TÉCNICO LISBOA

Boosting the performance of multi-objective epistasis detection

Francisco Sousa Lobo Theriaga Gonçalves

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Doctor Aleksandar Ilic

Doctor Leonel Augusto Pires Seabra de Sousa

Examination Committee

Chairperson: Doctor Teresa Maria Sá Ferreira Vazão Vasques

Supervisor: Doctor Aleksandar Ilic

Member of the committee: Doctor Nuno Cavaco Gomes Horta

June 2019

Declaração

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank my supervisors, Dr. Aleksandar Ilic, Dr. Leonel Sousa and Dr. Sergio Santander-Jiménez for their support, guidance and patience through this Thesis, where their helpful insights, ideas and solutions to my small problems were valuable for the development of the work here presented.

I would also like to thank all my friends, colleagues and family for their help through the entire course at IST.

Abstract

In recent years, studies have shown that genetic interaction can play a major role in the occurrence of diseases that affect human beings. An increasing number of approaches to detect these interactions have been developed, but due to the complexity of certain diseases, higher orders of genetic interaction need to be studied. Due to the size of the search space, fast and efficient approaches are necessary to identify high-order interactions. To tackle this problem, this Thesis proposes improvements over the multi-objective genetic algorithm, NSGA-II, and presents two parallelization designs of the proposed algorithm. Interaction orders up to 10 were tested, with three different problem instances, on an Intel Xeon Gold multicore multiprocessor system. Great improvements over all interaction orders was achieved, reaching up to 196×. In the parallel designs, great scalability was obtained in higher interaction orders in one of the approaches. Overall a combined speedup, over the base work, of more than 2000× was achieved.

Keywords: Epistasis detection, Parallelism, SNP detection, Genetic Algorithm

Resumo

Nos últimos anos, estudos têm demonstrado que a interação genética pode representar um papel fundamental no aparecimento de doenças que afetam o ser humano. Tem sido desenvolvido um número cada vez maior de abordagens para detetar estas interações, mas devido à complexidade de certas doenças, é necessário estudar graus de interação genética cada vez maiores. Devido à dimensão do espaço de procura, são necessárias abordagens rápidas e eficientes para detetar interações de alta ordem. Para abordar este problema, esta Tese propõe melhorias para o algoritmo genético multi-objetivo, NSGA-II, e apresenta dois designs de paralelização do algorítmo proposto. As melhorias apresentadas foram testadas até 10 ordens de interação, em três problemas diferentes, num sistema multiprocessador Intel Xeon Gold multicore. Foram atingidas melhorias significativas para todas as ordens de interação, chegando as mesmas a ser de até 196$\times$. Foi obtida ótima escalabilidade, num dos designs de parallelização propostos, para ordens de interação altas. No geral, foi atingido um *speedup* combinado, sobre o trabalho base, de mais de 2000$\times$.

Palavras-chave: Epistasia, SNP deteção, Parallelismo, Algoritmo Genético

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Human genetics, the study of inheritance in human beings, has as one of its goals the identification of genes that increase the risk of a certain disease. Identifying those genes can lead to the development of new treatment, prevention and diagnostic methods, thus leading to the improvement of human health. Even though the great importance of this task, it is not easily accomplished due to the complex interaction between multiple genes and multiple environmental factors [1].

The focus of this work is on single nucleotide polymorphisms (SNPs), and their complex interactions. A SNP is a single point in a DNA sequence that differs between individuals [2]. The traditional approach to linking the state of these SNPs to the risk of a certain disease, is to measure the genotypes of people with, case samples, and without, control samples, a certain disease with a relevant number of SNPs, these studies are called Genome-Wide Association Studies (GWAS). Afterwards each SNP is individually tested for a direct association with the disease under study. This approach has had some success, but by only examining the association of single SNPs, these studies ignore complex interactions that may be critical to understanding more complex diseases [2, 3]. The combined effect of multiple interacting SNPs is known as epistasis [4]. Epistasis seems to be the main responsible for the appearance of complex traits like human diseases [5]. Methods designed for single SNP detection do not work for these interactions, which cannot be modeled [6].

A number of approaches have been proposed to detect epistasis, these can generally be classified into three categories: exhaustive search, stochastic search and machine-learning approaches [7]. Since a number of studies revealed that these single correlation model or objective function approaches perform inconsistently with different disease models, and that even the same approach will often vary when applied to different disease models, epistasis detection researches have been moving towards multi-objective methods, methods with multiple decision criteria [3]. These methods have been described as having better accuracy and thus avoiding those inconsistencies [3]. Epistasis introduces a greater level of complexity, since the number of SNPs involved in the interactions (k) can vary from 2 to unknown numbers. The search space of this optimization problem grows exponentially with the growth of the epistasis interaction order [8]. More specifically, for a k-order epistasis and M SNPs, the number of possible combinations is given by $\frac{M!}{k! \times (M-k)!}$. Additionally, to increase the reliability of a study a greater number

of case-control samples is needed. Due to the complexity of epistasis interaction detection brute-force approaches are impractical [9]. Recently, genetic and evolutionary algorithms have been used to solve high-order epistasis interactions, of up to k=8 [9].

1.1 Objectives

Epistasis detection raises computational challenges since analyzing every SNP combination is not viable at a genome-wide scale. Using as a reference the works from [9] and [3], this problem can be surpassed. Even with genetic algorithms, the complexity of the problem grows linearly with the number of individuals in the studied population, but it has an exponential growth with the increase of the interaction order degree. Thus high order epistasis detection is at the moment an unpractical problem to tackle.

This thesis has the Non-dominated Sorting Genetic Algorithm (NSGA-II) of [9] as base, with the main goal being reducing the time taken in epistasis detection, in order to reach epistasis interaction order of 10, previously not tackled. To achieve this goal, improvements over the main functions of NSGA-II are proposed. Additionally, to further improve computational times and to exploit CPU potential, based on the work of [10], two parallelization schemes using OpenMP are proposed. In this context parallelization is not trivial due to an intrinsic serial component of the algorithm and data dependencies but this combination of approaches has already proven results in other complex optimization problems [11]. More specifically, this work has the following objectives:

- Proposing methods to overcome the limitations in high order epistasis detection, more specifically by improving bottle-necks of state-of-the-art works in epistasis detection, and by proposing two parallel designs, based on [10], to exploit the CPU potential, and reach interactions of size 10, previously not addressed.
- Proposing methods to improve low order epistasis interaction computational detection times;
- Analyzing the proposed methods with epistasis interaction orders of up to 10, (epistasis interactions order of up to 8 have been tested in the literature) as well as both parallelization designs, by using benchmark and real-world problem instances, and comparing them with the reference works, on a server-grade Intel Xeon Gold multiprocessor system.

1.2 Report Outline

In the first chapter (Chapter 1) of this dissertation, an introduction to the work of this report has been done. Thereafter, this works is divided in the following way:

- In Chapter 2, a summary regarding state-of-the-art is presented, the problem tackled in this work is formulated, and an explanation of NSGA-II, which serves as base for this work, is made.
- In Chapter 3, the proposed methods to improve execution time in epistasis detection are presented, and in order to tackle high order epistasis interactions two parallel implementations are introduced.

- In Chapter 4, experimental results of the proposed improvement methods are presented, experimental results from the two proposed parallel designs are presented and analyzed, a comparison of execution times with NSGA-II is presented, and a brief evaluation of solution quality is made.
- In Chapter 5, the conclusions obtained from the work performed in this Thesis are presented, as well as, possible future works.

Chapter 2

Background in epistasis detection methods

The work presented in this Thesis is focused on proposing methods to reduce computational time in epistasis detection methods, in order to achieve high order epistasis interactions. To achieve this goal, in this chapter an overview of state-of-the-art methods in epistasis detection is made, and the problem formulation is stated. To facilitate the understanding of the proposals presented, a special emphasis is given to state-of-the-art method NSGA-II (Non-dominated Sorting Genetic Algorithm), which serves as starting point for the work presented in this Thesis.

2.1 State-of-the-art methods in epistasis detection

Results from Genome-Wide Association Studies (GWAS) have been focused on single associations between the disease under study and Single Nucleotide Polymorphisms (SNPs), where only addictive effects are considered. Limited results have been achieved with these studies [12], due to these single associations only explaining a small fraction of the estimated heritability [13] (heritability is the proportion of variation in a trait explained by inherited genetic variants). One factor that could explain this, is the multiple interaction between SNPs, known as epistasis [14]. Recently a great number of statistical methods have been developed for epistasis detection in GWAS, ranging from exhaustive search to various machine learning approaches.

The search space for epistasis detection grows exponentially with the interaction order. This leads to exhaustive search algorithms being limited to an order of interaction of two or three SNPs, due to the huge computational time required to finish the analysis [8]. In order to reduce execution time, works using Graphical Processing Units (GPUs) like [15, 16] can be found. In [17], two alternatives to solve this problem are presented, the first being multiple GPUs, and the second multiple Field Programmable Gate Arrays (FPGAs). Cloud based approaches can also be found [18]. [19] proposes a multi-threaded java interface. Clusters based on Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) are also employed to solve this problem. [20] proposes a combination of a GPU and a FPGA, to solve

exhaustive third order epistasis interaction detection, claimed to be faster than proposals using GPU only, as [21]. Finally, [22] proposes using FPGAs to tackle third order epistasis detection, but stating that problems with half a million SNPs could take as much as three years. All the previous works can only tackle second and third order epistasis interactions, leading to the conclusion that for higher order of interaction non-exhaustive approaches are needed.

To efficiently tackle higher than two epistasis interaction orders, an exhaustive exploration of the full search space can not be executed. Following this idea, [23] tries to identify variants that have a non-zero interaction effect with other variants, in this way avoiding the direct search for pairwise or higher order interactions. [24] applies a pre-processing filter, SingleMI, to reduce the search space, and GPUs to improve execution time. In [25], Sure Independence Screening (SIS) is used to reduce the search space, with SIS the SNPs are ordered according to their marginal correlation with the trait and afterwards a number of best candidate SNPs is selected to be tested. All the previous works with an incorrect filtering could be losing important SNPs.

Data-mining algorithms such as random forests, Monte Carlo logic regression, and multifactor dimensionality reduction, have also been proposed, and have their performances evaluated in [26]. More recently, machine learning approaches as [27, 28, 29] can be found. These approaches suffer from ”short fat data” problem, where the number of SNPs outnumbers the sample size, and the necessity to train the algorithm.

Another approach to this problem are evolutionary algorithms, the approach tackled in this thesis. Evolutionary algorithms are a more recent trend in epistasis detection. MACOED, is a memory-based multi-objective ant colony optimization algorithm, which is able to retain non-dominated solutions found in past iterations, to solve the space and time complexity for high-dimension problems, for pairwise interactions, with proven results [3]. [30] combines a Bayesian scoring function with an evolutionary-based heuristic search algorithm to solve epistasis interaction. FSHA-SED is a multiobjective second order epistasis detection method based on the harmony search algorithm [31]. Finally the Non-dominated sorting algorithm of [32] was applied to epistasis detection in [9], reducing the memory required for epistasis detection and tackling high interaction orders of up to 8 SNPs.

Every epistasis detection algorithm requires a metric to evaluate each solution, i.e., an objective score function. Traditionally single objective scores have been used in epistasis detection methods, but more recently a trend of multiobjective, mostly two, scores has emerged due to some inconsistencies in single objective methods and a better performance using multiobjective approaches [3]. A multiobjective problem is aimed at finding a set of multiple solutions, called Pareto optimal solutions or Pareto front. The Pareto front represents the trade-off between the two objective score functions. Some of the most used scores are the Akaike information criterion (AIC) [3, 9] and Bayesian network based scores [3, 9, 31, 30]. Other scores like Gini-score can be found [31]. Table 2.1 presents a summary of the state-of-the-art methods, with the maximum epistasis interaction order studied.

To evaluate the effectiveness of these algorithms solution quality measurements such as Power, Precision, Recall and F-Measure [3] can be used. For multiobjective optimization methods the hypervolume metric can be used to evaluate and compare sets of compromise solutions [33].

Method	Epi. Order Allowed	Multiobj.	Paper
Epiblaster-fast exhaustive strategy using GPUs	2	X	[15]
Two-stage search strategy on GPUs	2	X	[16]
Parallel exhaustive epistasis detection on FPGA and GPU	2	X	[17]
Exhaustive combination of GPU and FPGA	2	X	[21]
MPI and OpenMP exhaustive approach	2	X	[19]
Cloud-based exhaustive approach	2	X	[18]
Exhaustive search on FPGAs	3	X	[22]
Identifying non-zero interactions avoiding direct search	X	X	[23]
Pre-procissing filter, SingleMI, with GPUs	6	X	[24]
Sure Independence Screening	3	X	[25]
A Deep Learning Approach	2	X	[28]
Combining Multiple Hypothesis Testing with Machine Learning	2	X	[29]
FSHA-SE (Genetic Algorithm)	2+	✓	[31]
MACOED (Genetic Algorithm)	2	✓	[3]
Cuckoo (Genetic Algorithm based)	5	X	[30]
NSGA-II (Genetic Algorithm)	8	✓	[9]

Table 2.1: State-of-the-art works

2.2 Problem Formulation

The combined effect of multiple interacting single-nucleotide polymorphisms (SNPs), a single point in a DNA sequence that differs between individuals, is known as epistasis. Epistasis seems to be the main responsible for the appearance of complex traits like human diseases [5]. Genome-Wide Association Studies (GWAS) provide SNP data from case samples, samples with the disease under study, and control samples, samples known not to have the disease under study, that can be used to determine epistasis interactions. The goal of this work is to identify which interacting SNP combinations have a higher probability of being responsible for the manifestation of a certain disease in the case samples, by analysing the SNP values present in the genotype of the dataset samples.

To make those analysis it is necessary to process a dataset containing N case-control samples, M SNPs and the information of the disease state, i.e. if the sample is a case or a control. For every sample the file contains the genotype value of every SNP marker under study. Each SNP is represented by a number from 0 to 2, to codify the possible allele representations, i.e. to codify each variant form of a given gene, 0 for homozygous major allele, 1 for heterozygous allele, or 2 for homozygous minor allele. The disease state is represented by either 0, a control sample, or 1, a case sample. Figure 2.1 shows an example of a dataset with 10 SNPs (from 0 to 9) and 10 samples- Each row represents a sample and

each column represents the value of a SNP for each sample. The last column represents the disease state of each sample. After analysis, the output (for a epistasis interaction order k) corresponds to a list of, k SNP markers.

Epistasis detection is a complex problem due to the number of SNPs and the epistasis interaction order, when the interaction order increases the search space increases exponentially. The number of possible combinations for a k-order epistasis analysis and M SNP markers is given by: $\frac{M!}{k!(M-k)!}$.

```
x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,Class
1, 0, 1, 1, 0, 1, 2, 0, 1, 0, 0
0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0
1, 0, 0, 0, 1, 1, 1, 2, 1, 0, 1
0, 1, 0, 0, 0, 0, 2, 1, 0, 1, 1
0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0
1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1
0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0
1, 0, 1, 1, 0, 0, 0, 2, 0, 1, 0
1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1
0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1
```

Figure 2.1: Example of a dataset

2.3 NSGA-II

This work has as base the structure of the state-of-the-art multi-objective tools for epistasis detection. More specifically the multi objective evolutionary search engine, Non-dominated Sorting Genetic Algorithm II (NSGA-II), a bi-objective optimization problem with the widely used K2 and AIC scores, based on the work of [32]. This algorithm has as its objective the improvement of a population of candidate solutions to form a Pareto front. In this implementation an individual (or solution) is represented by an integer array of k-elements, where each element represents an SNP from the input dataset. In this way, this array codifies the interactive effect between the specified k SNPs. NSGA-II sorts the population hierarchically into sets of individuals, with the best set being called the Pareto front approximation. Sorting is done based on the objective score values of each individual, explained in the following sections. This algorithm uses an evolutionary process, to improve the candidate population, based on evolutionary operators. These operators are selection, crossover, and mutation. Selection is the process of choosing the parents of an offspring individual, i.e. a set of k SNPs. Each parent is obtained via a binary tournament selection between two candidate individuals chosen randomly. From the two individuals picked, the individual chosen to become one of the parents is the one with the lowest rank, or if they belong to the same rank the individual with the highest crowding distance (the concepts of crowding distance and rank are explained in section 2.3.2). In order to have both parents the selection operator has to be applied twice. Crossover is a genetic operator that stochastically combines the SNP values, from the two selected parents, to generate a new individual. Mutation is a genetic operator used to preserve diversity in the population. This operator randomly changes an individual's SNP values, with a certain mutation probability, to create a different individual.

