Survival of the fittest in the jungle of Open Educational Resources

a Master Thesis by

Sander Latour

UNIVERSITY OF AMSTERDAM

TutOER

ONLINE CURRICULUM SEQUENCING USING GENETIC ALGORITHMS BASED ON MEASURED LEARNING GAIN.

A thesis submitted in conformity with the requirements for the degree of
MSc. in Artificial Intelligence

MARTIN SANDER LATOUR
sanderlatour@gmail.com

Supervisors:

MAARTEN VAN SOMEREN and DIEDERIK ROIJERS
Informatics Institute, Faculty of Science,
Universiteit van Amsterdam
Science Park 904, 1098 XH Amsterdam

Defended on 15th of August, 2014

Contents

List of Figures

List of Tables

1

Introduction

Open Educational Resources (OER) are a well-studied topic in communities of scientists [58, 30, 24], practitioners [31, 60, 59] and policy-makers [11, 44]. In addition to standard learning objects [39], OER are free to be reused, revised (i.e. altered), remixed (i.e. combined with others) and afterwards redistributed [26]. As a consequence the threshold for innovation of education material is lowered. This results in a less stable quality level of the shared learning objects due to a larger diversity in authors [56].

The selection of OER from repositories and the subsequent determination of the quality are particularly dependent on actions of instructors. Ochoa and Duval [42] show that the size of OER collections vary from hundreds to millions of objects depending on the type of repository. Ochoa and Duval expect that exponential growth will occur when the repositories are capable of retaining its productive users. Determining the quality of an exponentially growing number of open educational resources by human effort is infeasible [14, 41, 61]. Acquiring a reliable automatic method for assessing OER quality is thus required.

Previous approaches have predominantly focussed on proxies of OER quality. Some automatically evaluated the quality of metadata [43, 51]. Cechinel et al. [14, 15] took a different approach by predicting quality ratings of learning objects based on intrinsic metrics (e.g. number of links in a document). Duval [23] proposed the context-dependent ranking algorithm LearnRank, where learning objects that are used in many contexts receive a higher rank. Ochoa and Duval [41] take the attention given to an OER as a proxy for its usefullness. A recent report from the European Commission on the quality issue of OER identified one of the challenges to be that educational resources of high quality are fragmented with no particular way of distinguishing them from the other available learning objects [12]. Duval [23] states that in an ideal case empirical data of the learning effect caused by a particular OER bootstraps its ranking. Camilleri et al. [12] refers to impact as one of the five important aspects of OER quality. However, in the proposed automatic mechanisms to assess OER quality, the impact a particular OER has on learning has thusfar been mostly neglected [32]. This is an undesirable situation as the sole purpose of any learning material is to have a positive effect on learners. This situation is particularly unsatisfactory because of the growing demand for evidence-based teaching decisions using quantitative data [55, 37, 49, 19].

A complication with the assessment of OER quality by measured impact is that in general educational material is sequenced with other educational material before being presented to students [10, 45]. Furthermore, the activities before and after an OER also affect its quality [23]. It is therefore necessary to determine the quality of an OER within the sequence it is part of. As a consequence, two issues need to be taken into account when automatically assessing OER quality. First of all, there are many OER sequences possible, even when there are only a few OER available for a particular topic. These sequences will require multiple evaluations before an estimate of learning impact can be made due to the inherent noise in the domain. This experimentation clearly does not come without cost. Each time a sequence shows to be less effective than a different one, a learner has received a lower level of education than necessary. This turns OER quality assessment in a curriculum sequencing task [2] with an *exploration vs. exploitation trade-off* [27]. Second of all, repositories are constantly updated with new OER. The collection of possible sequences of OER will therefore continue to grow during a quality assessment process. This is known as the *open corpus* problem [6].

This thesis introduces and evaluates **TutOER**, a novel approach to automatic assessment of OER quality. **TutOER** estimates the impact of a sequence of OER by measuring the knowledge level of a student before and after the sequence is presented to the student. Figure 1.1 depicts the situation. The curriculum sequencing task is executed by a genetic algorithm with generational replacement and elite preservation. Sequences of OER are evolved by crossover and mutation in order to consider better sequences over time. The survival chance of an OER sequence is proportional to its measured impact. An additional confidence-based selection mechanism was added to the genetic algorithm for better cost boundaries.

The **TutOER** system was evaluated in a newly created online course around the mathematical game Nim. The course contained four lessons through which a participant was instructed about the game and its winning strategy. Each lesson contained a sequence of OER that was selected by the **TutOER** system to be evaluated. Additionally a series of simulations were executed to explore the behavior of the system in several theoretical situations.

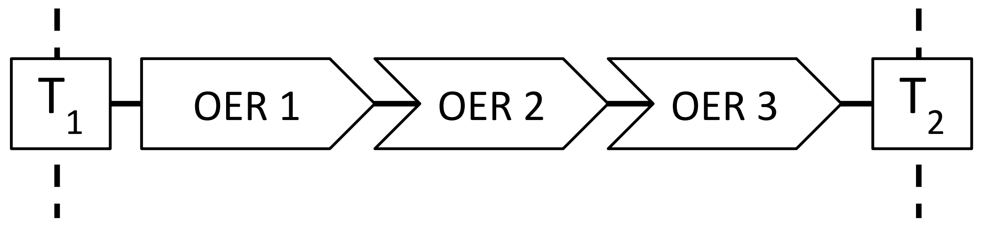


*Figure 1.1: The impact of an OER sequence is measured by a knowledge test before and after the sequence was presented to the student. The sequence is selected by the **TutOER** system.*

This thesis is structured as follows. Chapter 2 provides background information and related work. Chapter 3 contains a detailed description of the task. The approach is described in Chapter 4 and the resulting software in Chapter 5. In Chapter 6 the simulations are discussed. Chapter 7 covers the setup of the experiment. The results of this experiment are discussed in Chapter 8. Chapter 9 concludes the findings in this thesis and provides general discussion.

2

Background

This thesis addresses a problem in the community of *Open Educational Resources*. It does so with a solution from the *Intelligent Tutoring Systems* community, called *curriculum sequencing*. More specifically, the balance between good quality estimates and the cost bad teaching is optimized as a curriculum sequencing problem. The curriculum sequencing is executed by a genetic algorithm. This chapter provides a background in OER, automatic assessment of the quality of OER and curriculum sequencing. Section 2.1 describes what OER are. In Section 2.2 an overview is given on how assessment of quality is done at the moment. Section 2.3 gives an overview of the relevant literature in the field of curriculum sequencing.

2.1 Open Educational Resources

Several definitions of the term Open Educational Resources (OER) are given, since the term was first coined by UNESCO in 2002 [12]. UNESCO defines OER as follows.

> "The open provision of educational resources, enabled by information and communication technologies, for consultation, use and adaptation by a community of users for non-commercial purposes" [52]

The William and Flora Hewlett Foundation, the key funders of OER initiatives, define OER as follows.

> "OER are teaching, learning, and research resources that reside in the public domain or have been released under an intellectual property license that permits their free use or re-purposing by others. Open educational resources include full courses, course materials, modules, textbooks, streaming videos, tests, software, and any other tools, materials, or techniques used to support access to knowledge." [3].

OER are closely related to reusable learning objects. The key difference lies in the emphasis on openness [12]. Hilton III et al. [26] provides a framework to think about openness in OER, called "The Four R's of Openness". The four R's are reuse, revise, remix and redistribute. In the context of OER, these can be explained as follows. OER are free to be used in any way (reuse). OER are free to be altered in any way (revise). OER are free to be combined with other work (remix). OER are free to be shared with others (redistribute).

Collis and Strijker [20] enumerates six stages in the lifecycle of a reusable learning object. First, a learning object is obtained or created. Second, the learning object is labelled with metadata information. Third, the learning object is offered, often in a learning object repository, to be selectable for potential use. Fourth, the learning object is selected to be used in an educational context. Fifth, the learning object is used either in a self-contained manner or in combination with other learning objects in an educational context. Sixth, after of during the learning object is used a decision is made whether or not to retain this learning object. Figure 2.1 depicts the cycle that these stages form.

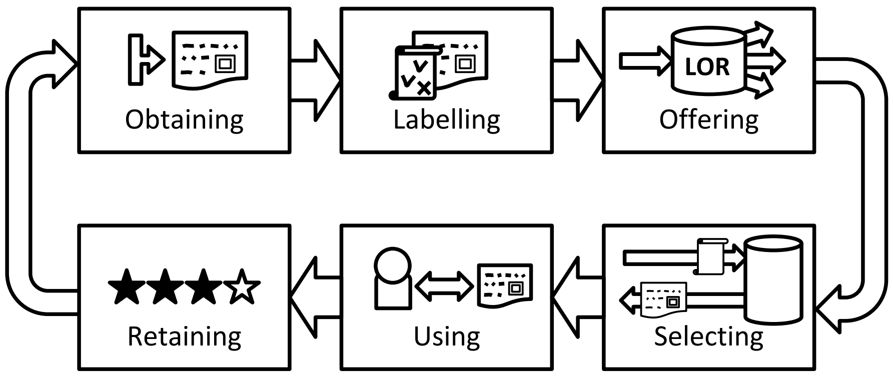


Figure 2.1: Lifecycle of reusable learning objects, as enumerated by [20]

The advent of OER lowered the threshold of creating and distributing educational material. Weller [56] states that the resulting diversity of authors will cause a less stable level of quality. However, a lot is expected from these authors. The human instructor plays an important role in almost all of the mentioned stages in Figure 2.1. In particular, the human instructor is responsible in most systems for quality assessment. According to [12] quality assurance ranges from strict top-down controlled production processes to peer-review and everything in between.

However, according to [42] the number of OER in learning object repositories vary from hundreds to millions of objects. The growth of these repositories is still linear in the number of authors [42]. However, Ochoa and Duval expect that this growth will become exponential when authors are better retained by the repositories. Determining the quality of an exponentially growing number of open educational resources by human effort is infeasible [14, 41, 61]. As a result, there is a growing interest in automatic assessments of OER quality.

2.2 Automatic assessment of OER Quality

One approach focuses on the quality of the metadata of the learning object [43, 51], as a proxy for the quality of the object itself. A problem with this approach is that metadata can be inaccurate [13] and incomplete [48]. A different approach is to automatically assess the quality of learning objects based on intrinsic metrics, such as the number of links in a learning object. Cechinel et al. [14] managed to find statistical profiles of different quality labels by comparing intrinsic metrics of good and poor learning objects in the MERLOT repository. Duval [23] however states that the quality of a learning object is context-dependent and intrinsically subjective. As examples of context Duval mentions a.o. learning goal, available time and educational level. Duval proposes the context-dependent ranking algorithm LearnRank, in which learning objects that are used in many contexts receive a higher rank. In [41] the “use of contextualized attention metadata for ranking and recommending learning objects” is proposed, where the attention given to a learning object by a student or a teacher within a certain context is considered to be a proxy for the usefulness of the object in that context. Duval however also mentions in [23] that “In an ideal world, we would actually bootstrap and steer this process [of establishing the LearnRank] through empirical data on the learning effect that specific objects have actually caused (or helped to realise) in specific contexts ...”. According to [32] the effect of a learning object on actual learning is underrepresented in the research on learning objects.

In a recent report from the European Commission, quality of Open Educational Resources had five aspects: efficacy, impact, availability, accuracy and excellence [12]. The aspect of impact referred to the extend to which an object or concept proves effective. However, little is said in Camilleri et al. [12] about methods that would measure this impact. Camilleri et al. further state as one of the challenges that educational resources of high quality are fragmented with no particular way of

Table 2.1: Different NLG values

$\mathbf{C^1}$	$\mathbf{C^2}$	**NLG**
0	0	0
0	$^{1}/_{3}$	$^{1}/_{3}$
0	$^{2}/_{3}$	$^{2}/_{3}$
0	1	1
$^{1}/_{3}$	0	$-^{1}/_{2}$
$^{1}/_{3}$	$^{1}/_{3}$	0
$^{1}/_{3}$	$^{2}/_{3}$	$^{1}/_{2}$
$^{1}/_{3}$	1	1
$^{2}/_{3}$	0	-2
$^{2}/_{3}$	$^{1}/_{3}$	-1
$^{2}/_{3}$	$^{2}/_{3}$	0
$^{2}/_{3}$	1	1

distinguishing them from the other available learning objects. They recommend creating specialised directories where exclusive lists of repositories of high-quality content are maintained. It is however not stated how these high-quality repositories could incorporate impact in their quality assessment.

This thesis presents work that assesses OER quality by the impact it has on learning. This impact is measured by assessing the competence level before and after the OER is presented. The normalized learning gain (NLG) is used to express the impact the OER has on the competence level. The NLG metric is a widely adopted measure of student learning. The main advantage of NLG is that student learning is measured irrespective of the student's incoming competence [18]. The NLG metric is calculated using formula (2.1), where C^1 and C^2 denote the pre-test and post-test scores respectively. The value 1 is assumed to be the maximum score of an assessment. Table 2.1 shows the NLG value for a variety of parameter values. The normalized learning gain is essentially a normalized distance metric. Thus, any linear function expressing competence with a known maximum value is applicable to C^1 and C^2. The experiments performed for this thesis use the ratio of correct answers. Using the NLG metric, sequences of OER can be evaluated by comparison of the assessment score before and after presentation.

$$nlg(C^1, C^2) = \frac{C^2 - C^1}{1 - C^1} \tag{2.1}$$

2.3 Curriculum Sequencing

Curriculum sequencing concerns generating a sequence of teaching operations that is optimal for an individual learning [8]. The level on which this takes place ranges from course sequencing to content sequencing. Curriculum sequencing is an established part of the larger field of intelligent tutoring systems [9]. Vanlehn [53] provides an introduction into intelligent tutoring systems. Although most intelligent tutoring systems used to be supported by extensive and explicit knowledge engineering, data-driven systems are now emerging [33]. Within and around the intelligent tutotoring systems community there have been different technologies involved with curriculum sequencing tasks.

The field of Adaptive Hypermedia Systems (AHS) explores adaptive presentation and adaptive navigation in hypertext documents [7]. Many existing AHS require the set of documents to be known in advance, referred to as a *closed corpus* [6]. According to Brusilovsky and Henze, these documents are annotated with metadata and linked to ontologies before presented to students. Based on this additional information adaptation takes place to cater for individual needs. Examples of such systems [1, 34] are used in formal education as authoring tool. Due to the *closed corpus* assumption, this systems are not useful in dealing with open educational resources [6]. Brusilovsky and Henze further state that AHS should move to work with an *open corpus.*

A technique that is much more suited for the open corpus task is *collaborative filtering* [36]. Together with *content-based filtering* and *hybrid models*, it is one of the three main recommendation techniques [54]. Collaborative filtering bases recommendations on actions of similar people. Content-based filtering compares content instead of people and bases recommendations on that. Hybrid models combine multiple recommender systems. [36] provides an extensive review of recommender systems using in technology-enhanced learning. A recent example that was published after the review is [54]. Verbert et al. [54] recommends learning designs to teachers based on patterns of existing learning designs of peers. Furthermore, resources are recommended within these designs, based on students' usage data.

A Markov Decision Processes are a branch of reinforcement learners that indirectly generate sequences. [18] presents a conversational physics tutor that makes micro-decisions about whether to tell or elicit a certain fact. Features from both the conversation and the student performance are used. As a result, a sequence of pedagogical actions is created.

Another popular approach to curriculum sequencing is evolutionary computing. Al-Muhaideb and Menai [2] provides an extensive overview of this literature. The main approach is taking curriculum sequencing to be a constraint problem. Several systems include a term that expresses how well a particular solution fits the pre-determined prerequisite structure of the learning objects [47, 17, 46]. Other work in this field includes a term that expresses how smooth the transitions are between learning objects in difficulty [28, 47, 16, 29] or how well their difficulty matches the compentency level of the student [47, 16, 17, 46, 29]. De-Marcos et al. [22], de Marcos et al. [21] redefines the sequencing problem to be a permutation problem and applies both a genetic algorithm and a particle swarm optimization. The sequencing done tries to match compentencies the students has or desires with compentency-related metadata of the learning objects. Both evolutionary algorithms work, but particle swarm optimization outperforms the genetic algorithm.

The proposed work in this thesis applies the genetic algorithm with many of the same features as ealier systems, such as generational replacement, integer-encoding for chromosomes and elitism. However, unlike all mentioned work, the approach taken in this thesis uses the normalized learning gain caused by the sequence to be its fitness. Furthermore, the learning objects are treated as black boxes in this thesis. No metadata or expert-driven ontologies are used. The only thing known about the learning objects, within the context of this thesis, is their topic.

3

OER Sequencing

The impact a sequence of OER has on learning is not known within OER repositories. Any system that would want to present the optimal sequence to a learner would have to empirically discover this impact. In the process of this discovery, suboptimal sequences are presented to a learner, which could reduce learning. The goal of this thesis is to find the optimal sequence of OER while minimizing the negative consequences of the search process. The rest of this chapter discusses this in more detail.

3.1 Educational context

We consider OER sequencing within a lesson of an online course. The course focuses on a particular topic and is split up into various lessons that cover a concept or knowledge area that is required for understanding the course topic. The lesson is taught by an automatic tutor through a sequence of educational material that is presented to an individual student. Several educational resources are available for this lesson, from which the tutor needs to make a selection. The result of this selection is an ordered sequence of educational resources that are consecutively presented. At the beginning and at the end of each lesson there is an assessment, refered to as the pre-test and post-test respectively. These assessments measure the relevant competence level before and after the presented sequence. The situation is depicted in Figure 3.1. The task central to this thesis is to find the optimal sequence of resources.

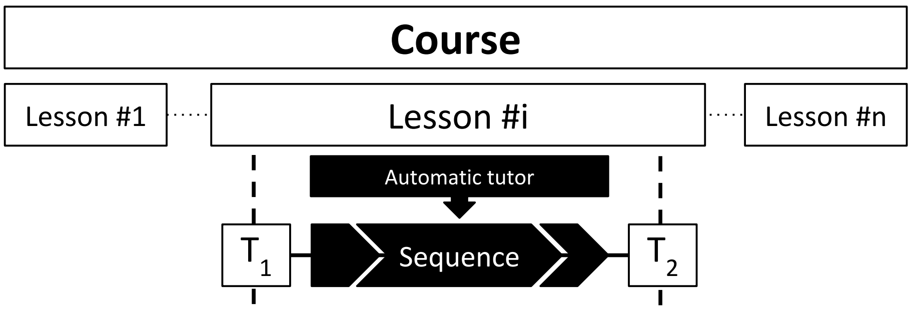


Figure 3.1: The sequencing task takes place within a lesson of a course.

There is no pre-defined length of the sequences for each lesson. Only one educational resource could be enough to explain it. The lesson might also require multiple resources. However, the lengths are restricted to a pre-defined lower and upper bounds. Additionally, to simplify the problem slightly more, the sequence must not contain an educational resource multiple times. This is partly to reduce the number of possibilities and partly because the time span between the duplicates would be rather short. In this short time span, it is unlikely that repetition would be useful.

3.1.1 Quality of OER

The quality of a particular sequence of educational material is determined by the impact it has on learning. The impact is defined as the measured normalized learning gain between the pre-test and the post-test, as given by Equation (2.1). A sequence is optimal when the measured normalized learning gain is maximized. The search task is thus straightforward. Namely finding the sequence with maximum expected impact out of all possible sequences. Equation (3.1) describes this mathematically, where R denotes the set of all educational resources available for that lesson, C^1 and C^2 denote the normalized pre-test and post-test scores respectively, a and b denote the minimum and maximum length of a sequence respectively and S denotes the set of all possible sequences for that lesson.

$$\underset{s \in S}{\arg\max} \left[\mathrm{E}[\, nlg(C^1, C^2_s)\,] \right], \quad S = \bigcup_{k=a}^{b} R^k \tag{3.1}$$

3.1.2 Solution space

The number of sequences of educational resources that need to be considered can increase quickly. A sequence of three slots can be instantiated from a collection of five resources in 60 $(5 * 4 * 3)$ different ways. When the collection is twice as large, in total 720 sequences would be possible. The number of possible sequence instantiations is given by the *k-permutation of n elements*, where k is the number of slots to instantiate and n the number of elements to draw from.

Apart from the number of different instantiations, sequences have a variable length. The total number of possible sequences is thus the sum of all possible instantiations of sequences of all possible lengths. This results in Equation 3.2, where $|R|$ denotes the number of resources for the lesson, a and b denote the minimum and maximum number of resources in the sequence respectively and $|S|$ denotes the number of possible sequences. The equation is essentially the formula for the number of *k-permutations of n* given by $\frac{n!}{(n-k)!}$ summed for all possible values of k. Table 3.1 shows the outcome of this equation for various values of $|R|$, a and b.

$$|S| = \sum_{k=a}^{b} \frac{|R|!}{(|R|-k)!}, \quad a \leq b \leq |R| \tag{3.2}$$

$\|R\|$	a	b	$\|S\|$
3	1	3	15
4	1	3	40
5	1	3	85
5	1	4	205
5	1	5	325
5	2	5	320
10	1	3	820
10	1	4	5860
10	2	4	5850
10	3	5	36000
10	1	10	9.864.100
10	5	10	9.858.240

Table 3.1: Solution space sizes calculated for a few parameter values using Equation (3.2). Parameters $|R|$, a and b represent the number of resources, the minimum length of a sequence and the maximum length of a sequence respectively. $|S|$ represents the number of possible sequences, given the parameters.

3.1.3 Student Groups

Within the scope of this thesis, the sequences are not optimized for every individual student. Instead, each student is assigned to a student group, for which the sequence of educational material is optimized. Students are assigned to the student group based on their normalized pre-test score. The score is descretized in a low and high value. Specifically, students who have less than half the questions correct are assigned to the low student group and the others are assigned to the high student group. The optimal sequence of education material must be found for each student group. In order words, students who know little about the topic might receive different material than students who know a lot. This results in a new situation, which is depicted by Figure 3.2.

