimization, whilst the inner loop, with many decision variables, can be solved with highly efficiently linear programming. Heuristic optimization algorithms are often used when there is no efficient way to find a solution quickly and accurately. These methods do not guarantee a global optimal solution, but often locate very good solutions within reasonable time for problems that are otherwise difficult to solve.

6.3 Two-level optimization model for simultaneous heat and work integration

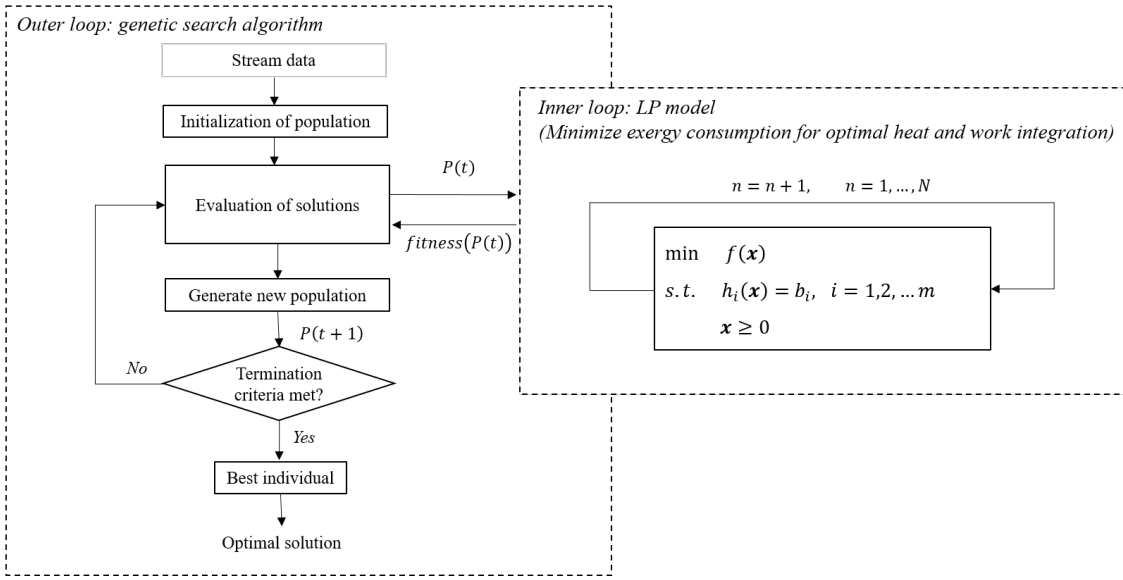

Figure 6.2: Flow chart representation of the two-level optimization model.

Figure 6.2 shows the framework of the two-level optimization model. The outer loop of the optimization process searches for the optimal inlet temperatures to the pressure changing

units. The inner loop locates the energy target for optimal heat and work integration. The outer loop generates a population of promising candidate solutions, $P(t)$. Each candidate solution is a set of inlet temperatures to the pressure changing units. The set of inlet temperatures generates new stream segments to be heat integrated. Optimal heat and work integration among the process streams is established in the inner loop, in which the minimum exergy consumption is calculated. The inner loop determines the optimal stream split arrangements with linear programming in order to minimize the exergy consumption. Whilst the inner loop assigns a quality measure to the candidate solutions, $fitness(P(t))$, the search algorithm identifies the best solution. The iterations continue until a termination criteria is satisfied.

In general, various heuristic algorithms can be applied to the outer loop. The model developed in this thesis was designed considering GAs. Two main reasons why GAs was thought to be successful are: (1) simultaneous heat and work integration of pressure changing streams can be modelled with linear programming as long as the inlet temperatures to the pressure changing units are fixed. Hence, the LP model can operate as a suitable fitness function to a GA, (2) the search algorithm must efficiently handle a large number of design variables and prevent the search from converge towards local optima.

6.3.1 Inner loop optimization

The inner loop consists of a preprocessing step and an LP model formulated by Maurstad [52]. The LP model determines the minimum exergy consumption for optimal heat and work integration, in which the heat cascade method is applied to ensure thermodynamically feasibility. The model requires linear heat balance constraints for each temperature interval. Applying the stream split arrangement illustrated in Figure 6.1, the supply and target temperatures for the stream segments are fixed for fixed inlet and outlet temperatures for the pressure changing units. Compression and expansion are modeled as polytropic processes with 100% polytropic efficiency. Thus, for fixed pressure ratios, the outlet temperatures are calculated with Equation 6.4,

$$T^{out} = T^{in}\left(\frac{P^{out}}{P^{in}}\right)^{\frac{\kappa-1}{\kappa}} \tag{6.4}$$

where P^{in} and P^{out} are the inlet and outlet pressures, and the temperatures are in unit Kelvin. The heat available and the heat required by the hot and cold streams are calculated in the preprocessing step. For the constant process streams, the heat loads are calculated with Equation 4.4 and Equation 4.5. The heat available and the heat required by the pressure changing streams are calculated with Equation 6.1 and 6.2. The LP model is described as follows:

Notation

Indices	Description
s	stream
b	branch
z	stream segment
k	temperature interval

Sets	Description
S	set of streams
S^P	set of pressure change streams
K	set of temperature intervals
B	set of stream splits
Z	set of stream segments

Parameters	Description
Q_{sk}^{CP}	total heat supply/demand in interval *k* for constant pressure stream $s \in S \backslash \left\{S^{\mathrm{P}}\right\}$
Q_{skbz}^{P}	total heat supply/demand in interval k for stream segment z
T_0	ambient temperature
T^{HU}	hot utility temperature
T^{CU}	cold utility temperature
T_{sb}^{in}	inlet temperature to pressure changing unit on branch b
T_{sb}^{out}	outlet temperature from pressure changing unit on branch b
$(mc_p)_s$	heat capacity flow rate of stream $s \in S$

There are one main decision to be made, which is the fraction of the flow rate flowing through each pressure changing unit. This is reflected in variable f_{sb}. The hot and cold utility requirements are the heat residuals entering and leaving the first and last intervals.

Variables	Description
r_k	heat residual from interval k
q^{HU}	hot utility consumption
q^{CU}	cold utility consumption
f_{sb}	fraction of heat capacity mass flow rate through branch b

Objective function

The objective function (6.5) minimizes exergy consumption. The hot and cold utilities, q^{HU} and q^{CU}, are multiplied with the Carnot factor to provide the exergy content. The fraction f_{sb} associated with branch b, multiplied with the heat capacity flow rate and the temperature difference over the pressure changing unit, constitute the work consumed or produced in each unit. In total, these terms represent the exergy consumption of the process.

$$\min \; Ex = \Big(1 - \frac{T_0}{T^{HU}}\Big)q^{HU} + \Big(\frac{T_0}{T^{CU}} - 1\Big)q^{CU} + \sum_{s\in S^P}\sum_{b\in B} f_{sb}(mc_p)_s\Big(T^{in}_{sb} - T^{out}_{sb}\Big) \tag{6.5}$$

Constraints

The objective function is subject to the following constraints:

$$r_1 - q^{HU} - \sum_{s\in S^P}\sum_{b\in B}\sum_{z\in Z} Q^P_{s1bz} f_{sb} = \sum_{s\in S\setminus\{S^P\}} Q^{CP}_{s1} \tag{6.6}$$

$$r_k - r_{k-1} - \sum_{s\in S^P}\sum_{b\in B}\sum_{z\in Z} Q^P_{skbz} f_{sb} = \sum_{s\in S\setminus\{S^P\}} Q^{CP}_{sk}, \qquad k \in K\setminus\{1, |K|\} \tag{6.7}$$

$$-r_{|K|-1} + q^{CU} - \sum_{s\in S^P}\sum_{b\in B}\sum_{z\in Z} Q^P_{s|K|bz} f_{sb} = \sum_{s\in S\setminus\{S^P\}} Q^{CP}_{s,|K|} \tag{6.8}$$

Constraint 6.6-6.8 are the heat balance constraints. Q^{CP}_{sk} and Q^P_{skbz} are the sum of the heat available and the heat required by the constant pressure streams and the pressure changing streams, respectively. Constraint 6.6 and Constraint 6.8 are the heat balances for the first and the last interval. The required heat entering the first interval is equivalent to the hot utility consumption, whilst the excess heat leaving the last interval corresponds to the cold utility required.

$$\sum_{b\in B_s} f_{sb} = 1, \qquad s \in S^P \tag{6.9}$$

$$f_{sb} \geq 0, \qquad s \in S^P, b \in B \tag{6.10}$$

$$r_k \geq 0, k \in K \tag{6.11}$$

Constraint 6.9 forces the sum of the fractions for each stream s to be 1, maintaining the mass balance. Constraint 6.10 ensures that the streams are flowing in the right direction and Constraint 6.11 maintains thermodynamically feasibility making sure that the heat is only transferred from a higher temperature to a lower temperature.

The objective value Ex is the minimum exergy consumption, which can be obtained with linear programming when the decision vector $\vec{t}_{sb}^{in}$ is fixed. The LP problem is solved repeatedly for varying decision vectors $\vec{t}_{sb}^{in}$ provided by the outer loop. The results from the inner loop provides a quality measure on a population of decision vectors, referred to as $fitness(P(t))$ in Figure 6.2.

6.3.2 Outer loop optimization

The outer loop searches for promising decision vectors $\vec{t}_{sb}^{in}$ that will optimize the objective function value Ex, with heuristic algorithms. Various heuristic algorithms can be applied to the outer loop. The problem under consideration in this thesis has been adapted to GAs as the method of search technique. Two algorithms have been developed and presented in the following section.

6.4 Heuristic optimization algorithms for the outer loop

Two GAs are implemented for the outer loop. First, a basic GA was implement. One of the main fallacies in using GAs is being trapped in a local optimum. In order to avoid getting trapped, a GA with crowding was implemented, maintaining the diversity within the population. Both algorithms are presented in this section.

6.4.1 Basic genetic algorithm

The basic GA consists of four main operations: selection, crossover, mutation and replacement. A graphical representation of the structure of the algorithm is illustrated in Figure 6.3. The only input information to the algorithm is the stream data and the maximum number of pressure changing units required by the process. The number of pressure changing streams and the number of maximum allowable stream splits must therefore be predetermined. The stream data contain supply and target temperatures, flow rates, heat capacities and pressure ratios. Prior to the evolution loop, an initial population is randomly generated, reducing the likelihood of biasing the results. The evolution loop generates new populations until the stopping criteria is met. Following is an overview of the implemented steps in the evolution loop.

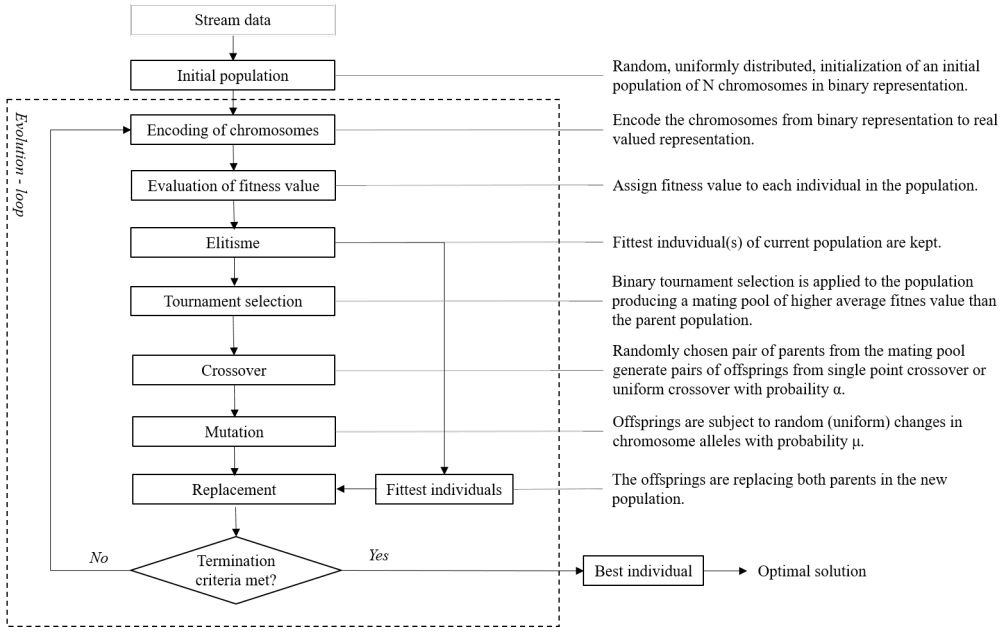

Figure 6.3: Flow chart representation of the basic GA.

Encoding

In both algorithms, binary representation is applied for encoding of chromosomes. The number of pressure changing units determines the number of genes in each chromosome, where each gene holds information about a unit inlet temperature. The chromosome representation is illustrated in Figure 6.4. Through the decoding function in Algorithm 2, the genes are decoded into numerical values for the inlet temperatures. Population $P(t)$ of decoded variables is the input to the evaluation of fitness values, which is carried out in the inner loop.

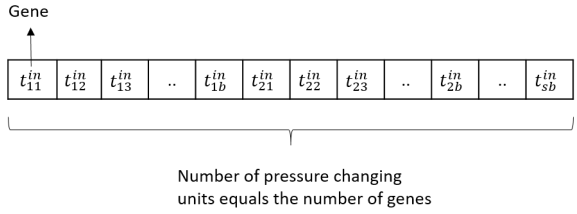

Figure 6.4: Chromosome representation.

Algorithm 2 Decoding function

```
1: function GENOTYPEMAP(population genotype)
2:     calculate resolution of variable range for all genes using Equation 5.1
3:     for all genes in each chromosome do
4:         num ← numerical value of gene
5:         calculate real value of gene using Equation 5.3
6:         assign decoded gene to the phenotype population
7:     end for
8:     return phenotype population
9: end function
```

Selection

The basic GA combines a selection mechanism with elitist selection. Elitist selection is implemented to prevent fit individuals from being lost through selection or destroyed by crossover and mutation. The elitist selection procedure is combined with tournament selection. Tournament selection was chosen over proportional selection in order to promote diversity in the population. Tournament selection is also implemented very efficiently as no sorting algorithm of the population is required, reducing the overall computational speed. The implemented algorithms for elitist selection and tournament selection are shown in Algorithm 3 and Algorithm 4.

Algorithm 3 Elitist selection

```
1: function ELITISTSEL(fitness(P(t)))
2:     n ← number of individuals to directly survive the next generation
3:     sort population according to fitness value in descending order
4:     assign the n first individuals to an elite population
5:     return elite population
6: end function
```

Algorithm 4 Tournament selection

```
1: function TOURNAMENTSEL(fitness(P(t), t))
2:     poolSize ← size of mating pool
3:     for i=1 to poolSize do
4:         matingPool[i] = best individual among t randomly selected from P(t)
5:     end for
6:     return matingPool
7: end function
```

Any selection mechanism intend to favour individuals of high quality, but also maintain diversity within the population. During the selection process poor individuals are lost and thereby also the gene material contained in the replaced individuals. The number of individuals lost in the selection process contributes to a loss of diversity in the population. Two factors may cause loss of diversity through tournament selection; some individuals may not get sampled to participate in a tournament while others might not be selected for the mating pool because they lost a tournament. The relation between loss of diversity and the tournament size was derived by Blickle and Thiele [7]. The same authors derived a mathematical relation between tournament size and selection intensity and selection variance. Selection intensity measures the change in average population fitness, whilst selection variance is the expected variance of the population fitness distribution. The relation between tournament size and the factors described above is plotted in Figure 6.5.

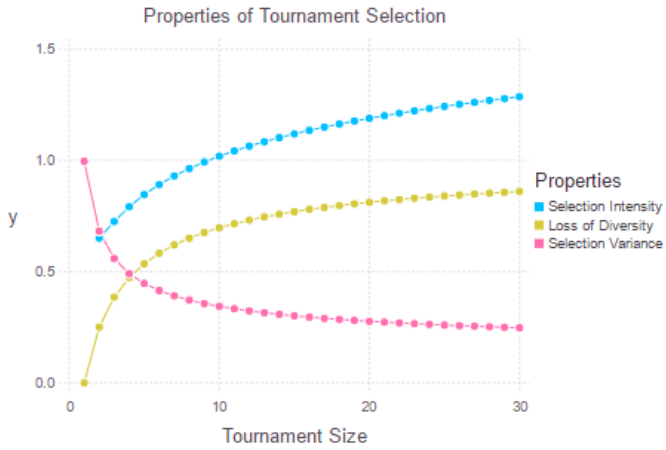

Figure 6.5: Influence of tournament size.

A larger tournament size t implies that a smaller number of individuals contributes to population diversity, making the search increasingly greedy in nature [6]. Figure 6.5 shows that for higher values of t the average population fitness increases, however, the selection variance decreases and the loss of diversity increases. A trade-off between increasing the average fitness and maintaining population diversity is required. A tournament size of 2-3 is regarded as a suitable size for most problems [26] [14]. The search tend to converge too slow for smaller sizes of t and too fast for larger sizes of t [14]. In combination with both parent replacement, the basic GA has implemented binary tournament selection, $t = 2$, in order to promote population diversity.

Crossover

Crossover is the process by which the genetic material in two or more parent individuals is combined to obtain one or more offsprings. Different crossover methods are discussed in Section 5.6. Traditionally, GAs have relied on single point crossover. However, Spears and De jong [65] analyzed the effects of crossover on GA performance. They showed that, for a large search space, a GA using uniform crossover often outperforms a GA using single-point crossover. In order to increase the exploration effect and the performance of the search, uniform crossover was implemented. The implementation of uniform crossover is illustrated in Algorithm 5.

Algorithm 5 Uniform crossover

```
1:  function CROSSUNIFORM(matingPool, P_c)
2:      popSize ← population size
3:      chromSize ← length of chromosomes
4:      i ← 1
5:      while i ≤ popSize -1 do
6:          parentA ← random individual chosen from the mating pool
7:          if a random number is ≤ P_c then
8:              parentB ← random individual chosen from the mating pool
9:              mask ← random generated bit string of length chromSize
10:             for m=1 to chromSize do
11:                 if mask[m] == 1 then
12:                     childA[m] = parentA[m]
13:                     childB[m] = parentB[m]
14:                 else if mask[m]==0 then
15:                     childA[m] = parentB[m]
16:                     childB[m] = parentA[m]
17:                 end if
18:             end for
19:             childA and childB are assigned to the population of offsprings
20:             i ← i+2
21:         else
22:             i ← i+1
23:         end if
24:     end while
25:     return population of offsprings
26: end function
```

P_c is the crossover rate. The crossover rate controls the capability of GAs in exploiting a located hill to reach the local optimum. The higher the crossover rate, the quicker the

exploitation proceeds. However, a crossover probability that is too large would disrupt individuals faster than they could be exploited [43], in which favourable gene material get lost. Typical values of the crossover probability are in the range 0.50-1.00 [43].

Mutation

Mutation introduces new genetic structures into the population by random modifications to the gene values. Flipping mutation was implemented in an effort to reduce the high convergence speed and increase the exploration rate. The mutation algorithm is illustrated in Algorithm 6.

Algorithm 6 Flipping mutation

```
1:  function MUTFLIPSWAP(population of offsprings, P_m)
2:      popSize ← population size
3:      chromSize ← length of each chromosome
4:      for i=1 to popSize do
5:          if a random number is ≤ P_m then
6:              mutChrom ← random generated bit string of length chromSize
7:              for j = 1 to chromSize do
8:                  if mutChrom[j] == 1 then
9:                      for offspring i, alter the bit value at position j
10:                 end if
11:             end for
12:         else
13:             no mutation
14:         end if
15:     end for
16:     return mutated population of offsprings
17: end function
```

The mutation probability P_m is usually kept low. From selection and crossover operations, new individuals are produced from parents of high quality. If the mutation rate is high and heavily disrupt the gene material, the offsprings will not inherent the advantageous features of its parents. The search will therefore seek towards a higher exploration rate of the search space in the expense of exploitation of favourable areas. A low mutation rate, however, may result in premature convergence. Typical values of the mutation rate are in the range 0.001-0.05.

6.4.2 Genetic algorithm with crowding

In order to prevent premature convergence to local optima, a GA with crowding was implemented. Crowding is a niching technique, in which the algorithm aim to (1) converge to multiple, highly fit and significantly different solutions and (2) to slow down convergence in cases where only one solutions is required [54]. Generalized crowing was presented in Section 5.10 and is the crowding technique implemented in the algorithm presented in this section. A graphical representation of the structure of the algorithm is illustrated in Figure 6.6.