NSGA-II can be generally described in 7 steps, presented in Figure 2.2. Step 1: Creation of a random population to initialize the search for the Pareto front. Step 2: Non dominated sorting process to rank the initial population. Step 3: Generation of the first offspring population based on the evolutionary operators. Step 4: Rank the combined population, parent and offspring populations. Step 5: Based on the determined ranks and crowding distance of each individual, determine the population of the next generation. Step 6: Generation of a new population. Steps 4 to 6 are repeated until a certain number of generation is reached, this number of generations can vary according to the problem tackled. Step 7: Final rank of the population, of which the first rank corresponds to the Pareto front approximation of the problem.

To better understand the work done in this thesis in the following sections some relevant steps of NSGA-II are explained in more detail.

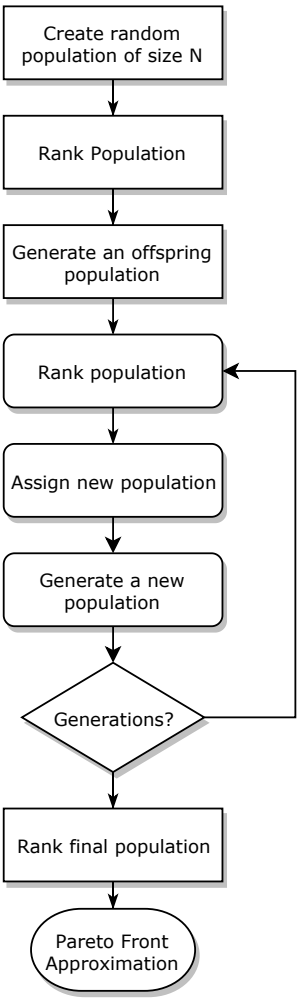


Figure 2.2: General diagram of the epistasis detection algorithm.

2.3.1 Generation of the offspring population

In this algorithm two types of approaches to generate the population are needed. The first is a simple random generation of individuals, ensuring that no repeated individuals are created, that constitutes the starting point of the algorithm for the search of the Pareto front. The second type is a generation of an offspring population by applying evolutionary operators to a parent population. The initial population constitutes the first parent population used for the generation of the first population of offspring individuals. In each generation of offspring population the number of generated individuals is equal to the number of individuals in the parent population. For each new individual generated, two parent individuals are necessary. To select each parent the selection operator is applied. Each time the selection operator is applied two individuals are randomly chosen among the parent population. From this two individuals the best is selected to become one of the parents. The criterion to chose the best individual is based on the rank of each individual and in its crowding distance, the best individual is the one with the lowest rank, or in the case of equal ranks, the one with the highest value in crowding distance, i.e. the sparsest solution. Having both parents selected, two offspring individuals are generated through the double-point crossover operation. In this operation two random points in the SNP array representing an individual are selected, those points are called crossover points. The offspring solution will have the SNPs represented between the crossover points from a parent and the SNPs from outside the crossover points from the other parent. The crossover between parents happens with a certain probability, the crossover probability, that can be fine tuned to improve solution quality. Also based on a probability, the mutation probability, that can also be fine tuned to improve solution quality, the mutation operation can be applied. This operation changes, or not, based on the mutation probability, each stored SNP that represents an individual, in order to maintain diversity in the population from one generation to the next. The mutation probability should be kept low, otherwise the algorithm will become a random search. After the generation of each individual a comparison with a database of previously generated individuals is made to determine if each individual generated is a new individual that has not been evaluated before, avoiding repetition of the solutions analyzed. If the individual has already been generated, a new random individual is created. Finally the objective scores that characterize each solution are computed before generating the next pair of individuals.

2.3.2 Pareto Ranks Determination

To understand the ranking system, it is necessary to understand the concept of dominance first. Each individual solution has two values corresponding to the objective scores. Based on these two values, it is possible to compare solutions. If one solution is better than another, it means that dominates the other. In order to dominate another solution, it is necessary to have one objective score strictly lower than the dominated solution, and the other objective score equal or lower, assuming that both objectives are to be minimized (lower is better). Any other combination means that a certain individual does not dominate another. In Figure 2.3 it is possible to see a set of individuals, each represented by a cross, marked in a plot according to their objective score values AIC and K2. The green line represents the Pareto

front approximation. Three of those individuals are marked with the numbers 1, 2 and 3. Individual 1 dominates individual 3, since both objective scores are lower, as does individual 2. But individual 1 does not dominate individual 2, and individual 2 does not dominate individual 1. Because individual 1 has objective score AIC lower than individual 2's score AIC and individual 2 has the objective score K2 lower than individual 1's K2 score.

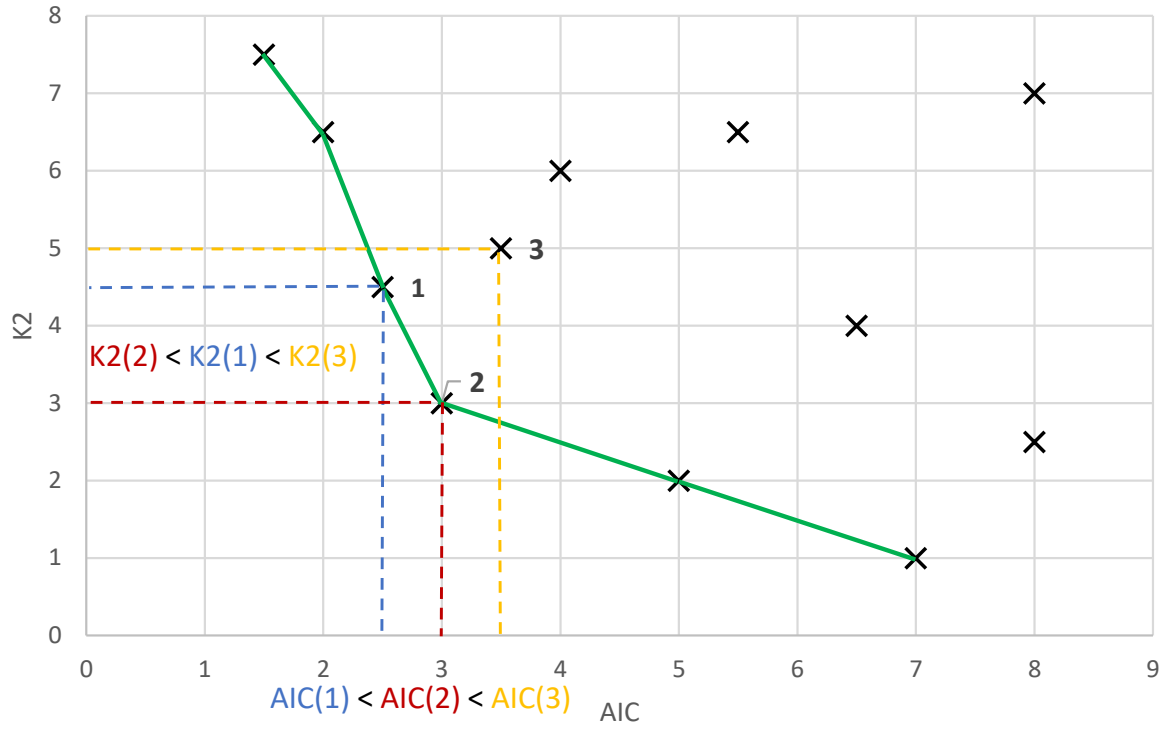


Figure 2.3: Explanation of the concept of domination.

The ranking system is based on the Fast Non-Dominated Sorting algorithm, which pseudo-code is presented in Algorithm 1. The goal of this algorithm is to sort the population in fronts, or ranks, with the first front being the Pareto front approximation given by the global algorithm. This can be done by cross-comparing every individual to determine which individuals dominate each other. Upon this comparison, every individual has a domination number, this number represents the number of individuals that dominate that specific individual. After this comparison the individuals who have zero as their domination number, those who are not dominated by any other individual, will form the first front. Additionally every individual has a list of the individuals that it dominates. This list serves to determine the following fronts. For example an individual who is dominated only by individuals belonging to the first front will belong to the second front. An individual who is dominated only by individuals from the first and second fronts belongs to the third front. To determine all the ranks of the population, this algorithm applies this process until there are no more individuals left to assign.

The Fast Non-dominated Sorting algorithm, shown in Algorithm 1, starts by comparing every individual of the population with all the other individuals, hence the two for loops, with upper limit PopSize

(Population Size), on lines 2 and 4. While an individual is compared with the others a counter, per every individual, is kept, the domination number. The domination number stores the number of individuals that dominate the evaluated individual, as it can be seen in the algorithm, on line 9, represented by n_p, with p being the currently evaluated individual. At the same time list is kept, per every individual, of all the individuals that are dominated by that particular one, in the algorithm, on line 7, represented by S_p. The individuals whose domination number, n_p is zero, line 12, meaning that these individuals are not dominated by anyone, belong to the the first front, line 14, and their rank will the best, rank 0, line 13. This rank is the approximation of the Pareto front.

The second stage of the algorithm starts with the individuals in the first front, lines 17 and 18. To discover the individuals that belong to the second front it is necessary to traverse the lists, S_p of the individuals in the first front. One by one, each dominated individual, q, has their domination number decremented, lines 21 and 22. If this number reaches 0, that individual, q, will belong to the front that is currently being filled, lines 23-25. When all individuals from a front are checked the next front will be checked. This process is repeated until a front is empty, which means that every individual in the population is ranked. Each front constitutes an approximation for a Pareto rank.

At the end of every generation the first N elements of the first fronts will make the next parent population. Rarely does the N-th individual coincide with the end of a front, so it is necessary to pick, for the remaining free spots, some individuals from the next front, but not all. This creates the necessity of having a criterion for choosing individuals inside a front. This criterion is the crowding distance. Crowding distance represents the relative distance from a point to its neighbours. It is computed using as reference the two extreme solutions in the objective scores, the one with the lowest K2 and highest AIC and the one with lowest AIC and highest K2. The difference between the objective score values of a solution to its neighbours, i.e., the solutions with the closest values in the objective scores, is taken, and then divided by the maximum values in the objective scores. A higher value of crowding distance indicates less solutions explored in that region, and so, to increase solution quality, a higher value of crowding distance is preferable when picking two solutions from the same front.

Algorithm 1 Fast Non-Dominated Sort

```
Initializations
for i=1 to PopSize do
    p ← Pop[i]
    for j=1 to PopSize do
        q ← Pop[j]
        if p < q then
            S_p ← q
        else if q < p then
            n_p = n_p + 1
        end if
    end for
    if n_p = 0 then
        p_rank = 0
        F_1 ← p
    end if
end for
i=1
while F_i ≠ ∅ do
    for all p in F_i do
        for all in S_p do
            q ← S_p
            n_q = n_q − 1
            if n_q = 0 then
                q_rank = i
                F_i ← q
            end if
        end for
    end for
    i=i+1
end while
```

2.3.3 K2 Score

In this bi-objective optimization problem, the first objective score function adopted is built upon the concept of a Bayesian network, a statistical model that uses a directed acyclic graph to represent a set of random variables and their conditional dependencies [34, 35]. In this kind of graph, the association between an SNP x and a certain disease state y is represented by a direct edge linking from node x to y. Thus, the K2 score can be expressed as:

$$K2 = \sum_{i=1}^{I} \Big(\sum_{b=1}^{r_i+1} log(b) - \sum_{j=1}^{J} \sum_{d=1}^{r_{ij}} log(d) \Big), \tag{2.1}$$

where I refers to the number of possible genotype combinations (for an epistasis interaction order k, $I = 3^k$ as an SNP can show a value of 0, 1, or 2), J the number of disease states (J = 2 in case-control scenarios), r_i represents the frequency of the i-th genotype combination in the samples at the SNP nodes $[x_1, x_2, ..., x_k]$, i.e. the number of times each combination of the analyzed SNPs appears in the samples, and r_{ij} the number of samples where the disease node takes the j-th state and the SNPs take the i-th genotype combination, i.e. the number of of times each SNP combination appears for each of the disease states of the samples. The goal in this problem is to minimize this function, the lower the score value the stronger the association between the SNP set and the disease.

2.3.4 AIC score

The second objective score function is based on a logistic regression, which has been widely used in state of the art multiobjective tools [36], that calculates the likelihood of occurrence of the disease for a certain SNP combination under a multilocus interaction model. The Akaike information criterion (AIC) is a metric that reflects the model fitness to the dataset and the complexity of the model used [37]. In this model $log(lik)$ represents the maximized log-likelihood of the model, and d the number of free parameters.

$$AIC = -2log(lik) + 2d \tag{2.2}$$

To compute the likelihood an additive-interactive model from North et al. (2005) [38] was used according to Equation 2.3.

$$lik = log\frac{p}{1-p} = \mu + \sum_{i=1}^{k} \beta_i x_i + \xi \prod_{i=1}^{k} x_i \tag{2.3}$$

In the previous equation x_i represents the i-th SNP in the evaluated solution, μ is a mean term of population prevalence and sample sizes, β and ξ refer to additive and interactive effect factors respectively, and p represents the conditional probability $P(y = 1|[x_1, ..., x_k])$. As in K2 score, this is also an objective function score to minimize.

2.4 Summary

This chapter has firstly presented an overview regarding the state-of-the-art epistasis detection methods. Afterwards, the problem formulation has been made. Finally, the chapter has introduced a detailed explanation of NSGA-II, a state-of-the-art method in epistasis interaction detection, that represents the base of this work. This explanation has detailed the most important steps in this algorithm that will be improved and optimized in Chapter 3.

Chapter 3

Proposed methods for reducing epistasis detection time

In this chapter the proposed optimizations to improve the state-of-the-art methods, in order to undertake high order epistasis study and to accelerate the study of low order epistasis interactions, will be explained. The most time consuming functions will be tackled, namely the objective scores, K2 and AIC, the database of generated individuals and the sorting algorithm. Each proposed improvement will be explained in detail, as will some of the problems encountered and the solutions to overcome them.

Also in this chapter, after improving the main functions of the evolutionary algorithm, in order to fully exploit the capabilities of the CPU and to further reduce the time taken in epistasis detection, the potential parallelism will be exploited. Two independent approaches will be investigated. The first approach being an iteration-level strategy that allows the definition of a problem-independent approach to parallelize epistasis detection methods, without modifying the behaviour of the search engine compared to the serial version [10]. In this sense, the generation and processing of new candidate solutions can be handled by distributing different solutions to different execution threads, thus representing the main source of problem-independent parallelism in the method. The second parallelization trend considered is implemented at the solution level [10], meaning that parallelism is applied over computationally demanding operations performed for each of the candidate solutions. This idea defines a problem-dependent parallelization approach, as the main target of parallelism is focused on the objective function inner loops. The achievable parallelism, with this approach, depends on specific variables from the optimization problem, being a harder parallelization to implement. Both of the parallelization designs do not have impact in the search capabilities of the algorithm, since their goal is to minimize processing time. The two approaches were implemented using the standard OpenMP.