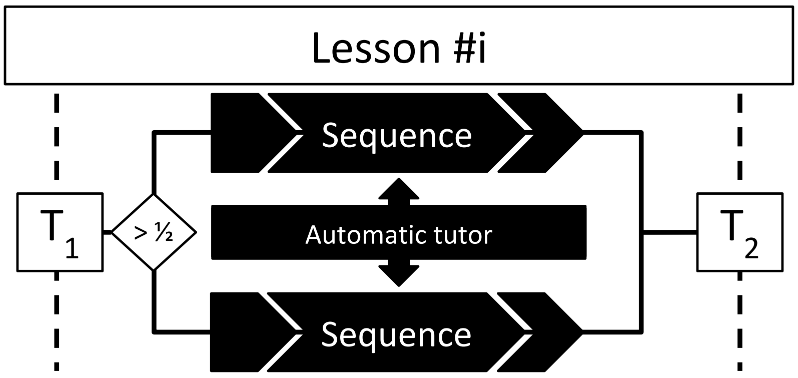


Figure 3.2: The sequencing task is performed separately for each student group.

There are several other features on which students could have been divided even further into more specific groups. Learning style, gender and age are not uncommon feature candidates. These are not used in this thesis for mainly two reasons. First, determining the values for these characteristics can be difficult and unreliable in a web context. Second, this would result in more groups. Each additional group requires new students in order to find the optimal sequence in that group. Acquiring many students is not possible within this thesis.

The coarse division of students in groups could cause a large diversity of students within each group. That means that the evaluation of a sequence by a student is not necessarily representative for the average evaluation. On top of that, the assessment of impact will be noisy. For example because students might be distracted during one of the assessments. That means that sequences must be evaluated multiple times in order to acquire a better estimate.

Although the optimal sequence is determined per student group, they are not entirely independent. This becomes apparent through an analogy. A teacher that teaches different cohorts will try to optimize his teaching for each separate group. If however the teacher would observe that a particular approach works really well for one group, the teacher might try it out on other groups as well.

3.2 Multiple objectives of OER sequencing

Evaluating the optimality of a sequence comes with a cost. Each evaluation requires a new student and there are only limited students available. Therefore the number of students required to find the optimal sequence need to be minized. Furthermore, presenting a less-than-optimal sequence to a student could limit the amount of learning. That is something one wants to avoid in an educational setting.

The “damage” done to the student’s learning can be expressed in *regret*. The definition of regret is the difference in the reward received between performing the optimal action and the current action. In other words how much reward is missed by not choosing the optimal action. In the context of this thesis, regret is the learning gain that is missed by not presenting the optimal sequence.

However, we do not know beforehand what the optimal sequence is. The process of searching for the optimal sequence will result in evaluations of less-than-optimal sequences. Recall that these evaluations involve actual students. Thus, the regret built up during this search process, known as the *online regret*, needs to be minimized. Furthermore recall that observations of the sequence’s optimality are noisy. Multiple evaluations are needed for accurate estimates. This results in two objectives, namely better estimates and minimizing online regret.

This situation is familiar to many fields and is known as the exploration vs. exploitation trade-off [27]. An often used example is the n-armed bandit problem [50] where a casino offers n different slot machines (a.k.a. one-armed bandits) to play. A player would want to optimize the total amount of money earned and is therefore looking for the slot machine that has the highest pay-off. It is tempting to stay with a slot machine that gives the highest return you have seen so far (exploitation), but it is important to also try out other slot machines to see if they perform even better (exploration). This can be applied to OER sequencing by aligning the bandits with sequences of learning objects and the pay-off with learning gain.

4

Approach

As described in Chapter 3, the goal of this thesis is to find the optimal sequence of OER while minimizing online regret. The **TutOER** system needs to select a sequence for each student within a lesson. This chapter describes how the system does this. The approach uses a genetic algorithm to search in a more structured manner through the space of possible sequences. The genetic algorithm paradigm is introduced in Section 4.1. Section 4.2 discusses how this paradigm is applied to this thesis.

4.1 The Genetic Algorithm

This section provides a brief introduction into genetic algoritms. The purpose of this section is to establish a shared understanding of the standard setup. For a more in-depth introduction, the reader is referred to [25].

The *genetic algorithm* [27] is a part of a family of search algorithms called *evolutionary computing*. The technique draws inspiration from Darwinian evolution and natural selection. In genetic algorithms, a population of individuals evolves over multiple generations to better perform on some metric. These individuals all have a set of traits that influence their performance. For example, a bird could have traits like a long tail or bright colors. Traits are caused by certain genes. A particular configuration of genes is called a chromosome. Individuals that perform better than others have a higher chance of survival. The new generation contains the offspring of the individuals that survived. The offspring inherits the chromosomes of their parents. That way, traits that have a positive influence on performance have a higher chance of ending up in new individuals. Surviving individuals pair up with others to form a set of two parents. The resulting offspring inherits a combination of the chromosomes of both parents. Usually two parents form two new individuals. This application of the *survival of the fittest* results in more individuals with successful traits and less individuals with unsuccessful traits.

The peppered moth, the most cited example of Darwinian evolution [35], illustrates this perfectly. The peppered moth rests during the day on the trunk of particular trees. At that moment birds pray on the moths. The color of the peppered moth is originally light gray. This trait camouflages them fairly well against the light bark of the trees. In other words, the fitness of these white moths is higher than moths of a different color. As a result most peppered moths had a light color. However, that all changed during the industrial revolution. The industrial revolution caused polution in the air that blackend the bark of the trees. As a result, the white peppered moths were easily spotted by praying birds. A variation of the peppered moth, with a dark color, suddenly had a better camouflage against the dark trunks. The black-bodied peppered moth produced more offspring because their increased chance of survival. As a result the occurence of dark peppered moths rose significantly. In modern times, the air is much cleaner and the color of the bark of the trees became light again. As a result the dark peppered moths were at a disadvantage. The frequency of light peppered moths rose to be the large majority once again.

New variations like the dark peppered moths, are caused by genetic mutations. A mutation is a copying error of the chromosome during inheritence. The mutated chromosome causes traits that could benefit an individual's chance of survival. In that case the mutated chromosome will occur more often in new generations of the population. As was the case in the example of the peppered moths.

The genetic algorithm uses the same approach to find optimal solutions in a large space of possibilities. In our case, finding the optimal sequence of OER. A particular application of this algorithm designs counterparts of the components of the natural selection mechanism. The components that need to be designed are briefly discussed in the rest of this section. The standard genetic algorithm follows roughly the following steps [25].

Genetic Algorithm Outline

1. Initialization
2. Evaluation of each candidate
3. Repeat until termination condition is satisfied:
    a) Parent selection
    b) Recombination of parent pairs
    c) Mutation of the resulting offspring
    d) Evaluation of each candidate
    e) Survivor selection

4.1.1 Representation of the domain

The *phenotype* is the collection of properties of an individual, such as a long tail or color. The *genotype* is the genetic information that causes those properties. In order words, the chromosome containing particular genes represents the aforementioned properties in genetic terms. This distinction is important for genetic algorithms, as it is often the case that there is a translation necessary between the two. Arguably, one of the first tasks in applying a genetic algorithm is finding a translation of your solution space to a genetic encoding.

A commonly used encoding is the *binary encoding*, where chromosomes are encoded as strings of 1's and 0's. This encoding allows for simple mutation and crossover operators that work on the bit level. However, for many problem types this can result in invalid individuals. Other encodings have been developed to improve this.

For example, suppose we want a genetic algorithm to work on the travelling salesman problem. In this problem, the salesman needs to find the shortest route through all cities. Each solution to this problem would be some travelling plan that enumerates cities in the order that the saleman should visit them *(phenotype)*. This could be encoded as an ordered list of numbers, where each number represents a city *(genotype)*. This encoding is known as the integer or *permutation encoding*. In the field of curriculum sequencing, the permutation encoding is also widely used [2].

4.1.2 Candidate evaluation

Individuals, carrying a particular chromosome, are exposed to the environment. This environment determines the chance of survival of the individuals, and thereby of their chromosomes. In the peppered moth example, the bark of the tree's trunk determines the survival chance of each individual moth. This survival chance, due to how well the individual fits its environment, is expressed by the fitness value. In more abstract terms, the fitness value expresses how good the found solution is. The fitness function can calculate these values for an arbitrary individual.

The example of the travelling salesmen has an obvious candidate. The fitness function could measure the distance travelled in the proposed travelling plan. Although in this particular example, you would want the distance to negatively correspond to the survival chance.

The selection of individuals to evaluate is done uniformly in the basic implementation. In domains where the fitness function is deterministic, each individual is evaluated once. In more noisy domains there is a larger number of evaluations that can be divided over the individuals uniformly in each generation.

4.1.3 Population

A population of individuals goes through several generations. Each generation is a new step in the search for the optimal solution. The initialization of a population is usually done by random sampling of chromosomes for its individuals. An important property is the number of individuals in each generation. Usually this number is fixed to one number, but it can also vary. The number of individuals is important as it determines the capacity for variation in one generation. The population is however a multiset. This means chromosomes can be contained by multiple individuals. Therefore it is also relevant to look at the diversity in the generation.

The standard implementation of a genetic algorithm needs to terminate. One obvious candidate for a termination condition is when the algorithm found an optimally performing individual. In other words, when the fitness of the individual is, within a small range of, the maximum possible value. However, there are no guarantees that the genetic algorithm would reach this point. Other termination conditions can be added to deal with this. One example is to stop after a fixed number of evaluations.

4.1.4 Evolution

Searching occurs in genetic algorithms by means of evolution. There are three important stages in this evolution. First parent pairs are selected from the survivors. The chromosomes of these parents are then recombined into two new chromosomes. After the recombination, mutation may take place on the resulting chromosomes. At the end, the two new chromosomes end up in the new generation. The three stages are further clarified in the next subsections.

4.1.4.1 Parent Selection

When all fitness evaluations have been made, some chromosomes will be selected to become a parent. Parents are selected in pairs, in order to let their chromosomes recombinate. This selection of parents is at least based on the fitness of the chromosomes. However, also other features such as a chromosome's age may influence the selection. There are also various ways in using these values for selection. One is *tournament selection*, where individuals compete against each other and the one with the highest fitness wins. Another is *ranking selection*, where the probability of a chromosome being selected is proportionate to its rank in fitness values. A common selection method is *roulette wheel selection* or *fitness-based selection.* This is a sampling method where the fitness value is proportionate to the probability of being sampled. The roulette wheel is a circle divided in a number of segments. Each segment corresponds to an individual. The size of the segment is proportionate to the fitness value of the individual. A random position on the circle is chosen. The segment that contains this position corresponds to the individual selected.

4.1.4.2 Recombination

Recombination is responsible for combining information from two chromosomes. This recombination occurs by a crossover operator. There many different possible crossover operations. Two common ones are *one-point crossover* and *two-point crossover.* The one-point crossover splits the chromosomes at the same random point. The resulting four halfs are recombined in the alternative way, while maintaining their position in the chromosome. Figure 4.1 illustrates the splitting of two parent chromosomes and their recombination into two child chromosomes.

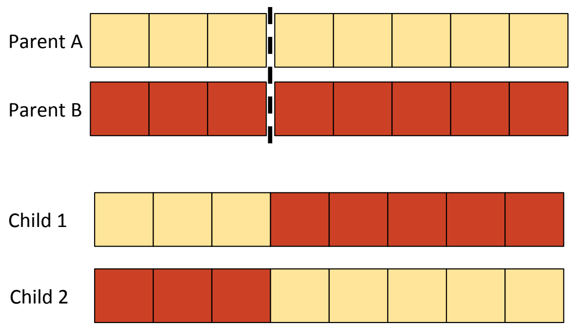


Figure 4.1: Example of a single-point crossover operation

The two-point crossover splits the chromosomes at two points. The segment of genes between the two split points in both both chromosomes is swapped. Figure 4.2 illustrates the application of the two-point crossover.

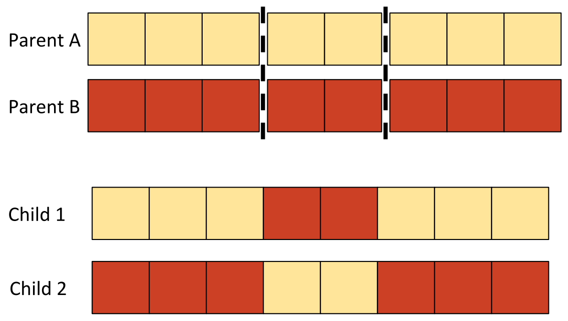


Figure 4.2: Example of a two-point crossover operation

4.1.4.3 Mutation

Selection and recombination steer the search towards the part of the search space that appears to be most promising. As a result, parts of the search space might never be reached and evaluated. There is no guarantee that this steering will converge to an optimal solution. This is especially true when the initial population does not hold the necessary variation. Therefore, most genetic algorithm implementations also include mutation. The mutation operator introduces random changes to chromosomes when passed through from parents to offspring. In the standard binary encoding of genes in a chromosome, the basic mutation operator flips a random bit. The permutation encoding facilitates a swap mutation, where to random positions in the chromosome are swapped.

Although mutations can be vital for performance, it is still a disruption. Too much mutation will prevent convergence. Mutation is therefore usually only applied with a very low probability. The mutation operator is applied after recombination took place.

4.1.4.4 Survivor selection

Populations often have a fixed size. The newly created offspring together with the existing population forms a group of candidates that exceeds this size. In the survivor selection step, the members of the new generation are selected from these candidates. Unlike the parent selection, which is stochastic, the survivor selection is often deterministic. An example selection method is to just take the top n chromosomes in order of their fitness. Another common method is *generational replacement*. In this method, the chromosomes in the generation are completely replaced by their offspring.

4.1.5 Extensions

Several extensions have been proposed to the standard genetic algorithm. Two of them are relevant for this thesis. They will be briefly explained.

4.1.5.1 Elite preservation

Although mutation ensures that every chromosome is theoretically reachable. However, the probability of this happening might be very small. In a finite number of generations there is no guarantee that the optimal solution will be found. There is furthermore no guarantee that good solutions will be kept in the population. Elitism, or *elite preservation*, ensures that the n best individuals of each generation are transferred to the next one. Elite individuals are not subjected to crossovers or mutation. Nor are they dependant on stochastic sampling. The number of elite individuals must be kept small. Otherwise the genetic algorithm will no longer have enough individuals to evolve.

4.1.5.2 Island model

In the island model, a population is split up in separate subpopulations, called *demes*. These demes evolve independantly from each other. However, periodically communication can occur through *migration* of individuals. The approach is a form of parallelisation and works particularly well when a problem consists of linearly separable subproblems [57].

According to [40], the island model is controlled by four parameters. These are the *topology*, the *migration interval*, the *migration scheme* and the *migration size*. The *topology* determines which demes are connected, which allows for migration. The *migration interval* determines the number of generations between a migration. The *migration scheme* determines which individual is picked from the source deme: the worst, the best or a random individual. It also determines which individual is replaced at the target deme: the worst, the best or a random individual. Replacement only occurs if the migrating individual has a higher fitness. The *migration size* determines how many individuals are exchanged during each migration. For a more extensive analysis of the island model in genetic algorithms the reader is referred to [38] and [57].

4.2 Applying the genetic algorithm

This section describes the way in which the task described in Chapter 3 is modeled using genetic algorithms. Each section discusses an important aspect of the modeling: representation (Section 4.2.1), initialization (Section 4.2.2.1), termination conditions (Section 4.2.2.2), fitness (Section 4.2.3) and parent selection, variation operators and survivor selection (Section 4.2.4). The applied island model to support for related student groups is described in Section 4.2.5.

4.2.1 Representation of the domain

The *permutation encoding* is used to represent each sequence. However, unlike most curriculum sequencing approaches [2], the chromosomes have variable length. The variable length is necessary because the OER sequences have variable length as well. Chromosomes can thus also be partial permutations, where not all OER are contained in the sequence.

4.2.2 Population

4.2.2.1 Initialization

The population is not initialized purely randomly. Instead, the first generation contains only sequences with exactly one gene. This is to introduce a bias towards smaller sequences. The individuals are generated according to the following steps:

1. For all resources in pool:

   a) If population is full, stop
   b) Else, add an individual with the chromosome that contains only that resource.

2. While there is room left in the population:
   a) Select a resource according to some probability density function
   b) Add an individual with the chromosome that contains only that resource

The probability density function (PDF) refered to in step 2a is a uniform distribution by default, but can also represent apriori weights of resources.

4.2.2.2 Termination

Given the inherent noise in the fitness values, the algorithm should not stop before the fitness of each possible[1] chromosome is determined with some certainty. That would seem to lead to a valid point of termination when all chromosomes are evaluated with enough certainty. However, the pool of resources is assumed to grow (i.e. new educational resources are made available) and each time a new resource is introduced it theoretically needs to be tried out in every combination with the already existing resources before the valid termination point would be reached. This would mean that the algorithm would never terminate, as it should wait for any new resources to arrive. If it is vital that the algorithm finishes, a practical approach could be to stop if the fitness of one or more individuals is within a small margin of the optimal value. Provided an optimal value can be defined.

The application presented in this thesis does not require the genetic algorithm to terminate. The web-based variation to the algorithm described in Section 5.3.1 ensures that computation only happens on a event basis. Furthermore, due to the nature of the application, it is not as interesting to have the best solution at the end as it is to select the best known solution at each point that an individual is tested. Naturally a exploration-explotation trade-off applies where occassionally individuals need to be tried out that could both be better or worse. So instead of termination, moving towards convergence is important.

4.2.3 Candidate evaluation

4.2.3.1 Fitness function

Learning objects are often not a perfect fit. They might explain too much or too little about some context. On top of that, it is not that well indexed in terms of the exact type of presentation that they have. Thus, what we want is a sequence of imperfect learning objects that together maximize the educational performance. We do not know what the order should be, given that the order is a matter of pedagogy and not knowledge engineering. And even if we were able to fully specify the right pedagogical order for each type of student perfectly. We would still not have the required information about these learning objects, or the information might be wrong. Thus, we are learning a sequence of black boxes of which we only know that they attempt to teach a particular knowledge component.

The only way we can measure the value of a particular sequence for a group of students, and thereby assess its fitness, is to look at the gain in knowledge as observed by the post-test. More precisely the fitness function used in this thesis is the normalized learning gain between the pre-test and post-test for a given knowledge component, given by $\frac{C^2-C^1}{1-C^1}$ where C^1 and C^2 represent the percentage of correct answers on the pre-test and post-test respectively of the student.

The observed fitness is probably not the same each time a chromosome is evaluated. This is due to the fact that students are not identical, especially not given the coarse division into student groups. A solution is to see the fitness as a stochastic variable that has some noise on top of the "true" value. In order to obtain an estimate of this true value, several approaches are possible. The most simple one is to take multiple samples and average over them. However, in this case, taking

[1]i.e. all possible sequences given the constraints on length and the uniqueness requirement

samples must be considered to be expensive. The approach taken must therefore try to minimize the number of samples while maximizing the certainty of the fitness value. Which is why Upper Confidence Bound selection was applied in this thesis, as described in Section 4.2.3.2.

4.2.3.2 UCB Selection

In this thesis, the Upper Confidence Bound (UCB) selection algorithm is used to determine which of the individuals will be evaluated. In particular the UCB-1 [4] algorithm is used. In UCB-1, first every individual is evaluated once. After this has been done, the individual is evaluated for which equation (4.1) is maximized, where $\overline{x_i}$ denotes the average fitness of the individual, n_i denotes the number of times the individual has been evaluated so far and n denotes the overal number of evaluations that occured.

$$\overline{x_i} + \sqrt{\frac{2 \ln n}{n_i}} \tag{4.1}$$

UCB-1 is proven in [4] to logarithmically bound the regret, which ensures that a suboptimal individual is selected logarithmically less often than the optimal individual. It is important to note that UCB-1 can only consider the individuals that are present in the current generation of the population for which the evaluation occurs. This means that there is some interplay between the UCB-1 mechanism and the selection mechanism of the genetic algorithm, where the genetic algorithm is responsible for searching through the solution space efficiently and the UCB-1 algorithm is responsible for reducing the regret.

4.2.4 Evolution

4.2.4.1 Parent selection

Parents are selected in pairs using *roulette wheel selection*. If the number of individuals in the population is odd, there will be one parent remaining after all pairs have been formed. That parent's chromosome is then added to the new generation as its own offspring.

4.2.4.2 Combination operator

When two parents are matched to create offspring, their chromosome's are combined using a crossover operation. The resulting chromosome is placed in a new individual. There are two crossover operations implemented for this thesis: *one-point crossover* and *append crossover*.

One-point crossover The one-point crossover operation is typically implemented by picking one point for both parents to split, after which the four halfs are recombined into two new children. The individuals in this thesis, however, can vary in length. That means that crossover points could be selected that do not exist in both parents. Naturally one could restrict the set of valid crossover points to be within the boundaries of both chromosomes. However, that would also limit valid chromosomes, even though they can be achieved by combining both parent chromosomes.

In this thesis, the one-point crossover operator is implemented differently. Instead of picking one point for both parents at once, one crossover point in each parent is randomly picked independent from the other. These two crossover points then split up both parents in two pieces each, allowing for the formation of two new children after recombination. The implementation ensures that only valid children are the result of the operation. When no valid children can be created, the one-point crossover is skipped and the append crossover is attempted.

Append crossover The append crossover was designed for the edge case where two parents cannot be split up and recombined into two new valid children. For example when one or both parents have a chromosome with one gene, which is impossible to split up. The append crossover operator simply appends one parent after the other. The two ways to do this result in two children.

4.2.4.3 Mutation operator

In this thesis three different mutation operators have been applied: *swap mutation*, *addition mutation*, *deletion mutation*.

Swap mutation The standard swap mutation for permutations is used. The chromosome needs to contain at least two genes in order to be applied to this mutation. If this is not the case, a different mutation is attempted.

Addition mutation The chromosome applied to the addition mutation operator will be appended with a new gene from the gene pool. The gene that is added must not already exist in the chromosome. If no gene can be selected from the pool that satisfies this constraint, a different mutation is attempted.

Deletion mutation The deletion mutation operation deletes a random gene from the chromosome, resulting in a shift in position of the genes after it. The resulting chromosome must have at least one gene left. If this is not possible, a different mutation is attempted.

4.2.4.4 Survivor selection

This thesis implements generational replacement with elite preservation, which are commonly used strategies for survivor selection in the curriculum sequencing domain [2].

4.2.5 Island model

Section 3.1.3 described how the sequencing task is done per student group and per knowledge component. Each combination is represented as separate populations. However, the populations that represent the same knowledge component but different student groups co-evolve.

The island model was used to model exchange of information between related populations. The migration scheme is set to migrate the best individual of the source population and replace the worst individual in the target population. This occurs at every new generation. The migrated individuals are copies, and do not change the occurrence of the migrated chromosomes in the source population. The topology links populations of the same knowledge component together. Only one individual is migrated per generation.