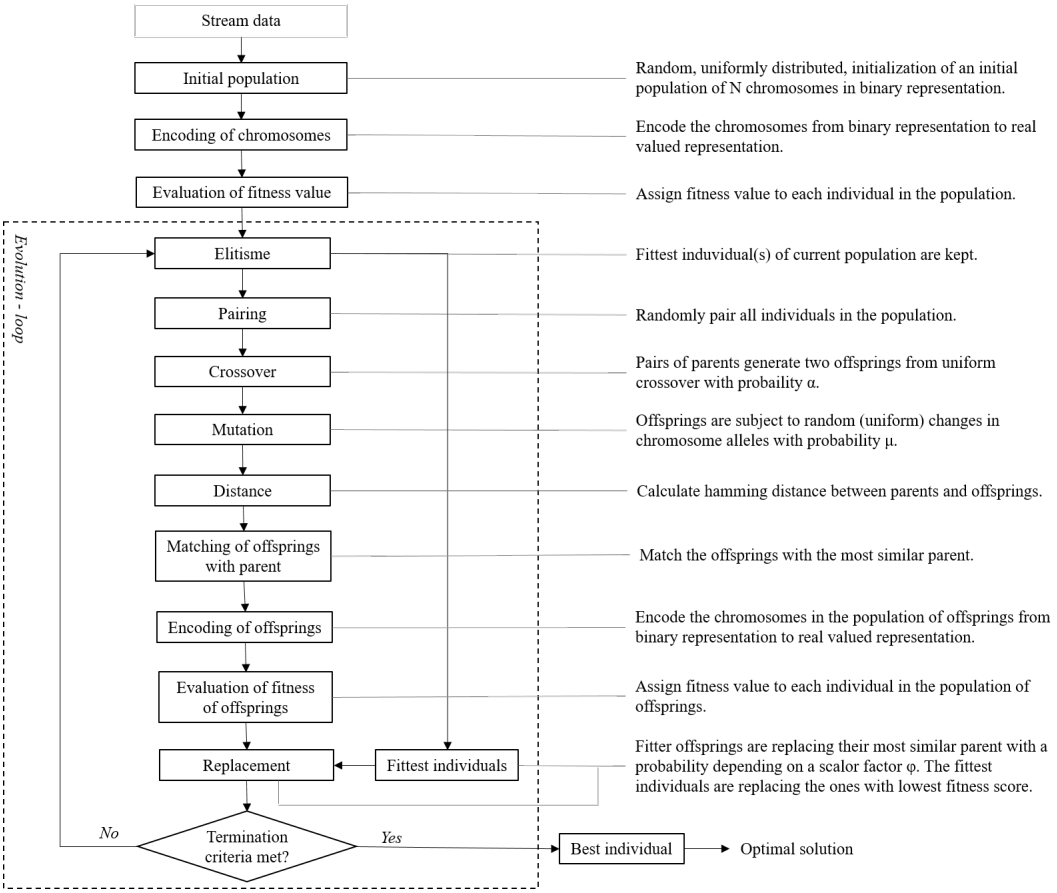

Figure 6.6: Flowchart representation of the GA with crowding

Equivalent to the previous algorithm, the GA with crowding applies binary representation for the encoding of chromosomes. Uniform crossover and flipping mutation operations are

utilized to generate new individuals. The main differences occur in the selection and the replacement processes. Parent selection is usually not applied under crowding and every individual in the population becomes a parent [22]. In order to prevent fit individuals from being destroyed by crossover or mutation, elitist selection is implemented.

A pseuducode of the GA with crowding is illustrated in Algorithm 7. The parents are randomly paired and each pair is subject to crossover. The offsprings generated by crossover are subject to mutation. Then follows the replacement process. The GA with crowding developed in this thesis has implemented generalized crowding. With generalized crowding one have the ability to adjust the scaling factor to influence the replacements. Equation 5.9 is applied to the replacement phase, where P_c denotes the probability that an offspring c replaces parent p in the population. The scaling factor ϕ can be adjusted to the problem under consideration.

Algorithm 7 Generalized crowding algorithm

popSize ← population size
genPop ← random generated initial population
phenPop ← **genotypeMap**(genotype population)
fitness(P(t)) received from the inner loop
while seach criteria not met **do**
    elitePop ← **elitistSel**(fitness(P(t)))
    randomly pair all individuals
    for i = 1 to number of pairs **do**
        parentPop[i] = (p_1, p_2)
    end for
    for i = 1 to length of parentPop **do**
        (c_1, c_2) ← **crossUniform**(parentPop[i], P_c)
        (c_1, c_2) ← **mutFlipSwap**((c_1, c_2), P_m)
        childPop[i] ← (c_1, c_2)
    end for
    Evaluate Hamming distance between each offspring and both of its parents
    Let $p_i c_i$ be the Hamming distance between parent p_i and offspring c_i
    for all pairs of parents and the corresponding two offsprings **do**
        $d_1 = p_1 c_1 + p_2 c_2$
        $d_2 = p_1 c_2 + p_2 c_1$
        if $d_1 \leq d_2$ **then**
            p_1 competes with c_1
            p_2 competes with c_2
        else
            p_1 competes with c_2
            p_2 competes with c_1
        end if
    end for
    phenChild ← **genotypeMap**(genotype population of offsprings)
    fitness(C(t)) received from the inner loop
    for all competing parents p and offspring c **do**
        if fitness(c) $\geq$ fitness(p) **then**
            replace parent p with offspring c with probability $P_r = \frac{f(c)}{f(c)+\phi \times f(p)}$
        else if fitness(c) == fitness(p) **then**
            replace parent p with offspring c with probability $P_r = 0.5$
        else if fitness(c) $<$ fitness(p) **then**
            replace parent p with offspring c with probability $P_r = \frac{\phi \times f(c)}{\phi \times f(c)+f(p)}$
        end if
    end for
end while

Chapter 7

Implementation

Different parameter values in GAs might lead to very different results. For an optimal configuration, the algorithm may converge to the best solution in a short time, whilst inferior settings may cause the algorithm to become trapped in a local optimum. The GA parameters are mutually dependent, thus, optimal values are difficult to obtain. An evaluation of adequate parameter values has been carried out in order to establish an initial parameter configuration, followed by a trial and error fine-tuning of these parameters. Section 7.1 introduces the software used in the implementation of the algorithms. The following section 7.2 states the hardware used for running the test cases. Section 7.3 provides a discussion of adequate parameter values and the choice of values for the implementation of the basic GA and the GA with crowding.

7.1 Software

The search algorithms and the optimization model are implemented in the programming language Julia (latest version 0.5.0). Julia is a high-level, high-performance dynamic programming language, primarily developed for scientific and technical computing [37]. Julia was designed to combine the simplicity of Python with a more sophisticated compiler and other improvements, that make the platform easier to use and better suited for numerical computation. Also, a well written code can achieve performance that is comparable to C.

IJulia is a Julia-language backend combined with the Jupyter interactive environment (also used by IPython) [24]. This combination allows you to interact with the Julia language using Jupyter [24]. Jupyter is the graphical notebook in IPython, which combines code, formatted text, math and multimedia in a single document [38]. There are other options of

environments for editing and running Julia code, such as Juno and Atom; however, IJulia and Jupyter have been utilized in this work.

Julia has a built-in package manager for installing add-on functionality [37]. JuMP is a package for modeling optimization problems [25]. The similarities between the syntax and the mathematical structure of the problem greatly simplifies the implementation. Through JuMP, solvers, such as CPLEX, are efficiently connected with Julia. CPLEX is a commercial solver for LP problems and MILP problems. The LP model in the inner loop was modelled with JuMP and solved with the CPLEX solver. The existing (not particular functional) abstract high-level GA packages in Julia, such as GeneticAlgorithms.jl, have not been applied in any of the implementations. Also, plotting in Julia is available through packages. For plotting and visualization, Gadfly [21] was applied in this work.

Julia is a young language, first released in 2012. The youth of Julia means that it is not as mature or developed as more established languages. The official Julia documentation is relatively good, however aimed primarily at developers and experienced programmers. The major disadvantage experienced throughout this work is the absence of debugging functionalities. Also, windows installation of IJulia was especially time consuming and unintuitive. The book by Kwon [9] is strongly recommended to those who are new to programming and aim to use Julia in operations research.

7.2 Hardware

The computations are carried out on a Dell laptop with Intel(R) Core(TM) i7 Processor (2.70GHz) and 8 GB RAM, running Windows 10.

7.3 Parameter settings

GAs involve complex interactions among multiple parameters, which are highly dependent on the function being optimized. Researchers have been trying to understand the mechanisms behind the genetic parameter interactions by using various techniques, such as empirical studies and Markov chain analysis. Although the internet has made it easier to communicate experiences for a wide range of problems, the choice of parameter values still relies mainly on trial and error. The GA parameter values that are implemented for the test cases studied in the following chapter, is listed in Table 7.1.

The inlet temperatures to the pressure changing units are restricted to lie within the range of ambient and hot utility temperature. Using GAs involves a trade-off between accuracy and computational time. A bit string can only represent a finite number of values, thus, the number of bits restricts the accuracy of the solution. For higher accuracy, more bits are required. However, large bit strings increase the computational demand. Thus, there

Parameter values		
Parameter	**Basic GA**	**GA crowding**
lower bound on design variables	T_0	T_0
upper bound on design variables	T^{HU}	T^{HU}
number of bits	11	11
population size	20×paraNum	20×paraNum
rate of elitism, ε	0.01	0.01
crossover probability, P_c	0.75	0.75
mutation probability, P_m	0.10	0.10
tournament size, t	2	-
scaling factor, ϕ	-	0.15
generations	10,000	10,000

Table 7.1: Implemented parameter values for the basic GA and the GA with crowding.

must be a trade-off between accuracy and computational time. 11 bits was considered to be sufficient in maintaining the accuracy of the solutions, thus enabling the algorithm to perform 10,000 iterations within reasonable time.

The population size affects both the ultimate performance and the efficiency of the algorithm. GAs generally do poorly with very small populations because the population provides an insufficient sample size [28]. With regards to schema theory, a large population is more likely to contain representatives from a large number of various schema. Hence, a large population discourages premature convergence to local optima [28]. On the other hand, larger populations require more evaluations in the inner loop for each generation. Increased computational time may result in an unacceptably slow convergence rate. Grefenstette [28] suggested a population size within the range of 10 to 160 individuals. However, the population size is generally advised to be as large as possible while maintaining a sufficient number of generations. In order to maintain diversity within the population, the population size is increasing proportional to the number of design variables (number of genes). The population size was chosen to be 20 times the number of design variables, in which was considered sufficient to preserve the population diversity while going through 10,000 generations within reasonable time.

The number of generations is related to improvements in the fitness values. The increase in population fitness improves in early generations for then to decrease and asymptotically approach an optimum. Since the global optimum cannot be verified with exact solution methods for the presented test cases, the number of generations are kept high. In order to stay within reasonable time while providing sufficient accuracy in the number of bits, both algorithms was restricted to 10,000 iterations.

The crossover probability influences the exploitation rate; for higher crossover probabilities, the exploitation rate increases as more new individuals are created from high quality parents. In an effort to exploit the accumulated information, whilst maintaining the diversity within the population, a crossover probability of 0.75 has been applied.

An insufficient mutation rate will not provide the required coverage of the search space. However, if the mutation rate is too high, the gene material in good candidate solutions will be disrupted, generating unacceptable solutions. In combination with elitist selection, the mutation rate is kept at 0.1 throughout the test cases. This is higher than what is regarded as typical values for the mutation rate. However, in comparison with other GAs developed for the heat integration problem, 0.1 is relatively low. The population size is kept large, hence a rate of elitism of 0.01 is considered sufficient. Thus, 1% of the fittest individuals will directly survive the next generation; for the test cases, this is equivalent to 1-4 individuals.

For the GA with crowding, the value of the scaling factor has to be adjusted to the problems under consideration. For greater numbers of ϕ, a larger proportion of the population occupies a less fit area of the search space. For low values of ϕ, areas of the search space of higher quality solutions is represented by a higher percentage of the population. A high value of ϕ early in the search process contributes to a higher exploration rate so that a larger area of the search space are being sampled. As the search converges towards areas of high quality solutions, emphasis on higher exploration rate becomes increasingly important in identifying the local optimum. Thus, lower values of ϕ are required. A constant value of 0.15 is applied for all test cases, in order to provide sufficient exploration of the search space early in the process and increase the competence in finding the local optimum as the algorithm converge towards areas of high quality solutions.

Chapter 8

Computational study

This chapter is a computational study of the two-level optimization model for simultaneous heat and work integration involving multiple pressure changing streams. The model aim to assist in the establishment of correct placement of pressure changing units when integrated in the HEN. In cooperation with Fu Chao from SINTEF Energy Research, four test cases were designed. The optimization model identifies minimum exergy consumption for optimal heat and work integration, which may serve as a guideline in designing the HEN.

Common practice in the industry is to let pressure changing streams be compressed at ambient temperature and expanded at hot utility temperature, in order to minimize compression work and maximize expansion work. The purpose of this study is to examine how pressure changing units can be integrated in the HEN to reduce energy consumption. The objective is to identify the exergy savings by allowing for stream splitting and pressure change at different temperatures. The results from the two-level optimization model are presented in comparison with common practice in the industry. Since the focus of this study is solely on energy efficiency, the economical consequences of stream splitting are not explicitly taken into consideration in the model. However, important economic implications are briefly discussed.

Maurstad [52] solved eight test cases with exact solution methods for heat and work integration involving one compressing and one expanding stream. These results are compared with the two-level optimization model, attached in Appendix A, in which the two-level optimization model was able to locate optimal or near-optimal solutions. The test cases presented in this chapter are derived from Case 7, increasing the number of pressure changing streams. Case 9 and Case 10 are conferred to Section 8.1 and Section 8.3, involving one compressing and two expanding streams. Case 11 concerns two compressing and one expanding stream. Finally, Case 12 studies heat and work integration of two compressing and two expanding streams.

8.1 Case 9

Case 9 is an extension of Case 7 adding one pressure changing stream. The stream data is listed in Table 8.1. The added stream is hot stream H4, which are to be expanded from 3 to 1 bar. The contribution from this case is the presence of two expanding streams. Correct integration of multiple expanding streams have not yet been studied. A solution to this scenario is presented in this section. A stream split arrangement is identified by the two-level optimization model and presented in comparison with a base case. The base case is the case of common practice in the industry, in which compression starts at ambient temperature and expansion starts at hot utility temperature.

Stream	T^S [°C]	T^T [°C]	mc_p [kW/°C]	P^S [bar]	P^T [bar]	ΔT_{min}
H1	400	35	2	2	1	20
H2	320	160	4	-	-	
H3	110	35	3	-	-	
H4	400	35	1.5	3	1	
C1	15	380	3	1	2	
C2	190	250	10	-	-	

Table 8.1: Stream data for Case 9.

The process pinch point is illustrated on the GCC in Figure 8.1, for heat integration of the process streams without pressure change. The pinch point is located at $210/190°C$ and two potential pinch points are identified; one above pinch and one below pinch at temperatures of $320/300°C$ and $110/90°C$, respectively.

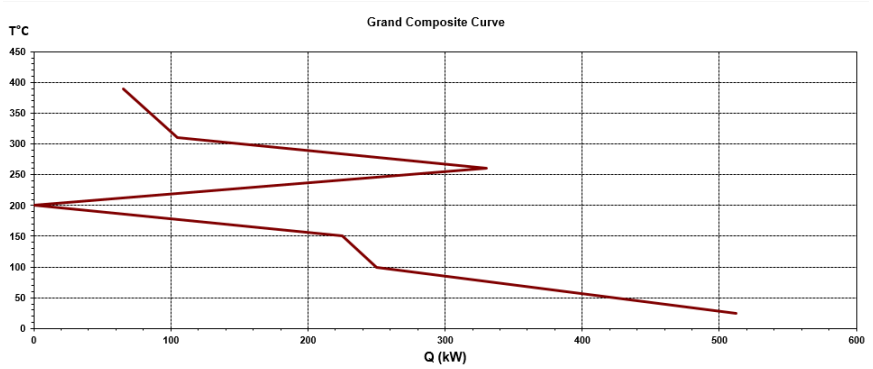

Figure 8.1: GCC for Case 9 without pressure change.

Results

Case 9 is solved with the basic GA and the GA with crowding in the outer loop. The results are presented in Table 8.2 in comparison with the base case. Close to the same solution is observed for the two algorithms. By allowing for splitting of the streams and pressure change to occur at multiple temperatures, the exergy savings are substantial. Integration of the pressure changing units eliminates the need of external heating at the expense of more compressor work and less expansion work. The negative exergy values imply that the process is producing exergy.

Case 9		**GA.v01**	**GA.v02**	**Base case**
Exergy [kW]		-142.93	-142.91	6.52
Hot utility [kW]		0.00	0.00	578.90
Work [kW]		-385.90	-383.53	-513.90
		242.98	243.04	189.33
T_s^{in}/T_s^{out} [°C]	H1	400.00 / 279.06	400.00 / 279.06	400.00 / 279.06
		210.06 / 123.25	210.04 / 123.23	-
	H4	209.02 / 79.78	210.04 / 79.87	400.00 / 218.65
	C1	189.80 / 291.20	189.35 / 290.65	-
		90.02 / 169.56	89.86 / 169.36	-
		15.00 / 78.12	15.00 / 78.12	15.00 / 78.11
$mc_{p,s}$ [kW/°C]	H1	0.50	0.50	2.00
		1.50	1.50	-
	H4	1.50	1.50	1.50
	C1	1.25	1.25	-
		0.37	0.37	-
		1.38	1.38	3.00

Table 8.2: Results from Case 9. Results from the basic GA are listed under GA.v01 and results from the GA with crowding are listed under GA.v02.

From Table 8.2, the stream split arrangements for the pressure changing streams are identified. For both solutions, H1 has positive flow rates in two branches; the first involves expansion at hot pinch temperature and the second involves expansion at hot utility temperature. H4 is subject to expansion at hot pinch temperature, whilst C1 is split in three branches; the first branch involves compression at cold pinch temperature, the second involves new pinch compression and the third branch involves ambient compression. An illustration of the proposed stream split arrangement is presented in Figure 8.2. In the base case, compression occurs at ambient temperature and expansion occurs at hot utility temperature, involving no stream splits, illustrated in Figure 8.3.

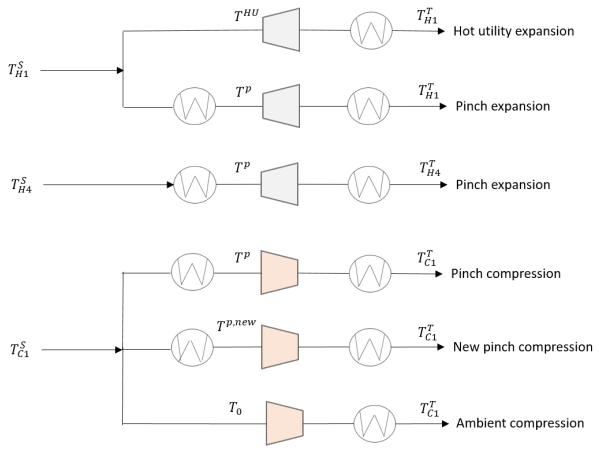

Figure 8.2: Stream split arrangement for Case 9.

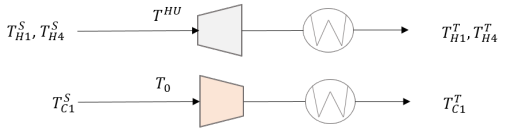

Figure 8.3: Common practice in the industry.

The GCC for the stream split arrangement in Figure 8.2 is illustrated in Figure 8.4. The two new pinch points can be observed at $320/300°C$ and $110/90°C$, respectively. The cold utility requirement has no influence on the exergy consumption when the cold utility temperature equals ambient temperature. However, Figure 8.2 shows that also the cold utility requirement is reduced.

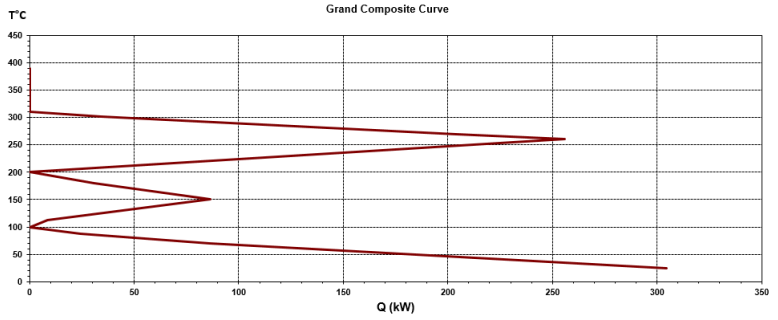

Figure 8.4: GCC for Case 9 with pressure change.

The results are in accordance with the theorems presented by Fu and Gundersen [18], discussed in Section 4.3. For optimal integration of compressors and expanders in above-ambient HENs, their insight suggests that the inlet temperatures to the compressors should be at pinch temperature, at new pinch temperature or at ambient temperature; and the inlet temperatures to expanders should be at pinch temperature, at new pinch temperature or at hot utility temperature.

Stream split analysis

The problem objective is to gain insight into how energy efficiency can be improved. However, additional branches with positive flow rates imply additional pressure changing units. Compressors and turbines are some of the most expensive equipment in the process industry, surpassing the value of heat exchange equipment [58]. Also, additional branches increase the size of the search space and thus the computational power required to find a good solution. In order to gain insight into energy efficient processes without unrealistic cost conditions, the maximum number of branches was restricted to three. However, it is of great interest to study the influence of the number of available branches on the objective value.

Case 9 has been solved for different maximum numbers of stream splits with both algorithms. The results are listed in Table 8.3. The objective values Ex and the computational times are recorded in the two rightmost columns. The implementation experienced issues in displaying the results, hence the incomplete list of computational times. The highest number of branches with positive flow rates among the pressure changing streams, is listed under b_{max}. Recorded below u_{tot} is the total number of compressors and expanders required by the solution.

	GA.v01				GA.v02			
|B|	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**
1	1	3	-125.88	487.17	1	3	-124.76	2258.02
2	2	6	-141.62	1551.31	2	6	-142.04	-
3	3	6	-142.93	4043.89	3	6	-142.91	17811.48
4	3	6	-142.91	7432.52	3	6	-142.91	31505.26

Table 8.3: Stream split analysis for Case 9.

In the presented solution above, a maximum of three branches was available. C1 had positive flow rates in all branches. Increasing the number of maximum branches to four did not improve the objective value. Also, none of the pressure changing streams had positive flow rates in more than three branches, suggesting that additional stream splits will not improve the energy efficiency any further. Restricting the number of branches to two yields a 12.50% and 13.85% increase in exergy production compared to no stream

splitting, for the basic GA and the GA with crowding, respectively. However, the increase in exergy production requires twice the number of pressure changing units. In the case of no stream splitting, the objective value is significantly improved in comparison to the base case. The system produces more exergy than is consumed with no additional pressure changing units; however, additional heat exchangers will be required in order to reach the inlet temperatures.

The objective of this work is to identify the energy target for maximum heat and work recovery. Integrating compressors and expanders with the HEN leads to substantial savings in exergy consumption. Stream splitting enhance the energy efficiency even further. The stream split analysis above suggests that, for this case, no more than three branches are required to obtain maximum heat and work recovery. The inlet temperatures to the pressure changing units are in accordance with the theorems presented by Fu and Gundersen [18] for integration of one compressing and one expanding stream. Increasing energy efficiency require investments in additional process equipment, in which the economic implications must be considered in order to evaluate the practicality of the solution.