3.1 Generation of the population

Upon the generation of the offspring population, the computation of the objective scores is performed immediately for every individual. With these objective score values, it is possible to check if an individual

will not be in the parent population of the next generation. If an individual is dominated by the last front in the parent population, it will have at least N individuals in the ranks above, and thus will not make the cut of N individuals chosen for the next parent population. By checking the quality of each solution upon generation it is possible to reduce the number of offspring individuals introduced in the offspring population. As a result, having a smaller offspring population, the time spent sorting, determining ranks and assigning a crowding distance can be reduced.

3.1.1 Database of the Generated Individuals

Since a more diverse population produces better results and the best individuals remain in the first rank of the population, it is not necessary to check the same individuals again, i.e. generating an individual more than once. For this purpose, it is necessary to keep a database of individuals generated and evaluated. Having to check each individual already generated (at each new generation round) would represent a huge bottleneck in the execution time. With the maximum number of individuals checked being Number_of_iterations * Population_size.

A solution for this problem that is both time and memory efficient is a hash table. Each entry of this hash table has the identifiers x_i of the SNPs markers that compose that particular solution. In a hash table, the time necessary to find a solution can be reduced up to Hash_{Size} times. To achieve this speedup, it would require a perfect hash function which is really hard to obtain and can also be computationally heavy. A simple and fast dispersion function is used: (SNP1+ ... + SNPN)/Hash_{Size}. Since it is not a perfect hash, and with every execution the hash is filled in a different way, the memory allocated initially is around 90% of what would be a obtained with a perfect dispersion function and in the necessary positions more memory is allocated.

3.2 Sorting of the population

In a genetic algorithm, the selection of the best individuals to proceed to the next generation is essential. To determine the Pareto ranks of the solutions in the population, NSGA-II relies on the use of fast non-dominated sorting, after which the candidates with the lowest ranks are selected, while the best solutions within the same rank are determined according to the crowding distance[32]. Fast non-dominated sorting detects the non-dominated solutions by performing an exhaustive search over the population to construct a complete set of dominated solutions for each individual and to determine the number of individuals that dominate each individual. In contrast to this approach, the ranking algorithm proposed in this Thesis focuses on the necessary conditions for the solutions to belong to the same rank. Assuming a minimization context, two solutions Υ_1 and Υ_2 necessarily belong to the same rank if and only if their scores for both objective functions are equal, i.e., $K2(\Upsilon_1) = K2(\Upsilon_2)$ and $AIC(\Upsilon_1) = AIC(\Upsilon_2)$ (equality condition, $\Upsilon_1 \approx \Upsilon_2$) or if they improve each other in different objective functions, i.e., $K2(\Upsilon_1) < K2(\Upsilon_2)$ ($\Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_2$) and $AIC(\Upsilon_2) < AIC(\Upsilon_1)$ ($\Upsilon_2 \xrightarrow{\text{AIC}} \Upsilon_1$). Furthermore, any given solution Υ_3 will belong to the same rank as the solutions Υ_1 and Υ_2 if and only if:

- $\Upsilon_3 \approx \Upsilon_1$ or $\Upsilon_3 \approx \Upsilon_2$ (equality conditions); or

- $\Upsilon_3 \xrightarrow{\text{K2}} \Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_2$ and $\Upsilon_2 \xrightarrow{\text{AIC}} \Upsilon_1 \xrightarrow{\text{AIC}} \Upsilon_3$; or
- $\Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_3 \xrightarrow{\text{K2}} \Upsilon_2$ and $\Upsilon_2 \xrightarrow{\text{AIC}} \Upsilon_3 \xrightarrow{\text{AIC}} \Upsilon_1$; or
- $\Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_2 \xrightarrow{\text{K2}} \Upsilon_3$ and $\Upsilon_3 \xrightarrow{\text{AIC}} \Upsilon_2 \xrightarrow{\text{AIC}} \Upsilon_1$.

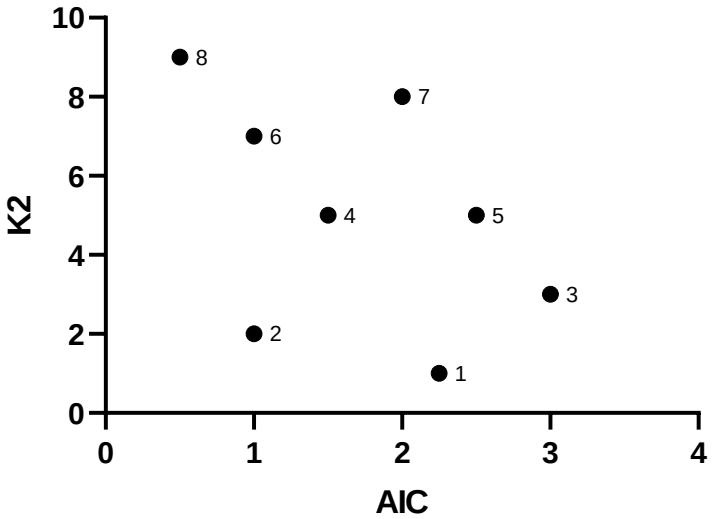
A set of solutions Υ_i will belong to the same rank if and only if the equality conditions are satisfied or if the order of solutions remains the same when they are sorted in the increasing order of their K2 score and in the decreasing order of their AIC score, i.e., $\Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_2 \xrightarrow{\text{K2}} ... \xrightarrow{\text{K2}} \Upsilon_n$ and $\Upsilon_1 \xleftarrow{\text{AIC}} \Upsilon_2 \xleftarrow{\text{AIC}} ... \xleftarrow{\text{AIC}} \Upsilon_n$. By relying on this observation, the proposed ranking procedure relies on sorting the candidate solutions in a increasing order of their K2 score, with the aim of reassuring the condition $\Upsilon_1 \xrightarrow{\text{K2}} \Upsilon_2 \xrightarrow{\text{K2}} ... \xrightarrow{\text{K2}} \Upsilon_n$. Figure 3.1a provides an example of a set of sorted solutions, which are enumerated according to their increasing K2 score value. After sorting, two adjacent solutions Υ_i and Υ_{i+1} will belong to the same rank if and only if:

- $\Upsilon_i \approx \Upsilon_{i+1}$ (#1.1); or
- $\Upsilon_{i+1} \xrightarrow{\text{AIC}} \Upsilon_i$ (#1.2), since the condition $\Upsilon_i \xrightarrow{\text{K2}} \Upsilon_{i+1}$ is necessary satisfied due to the sorting.

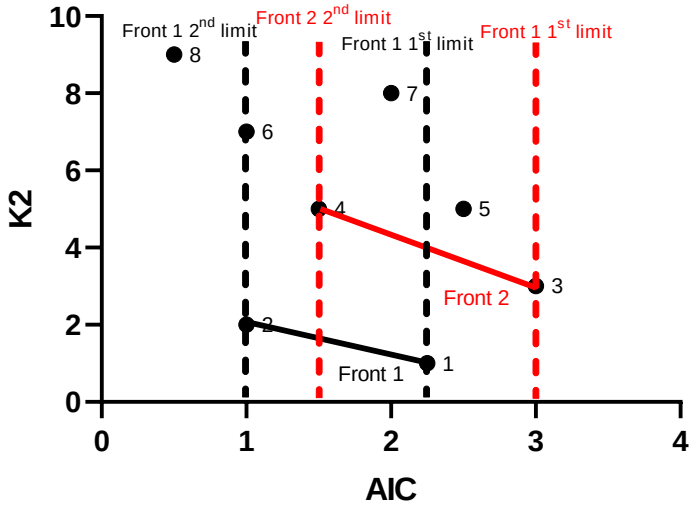
In all other cases, two adjacent solutions Υ_i and Υ_{i+1} will belong to different ranks, most notably if:

- $K2(\Upsilon_i) = K2(\Upsilon_{i+1})$, the solution with a lower AIC score will be placed in the lower rank (#2.1); or
- $\Upsilon_i \xrightarrow{\text{AIC}} \Upsilon_{i+1} (\Upsilon_i \xrightarrow{\text{K2}} \Upsilon_{i+1})$, the solution Υ_i will belong to the lower rank (#2.2); or
- $AIC(\Upsilon_i) = AIC(\Upsilon_{i+1}) (\Upsilon_i \xrightarrow{\text{K2}} \Upsilon_{i+1})$,the solution Υ_i will belong to the lower rank (#2.3).

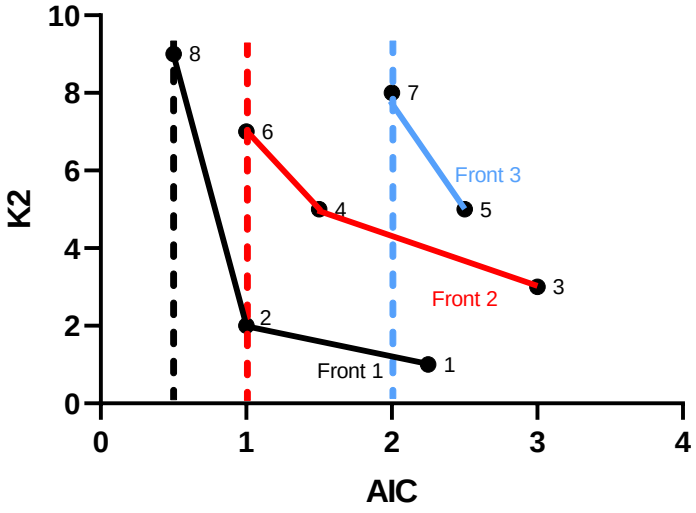
In this ranking procedure, the creation of ranks starts from the solution with the minimum K2 score. This process is depicted in Figure 3.1b when creating the first two fronts (ranks). Since the solution 1 has the minimum K2 score, it is placed in the Front 1. Solution 2 will join solution 1 in Front 1 since it satisfies #1.2 (i.e., it has a lower AIC score than the solution 1). The solution 3 cannot belong to Front 1 due to #2.2 (both K2 and AIC scores are higher when compared to the solutions 1 and 2), thus it is placed in the Front 2. Since solution 4 satisfies #1.2 it is placed in the Front 2 together with the solution 3. Solution 5 will belong to Front 3 due to #2.1 (equal K2 cores and a higher AIC score when compared to the solution 4 from Front 2). In this ranking procedure, each front (rank) is additionally assigned with the minimum AIC score among the solutions in that rank. For example, for the fronts depicted in Figure 3.1b, the AIC score of solution 2 will be considered for Front 1, while the AIC score of solution 4 will be assigned to Front 2 (see the vertical dashed lines). The consideration of the minimum AIC scores for each front is crucial to allow insertion of additional solutions in the already created fronts. Given the fact that all other non-examined solutions (that do not satisfy #2.1) must have a higher K2 score than the already ranked solutions (due to the initial sorting), the solution under evaluation can only belong to an existing front if and only if its AIC score is strictly lower than the AIC scores of all solutions already included in that front (see condition #1.2). As a result, before creating a new front, the AIC score of the currently evaluated solution is compared with the minimum AIC score of the existing fronts (starting from the last created front up to Front 1).



(a) Example sorted by K2



(b) Step 4 of ranking



(c) Final result

Figure 3.1: Example of the proposed Ranking Algorithm

If there exists a front with a minimum AIC score that is higher than the AIC score of the current solution, the solution is appended to that front and the minimum AIC score of the front is updated with

the AIC score of the appended solution. This process is depicted in Figure 3.1c. Since the solution 6 cannot be added to Front 1 due to #2.3 (equal AIC score and higher K2 score than the solution 2), it is appended to Front 2 and the minimum AIC score of the Front 2 is updated with the AIC score of the solution 6 (see the vertical dashed line moving to the left). Since the AIC score of solution 7 is higher than the minimum AIC scores of Fronts 1 and 2, it is added to the Front 3 under the condition #1.2 with respect to the solution 5. Finally, the solution 8 is appended to the Front 1, since it is the best front that allows reduction of its minimum AIC score. Once all solutions are ranked, the best popSize solutions are selected (starting from Front 1). The individuals belonging to the same front are selected according to the crowding distance. It is worth noting that, in the proposed approach, the calculation of crowding distance does not require any additional sorting, since the applied K2 sorting and the solution appending process guarantee that the solutions within the same rank are sorted according to their increasing K2 score and decreasing AIC score. As such, since the first and the last solutions mark the relative points for crowding distance calculation, the crowding distance of the solutions is directly obtained by iterating over the list of individuals of every front.

Algorithm 2 presents the pseudo-code of the proposed sorting and ranking algorithm. In this algorithm, p represents the solution under evaluation, F_i the list of solutions belonging to Front i, $N[i]$ the minimum value in AIC score for Front i, $rank$ represents the highest rank assigned (the worst rank), and x represents the rank that will be assigned to solution p.

Algorithm 2 Sorting and Ranking

$Pop \leftarrow Sort_{K2}(Pop)$
$p \leftarrow Pop[1]$
$p_{rank} = 0$
$rank = 0$
$F_1 \leftarrow p$
$N[1] \leftarrow p_{AIC}$
for $i = 2$ to $PopSize$ **do**
    $x = rank$
    $p \leftarrow Pop[i]$
    while $p_{AIC} < N[x]$ **do**
        $x = x - 1$
    end while
    if $x = rank$ **then**
        $rank = rank + 1$
    end if
    $p_{rank} = x$
    $F_x \leftarrow p$
    $N[x] = p_{AIC}$
    $p \leftarrow CrowdDistance$
end for

3.3 Objective Score Functions

In order to evaluate the quality of each solution objective scoring functions are needed. Each solution needs to be evaluated in each of the objective scores, with N solutions being generated at every generation efficient objective scoring functions are of the upmost importance. In the following sub-sections proposed improvements over both objective score functions will be explained.

3.3.1 K2 Score

As it can be observed in Equation 2.1, there are two main stages to be performed when computing Bayesian K2 score. The first is the determination of frequencies for each genotype combination i (at the SNPs of the evaluated solution) with respect to the j-th disease state r_{ij} , as well as the overall observed frequency r_i. The second stage refers to the calculation of K2 score as a sum of the logarithms with respect to the determined r_i and r_{ij} frequencies. The first and more computationally expensive step implies traversing through the values of each SNP marker of each sample. Instead of computing the index of the array, that stores the frequency of the i-th genotype combination in the samples at the SNP nodes using the formula

$$Index = \sum_{j=0}^{k} SNP_j \times 3^{k-1-j} \tag{3.1}$$

a three branched tree could be used. A tree with a branch for each disease state and height equal to the epistasis interaction order (plus one to account for the root). An example of a tree for epistasis interaction orders of two is depicted in Figure 3.2. The leaves point to the array r of size 3^k, which encodes all possible genotype combinations of SNP markers given the epistasis interaction order, k. In the example presented in Figure 3.2, the array r has nine positions, corresponding to 3^2 genotype combinations. For each genotype combination i (in the samples at the evaluated SNPs), the frequency vector r holds the number of samples, the frequency, where the disease node takes the j-th state, i.e., r_{ij}, j $\in \{0(controls), 1(cases)\}$. Each genotype combination is encoded with a single integer value in the r vector, which is directly obtainable from the genotype values observed at the selected SNPs of the sample under evaluation.

This approach has a considerable improvement over the previous formula for small epistasis interactions, which relied on the use of powers. The exponential growth of the tree size also implies an exponential growth in memory necessary for its allocation. For high degrees of epistasis interaction, it can become a bottleneck due to the constraints in its access, since traversing the tree takes considerably more time to be performed.