Important to note is that all other actions of the genetic algorithm in each population are still independent, meaning that the populations can also evolve at different speeds. As a consequence, a population that evolves really slowly will continue to migrate the same individual towards the target population. Even if it turned out not to work well. To counter this, the migrating individual of the source population competes with the worst individual in the target population through roulette selection. If the fitness of the migrating individual is worse than the worst individual in the target population, the replacement is not likely to proceed.

5

Software

In order to test the approach described in Chapter 4, the **TutOER** system was built[1]. **TutOER** is a web-based tutor that optimizes the sequence of OER to teach a concept to students. The two main software modules are the Tutor Module and the Genetic Algorithm (GA) Module. The latter implements the genetic algorithm approach chosen in this thesis in order to assist the Tutor module in selecting educational material. The software is web-based for mainly two reasons. First, it will make it easier to have people interact with the system, which is important when one needs to collect large amounts of data. Second, given the inherit intention of OER to be distributable, most OER are created for a web environment.

This chapter is structured as follows. Section 5.1 will describe the interface of the **TutOER** system. The Tutor Module is covered in Section 5.2. In Section 5.3, the implementation of the genetic algorithm approach in the GA Module will be discussed. Section 5.4 describes the Monitor Module, which allows for the analysis of the live system. The Logging Module is discussed in Section 5.5. The database schema of the system is shown in Appendix A.

5.1 Interface

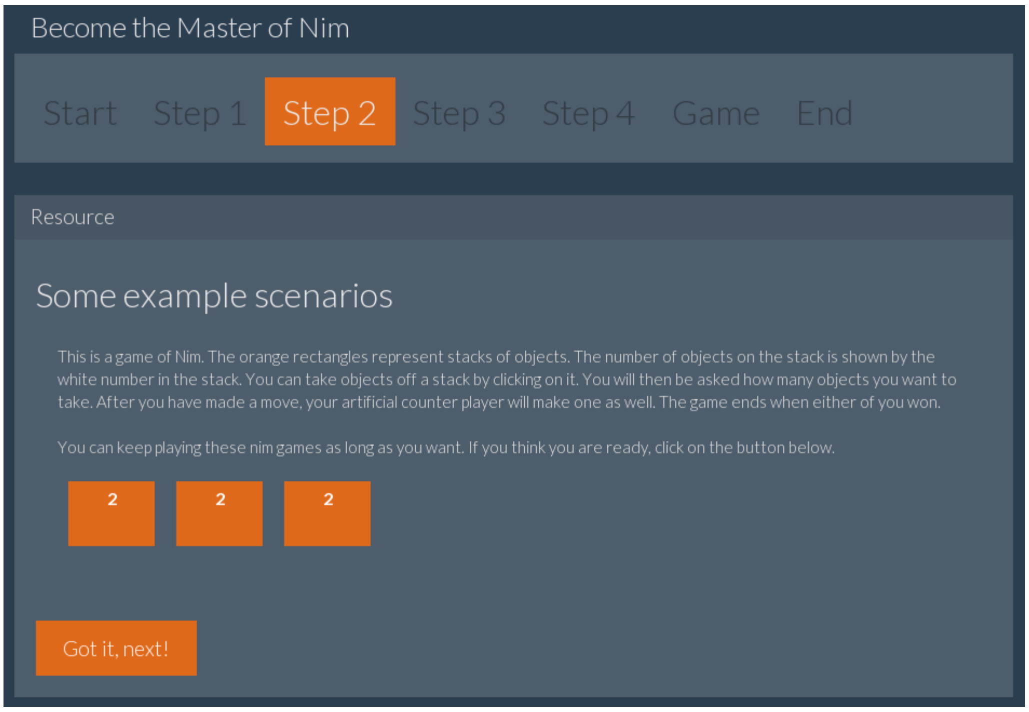
The **TutOER** software provides an online interface for students[2] which presents the educational material and assessments for each knowledge component to the student. The interface can be seen in Figure 5.1 and consist of two main boxes. The top box indicates how far the student is in the course, which is based on the knowledge component the student is currently in. The middle box contains either the educational resource or the test questions. The middle box also always contains a button through which the user can advance to the next page. Section 5.2.1 describes the user flow through the system, which determines the result of clicking the button. The educational content is displayed in an iframe, which means that the content could also be an independant online resource. Most open educational resources are of that nature at the moment. That being said, the experiments described in Chapter 7 only utilize material that was made by the author and specifically designed to fit within the layout of the **TutOER** interface.

5.2 Tutor Module

The tutor module contains all program logic and database models related to the educational task. It handles all interactions with the student and connects with the genetic algorithm module. This module is responsible for implementing the user flow, through which the student is guided towards the end of the course. This flow is described in Section 5.2.1.

[1]Software can be found on `https://github.com/mslatour/oertutor`

[2]The term student is used in the broadest sense: anyone who wants to learn something



Some example scenarios

This is a game of Nim. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. You can take objects off a stack by clicking on it. You will then be asked how many objects you want to take. After you have made a move, your artificial counter player will make one as well. The game ends when either of you won.

You can keep playing these nim games as long as you want. If you think you are ready, click on the button below.

Figure 5.1: Screenshot of the application presenting an education resource.

5.2.1 User Flow

The software enforces a specific flow through the system on the student. This flow is divided up in phases. The path between the phases is shown in Figure 5.2. The button described in Section 5.1 almost always triggers a change in phase, as denoted by the arrows in the figure. The rest of this section explains the different phases and the exact effect of clicking on that button in each phase.

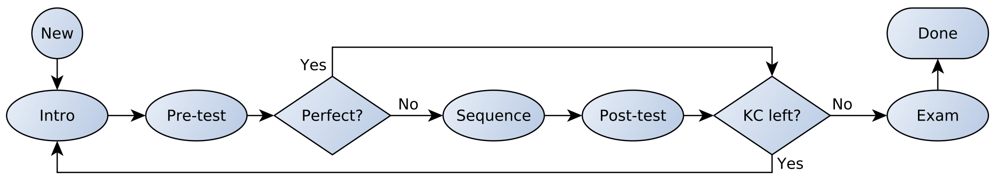


Figure 5.2: Diagram depicting the phases that a user goes through and the path between them.

New A student that is new to the system is shown an explanation of the course. Provided a student does not clear the stored cookie in the browser or switches browsers altogether, this phase only occurs once in the interaction between the student and the software. A button is shown to start the course, which would put the student in the introduction phase.

Introduction The introduction phase presents the student with the description of the current knowledge component. The knowledge component is either the first one, or the knowledge component set in later phases. The introduction phase is encountered for each knowledge component, provided the student finishes the course. A button is shown to move to the pre-test phase.

Pre-test In the pre-test phase the knowledge of the student on the current knowledge component is assessed. It shows all the questions at the same time underneath each other on one page. The student is not forced to answer the questions by form validation or otherwise which refuses to submit a test without an answer for each question. The button shown at the buttom submits the answers given to be graded. Based on the score, the student will either move on to the sequence phase or, if the score is perfect, the student will skip the current knowledge component. If the knowledge component is skipped, the student will either be sent to the introduction phase of the next knowledge component or, if this was the last knowledge component, move forward to the exam phase.

Sequence The student in the sequence phase is presented the sequence of educational material that has been selected by the system[3]. Sequences can contain more than one educational resource. In that case, only one resource will be shown at a time, starting with the first of the sequence. A button is displayed which will bring the student to the next sequence, if there is one. If the student has reached the end of the sequence, the button will sent the student to the post-test phase.

Post-test The post-test phase displays the questions of the post-test for the current knowledge component. The appearance, the questions and the button function is identical to the situation in the pre-test phase. When the answers are graded, the normalized learning gain is calculated to feed back into the genetic algorithm as fitness value. If there is a next knowledge component in the course, the student is sent to the introduction phase of that knowledge component. If this was the last knowledge component, the student is sent to the exam phase.

Exam When the student has passed through all phases of all knowledge components (or skipped them), he or she is sent to the exam phase. In this phase an exam is presented to the student that needs to be completed before the student can move on. The exam grade is not used in any way by the genetic algorithm, but merely provides a way to evaluate the level of the student after having interacted with the system. When the exam has been submitted, the student is sent to the done phase.

Done When all other phased have been completed, a student enters the last phase. Here a questionaire is presented to the student, which is optional to fill in. In the experiment there are two types of participants, one group is coming via Amazon Mechanical Turk and the other through a different source. The group from Mechinal Turk is shown a button to return to the Mechinal Turk website to collect their reward, while at the same time submitting the answers to the questionaire, regardless of whether they are empty. The other group is shown a button to submit their questionaire answers, but there is no side-effect. The student remains in this phase and has finished his or her participation. This is also explained to the student.

5.3 GA Module

The genetic algorithm (GA) module is responsible for selecting the sequence of educational material to be presented to a student. This module is separate from the tutor module in order to be replaceable by a different approach, as was already needed earlier in the thesis process when this module replaced its predecessor that applied a Markov Decision Process. The module implements the approach described in Chapter 4, but several adjustments were made to the standard genetic algorithm in order to make it work in a web-based environment. These adaptations resulted in the web-based genetic algorithm as described in Section 5.3.1.

5.3.1 Web-based genetic algorithm

The implementation of the genetic algorithm had to be adjusted in order to be applicable in a web context. In particular, the asynchronous nature of the HTTP protocol requires an adapted step-wise version of the straightforward implementation using loop constructs.The reader can compare this situation with that of a parallel implementation where fitness evaluations are run in parallel threads, where in this analogy the students play the role of the parallel threads. In this parallel case, it is

[3]See Section 4.2.3.2

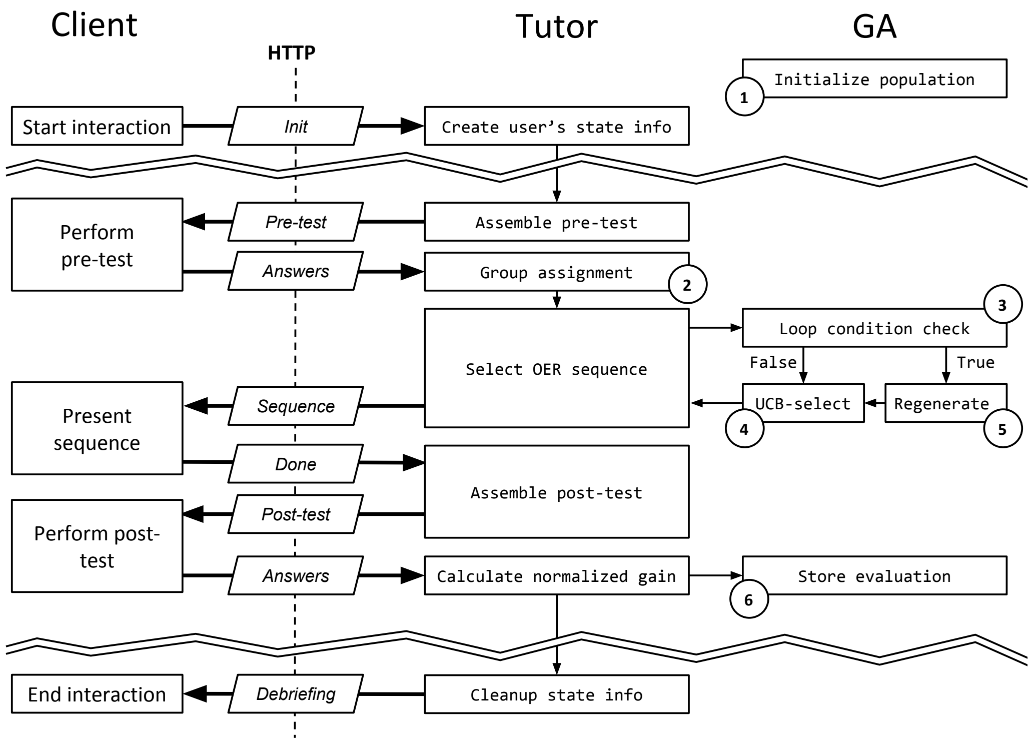


Figure 5.3: Schema of client-server interaction in a web-based adaption of the genetic algorithm loop

already necessary to deal with fitness values being sent back from the threads in a different order than the threads were started in. A difference between the analogy and the actual situation is that the students are not guaranteed to provide a fitness value, since they can decide to close the website. That difference is important because it complicates the decision on when enough sequences have been evaluated, since you don't know whether a sequence you assigned to a student for evaluation will actually be evaluated by that student or whether the system needs to assign it to another student.

The web-based genetic algorithm is best described by its interactions with one student. The entire list of interactions between the student's browser and the **TutOER** software is given by the phases described in Section 5.2.1. Only the pre-test, sequence and post-test phases are of interest for the web-based genetic algorithm. Figure 5.3 shows the relevant parts of the client-server interaction between the student's browser and the **TutOER** software. Note however that these interactions are likely to be interrupted by interactions with other students. The figure also shows the communication between the tutor module and the genetic algorithm. There are six numbered components in the diagram. The rest of the section describes these components in more detail.

1: Initialize population Before any interaction occurs with students, the population of the genetic algorithm is initialized. This is done for all populations that are required later on and do not require a trigger from the client. As such it is not an interaction, but it has been added to this list for completeness.

2: Group assignment The pre-test grade is used to determine which student group the student will be assigned to. Each student group and knowledge component combination is captured in a separate population in the genetic algorithm. The assignment of the student to a student group, within the context of a knowledge component, determines the population from which an individual will be selected to be evaluated.

3: Loop condition check In a parallel implementation you would start each evaluation thread in a loop, iterating for the desired amount of episodes. Translating that to this situation would mean that the system somehow activates the student to evaluate it. In a web-based implementation, everything is client-driven. Nonetheless, the genetic algorithm still consist of an, at least implicit, loop for each desired episode. This component is designed to bring the two together. It checks how many evaluation episodes have been stored in the current generation and compares that to the desired total number of episodes. If the total has not been reached yet, the UCB-select component is executed. If enough evaluations have been collected however, the implicit loop has reached its termination condition. This means the population must evolve to a new generation, which happens in the regerate component.

4: UCB-select For the largest part this component is identical to what it would be in any other implementation. It uses the UCB-1 formula to select which sequence of educational resources it wishes to evaluate, while balancing exploration and explotation. Because the evaluation is done asynchronously, takes up significant time and is client-driven, it is necessary to keep track of which sequences have already been selected by UCB and assigned to a student. This in order to prevent UCB from unintendedly assigning the same sequence to many students, simply because the evaluation results have not come back yet. Therefore sequences are locked when they are assigned to a student and UCB can only choose from the sequences that are not locked. This is probably similar to what you would do in a parallel implementation.

However, unlike in a typical parallel implementation, it is not at all guaranteed that a student will actually study the entire sequence and submit the post-test afterwards. When a student decides to stop participating for whatever reason, the sequence that was assigned to the student would be forever locked. That could result in a situation where there are not enough evaluations stored to proceed to a next generation, but since all sequences are locked UCB has no way of assigning sequences to students. This is solved as follows. First, UCB attempts to select a sequence that is not locked. Second, if that is no longer possible, the oldest lock is discarded. Third, the method is tried again. This could mean that the system wrongly decides to assign the sequence to another student. With the consequence that a sequence is evaluated more often than UCB chose to. On the other hand, the lock could have prevented the UCB in the first place from being able to deliberately select the sequence twice. This mechanism is thus choosing between two evils, however it will likely not be applied often.

A side-effect of this is that an evaluation result could actually be submitted to the genetic algorithm when it already moved to a new generation. The result of this is stored in connection to the new generation, which means that effectively a sequence that was not part of the new generation could still be evaluated. The sequence could not be selected as one of the survivors at the next generation switch, but its fitness value is still stored and can be used whenever the sequence reappears due to combination or mutation. This seemed to be the best solution.

5: Regenerate When enough evaluations have been gathered, this component executes the generation switch for the current population as described in Section 4.2.4. The Django web framework that was used to implement the **TutOER** software should ensure that during this process the database tables were locked, preventing synchronization issues. This was however not tested. After a the new generation has been formed, the loop condition check component is retried again.

6: Store evaluation When a post-test is graded and the normalized learning gain is calculated, the resulting fitness value is stored for the sequence that was evaluated in the context of the current generation. This could be a different generation than the one the sequence was selected from.

5.4 Monitor Module

The monitor module provides a real-time insight in the relevant processes, events and data stored in the database. This module has only been used to monitor the experiment while it was running and contained three views: *log*, *student* and *population*. The log view showed a long list of logged events, as described in Section 5.5. The student view showed a list of the logged events related to a particular student. It also showed an overview of the test scores, the assigned student group and the presented sequence for each knowledge component the student participated in so far. The population view displayed a list of the individuals in each generation for a specific population. This included observed fitness values and the cumulative regret graph for that population.

5.5 Logging Module

All logging of events is done by the logging module. This module is tighly connected to the tutor and genetic algorithm modules and provides input to the monitor module. The tutor and genetic algorithm modules send out signals when certain events occur, these signals are picked up by the logging module and stored in the database as log entries. These log entries are then presented again by the monitor module. The purpose of the logging activities is to aid monitoring and analysis of what goes on within the **TutOER** software and its modules. The following events are stored in the log entry table.

Error Any unexpected errors that were caught in the system are logged by the module.

New participant from external source The module logs each new participant that came from an external source. In particularly it stores the identifiers of people participating via Amazon Mechanical Turk, in order to be able to reconstruct the connection between the student object in the database and the Mechanical Turk user in case something went wrong. This was especially applicable during the experiment period.

Change in student state When a student object is created in the system en each time the student enters a new phase in the user flow, the module logs this.

Trial created A trial is the particular instance of the tutor teaching actions for a specific knowledge component and a specific student. The trial object contains the pre-test and post-test results and the sequence that was presented to the student. The module logs when this object is created it.

New generation in population Each time a population moves to a new generation, this is logged by the module.

New immigrant in the population The module logs each time an individual immigrates in a population. The individual that the immigrant replaced is also stored in the log entry.

5.6 Bootstrap values after restart

The experiment had to be restarted after 492 evaluations were already gathered. This was due to some technical changes that had to be made. No changes were made to the resources or the assessments. As such the gathered evaluations would still be valid observations. In order to reuse these evaluations at least partly, the **TutOER** system was equiped with a bootstrap mechanism. This mechanism reuses evaluations from earlier experiments where possible.

The technical changes affected the search behavior of the genetic algorithm. It is therefore not likely that all evaluations will be reusable. In fact only when UCB-1 selects a particular sequence to be evaluated, that evaluation could be bootstrapped. The bootstrap procedure extends step 4 in Figure 5.3 and goes as follows.

1. Retrieve UCB-1 selection
2. If selected sequence is present in unused bootstrap values,
   a) Submit bootstrap value as if a student evaluated it at that moment
   b) Return to step 1.
3. Else, present sequence to student.

6

Simulations

Apart from a modeling of the domain, genetic algorithms also require several parameter values to be set. These values can, together with modeling decisions, have an enormous impact on performance. It is therefore important to gain more insight in the properties and effectiveness of the genetic algorithm approach under various parameter values. This chapter describes and analyses a series of simulations that explored this. The goal of the simulations is to find good parameter values for the experiment. Additionally they provide some insight in the behavior and performance of the **TutOER** system after a larger number of evaluations that would be possible in an experiment. The simulations are executed using the same software that is used for the experiment. However, for the sake of simplicity and speed the actual web interaction was left out of the simulation. Sequences are therefore evaluated using a handcrafted model instead of using actual students. Each simulation is repeated ten times to deal with the random components of the algorithm.

This chapter is organized as follows. The parameters that are examined are described in Section 6.1. The simulations that cover these parameters are enumerated in Section 6.2. Section 6.3 describes the general setup of each simulation. In Section 6.4 the simulation results are listed and an analysis presented of the findings.

6.1 Parameters

The approach described in Chapter 4 requires several parameters to be set in advance. These parameters influence the behavior and performance of the algorithm and are often not independent. This section describes the effect of the four parameters that need to be set: *population size*, *number of episodes*, *number of elite* and *mutation rate.*

Population size In the genetic algorithm modeling of the thesis the population has the same amount of individuals in each generation. At the end of the generation, fitness-based selection chooses individuals to produce offspring using crossover operations. The resulting offspring is then potentially mutated. The amount of crossover and mutation operations are influenced by the *population size.* A larger value allows for more diversity in the population, although it is not a guarantee. Diversity is important when dealing with the possibility of local optima. However, a smaller value allows for a more directed search towards an optimal solution.

Number of episodes The number of evaluations that will be assigned by UCB-1 within a single generation is referred to as the *number of episodes.* A higher value of this parameter raises the certainty of the estimated fitness values of each individual. This certainty is important when the fitness value is subject to noise. UCB-1 however only ranks individuals and does not express whether more evaluations will be beneficial. If the fitness of the individuals in the current generation is rather low, a higher number of episodes will also increase the cumulative regret. Furthermore, the number of episodes determines the “duration” of a single generation. A lower number of episodes results in more completed generations, which is important for effective search.

Number of elite Elitism, as described in Section 4.2.4, preserves the best performing individuals of the previous generation. The *number of elite* members reduce the number of crossover operations, since elite members are maintained as is and are not replaced by their offspring. Moreover elite members are protected against mutation. A higher number of elite members reduces the influence of randomness on the population. However, elite members slow down the exploration. A high number of elite members will cause the algorithm to get stuck in local optima.

Mutation rate The *mutation rate* determines the chance that a mutation occurs in the chromosome of a new individual. Mutation ensures that areas of the search space are explored, even when the genetic algorithm moved in a different direction. When this parameter value increases, more exploration occurs. Setting this parameter too high will result in a lack of convergenge.

6.2 Simulation setups

The parameter values of the genetic algorithm that need to be determined by simulation are combined in parameter setups. Each setup contains a specific combination of values for each parameter. The parameter setups are shown in Table 6.1. Two groups of six have been formed out of these twelve parameter setups. The analysis of the **TutOER** system under specific parameter setups is done per group. Group 1 contains the first six parameter setups and covers the interplay between *population size* and the *number of episodes*. Group 2 contains the second six parameter setups and covers the interplay between the *number of elite* and the *mutation rate*. Each parameter setup is labelled with a systematic name for later reference.