8.2 Case 10

Case 10 is similar to Case 9, the only difference being a decrease in flow rate for stream C2. The stream data is listed in Table 8.4. Case 10 is considering a threshold process, in which no pinch point is dividing the process into two parts; one region above pinch having a deficit of heat and one region below pinch with an excess of heat. Characteristic for threshold problems is that these problems either require hot utility or cold utility, and not both. Threshold problems are, in fact, quite common in practice. In this case, the pinch point can be considered located at the hot utility temperature, hence only cold utility is required. The threshold is illustrated in Figure 8.5. The figure depicts the GCC for the process streams in Table 8.4 in the case of all streams having constant pressure. The flow rate of C2 is reduced, hence the considerable increase in demand for cold utility. The pinch point location is considered at the hot end, at $400/380°C$. Furthermore, three potential pinch points can be identified at $320/300°C$, $210/190°C$ and $110/90°C$.

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	35	2	2	1	20
H2	320	160	4	-	-	
H3	110	35	3	-	-	
H4	400	35	1.5	3	1	
C1	15	380	3	1	2	
C2	190	250	4	-	-	

Table 8.4: Stream data for Case 10.

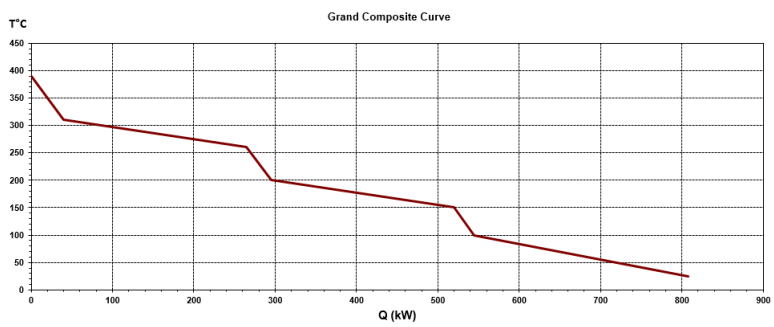

Figure 8.5: GCC for Case 10 without pressure change.

Results

Case 10 is solved with the basic GA and the GA with crowding in the outer loop. The results are listed in Table 8.5 in comparison with a base case. The base case is similar to that of Case 9, in which compression takes place at ambient temperature and expansion takes place at hot utility temperature. In the base case, maximum work are produced in the turbines, increasing the hot utility required. The two-level optimization model presents two different solutions, in which both solutions eliminate the need of hot utility in the expense of a loss in work generated by the turbines. In terms of energy efficiency, the two-level optimization model clearly presents the most beneficial solutions, increasing the exergy production with 35.69% and 35.70% for the basic GA and the GA with crowding, respectively.

The GAs identified two different solutions. The variation in the objective values is very small, however, the unit inlet temperatures are somewhat different. H1 has positive flow rates in all three branches in both solutions. Two branches involve expansion at intermediate temperatures below hot utility and above the new pinch at $320/300°C$. The third branch involves new pinch expansion at $210/190°C$. Expansion of H4 occurs at the highest new pinch point in both solutions, whilst C1 is compressed at ambient temperature. A graphical illustration of the stream split arrangements is presented in Figure 8.6. The GCCs for the two solutions are presented in Figure 8.7 and Figure 8.8. The shape of the curves illustrate the small difference between the two solutions in the upper pocket. The effect of the difference on the objective value is very small and both solutions require the same number of heat exchangers and pressure changing units.

Fu and Gundersen [17] recognized that there are cases where expansion at some intermediate temperature ($T^p < T^{in} < T^{HU}$) can achieve the same minimum exergy consumption as if expansion would occur at hot utility temperature and pinch temperature. Running the optimization model, in which expansion of H1 is restricted to occur at either hot utility temperature or pinch temperature, generated an objective value of -241.04. One

Case 10		GA.v01	GA.v02	Base case
Exergy [kW]		-253.93	-253.94	-187.14
Hot utility [kW]		0.00	0.00	240.00
Work [kW]		-443.26	-443.27	-513.75
		189.33	189.33	189.33
T_s^{in}/T_s^{out} [°C]	H1	390.97 / 271.65	368.46 / 253.19	400.00 / 279.06
		330.22 / 221.82	329.83 / 221.49	-
		210.60 / 123.69	210.02 / 123.21	-
	H4	320.07 / 160.25	320.7 / 160.25	400.00 / 218.65
	C1	15.00 / 78.12	15.00 / 78.12	15.00 / 78.12
$mc_{p,s}$ [kW/°C]	H1	0.47	0.74	2.00
		0.68	0.41	-
		0.85	0.85	-
	H4	1.50	1.50	1.50
	C1	3.00	3.00	3.00

Table 8.5: Results from Case 10. Results from the basic GA are listed under GA.v01 and results from the GA with crowding are listed under GA.v02.

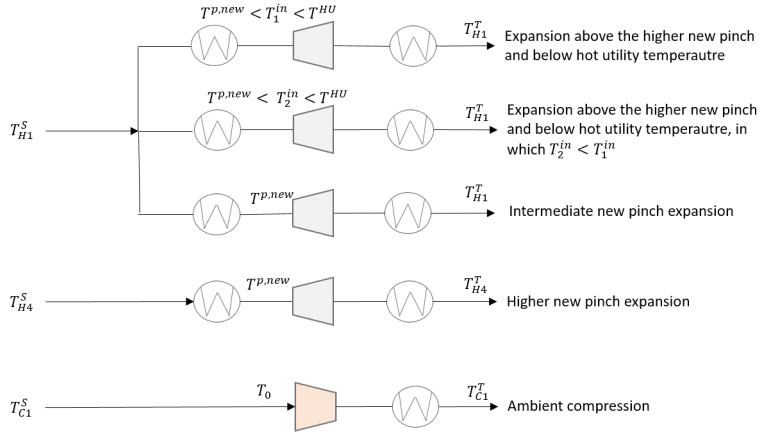

Figure 8.6: Stream split arrangements for Case 10.

quarter of the flow rate enters the expander at hot utility temperature, whilst three quarters flow through the expander at pinch temperature. These results suggests that expansion at intermediate temperatures may be more energy efficient than expansion at hot utility and pinch temperature.

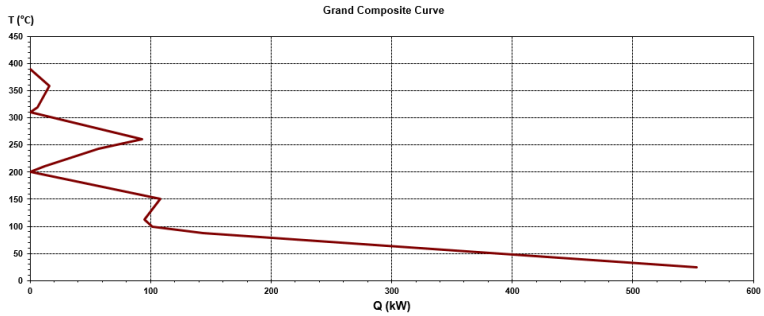

Figure 8.7: GCC for Case 10 solved with GA.v01.

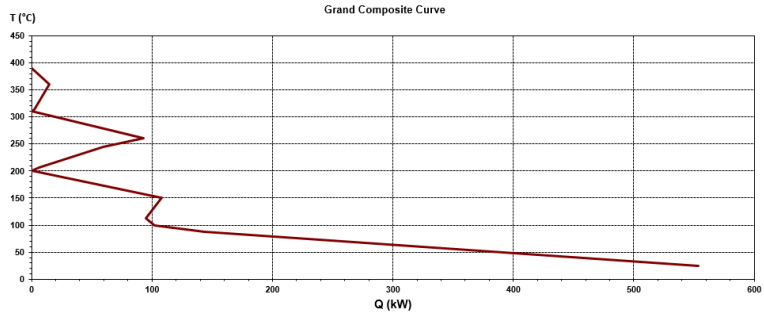

Figure 8.8: GCC for Case 10 solved with GA.v02.

Stream split analysis

Similar to Case 9, Case 10 is solved for different numbers of maximum stream splits. The results are listed in Table 8.6. In the solutions presented above, H1 had positive flow rates in all three branches. Increasing the number of maximum branches to four did not improve the objective value. Also, none of the pressure changing streams had positive flow rates in more than three branches, suggesting that additional stream splits will not improve the energy efficiency any further. The solutions presented above require five pressure changing units. Restricting the number of branches to one, a 1.24% and 1.56% loss in exergy production will save the investment costs of two pressure changing units, for the basic GA and the GA with crowding respectively. In comparison with the base case, the solutions from the basic GA and the GA with crowding for no stream splits, will increase the exergy production with 34.01% and 33.57%, respectively.

In terms of energy efficiency, at least three branches are required to obtain maximum heat and work recovery. However, the improvements in the objective value is not significant in adding more stream splits. Thus, from an economically point of view, a solution requiring less pressure changing units may be preferable.

	GA.v01				GA.v02			
|B|	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**
1	1	3	-250.78	498.90	1	3	-249.97	2285.06
2	2	5	-253.68	1519.58	2	5	-253.88	8601.86
3	3	5	-253.93	4421.31	3	5	-253.94	-
4	3	5	-253.92	6229.81	3	5	-253.93	38890.26

Table 8.6: Stream split analysis for Case 10.

8.3 Case 11

Case 11 is an extension of Case 7 with one added pressure changing stream. The stream data is listed in Table 8.7. The added stream is cold stream C3, which are to be compressed from 1 to 5 bar. The contribution from this case is the presence of two compressing streams. Similar to multiple expanding streams, correct integration of multiple compressing streams have not yet been studied. Two different solutions to this scenario are presented in this section. The results are presented in comparison with a base case of common practice in the industry, in which compression starts at ambient temperature and expansion starts at hot utility temperature.

The process pinch point is illustrated on the GCC in Figure 8.9 for heat integration of the process streams without pressure change. The pinch point is located at $110/90°C$ and two potential pinch points are identified at $210/190°C$ and $320/300°C$.

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	35	2	2	1	20
H2	320	160	4	-	-	
H3	110	35	3	-	-	
C1	15	380	3	1	2	
C2	190	250	10	-	-	
C3	90	250	2	1	5	

Table 8.7: Stream data for Case 11.

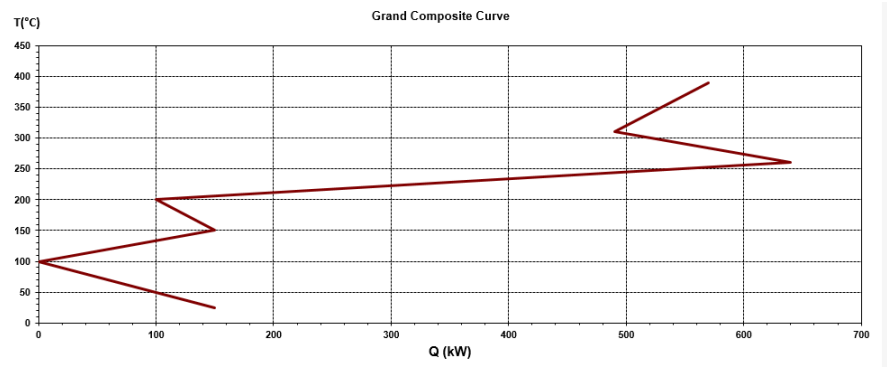

Figure 8.9: GCC for Case 11 without pressure change.

Results

Case 11 is solved with the basic GA and the GA with crowding in the outer loop. The results are presented in Table 8.8 in comparison with the base case. The base case ensures maximum expansion work and minimum compressor work, however, the utility demand increase. The two-level optimization model eliminates the hot utility requirement in the expense of a loss in generated expansion work and higher compressor work consumption.

The basic GA and the GA with crowding located two different solutions. The objective value is only slightly better for the GA with crowding, however, the stream split arrangements are somewhat different. H1 is, in both solutions, split in two branches; one branch with expander inlet at hot pinch temperature and one at new pinch temperature. The compressing streams are split in different arrangements: In the solution found by the basic GA, C1 has positive flow rates in three branches and C3 in one; in the solution found by the GA with crowding, both C1 and C3 have positive flow rates in two branches. The compressor inlet temperatures are conferred to cold pinch, new pinch and ambient temperatures in both solutions. Thus, the results are in accordance with the theorems presented by Fu and Gundersen [18].

Illustrations of the proposed stream split arrangements are presented in Figure 8.10 and Figure 8.11. In the base case, compression occur at ambient temperature and expansion occur at hot utility temperature, involving no stream splits. The base case is illustrated in Figure 8.12.

Case 11		GA.v01	GA.v02	Base case
Exergy [kW]		557.53	557.45	691.10
Hot utility [kW]		0.00	0.00	711.88
Work [kW]		-170.44	-157.51	-241.88
		727.97	714.96	525.82
T_s^{in}/T_s^{out} [°C]	H1	210.60 / 123.69	210.04 / 123.23	400.00 / 279.06
		111.30 / 42.23	110.00 / 41.15	-
	C1	300.00 / 425.36	299.94 / 425.46	15.00 / 78.12
		189.35 / 290.64	189.92 / 291.34	-
		89.86 / 169.36	-	-
	C3	89.86 / 301.79	89.86 / 301.79	15.00 / 183.23
		-	15.00 / 183.23	-
$mc_{p,s}$ [kW/°C]	H1	1.81	1.10	2.00
		0.19	0.90	-
	C1	0.64	0.64	3.00
		1.66	2.36	-
		0.70	-	-
	C3	2	1.02	2.00
		-	0.98	-

Table 8.8: Results from Case 11. Results from the basic GA are listed under GA.v01 and results from the GA with crowding are listed under GA.v02.

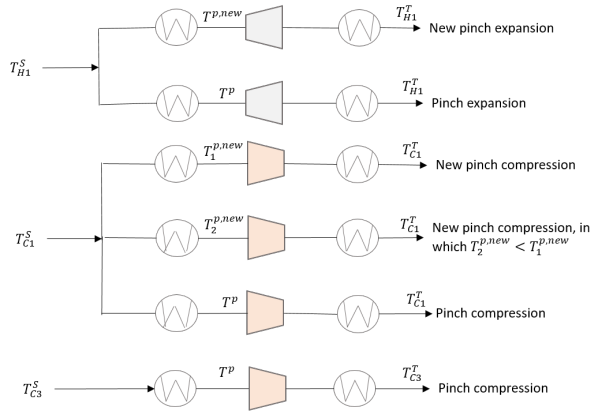

Figure 8.10: Stream split arrangement for Case 11 solved with GA.v01.

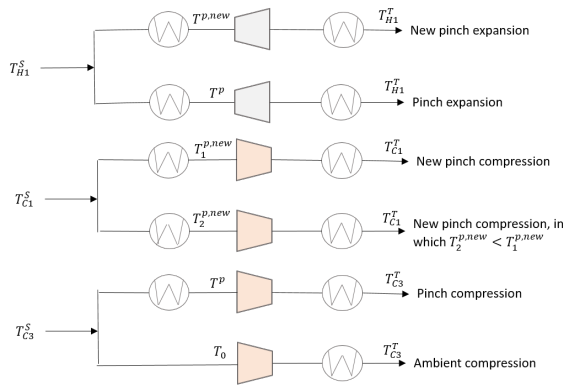

Figure 8.11: Stream split arrangement for Case 11 solved with GA.v02.

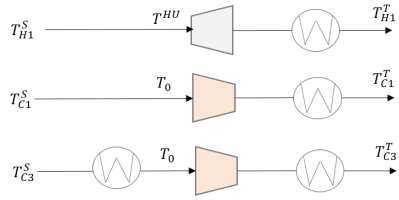

Figure 8.12: Common practice in the industry.

The GCCs curves for the two solutions are presented in Figure 8.13 and Figure 8.14. The shape of the curves illustrate the difference between the two solutions. The cold utility requirement has no effect on the exergy consumption, however, the GCCs reveal a higher cold utility requirement for the solution generated with the basic GA.

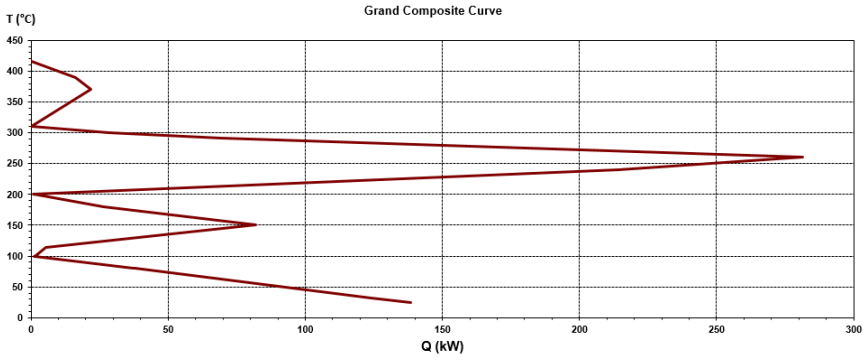

Figure 8.13: GCC for Case 11 solved with GA.v01.

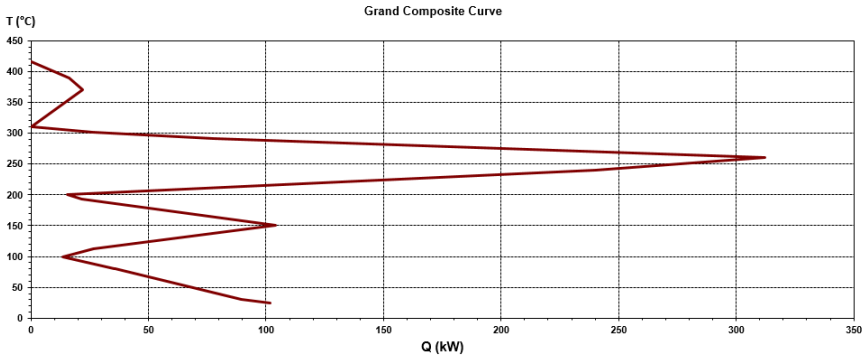

Figure 8.14: GCC for Case 11 solved with GA.v02.

Stream split analysis

Case 11 has been solved for different maximum numbers of stream splits with both algorithms. The results are listed in Table 8.9. In the solutions presented above for maximum three branches, C1 had positive flow rates in all branches. Increasing the number of branches to four did not improve the objective value. Also, none of the pressure changing streams had positive flow rates in more than three branches, suggesting that additional stream splits will not improve the energy efficiency any further. Restricting the number of branches to two yields a 1.43% and 1.58% decrease in exergy consumption compared to no stream splitting, for the basic GA and the GA with crowding, respectively. The small decrease in exergy consumption requires twice the number of pressure changing units. In the case of no stream splitting, the exergy consumption decrease with 17.99% with the basic GA and 18.03% with the GA with crowding in comparison with the base case, requiring no additional units. However, additional heat exchangers will be required in order to reach the inlet temperatures.

	GA.v01				GA.v02			
|B|	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**	$\mathbf{b}_{max}$	$\mathbf{u}_{tot}$	**Ex [kW]**	**time [sec]**
1	1	3	566.75	495.03	1	3	566.47	2204.76
2	2	6	558.65	1460.62	2	6	557.54	8774.94
3	3	6	557.53	3450.82	3	6	557.45	18145.51
4	3	6	557.53	7287.98	4	6	557.46	31766.64

Table 8.9: Stream split analysis for Case 11.

The objective of this work is to identify the energy target for maximum heat and work recovery. Integrating compressors and expanders with the HEN leads to savings in exergy consumption. Stream splitting enhance the energy efficiency even further. The stream split analysis above suggests that, for this case, no more than three branches are required to obtain maximum heat and work recovery. The inlet temperatures are in accordance with the theorems presented by Fu and Gundersen [18], for integration of one compressing and one expanding stream. However, increasing energy efficiency requires investments in additional process equipment, in which the economic implications must be considered in order to evaluate the practicality of the solutions.

8.4 Case 12

Case 12 is an extension of Case 7 adding one pressure changing stream and one expanding stream. The stream data are listed in Table 8.10. The added streams are H4, which are to be expanded from 3 to 1 bar and C3, which are to be compressed from 1 to 5 bar. The contribution from this case is the presence of two compressing and two expanding streams. Two different solutions to this scenario are presented in this section. The results are presented in comparison with a base case of common practice in the industry, in which compression starts at ambient temperature and expansion starts at hot utility temperature.

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	35	2	2	1	20
H2	320	160	4	-	-	
H3	110	35	3	-	-	
H4	400	35	1.5	3	1	
C1	15	380	3	1	2	
C2	190	250	10	-	-	
C3	90	250	2	1	5	

Table 8.10: Stream data for Case 12.

The process pinch point is illustrated on the GCC in Figure 8.15 for heat integration of the process streams without pressure change. The pinch is located at $210/190°C$ and two potential pinch points are identified; one above pinch at $320/300°C$ and one below pinch at $110/90°C$.

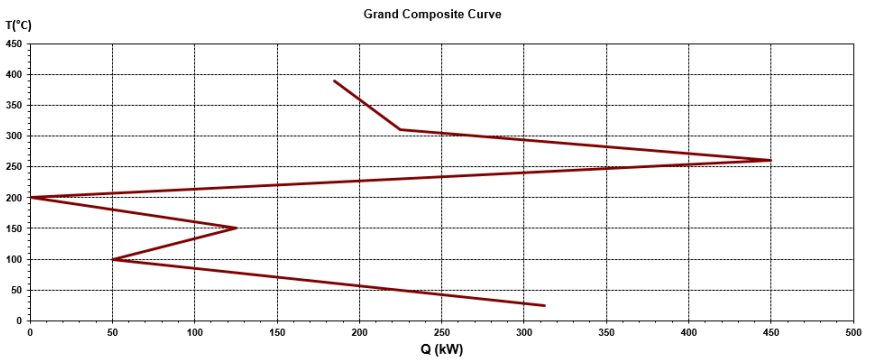

Figure 8.15: GCC for Case 12 without pressure change.