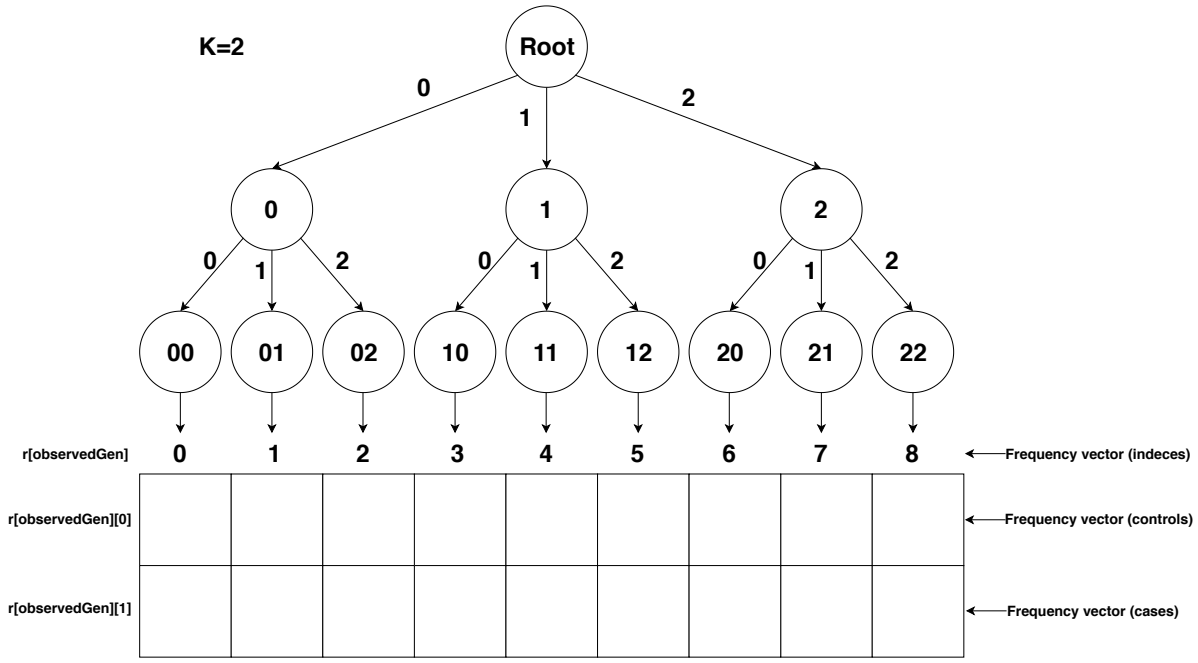


Figure 3.2: Diagram of tree approach to solve K2 indexing for epistasis of size 2

Although this first proposal being relevant for low interaction orders, a second proposal is considered to tackle high interaction orders. To efficiently determine frequency of each genotype combination, a fast way of indexing an array based on the values of the SNP markers for each sample is necessary. The index of genotype combination ($observedGen$), of the array r, can be obtained by recursively applying the expression $observedGen = 3 \times observedGen + D[s, x]$, where different genotype values $D[s, x]$ (observed for the s-th sample at the SNP x) are used as the relative indexing offsets. This process is illustrated in Figure 3.3, for the interaction order k = 3 and the genotype values D[s] = 1,2,1, whose genotype combination index in the r array ($observedGen$) corresponds to the value of 16. In the first step, with only one marker, the offset starts at one. By adding the second marker, in the second iteration, the offset of the array increases to 5. Finally with the last value the final offset of 16 is reached. Depending on the disease state either X (for controls) or Y (for cases) will be incremented. Once the genotype combination index is determined, the respective frequency position (r_{ij}, $i = observedGen$) is incremented depending on the j-th disease state.

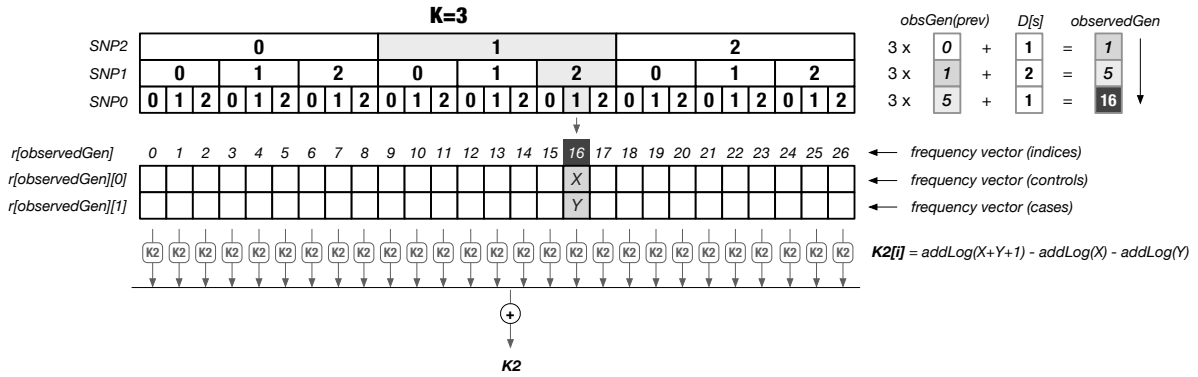


Figure 3.3: Example, for epistasis size 3, of index calculation

As for the second stage of the objective score K2, it is necessary to perform 3^k partial K2 score calculations, as it can be seen in Equation 2.1. By simplifying this Equation the number of computations can be reduced. The formula expressed in Equation 2.1 can also be expressed as in Equation 3.2, to allow to reduce the number of logarithm computations.

$$K2 = \sum_{i=1}^{I} \begin{cases} \sum_{b=r_{i1}+1}^{r_i+1} log(b) - \sum_{d=1}^{r_{i0}} log(d), & if\ r_{i0} < r_{i1} \\ \sum_{b=r_{i0}+1}^{r_i+1} log(b) - \sum_{d=1}^{r_{i1}} log(d), & otherwise \end{cases} \tag{3.2}$$

To avoid repeated computations in the evaluation of an individual and in the evaluation of different individuals, a look-up table is used. Since the upper limit of this frequency is the sample size, it is possible to create a look-up table with a reasonable size. By pre-computing the logarithm values and storing them, at initialization, in the table, those expensive computations can be replaced by a simple read from an array.

To pre-compute the values for the table, it is necessary to know what are the upper and lower limits of the computations. For the second part of Equation 3.2, it can be noted that d initializes as 1 and the upper limit, r_{ij} is the sample size. For the first part, $\sum_{b=r_{ij}+1}^{r_i+1} log(b)$ the lower limit, r_{ij} is not predictable, since the number of samples with a certain genotype combination can vary from zero to the sample size. This means that the creation of a look-up table is not a trivial task. In contrast, if Equation 2.1 is taken in consideration those limits are very predictable, and equal to the sample size plus one. Thus, the most efficient approach to determine K2 score is compute according to Equation 2.1, and reading the necessary values from the proposed look up table.

3.3.2 AIC Score

The computation of AIC score represents the main bottleneck in terms of execution time. This is due to the heavy computations to estimate the likelihood.

To determine the likelihood of occurrence of the disease for the SNPs in the evaluated solution, an iterative method is used. According to Equation 2.3, this procedure implies solving the system of

equations $P = XB$, where P is the vector of probabilities $log(\frac{p_s}{1-p_s})$ calculated for each sample s, X a design matrix with rows $[1, x_1, x_2, ..., x_k, x_1x_2...x_k]$ and x_i being the i-th SNP marker of a given sample in the evaluated solution, and B the parameters of the model $[\mu, \beta_1, \beta_2, ..., \beta_k, \xi]$. The parameters B must be estimated prior to these calculations, by applying the iterative re-weighted least squares (IRLS) method [39]:

$$B_{g+1} = (X^T W_g X)^{-1} X^T (W_g X B_g + y - \lambda_g), \tag{3.3}$$

where g is the current iteration of the estimation method, $\lambda_g = \frac{e^{XB_g}}{1+e^{XB_g}}$ represents the expected values, y $\in \{0, 1\}$ the response variables (the observed disease state), and $W = diag(\lambda_g(1 - \lambda_g))$ a diagonal weighting matrix. The parameters of the model are iteratively calculated based on this procedure, whose stop criterion is set to an improvement precision of 0.001 according to the literature [3].

To solve the previous system represented in Equation 3.3, a generic solver could be used, but even though being easier to implement, it can be computationally heavy. By using a dedicated solver some improvements are achieved. To optimize the execution time and since that even for high order epistasis interaction orders the final systems are relatively small, Gauss-Jordan elimination is applied to solve the system, avoiding the full computation of inverse matrices, which is a heavy process.

Computing $X^T W X$ is quite expensive since it depends on the sample size, as does the estimation of the parameters λ, y, and consequently W. It can be noted that the maximum number of genotype combinations depends on the epistasis interaction order, more specifically it is 3^k. When $3^k < SampleSize$ there have to be samples with the same genotype combination, i.e. there are samples with the same SNP values in the respective SNP markers. For example, for k=2, there are only 9 possible combinations of the values of the SNP markers to be evaluated. Since the average number of samples in an input file is large, this opens the possibility to improve the way $X^T W X$ and the parameters are computed. Instead of solving the system of equations for every sample and repeating computations, it is possible to solve it for every genotype combination and weighting its importance by multiplying by its frequency. More specifically, the parameters can be computed for every genotype combination, independently of the number of samples and its values present in the dataset, and later (when using those parameters) it is necessary to multiply them by the genotype frequency of each combination, i.e. to multiply them by the number of samples which have that combination of values in its SNP markers. The determination of this frequency is a task that already has to be done for the computation of K2 score. This way, it is possible to reduce the number of computations when the number of samples is greater than 3^k. This is particularly useful for low order epistasis interactions. The larger the sample size, the more useful it is and higher performance improvements can be obtained. Computing the AIC score in this way can represent gains with a theoretical limit of $\frac{Sample\_Size}{3^k}$. To compute AIC score in the fastest way possible, a decision between the two methods can be made at the beginning of the execution based on the input dataset sample size. When the sample size is lower than the number of possible genotype combinations, AIC score is computed based on each sample, and when the sample size is greater than the number of possible genotype combinations, AIC score is computed based on each genotype combination and the frequency of each one.

3.4 Parallelization of the improved method

In order to improve the computational times of the algorithm two parallelization approaches are investigated, under the work of [10]. The first approach, called problem-independent, tackles the problem at the generation of the solutions, assigning a set solutions to each execution thread. The second approach, called problem-dependent, tackles the problem at the level of each solution, parallelizing, individually, each of its most demanding computations. Both parallelization designs are explained in detail in the following sub-sections.

3.4.1 Problem-independent Parallelization

The first approach considered is an iteration-level strategy that allows the definition of a problem-independent approach to parallelize epistasis detection methods, without modifying the behaviour of the search engine compared to the serial version [10]. The generation and processing of new candidate solutions is tackled by distributing different solutions to different execution threads. Each thread is responsible for the generation, comparison with the database and scoring a certain number of new solutions. This parallelization is achieved by using worksharing directives like `#pragma omp for`, while serial components of this application are ensured by using the clause `#pragma omp single`. Potential data dependencies are protected using *omp locks*.

Algorithm 3 presents the pseudo-code of the proposed approach. Having an initial population, the parallel region is initialized with the clause `#pragma omp parallel`. During the execution of each generation there are two main stages to consider. The first stage is the ranking of the population, with crowding distance assignment, and the identification of the next generation of parents. These steps are sequential and have to be executed serially, which is ensured with the `#pragma omp single` clause (line 4 in Algorithm 3). The second stage of the proposed approach refers to the generation of the offspring population. Since each generation of an offspring solution is independent from the other generations, this generation can be ran in parallel, thus the clause `#pragma omp for` is applied (lines 7 and 8 in Algorithm 3). Each thread is responsible for the generation of a set of new solutions. Generating that new set of solutions involves the selection of the parents of each solution, determining the mutation and crossover, validating each solution in the database, evaluating each solution in both objective scores, and to integrate these solutions in the offspring population (lines 9-13 in Algorithm 3). When validating a new solution in the database, it is necessary to keep a consistency in the shared hash table. This consistency is ensured with the use of *omp locks*. To ensure maximum eficiency when accessing the database, an array of *omp locks* with the same size of the hash table is used, where each position of this array corresponds to an entry of the hash table.

When generating a new offspring individual, there is a possibility that it has been generated in a previous generation, which causes a second attempt at the generation of a different offspring individual. This effect can happen to both individuals of the pair that is generated simultaneously. If ran in parallel, this could cause a load imbalance between threads. To prevent this effect the offspring individuals are generated individually instead of being generated in pairs. These issues can be addressed with the

use of a dynamic scheduling policy in the `#pragma omp for` clause. These dynamic policies attribute iterations of the parallelized loop based on the availability of threads during runtime. In opposition to this schedule, if a static schedule was used, it can happen that a thread is assigned with iterations, with an execution time above average, and consequently the other threads would be ”waiting”, instead of being allocated to computations, contributing to a higher execution time and reduced efficiency in the parallelization.

Algorithm 3 Problem-independent parallel design

```
1: Initialize Population (Population)
2: #pragma omp parallel
3: while ! stop criterion (maximum generations) do
4:     #pragma omp single
5:     Fronts ← Ranking and Crowding (Population)
6:     Parents ← Parent Identification (Fronts) /*selecting best popSize individuals*/
7:     #pragma omp for schedule (dynamic)
8:     for i = 1 to popSize do
9:         p1, p2 ← Parent Selection (Parents)
10:        q ← Crossover and Mutation (p1, p2)
11:        q ← Validate Offspring Solution (q, HashTable)
12:        q.K2, q.AIC ← Evaluate Offspring Solution (q)
13:        Population ← Integrate Offspring Solution (q)
14:    end for
15: end while
16: Return Pareto Solutions
```

3.4.2 Problem-dependent Parallelization

The second approach follows a trend of parallelizing at the solution level [10], by parallelizing the computations performed at each individual solution. The potential parallelism of this trend depends on the specific variables and implementations of the optimization problem. The goal is to reduce time in the problem dependent computations since those are the most expensive computations. More specifically, in the proposed approach, the two objective functions, thus parallel designs of both scores are proposed.

Algorithm 4 presents the pseudo-code of the parallelization of the Bayesian K2 score, which consists of two stages. The first stage refers to a parallelization of the calculation of the genotype frequencies. To determine the frequency of each genotype combination, it is necessary to iterate through every sample, determining its genotype combination and incrementing the corresponding value, given by the disease state of that sample, in the frequency array $r[observedGen][D\_State]$. Since the identification of genotypes does not show dependencies from one sample to another, this process can be ran in parallel using the `#pragma omp for` clause, that assigns samples to be verified by each thread. During this calculation it is necessary to keep the values of r consistent, which can be achieved using the clause

`#pragma omp atomic`. Without this clause it could happen that between a thread reading and incrementing the value, another thread could read the same value and to make exactly the same increment, and the final result appearing as only an increase of one instead of the real increase of two. This first stage is presented in Algorithm 4 on lines 4 to 10. Since this loop can imply a very high number of iterations (depending on the number of samples N), guided scheduling policies are applied instead of dynamic in order to reduce thread management overhead. The second and final stage corresponds to the final calculation of K2 score, expressed in Equation 2.1. To compute K2 score a parallel reduction over the sum of the logarithms, provided by the look-up table, is performed. This operation will give as a result the final K2 score of the evaluated solution. This step is shown from line 13 to 16 of the Algorithm 4. Since in *omp reduction* each thread keeps a private copy of $K2$, there are no synchronization problems.