*Table 6.1: Parameter setups for the **TutOER** system divided in two groups*

Label	Population size	# Episodes	# Elite	Mutation rate
`pop5ep5`	5	5	2	0.05
`pop5ep10`	5	10	2	0.05
`pop5ep20`	5	20	2	0.05
`pop10ep10`	10	10	2	0.05
`pop10ep20`	10	20	2	0.05
`pop20ep20`	20	20	2	0.05
`el0mu05`	10	10	0	0.05
`el1mu05`	10	10	1	0.05
`el2mu05`	10	10	2	0.05
`el0mu25`	10	10	0	0.25
`el1mu25`	10	10	1	0.25
`el2mu25`	10	10	2	0.25

The parameter setups are tested in two artificial environments. The first provides evaluation outcomes using the handcrafted model displayed in Table 6.2. The handcrafted model contains patterns that match to sequences, the explanation of these patterns can be found in Table 6.3. Sequences that are not matched by any pattern get a fitness of 0. The model is based on the author's expectation of the true quality of the sequences containing resources about the intuition of the game. The first environment will be referred to as the *normal environment.*

The second environment uses the same handcrafted model as the first environment, but adds gaussian noise of 0.2 standard deviation. The purpose of the second environment is to see how the performance of the **TutOER** system under each paramater setup is changed when the observed fitness contains noise. The second environment will be referred to as the *noisy environment.*

In all simulations the sequences are restricted in length. Each sequence must contain one, two or three resources. This restriction is the same as in the experiment described in Chapter 7.

Table 6.2: Fitness values for each sequence.

Sequence pattern	Fitness
`[5]`	0.7
`[6]`	0.8
`[7]`	0.1
`[8]`	0.8
`{6, ( 5 | 8 )}`	1
`{5, 6, 8}`	1
`{5, 6, 7}`	0.6
`{*, *, ( 5 | 6 ), ( 5 | 6 )}`	0.7

Table 6.3: Explanation of sequence patterns.

Construct	Meaning
`[5]`	Resource 5.
`[*]`	Any resource.
`[5, 6]`	First resource 5 followed by resource 6.
`{5, 6}`	Resources 5 and 6 in any order.
`[( 5 | 6 )]`	Either resource 5 or 6.
`{7, ( 5 | 6 )}`	Resources 7 and either 5 or 6 in any order.

6.3 General setup

In order to compare the results from simulations more easily, each simulation is terminated after 1000 evaluations. This will give a genetic algorithm time to complete 100 generations with 10 evaluations in each generation. Similarly, a genetic algorithm setup with 20 evaluations in each generation will only complete half as much generations. The comparison between the two might not seem fair when looking at the number of generations. However, the task at hand defines the evaluation to be a scarse resource that will require a new student each time. It is therefore more interesting to know what a particular parameter setup accomplishes in a limited amount of evaluations, rather than anything else. The number of generations that have been completed after this fixed amount of evaluations is thus a consequence of the parameter setup. Section 6.1 describes the trade-off between number of generations and number of episodes that follows from this restriction.

6.3.1 Evaluation metrics

To aid in the analysis of each simulation result, three views are applied to the data: *the cumulative regret curve*, *the coverage curve* and *the convergence plot*.

Cumulative Regret Curve Recall that regret is defined as the difference in received reward (i.e. learning gain) between the presented sequence and the optimal sequence. The cumulative regret curve shows the built-up of regret after each evaluation. It provides insight in the handling of exploration versus exploitation of the system under a particular parameter setup. Normally, the use of the UCB-1 selection algorithm ensures that this curve decreases logarithmically over time. However, due to the combination with the genetic algorithm, the options from which the UCB-1 algorithm can choose are limited. The interplay between the genetic algorithm and UCB-1 can either boost or reduce performance. This curve visualizes the amount to which the performance of UCB-1 is disrupted by this interplay. Each value of the curve is an average of the values of ten repetitions on the same point. At each point the maximum and minimum value are indicated using errorbars.

Coverage curve The coverage of the system is defined as the percentage of all possible sequences that was evaluated at least once. The coverage curve shows how this percentage is built-up per evaluation. In other words, the coverage curve shows when an evaluation explored uncharted territory of the search space. New sequences become part of a generation due to crossover, mutation and immigration. UCB-1 ensures that all sequences that are selectable will be evaluated at least once[1]. Therefore, this curve provides an observation of the amount and timing of the introduction of new individuals in the population. Similar to the cumulative regret curve, each value is an average of the values of ten repetitions on the same point. At each point the maximum and minimum value are indicated using errorbars.

[1]Provided the number of episodes is not smaller than the number of individuals in each generation.

Convergence plot The **TutOER** system is said to be converged when ten evaluations in a row regarded an optimally performing sequence. A sequence is optimal when its true fitness value is equal to the maximum. In the case of multiple optimal sequences, an evaluation of any of the optimal sequences counts as the same optimal evaluation. This could mean that the system is said to have converged after a series of ten evaluations of different optimal sequences. The flexibility in this definition is required, because otherwise the system could be alternating between two equally optimal sequences until termination without being said to converge. The number of evaluations needed to reach that convergence point in each repetition is plotted in a boxplot. The fact that only one datapoint derives from each repetition allows for this detailed look at the distribution. Parameter setups that ensure a quick convergence are desired when acquiring evaluations comes with a cost.

6.4 Results

6.4.1 Group 1 (Population size & # Episodes)

Analysis

Figure 6.1 shows the cumulative regret built-up by the **TutOER** system under the parameter setups of Group 1 in the normal environment. Each simulation setup has been repeated ten times. Each individual plot shows the averaged cumulative regret after each evaluation, together with vertical error bars indicating the maximum and minimum value at every twenty evaluations. The `pop5ep20` setup stands out with a much higher cumulative regret built-up than the others. Due to the number of episodes in the `pop5ep20` parameter setup, only 50 generations have been completed after 1000 evaluations. Also the `pop10ep20` and `pop20ep20` setups complete 50 generations. In comparison, the `pop5ep5` setup ensures the completion of 200 generations in the same number of evaluations. The difference between the number of episodes and the number of individuals in each generation is the largest for the `pop5ep20` setups. Recall that UCB-1 selection is applied to determine which individuals are evaluated. In the `pop5ep20` setup, UCB-1 has very few individuals to choose from (namely five) and relatively many evaluations to assign (namely twenty). This will allow the **TutOER** system to aquire fitness estimates with much more confidence. However, when none of the five individuals performs optimal, the **TutOER** system is forced to built-up regret until the end of the generation before the genetic algorithm can continue its search. This is true for all paramater setups, but because the number of individuals in each generation is relatively low compared to the number of episodes in the `pop5ep20` setup, it is more likely to happen than in a setup where that difference is smaller.

The `pop20ep20` setup is an example where that difference is much smaller. It has the same amount of episodes as the `pop5ep20` setup but a higher number of individuals in each generation (namely twenty). That allows for more diversity in a generation. This has no effect in the first generation, because the population is initialized with sequences of only one resource. Since there are only four resources in the population, the first generation under the `pop20ep20` setup will contain sixteen duplicates. From the perspective of the UCB-1 selection, this has the same effect as in the `pop5ep20` setup. However, after the first generation there are more possibilities for exploration. This is also reflected in the percentage of sequences that have been observed. Figure 6.3 shows the percentage of possible sequences that have been encountered by the **TutOER** system under the various parameter setups within Group 1. The `pop20ep20` setup causes the most exploration. Followed by the other parameter setups of Group 1 in decreasing order of their *population size* parameter. However, some of the differences are too close to call. When comparing parameter setups with the same *population size*, a second ordering becomes apparent. Parameter setups with a high number of episodes (e.g. `pop5ep20`) have encountered a lower percentage of sequences than parameter setups with a lower number of episodes (e.g. `pop5ep5`). This reverse relation comes from the fact that a higher number of episodes means the genetic algorithm will complete less generations in the same amount of evaluations. Completing less generations means less crossover and mutation applications, and thereby also less opportunities for diversity.

The observation of these orderings in relation to the number of episodes and the population size also hold in a more noisy environment. Figure 6.8 shows the percentage of sequences encountered under parameter setups of Group 1 in the second environment. Noise on the observed fitness values should not have a direct impact on the level of exploration in the genetic algorithm setup. What it could affect however is the direction of exploration and the regret as a result that. Figure 6.2 shows the cumulative regret built-up by the **TutOER** system under the parameter setups of Group 1 in a noisy environment. The differences with the results in the normal environment are not that big. The parameter setups are slightly more apart in terms of cumulative regret, but that difference is made early on in the simulation (Figure 6.2g). Given that the lines plotted in the overview are averages of ten repetitions, there could also be an alternative explanation. For example a mutation may have occured at a different time causing a different outlier in the simulated cumulative regrets. This means that there might actually be no effect of the added noise on the performance of the **TutOER** system under the parameter setups of Group 1. The impact of the values for the mutation rate and number of elite in all parameter setups of Group 1 on this noise resillience will be discussed in Section 6.4.2.

Figure 6.6a shows the boxplot for the ten repetitions of the number of evaluations needed to converge under each parameter setup in Group 1. The actual numbers of the convergence at each repetition can be found in Appendix D. The **TutOER** system converges the fastest on average under the `pop10ep10` parameter setup. Parameter setups `pop5ep5` and `pop5ep10` follow in second and third place, even though they have a better median value. The one that stands out is the parameter setup `pop5ep20` with extreme outliers as well as a relatively high median value. However, apart from one trial that took 769 evaluations to converge, all trials under any parameter setup of Group 1 converged within 100 evaluations. In other words, all simulation setups resulted in sticking to the optimal sequence of OER for at least ten consecutive evaluations using less than 100 simulated students. And on average almost all parameter setups needed less than 50 simulated students to converge. However, when the number of repetitions is small the outliers matter. Therefore `pop10ep10` and `pop20ep20` should be favored over `pop5ep10` and `pop5ep5`. Figure 6.6b shows the boxplot distribution of the same parameter setups in a noisy environment. The same parameter setups have outliers, with roughly the same number of evaluations that are needed to converge. Parameter setup `pop10ep10` outperforms the rest, including `pop20ep20`, on convergence in this noisy environment.

Conclusion

From the plots of cumulative regret, coverage and convergence can be concluded that `pop10ep10` scores best overal. The **TutOER** system had on average the lowest cumulative regret under the parameter setup `pop5ep5`, but `pop10ep10` and `pop5ep10` followed in second and third place. The most unique sequences were encountered on average by the **TutOER** system under `pop20ep20` parameters, followed by the `pop10ep10` parameters. In terms of convergence, the `pop10ep10` parameters were most favorable for the **TutOER** system. Thus, out of the parameter setups simulated, the population size and number of episodes of both 10 are overall best.

6.4.2 Group 2 (# Elite & Mutation rate)

Analysis

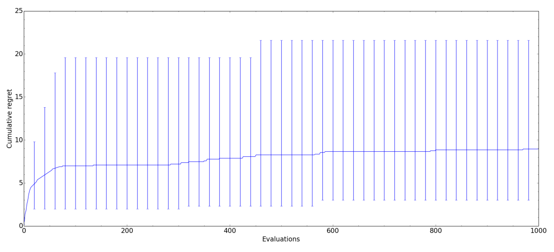
Figure 6.4 shows the cumulative regret of the **TutOER** system under the parameter setups of Group 2. A clear distinction can be made between parameter setups with a low mutation rate and those with a high mutation rate. The **TutOER** systems builds up less cumulative regret under `el0mu05`, `el1mu05` and `el2mu05` parameters than under their counterparts with higher mutation rates. This can be explained by the fact that a higher mutation rate typically means more exploration. Exploration in turn causes higher regret when the newly found sequences turn out to be less optimal than the current best. The number of elites doesn't seem to affect the cumulative regret that much in Figure 6.4. This is however different when operating in a noisy environment. Figure 6.5 shows the cumulative regret under the parameter setups of Group 2 in the second environment. Observing the cumulative regret under `el0mu05` parameters we see that regret rises significantly from around 100 evaluations. Figure 6.6d shows the worst convergence result under the `el0mu05` parameters is 80 evaluations. That means that the **TutOER** system already found the optimal sequence but lost it in a generation transition. Otherwise UCB-1 would have ensured that the cumulative regret curve continued its logarithmic path, by continuing to select the optimal sequence. Losing the optimal sequence in a generation transition can easily be explained by the fact that the `el0mu05` parameters set the number of elites to zero. With no elite preservation, the optimal sequence needs to compete in the fitness-based roulette wheel selection. In a noisy environment the observed fitness value of the optimal sequence can be lower than the observed fitness values of less optimal sequences. Additionally elitism protects the individual from being applied to crossover or mutation operators. In lack of this protection, the optimal sequence may have been subject to mutation or crossover and was thereby no longer present in the generation. The counterpart of `el0mu05` with a high mutation rate is `el0mu25`. The cumulative regret curve of `el0mu25` is indeed worse than the parameter setups with one or two elites preserved. However, curiously the difference is not as extreme as with `el0mu05`. One would expect that having no elite members would have a bigger impact when the mutation rate is higher given the larger risk of mutating an already optimal sequence.

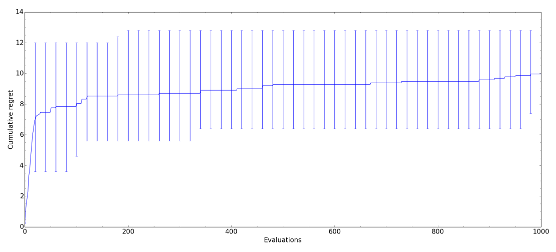
The parameter setup `el2mu05` has the lowest cumulative regret in both environments. The `el2mu05` parameter values are the most conservative of all. A high number of elites and a low mutation rate keeps the performance of the **TutOER** system largely unaffected by the added noise in the second environment. The consequence of a low mutation rate becomes apparent in Figure 6.7c. The percentage of sequences evaluated by the system is one of the lowest under the `el2mu05` parameters. Other parameter setups with a low mutation rate share the low coverage value. This is also what you would expect as a result of a low mutation rate. Given the similarly low cumulative regret under `el2mu05`, it is not necessarily a bad thing. Furthermore, the convergence is the best under the `el2mu05` parameter setup in both environments.

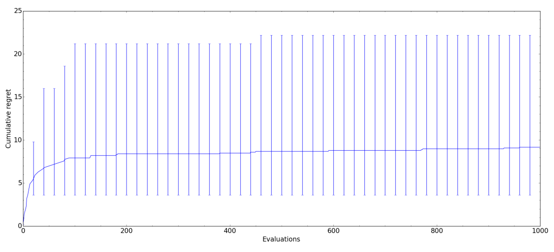
The parameter setup `el2mu25`, which also has a high number of elites, performs much worse than `el2mu05`. The cumulative regret under `el2mu25` is a lot higher in comparison, though better than the other parameter setups with a high mutation rate. The number of evaluations needed to converge under the `el2mu25` setup is comparable to others in its median value, but several big outliers were the results of `el2mu25` in both environments. The high percentage of sequences that were evaluated under the `el2mu25` values is the other side of that same coin.

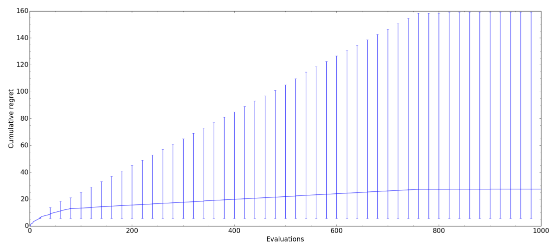
Conclusion

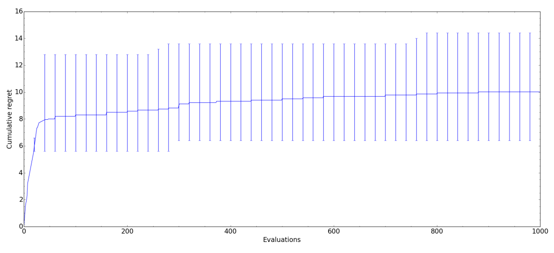
From the plots of cumulative regret, coverage and convergence can be concluded that `el2mu05` scores best overal. However, this conclusion is drawn while putting weight on the stability of a low cumulative regret and conversion points. The coverage of sequences appears to be mostly affected by a high mutation rate and to a lesser extend by a low number of elite members. The **TutOER** system performs understandably poorly under the `el2mu05` parameter setup. However, better coverage is only favorable when it leads to quicker convergence and lower regret. In both metrics, `el2mu05` outperforms the rest. Albeit with varying margins.

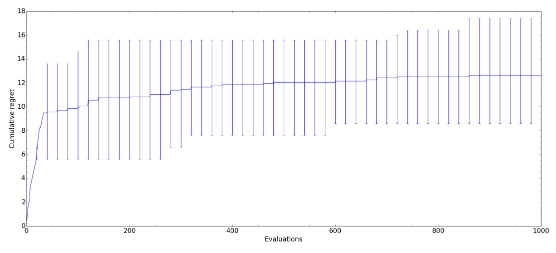

(a) Population size: 5, Nr. of episodes: 5

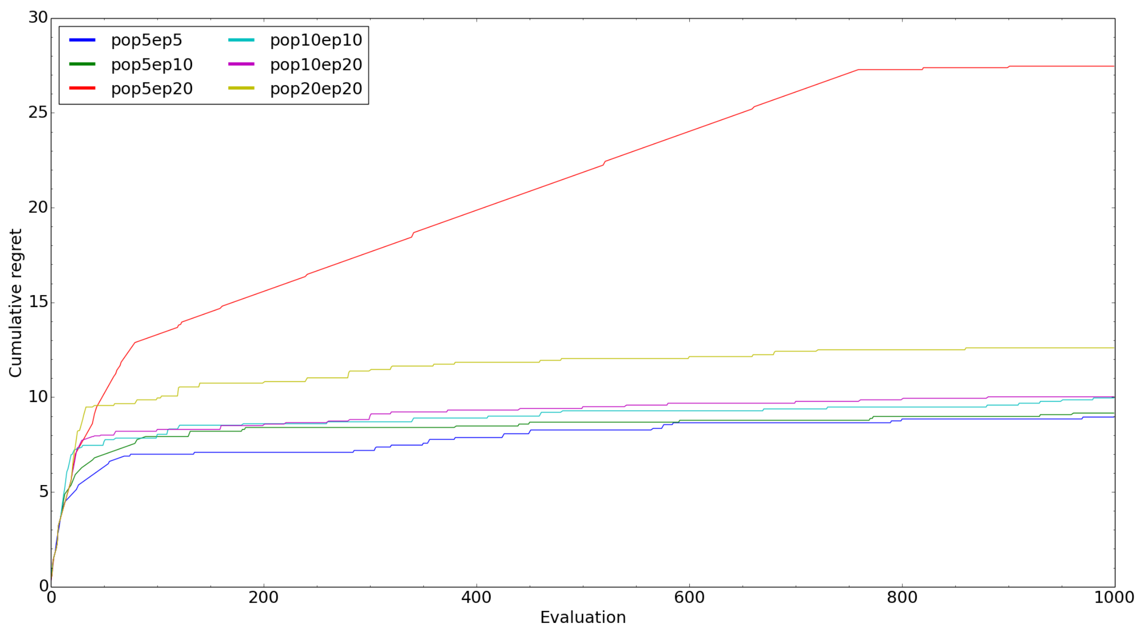

(b) Population size: 10, Nr. of episodes: 10


(c) Population size: 5, Nr. of episodes: 10


*(d) **pop5ep20** parameters*

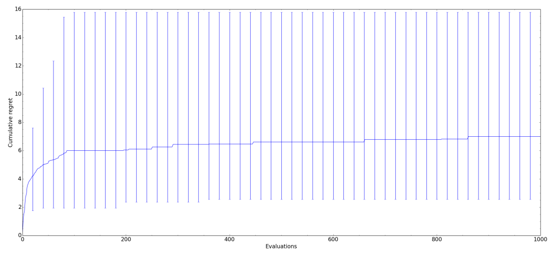

(e) Population size: 10, Nr. of episodes: 20


(f) Population size: 20, Nr. of episodes: 20

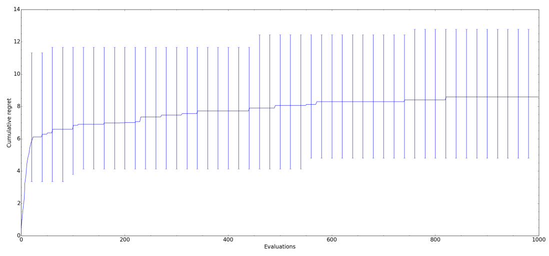


(g) overview

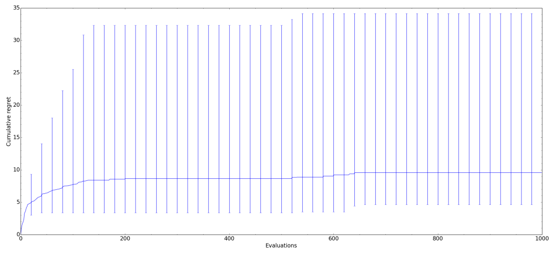
*Figure 6.1: Plot with error bars of the cumulative regret of the **TutOER** system using Group 1 parameter setups in the normal environment. Plot 6.1g shows the different parameter setups in one graph.*



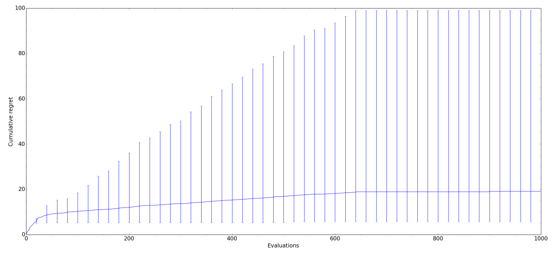
(a) Population size: 5, Nr. of episodes: 5



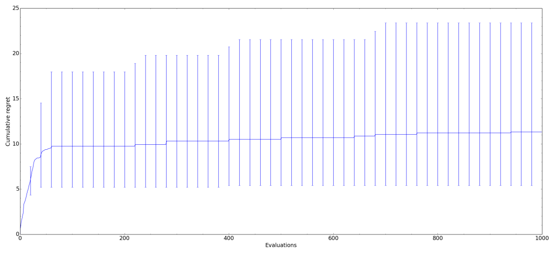
(b) Population size: 10, Nr. of episodes: 10