Results

Case 12 is solved with the basic GA and the GA with crowding in the outer loop. The results are presented in Table 8.11. Two different solutions are observed, in which the GA with crowding identified a slightly better objective value than the basic GA. Compression and expansion at ambient and hot utility temperature increase the hot utility requirement. By allowing splitting of streams and pressure change to occur at multiple temperatures, the exergy consumption decrease. Integration of the pressure changing units eliminates the need of external heating at the expense of more compressors work and less expansion work.

From Table 8.11, two stream split arrangements for the pressure changing units are identified. In the solution generated by the basic GA, H1 has positive flow rates in two branches; one that is expanded at hot utility temperature and one that is expanded at hot pinch temperature. In the solution generated by the GA with crowding, H1 is split in one additional branch, expanding at a new pinch temperature. In both solutions, H4 is expanded at the hot pinch temperature. The stream split arrangement for C1 is also similar in the two solutions; one branch involves cold pinch compression, the second involves new pinch compression and the third involves ambient compression. C3 is split in two branches in the solution generated by the basic GA, in which compression occur at new pinch temperature and ambient temperature. In the solution generated by the GA with crowding, C3 is compressed at ambient temperature.

Case 12		**GA.v01**	**GA.v02**	**Base case**
Exergy [kW]		241.73	241.68	411.64
Hot utility [kW]		0.00	0.00	698.90
Work [kW]		-385.95	-377.51	-513.91
		627.67	619.19	525.82
T_s^{in}/T_s^{out} [°C]	H1	400.00 / 279.06	400.00 / 279.06	400.00 / 279.06
		210.04 / 123.23	210.04 / 123.23	-
		-	109.98 / 41.15	-
	H4	210.04 / 79.87	210.04 / 79.87	400.00 / 218.65
	C1	189.35 / 290.65	189.92 / 291.33	-
		89.86 / 169.36	89.86 / 169.36	-
		15.00 / 78.12	15.00 / 78.12	15.00 / 78.12
	C3	89.67 / 301.49	-	-
		15.00 / 183.23	15.00 / 183.23	15.00 / 183.23
$mc_{p,s}$ [kW/°C]	H1	0.50	0.50	2.00
		1.50	1.03	
		-	0.47	
	H4	1.50	1.50	1.50
	C1	2.11	2.42	-
		0.51	0.04	-
		0.38	0.54	3.00
	C3	0.30	-	-
		1.70	2.00	2.00

Table 8.11: Results from Case 12. Results from the basic GA are listed under GA.v01 and results from the GA with crowding are listed under GA.v02.

Illustrations of the proposed stream split arrangements are presented in Figure 8.16 and Figure 8.17. In the base case, compression starts at ambient temperature and expansion starts at hot utility temperature, involving no stream splits, illustrated in Figure 8.18. The GCCs for the stream split arrangements are illustrated in Figure 8.19 and Figure 8.20. The small difference in the two solutions can be observed in the lower pocket.

The results are in accordance with the theorems presented by Fu and Gundersen [18], discussed in Section 4.3. For optimal integration of compressors and expanders in above-ambient HENs, their insight suggests that the inlet temperatures to the compressors should be at pinch temperature, at new pinch temperature or at ambient temperature; and the inlet temperatures to expanders should be at pinch temperature, at new pinch temperature or at hot utility temperature.

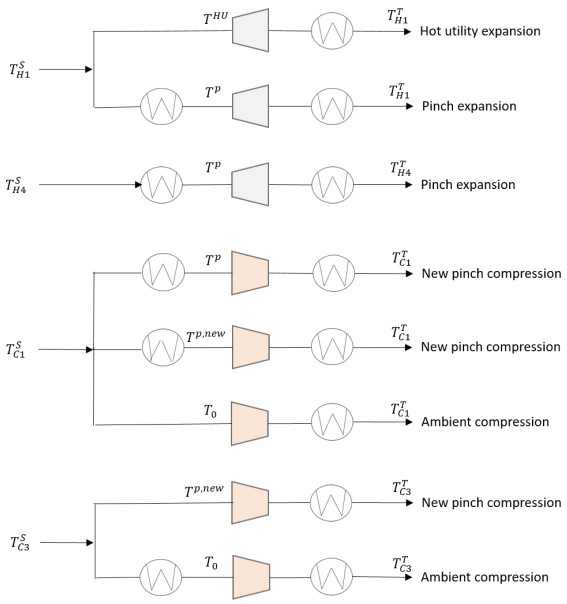

Figure 8.16: Stream split arrangement for Case 12 solved with GA.v01.

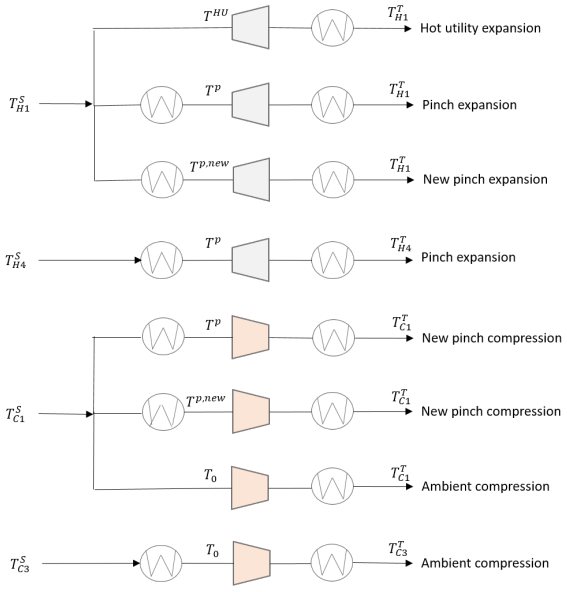

Figure 8.17: Stream split arrangement for Case 12 solved with GA.v02.

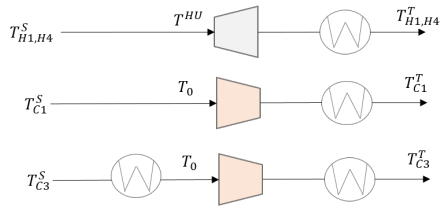

Figure 8.18: Common practice in the industry.

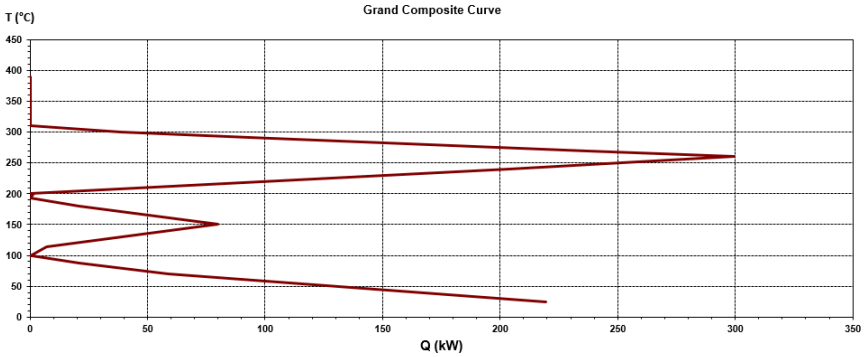

Figure 8.19: GCC for Case 12 solved with GA.v01.

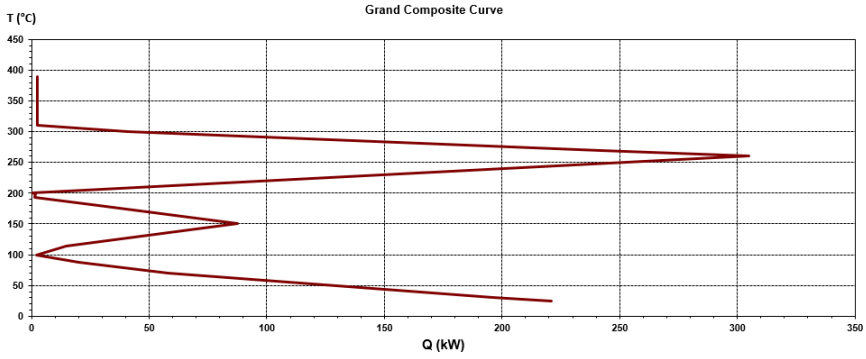

Figure 8.20: GCC for Case 12 solved with GA.v02

Stream split analysis

Case 12 is solved for different values of maximum stream splits. The results are listed in Table 8.12. The implementation experienced issues in displaying the results, hence the incomplete list of computational times. In the solutions presented above, C3 had positive flow rates in all three branches. Increasing the number of maximum branches to four did not improve the objective value. Also, none of the pressure changing streams had positive flow rates in more than three branches, suggesting that additional stream splits will not improve the energy efficiency any further. However, an interesting observation is that the total number of pressure changing units decrease; in which only seven units are required. The solutions found in the case of maximum four stream splits represent slightly less objective values saving the costs of one additional pressure changing unit. A complete outline of the solutions are attached in Appendix B.

	GA.v01				GA.v02			
|B|	b_{max}	u_{tot}	**Ex [kW]**	**time [sec]**	b_{max}	u_{tot}	**Ex [kW]**	**time [sec]**
1	1	4	249.47	849.65	1	4	248.57	4063.82
2	2	7	242.73	-	2	7	241.86	14978.48
3	3	8	241.73	7670.75	3	8	241.68	33855.86
4	3	7	241.82	-	3	7	241.70	-

Table 8.12: Stream data for Case 12.

The objective of this work is to identify the energy target for maximum heat and work recovery. Integrating compressors and expanders with the HEN leads to substantial savings in exergy consumption. Stream splitting enhance the energy efficiency even further. The stream split analysis above suggests that, for this case, no more than three branches are required to obtain maximum heat and work recovery. The inlet temperatures to the pressure changing units are conferred to pinch, ambient and hot utility temperatures, which is in accordance with the theorems presented by Fu and Gundersen [18] for integration of one compressing and one expanding stream. However, increasing energy efficiency require investments in additional process equipment, in which the economic implications must be considered in order to evaluate the practicality of the solution.

Chapter 9

Concluding remarks

This master's thesis studies energy targeting for optimal heat and work integration involving multiple compressing and expanding process streams. The problem of minimizing energy consumption has been formulated as a two-level optimization model utilizing GAs in the search process. The model decouples the problem in two loops; the outer loop uses GAs to locate the optimal inlet temperatures to the pressure changing units and the inner loop identifies the energy target for optimal heat and work integration. The work aim to provide insight into optimal integration of multiple pressure changing streams so to enhance the energy efficiency.

GAs is a heuristic search technique, in which the main limitation is the inability to guarantee a global optimum. The results from the test cases therefore present the hitherto best solutions found. The computational study presents good candidate solutions and provides suggestions for improvements of heat and work integration in the process industry.

The two-level optimization model has solved the test cases using two different GAs in the outer loop, a basic GA and a GA with crowding. Crowding is a niching technique that is suitable for locating the optimum for multimodal problems by allowing for subpopulations to form, exploiting multiple promising areas in the search space. For each test case, an illustration of the search process in finding an optimal solution is attached in Appendix C. The average population fitness and the best fitness value among the individuals are recorded in each generation. Illustrations of the complete search are presented in C.2, whilst illustrations of the early search are presented in C.1. Both algorithms strongly converge early in the search; the GA with crowding converges slightly slower than the basic GA. The GA with crowding maintain a higher population diversity in early generations. When the algorithm has converged towards an area(s) in the search space with high quality solutions, the exploitative effect dominates. The basic GA converges more quickly, and maintain a higher average fitness also later in the search. The GA

with crowding provides slightly better solutions than the basic GA, however, it requires significantly more computational time.

The results from the computational study in the previous section highlight the significant improvements in energy efficiency in applying simultaneous heat and work integration of process streams. In comparison with common practice in the industry, optimization of heat and work integration may eliminate the need of external heating. The results further suggests that compression and expansion at certain temperatures ensure minimum exergy consumption. These temperatures are located at the pinch point, at new pinch points, at ambient temperature and at hot utility temperature. This is in accordance with the theorems presented by Fu and Gundersen [18]. The results from the threshold problem in Case 10, however, suggest that there are cases in which expansion at intermediate temperatures lead to lower exergy consumption.

The stream split analyses illustrate the benefit of stream splitting in terms of energy efficiency. Pressure based energy can effectively be transformed to heating and cooling duty and stream splitting provides an advantageous distribution of this heat in order to increase the energy efficiency. Increasing the stream splits to more than three branches did not lead to improvements in the objective value in any of the test cases, suggesting that more than three branches will not improve the energy efficiency any further. However, the increase in energy efficiency requires investments in additional pressure changing units. Compressors and turbines are some of the most expensive equipment in the process industry, thus, highly energy efficient solutions may be economically impractical. The economic implications of the presented solutions are briefly discussed. An interesting observation from the stream split analyses is that the two-level optimization model, adding the restriction of no stream splits, provides significantly better objective values than the base case. Thus, integration of the pressure changing units with the HEN, increasing the compressor inlet temperatures and decreasing the expander inlet temperatures, yield higher energy efficiency without additional pressure changing units.

The two-level optimization model has proved to be a valuable tool in approaching larger problem sizes. The model efficiently provides good solutions for heat and work integration of multiple pressure changing streams. Additionally, the model has great potential in being extended to include the costs related to the required process equipment. For increased problem sizes, heuristic search methods are more likely to outperform exact solution methods. Combining the two methods reduces the number of decision variables that must be handled by the search algorithm. A distribution of tasks between heuristic search algorithms and exact methods may be an efficient solution method in approaching industrial size problems.

Chapter 10

Future research

There are many possible areas for future research into the heat and work integration problem, and the two-level optimization model. For this thesis, the objective was to identify the energy target for optimal heat integration of processes involving multiple pressure changing streams. In order to enhance the energy efficiency, the model allows for stream splitting, which requires additional operating units. Heat exchanger equipment and pressure changing units are costly. The solutions presented by the two-level optimization model are energy efficient, but may be economically impractical. Considering maximum heat integration and minimum total costs simultaneously may be resolved with multi-objective optimization, in which GAs are a suitable approach. This can make the solution method more applicable for the decision makers in the process industry. Further work should also concern the network design, satisfying the energy target identified by the two-level optimization model.

The heat and work integration problem can be further extended by allowing for the pressure ratios to be subject to optimization. Motivated by the work of Wechsung et al. [71], another possibility is to let any stream go through a series of pressure changes with the pressure ratios and the inlet temperatures as variables. In this case, the streams will partially act as utilities. The LP model assumes constant heat capacities and neglect any pressure loss over the heat exchangers. A more accurate model may take these simplifications into consideration.

Another possible area of research is to improve the performance of the outer loop. GAs are often used for large sized global optimization problems, but are not very efficient in local search. Similar to GAs, the Nelder-Mead simplex algorithm do not use function derivatives and deals with a population of points. However, this algorithm can only find a local optimum close to the starting point [13]. In order to improve the local search, a combined Nelder-Mead simplex and genetic algorithm may improve the performance of

the two-level optimization model. The aim of the GA is to find promising areas for the simplex algorithm whereas the latter will find the local optimum in this area. An individual is generally coded by the variables of the problem, however, in a combined Nelder-Mead simplex and genetic algorithm each individual can be defined as a simplex. The GA selects and recombine good simplexes. In addition, a few steps of the Nelder-Mead algorithm are executed at each generation to improve the local search.

The GAs developed in this work make use of binary representation, which is not always well suited for real value decisions. There are several drawbacks to binary representation, one being the discretization of the variable range. The number of bits representing each gene has a large influence on the computational efficiency. With real-value encoding there is no need to convert bit strings. Also, real-value encoding increases the precision in not having to discretize the value range.

For further improvements of the GAs, the parameters can be automatically controlled during execution as opposed to manually tuned in advanced. With automatic parameter control the parameters can be adapted to the state of the search process; e.g., it is reasonable to gradually reduce the degree of exploration during the search. Diversity-adaptive control of the scaling factor ϕ uses feedback from the GA population to determine the magnitude of the change in ϕ [53]. High values of ϕ in early generations prevent premature convergence; low values of ϕ in the last generations favours regions close to local optima, making sure these region are effectively exploited. In applying linear ranking selection, the c parameter may be adapted to control the selection bias throughout the search. Self-adapted mutation probability may also improve the search process.

For increasing problem sizes the use of GAs becomes computationally demanding. GAs are easily parallelized, consisting of a number of independent tasks that can be calculated simultaneously, which will reduce the computational time considerably.

Bibliography

[1] Audun Aspelund, David Olsson Berstad, and Truls Gundersen. An extended pinch analysis and design procedure utilizing pressure based exergy for subambient cooling. *Applied Thermal Engineering*, 27(16):2633–2649, 2007.

[2] Thomas Back. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 57–62. IEEE, 1994.

[3] Thomas Back, Ulrich Hammel, and H-P Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[4] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[5] Jerker Björkqvist. *Discrete and disjunctive optimization: Parallel strategies and applications in industrial scheduling*. Åbo Akademi University, 2001.

[6] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms, 1995.

[7] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In *ICGA*, pages 9–16. Citeseer, 1995.

[8] Jaime Cerda, Arthur W Westerberg, David Mason, and Bodo Linnhoff. Minimum utility usage in heat exchanger network synthesis a transportation problem. *Chemical Engineering Science*, 38(3):373–387, 1983.

[9] Changhyun Kwon. *Julia Programming for Operations Research; A Primer on Computing*. Changhyun Kwon, 2016.

[10] Jie Chen, Bin Xin, Zhihong Peng, Lihua Dou, and Juan Zhang. Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization. *IEEE*

Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 39(3):680–691, 2009.

[11] Alexander W. Dowling. Mathematical programming approaches to balance heat-work trade-offs in above ambient systems. 2015.

[12] Marco A Duran and Ignacio E Grossmann. Simultaneous optimization and heat integration of chemical processes. *AIChE Journal*, 32(1):123–138, 1986.

[13] Nicolas Durand and Jean-Marc Alliot. A combined nelder-mead simplex and genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, volume 99, pages 1–7, 1999.

[14] Vladimir Filipović. Fine-grained tournament selection operator in genetic algorithms. *Computing and Informatics*, 22(2):143–161, 2012.

[15] Christodoulos A Floudas, Amy R Ciric, and Ignacio E Grossmann. Automatic synthesis of optimum heat exchanger network configurations. *AIChE Journal*, 32(2):276–290, 1986.

[16] Chao Fu and Truls Gundersen. Integrating compressors into heat exchanger networks above ambient temperature. *AIChE Journal*, 61(11):3770–3785, 2015.

[17] Chao Fu and Truls Gundersen. Integrating expanders into heat exchanger networks above ambient temperature. *AIChE Journal*, 61(10):3404–3422, 2015.

[18] Chao Fu and Truls Gundersen. Correct integration of compressors and expanders in above ambient heat exchanger networks. *Energy*, 116:1282–1293, 2016.

[19] Chao Fu, Preben U. Maurstad, Bjørn Nygreen, and Truls Gundersen. Compression and expansion at the right pinch temperature. *Chemical Engineering Transactions*, 2016.

[20] Kevin C Furman and Nikolaos V Sahinidis. A critical review and annotated bibliography for heat exchanger network synthesis in the 20th century. *Industrial & Engineering Chemistry Research*, 41(10):2335–2370, 2002.

[21] Gadfly homepage. Gadfly. `http://gadflyjl.org/stable/`, 2017. Online; accessed 17-06-2017.

[22] Severino F Galan and Ole J Mengshoel. Generalized crowding for genetic algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 775–782. ACM, 2010.

[23] Mitsuo Gen and Runwei Cheng. *Genetic algorithms and engineering optimization*, volume 7. John Wiley & Sons, 2000.

[24] Github, Inc. Julia kernel for Jupyter . `https://github.com/JuliaLang/IJulia.jl`, 2017. Online; accessed 17-06-2017.

[25] Github, Inc. JuMP â€" Julia for Mathematical Optimization. https://jump.readthedocs.io/en/latest/, 2017. Online; accessed 17-06-2017.

[26] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991.

[27] Louis Gosselin, Maxime Tye-Gingras, and François Mathieu-Potvin. Review of utilization of genetic algorithms in heat transfer problems. *International Journal of Heat and Mass Transfer*, 52(9):2169–2188, 2009.

[28] John J Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, 16(1):122–128, 1986.

[29] Lewis E Grimes, Michael D Rychener, and Arthur W Westerberg. The synthesis and evolution of networks of heat exchange that feature the minimum number of units. *Chemical Engineering Communications*, 14(3-6):339–360, 1982.

[30] Ignacio E Grossmann, Héctor Yeomans, and Zdravko Kravanja. A rigorous disjunctive optimization model for simultaneous flowsheet optimization and heat integration. *Computers & chemical engineering*, 22:S157–S164, 1998.

[31] T Gundersen and L Naess. The synthesis of cost optimal heat exchanger networks: an industrial review of the state of the art. *Computers & chemical engineering*, 12(6):503–530, 1988.

[32] Truls Gundersen. An introduction to the concept of exergy and energy quality, 2011.

[33] Edward Charles Hohmann. *Optimum networks for heat exchange*. PhD thesis, University of Southern California Los Angeles, CA, 1971.

[34] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[35] Jacek Jezowski. Heat exchanger network grassroot and retrofit design, the review of the state of the art: Part ii, heat exchanger network synthesis by mathematical methods and approaches for retrofit design. 1994.

[36] KAD Jong. *An Analysis of the Behavior of a Class of Genetic Algorithm Adaptive System*. PhD thesis, Ann Arbor, USA: University of Michigan, 1975.

[37] Julia homepage. Julia. https://julialang.org/, 2017. Online; accessed 09-06-2017.

[38] Jupyter homepage. The Jupyter Notebook. http://jupyter.org/, 2017. Online; accessed 17-06-2017.

[39] Ian C. Kemp. *Pinch Analysis and Process Integration*. Elsevier Ltd., 2007.