Algorithm 4 Problem-dependent parallel design - K2 score

```
1: r ← Initialize Genotype Frequencies /* r = 0 */
2: /* Identifying frequencies for each potential genotype combination and disease state */
3: #pragma omp for schedule (guided)
4: for s=1 to N do
5:     for i=k to 1 do
6:         observedGen ← Identify Genotype Combination (observedGen, dataset[s][q.SNP[i]])
7:     end for
8:     D_State ← Get Disease State(dataset[s][M+1])
9:     #pragma omp atomic
10:    r[observedGen][D_State]++
11: end for
12: /* Applying Equation 2.1 with look-up table */
13: #pragma omp for reduction(+:K2)
14: for i=1 to 3^k do
15:    K2 ← K2_Comp(r[i][0], r[i][1], r[i][0]+r[i][1])
16: end for
17: return K2
```

As for the second objective score, the pseudo-code for its parallelization can be seen in Algorithm 5. The first step is the initialization of the parameters to compute the likelihood using the IRLS method, which involves initializing B and X. Since X is a matrix, its initialization can be easily parallelizable. These steps can be seen in the algorithm in lines 1 to 5. The next step, on line 6, represents the loop where B is estimated through the IRLS method. The adopted stop criterion for this iterative method corresponds to a precision of 0,001, according to the literature [3]. Equation 3.3 can be divided in two parts, $(X^T W X)^{-1}$, the left part, B_l, and $X^T(W_i X B_i + y - \lambda_i)$, the right part, B_r. From line 9 to 14 the parameters that compose B_r are computed for every sample and the corresponding entry of B_r is also computed. To obtain B_l, it is necessary to iterate through each sample, a process that can be executed in parallel. To have the estimation of B it necessary to solve the system on line 20. Finally to

obtain the AIC score it is necessary to compute Equation 2.2. Since the likelihood is dependent on every sample, it is necessary to iterate through each one and add each sub-score. This loop can be executed in parallel and so a reduction is done over the final AIC score, finally just remains the addition of $2d$, where d represents the number of free parameters. Due to the floating point nature of the data present in these computations the main challenge is to have consistency in the operations performed. Since a floating point number is represented by a significand and an exponent, the conversion of exponents and the corresponding change in the significand, can lead to rounding errors that can propagate when high numerical precision is involved. To ensure consistent computations, in each operation changes in the numerical representation were made, and private copies of some variables were created for each thread, leading to additional reductions performed after those computations.

Algorithm 5 Problem-dependent parallel design - AIC score

1: $B \leftarrow$ Initialize Model Parameters /* $B = 0*$/
2: `#pragma omp for schedule (guided)`
3: **for** s=1 to N **do**
4: X $\leftarrow$ Initialize Matrix (dataset[s][q.SNP[i]]) /* $\forall$ *i,i = 1 to k* */
5: **end for**
6: **while** ! stop criterion (B.improvement < 0.001) **do**
7: /* $Computing\ B_r = X^T(W_iXB_i + y - \lambda_i)$*/
8: `#pragma omp for schedule (guided)`
9: **for** s=1 to N **do** do
10: $\lambda \leftarrow$ Compute Expected Values ($X[s], B$)
11: $y \leftarrow$ Get Disease State (dataset[s][M+1])
12: $W \leftarrow ComputeWeights(\lambda)$
13: $B_r \leftarrow ComposeB_rentry(W, X[s], B, y, \lambda)$
14: **end for**
15: /* $ComputingB_l = X^TWX$*/
16: `#pragma omp for schedule (guided)`
17: **for** s=1 to N **do**
18: $B_l \leftarrow ComposeB_lentry(W, X[s])$
19: **end for**
20: $B \leftarrow B_lB_r$
21: **end while**
22: `#pragma omp for reduction(+:$AIC$)`
23: **for** s=1 to N **do**
24: $lik \leftarrow$ Compute likelihood ($X[s], B$)
25: $AIC \leftarrow$ Compute AIC Score (lik)
26: **end for**
27: $AIC = AIC + 2d$
28: return AIC

3.5 Summary

In this chapter, in order to reduce execution time of epistasis detection methods, improvements to state of the art methods, namely NSGA-II, are presented and explained. The methods improved constitute the main bottle necks of NSGA-II, namely both objective score functions, AIC score and K2 score, the Fast Non-dominant sorting algorithm and the crowding distance assignment method, and finally the database of generated solutions to avoid repeated analysis. Additionally, to achieve reasonable times in the high order epistasis analysis, and to fully exploit CPU resources, two parallel designs of the proposed algorithm, based on [10], are introduced, a problem independent approach that represents a parallelization at the level of the solution generation, and a problem dependent approach that represents a parallelization at each solution, in the evaluation of each objective score.

Chapter 4

Performance evaluation of the proposed methods

In this chapter the results of the improvements to the serial design will be discussed, their viability and time improvement. Additionally the speedups and efficiency of the parallel approaches will be discussed.

For experimentation purposes, three problem instances with different characteristics (in terms of numbers of SNPs and case/control samples) have been considered:

- DB23x10000: a real-world breast cancer dataset composed of 10,000 samples (5,000 cases and 5,000 controls), each one characterized by 23 SNPs from the genes COMT, CYP19A1, ESR1, PGR, SHBG, and STS [40];
- DB1000x4000: a benchmark dataset containing 4,000 samples (2,000 controls and 2,000 cases) with 1,000 SNPs, generated by using the GAMETES software [41];
- DB31341x146: a benchmark dataset containing 146 samples (50 controls and 96 cases) with 31,341 SNPs, generated by using the GAMETES software [41].

The input parameters were configured according to the state of the art literature[3, 9]. The crossover probability was set to 50%, the mutation probability to 20%, and the mutation range to 30%. The population size and the stop criterion were established to 64 individuals and 100 generations for DB23x10000, 100 individuals and 1,500 generations for DB1000x4000, and 500 individuals and 2,000 generations for DB31341x146, due to the differences in the search space between each file, a different number of generations and a different population size is needed to reach the optimal solution in each dataset.

All experimental results were obtained by averaging at least five independent runs on a multicore multiprocessor system composed of two Intel Xeon Gold 6140 at 2.3GHz (a total of 36 physical cores in the system) with 25M Cache and 4x16GB DDR4-2666 RAM. CentOS 7.5 is used as operating system and the software tested in this research was compiled by using GCC 4.8.5. All experimental times presented are in seconds.

4.1 Evaluation of Design Features and Optimizations

In the following sub-sections experimental results of the proposed improvements will be presented and an analysis will be made. Each design feature will be tested with all three datasets as well as for epistasis interaction orders of 2, 4, 6, 8 and 10.

4.1.1 Database

To guarantee that an individual is not generated and evaluated more than once a database of previously generated solutions is needed, to guarantee an efficient algorithm it is necessary an efficient database. To improve this database, mainly the search time in the database, a hash table is proposed. In Table 4.1 the improvements in speed from a single array to a hash table can be seen. Experimental tests were run for epistasis sizes 2, 4,6, 8 and 10. The firsts column, of data, represents the total execution time for an implementation with database present in state-of-the-art method NSGA-II, the second column shows the time taken just searching in the database, and the third column represents the percentage of total time spent searching in this database. In this experiment the only different between both executions is the implementation of the database that stores previously generated solutions, everything else was kept the same in both experiments. The fourth, fifth and sixth columns represent the same values as the previous three columns but for an implementation having a hash table as a database. The times presented for a database implemented with a hash table are for a hash of size 1511, which from various sizes tested, proved to be the best compromise between small and large files, representing a trade off between a sparse populated hash, for small files, with a large amount of unused memory, and a densely populated hash, for large files, with relatively high search times.

Looking at the first and second columns of Table 4.1 it is possible to see that the database is a bottleneck and represents the vast majority of execution time in large files, reaching up to 99,5% of the execution time in DB31341x146, which the dataset that implies a higher number of total individuals generated due to its high search space. The implications of having a simple database are not so relevant when using smaller datasets, that imply solving for a smaller population and fewer generations, as it can be seen in the results for file DB23x10000, which is a small file in terms of SNP markers. Replacing the database with a hash table represents an improvement of up to 93,3% in the time dedicated to search for a solution in the database, in relation to the overall time. This change in database revealed to be the change necessary to run large files in a reasonable time, reducing 4000 seconds to just 25. For large files a hash table with a higher size could be used but for smaller files having a hash table too large represents unused memory, and so the balance was found at 1511, for which the execution times are presented. With the increase of epistasis interaction order the impact of the search time in the overall execution time diminishes, due to an increase in time spent in the computation of the objective scores.

DB31341x146						
Dim_Epi	Total time SOA	Time Database	% of total time	Total time hash table	Time Database	% of total time
k=2	4058.84	4038.89	99.51%	25.19	3.79	15.05%
k=4	5603.91	5473.17	97.67%	135.21	6.15	4.55%
k=6	6280.95	6088.41	96.93%	191.20	7.61	3.98%
k=8	8174.19	7875.38	96.34%	294.71	8.89	3.02%
k=10	9173.01	8260.17	90.05%	873.55	10.04	1.15%
DB23x10000						
Dim_Epi	Total time SOA	Time Database	% of total time	Total time hash table	Time Database	% of total time
k=2	0.07	0.01	15.68%	0.06	0.001	2.55%
k=4	1.95	0.19	9.62%	1.74	0.009	0.49%
k=6	9.14	0.31	3.37%	8.74	0.012	0.14%
k=8	77.43	0.39	0.5%	76.76	0.013	0.02%
k=10	173.90	0.47	0.27%	172.14	0.015	0.01%
DB1000x4000						
Dim_Epi	Total time SOA	Time Database	% of total time	Total time hash table	Time Database	% of total time
k=2	94.87	79.14	83.43%	35.77	0.13	0.37%
k=4	176.22	127.34	72.26%	51.98	0.18	0.35%
k=6	363.57	151.63	41.70%	211.15	0.24	0.11%
k=8	1398.49	164.46	11.76%	1223.96	0.28	0.02%
k=10	1965.50	193.68	9.85%	1749.85	0.36	0.02%

Table 4.1: Improvements in time for database.

4.1.2 Sorting

After every generation, the new population (composed by the old individual plus the offspring generated) needs to be sorted and assigned a crowding distance. This step is implicitly sequential and it needs to be executed as fast as possible. In Table 4.2 it is possible to see the improvements in time when sorting and assigning the crowding distance to the population. Experiments were ran for epistasis interaction orders of 2, 4, 6, 8 and 10. The first three data columns represent, respectively, the total execution time taken with a state-of-the-art sorting method, namely fast non-dominated sorting, the time spent just sorting the population and its percentage of total time. The last three columns represent the total execution time, the time spent sorting and assigning crowding distance, and the percentage of total time, when using the proposed bi-objective sorting algorithm. In this experiment the only test subject was the improvements in time made with the new sorting algorithm, all the other functions were kept the same

in both experiments.

Overall an increase in epistasis interaction order represents an in increase in the search time, due to a higher number of SNPs in each solution, which when searching for a solution in the database are necessary to compare with. Since DB31341x146 requires a higher population per generation higher times when sorting were obtained. For this dataset, the improvements represent at most a 25,9% reduction in total time, or an improvement of 29 times in the sorting time of the algorithm. As for the other two datasets, even though sorting took reasonably low time considerable improvements were obtained. For DB23x10000 improvements in sorting time reached 5.5 times, and for DB1000x4000 reached 5.9 times in the spent sorting the population and assigning a crowding distance to the solutions.

DB31341x146						
Dim_Epi	Total time SOA	Time Sort	% of total time	Total time Proposal	Time Sort	% of total time
k=2	32.959	9.119	27.67%	26.128	0.469	1.79%
k=4	141.914	11.384	8.02%	135.784	0.594	0.44%
k=6	194.134	11.607	5.98%	191.892	0.650	0.34%
k=8	242.856	21.370	8.80%	296.302	0.747	0.25%
k=10	854.741	24.405	2.86%	871.156	1.061	0.12%
DB23x10000						
Dim_Epi	Total time SOA	Time Sort	% of total time	Total time Proposal	Time Sort	% of total time
k=2	0.06	0.0092	14.69%	0.0567	0.00167	2.94%
k=4	1.75	0.0092	0.52%	1.740	0.00241	0.14%
k=6	8.84	0.0101	0.11%	8.766	0.00307	0.03%
k=8	76.19	0.0124	0.02%	76.803	0.00336	0.004%
k=10	173.57	0.0140	0.01%	173.618	0.00360	0.002%
DB1000x4000						
Dim_Epi	Total time SOA	Time Sort	% of total time	Total time Proposal	Time Sort	% of total time
k=2	37.56	0.296	0.79%	34.552	0.057	0.17%
k=4	54.27	0.256	0.47%	48.498	0.065	0.14%
k=6	214.06	0.269	0.13%	211.321	0.075	0.04%
k=8	1152.22	0.365	0.03%	1240.662	0.073	0.01%
k=10	1446.46	0.440	0.03%	1766.957	0.075	0.004%

Table 4.2: Improvements in time in population sorting and crowding distance assignment.

4.1.3 Objective Score Functions

Table 4.3 shows the execution times and the total time spent for the objective scores. The first two columns display the execution times when the objective scores are computed in the same way as the state-of-the-art approaches, with all the other functions optimized and improved. The goal of this experiment is to see the improvements in time made in both objective score functions. The three middle columns show the total time of execution, the time taken in the combined objective functions and the speedup relative to the previous version, respectively, of a AIC score computed with a dedicated solver, explained in Section 3.3.2 and K2 score implemented with a tree to determine the frequency of SNP combinations in the samples, explained in Section 3.3.1. Finally, the last three columns show the total time, the time taken in the combined objective scores and the speedup over the first times. These times were obtained using a recursive function to compute K2 score, proposed in Section 3.3.1, and for AIC score they were obtained using, besides the dedicated solver, a function that either computes this score using every sample or the frequency of every SNP marker combination, depending on the epistasis dimension and the size of the sample set, proposed in Section 3.3.2.

By using a tree and dedicated Gaussian solver to compute AIC score a considerable speedup can be obtained across all the test files. For DB31341x146 and epistasis interaction order of 10, it is possible to see the downside of using a tree in the calculation of the genotype frequencies. For the other two test files this effect is not visible due to the higher gains in the computation of AIC score. This two files obtain higher gains in the computation of AIC score due to its computation being more dependent on the sample size, and this two datasets having considerable more samples than DB31341x146, a difference of 146 to 10000 and 4000 samples.

As for the final proposed approach, where a recursive function computes the index of the array of genotype frequencies and AIC score is computed either based on the number of genotype combinations or in the number of samples in the dataset, higher improvements were obtained. This difference on how AIC score is computed justifies the higher speedups obtained for smaller epistasis sizes. When analysing the speedups obtained for the test file DB1000x4000 it is possible to see a large decrease in the speedup obtained from k=6 to k=8. For this file this is the moment when the number of possible genotype combinations, 3^k, is greater than the sample size, thus AIC is no longer computed based on each genotype combination and is instead computed with the traditional approach based on each sample. A higher speedup is not achieved for k=2 due to the genotype frequency determination, a recursive function is not the most efficient method for low epistasis interaction orders (k=2). For DB23x10000 the change in the AIC score computation method happens from k=8 to k=10, but since $3^8 = 6561$ the hypothetical speedup, with a change of method in the computation, is only $\frac{10000}{6561} = 1.52$, a reduced improvement for k=8 is verified in this file. As for DB31341x146 a higher speedup is verified for k=2, due to the improved method of AIC score computation. For k=4 the improvement is not significant, once again due to $3^4 = 81$, and the hypothetical speedup being only $\frac{146}{81} = 1.8$. In this dataset, due to the small sample size, improvements in K2 score are more noticeable, as well as small improvements in the computation of AIC score.