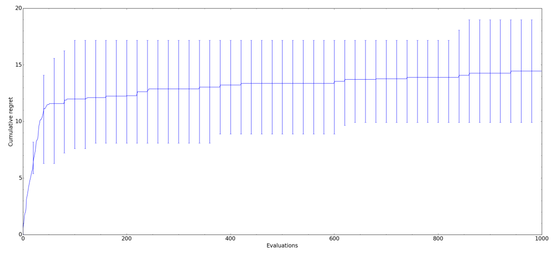
(c) Population size: 5, Nr. of episodes: 10



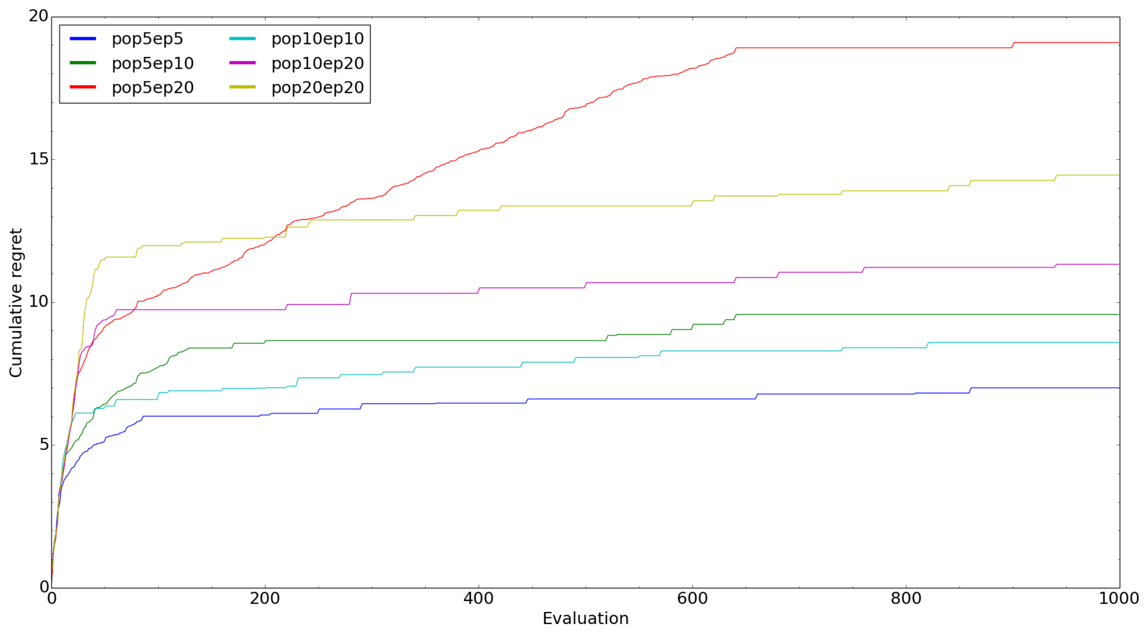
*(d) **pop5ep20** parameters*



(e) Population size: 10, Nr. of episodes: 20

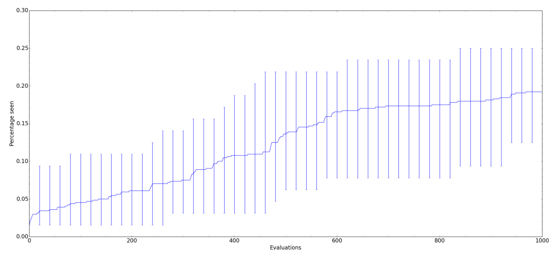


(f) Population size: 20, Nr. of episodes: 20

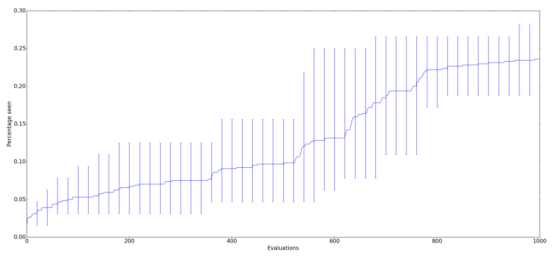


(g) overview

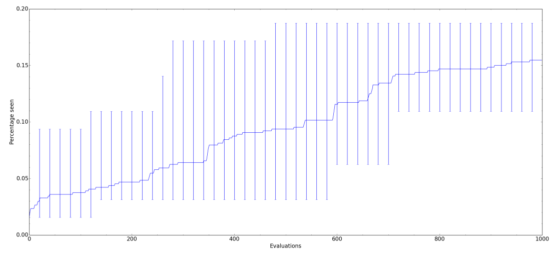
*Figure 6.2: Plot with error bars of the cumulative regret of the **TutOER** system using Group 1 parameter setups in the noisy environment. Plot 6.2g shows the different parameter setups in one graph.*



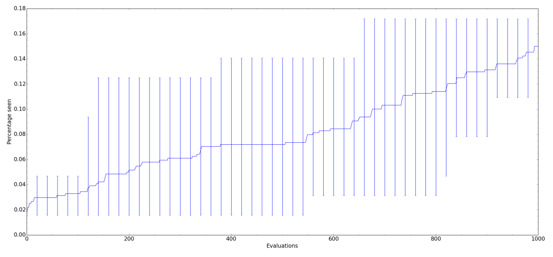
(a) Population size: 5, Nr. of episodes: 5



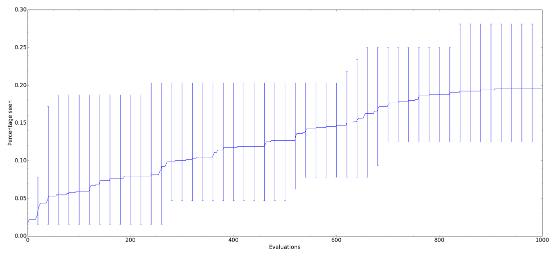
(b) Population size: 10, Nr. of episodes: 10



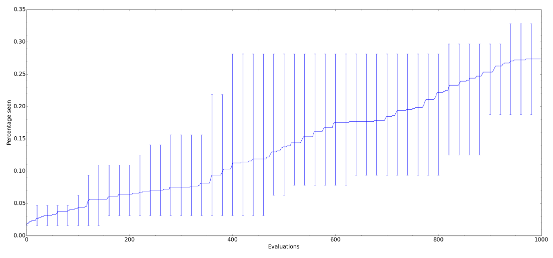
(c) Population size: 5, Nr. of episodes: 10



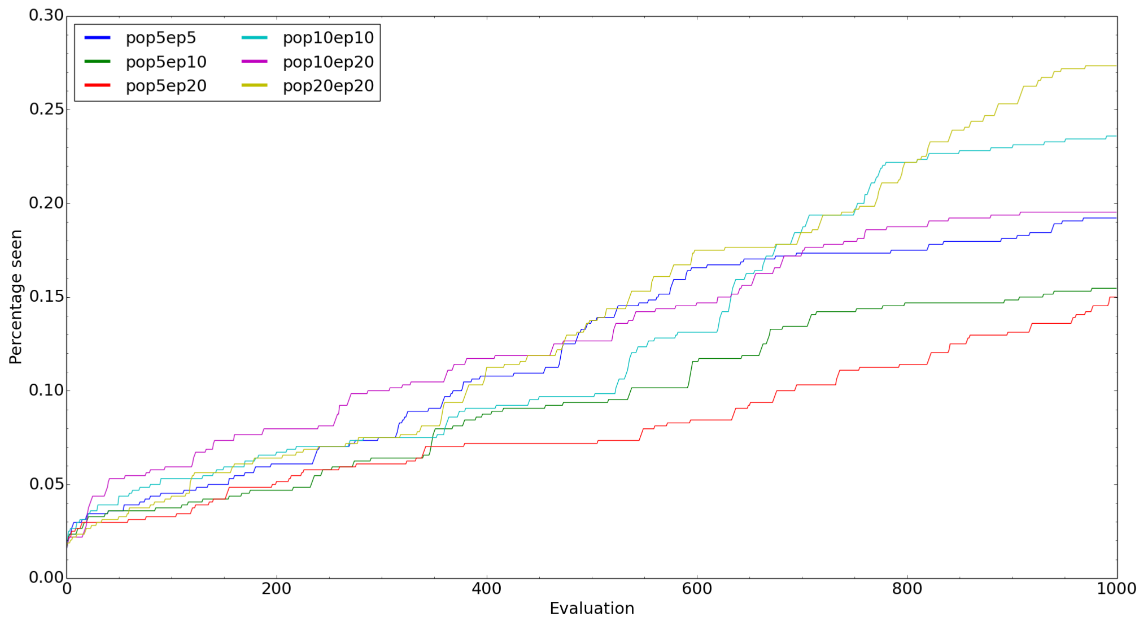
(d) `pop5ep20` parameters



(e) Population size: 10, Nr. of episodes: 20

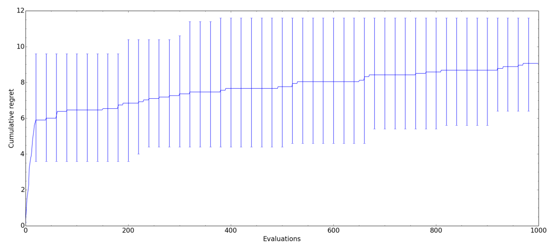


(f) Population size: 20, Nr. of episodes: 20

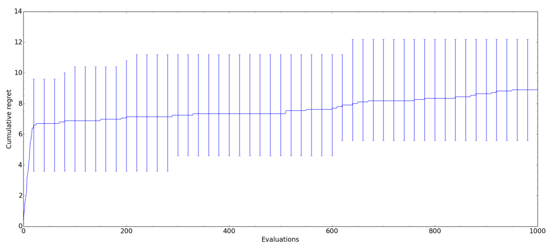


(g) overview

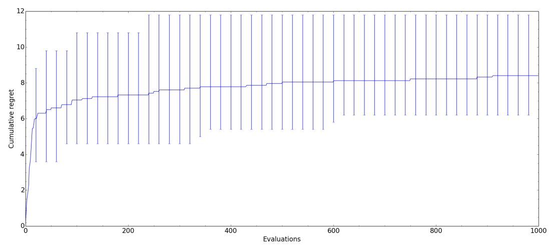
*Figure 6.3: Plot with error bars of the percentage of chromosomes seen by the **TutOER** system using Group 1 parameter setups in the normal environment Plot 6.3g shows the different parameter setups in one graph.*



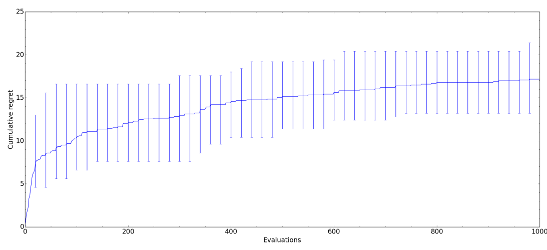
(a) Nr. of elite: 0, Mutation rate: 0.05



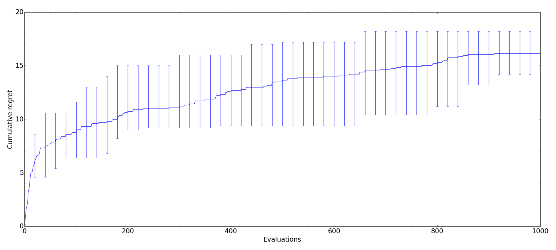
(b) Nr. of elite: 1, Mutation rate: 0.05



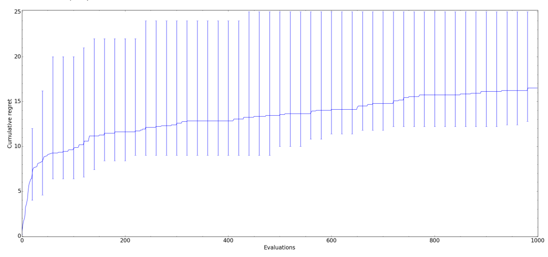
(c) Nr. of elite: 2, Mutation rate: 0.05



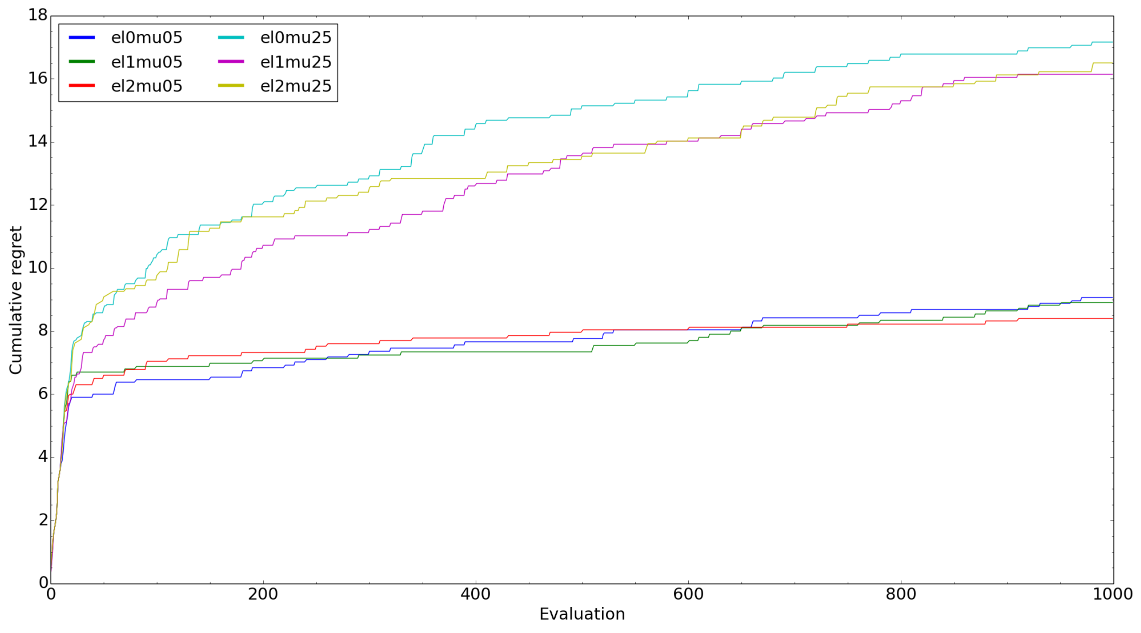
(d) Nr. of elite: 0, Mutation rate: 0.25



(e) Nr. of elite: 1, Mutation rate: 0.25

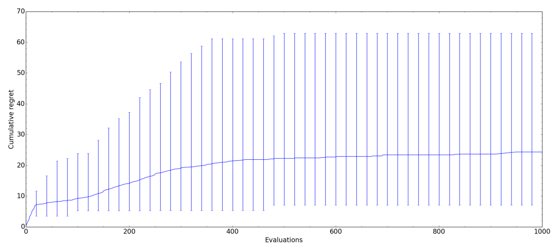


(f) Nr. of elite: 2, Mutation rate: 0.25

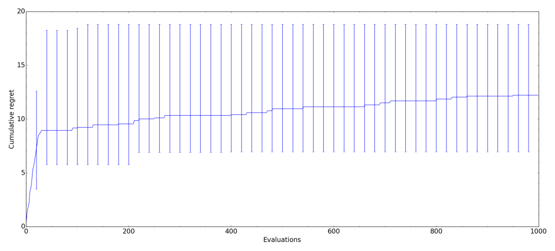


(g) overview

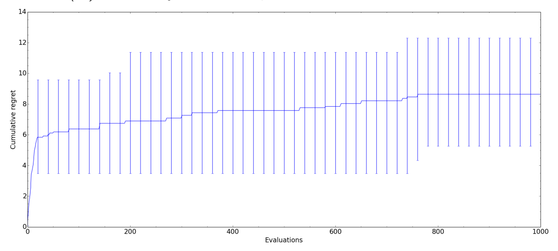
Figure 6.4: Plot with error bars of the cumulative regret of the ***TutOER*** *system using Group 2 parameter setups in the normal environment. Plot 6.4g shows the different parameter setups in one graph.*



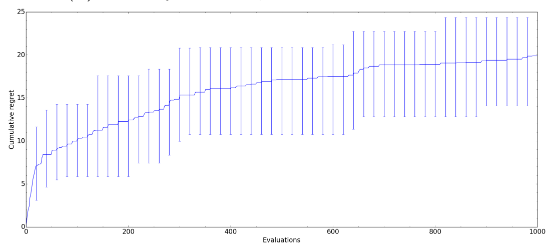
(a) Nr. of elite: 0, Mutation rate: 0.05



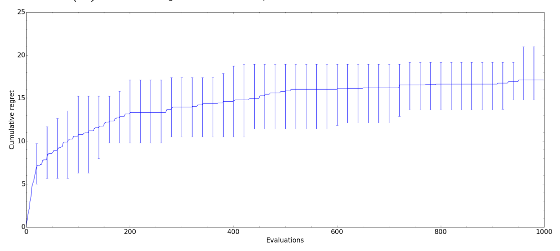
(b) Nr. of elite: 1, Mutation rate: 0.05



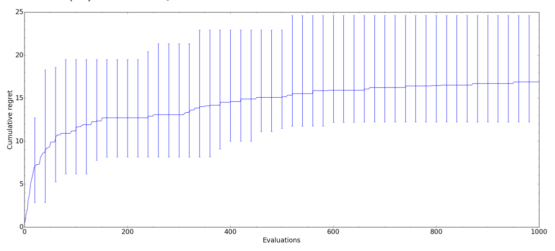
(c) Nr. of elite: 2, Mutation rate: 0.05



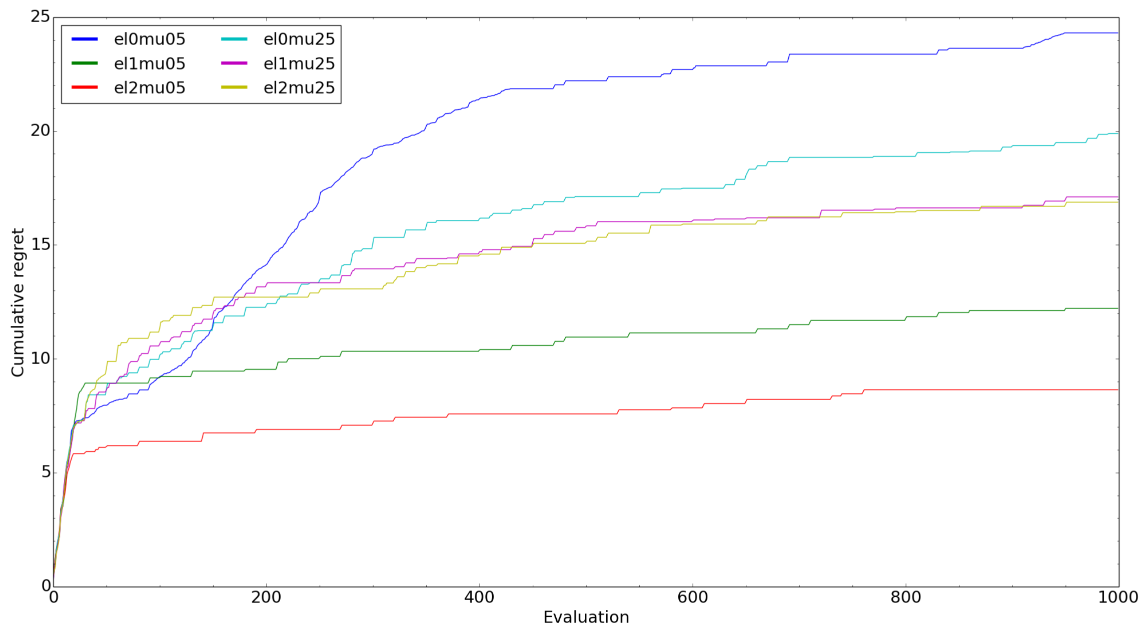
(d) Nr. of elite: 0, Mutation rate: 0.25



(e) Nr. of elite: 1, Mutation rate: 0.25

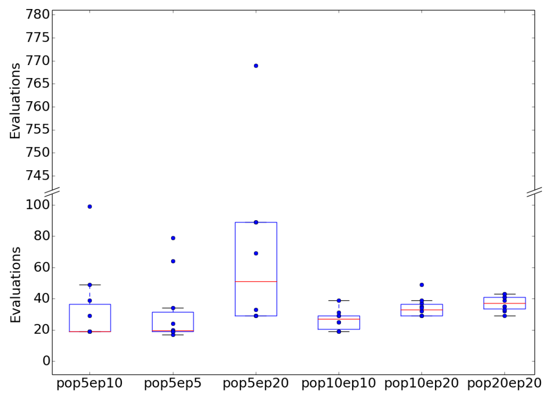


(f) Nr. of elite: 2, Mutation rate: 0.25

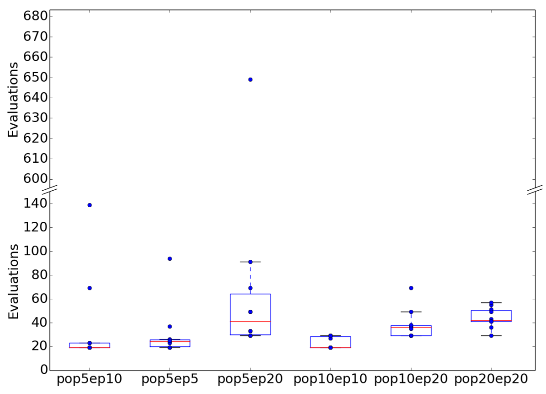


(g) overview

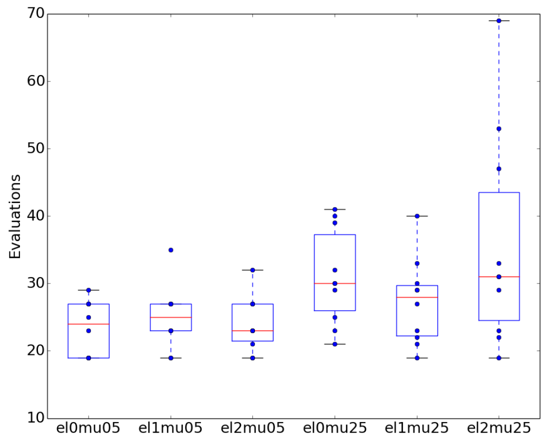
*Figure 6.5: Plot with error bars of the cumulative regret of the **TutOER** system using Group 2 parameter setups in the noisy environment. Plot 6.5g shows the different parameter setups in one graph.*