[40] Daniel R Lewin. A generalized method for hen synthesis using stochastic optimizationâ€"ii.: The synthesis of cost-optimal networks. *Computers & chemical engineering*, 22(10):1387–1405, 1998.

[41] Daniel R Lewin, Hao Wang, and Ofir Shalev. A generalized method for hen synthesis using stochastic optimization–i. general framework and mer optimal synthesis. *Computers & chemical engineering*, 22(10):1503–1513, 1998.

[42] Fang-Fang Li, Christine A Shoemaker, Jun Qiu, and Jia-Hua Wei. Hierarchical multi-reservoir optimization modeling for real-world complexity with application to the three gorges system. *Environmental Modelling & Software*, 69:319–329, 2015.

[43] Wen-Yang Lin, Wen-Yung Lee, and Tzung-Pei Hong. Adapting crossover and mutation rates in genetic algorithms. *J. Inf. Sci. Eng.*, 19(5):889–903, 2003.

[44] B Linnhoff and S Parker. Heat exchanger networks with process modifications. In *IChemE Annual Research Meeting*. Bath, 1984.

[45] Bodo Linnhoff and John R Flower. Synthesis of heat exchanger networks: I. systematic generation of energy optimal networks. *AIChE Journal*, 24(4):633–642, 1978.

[46] Bodo Linnhoff and Eric Hindmarsh. The pinch design method for heat exchanger networks. *Chemical Engineering Science*, 38(5):745–763, 1983.

[47] Bodo Linnhoff and R Vredeveld. Pinch technology has come of age. *Chemical engineering progress*, 80(7):33–40, 1984.

[48] Xing Luo, Qing-Yun Wen, and Georg Fieg. A hybrid genetic algorithm for synthesis of heat exchanger networks. *Computers & Chemical Engineering*, 33(6):1169–1181, 2009.

[49] Noriyasu Maehara and Yoshiyuki Shimoda. Application of the genetic algorithm and downhill simplex methods (nelder–mead methods) in the search for the optimum chiller configuration. *Applied Thermal Engineering*, 61(2):433–442, 2013.

[50] Samir W Mahfoud. Crowding and preselection revisited. *Urbana*, 51:61801, 1992.

[51] Samir W Mahfoud. Niching methods for genetic algorithms. *Urbana*, 51(95001):62–94, 1995.

[52] Preben Uv Maurstad. Optimal Design of Heat Exchanger Networks with Pressure Changes. Master's thesis, Norwegian University of Science of Technology, Trondheim, Norge, 2016.

[53] Ole J Mengshoel, Severino F Galán, and Antonio De Dios. Adaptive generalized crowding for genetic algorithms. *Information Sciences*, 258:140–159, 2014.

[54] Ole J Mengshoel and David E Goldberg. The crowding approach to niching in genetic algorithms. *Evolutionary computation*, 16(3):315–354, 2008.

[55] Michael J Moran. Engineering thermodynamics. In *The CRC Handbook of Mechanical Engineering, Second Edition*. CRC Press, 2004.

[56] Naonori Nishida, George Stephanopoulos, and Arthur W Westerberg. A review of process synthesis. *AIChE Journal*, 27(3):321–351, 1981.

[57] Viviani C Onishi, Mauro ASS Ravagnani, and José A Caballero. Simultaneous synthesis of heat exchanger networks with pressure recovery: optimal integration between heat and work. *AIChE Journal*, 60(3):893–908, 2014.

[58] Viviani C Onishi, Mauro ASS Ravagnani, and José A Caballero. Simultaneous synthesis of heat exchanger networks with pressure recovery: optimal integration between heat and work. *AIChE Journal*, 60(3):893–908, 2014.

[59] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. A comparative review of approaches to prevent premature convergence in ga. *Applied Soft Computing*, 24:1047–1077, 2014.

[60] Soterios A Papoulias and Ignacio E Grossmann. A structural optimization approach in process synthesis 2: Heat recovery networks. *Computers and Chemical Engineering*, 7(6):707–721, 1983.

[61] Petter Rökke. Energy efficiency in industrial processe. `http://www.eera-set.eu/wp-content/uploads/2015-06-16-EUSEW-Energy-efficiency-Petter-Roekke-1.pdf`, 2016. Online; accessed 28-10-2016.

[62] Frank Pettersson and Jarmo Söderman. Design of robust heat recovery systems in paper machines. *Chemical Engineering and Processing: Process Intensification*, 46(10):910–917, 2007.

[63] MASS Ravagnani, AP Silva, PA Arroyo, and AA Constantino. Heat exchanger network synthesis and optimisation using genetic algorithm. *Applied Thermal Engineering*, 25(7):1003–1017, 2005.

[64] SN Sivanandam and SN Deepa. *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[65] William M Spears and Kenneth D De Jong. On the virtues of parameterized uniform crossover. Technical report, DTIC Document, 1995.

[66] Tet al Umeda, J Itoh, and K Shiroko. Heat-exchange system synthesis. *Chemical Engineering Progress*, 74(7):70–76, 1978.

[67] Tomio Umeda, Kazuo Niida, and Katsuo Shiroko. A thermodynamic approach to heat integration in distillation systems. *AIChE Journal*, 25(3):423–429, 1979.

[68] U.S. Energy Information Administration. 2010 MECS Survey Data, Table 1.2. https://www.eia.gov/consumption/manufacturing/data/2010/#r1, 2010. Online; accessed 03-05-2017.

[69] U.S. Energy Information Administration. Circor Energy. http://www.circorpowerprocess.com/markets-served/process-industry-industrial-gas.php, 2017. Online; accessed 03-05-2017.

[70] Kefeng Wang, Yu Qian, Yi Yuan, and Pingjing Yao. Synthesis and optimization of heat integrated distillation systems using an improved genetic algorithm. *Computers & chemical engineering*, 23(1):125–136, 1998.

[71] Achim Wechsung, Audun Aspelund, Truls Gundersen, and Paul I Barton. Synthesis of heat exchanger networks at subambient conditions with compression and expansion of process streams. *AIChE Journal*, 57(8):2090–2108, 2011.

[72] Hongmei Yu, Haipeng Fang, Pingjing Yao, and Yi Yuan. A combined genetic algorithm/simulated annealing algorithm for large scale system energy integration. *Computers & Chemical Engineering*, 24(8):2023–2035, 2000.

Appendix A

Benchmark cases

The results from the two-level optimization model, using a basic GA and a GA with crowding in the outer loop, in comparison with the results presented in the master thesis of Maurstad [52] are conferred to this appendix. The comparable methods are the graphical procedure presented by Fu and Gundersen [16][17] and two exact solution methods developed by Maurstad. The exact solution methods are an LP model using insight from the theorems presented by Fu and Gundersen [16][17] and an MINLP model without the use of insight. Eight cases were tested in total, whereof Case 7 and Case 8 were the only two cases involving two pressure changing streams, one compressing and one expanding stream. Case 8 investigate heat and work integration for below ambient temperatures. The GA parameter values are listed in Table A.1

	Parameter values	
Parameter	**Basic GA**	**GA crowding**
lower bound on design variables	$T_0(T^{CU})$	$T_0(T^{CU})$
upper bound on design variables	$T^{HU}(T_0)$	$T^{HU}(T_0)$
number of bits	11	11
population size	20×paraNum	20×paraNum
rate of elitism, ε	0.01	0.01
crossover probability, P_c	0.75	0.75
mutation probability, P_m	0.10	0.10
tournament size, t	2	-
scaling factor, ϕ	-	0.15
iterations	1000	1000

Table A.1: Parameter values for the basic GA and the GA with crowding.

A.1 Case 1

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	60	3	-	-	20
C1	300	380	2	3	1	
C2	300	380	6	-	-	

Table A.2: Stream data for Case 1.

Case 1		**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy	[kW]	168.2	157.5	157.5	157.5	157.5
Hot utility	[kW]	740.0	740.0	740.0	740.0	740.0
Work	[kW]	-255.0	-265.7	-265.7	-265.7	-265.7
T_{in}/T_{out}	[°C]	200.0 / 72.5	220.0 / 87.1	220.0 / 87.1	220.0 / 87.1	220.0 / 87.1
mc_p	[kW/°C]	2.0	2.0	2.0	2.0	2.0

Table A.3: Results from Case 1.

A.2 Case 2

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	110	2	-	-	20
H2	400	280	3	2.5	1	
C1	200	380	8	-	-	

Table A.4: Stream data for Case 2.

Case 2		**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy	[kW]	143.3	152.3	134.6	134.6	134.6
Hot utility	[kW]	923.1	853.4	853.5	853.5	853.5
Work	[kW]	-384.7	-336.3	-353.5	-353.5	-353.5
T_{in}/T_{out}	[°C]	220.0/106.4	220.0/106.4	220.0/106.4	220.0/106.4	220.0/106.4
		400.0/245.0	210.4/99.9	400.0/225.0	400.0/245.0	400.0/245.0
		-	-	126.4 / 34.4	195.4 / 87.5	198.4 / 89.8
mc_p	[kW/°C]	1.9	1.0	2.0	0.6	0.5
		1.1	2.0	0.5	0.6	0.5
		-	-	0.5	1.8	2.0

Table A.5: Results from Case 2.

A.3 Case 3

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	130	2	-	-	20
H2	400	130	3	5	1	
C1	200	380	8	-	-	

Table A.6: Stream data for Case 3.

Case 3		**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy	[kW]	-205.8	-205.8	-205.8	-205.8	-205.8
Hot utility	[kW]	691.0	691.0	691.0	691.0	691.0
Work	[kW]	- 601.0	- 601.0	- 601.0	- 601.0	- 601.0
T_{in}/T_{out}	[°C]	400.0/151.9	270.3/70.0	400.0/151.9	396.4/149.6	396.4/149.6
		220.0 / 38.2	-	220.0 / 38.2	244.1 / 53.4	244.1 / 53.4
mc_p	[kW/°C]	0.8	3	0.8	0.6	0.6
		2.2	-	2.2	2.4	2.4

Table A.7: Results from Case 3.

A.4 Case 4

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	60	3	-	-	20
H2	400	280	2	25	1	
C1	200	380	8	-	-	

Table A.8: Stream data for Case 4.

Case 4		**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy	[kW]	-203.3	-203.3	-203.3	-203.3	-203.3
Hot utility	[kW]	1060.0	1060.0	1060.0	1060.0	1060.0
Work	[kW]	-809.6	-809.6	-809.6	-809.6	-809.6
T_{in}/T_{out}	[°C]	400.0 / -4.8	400.0 / -4.8	400.0 / -4.8	400.0 / -4.8	400.0 / -4.8
mc_p	[kW/°C]	2	2.0	2.0	2.0	2.0

Table A.9: Results from Case 4.

A.5 Case 5

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	60	3	3	1	20
H2	330	80	9	-	-	
C1	15	220	6	-	-	
C2	140	380	8	-	-	

Table A.10: Stream data for Case 5.

Case 5	**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy [kW]	-206.4	-203.8	-206.3	-206.3	-206.3
Hot utility [kW]	350.0	350.0	350.0	350.0	350.0
Work [kW]	-406.6	-404.0	-406.4	-406.4	-406.4
T_{in}/T_{out} [°C]	330.0/167.5	226.7/92.0	330.0/167.5	330.0/167.5	330.0/167.5
	160.0 / 43.3	-	160.0 / 43.3	160.0 / 43.3	160.0 / 43.3
mc_p [kW/°C]	1.2	3	1.2	1.2	1.2
	1.8	-	1.8	1.8	1.8

Table A.11: Results from Case 5.

A.6 Case 6

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	60	6	3	1	20
H2	330	80	9	-	-	
C1	15	220	6	-	-	
C2	140	380	8	-	-	
C3	40	380	3	-	-	

Table A.12: Stream data for Case 6.

Case 6	**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy [kW]	-470.2	-436.2	-470.4	-470.4	-470.4
Hot utility [kW]	819.0	738.8	818.5	818.5	818.5
Work [kW]	-938.6	-858.8	-938.5	-938.5	-938.5
T_{in}/T_{out} [°C]	400.0/218.7	258.1/115.0	400.0/218.7	400.0/218.7	400.0 / 218.7
	160.0 / 43.3	-	160.0 / 43.3	160.0 / 43.3	160.0 / 43.3
mc_p [kW/°C]	3.7	6	3.7	3.7	3.7
	2.3	-	2.3	2.3	2.3

Table A.13: Results from Case 6.

A.7 Case 7

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	400	35	2	2	1	20
H2	320	160	4	-	-	
H3	110	35	3	-	-	
C1	15	380	3	1	2	
C2	190	250	10	-	-	

Table A.14: Stream data for Case 7.

Case 7		**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy	[kW]	175.6	178.1	175.6	175.6	175.6
Hot utility	[kW]	37.6	27.7	37.5	37.5	37.5
Work	[kW]	-158.3	-189.7	-158.4	-158.4	-158.4
		312.4	352.0	312.5	312.5	312.5
T_{in}/T_{out}	[°C]	210.0/123.2	254.9/160.0	210.0/123.2	210.0/123.2	210.0/123.2
		110.0/41.2	262.7/380.0	110.0/41.2	111.3/42.2	110.0/41.2
		190.0/291.4	-	190.0/291.4	190.1/291.6	190.1/291.6
		300.0/425.5	-	300.0/425.5	297.7/422.7	300.0/425.5
mc_p	[kW/°C]	1.15	2.0	1.15	1.14	1.15
		0.85	3.0	0.85	0.86	0.85
		2.66	-	2.66	2.56	2.66
		0.34	-	0.34	0.34	0.34

Table A.15: Results from Case 7.

A.8 Case 8

Stream	$\boldsymbol{T^S}$ [°C]	$\boldsymbol{T^T}$ [°C]	$\boldsymbol{mc_p}$ [kW/°C]	$\boldsymbol{P^S}$ [bar]	$\boldsymbol{P^T}$ [bar]	$\boldsymbol{\Delta\, T_{min}}$
H1	15	-149	2	3	1	20
H2	-21	-105	4	-	-	
C1	-135	11	3	1	2	
C2	-75	-38	7	-	-	

Table A.16: Stream data for Case 8.

Case 8	**Graphical**	**MINLP**	**LP**	**GA.v01**	**GA.v02**
Exergy [kW]	54.6	63.0	52.7	53.0	52.8
Hot utility [kW]	13.0	6.9	23.2	23.7	23.6
Work [kW]	-99.1	-97.1	-103.5	-104.5	-103.6
	135.5	139.2	123.8	124.8	123.8
T_{in}/T_{out} [°C]	-71.0/-125.4	-93.0/-141.5	-71.0/-125.4	-69.0/-124.0	-71.0/-125.4
	-131.0/-169.3	-61.3/-14.9	-131.0/-169.3	-129.4/-168.1	-130.6/-169.0
	-75.0/-31.6	-	-75.0/-31.6	-75.0/-31.6	-75.0/-31.6
	-25.0/27.2	-	-135.0/-104.7	-134.6/-104.3	-75.0/-31.6
mc_p [kW/°C]	1.4	2.0	1.7	1.7	1.7
	0.6	3.0	0.3	0.3	0.3
	2.5	-	2.5	2.6	2.5
	0.5	-	0.5	0.4	0.5

Table A.17: Results from Case 8.

Appendix B

Additional results from Case 12

Case 12		**GA.v01**	**GA.v02**
Exergy [kW]		241.82	241.70
Hot utility [kW]		0.00	0.00
Work [kW]		-385.95	-377.51
		627.67	619.19
T_s^{in}/T_s^{out} [°C]	H1	399.25 / 278.44	400.00 / 279.06
		210.04 / 123.23	210.04 / 123.23
		-	111.30 / 42.23
	H4	210.04 / 79.87	210.04 / 79.87
	C1	188.60 / 289.73	189.92 / 291.33
		89.10 / 168.44	-
		18.01 / 81.78	15.00 / 78.12
	C3	15.00 / 183.23	15.00 / 183.23
$mc_{p,s}$ [kW/°C]	H1	0.50	0.50
		1.50	1.00
		-	0.50
	H4	1.50	1.50
	C1	2.47	2.42
		0.51	-
		0.02	0.58
	C3	2.00	2.00

Table B.1: Results from Case 12 with four allowable branches.

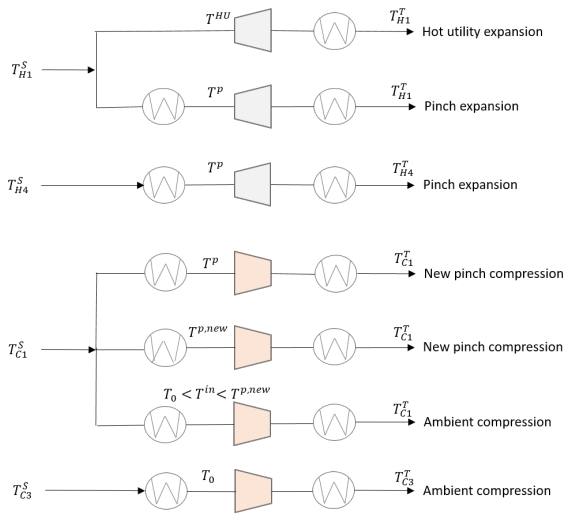

Figure B.1: Stream split arrangement for Case 12 solved with GA.v01.

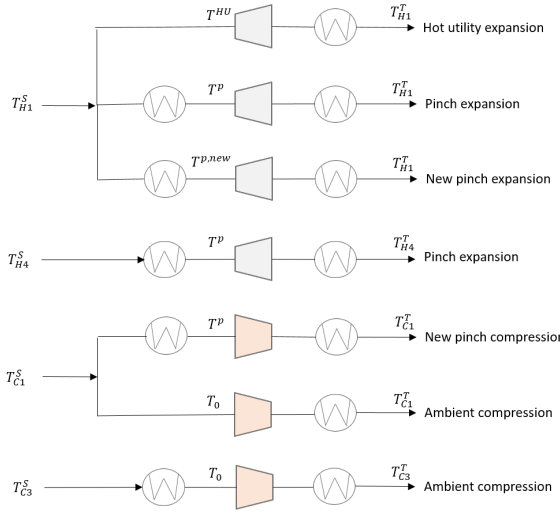

Figure B.2: Stream split arrangement for Case 12 solved with GA.v02.

Appendix C

Illustration of search process

C.1 Illustration of early search process

Case 9

Figure C.1: Search process for basic GA.

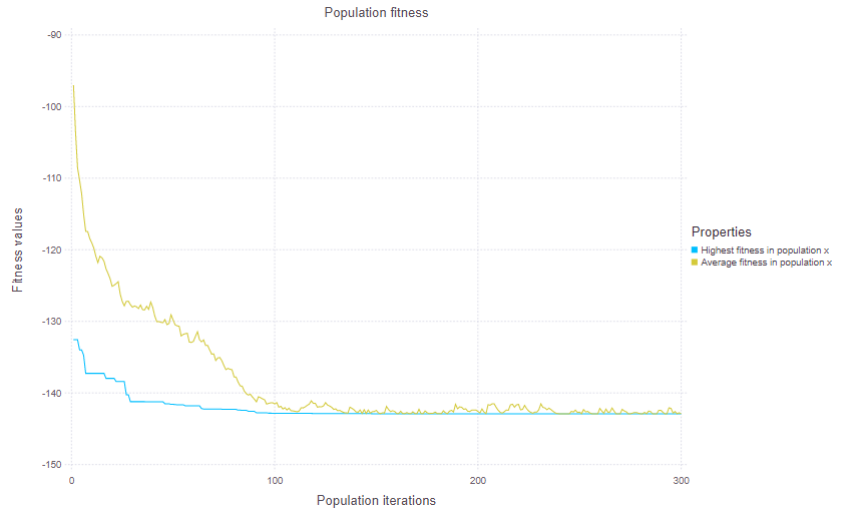

Figure C.2: Search process for GA with crowding.

Case 10

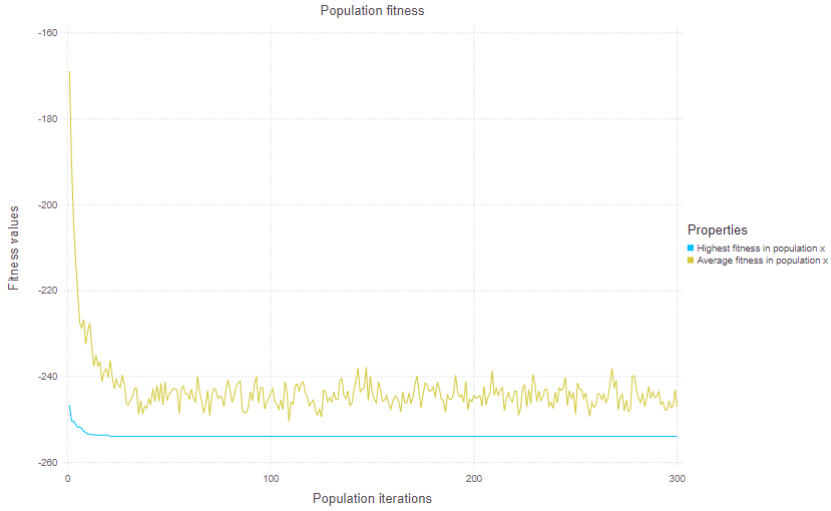

Figure C.3: Search process for basic GA.

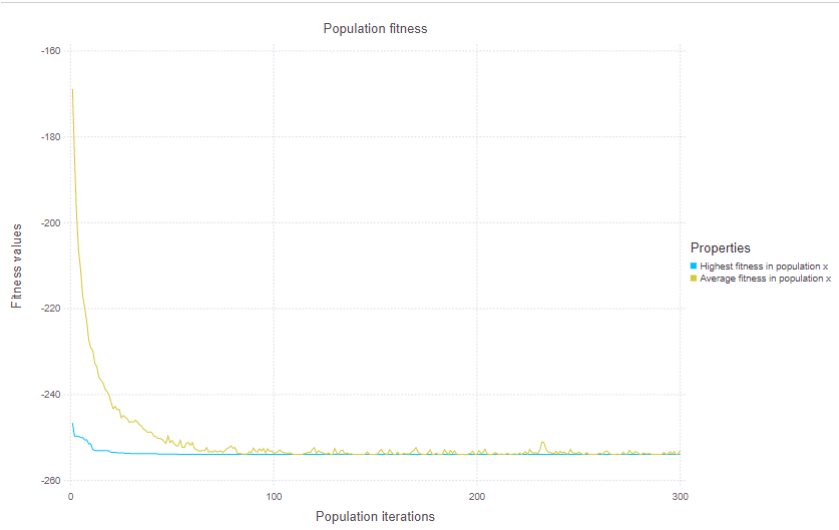

Figure C.4: Search process for GA with crowding.