DB31341x146								
	SOA		Tree+Dedicated Solver			Recursive+Combinations		
Dim_Epi	Total time	Time Obj. Scores	Total time	Time Obj. Scores	Speedup obj. sco. time	Total time	Time Obj. Scores	Speedup obj. sco. time
k=2	490.46	483.34	204.82	197.97	2.44	25.35	18.78	25.74
k=4	1568.20	1557.87	554.59	543.54	2.87	135.77	126.41	12.32
k=6	2270.11	2257.76	887.81	872.95	2.59	192.84	180.72	12.49
k=8	2961.80	2948.38	1348.16	1348.16	2.19	296.04	282.19	10.45
k=10	6738.22	6720.27	6945.77	6924.33	0.97	871.44	855.64	7.85
DB23x10000								
	SOA		Tree+Dedicated Solver			Recursive+Combinations		
Dim_Epi	Total time	Time Obj. Scores	Total time	Time Obj. Scores	Speedup obj. sco. time	Total time	Time Obj. Scores	Speedup obj. sco. time
k=2	1.73	1.72	1.34	1.33	1.29	0.06	0.05	36.92
k=4	58.25	58.22	39.95	39.93	1.46	1.76	1.74	33.53
k=6	135.48	135.44	84.86	84.82	1.60	8.74	8.70	15.56
k=8	222.02	221.97	134.34	134.28	1.65	77.44	77.39	2.87
k=10	330.94	330.86	224.53	224.43	1.47	173.18	173.11	1.91
DB1000x4000								
	SOA		Tree+Dedicated Solver			Recursive+Combinations		
Dim_Epi	Total time	Time Obj. Scores	Total time	Time Obj. Scores	Speedup obj. sco. time	Total time	Time Obj. Scores	Speedup obj. sco. time
k=2	395.56	395.06	294.91	294.47	1.34	38.46	38.05	10.38
k=4	1196.48	1195.80	967.80	967.12	1.24	48.84	48.25	24.78
k=6	3716.77	3715.91	1114.54	1113.67	3.34	214.20	213.42	17.41
k=8	3645.50	3644.58	2165.41	2163.94	1.68	1235.53	1234.63	2.95
k=10	5925.45	5924.29	4068.80	4062.42	1.46	1765.71	1764.72	3.36

Table 4.3: Improvements in time for the combined time of objective score functions.

Table 4.4 presents the percentages of total time taken by each objective scores, for the same implementations as in the previous table. The last column represents the determination of the genotype frequency, which is a common part between AIC and K2, for k=2 and 4 in DB31341x146, for k=2, 4, 6 and 8 in DB23x10000, and for k=2, 4 and 6 in DB1000x4000.

The first two data columns present the percentage of time taken each objective score for state-of-the-art implementations of K2 and AIC scores. Across all three datasets, for low epistasis orders AIC score

takes most of the execution time, but, with the increase of the number of possible genotype combinations (3^k), K2 starts to use more of the execution time. In the middle data columns, which represent the percentage of time taken by the objective scores for an implementation with a tree and a Gaussian solver, the tendency is the same as in the previous implementations, but due to more significant improvements in AIC, K2 score presents higher percentages. For the final proposed implementation, presented in the last 3 columns, this change is not linear due to the different methods used in the computation of AIC score. In the last column, the weight, in the execution time, of the determination of the frequency of each genotype combination, can be noted. A large percentage of execution time happens when both objective scores depend on this frequency, which reduces their execution times. Due to a higher number of samples in the datasets DB1000x4000 and DB23x10000, over the DB31341x146, which contains only 146 samples, longer computations in AIC score are necessary, and thus K2 does not take so much importance in the total execution time as in DB31341x146.

DB31341x146							
Dim_Epi	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% SNP comb.
k=2	3.25%	95.30%	2.23%	94.43%	0.46%	41.70%	31.91%
k=4	2.33%	97.01%	3.05%	94.96%	0.48%	84.99%	7.62%
k=6	4.40%	95.06%	11.28%	87.05%	2.86%	84.12%	6.74%
k=8	15.06%	84.48%	36.95%	63.05%	16.54%	66.30%	12.48%
k=10	50.12%	49.61%	84.78%	14.91%	50.49%	20.63%	27.06%
DB23x10000							
Dim_Epi	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% SNP comb.
k=2	16.47%	82.88%	7.91%	91.25%	0.08%	2.57%	78.25%
k=4	18.36%	81.59%	5.83%	94.10%	0.22%	19.34%	79.06%
k=6	18.90%	81.07%	5.39%	94.56%	0.40%	69.50%	29.69%
k=8	18.80%	81.17%	8.58%	91.38%	0.41%	94.72%	4.81%
k=10	21.82%	78.16%	25.11%	74.84%	2.01%	94.73%	3.22%
DB1000x4000							
Dim_Epi	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% of K2 score	% of AIC score	% SNP comb.
k=2	15.68%	84.19%	5.89%	93.96%	0.07%	2.21%	96.66%
k=4	11.20%	88.74%	3.53%	96.40%	0.24%	26.64%	71.94%
k=6	6.79%	93.19%	5.71%	94.21%	0.39%	82.82%	16.42%
k=8	11.82%	88.15%	9.82%	90.11%	0.73%	95.50%	3.69%
k=10	16.62%	83.36%	24.75%	75.091%	4.60%	90.59%	4.76%

Table 4.4: Evolution of the percentage of the objective score time in relation to the overall execution time

4.2 Evaluation of Parallel Performance

In the following two sections the performance and efficiency of both parallel designs, and a comparison between both designs and their scalability, will be presented and evaluated. A comparison between parallelization schedules will be presented. Additionally a comparison with SOA method NSGA-II, showing the total improved time, will be made.

4.2.1 Problem-independent design

Table 4.5 presents the mean speedups and efficiencies of the independent parallelization design approach 4.2.1, for 4, 8, 16 and 32 cores. Across all datasets effective reductions in time were obtained, with efficiencies reaching 97.5% high order epistasis interactions (k=10).

For DB31341x146, the problem with lower number of samples, and consequentially with less potential for parallelism than the other datasets, reasonable speedup and efficiencies were obtained for k=2. For k=4 the speedups obtained are good but the scalability is reasonable. For higher orders (k=6, 8 and 10), the speedups obtained are good and present a good scalability, specially for k=10. For DB23x10000, the problem with the highest number of samples and consequentially with a greater potential for parallelism, for k=4 to 10, mean speedups of 3.9 (4 cores), 7.6 (8 cores), 14.5 (16 cores) and 27 (32 cores) were obtained, presenting really good scalability. It is worth noting that a speed up of 31.2 was obtained for k=10 with 32 cores, revealing good scalability for problems of high epistasis interaction orders. For k=2, due to an already reduced execution time, of under 0.1 seconds, worse speedups were obtained, revealing a higher relation of overheads introduced to gains obtained with a parallelization. DB1000x4000 is a dataset characterized by some load balancing issues and a lower number of samples than DB23x10000. Despite these constraints good speedups were obtained, and except for k=2, a problem of an already small dimensions, reasonably good scalability was obtained. Average speedups of 3.8 (4 cores), 7.2 (8 cores), 13.6 (16 cores) and 23 (32 cores) were obtained for k=4 to 10. Across the three datasets this parallelization design presents an average efficiency of 53.2% for k=2, 77.2% for k=4, 87.8% for k=6, 90.8% for k=8 and 93.9% for k=10, representing a good parallelization design and a good approach to tackle problems of even higher dimension than k=10.

DB31341x146								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	3.12	77.92%	4.97	62.08%	6.62	41.38%	7.60	23.73%
k=4	3.59	89.71%	6.70	83.70%	11.22	70.14%	16.51	51.59%
k=6	3.86	96.46%	7.47	93.40%	13.47	84.20%	24.64	77.01%
k=8	3.77	94.35%	7.39	92.35%	14.10	88.16%	24.67	77.10%
k=10	3.96	98.94%	7.78	97.23%	15.34	95.89%	28.66	89.56%
DB23x10000								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	2.74	68.54%	3.98	49.80%	4.74	29.61%	4.69	14.66%
k=4	3.85	96.30%	7.32	91.49%	13.22	82.61%	21.38	66.80%
k=6	3.90	97.58%	7.55	94.42%	14.18	88.60%	26.30	82.19%
k=8	3.91	97.87%	7.63	95.40%	14.86	92.88%	29.21	91.29%
k=10	3.94	98.61%	7.88	98.50%	15.67	97.96%	31.21	97.54%
DB1000x4000								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	3.72	92.96%	5.66	70.75%	10.14	63.39%	17.76	55.50%
k=4	3.68	92.03%	6.20	77.48%	12.93	80.84%	21.19	66.21%
k=6	3.85	96.25%	7.45	93.07%	13.33	83.30%	21.49	67.15%
k=8	3.88	97.09%	7.59	94.82%	14.15	88.46%	25.53	79.79%
k=10	3.85	96.30%	7.46	93.23%	14.09	88.07%	23.96	74.88%

Table 4.5: Speedup and efficiency values for independent parallelization approach

When implementing a parallelization one thing to consider is the loop scheduling, i.e. the assignment of iterations in a parallelized loop to each execution thread. When using *Openmp* different schedules are available. Between all those schedules the most efficient schedules are the dynamic schedules due to the random nature of the problem, so the most effective schedules the dynamic and guided, thus their respective speedups are presented in Table 4.6. Dynamic schedule was ran with a chunk size of one. As it can be seen the differences are not considerable, except for DB31341x146, which has a higher number of iterations to parallelize, due to a higher population in each generation, and an AIC score computationally less heavy due to a smaller sample size. In this dataset for the lower epistasis interaction orders, of k=2 and 4, overheads in thread management, for the dynamic schedule, can be noted, due to the small chunk size of one. Otherwise they are complimentary and picking one over the other is not straight forward.

DB31341x146								
	4 cores SU		8 cores SU		16 cores SU		32 cores SU	
Dim_Epi	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided
k=2	3.12	3.32	4.97	5.64	6.62	8.34	7.60	10.30
k=4	3.59	3.63	6.70	6.94	11.22	12.37	16.51	19.68
k=6	3.86	3.66	7.47	7.61	13.47	13.67	24.64	24.86
k=8	3.77	3.86	7.39	7.62	14.10	13.81	24.67	24.23
k=10	3.96	3.996	7.78	7.75	15.34	15.07	28.66	27.74
DB23x10000								
	4 cores SU		8 cores SU		16 cores SU		32 cores SU	
Dim_Epi	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided
k=2	2.74	2.65	3.98	3.88	4.74	4.77	4.69	4.92
k=4	3.85	3.75	7.32	7.30	13.22	13.12	21.38	21.61
k=6	3.90	3.89	7.55	7.56	14.18	14.16	26.30	26.31
k=8	3.91	3.90	7.63	7.64	14.86	14.86	29.21	29.27
k=10	3.94	3.95	7.88	7.85	15.67	15.70	31.21	31.18
DB1000x4000								
	4 cores SU		8 cores SU		16 cores SU		32 cores SU	
Dim_Epi	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided	Dynamic	Guided
k=2	3.72	3.93	5.66	4.92	10.14	10.59	17.76	19.45
k=4	3.68	3.80	6.20	5.51	12.93	11.56	21.19	18.89
k=6	3.85	3.77	7.45	7.47	13.33	13.36	21.49	21.63
k=8	3.88	3.83	7.59	7.69	14.15	14.35	25.53	25.26
k=10	3.85	3.87	7.46	7.48	14.09	14.00	23.96	24.25

Table 4.6: Speedup values for schedule dynamic and guided used in the independent parallelization approach

4.2.2 Problem-dependent design

The second parallelization design is a parallelization at the solution level, i.e., at the objective score level, since those are the most time consuming functions of the total execution time. The potential parallelism of this approach is directly related to the number of case/control samples in the dataset and with the epistasis interaction order under study (k). Table 4.7 presents the average speedups and efficiencies for the experiments ran on this problem-dependent design, for 4, 8, 16 and 32 cores, across all three datasets. Since this parallelization is at a much lower level, lower speedups, and consequently efficiencies, were expected due to a higher number of computational constraints, data dependencies and synchronization needs. DB31341x146 presents a problem with a less demanding AIC score due to a smaller number of samples, 146, so lower speedups are observed. This problem presents several speedups under 1,

meaning that the benefit of a parallel implementation does not overcome the overheads and synchronization barriers introduced and the serialization of part of the computations to preserve data consistency, and thus making this considerable more complex paralellization not worth it for datasets with a low number of samples. For problems with a higher number of samples, like DB1000x4000 and DB23x10000, for high order epistasis interaction orders this design presents some reasonable efficiencies but presents with poor scalability, with a 32 core paralellization presenting a maximum in efficiency of 43%, and an average of 33.7%. For k=2 the average efficiency in this two datasets was 26%, for k=4 39%, for k=6 50%, for k=8 55% and for k=10 57%. Once again for low epistasis interaction orders, this parallelization design presents poor speedups and consequentially poor scalability. Overall the best efficiencies obtained are in DB23x10000, for k=10, with 73.3% (4 cores), 67.9% (8 cores), 60.2% (16 cores) and 43.3% (32 cores), due to being a problem highly dominated by the heavy computations of the AIC score due to the large number of case/control samples.

DB31341x146								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	0.92	23.01%	0.95	11.83%	0.67	4.17%	0.39	1.23%
k=4	1.28	32.07%	1.29	16.09%	0.97	6.06%	0.58	1.81%
k=6	1.48	37.00%	1.60	20.03%	1.18	7.37%	0.67	2.08%
k=8	1.46	36.55%	1.66	20.79%	1.34	8.36%	0.85	2.65%
k=10	1.27	31.69%	1.55	19.39%	1.63	10.18%	1.35	4.22%
DB23x10000								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	1.53	38.24%	2.50	31.21%	3.88	24.28%	4.82	15.05%
k=4	2.11	52.71%	3.62	45.26%	6.20	38.76%	7.93	24.78%
k=6	2.55	63.69%	4.73	59.14%	8.41	52.56%	11.47	35.83%
k=8	2.64	65.90%	5.06	63.21%	9.30	58.15%	13.24	41.39%
k=10	2.93	73.25%	5.43	67.85%	9.64	60.23%	13.87	43.34%
DB1000x4000								
	4 cores		8 cores		16 cores		32 cores	
Dim_Epi	SU	EF	SU	EF	SU	EF	SU	EF
k=2	1.45	36.22%	2.42	30.26%	3.48	21.77%	3.23	10.10%
k=4	2.05	51.24%	3.65	45.68%	5.68	35.48%	5.82	18.19%
k=6	2.35	58.65%	4.75	59.37%	7.37	46.07%	7.91	24.72%
k=8	2.70	67.39%	4.92	61.45%	8.26	51.62%	8.94	27.93%
k=10	2.80	69.91%	4.96	61.94%	7.95	49.67%	9.23	28.85%

Table 4.7: Speedup and efficiency values for dependent parallelization approach

Figures 4.1, 4.2 and 4.3 present the comparison between the parallel-dependent and parallel--independent approaches, for the files DB31431x146, DB23x10000 and DB4000x1000, respectively. Each figure shows the results for 4, 8, 16 and 32 cores, for epistasis interactions sizes of 2, 4, 6, 8 and 10.

A problem-dependent parallelization is not very effective on an input file with the characteristics of DB31341x146, due to the small sample size, of just 146 samples. This small sample size makes the computations, of the objective scores, not heavy enough to overcome the weight of the overheads introduced by the parallelization. Due to the large number of SNP markers, 31 thousand, the search space is very large and thus it is necessary to work with a large population for a higher number of generations. This contributes for a reasonably good parallelization for low order epistasis interactions and a good parallelization for high epistasis interaction orders, with an efficiency of 90% for an order of 10.

The input file DB23x10000 represents a file with a low number of SNP markers and a high number of samples. This represents a relatively small search space but objective score functions that take longer to compute. Thus contributing for the best results in both parallelization approaches of the three input files. The problem dependent design has a reasonable scalability for high order epistasis interactions. The problem independent design has a really good scalability for all epistasis orders, with the exception being k=2 which is a problem whose solution is found in a really quickly and so does not have much room for improvement with a parallel design.