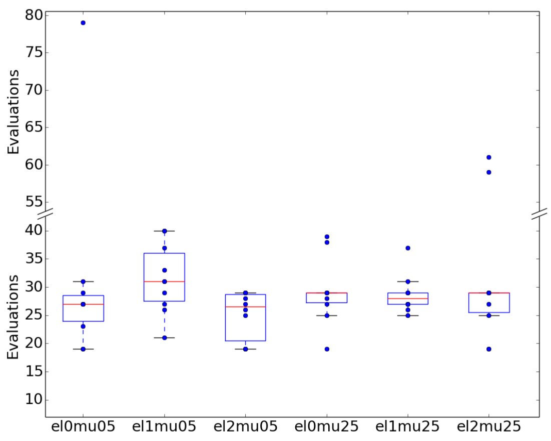
(a) Group 1 parameter setups



(b) Group 1 parameter setups, with noise

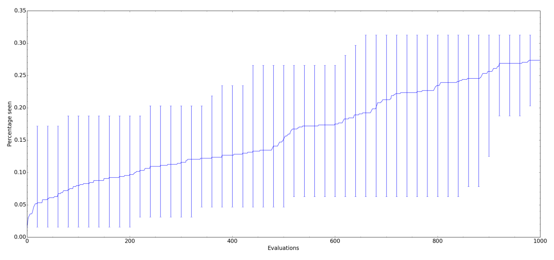


(c) Group 2 parameter setups

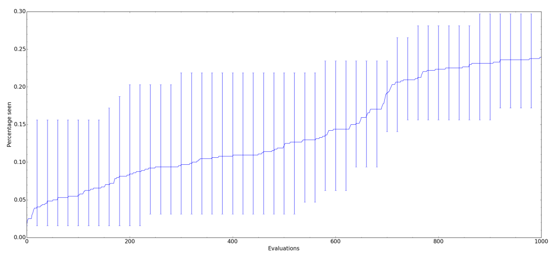


(d) Group 2 parameter setups, with noise

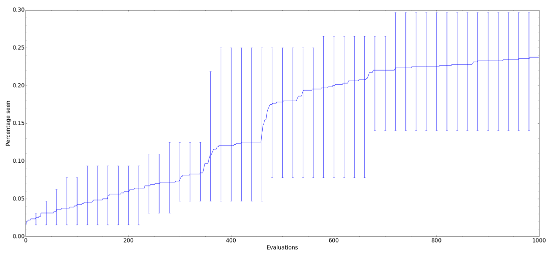
Figure 6.6: Convergence plots



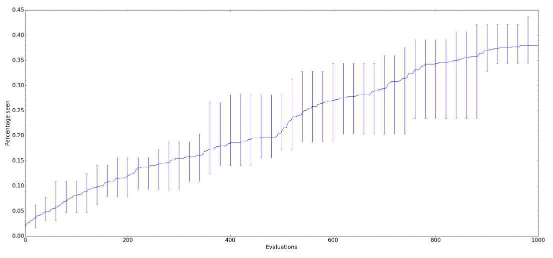
(a) Nr. of elite: 0, Mutation rate: 0.05



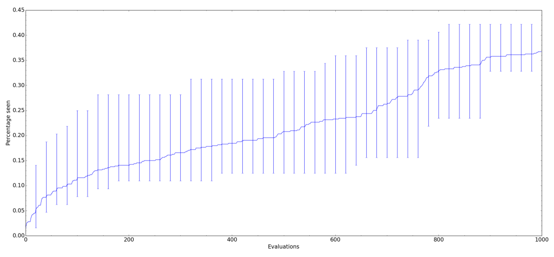
(b) Nr. of elite: 1, Mutation rate: 0.05



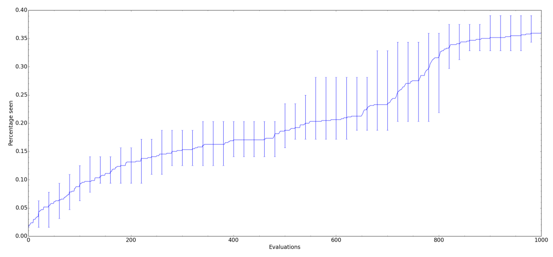
(c) Nr. of elite: 2, Mutation rate: 0.05



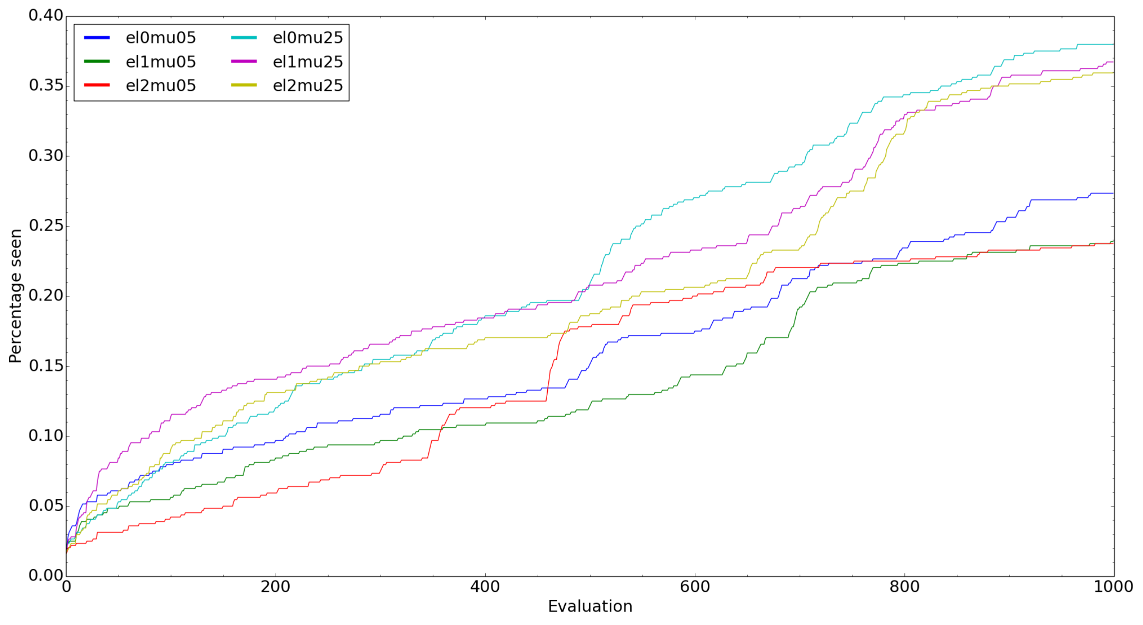
(d) Nr. of elite: 0, Mutation rate: 0.25



(e) Nr. of elite: 1, Mutation rate: 0.25

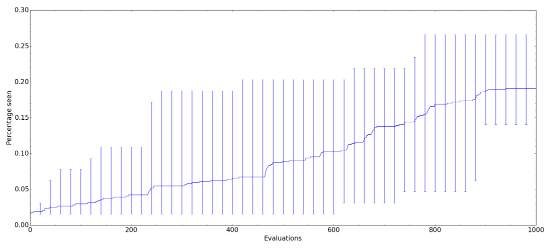


(f) Nr. of elite: 2, Mutation rate: 0.25

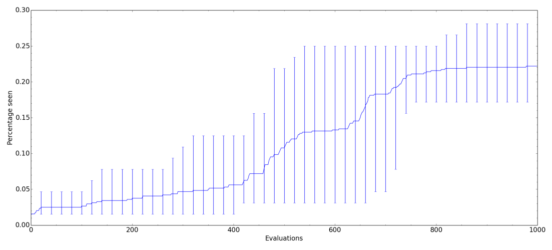


(g) overview

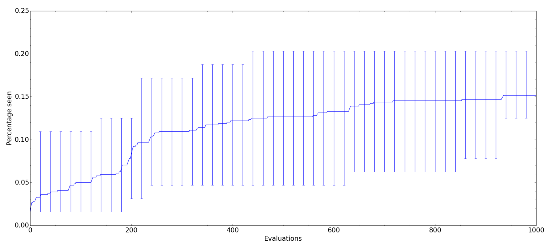
Figure 6.7: Plot with error bars of the percentage of chromosomes seen by the ***TutOER*** *system using Group 2 parameter setups in the normal environment. Plot 6.7g shows the different parameter setups in one graph.*



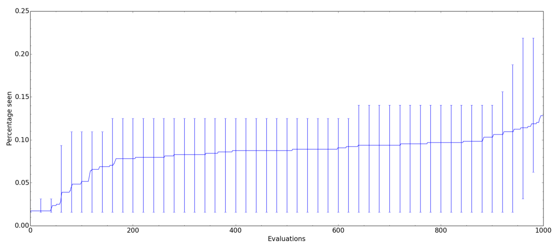
(a) Population size: 5, Nr. of episodes: 5



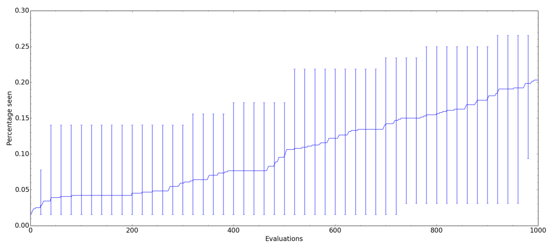
(b) Population size: 10, Nr. of episodes: 10



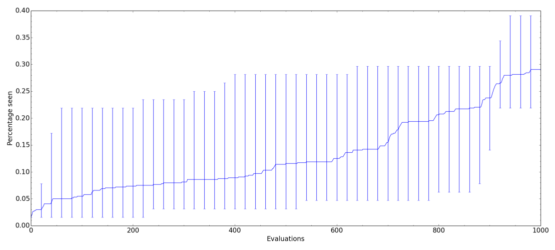
(c) Population size: 5, Nr. of episodes: 10



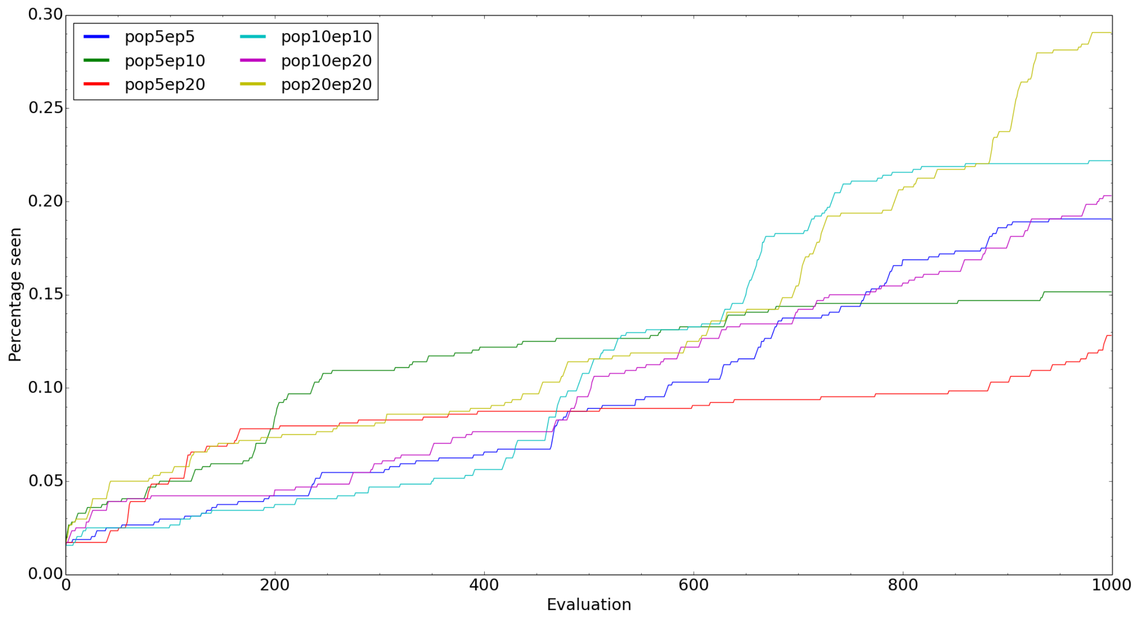
(d) `pop5ep20` parameters



(e) Population size: 10, Nr. of episodes: 20

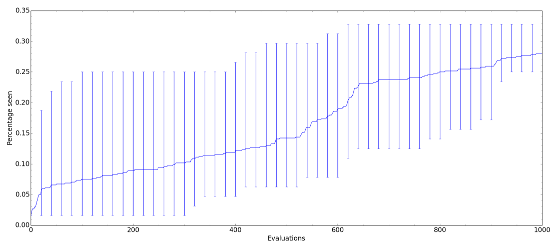


(f) Population size: 20, Nr. of episodes: 20

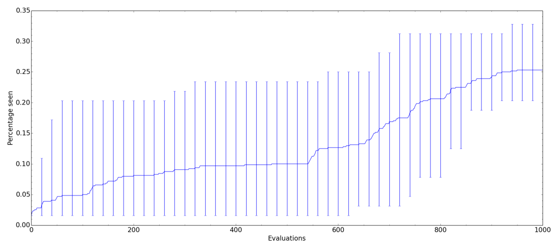


(g) overview

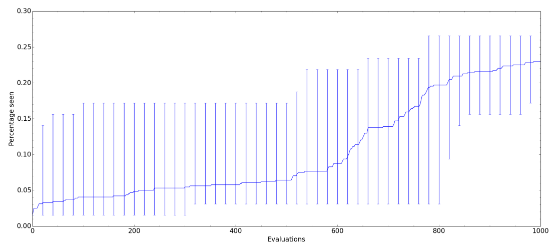
*Figure 6.8: Plot with error bars of the percentage of chromosomes seen by the **TutOER** system using Group 1 parameter setups in the noisy environment. Plot 6.8g shows the different parameter setups in one graph.*



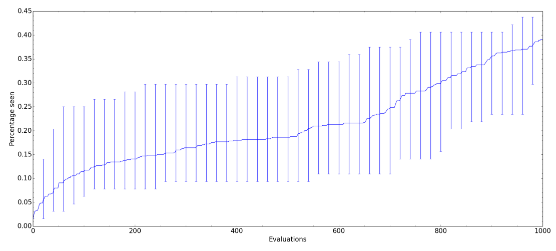
(a) Nr. of elite: 0, Mutation rate: 0.05



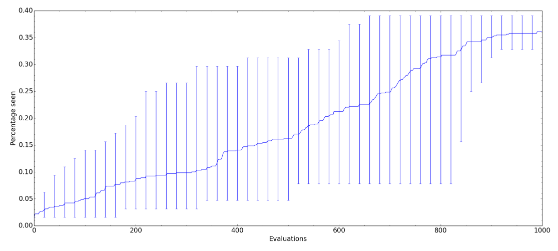
(b) Nr. of elite: 1, Mutation rate: 0.05



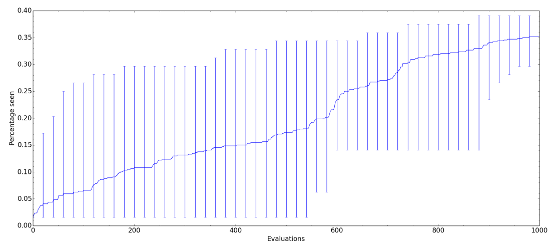
(c) Nr. of elite: 2, Mutation rate: 0.05



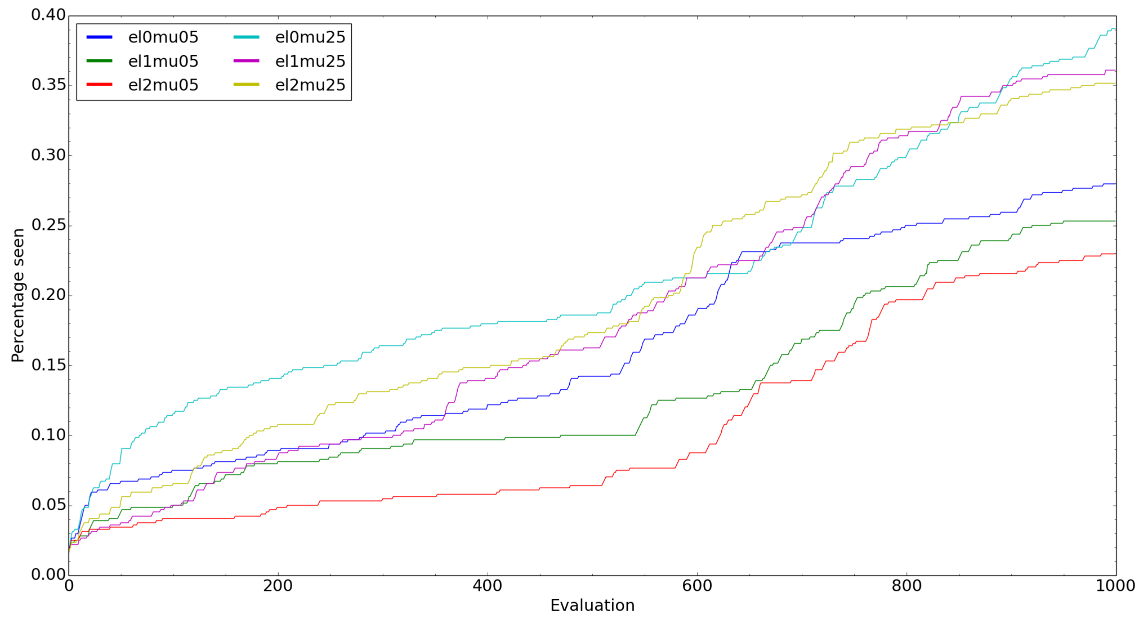
(d) Nr. of elite: 0, Mutation rate: 0.25



(e) Nr. of elite: 1, Mutation rate: 0.25



(f) Nr. of elite: 2, Mutation rate: 0.25



(g) overview

Figure 6.9: Plot with error bars of the percentage of chromosomes seen by the ***TutOER*** *system using Group 2 parameter setups in the noisy environment. Plot 6.8g shows the different parameter setups in one graph.*

7 Experimental setup

This chapter describes the experiment that has been performed to test the **TutOER** system. The purpose of the experiment was to verify whether the presented apporach would also work in a more realistic environment. The experiment took the form of an online course. In this course, sequences of OER are presented to a student in four different lessons. The sequences are selected by the **TutOER** system. Students are requested to answer a few multiple-choice questions at the beginning and the end of each sequence. These questions assess the competency level of the student on the topic of the lesson.

The educational materials that were used in the course are described in Section 7.1.1. The pre-test and post-test questions are listed in Section 7.1.2. Section 7.1.3 describes the exam that is presented at the end of the course. The experiment is ended with a questionaire that is presented in Section 7.1.4. Section 7.1.5 discusses the participants. Section 7.2.1 lists the groups to which the students will be assigned. Section 7.2.2 enumerates the parameter values chosen for the genetic algorithm during this experiment.

7.1 Experiment

7.1.1 Open Educational Resources

Participants in the experiment went through a curriculum about the game of Nim. Nim [5] is a mathematical game where two players alternate turns in taking away at least one object from exactly one of stacks of objects on the table. The experiment only looks at the normal version of the Nim game, where the person that takes the last object of the table wins. In the misère version of the game, this person would have lost. Nim is a zero-sum game and is part of a large collection of related games which have mathematically grounded strategies. Nim's winning strategy, provided you are in a winnable position, involves applying the nim-sum operator.

The Nim curriculum is divided into four knowledge components. First, the rules of the game are explained. Second, some intuition about good strategies is formed. Third, binary numbers are covered. Fourth, the binary nim-sum operation is explained. For each of these knowledge component, four educational resources were created from scratch or composed of explanations found on the internet. For the genetic algorithm to have a chance of learning something, it is important that the collection of resources contains both good and bad material. The resources for each knowledge component are described in Appendix B, the rest of this section will describe each knowledge component.

Rules of the game The first knowledge component covers the rules of the Nim game. The playing field of the game contains two or more stacks of objects. Nim is played by two players that in turn take away at least one object from exactly one stack of objects. The player that takes a way the last object from the playing field wins.

Intuition The intuition behind a winning strategy is the subject of the second knowledge component. In any Nim game, it is desirable to leave your opponent with two equally sized stacks. From that position, you can mimic any move your opponent made and return to the same situation until only one object is left for you to take away and win. The understanding that it is beneficial to leave your opponent with two equal stacks is the intuition behind the winning strategy for the Nim game.

Binary Numbers The third knowledge component covers binary numbers. In order to generalize the intuition to a winning strategy for any winneable situation a student needs to be able to work with binary representations of numbers. Specifically, a student needs to be to able to convert decimal numbers into binary numbers.

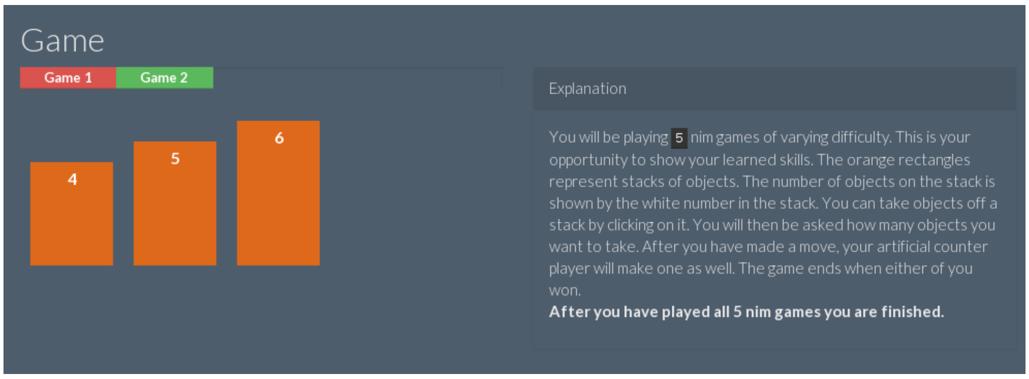
Nim-Sum The key to winning the Nim game in any winnable situation is to leave the game in a state where the nim-sum of the stacks is zero. The nim-sum operator is in fact the *exclusive or* (XOR) operator, which describes the sum of two binary numbers neglecting all carries between digits. A different way to put this is that the XOR operator outputs a 'one' when exactly one of the two inputs contains a 'one', and 'zero' in the other case. In a winnable situation, the nim-sum is not equal to zero at the beginning of your turn. You then remove a number of objects from a single stack, such that the nim-sum becomes zero. The nim-sum operator can not only be used to determine whether the nim-sum will be zero after an action, it can also be used to identify which action you can take.

A slightly easier operation to perform mentally is to write down the stacks underneath each other in binary format, count the number of 'ones' in each digit column (2^0, 2^1, 2^2 etc) and ensure that each column has an even number of 'ones'. In the resources used in this thesis this method is referred to as *pair cancelling*.