Case 11

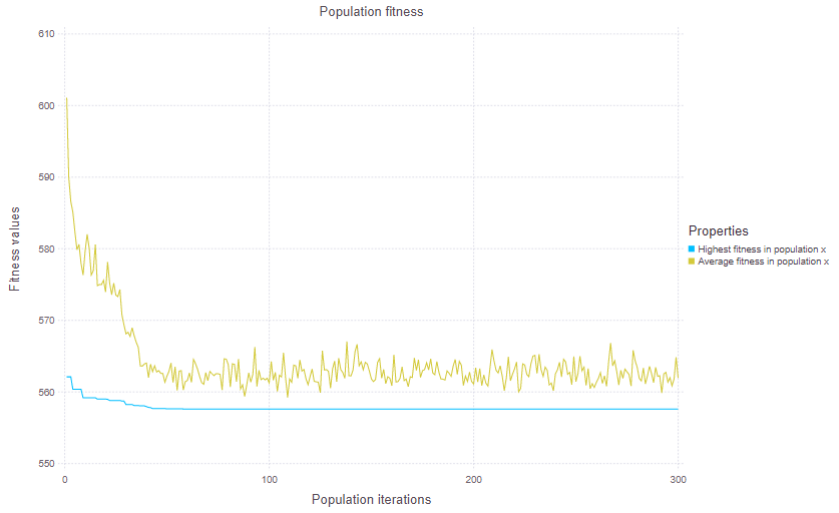

Figure C.5: Search process for basic GA.

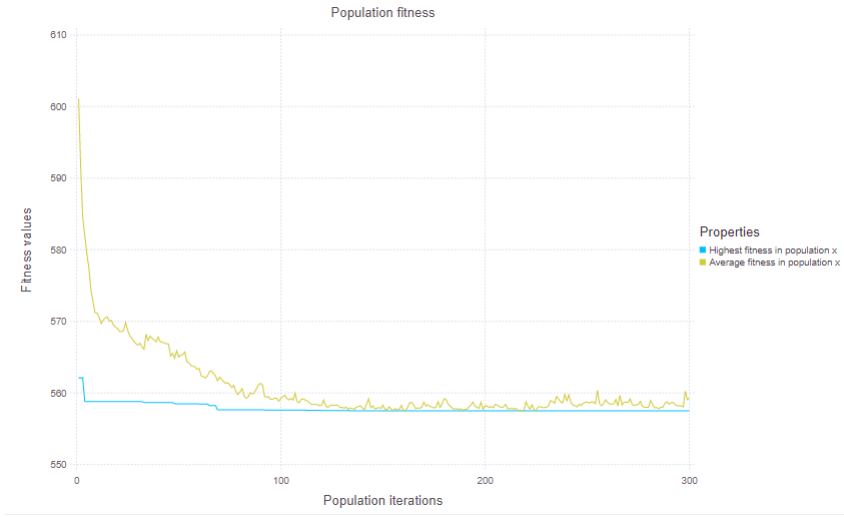

Figure C.6: Search process for GA with crowding.

Case 12

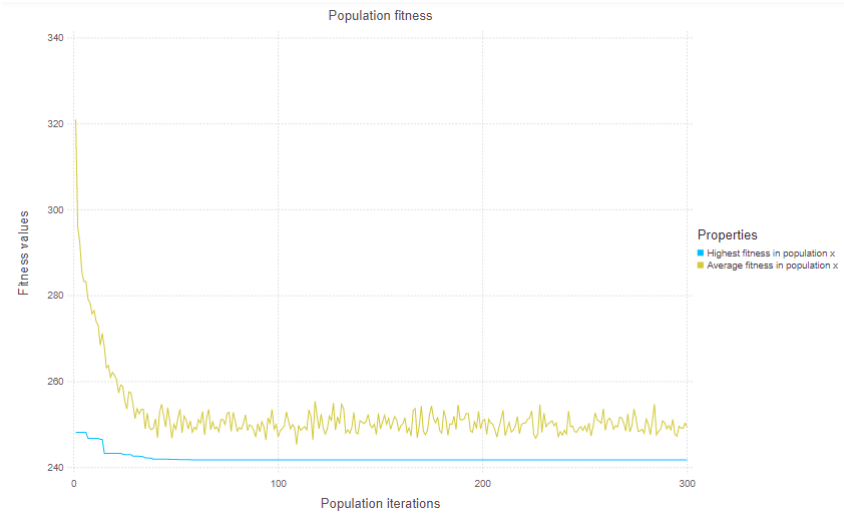

Figure C.7: Search process for basic GA.

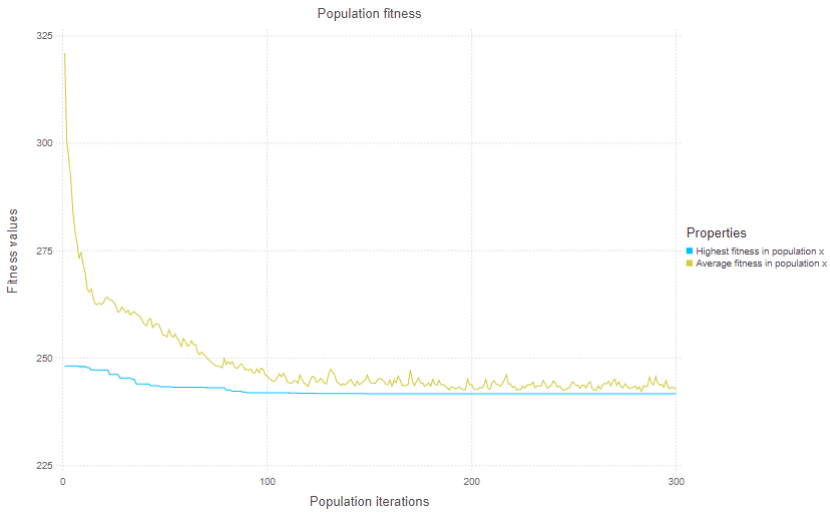

Figure C.8: Search process for GA with crowding.

C.2 Illustration of complete search process

Case 9

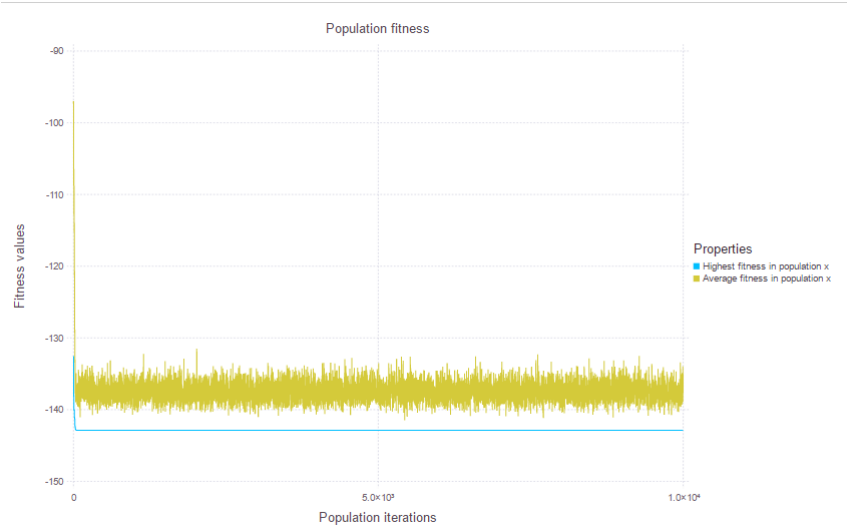

Figure C.9: Search process for basic GA.

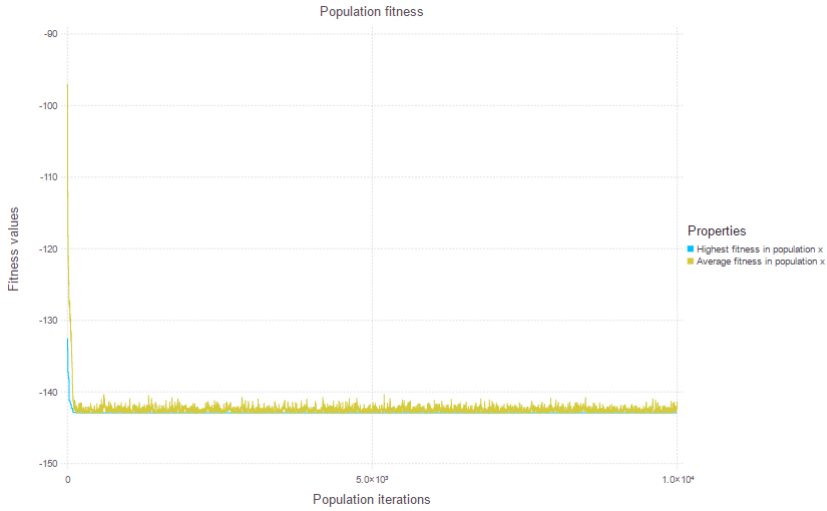

Figure C.10: Search process for GA with crowding.

Case 10

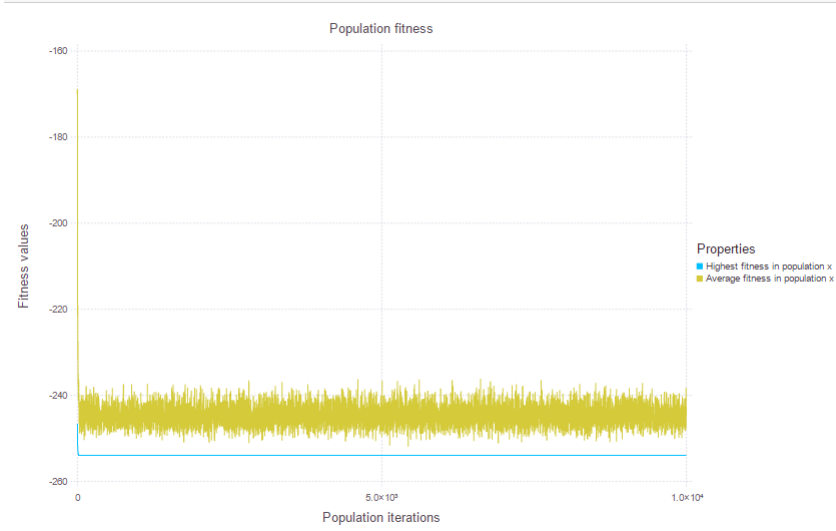

Figure C.11: Search process for basic GA.

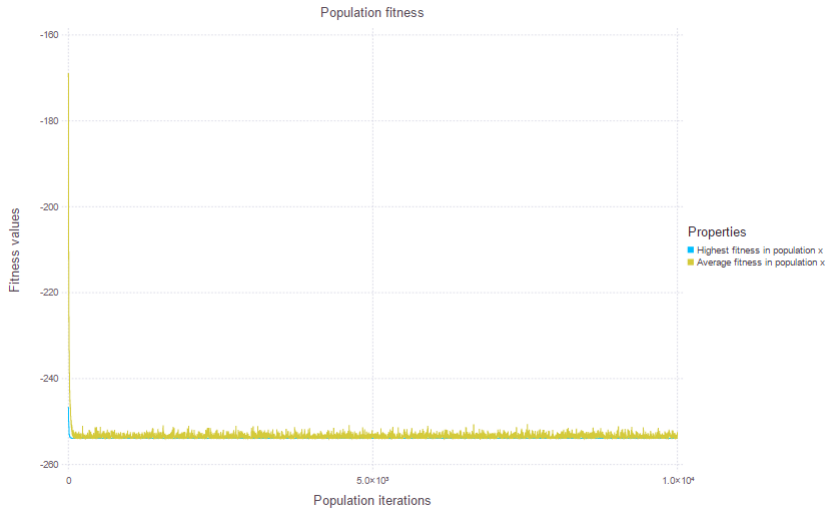

Figure C.12: Search process for GA with crowding.

Case 11

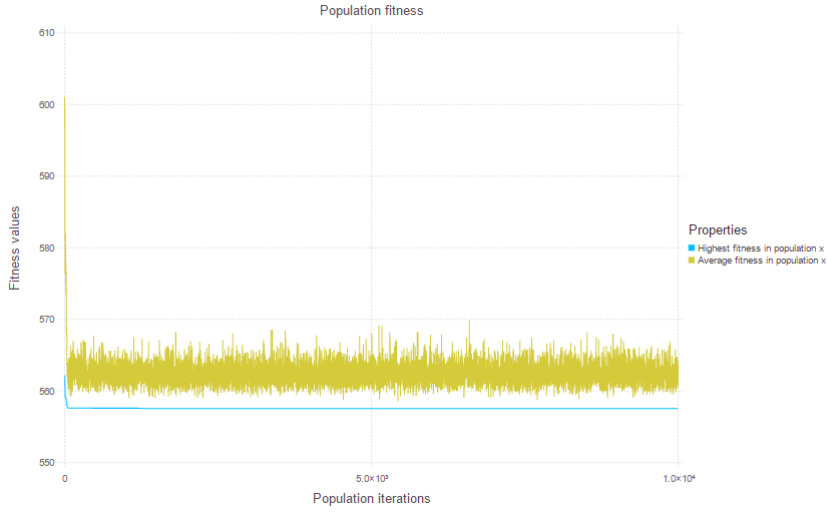

Figure C.13: Search process for basic GA.

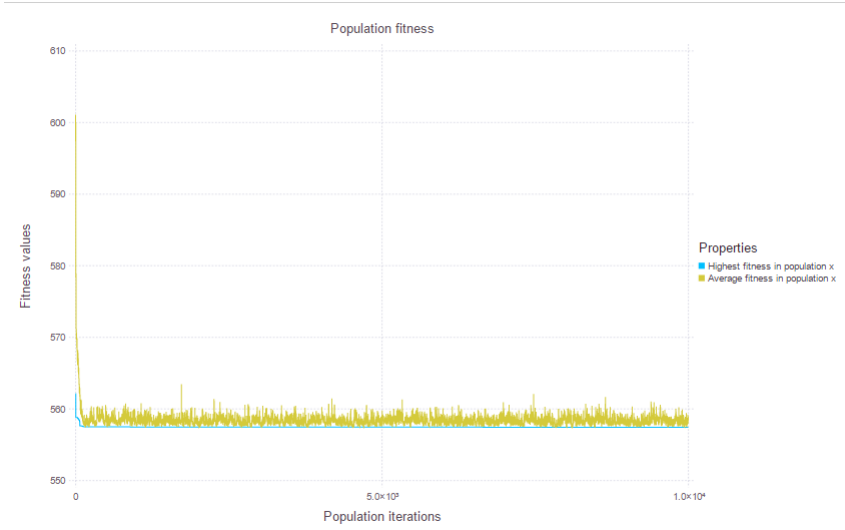

Figure C.14: Search process for GA with crowding.

Case 12

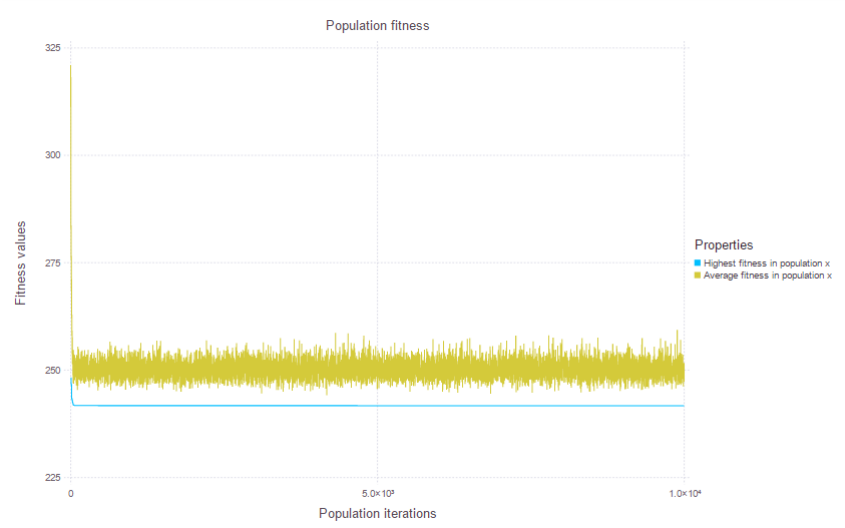

Figure C.15: Search process for basic GA.

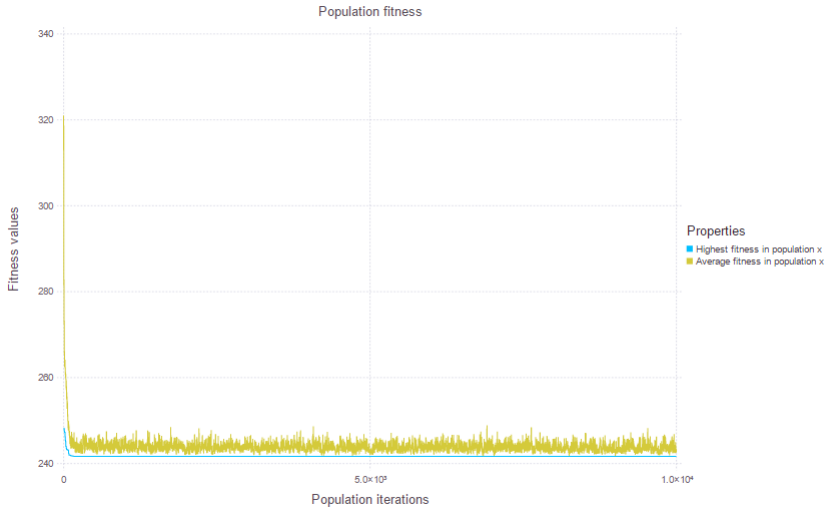

Figure C.16: Search process for GA with crowding.

Appendix D

Julia source code

D.1 Basic genetic algorithm

GA

June 29, 2017

In [ ]:
```
#Input stream data

#reading stream data is time consuming
#Therefore, placed in separate cell in order to load values from excel only once
#OBS! "using xxxx" cannot be placed inside functions, hence call functions globally

using JuMP    #use of package JuMP to formulate optimization problem
using CPLEX   #use of package CPLEX to solve the LP problem

#[IMPORTANT]
#the algorithm is designed so that pressure changing streams must be listed first
#do not distinguish between hot and cold, can be found by comparing TS and TT
#the mapping is done such that the stream on location x in vector TS..
#..has corresponding values at location x in TT, mCp etc.

using ExcelReaders

f = openxl("Input_streamdata.xlsx")

#remember to update cell G2
nPr=readxl(f, "Streamdata!G2:G2"); nPr=Int(nPr[1]);          #no. of dynamic streams
nSt=readxl(f, "Streamdata!I2:I2"); nSt=Int(nSt[1]);          #no. of static streams

#remember to update cellnumbers
TS=readxl(f, "Streamdata!C5:C10");                           #supply temp
TT=readxl(f, "Streamdata!D5:D10");                           #target temp
Pr=readxl(f, "Streamdata!G5:G7");                            #pressure ratio
mCp=readxl(f, "Streamdata!H5:H10");                          #flow rate capacity

#remember to update values for below ambient cases
DTmin=readxl(f, "Streamdata!A2:A2"); DTmin=Int(DTmin[1]);    #delta T min
Tamb=readxl(f, "Streamdata!B2:B2");  Tamb=Float64(Tamb[1]); #ambient temp
Thu=readxl(f, "Streamdata!C2:C2");   Thu=Float64(Thu[1]);    #hot utility temp
Tcu=readxl(f, "Streamdata!D2:D2");   Tcu=Float64(Tcu[1]);    #cold utility temp
=readxl(f, "Streamdata!E2:E2");     =Float64([1]);       #kappa
=readxl(f, "Streamdata!F2:F2");     =Float64([1]);       #polytropic efficiency
n=readxl(f, "Streamdata!H2:H2");     n=Int(n[1]);           #number of stream splits
T=273.15;                                                    #kelvin temperature
```

In [ ]:
```
#decoding of genes from genotype to phenotype representation

function genotypeMap(genPop, paraNum, ub, lb)

popSize = length(genPop[:,1]);      #population size equals number of chromosomes
phenPop = zeros(popSize, paraNum);  #empty population in phenotype representation
stepSize = zeros(Float64, paraNum); #discretization of variable range

#calculate the resolution for each gene
for i=1:paraNum
    stepSize[i] = (ub-lb)/((2^nBits)-1); #evenly distr density of variable range
end

#for all individuals calculate the phenotype representation of each gene
for i = 1:popSize
    k = 1;
    for j = 1:paraNum
        phenPop[i,j] = (parse(Int64, genPop[i][k:k+nBits-1], 2))*stepSize[j]+lb
        k = k+nBits
    end
end
return phenPop                        #return a population of phenotype values
end
```