The third input file represents a more balanced problem but on a higher number of SNP markers. The results for this file represent a reasonably good scalability for the parallel independent design, even though this file having a load imbalance. As for the dependent design the scalability is poor since it reaches its upper limit with 16/32 cores.

The limiting factor for a parallel dependent design is data consistency. Since it is a parallelization at a lower level it is necessary to ensure a consistent access to the variables in memory. With the increase of the number of cores this becomes a bottleneck. In the objective score function K2, where the computations are simpler and more related to memory reads and writes, for a higher number of cores engaged, an increase in the total time spent in that function is seen.

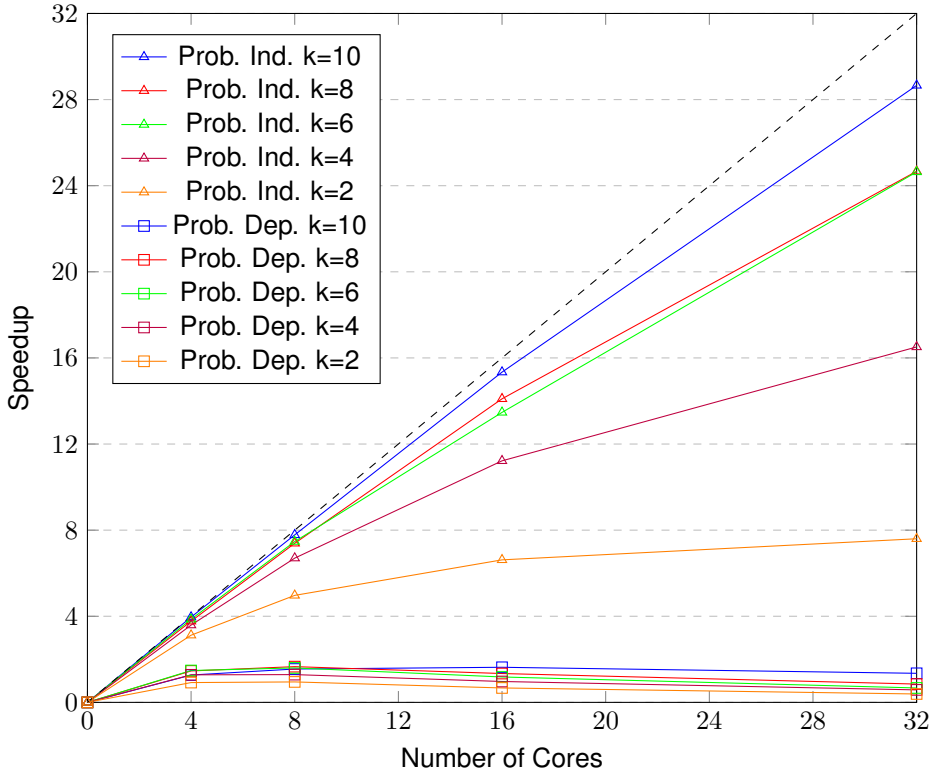


Figure 4.1: Speedup comparison, for DB31341x146, between problem-dependent and problem-independent approaches.

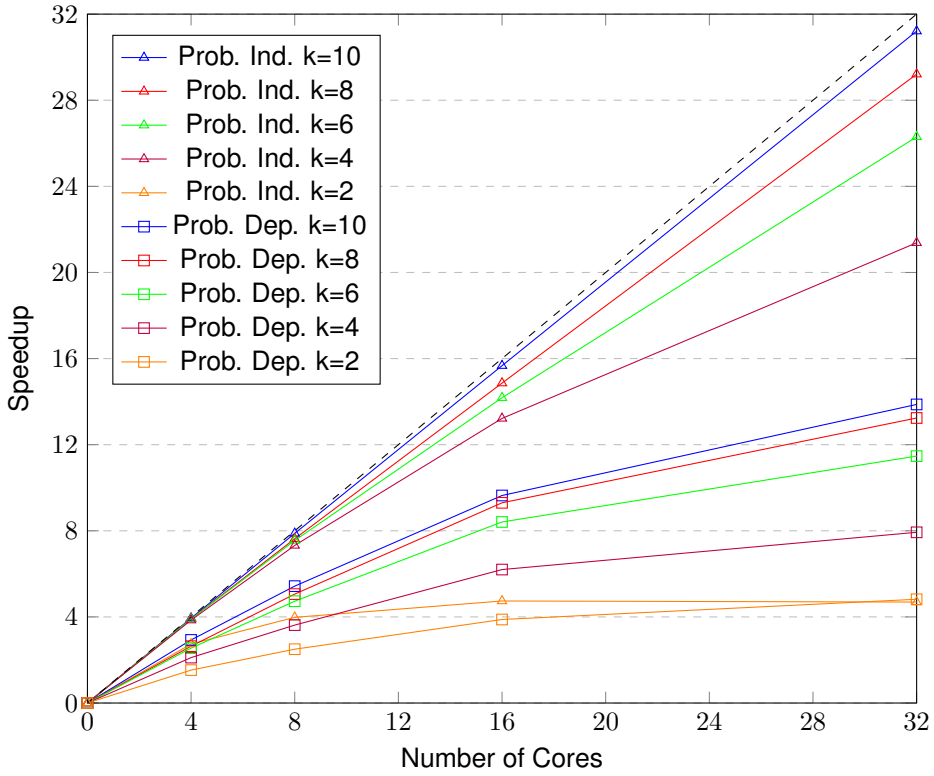


Figure 4.2: Speedup comparison, for DB23x10000, between problem-dependent and problem-independent approaches.

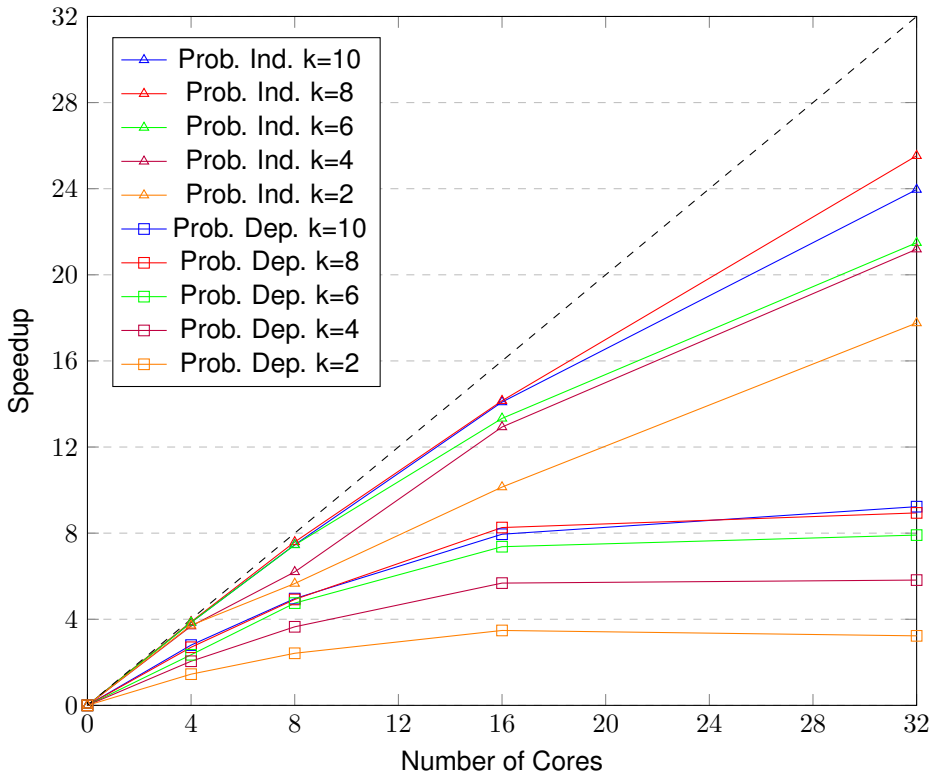


Figure 4.3: Speedup comparison, for DB4000x1000, between problem-dependent and problem-independent approaches.

4.3 Total performance evaluation

Due to the main goal of this work being the improvement of execution time of the state-of-the-art multiobjective tools, Table 4.8 presents a comparison between the execution times of the state-of-the-art method NSGA-II [32] with a serial implementation of the proposed improvements and a 36 core parallelization of the proposed problem-independent design. Additionally the relative speedup, of both implementations, with NSGA-II are also presented, representing the total speedup obtained with this work.

For the proposed serial implementation these speedups range from 1,93 to 196. DB31341x146 presents the higher speedups due to the larger benefit from an improved database of solutions generated and tested. Across all three datasets lower epistasis interaction orders present larger speedups than high epistasis interaction orders due to improved AIC score computation method.

For the proposed parallelization design, the problem-independent approach, the speedups range from 60 to 2036. An average total speedup of 834 for k=2, 818 for k=4, 679 for k=6, 393 for k=8 and 261 for k=10 was obtained with the proposed methods. With this parallelization it was possible to achieve an average of 1.5 seconds in execution time for k=2, 3.3 seconds for k=4, 5.45 seconds for k=6, 19.4 for k=8, and 33.3 for k=10, presenting a reduction of several hours of computation to a few dozens of seconds in the worst case for high epistasis interaction orders (k=10).

DB31341x146					
	k=2	k=4	k=6	k=8	k=10
NSGA-II	4596.83	6873.16	8293.05	10756.80	17420.79
Serial	23.41	133.05	189.49	292.21	870.66
Speedup	196.37	51.66	43.76	36.81	20.01
36 cores	2.26	6.42	6.96	10.62	27.43
Speedup	2036.27	1070.67	1191.85	1013.24	635.21
DB23x10000					
	k=2	k=4	k=6	k=8	k=10
NSGA-II	1.83	58.82	136.44	222.86	331.30
Serial	0.05	1.76	8.77	76.88	172.00
Speedup	33.55	33.48	15.56	2.90	1.93
36 cores	0.02	0.08	0.33	2.63	5.51
Speedup	112.94	704.00	412.01	84.81	60.17
DB1000x4000					
	k=2	k=4	k=6	k=8	k=10
NSGA-II	480.19	1336.52	3923.73	3653.63	5922.10
Serial	32.74	52.32	211.24	1222.92	1753.20
Speedup	16.24	30.19	18.57	2.99	3.38
36 cores	1.36	1.97	9.05	44.95	67.03
Speedup	354.11	678.68	433.46	81.28	88.35

Table 4.8: Execution times of NSGA-II vs serial implementation of the proposed improvements vs 36 cores parallelization of proposed problem-independent design

4.4 Evaluation of Solution Quality

Two types of analysis can be done to determine solution quality, computing the hypervolume indicator to measure multi-objective performance and computing biological metrics such as Recall, Precision and F-measure [3].

To compute the hypervolume all the results from all the independent runs, from all the analysed algorithms, are gathered. With those results the Nadir point can be determining by searching for the worst results in both objective scores. Having the Nadir point the hypervolume is computed as figure 4.4 shows, by summing the areas from the points of the Pareto front approximation to the Nadir point. The higher the value of the better, the hypervolume is a measurement to maximize. Since the values in the objective scores have different ranges they have to be normalized to a Nadir point with coordinates (1,1). Table 4.9 shows the values of hypervolume, percentually, of the proposed algorithm, previously described, with the state of the art approach NSGA-II. Additionally P-value, from the Wilcoxon-Mann-Whitney tests, is also shown, when the P-value is $<$0.050 there are statistically significant differences,

when it is $>$0.050 the differences are non-significant. Table 4.10 shows the same comparison for the state-of-the-art approach MACOED. Excluding the file DB23x10000 where it reports exactly the the same value in both approaches, superior results in terms of hypervolume values where achieved. Additionally a time comparison is shown, even when the results are similar a reduction in execution time of 28 times was achieved.

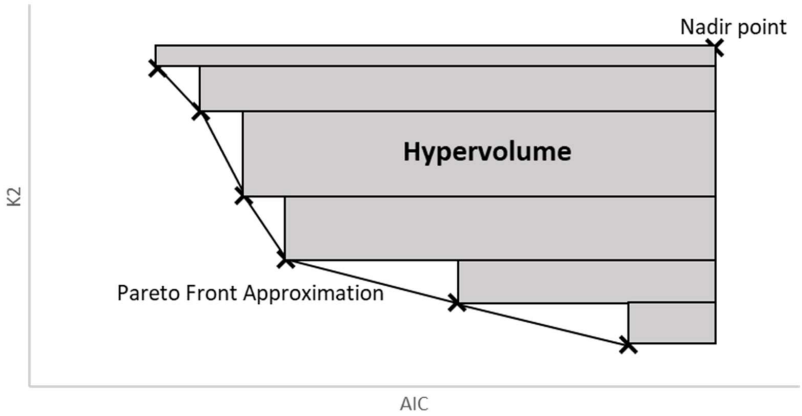


Figure 4.4: Explanation of the Hypervolume computation.

DB31341x146					
	k=2	k=4	k=6	k=8	k=10
Proposal	79.293%	74.421%	68.398%	64.180%	52.649%
NSGA-II	79.327%	73.940%	71.647%	61.523%	54.305%
P-value	0.335	0.647	0.188	0.223	0.588
DB23x10000					
	k=2	k=4	k=6	k=8	k=10
Proposal	99.787%	99.612%	98.631%	99.128%	99.883%
NSGA-II	99.792%	99.613%	98.632%	99.129%	99.882%
P-value	1.000	1.000	1.000	0.982	0.446
DB1000x4000					
	k=2	k=4	k=6	k=8	k=10
Proposal	90.788%	89.119%	78.379%	84.385%	81.340%
NSGA-II	90.783%	89.169%	78.052%	82.948%	81.510%
P-value	0.293	0.719	0.741	0.088	0.597

Table 4.9: Hypervolume comparisons with NSGA-II and statistical evaluation

	DB31431x146	DB23x10000	DB1000x4000
Proposal	79.293%	99.787%	90.788%
MACOED	36.834%	99.787%	85.858%
P-Value	1.32E-11	1.00	4.56E-10
Execution time			
Proposal	25.35	0.06	38.46
MACOED	567.28	1.69	407.99

Table 4.10: Hyper volume comparison with MACOED, for k=2, statistical evaluation and time comparison between serial implementations.

For a biological analysis of solution quality the formulas for the metrics used are presented in Equations 4.1, 4.2 and 4.3 [3], where TP stands for true positives, the solutions found that are in fact real solutions, FN for false negatives, the solutions that where discarded and are in fact real solutions, and FP for false positive, the solution that where considered real solutions and in fact are not. Since the real solutions of the test files are not known those measurements could not be computed for the presented approach. But since there is no statistical difference with NSGA-II it is reasonable to assume that similar values could be obtained. Table 4.11 presents the comparison of those metrics between NSGA-II and MACOED, for the synthetic datasets employed in [3] to measure biological performance. The values represent mean$\pm$standard deviation. With both Recall and F-measure better values are achieved, and in Precision the same value is achieved but with better precision since the standard deviation is lower.

$$Recall = \frac{TP}{TP + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$F - measure = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{4.3}$$

Algorithm	Recall	Precision	F-measure
NSGA-II	0.91$\pm$0.14	0.95$\pm$0.11	0.93$\pm$0.13
MACOED	0.84$\pm$0.31	0.95$\pm$0.12	0.86$\pm$0.28

Table 4.11: Biological values of Recall, Precision and F-measure for state-of-the-art methods MACOED and NSGA-II.