7.1.2 Pre-test & post-test

In this experiment, each knowledege component has the same questions for both the pre-test and post-test. Given the low threshold for participants to stop participating in the experiment it was necessary to keep the time needed to fill in the questions limited. This resulted in three multiple-choice questions for each knowledge component. All questions have at least the option that indicates that the user doesn't know the answer. This is still counted as a wrong answer, but is more informative for later analysis than a random guess which could be right by accident. The questions for each knowledge component are listed in Appendix C.

7.1.3 Exam



You will be playing 5 nim games of varying difficulty. This is your opportunity to show your learned skills. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. You can take objects off a stack by clicking on it. You will then be asked how many objects you want to take. After you have made a move, your artificial counter player will make one as well. The game ends when either of you won.
After you have played all 5 nim games you are finished.

Figure 7.1: Screenshot of the exam setup with five nim game scenarios

After having completed all knowledge components, a student is presented a final exam. The purpose of this exam is to have an opportunity to compare students at the beginning of the experiment with those at the end. The exam questions are nim game scenarios that the student has to play.

The game interface is described in Appendix B.1. If the student wins, the question is answered correctly. If the computer wins, the question is counted as wrong. The exam grade is calculated as the percentage of won games. The result of each game is displayed in a progress bar, as can be seen in Figure 7.1. The exam is introduced with the following explanation.

> You will be playing 5 nim games of varying difficulty. This is your opportunity to show your learned skills. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. You can take objects off a stack by clicking on it. You will then be asked how many objects you want to take. After you have made a move, your artificial counter player will make one as well. The game ends when either of you won. After you have played all 5 nim games you are finished.

The nim game scenarios are not randomly generated and are increasing in difficulty. The following list enumerates the game configurations for each question.

1. Two stacks consisting of one and two objects.
2. Three stacks with two, three and two objects.
3. Three stacks containing four, five and six objects.
4. Four stacks consisting of one, three, four and five objects.
5. Four stacks with ten, four, six and nine objects.

7.1.4 Questionaire

At the end of the course, after all knowledge components and the exam have been completed by the student. A small optional questionaire is presented to the student. The questionaire consisted of two multiple choice questions and one open ended question.

1. How good were you at Nim before these lessons?
   - I could not play it at all.
   - I could play it a little bit.
   - I could play it very well.
2. How good do you think you are after these lessons?
   - I cannot play it at all.
   - I can play it a little bit.
   - I can play it very well.
3. Do you have any last comments you whish to make?

7.1.5 Participants

7.1.5.1 Amazon Mechanical Turk

Participants for the experiment are partly found via the Amazon Mechanical Turk[1] service, where tasks that are currently not possible to execute using artificial intelligence can be done by humans in return for a small fee in the form of Amazon credits. The site forms a market place where a *requester* can post a certain *Human Intelligence Task* (HIT) that he or she wants to have crowdsourced for a certain price and where *providers* choose which tasks they want to perform. The *provider* chooses the time and place of the execution, as most if not all *HIT*s are only bound to existence of an internet connnection. Only when a *provider* completes the *HIT* to the approval of the *provider*

[1] `https://www.mturk.com/mturk/`

will he or she be paid the agreed amount. A *provider* is not forced to finish a *HIT* and can decide to give up participation at any moment during the *HIT*. Typically a preview is shown of the task before the *provider* makes the decision to accept.

In the context of this thesis, a *HIT* represents an entire walkthrough of the experiment including the different knowledge components and the final exam of one participant. *Providers* were initially paid ten cents for this task, which was raised to twenty cents during the experiment to speed up the data collection. All *providers* that completed the experiment were paid, regardless of the *provider*'s performance or usefulness of the data. *Providers* that dropped out during the experiment were not paid, but the data that was collected as a result of their actions so far was still kept.

7.1.5.2 Social network

Other participants were found via Twitter, Facebook, LinkedIn and Reddit. No further explanation was given about the game or the nature of the experiment. Participants that arrived at the experiment via these communication lines did so without a form of financial reward as *providers* did in Mechanical Turk. The call on social networks was sent out in the second half of the experiment.

7.2 Genetic algorithm setup

7.2.1 Populations

Recall that each student group is a separate population. In order to reduce the number of required data points, the students were split up in only two different student groups per knowledge component. The splitting criteria was based on the achieved pre-test score. If a student scored more than 50% then he or she was assigned to the High group of that knowledge component, else he or she was assigned to the Low group of that knowledge component. This resulted in the following groups.

Knowledge Component	Pre-test score	Group
Rules of the game	$\leq 50\%$	Rules Low
Rules of the game	$> 50\%$	Rules High
Intuition	$\leq 50\%$	Intuition Low
Intuition	$> 50\%$	Intuition High
Binary Numbers	$\leq 50\%$	Binary Low
Binary Numbers	$> 50\%$	Binary High
Nim-Sum	$\leq 50\%$	NimSum Low
Nim-Sum	$> 50\%$	NimSum High

7.2.2 Parameters

Chapter 6 analyzed what the optimal parameter values were for the genetic algorithm using evaluations based on the handcrafted model. These parameter values will be used for the experiment as well, apart from one exception. The number of individuals in the population was set back from ten (as it was in the `pop10ep10` setup) to seven. This was out of precaution to avoid the situation where a too large diversity would require more students to show signs of convergence than would be available to the experiment. This set the parameter values of the genetic algorithm in the experiment to the following.

Parameter	Value
Population size	7
Number of episodes	10
Number of elite	2
Mutation rate	0.05
Minimum length	1
Maximum length	3

7.3 Evaluation

The purpose of the experiment is to show that the approach taken in this thesis could work in a more realistic setting with actual students. To determine that, two types of success indicators are used. Furthermore several other views on the data are utilized to gain more insight in what happened. That is useful to find the reason for the success, or lack thereof. Both sets of visualisations and metrics are described in this section.

7.3.1 Indicators of success

The main question is: does the system learn to pick sequences with more learning impact over those with less impact? This question is answered using two metrics: cumulative regret and convergence.

7.3.1.1 Cumulative regret

The cumulative regret metric is similar to the one used in evaluating the simulations. There, regret was defined as the difference in received reward between the presented sequence and the optimal sequence. However, unlike in the simulation environment, in the experiment setting the optimal sequence is not known. We can only compare to the best sequence we have seen so far, which may or may not be the global optimum. Thus, the cumulative regret is defined as the cumulative difference between the estimated fitness of the presented sequence and the highest known estimated fitness, within the same population.

The estimates used in the regret calculation are the estimated values at the end of the experiment. The reason is that comparing with the estimated values at the moment of the decision is less interesting. Theoretical results from the literature about genetic algorithms and UCB already tell us how they will deal with exploration versus exploitation and the regret that would be caused by that. Furthermore from the simulations performed in this thesis it appears that the algorithm is properly implemented. What the experiment is intended to find out is whether the task of OER quality assessment by curriculum sequencing is feasible in a realistic scenario with noise due to real students. For that purpose it is more interesting to see whether the **TutOER** system would temporarily discard optimal sequences that have seem to perform poorly at the time, or whether the system is robust enough.

If the approach works, the regret will decrease over time. As a result the growth of the cumulative regret will, approximately, stop. The **TutOER** system will always continue to try out sequences that appear to be suboptimal, but the intervals between these explorations will increase. Small peaks in regret are thus expected, but the trend of the line should be flat. The **cumulative regret curve** shows the development of the cumulative regret after each student.

Furthermore, there are quantitative metrics that can be extracted from the cumulative regret curve. The final cumulative regret value is not particularly interesting, since it doesn't capture the trend of the cumulative regret curve. However, the cumulative regret of the **first 20%** and **last 20%** of evaluations do capture the trend, albeit in a low resolution. If the approach works, the cumulative regret of the last 20% of evaluations must be lower than that of the first 20%. Ideally the last 20% of evaluations have no further regret at all, which would result in a cumulative regret of zero for those evaluations. However, due to the incidental explorations it is possible that a small peak occurs in the last 20% evaluations. In that case, the cumulative regret curve will give a definitive answer about the trend.

7.3.1.2 Convergence point

In general we want any learning algorithm to converge. The notion of convergence however needs some adjustment, because of the mentioned incidental exploration. The used definition of the point of convergence is the evaluation from which twenty-five consecutive evaluations targeted the same sequence. This metric does not indicate whether the system converged on presenting the optimal sequence, only that it converged on some sequence.

7.3.2 Insight in behavior

7.3.2.1 Coverage curve

The coverage is defined as the percentage of sequences that was evaluated at least once. This metric is identical to the one used for the simulation analysis. The total number of possible sequences is 40. That means each new encountered sequence adds 2.5% to the coverage. There are limits to what percentages can theoretically be achieved in each generation. Each population is initialized with four different sequences in its first generation. Furthermore, there are seven individuals in each consecutive generation. That means that the maximum coverage is given by $4 + 7 \cdot (i - 1)$, for each ith generation.

More coverage is not necessarily a good thing. The genetic algorithm should steer the search towards the more promising areas of the search space. However, the coverage can offer some indication as to how reliable the statements about optimality are. When a larger area of the search space has been encountered, it is more likely that the found optimum is a global one.

7.3.2.2 Evaluation noise

The observed learning gain of each sequence is expected to vary per student. The amount of noise in this learning gain has an impact on the performance of the system. The evaluation value for the best sequence of each population is plotted (in blue) for each student. Alongside, the running average is shown (in green), which is the estimate of the sequence's fitness after each evaluation.

7.3.2.3 Sorted number of evaluations

The genetic algorithm, together with UCB-1 selection, introduces a bias towards evaluating certain sequences over others. That is also what they are intended to do. However, a very skewed distribution of evaluations indicates that sequences did not receive the same opportunity to prove optimal. Particularly when the evaluation noise is large.

7.3.2.4 Diversity table

The diversity table shows the percentage of unique sequences at each generation for each population. A diversity of 43% indicates that across the seven individuals three distinct chromosomes were divided. Similarly, a diversity of 14% means that all individuals have the same chromosome. The table does not capture how many occurences each chromosome has. This is also not relevant. The UCB-1 algorithm selects which sequence to evaluate regardless of their number of occurences in the generation.

8

Results

The experiment described in Section 7 offers an online course consisting of four lessons. A participant's competence is assessed at the beginning and end of the lesson. The pre-test result determines the student group to which the participant is assigned. The **TutOER** system selects the sequence to be presented to the participant within that student group. The approach balances exploration and exploitation to maximize the expected learning gain while minimizing online regret. The purpose of the experiment is to answer the central question: does the system learn to pick sequences with more learning impact over those with less impact? There are two metrics that answer this question. Firstly, we expect to see the slope of the cumulative regret to decrease. This is captured in the difference between the cumulative regret during the first and last 20% of evaluations. Secondly, we expect the **TutOER** system to converge, as captured in the convergence point. In this chapter the result of the experiment are enumerated and discussed.

Table 8.1 shows an overview of the important metric values for each population. In the populations *Rules low*, *Intuition low*, *Binary low* and *Nim-sum low*, the system reduced and stabilized the growth of the regret. In those populations, the system converged to the best sequence encountered. The *Rules high*, *Binary high* and *Nim-sum high* populations did not receive enough participants to conclude anything. Table 8.2 shows the number of evaluations received for each population. In population *Intuition high*, the system was not successful in stabilizing the regret. This is likely due to a bug, which is discussed in more detail in Section 8.2. The same has occured in population *Binary high.*

The participants that made it to the end of the experiment also completed the final exam, where they played five consecutive Nim games. Table 8.3 shows the frequency of exam scores for the first, second, third and last 25% of those participants. The average score is slightly higher in the last 25% compared to the first. However, there is not a clear trend upwards. The experiment has thus not convincingly shown a clear improvement in participants' understanding of how to play the Nim game. In other words, the sequences presented in each lesson did not improve over time enough to cause a clear effect on the displayed Nim competence.

The answer to the central question is, yes in some populations it did. It is not known whether the sequences that appeared to be the best are actually the global optimum. The estimated fitness values of some of the best sequences are quite low, especially in the case of the *Nim-sum low* population. It is not unlikely that the approach got stuck in a local optimum in this case. Although it is also possible that none of the sequences would have been effective. The estimated fitness of the best sequence in the *Rules low* population is however much more convincing. The populations in which the system did not seem to work had either too little data or were affected by a bug. However, it is of course not certain that the system would've been successful otherwise.

To better understand what the effects were of the **TutOER** system, the results per lesson are described in more detail in the rest of this chapter.

Table 8.1: Overview of evaluation metric scores for each population. The convergence column is the evaluation from when at least 25 students got the same sequence presented. The best column shows the estimated fitness of the best seen sequence. The cumulative regret scores show the cumulative regret for both the first and the last 20% of the evaluations. The coverage indicates the percentage of sequences evaluated at least once.

Population	Convergence	Best	First 20%	Last 20%	Coverage
Rules low	14	0.7634	12.115	0.000	$^{8}/_{40}$
Rules high	No	0.4388	2.286	0.514	$^{5}/_{40}$
Intuition low	20	0.4576	7.043	0.000	$^{7}/_{40}$
Intuition high	8	0.5959	16.540	4.996	$^{6}/_{40}$
Binary low	65	0.5251	3.523	0.000	$^{11}/_{40}$
Binary high	No	0.3265	1.347	1.684	$^{4}/_{40}$
Nim-sum low	20	0.2557	2.717	0.000	$^{8}/_{40}$
Nim-sum high	No	1.000	7.980	6.263	$^{10}/_{40}$

Table 8.2: The table shows five student statistics for each lesson. First, the number of students that completed the low *category. Second, the number of bootstrap values that were used in the* low *category. Third, the number of students that completed the* high *category. Fourth, the number of bootstrap values that were used in the* high *category. Last, the number of students that skipped the lesson.*

	Category *low*		Category *high*		
Lesson	Completed	Bootstrapped	Completed	Bootstrapped	Skipped
Rules	204	19	19	10	14
Intuition	85	23	71	30	81
Binary	106	26	16	11	117
Nim-sum	154	12	13	9	6

Table 8.3: Frequency of exam scores for the first, second, third and last 25% of the participants. Exam scores are defined as the number of Nim games won.

	Frequency of number of games won						
Participants	0 wins	1 win	2 wins	3 wins	4 wins	5 wins	Average win
First 25%	9	19	9	1	2	0	1.2 games
Second 25%	7	13	9	5	2	5	1.9 games
Third 25%	9	17	9	2	2	2	1.4 games
Last 25%	6	13	13	0	5	4	1.9 games

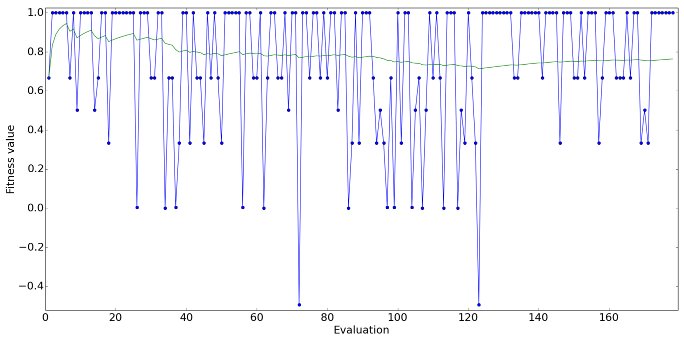
8.1 Lesson: Rules

The *Rules low* population completed 22 generations and received 223 evaluations by students. At the end of the experiment this population converged to the best sequence encountered. The sequence contains Resource 3 and Resource 1, in that order, and will be referred to as *Sequence [3,1]* for the rest of the chapter. In total 103 out of 178 students that got *Sequence [3,1]* answered all post-test questions correctly. A graph of all evaluations of this sequence can be seen in Figure 8.1a. The estimated fitness of *Sequence [3,1]* at the end of the experiment was 0.7634. *Sequence [3,1]* was created by a mutation on Resource 3 in the transition from the fourth to the fifth generation that added Resource 1. At first, Resource 1 dissapeared from the population after not being selected for the second generation. Resource 3 had the highest fitness of the first four generations

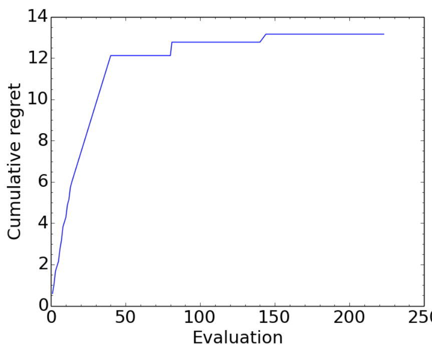
Figure 8.1b shows that after 40 evaluations, in the fifth generation, **TutOER** switches to the evaluation of *Sequence [3,1]*. This is only deviated from at two points. First, at the 81st evaluation, the **TutOER** system tried out Resource 1 once more. Second, from evaluation 141 till 144 the **TutOER** system evaluated the mirrored version of *Sequence [3,1]* that ended with Resource 3. In both cases however, the **TutOER** system returned to the optimal sequence.

Figure 8.5a shows the *Rules low* population encountered a fifth of all sequences, namely eight out of 40. Although this was enough to find *Sequence [3,1]*, the large majority of sequences was not attempted. No sequences of three resources were attempted. These sequences filled 60% of the total number of possible sequences in this experiment. When comparing to the possible sequences of two resources maximum, the **TutOER** system evaluated half.

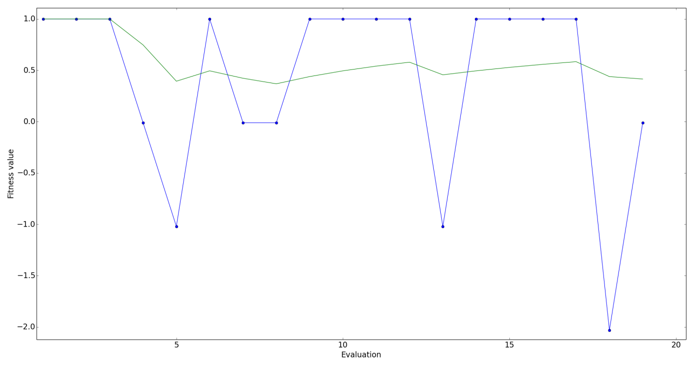
The *Rules high* population encountered five out of 40 possible sequences. However, in contrast to *Rules low*, only 30 evaluations were collected. As a result, the population completed just two generations. The best sequence at the end of the experiment contained only Resource 1. The estimated fitness of this sequence is 0.4388. The number of evaluations is however too low to put much value on the estimate. Figure 8.1d shows the cumulative regret built-up in *Rules high*. The best sequence at the end of the experiment is already present in the first generation and remains present in each consecutive generation. In the third generation, *Sequence [3,1]* enters the population by means of immigration from the *Rules low* population. The new sequence is tried four times in a row and two times later on, but the measured fitness was lower than the current best sequence.



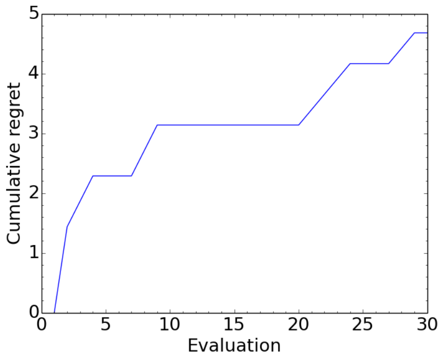
(a) Sequence [3,1] in the Rules low *population*



(b) Rules low



(c) Sequence [1] in the Rules high *population*



(d) Rules high

Figure 8.1: The left side contains observed normalized learning gain values (blue) plotted with the running average (green) of the best sequences for the Rules *lesson. The right side contains the cumulative regret curves for the* Rules *lesson.*

8.2 Lesson: Intuition

Figure 8.2b shows the cumulative regret of the *Intuition low* population. During the experiment 108 evaluations were received from students. The best sequence at the end of the generation is *Sequence [6]*. The sequence was already present in the first generation and remained present throughout all eleven generations. The estimated fitness of *Sequence [6]* is 0.4576. The range of observed fitness values for *Sequence [6]* is plotted in Figure 8.2a.

In the second generation of the *Intuition low* population, the **TutOER** system tried out the sequence of Resource 5 and Resource 6 several times in both orderings. Both however scored lower than the best sequence in that generation. As a result, in the third generation almost all individuals contained *Sequence [6]*. This is also visualized in Table 8.4 that shows the diversity of the *Intuition low* population dropped to 29%, which means there are two types of sequences in the generation. From the second generation onwards, the **TutOER** system sticks to presenting *Sequence 6*. The only exception is the 61st evaluation where *Sequence [6,8]*, coming from the *Intuition high* population, is tried once. At the end of the experiment seven out of the 40 sequences were encountered, which is also shown in Figure 8.5c.

The best sequence in the *Intuition high* population at the end of the experiment is *Sequence [6,8]*. The estimated fitness value of this sequence is 0.5959. However, the **TutOER** system did not manage to hold on to this sequence. As a result, it was only evaluated ten times. Figure 8.2c displays the observed fitness values. After the first generation, three consecutive generations were dominated by *Sequence [6]*. All other sequences that were present in the first generation had a negative estimated fitness. In the roulette wheel selection these sequences have a zero chance of being selected. Furthermore, the elite member slots were filled with the two occurences of *Sequence [6]*. Table 8.4 shows the drop in diversity as a result. Figure 8.5d shows that at the end of the experiment six out of the 40 sequences were encountered.

In the fifth generation a mutation on *Sequence [6]* resulted in *Sequence [6,8]*. The **TutOER** system lost the sequence in the transition to the seventh generation. This is likely due to a bug in the software related to sequences that migrated. As a consequence the aggregated fitness value of those sequences was altered by both populations in which it was present. This aggregated value was used by both the UCB algorithm and roulette wheel selection. As a result, *Sequence [6]* appeared to have a higher fitness than *Sequence [6,8]* when this was not the case. The bug was discovered after the experiment was completed. The fitness estimates used in this report are not affected by this bug.