In [ ]:
```
#fitness function/the innter loop

function computeFitness(phenPop)

#output values for comparison of solutions

exergy = zeros(length(phenPop[:,1]));       #fitness value of individuals in pop p
Uhot = zeros(length(phenPop[:,1]));         #hot utility of indiv solutions in pop p
Ucold = zeros(length(phenPop[:,1]));        #cold utility of indiv solutions in pop p
mfr = zeros(nPr, n, length(phenPop[:,1])); #mCp of each stream split in pop p
TIN = zeros(nPr, n, length(phenPop[:,1])); #input temp to pressure changing units
TOUT = zeros(nPr, n, length(phenPop[:,1])); #output temp to pressure changing units
Work = zeros(nPr, n, length(phenPop[:,1])); #work from each pressure changing unit

    #------------------------start preprocessing step-----------------------

    #for each individual solution in population p we calculate the fitness value
    for i=1:length(phenPop[:,1])

        #extrac info from genes and calculate pressure ratio and output temperatures
        Tin = collect(reshape(phenPop[i,:], nPr, n));

        PR = []; Tout = zeros(nPr, n);
```

```
PR = collect((Pr[s])^((-1)/) for s=1:length(Pr));

for s = 1:nPr
    for b = 1:n
        Tout[s,b] = ((Tin[s,b]+273.15)/(PR[s]))-273.15;
    end
end


#[split operator]
#generate new streams; one segment before unit, one after unit
#[OBS!] pressure changing streams must be listed first in stream data

TSnew = zeros(2*nPr*n);
TTnew = zeros(2*nPr*n);
mCpnew = zeros(2*nPr*n);
W = zeros(nPr, n);

k = 1;
for s = 1:nPr
    for b = 1:n
        TSnew[k] = TS[s];              #supply temp of segment before unit
        TSnew[k+1] = Tout[s,b];        #supply temp of segment after unit
        TTnew[k] = Tin[s,b];           #target temp of segment before unit
        TTnew[k+1] = TT[s];            #target temp of segment after unit
        mCpnew[k] = mCp[s];            #mCp of segment before unit
        mCpnew[k+1] = mCp[s];          #mCp of segment after unit
        k = k+2;

        W[s,b] = (Tout[s,b]-Tin[s,b])*mCp[s]; #work in each unit
    end
end

#for later simplicity we add the static streams to the new vectors

append!(TSnew, TS[nPr+1:end]);
append!(TTnew, TT[nPr+1:end]);
append!(mCpnew, mCp[nPr+1:end]);

#generate the heat cascade:
#1. the cold temperature side
#2. the hot temperature side
#3. number of temperature intervals

Tcold = []; Thot = []; K = [];
for j = 1:length(TSnew)                #create heat casc from all supply temp
    if TSnew[j] >= TTnew[j]            #hot stream
       push!(Tcold, TSnew[j]-DTmin);   #adjust for delta Tmin
```

```
            else push!(Tcold, TSnew[j])        #cold stream
            end
        end

        #to create the heat cascade we need to add the max/min temps in the system
        push!(Tcold, min(Tamb,Tcu));

        Tcold = reverse(sort(union(Tcold)));  #sort the cold temp side
        Thot = Tcold+DTmin;                   #create hot side
        K = length(Tcold);                    #no. of temp intervals

        #for testing/debugging of function:
        #println("Tin:  ", Tin)
        #println("Tout:  ", Tout)
        #println("TSnew:  ", TSnew)
        #println("TTnew:  ", TTnew)
        #println("mCpnew:  ", mCpnew)
        #println("Work:  ", W)
        #println("Tcold:  ", Tcold)
        #println("Thot:  ", Thot)
        #println("K:  ", K)

        #for simplicity, the heat in each interval from the static streams..
        #..and the pressure changing streams are calculated separately.

        #STATAIC STREAMS
        #heat from hot and cold streams to interval k in K

        QS = zeros(nSt, K); q = 1;

for s = nPr+1:length(TS)
    if TS[s] > TT[s]

    for k = 1:K-1
        if (TT[s]<=Thot[k+1]||(TT[s]<=Thot[k]&&TT[s]>=Thot[k+1]))&&TS[s]>=Thot[k]
            QS[q,k] = mCp[s]*(Thot[k]- max(Thot[k+1], TT[s]))
        end
    end


    elseif TS[s] < TT[s]

    for k = 1:K-1
        if TS[s]<=Tcold[k+1]&&(TT[s]>=Tcold[k]||(TT[s]<=Tcold[k]&&TT[s]>=Tcold[k+1]))
            QS[q,k] = mCp[s]*(Tcold[k+1] - min(Tcold[k], TT[s]))
        end
    end
```

```
end
q = q+1;
end

        #for simplicity, sum heat in each interval

        Qksum = zeros(K)
        for k = 1:K
            Qksum[k] = sum(QS[1:end,k])
        end


        #PRESSURE CHANGING STREAMS
        #heat from pressure changing streams depends on the variable mass fraction
        #we calculate the heat supply/demand beforehand -
        #and multiply with a fraction variable 0<=f<=1 in the optimization model

        #heat from hot and cold streams to interval k in K

        QP = zeros(2*nPr*n, K); p = 1;

for s = 1:2*nPr*n

if TSnew[s] > TTnew[s]

for k = 1:K-1
if (TTnew[s]<=Thot[k+1]||
    (TTnew[s]<=Thot[k]&&TTnew[s]>=Thot[k+1]))&&TSnew[s]>=Thot[k]
     QP[p,k] = mCpnew[s]*(Thot[k]- max(Thot[k+1], TTnew[s]));
end
end


elseif TSnew[s] < TTnew[s]

for k = 1:K-1
if TSnew[s]<=Tcold[k+1]&&(TTnew[s]>=Tcold[k]||
    (TTnew[s]<=Tcold[k]&&TTnew[s]>=Tcold[k+1]))
    QP[p,k] = mCpnew[s]*(Tcold[k+1] - min(Tcold[k], TTnew[s]));
end
end

end
p = p+1;
end

        #heat content of pressure changing stream segments are calculated above
        #these are listed in the vector QP[p,k]
```

```
#p is the no. of stream segments and k is the no. of temp intervals
#segments belonging to the same branch has the same flow fraction, f
#hence, the array is reshaped to the form QP[z,n,nPr,k]
#for each interval k -
#sum for all pressure changing streams and all branches
#ex. sum(QP[z,1,1,1]*f[1,1] for z = {1,2}, s = 1, b = 1, k = 1)

#reshape QP so to be compatible with f[s,b]

Z = 2;
QPr = reshape(QP, Z, n, nPr, K);

#-------------------------end preprocessing step-------------------------

#LP-model

HeatCasc=Model(solver=CplexSolver())

@variable(HeatCasc, r[1:K] >= 0);
@variable(HeatCasc, Qhu >= 0);
@variable(HeatCasc, 0 <= f[s = 1:nPr, b = 1:n] <= 1);

@objective(HeatCasc, Min,
            (1-(Tamb+T)/(Thu+T))*Qhu + ((Tamb+T)/(Tcu+T)-1)*r[K-1] +
            sum(f[s,b]*W[s,b] for s=1:nPr, b=1:n));


@constraint(HeatCasc, FirstInt[k = 1],
            r[k] - Qhu ==
            sum(QPr[z,b,s,k]*f[s,b] for s=1:nPr, b=1:n, z=1:Z) + Qksum[k]);

@constraint(HeatCasc, ResInt[k = 2:K],
            r[k] - r[k-1] ==
            sum(QPr[z,b,s,k]*f[s,b] for s=1:nPr, b=1:n, z=1:Z) + Qksum[k] );

@constraint(HeatCasc, frac[s = 1:nPr], sum(f[s,b] for b = 1:n) == 1);

#avoid print put from the optimization
WW = STDOUT;
redirect_stdout();
solve(HeatCasc);
redirect_stdout(WW);

exergy[i] = getobjectivevalue(HeatCasc);
Uhot[i] = getvalue(Qhu);
Ucold[i] = getvalue(r[K-1]);
for s = 1:nPr
    for  b = 1:n
```

```
                    mfr[s,b,i] = getvalue(f[s,b]);
                    Work[s, b, i] = W[s, b];
                    TIN[s, b, i] = Tin[s, b];
                    TOUT[s, b, i] = Tout[s, b];
                end
            end

            #for testing/debugging of function:
            #println("exergy:    ",exergy)
            #println("Qhu:       ",Uhot)
            #println("mfr:       ",mfr)
            #println("Work:      ",Work)
            #println("Tin:       ",TIN)
            #println("Tout:      ",TOUT)

        end

            indEx = indmin(exergy);
            bestfit_Uhot = Uhot[indEx];
            bestfit_work = collect(Work[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_TIN = collect(TIN[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_TOUT = collect(TOUT[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_mfr = collect(mfr[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);

        return exergy, Uhot, Work, TIN, TOUT, mfr; #return solution and fitness measure
    end
```

In [ ]:
```
    #elitist selection
    #the very best individuals directly survive to the next generation

    function elitistSel(fitness, )

        popSize = length(fitness[:,1]);    #population size
        M = maximum(fitness) + 100;        #avoid choosing the same individual
        n = Int(round(popSize*));         #number of "free survivors"
        fitVal = collect(fitness);

        E = zeros(Int, n)
        for i=1:n
            bestfit=indmin(fitVal);        #returns a tuple of (val,indx)
            E[i] = bestfit;                #save the position of the best individual
            fitVal[bestfit] = M;           #remove best value to find second best one
        end
        return E
    end

    #how easily find the index of the n maximum values of a vector?!
    #should be a function for this in julia, but cannot find any
```

In [ ]:
```
#tournament selection

function tournamentSel(tourSize, fitness, )

  popSize = length(fitness[:,1]);
  indxPool = [];                          #indexes of winners from the tournament
  n = Int(round(popSize*));              #number of "free survivors"

  for i = 1:(popSize-n)                  #number of parents, , to be selected
    indx = rand(1:popSize, tourSize);    #randomly choose a sub group of the pop
    tourfit = fitness[indx];             #assign fitness measure
    winner = indmin(tourfit);            #choose the fittest
    push!(indxPool, indx[winner]);       #the indiv with highest fitness selected
  end
  return indxPool                        #return the mating pool of indx values

end
```

In [ ]:
```
#single point crossover operator

function crossSinglePoint(selPop, )

popSize = length(selPop);                #population size
selVec = collect(1:popSize);             #selection vector
crossPop = Array{String}(popSize);       #population of offsprings

i = 1
while i <= popSize-1

      indxA = rand(1:length(selVec));    #random parent chosen from mating pool
      parentA = selPop[selVec[indxA]];   #parent subject to crossover only once
      deleteat!(selVec, indxA);

     if rand() <=                        #probability of crossover
        indxB = rand(1:length(selVec));
        parentB = selPop[selVec[indxB]]; #second parent chosen from mating pool
        deleteat!(selVec, indxB);        #parent subject to crossover only once

            #location of pointer
            pointer = rand(1:length(parentA));

            #generate new individuals from parents
            childA = parentA[1:pointer] * parentB[pointer+1:end];
            childB = parentB[1:pointer] * parentA[pointer+1:end];

            #place new individuals in new population
            crossPop[i] = childA
            crossPop[i+1] = childB
```

```
                i = i+2;
            else crossPop[i] = parentA;          #parent survive
                 i = i+1;
            end
        end

        if isempty(selVec) == false
           crossPop[popSize] = selPop[selVec[1]];
        end

        return crossPop                          #return new population
        end
```

In [ ]:
```
        #uniform crossover operator

        function uniformCrossover(selPop, )


        popSize = length(selPop);                #population size
        selVec = collect(1:popSize);             #selection vector
        crossPop = Array{String}(popSize);       #population of offsprings

        i = 1
        while i <= popSize-1

              indxA = rand(1:length(selVec));    #random parent chosen from mating pool
              parentA = selPop[selVec[indxA]];   #parent subject to crossover only once
              deleteat!(selVec, indxA);

             if rand() <=                        #probability of crossover
                indxB = rand(1:length(selVec));
                parentB = selPop[selVec[indxB]]; #second parent chosen from mating pool
                deleteat!(selVec, indxB);        #parent subject to crossover only once

                    #uniform crossover mask
                    mask = join(rand(0:1, nBits*nPr*n));

                    #generate new individuals from parents
                    newbornA = zeros(Int, length(mask));
                    newbornB = zeros(Int, length(mask));

                    for a = 1:length(mask)
                        if parse(Int, mask[a]) == 1
                            newbornA[a] = parse(Int, parentA[a]);
                            newbornB[a] = parse(Int, parentB[a]);

                        elseif parse(Int, mask[a]) == 0
                            newbornA[a] = parse(Int, parentB[a]);
```

```
                        newbornB[a] = parse(Int, parentA[a]);
                    end
                end

                childA = join(newbornA);
                childB = join(newbornB);

                #place new individuals in new population
                crossPop[i] = childA
                crossPop[i+1] = childB
                i = i+2;
            else crossPop[i] = parentA;          #parent survive
                i = i+1;
            end
        end

        if isempty(selVec) == false
           crossPop[popSize] = selPop[selVec[1]];
        end

        return crossPop                          #return new population
        end
```

In [ ]:
```
        #single bitswap mutation operator

        function mutationSingleBitswap(crossPop, )

        popSize = length(crossPop);      #population size
        chromSize = length(crossPop[1]); #chromosome size

        for i = 1:popSize

            if rand() <=                 #mutation probability

               flip = rand(1:chromSize); #flip position

               newGen = zeros(Int, chromSize);
               for j = 1:chromSize
                   if j == flip
                      if parse(Int, crossPop[i][j]) == 0
                         newGen[j] = 1;
                      elseif parse(Int, crossPop[i][j]) == 1
                         newGen[j] = 0;
                      end
                   else
                        newGen[j] = parse(Int, crossPop[i][j]);
                   end
            end
```

```
            crossPop[i] = join(newGen);

        #else println("no mutation")
        end
    end

    return crossPop                  #return mutated population
    end
```

In [ ]:
```
#flipping bitswap mutation operator

function mutationFlippingBitswap(crossPop, )

popSize = length(crossPop);              #population size
chromSize = length(crossPop[1]);         #chromosome size

for i = 1:popSize

    if rand() <=                        #mutation probability

       mutChrom = rand(0:1, chromSize); #mutation chromosome
       newGen = zeros(Int, chromSize);
       for j = 1:chromSize

           if mutChrom[j] == 1

              if parse(Int, crossPop[i][j]) == 0
                 newGen[j] = 1;
              elseif  parse(Int, crossPop[i][j]) == 1
                  newGen[j] = 0;
              end
           else
                newGen[j] = parse(Int, crossPop[i][j]);
           end
        end

           crossPop[i] = join(newGen);

    #else println("no mutation")
    end
end

return crossPop                          #return mutated population
end
```

In [ ]:
```
#transition to next generation
```

```
function nextGeneration(crossPop, Eindx, genPop)

    crossSize = length(crossPop);              #population size
    newGenPop = Array{String}(length(genPop)); #new population

    for i = 1:crossSize
        newGenPop[i] = crossPop[i];
    end

    k = 1
    for j = crossSize+1:popSize
        newGenPop[j] = genPop[Eindx[k]];
        k = k+1
    end

return newGenPop #return new population in binary representation
end
```

In [ ]:
```
#BGA MAIN FILE

tic()                   #start timing the computation

# ------------initialization of the GA-----------------------------------------
paraNum = nPr*n;        #no. of decision variables featuring one solution
lb = Tamb;              #lower bound on range for decision variables in paraNum
ub = Thu;               #upper bound on range for decision variables in paraNum
nBits = 11;             #number of bits needed to represent the variable range
                        #in binary numbers
 = 0.01;               #rate of elitism
 = 0.75;               #crossover probability
 = 0.10;               #mutation rate 1/(nBits*nPr*n)
tourSize = 2;           #tournament size
popSize = 20*paraNum;   #size of a population
genCount = 10000;       #generation counter
#-------------------------------------------------------------------------------

#initial population in chromosome representation

#genPop -> [Tin,sb] -> [Tin,11 Tin,21 Tin,12 Tin,22 Tin,13 Tin,23 for s=2, n=3]
genPop = Array{String}(popSize)
seed = 123;
for i = 1:popSize
    srand(seed);
    genPop[i] = join(rand(0:1, nBits*nPr*n));
    seed = seed + i;
end

bestfit = [];
```

```
averagefit = [];

#------------------------------start outer loop---------------------------------

for m = 1:genCount

    #mapping of genotype to phenotype
    phenPop = genotypeMap(genPop, paraNum, ub, lb);

    #-------------------------inner loop-----------------------------------

    #compute fitness value of the individuals
    fitness = computeFitness(phenPop);

    #----------------------------------------------------------------------

    #best solution found for this population
    push!(bestfit, minimum(fitness[1]));
    push!(averagefit, sum(fitness[1]/length(fitness[1])));

    #selection
    #directly winners to next generation
    Eindx = elitistSel(fitness[1], );

    #mating pool with higher average fitness values
    matPool = tournamentSel(tourSize, fitness[1], );

    #binary representation of selected individuals
    selPop = genPop[matPool];

    #crossover
    crossPop = uniformCrossover(selPop, );

    #mutation
    crossPop = mutationFlippingBitswap(crossPop, );

    #transition to next generation
    genPop = nextGeneration(crossPop, Eindx, genPop);

    #for testing/debugging of function:
    #println("fitness:    ",fitness)
    #println("winners:    ",winners)
    #println("selPop:     ",selPop)
    #println("Elitism:    ",E)
    #println("crossPop:   ", crossPop)
    #println("genPop:     ",genPop)
    #println("iteration:  ", m)
```

```
        if m == genCount

        #for testing/debugging of function:
        #println("exergy:  ", fitness[1])
        #println("hot utility:  ", fitness[2])
        #println("mass flow fraction:  ", fitness[6])
        #println("work:  ", fitness[3])
        #println("Tin:  ", fitness[4])
        #println("Tout:  ", fitness[5])


        bf = minimum(fitness[1]);
        indVec = [];
        for i = 1:length(fitness[1])
            if fitness[1][i] == bf;
               push!(indVec, i);
            end
        end

        numSol = length(indVec);
        println("                                                                    ")
        println("Results from SGA last iteration m = ",m)
        println("                                                                    ")
        println("number of optimal solutions found:  ",numSol)
        println("____________________________________________________________________")
        println("                                                                    ")
        for i = 1:length(indVec)
            println("Solution $i:                                                    ")
            println("                                                                ")
            println("Exergy value:                ", fitness[1][indVec[i]])
            println("Hot utility:                 ", fitness[2][indVec[i]])
            println("Work:                        ", fitness[3][:,:,indVec[i]])
            println("Fraction of mass flow rate:  ", fitness[6][:,:,indVec[i]])
            println("Input temperature to unit:   ", fitness[4][:,:,indVec[i]])
            println("Output temperature from unit: ", fitness[5][:,:,indVec[i]])
            println("____________________________________________________________________")
            println("                                                                    ")
            println("                                                                    ")
        end
    end

end

#------------------------------end outer loop-----------------------------------

toc() #end timing
```

In [ ]:
```
#plot average population fitness and best fitness value throughout the search
```

```
using Gadfly   #for plotting
plot(x=1:genCount, y=bestfit, Geom.line)

using DataFrames

xs=1:genCount;

df_bestfit=DataFrame(x=xs, y=bestfit,
    Properties="Highest fitness in population x")
df_averagefit=DataFrame(x=xs,y=averagefit,
    Properties="Average fitness in population x")
df=vcat(df_bestfit, df_averagefit)
p=plot(df, x=:x, y=:y, color=:Properties, Geom.line,
      Guide.title("Population fitness"),
      Guide.xlabel("Population iterations"), Guide.ylabel("Fitness values") )

draw(SVGJS(25cm, 16cm), p)
```

0.1

D.2 Genetic algorithm with crowding

CrowdingGA

June 29, 2017

In [ ]:
```
#Input stream data

#reading stream data is time consuming
#Therefore, placed in separate cell in order to load values from excel only once
#OBS! "using xxxx" cannot be placed inside functions, hence call functions globally

using JuMP       #use of package JuMP to formulate optimization problem
using CPLEX      #use of package CPLEX to solve the LP problem
using Distances  #use package to calculate hamming distances

#[IMPORTANT]
#the algorithm is designed so that pressure changing streams must be listed first
#do not distinguish between hot and cold, can be found by comparing TS and TT
#the mapping is done such that the stream on location x in vector TS..
#..has corresponding values at location x in TT, mCp etc.

using ExcelReaders

f = openxl("Input_streamdata.xlsx")

#remember to update cell G2
nPr=readxl(f, "Streamdata!G2:G2"); nPr=Int(nPr[1]);          #no. of dynamic streams
nSt=readxl(f, "Streamdata!I2:I2"); nSt=Int(nSt[1]);          #no. of static streams

#remember to update cellnumbers
TS=readxl(f, "Streamdata!C5:C10");                           #supply temp
TT=readxl(f, "Streamdata!D5:D10");                           #target temp
Pr=readxl(f, "Streamdata!G5:G7");                            #pressure ratio
mCp=readxl(f, "Streamdata!H5:H10");                          #flow rate capacity

#remember to update values for below ambient cases
DTmin=readxl(f, "Streamdata!A2:A2"); DTmin=Int(DTmin[1]);    #delta T min
Tamb=readxl(f, "Streamdata!B2:B2");  Tamb=Float64(Tamb[1]); #ambient temp
Thu=readxl(f, "Streamdata!C2:C2");   Thu=Float64(Thu[1]);    #hot utility temp
Tcu=readxl(f, "Streamdata!D2:D2");   Tcu=Float64(Tcu[1]);    #cold utility temp
=readxl(f, "Streamdata!E2:E2");      =Float64([1]);        #kappa
=readxl(f, "Streamdata!F2:F2");      =Float64([1]);        #polytropic efficiency
```

```
n=readxl(f, "Streamdata!H2:H2");     n=Int(n[1]);          #number of stream splits
T=273.15;                                                  #kelvin temperature
```

In [ ]:
```
#mapping of genes from genotype to phenotype

function genMapping(genPop, paraNum, ub, lb)

popSize = length(genPop);                #population size
phenPop = zeros(popSize, paraNum);       #phenotype representation of population
stepSize=zeros(Float64, paraNum);        #discretization of variable range

for i=1:paraNum
    stepSize[i] = (ub-lb)/((2^nBits)-1); #evenly distr density of variable range
end

for i = 1:popSize
    k = 1;
    for j = 1:paraNum
        phenPop[i,j] = (parse(Int64, genPop[i][k:k+nBits-1], 2))*stepSize[j]+lb
        k = k+nBits
    end
end

return phenPop                           #return a population of phenotype values
end
```