4.5 Summary

In this chapter, the experimental results from the proposed methods in Chapter 3 are presented and analysed. The methods experimentally tested and analyzed are proposals to improve state of the art methods in order to tackle high order epistasis detection, and to accelerate low order analysis. Thus experiments of each proposed method includes comparisons with state of the art method NSGA-II. Additionally to further improve execution times and fully exploit the CPU capabilities, experiments of the two proposed parallel implementations, proposed in Sections 3.4.1 and 3.4.2, are presented. Their speedup, efficiency and scalability is analysed, and a comparison between the two is made. An analysis of the total performance of the proposed methods is made, as is a brief analysis of solution quality.

Experiments were ran on a multicore multiprocessor system composed of two Intel Xeon Gold 6140 at 2.3GHz (a total of 36 physical cores in the system) with 25M Cache and 4x16GB DDR4-2666 RAM, with CentOS 7.5 as the operating system. Three datasets with different characteristics were used, one with 23 SNPs a 10000 samples, a second with 1000 SNPs and 4000 samples, and the third with 31341 SNPs and 146 samples. In order to evaluate epistasis detection times from low to high order interactions, experimental tests were ran for epistasis interaction orders of 2, 4, 6, 8 and 10.

The results presented throughput this chapter allow to conclude that proposed methods allow high epistasis interaction orders to be tested in reasonable times and present a considerable reduction in the time taken to analyze lower epistasis interaction orders.

Chapter 5

Conclusions

Epistasis detection is a hard to tackle problem, with its complexity increasing exponentially with the increase of interaction order. With the growing number of GWAS, and consequently genetics research, epistasis detection has become an increasingly important problem to solve. Hence the importance of reducing execution times without losing solution quality.

To tackle this problem state-of-the-art multiobjective epistasis detection methods were revised and improved in high-order scenarios. Additionally two parallel implementations were introduced to exploit the opportunities of a CPU based method in a multicore multiprocessor system based on the latest generation of Intel Xeon CPU architectures. Real-world and benchmark problem instances have been used to evaluate the parallel performance and solution quality for epistasis interaction orders of k = 2, 4, 6, 8 and 10.

In terms of a serial implementation improvements of 196x were achieved, with the main contribution being a high reduction on the time taken in the computation of AIC score, with a theoretical limit of $\frac{Sample\_Size}{3^k}$. This means that the higher the sample size, the higher the gains with this method. This opens a door to an increase in the size of these important biological studies. Significant gains were also achieved in the management of the database containing all the evaluated solutions, contributing to a possible increase in the number of SNP markers in the dataset without a decrease in performance. A new method of bi-objective sorting was also introduced with important relative gains over the previous fast non-dominant sort.

As for a parallel implementation, two designs were explored, a problem-independent and a problem-dependent. The problem-independent design proved to have the higher gains and scalability, reaching speedups of 31.21 with 32 cores, for epistasis interactions of k=10. For lower epistasis interaction order speedups of 17.76 were obtained, these speedups were not because of the improvements already made in the serial implementation, and thus it was a small problem to be paralellized. In absolute values the times obtained were in the order of just a few seconds. In terms of scheduling the two most efficient were dynamic and guided, the two being complimenting each other in different situations.

The second parallelization design, a problem-dependent approach, at the objective score level, revealed to be the less efficient and with poor scalability. Besides being harder to implement due to data

consistency problem, in reproducing operations due to rounding differences in floating point numbers representation, it also has a difficult data consistency management. This leads to a parallel-dependent implementation being a less recommendable approach.

All the improvements were made without losing solution quality over the state-of-the-art methods NSGA-II and MACOED.

5.1 Future Work

Future work over this problem might include an improvement over the AIC score, mainly over the way the likelihood is computed, due to it being the main bottle neck in the overall execution. Additionally improvements over solution quality can be made. Another idea that can be followed is identifying interaction patterns from $k-1$ to k in order to reduce the number of candidate solutions to be considered. In other contexts parallel design integrating hardware co-processors (GPUs, FPGAs, ...) can be explored.

Bibliography

[1] Moore, J.H., Hahn, L.W., Ritchie, M.D., Thornton, T.A., White, B.C., "Application of genetic algorithms to the discovery of complex models for simulation studies in human genetics," *Proceedings of the Genetic and Evolutionary Computation Conference. Genetic and Evolutionary Computation Conference; 1150-1155. Epub Jul 1*, 2002.

[2] Greene, C. S., Himmelstein, D. S., Moore, J. H., "A model free method to generate human genetics datasets with complex gene-disease relationships," *Pizzuti C., Ritchie M.D., Giacobini M. (eds) Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. EvoBIO 2010. Lecture Notes in Computer Science, vol 6023. Springer, Berlin, Heidelberg*, 2010.

[3] Jing, P., Shen, H., "Macoed: a multi-objective ant colony optimization algorithm for snp epistasis detection in genome-wide association studies," *Bioinformatics 31(5):634–641*, 2015.

[4] Rieger, R., Michaelis, A., Green, A., "A glossary of genetics and cytogenetics," *Springer*, 1968.

[5] Mackay, T.F., "Epistasis and quantitative traits: using model organisms to study gene-gene interactions," *Nature Reviews Genetics 15(1): 22–33*, 2014.

[6] Moore, J.H. et al., "Bioinformatics challenges for genome-wide association studies," *Bioinformatics, 26, 445–455*, 2010.

[7] Shang, J., Zhang, J., Sun, Y., Liu, D., Ye, D., Yin, Y., "Performance analysis of novel methods for detecting epistasis," *BMC Bioinformatics 2011 12:475*, 2011.

[8] Ritchie, M.D., "Finding the epistasis needles in the genome-wide haystack," *Epistasis, Methods in Molecular Biology vol. 1253). Springer, pp. 19–33*, 2014.

[9] Gallego-Sánchez, D., Granado-Criado, J.M., Santander-Jiménez, S., Rubio-Largo, A., Vega-Rodríguez, M.A., "Parallel multi-objective optimization for high-order epistasis detection," *Algorithms and Architectures for Parallel Processing, LNCS, volume 10393. Springer International Publishing, pp. 523–532*, 2017.

[10] Talbi, E.G., "Parallel evolutionary combinatorial optimization," *Springer Handbook of Computational Intelligence. Springer, pp. 1107–1125*, 2015.

[11] Luna, F., Alba, E., "Parallel multiobjective evolutionary algorithms," *Springer Handbook of Computational Intelligence. Springer, pp. 1017–1031.*, 2015.

[12] Niel, C., Sinoquet, C., Dina, C. Rocheleau, G., “A survey about methods dedicated to epistasis detection,” *Front Genet; 6: 285*, 2015.

[13] Manolio, T.A., Collins, F.S., Cox, N.J., Goldstein, D.B., Hindorff, L.A., Hunter, D.J., McCarthy, M.I., Ramos, E.M., Cardon, L.R., Chakravarti, A., Cho, J.H., Guttmacher, A.E., Kong, A., Kruglyak, L., Mardis, E., Rotimi, C.N., Slatkin, M., Valle, D., Whittemore, A.S., Boehnke, M., Clark, A.G., Eichler, E.E., Gibson, G., Haines, J.L., Mackay. T.F., McCarroll, S.A., Visscher, P.M., “Finding the missing heritability of complex diseases,” *Nature 461: 747-753*, 2009.

[14] Phillips, P. C., “Epistasis—the essential role of gene interactions in the structure and evolution of genetic systems,” *Nat Rev Genet. 2008 Nov; 9(11): 855–867*, 2008.

[15] Kam-Thong, T., Czamara, D., Tsuda, K., Borgwardt, K., Lewis, C. M., Erhardt-Lehmann, A., Hemmer, B., Rieckmann, P., Daake, M., Weber, F., Wolf, C., Ziegler, A., Ptz, B., Holsboer, F., Schlkopf, B., Mller-Myhsok, B., “Epiblaster-fast exhaustive two-locus epistasis detection strategy using graphical processing units,” *European Journal of Human Genetics 19(4):461-471*, 2010.

[16] Wan, X., Yang, C., Yang, Q., Zhao, H., Yu, W., “The complete compositional epistasis detection in genome-wide association studies,” *BMC Genetics 14:7*, 2013.

[17] Gonzalez-Dominguez, J., Wienbrandt, L., Kassens, J. C., Ellinghaus, D., Schimmler, M., Schmidt, B., “Parallelizing epistasis detection in gwas on fpga and gpu-accelerated computing systems,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics 12(5):982-994*, 2015.

[18] Wang, Z., Wang, Y., Tan, K., Wong, L., Agrawal, D., “eCEO: an efficient Cloud Epistasis computing model in genome-wide association study,” *Bioinformatics, 27(8), 15 April 2011, Pages 1045–1051,*, 2011.

[19] Gyenesei, A., Moody, J., Laiho, A., Semple, C.A.M., Haley, C.S., Wei, W., “Biforce toolbox: powerful high-throughput computational analysis of gene–gene interactions in genome-wide association studies,” *Nucleic Acids Res. 40(Web Server Issue): W628-W632*, 2012.

[20] Wienbrandt, L., Kässens, J. C., Hbenthal, M., Ellinghaus, D., “Fast genome-wide third-order snp interaction tests with information gain on a low-cost heterogeneous parallel fpga-gpu computing architecture,” *Procedia Computer Science 108: 596-605*, 2017.

[21] González-Domínguez, J., Schmidt, B., “Gpu-accelerated exhaustive search for third-order epistatic interactions in case–control studies,” *Journal of Computational Science 8: 93-100*, 2015.

[22] Kässens, J. C., Wienbrandt, L., González-Domínguez, J., Schmidt, B., Schimmler, M., “High-speed exhaustive 3-locus interaction epistasis analysis on fpgas,” *Journal of Computational Science 9 131–136*, 2015.

[23] Crawford, L., Zeng, P., Mukherjee, S., Zhou, X., “Detecting epistasis with the marginal epistasis test in genetic mapping studies of quantitative traits,” *PLOS Genetics 13(7): e1006869*, 2017.

[24] Jünger, D., Hundt, C., González-Domínguez, J., Schmidt, B., “Speed and accuracy improvement of higher-order epistasis detection on cuda-enabled gpus,” *Cluster Computing 20(3):1899-1908*, 2017.

[25] Mathew, B., Léon, J., Sannemann, W., Sillanp, M.J., “Detection of epistasis for flowering time using bayesian multilocus estimation in a barley magic population,” *Genetics 208(2): 525-536*, 2018.

[26] Molinaro, A.M., Carriero, N., Bjornson, R., Hartge, P., Rothman, N., Chatterjee, N., “Power of data mining methods to detect genetic associations and interactions,” *Hum Hered. 72(2): 85-97*, 2011.

[27] Verma, S.S., Lucas, A., Zhang, X., Veturi, Y., Dudek, S., Li, B., Li, R., Urbanowicz, R., Moore, J.H., Kim, D., Ritchie, M.D., “Collective feature selection to identify crucial epistatic variants,” *BioData Mining 11(5)*, 2018.

[28] Uppu, S., Krishna, A., Gopalan, R.P., “A deep learning approach to detect snp interactions,” *Journal of Software 11(10): 965-975*, 2016.

[29] Mieth, B., Kloft, M., Rodríguez, J.A., Sonnenburg, S., Vobruba, R., Morcillo-Suárez, C., Farré, X., Marigorta, U.M., Fehr, E., Dickhaus, T., Blanchard, G., Schunk, D., Navarro A., Müller, K.R., “Combining multiple hypothesis testing with machine learning increases the statistical power of genome-wide association studies,” *Scientific Reports 6 Article number: 36671*, 2016.

[30] Aflakparast, M., Salimi, H., Gerami, A., Dubé, M-P. Visweswaran, S., Masoudi-Nejad, A., “Cuckoo search epistasis: a new method for exploring significant genetic interactions,” *Heredity (Edinb). 112(6): 666-674*, 2014.

[31] Tuo, S., Zhang, J., Yuan, X., Zhang, Y., Liu, Z., “Fhsa-sed: Two-locus model detection for genome-wide association study with harmony search algorithm,” *PLoS ONE 11(3)*, 2016.

[32] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation 6(2): 182–197*, 2002.

[33] Beume, N., Fonseca, C.M., López-Ibáñez, M., Paquete, L., Vahrenhold, J., “On the complexity of computing the hypervolume indicator,” *IEEE Transactions on Evolutionary Computation, 13(5), October*, 2009.

[34] Han, B., Chen, X., Talebizadeh, Z., Xu, H., “Genetic studies of complex human diseases: characterizing snp-disease associations using bayesian networks,” *BMC Syst. Biol. 6(Suppl. 3), S14*, 2012.

[35] Jiang, X., Neapolitan, R.E., Barmada, M.M., Visweswaran, S., “Learning genetic epistasis using bayesian network scoring criteria,” *BMC Bioinform. 12(1), 89*, 2011.

[36] Wu, T.T., Chen, Y.F., Hastie, T., Sobel, E., Lange,K., “Genome-wide association analysis by lasso penalized logistic regression,” *Bioinformatics 25(6): 714-721*, 2009.

[37] Akaike, H., “Information theory and an extension of the maximum likelihood principle,” *Parzen E., Tanabe K., Kitagawa G. (eds) Selected Papers of Hirotugu Akaike*, 1998.

[38] North, B.V., Curtis, D., Sham, P.C., “Application of logistic regression to case-control association studies involving two causative loci,” *Human Heredity 59(2): 79–87*, 2005.

[39] Yang, C., Wan, X., Yang, Q., Xue, H., Yu, W., “Identifying main effects and epistatic interactions from large-scale snp data via adaptive group lasso,” *BMC Bioinformatics. 2010; 11(Suppl 1): S18*, 2010.

[40] Yang, C.H., Lin, Y.D., Chuang, L.Y., Chang, H.W., “Evaluation of breast cancer susceptibility using improved genetic algorithms to generate genotype snp barcodes,” *Transactions on Computational Biology and Bioinformatics 10(2): 361–371*, 2013.

[41] Urbanowicz, R.J., Kiralis, J., Sinnott-Armstrong, N.A., Heberling, T., Fisher, JM, Moore, J.H., “Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures,” *BioData Mining 5(1): 16*, 2012.

[42] Brunham, L.R., Hayden, M.R., “Hunting human disease genes: lessons from the past, challenges for the future,” *Human Genetics 32(6): 603-617*, 2013.

[43] Deb, K., “Multi-objective evolutionary algorithms,” *Springer Handbook of Computational Intelligence. Springer, pp. 995–1015*, 2015.

[44] Visscher, P.M., Wray, N.R., Zhang, Q., Sklar, P., McCarthy, M.I., Brown, M.A., Yang, J., “10 years of gwas discovery: Biology, function, and translation,” *The American Journal of Human Genetics. 101(1): 5–22*, 2017.

[45] Dinu, I. et al., “Snp-snp interactions discovered by logic regression explain crohn’s disease genetics,” *PLOS ONE 7(10): 1–6*, 2012.

[46] Sun, J., et al, “Hidden risk genes with high-order intragenic epistasis in alzheimers disease,” *Journal of Alzheimers Disease 41(4): 1039–1056*, 2014.

[47] Yang, J.K., Zhou, J.B., Xin, Z., Zhao, L., Yu, M., Feng, J.P., Yang, H., Ma, Y.H., “Interactions among related genes of renin-angiotensin system associated with type 2 diabetes,” *Diabetes Care 33(10): 2271–2273*, 2010.

[48] Cordell, H.J., “Epistasis: what it means, what it doesn’t mean, and statistical methods to detect it in humans,” *Human Molecular Genetics, 2002, 11(20):2463–2468*, 2002.

[49] Shriner, D., Vaughan, L.K., Padilla, M.A., Tiwari, H.K., “Problems with genome-wide association studies,” *Science 316(5833), 1840–1841*, 2007.

[50] Chatelain, C., Durand, G., Thuillier, V., Augé, F., “Performance of epistasis detection methods in semi-simulated gwas,” *BMC Bioinformatics. 2018; 19: 231*, 2018.