In [ ]:
```
#fitness function/inner loop

function computeFitness(phenPop)

#output values to comparison of solutions

exergy = zeros(length(phenPop[:,1]));       #fitness value of individuals in pop p
Uhot = zeros(length(phenPop[:,1]));         #hot utility of indiv solutions in pop p
Ucold = zeros(length(phenPop[:,1]));        #cold utility of indiv solutions in pop p
mfr = zeros(nPr, n, length(phenPop[:,1])); #mCp of each stream split in pop p
TIN = zeros(nPr, n, length(phenPop[:,1])); #input temp to pressure changing units
TOUT = zeros(nPr, n, length(phenPop[:,1])); #output temp to pressure changing units
Work = zeros(nPr, n, length(phenPop[:,1])); #work from each pressure changing unit


    #for each individual solution in population p we calculate the fitness value
    for i=1:length(phenPop[:,1])

        #extract info from each gene and calculate pressure ratio and output temp
        Tin = collect(reshape(phenPop[i,:], nPr, n));

        PR = []; Tout = zeros(nPr, n);
```

```
PR = collect((Pr[s])^((-1)/) for s=1:length(Pr));

for s = 1:nPr
    for b = 1:n
        Tout[s,b] = ((Tin[s,b]+273.15)/(PR[s]))-273.15;
    end
end

#[split operator]
#generate new streams; one segment before unit, one after unit
#[OBS!] pressure changing streams must be listed first in stream data

TSnew = zeros(2*nPr*n);
TTnew = zeros(2*nPr*n);
mCpnew = zeros(2*nPr*n);
W = zeros(nPr, n);

k = 1;
for s = 1:nPr
    for b = 1:n
        TSnew[k] = TS[s];              #supply temp of segment before unit
        TSnew[k+1] = Tout[s,b];        #supply temp of segment after unit
        TTnew[k] = Tin[s,b];           #target temp of segment before unit
        TTnew[k+1] = TT[s];            #target temp of segment after unit
        mCpnew[k] = mCp[s];            #mCp of segment before unit
        mCpnew[k+1] = mCp[s];          #mCp of segment after unit
        k = k+2;

        W[s,b] = (Tout[s,b]-Tin[s,b])*mCp[s]; #work from each unit
    end
end

#for later simplicity we add the static streams to the new vectors

append!(TSnew, TS[nPr+1:end]);
append!(TTnew, TT[nPr+1:end]);
append!(mCpnew, mCp[nPr+1:end]);

#generate the heat cascade:
#1. the cold temperature side
#2. the hot temperature side
#3. number of temperature intervals

Tcold = []; Thot = []; K = [];
for j = 1:length(TSnew)                #create heat casc from all supply temp
    if TSnew[j] >= TTnew[j]            #hot stream
       push!(Tcold, TSnew[j]-DTmin);   #adjust for delta Tmin
```

```
            else push!(Tcold, TSnew[j])        #cold stream
            end
        end

        #to create the heat cascade we need to add the max/min temps in the system
        push!(Tcold, min(Tamb,Tcu));

        Tcold = reverse(sort(union(Tcold)));  #sort the cold temp side
        Thot = Tcold+DTmin;                   #create hot side of heat casc
        K = length(Tcold);                    #no of temp intervals in heat casc

        #for testing/debugging of function:
        #println("Tin:  ", Tin)
        #println("Tout:  ", Tout)
        #println("TSnew:  ", TSnew)
        #println("TTnew:  ", TTnew)
        #println("mCpnew:  ", mCpnew)
        #println("Work:  ", W)
        #println("Tcold:  ", Tcold)
        #println("Thot:  ", Thot)
        #println("K:  ", K)

         #for simplicity, the heat in each interval from the static streams..
        #..and the pressure changing streams are calculated separately.

        #STATAIC STREAMS
        #heat from hot and cold streams to interval k in K

        QS = zeros(nSt, K); q = 1;

for s = nPr+1:length(TS)
    if TS[s] > TT[s]

    for k = 1:K-1
    if (TT[s]<=Thot[k+1]||(TT[s]<=Thot[k]&&TT[s]>=Thot[k+1]))&&TS[s]>=Thot[k]
    QS[q,k] = mCp[s]*(Thot[k]- max(Thot[k+1], TT[s]))
    end
    end


    elseif TS[s] < TT[s]

    for k = 1:K-1
    if TS[s]<=Tcold[k+1] && ( TT[s]>=Tcold[k] || ( TT[s]<=Tcold[k] && TT[s]>=Tcold[k+1]
        QS[q,k] = mCp[s]*(Tcold[k+1] - min(Tcold[k], TT[s]))
    end
    end
```

```
    end
    q = q+1;
end

        #for simplicity, sum of heat in each interval for the static streams

        Qksum = zeros(K)
        for k = 1:K
            Qksum[k] = sum(QS[1:end,k])
        end

        #PRESSURE CHANGING STREAMS
        #heat from pressure changing streams depends on the variable mass fraction
        #we calculate the heat supply/demand beforehand -
        #and multiply with a fraction variable 0<=f<=1 in the optimization model

        #heat from hot and cold streams to interval k in K

        QP = zeros(2*nPr*n, K); p = 1;

for s = 1:2*nPr*n

if TSnew[s] > TTnew[s]

for k = 1:K-1
if (TTnew[s]<=Thot[k+1]||(TTnew[s]<=Thot[k]&&TTnew[s]>=Thot[k+1]))&&TSnew[s]>=Thot[k]
    QP[p,k] = mCpnew[s]*(Thot[k]- max(Thot[k+1], TTnew[s]));
end
end


elseif TSnew[s] < TTnew[s]

for k = 1:K-1
if TSnew[s]<=Tcold[k+1]&&(TTnew[s]>=Tcold[k]||
            (TTnew[s]<=Tcold[k]&&TTnew[s]>=Tcold[k+1]))
    QP[p,k] = mCpnew[s]*(Tcold[k+1] - min(Tcold[k], TTnew[s]));
end
end

end
p = p+1;
end

        #heat content of pressure changing stream segments are calculated above..
        #..and listed in a vector QP[p,k].
        #p is the no. of stream segments and k is the no. of temp intervals.
        #Segments belonging to the same branch has the same flow fraction, f,..
```

```
#..hence the array must be reshaped to the form QP[z,n,nPr,k].
#For each interval k,..
#..sum for all pressure changing streams and for all branches of these streams
#ex. sum(QP[z,1,1,1]*f[1,1] for z = {1,2}, s = 1, b = 1, k = 1)

#reshape QP so to be compatible with f[s,b]
Z = 2;
QPr = reshape(QP, Z, n, nPr, K);

#LP-model
HeatCasc=Model(solver=CplexSolver())
@variable(HeatCasc, r[1:K] >= 0);
@variable(HeatCasc, Qhu >= 0);
@variable(HeatCasc, 0 <= f[s = 1:nPr, b = 1:n] <= 1);

@objective(HeatCasc, Min,
        (1-(Tamb+T)/(Thu+T))*Qhu + ((Tamb+T)/(Tcu+T)-1)*r[K-1] +
        sum(f[s,b]*W[s,b] for s=1:nPr, b=1:n));


@constraint(HeatCasc, FirstInt[k = 1],
            r[k] - Qhu ==
            sum(QPr[z,b,s,k]*f[s,b] for s=1:nPr, b=1:n, z=1:Z) + Qksum[k] );

@constraint(HeatCasc, ResInt[k = 2:K],
            r[k] - r[k-1] ==
            sum(QPr[z,b,s,k]*f[s,b] for s=1:nPr, b=1:n, z=1:Z) + Qksum[k] );

@constraint(HeatCasc, frac[s = 1:nPr], sum(f[s,b] for b = 1:n) == 1);


#to avoid print put from the optimization
WW = STDOUT;
redirect_stdout();
solve(HeatCasc);
redirect_stdout(WW);

exergy[i] = getobjectivevalue(HeatCasc);
Uhot[i] = getvalue(Qhu);
Ucold[i] = getvalue(r[K-1]);
for s = 1:nPr
    for  b = 1:n
         mfr[s,b,i] = getvalue(f[s,b]);
         Work[s, b, i] = W[s, b];
         TIN[s, b, i] = Tin[s, b];
         TOUT[s, b, i] = Tout[s, b];
    end
end
```

```
            #for testing/debugging of function:
            #println("exergy:    ",exergy)
            #println("Qhu:     ", Uhot)
            #println("mfr:     ", mfr)
            #println("Work:   ", Work)
            #println("Tin:    ", TIN)
            #println("Tout:    ", TOUT)

        end

            indEx = indmin(exergy);
            bestfit_Uhot = Uhot[indEx];
            bestfit_work = collect(Work[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_TIN = collect(TIN[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_TOUT = collect(TOUT[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);
            bestfit_mfr = collect(mfr[s,b,i] for s = 1:nPr, b = 1:n, i = indEx);

        return exergy, Uhot, Work, TIN, TOUT, mfr; #return solution and fitness measure
    end
```

In [ ]:
```
#elitist selection
#the very best individuals directly survive to the next generation

function elitistSel(fitness, )

    popSize = length(fitness[:,1]);    #population size
    M = maximum(fitness) + 100;        #avoid choosing the same individual
    n = Int(round(popSize*));         #number of "free survivors"
    fitVal = collect(fitness);

    E = zeros(Int, n)
    for i=1:n
        bestfit=indmin(fitVal);        #returns a tuple of (val,indx)
        E[i] = bestfit;                #save the position of the best individual
        fitVal[bestfit] = M;           #remove best value to find second best one
    end
    return E
end

#how easily find the index of the n maximum values of a vector?!
#should be a function for this in julia, but cannot find any
```

In [ ]:
```
#pairing of parent individuals

function crowdingPairing(genPop)

pairSize = Int(length(genPop)/2);
```

```
selVec = collect(1:length(genPop));

#pairing of parents
parents = Array{Tuple}(pairSize);
for i = 1:pairSize

    indxA = rand(1:length(selVec));
    parentA = selVec[indxA];
    deleteat!(selVec, indxA);

    indxB = rand(1:length(selVec));
    parentB = selVec[indxB];
    deleteat!(selVec, indxB);

   parents[i] = (parentA, parentB);

end

return parents  #return population of paired parent individuals
end
```

In [ ]:
```
#uniform crossover

function crowdingCrossover(parents, genPop, )

pairSize = length(parents);                 #number of pairs
crossVec = zeros(Int, pairSize);

for i = 1:pairSize
    if rand() <=
        crossVec[i] = 1;
    end
end

offsprings = Array{Tuple}(sum(crossVec));

j = 1;
for i = 1:pairSize

    if crossVec[i] == 1                     #crossover probability

       parentA = genPop[parents[i][1]];     #random chosen individuals
       parentB = genPop[parents[i][2]];     #random chosen individuals

       mask = rand(0:1, length(genPop[1])); #uniform crossover mask

       newbornA = zeros(Int, length(mask)); #generate new individuals
       newbornB = zeros(Int, length(mask)); #generate new individuals
```

```
            for a = 1:length(mask)

                if mask[a] == 1

                   newbornA[a] = parse(Int, parentA[a]);
                   newbornB[a] = parse(Int, parentB[a]);

                elseif mask[a] == 0

                       newbornA[a] = parse(Int, parentB[a]);
                       newbornB[a] = parse(Int, parentA[a]);
                end
            end

            childA = join(newbornA);
            childB = join(newbornB);

            #create new individuals from parents
            offsprings[j] = (childA, childB);
            j = j+1;
           end
        end

        par = Array{Tuple}(length(offsprings));

        j = 1;
        for i = 1:length(crossVec)
            if crossVec[i] == 1
                par[j] = parents[i];
                j = j+1;
            end
        end

        return offsprings, par   #return tuple of offsprings
        end
```

In [ ]:
```
#mutation of offsprings

function crowdingMut(offsprings, )

pairSize = length(offsprings);          #number of pairs
chromSize = length(offsprings[1][1]);  #size of chromosomes

for i = 1:pairSize

    newGenes = zeros(Int, 2, chromSize);
    for j = 1:2
```

```
            if rand() <=                    #mutation probability

               mutChrom = rand(0:1, chromSize);

               for k = 1:chromSize

                   if mutChrom[k] == 1

                      if parse(Int, offsprings[i][j][k]) == 0;
                         newGenes[j,k] = 1;

                       elseif  parse(Int, offsprings[i][j][k]) == 1
                               newGenes[j,k] = 0;
                       end

                    else newGenes[j,k] = parse(Int, offsprings[i][j][k]);

                    end
                end


            else
                 for k = 1:chromSize
                     newGenes[j,k] = parse(Int, offsprings[i][j][k]);
                 end

            end
        end
        offsprings[i] = (join(newGenes[1,:]), join(newGenes[2,:]))
    end

    return offsprings #return mutated population
    end
```

In [ ]:
```
#record hamming distance between parent genes and offspring genes

function DISTANCE(par, genPop, offsprings)

distArr = zeros(Int, 2, 2, length(offsprings));

for i = 1:length(offsprings)
    for m = 1:2
        for n = 1:2
            parent = parse.(split(genPop[par[i][m]],""));
            child = parse.(split(offsprings[i][n],""));
            distArr[m,n,i] = evaluate(Hamming(), parent, child);
        end
    end
```

```
        end

        return distArr #return array of hamming distances
        end
```

In [ ]:
```
#match each offspring with the most similar parent

function MATCH(distArr, offsprings, par)

match = Array{Tuple}(length(offsprings));
for i = 1:length(offsprings)
    d1 = distArr[1,1,i] + distArr[2,2,i];
    d2 = distArr[1,2,i] + distArr[2,1,i];
    if d1 <= d2
         match[i] = (par[i][1], offsprings[i][1]), (par[i][2], offsprings[i][2])
    else match[i] = (par[i][1], offsprings[i][2]), (par[i][2], offsprings[i][1])
    end
end

genOff = Array{String}(length(offsprings)*2);
k = 1;
for i = 1:length(match)
    for j = 1:2
        genOff[k] = match[i][j][2]
        k = k+1
    end
end

return match, genOff   #return the matches
end
```

In [ ]:
```
#replacement; generalized replacement rule

function replacement(genPop, phenPop, fitness, genOff, phenOff, fitOff, match, , elit)

k = 0;
for i = 1:length(match)
    for j = 1:2

        k = k+1

        indxP = match[i][j][1];
        Pf = fitness[1][indxP];
        Cf = fitOff[1][k];

        if Cf < Pf
            Pc = Cf/(Cf + *Pf);
        elseif Cf == Pf
```

```
            Pc = 0.5;
        elseif Cf > Pf
            Pc = (*Cf)/(*Cf + Pf);
        end

        indx = indmax(fitness[1]);
        N = 0;
        for n = 1:length(elit)
            if indxP == elit[n]
               N = 1;
            end
        end

        if rand() <= Pc && (N != 1 || (N == 1 && indx == indxP))

           genPop[indxP] = genOff[k];
           phenPop[indxP,:] = phenOff[k,:];
           fitness[1][indxP] = fitOff[1][k];
           fitness[2][indxP] = fitOff[2][k];
           for f = 3:6
               fitness[f][:,:,indxP] = fitOff[f][:,:,k];
           end

        elseif N == 1 && indx != indxP
               genPop[indx] = genPop[indxP];
               phenPop[indx,:] = phenPop[indxP,:];
               fitness[1][indx] = fitness[1][indxP];
               fitness[2][indx] = fitness[2][indxP];
               for f = 3:6
                   fitness[f][:,:,indx] = fitness[f][:,:,indxP];
               end

               if rand() <= Pc
                  genPop[indxP] = genOff[k];
                  phenPop[indxP,:] = phenOff[k,:];
                  fitness[1][indxP] = fitOff[1][k];
                  fitness[2][indxP] = fitOff[2][k];
                  for f = 3:6
                      fitness[f][:,:,indxP] = fitOff[f][:,:,k];
                  end
               end
        end

    end
end


return genPop, phenPop, fitness
```

```
        end
```

In [ ]:
```
        #GA with crowding
        #MAIN FILE

        tic() #start timing

        # ------------initialization of the GA----------------------------------------
        paraNum = nPr*n;        #no. of decision variables featuring one solution
        lb = Tamb;              #lower bound on range for decision variables in paraNum
        ub = Thu;               #upper bound on range for decision variables in paraNum
        nBits = 11;             #number of bits needed to represent the variable range
                                #in binary numbers
         = 0.01;               #rate of elitism
         = 0.15;               #scaling factor
         = 0.10;               #mutation rate 1/(nBits*nPr*n)
         = 0.75;               #crossover probability
        popSize = 20*paraNum;   #size of a population must be an even number
        genCount = 10000;       #generation counter
        #-------------------------------------------------------------------------

        #initial population in chromosome representation
        genPop = Array{String}(popSize)
        seed = 123;
        for i = 1:popSize
            srand(seed);
            genPop[i] = join(rand(0:1, nBits*nPr*n));
            seed = seed + i;
        end

        #initial mapping to phenotype representation
        phenPop = genMapping(genPop, paraNum, ub, lb);

        #initial fitness evaluation
        fitness = computeFitness(phenPop);

        #------------------------------start outer loop---------------------------------
        bestfit = [];
        averagefit = [];
        for m = 1:genCount

            #print of found optimum
            push!(bestfit, minimum(fitness[1]));
            push!(averagefit, sum(fitness[1])/length(fitness[1]));

            #elitism
            elit = elitistSel(fitness[1], );
```

```
#----------------crowding--------------------------------
#random pairing of parents
parents = crowdingPairing(genPop);

#crossover: uniform crossover
outCross = crowdingCrossover(parents, genPop, );
offsprings = outCross[1];
par = outCross[2];

#mutation: Flipping based on mutation mask
mutOff = crowdingMut(offsprings, );

#calculate distance
distArr = DISTANCE(par, genPop, offsprings);

#matching of most similar parent/child
matching = MATCH(distArr, offsprings, par);
match = matching[1];
genOff = matching[2];

#assign fitness to offsprings
phenOff = genMapping(genOff, paraNum, ub, lb);

#--------------------------inner loop-----------------------------------
fitOff = computeFitness(phenOff);
#-----------------------------------------------------------------------


#---------Replacement-----------------------------------------------------
#local tournament
outRep = replacement(genPop,phenPop,fitness,genOff,phenOff,fitOff,match,,elit);
genPop = outRep[1];
phenPop = outRep[2];
fitness = outRep[3];
#-----------------------------------------------------------------------

#for testing/debugging of function:
#println("parents:      ",parents)
#println("offsprings:   ",offsprings)
#println("par:          ",par)
#println("mutOff:       ",mutOff)
#println("distArr:      ",distArr)
#println("match:        ",match)
#println("genOff:       ",genOff)
#println("phenOff:      ",phenOff)
#println("fitOff:       ",fitOff)
#println("iteration:  ", m)
```

```
    #print of found optimum
    #push!(bestfit, minimum(fitness[1]));
    #push!(averagefit, sum(fitness[1]/length(fitness[1])))

    if m == genCount

    #for testing/debugging of function:
    #println("exergy:  ", fitness[1])
    #println("mass flow fraction:  ", fitness[6])
    #println("hot utility:  ", fitness[2])
    #println("work:  ", fitness[3])
    #println("Tin:  ", fitness[4])
    #println("Tout:  ", fitness[5])


    bf = minimum(fitness[1]);
    indVec = [];
    for i = 1:length(fitness[1])
        if fitness[1][i] ==bf;
           push!(indVec, i);
        end
    end

    numSol = length(indVec);
    println("                                                                    ")
    println("Results from GA CROWDING last iteration m = ",m)
    println("                                                                    ")
    println("number of optimal solutions found:  ",numSol)
    println("____________________________________________________________________")
    println("                                                                    ")
    for i = 1:length(indVec)
        println("Solution $i:  ")
        println("                                                                ")
        println("Exergy value:                  ", fitness[1][indVec[i]])
        println("Hot utility:                   ", fitness[2][indVec[i]])
        println("Work:                          ", fitness[3][:,:,indVec[i]])
        println("Fraction of mass flow rate:    ", fitness[6][:,:,indVec[i]])
        println("Input temperature to unit:     ", fitness[4][:,:,indVec[i]])
        println("Output temperature from unit:  ", fitness[5][:,:,indVec[i]])
        println("____________________________________________________________")
        println("                                                                ")
        println("                                                                ")
    end
 end

end

#------------------------------end outer loop-----------------------------------
```

```
toc() #end timing


#printing of population evolution

#println("bestfit:  ",bestfit)
#println("averagefit:  ",averagefit)
#using Gadfly   #for plotting
#plot(x=1:genCount+1, y=bestfit, Geom.line)
#plot(x=1:genCount+1, y=averagefit, Geom.line)

#using DataFrames
#xs=1:genCount+1;
#df_bestfit = DataFrame( x=xs, y=bestfit, Properties="Highest fitness in population x"
#df_averagefit = DataFrame(x=xs, y=averagefit, Properties="Average fitness in populati
#df=vcat(df_bestfit, df_averagefit)
#p = plot(df, x=:x, y=:y, color=:Properties, Geom.line,
#      Guide.title("Population fitness"),
#      Guide.xlabel("Population iterations"), Guide.ylabel("fitness values") )

#draw(SVGJS(20cm, 12cm), p)
```

In [ ]:
```
#plot average population fitness and best fitness value throughout the search

using Gadfly   #for plotting
using DataFrames

xs=1:genCount;
df_bestfit = DataFrame( x=xs, y=bestfit,
    Properties="Highest fitness in population x")
df_averagefit = DataFrame(x=xs, y=averagefit,
    Properties="Average fitness in population x")
df=vcat(df_bestfit, df_averagefit)
p = plot(df, x=:x, y=:y, color=:Properties, Geom.line,
      Guide.title("Population fitness"),
      Guide.xlabel("Population iterations"), Guide.ylabel("Fitness values") )

draw(SVGJS(25cm, 16cm), p)
```

0.1

In [ ]:
```
```