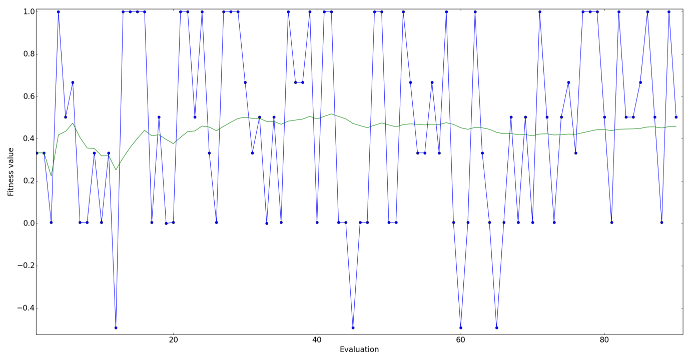
8.3 Lesson: Binary

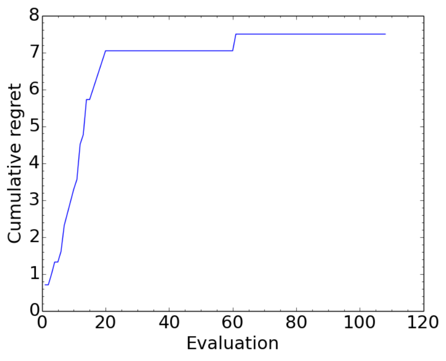
The *Binary low* population received 132 evaluations by students. *Sequence [9]* turned out to be the best with an estimated fitness of 0.5251. Figure 8.3a shows the observed fitness values for each of the 103 students that evaluated *Sequence [9]*. Figure 8.6e shows that these 103 evaluations make *Sequence [9]* the most evaluated sequence in this population by far. Yet eleven out of 40 possible sequences have been encountered, as can be seen in Figure 8.5e. This is the highest coverage of the eight populations.

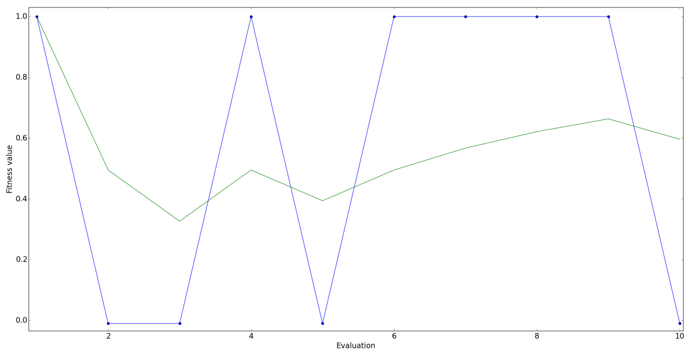
Figure 8.3b shows the cumulative regret in the *Binary low* population. The best sequence, *Sequence [9]*, is already present in the first generation. From the 40th evaluation, the **TutOER** system sticks to *Sequence [9]*. In the seventh generation, a few evaluations were assigned to *Sequence [9,11]*. However, with an estimated fitness of 0.4676 this sequence did not outperform *Sequence [9]*.

Contrary to its counterpart, population *Binary high* has the lowest coverage of all populations. Figure 8.5f shows that only 4 out of 40 sequences have been encountered. That is the coverage that every population starts with. No crossover operations were applied in the transition to the second population. The order of the sampling of survivors resulted in two pairs of equal chromosomes and one different chromosome at the end. The parent selection selects pairs from this sampled list of survivors from top to bottom to be potential parents. The first two pairs of parents were identical and therefore unfit for crossover. The last survivor could not be paired with anything. As a result, the second generation of the *Binary high* population was filled with survivors and not with their offspring.

The third generation contained only *Sequence [12]* apart from the two elite members that carried *Sequence [9]*. When only the same sequence is sampled, no crossover can be applied. The reasoning


(a) Sequence [6] in the Intuition low *population*


(b) Intuition low


(c) Sequence [6,8] in the Intuition high *population*

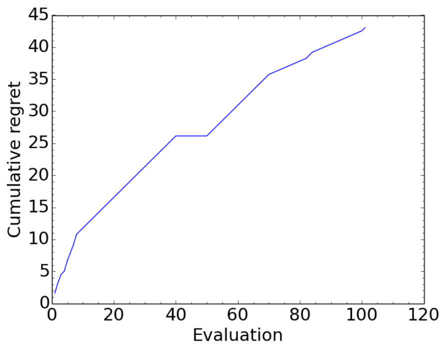

(d) Intuition high

Figure 8.2: The left side contains observed normalized learning gain values (blue) plotted with the running average (green) of the best sequences for the Intuition *lesson. The right side contains the cumulative regret curves for the* Intuition *lesson.*

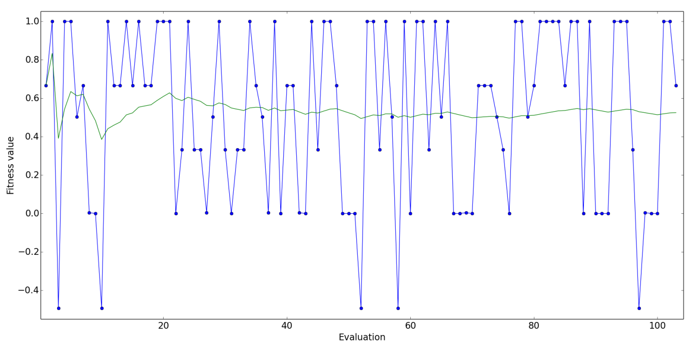
is similar to that in the previous paragraph. Table 8.4 shows the reduction in diversity in three generations.

After 27 evaluations, the best sequence in *Binary high* is *Sequence [10]* with an estimated fitness of 0.3265. However, Figure 8.3d shows the **TutOER** system does not hold on to the best sequence. The bug relating to immigrated chromosomes that was discussed in Section 8.2 is likely the cause of that. *Sequence [9]* that immigrated from *Binary low* appeared to be the best performing sequence. Even though *Sequence [10]* was in fact the only sequence with a non-negative fitness estimate. However, with so few evaluations it is hard to say whether *Sequence [10]* was actually good.

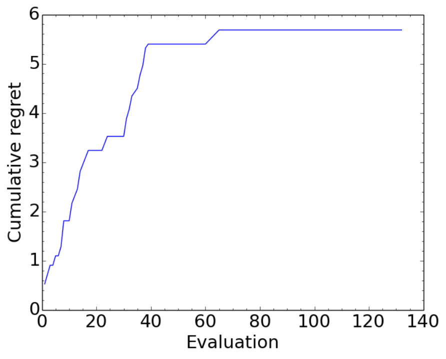
8.4 Lesson: Nim-sum

The *Nim-sum low* population gathered 166 evaluations by students at the end of the experiment. From the 15th evaluation onwards, the **TutOER** system mainly stuck to evaluating *Sequence [16]*. This is also reflected in the cumulative regret curves shown in Figure 8.4b. With an estimated fitness of 0.2557, *Sequence [16]* turned out to be the best sequence of the population. Figure 8.6g shows the difference between the number of evaluations of each sequence. The difference between the first and second most evaluated sequence is 150 evaluations. This difference is the largest of all other populations, but *Rules low* comes close with a difference of 148 evaluations. Also similar to in *Rules low*, eight out of 40 possible sequences were encountered. However, the diversity in each generation is rather low. From the fourth to the eighth generation, only one sequence was present. Table 8.4 shows the diversity per generation.

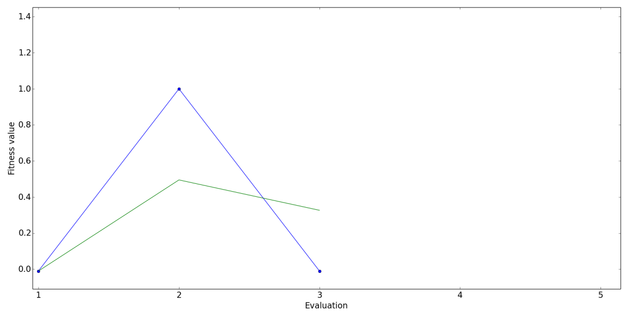
The least matured population is *Nim-sum high* with only 22 evaluations. Despite the limited number of evaluations, ten out of 40 sequences were encountered (Figure 8.5h). Table 8.4 shows the



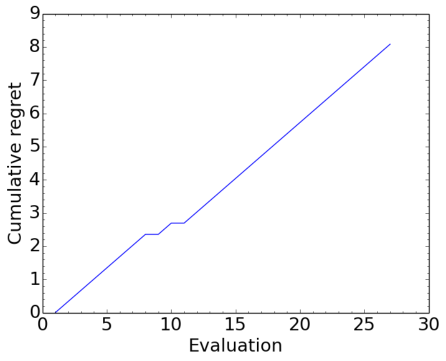
(a) Sequence [9] in the Binary low *population*



(b) Binary low



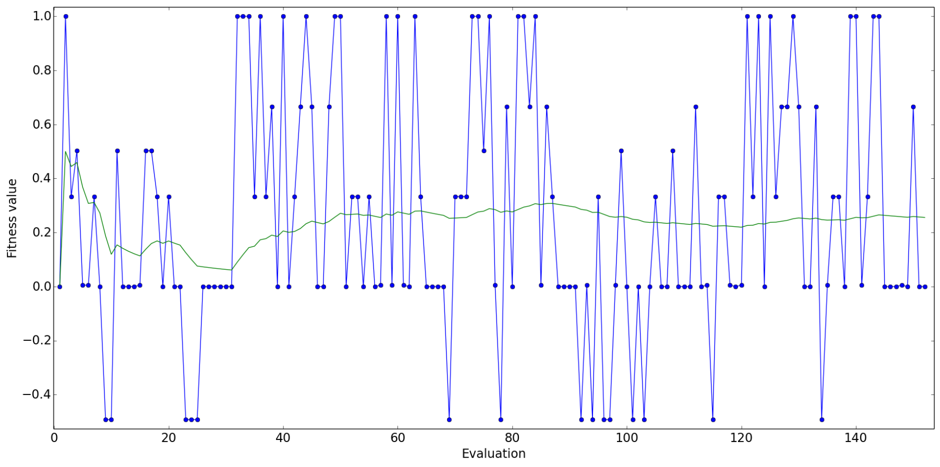
(c) Sequence [10] in the Binary high *population*



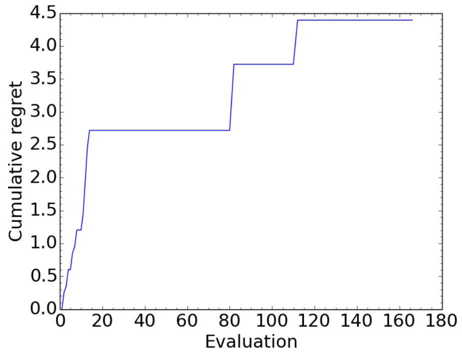
(d) Binary high

Figure 8.3: The left side contains observed normalized learning gain values (blue) plotted with the running average (green) of the best sequences for the Binary *lesson. The right side contains the cumulative regret curves for the* Binary *lesson.*

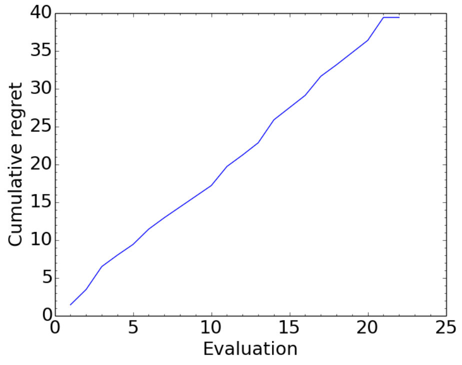
high diversity that is the cause of this. The best sequence is *Sequence [15,16]* with an estimated fitness of 1.0. However, this sequence has only been evaluated once. The variance in the fitness of all other learning materials shows that one evaluation provides very little information.



(a) Sequence [16] in the Nim-sum low *population*

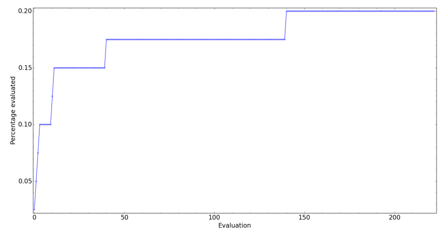


(b) Nim-sum low

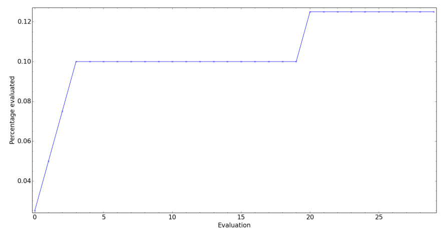


(c) Nim-sum high

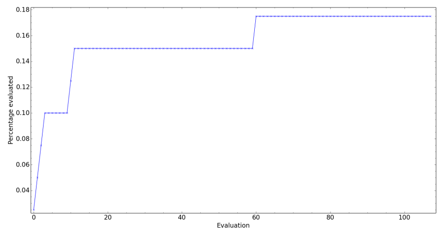
Figure 8.4: The left side contains observed normalized learning gain values (blue) plotted with the running average (green) of the best sequences for the Nim-sum *lesson. The graph for the* Nim-sum high *population is however not shown, since the best sequence only had one evaluation. The right side contains the cumulative regret curves for the* Nim-sum *lesson.*



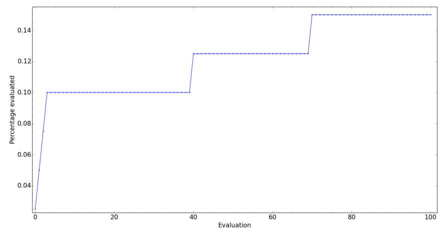
(a) Rules low



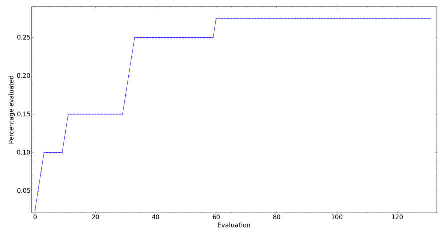
(b) Rules high



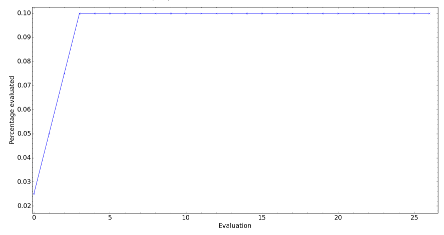
(c) Intuition low



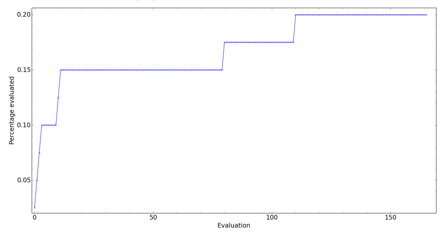
(d) Intuition high



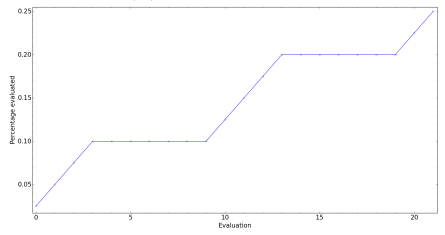
(e) Binary low



(f) Binary high

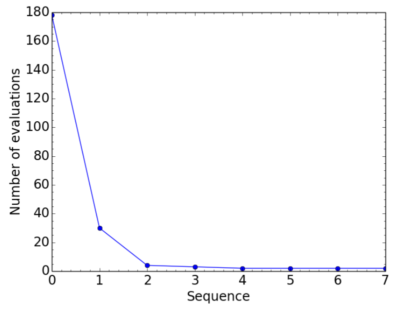


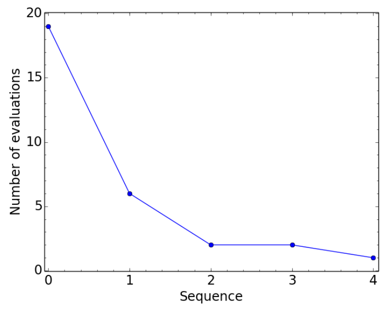
(g) Nim-sum low

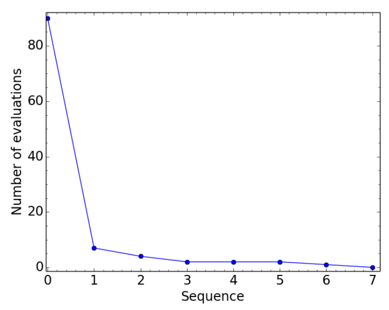


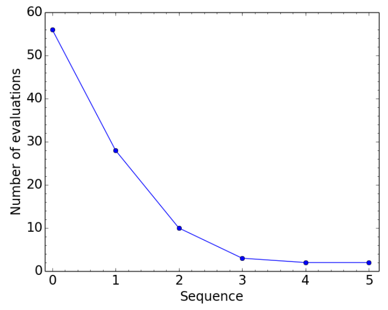
(h) Nim-sum high

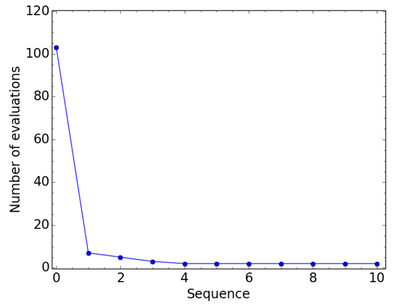
Figure 8.5: Percentage of sequences evaluated after each evaluation in each population.

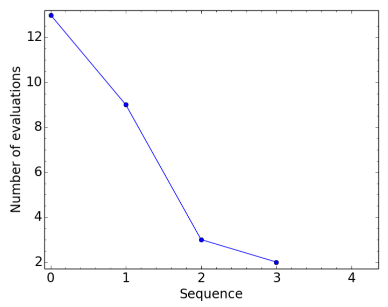

(a) Rules low

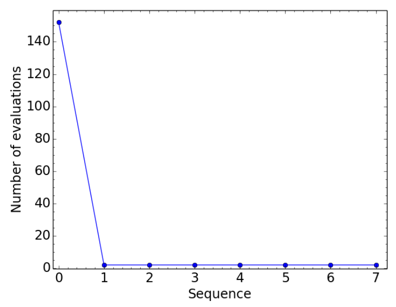

(b) Rules high


(c) Intuition low


(d) Intuition high


(e) Binary low


(f) Binary high


(g) Nim-sum low

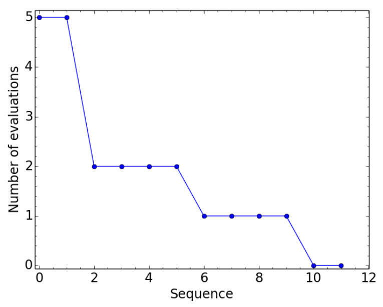

(h) Nim-sum high

Figure 8.6: Sorted number of evaluations of each sequence in each population.

Table 8.4: *Diversity percentage at each generation*

(a) generation 1-12

	Generation											
Population	1	2	3	4	5	6	7	8	9	10	11	12
Rules Low	57%	57%	43%	57%	43%	29%	14%	14%	29%	14%	29%	29%
Rules High	57%	29%	29%	—	—	—	—	—	—	—	—	—
Intuition Low	57%	43%	29%	29%	29%	14%	29%	14%	14%	14%	29%	—
Intuition High	57%	14%	14%	14%	29%	29%	14%	29%	29%	14%	29%	—
Binary Low	57%	57%	43%	86%	29%	14%	43%	43%	29%	29%	43%	14%
Binary High	57%	43%	29%	—	—	—	—	—	—	—	—	—
NimSum Low	57%	57%	29%	14%	14%	14%	14%	14%	29%	14%	14%	29%
NimSum High	57%	86%	86%	—	—	—	—	—	—	—	—	—

(b) generation 13-24

	Generation											
Population	13	14	15	16	17	18	19	20	21	22	23	24
Rules Low	43%	29%	29%	29%	29%	29%	29%	14%	14%	29%	43%	—
Rules High	—	—	—	—	—	—	—	—	—	—	—	—
Intuition Low	—	—	—	—	—	—	—	—	—	—	—	—
Intuition High	—	—	—	—	—	—	—	—	—	—	—	—
Binary Low	29%	14%	—	—	—	—	—	—	—	—	—	—
Binary High	—	—	—	—	—	—	—	—	—	—	—	—
NimSum Low	14%	14%	14%	14%	14%	—	—	—	—	—	—	—
NimSum High	—	—	—	—	—	—	—	—	—	—	—	—

9

Conclusion / Discussion

In this thesis a novel approach to automatic assessment of OER quality has been presented and evaluated. Measurements of the impact of OER on learning as component of the OER quality was largely missing in the literature. The **TutOER** system evaluates OER in a sequence by assessing students' competence levels before and after the presentation of the OER sequence . The confidence of the estimate had to be balanced with the cost of presenting bad educational content to students. The curriculum sequencing task from the intelligent tutoring systems community has been used as framework to do this. The UCB-1 selection mechanism was applied to determine which OER te evaluate. A genetic algorithm was used to limit the scope of UCB-1 and to have a more directed and structured search. The system was evaluated in an online experiment.

The experiment shows that fitness evaluations can vary widely for one sequence. There are three possible causes for that. Firstly, the assessments presented in the experiment had only three multiple-choice questions. The results from these assessments are thus more susceptible for noise than a more extensive assessment. In particular, potential guessing of students has a high impact on the outcome. Secondly, the division of students in student groups is rather coarse. It is highly likely that students within one student group form everything but a homogeneous group. Not only did students who had zero questions or one question correct in the pre-test end up in the same group. Students also potentally differed on learning style, age or prior knowledge on related topics. Not to mention that students who participated presumingly did so in different environments, levels of concentration, committment and time to spare. These factors could all have an impact on the measured learning gain, especially in the more complex lessons at the end. Lastly, the system assumes that the performance on the current sequence is independent of the sequences presented in other lessons. This assumption is of course a simplification of reality, since whether a student understands binary numbers will affect the chance of learning the nim-sum. Part of the noise in the fitness evaluations could be explained by the fact that students in the same group had different learning experiences in previous lessons.

Despite the varying fitness observations, the **TutOER** was capable of finding the *best*[1] sequence in at least one student group for each lesson in the experiment. The *best* sequences had on average a higher fitness than the other sequences. In some cases the *best* sequence appears promising enough to be close to a global optimum. For example, the sequence found in the *Rules low* population had a high fitness of 0.7634. Furthermore, 103 out of 178 students got all post-test question correct after seeing this sequence. Many more students at least improved their score.

In some other cases the *best* sequence is less convincing. For example, the *best* sequence in the *Nim-sum low* population had a low fitness of 0.2557. More than half of the students had an equal or even lower post-test score compared to their pre-test. In these cases it is not a stretch to imagine that a better sequence might have existed. Even though on average this sequence was better than the rest on decisive moments. Due to the chosen parameter values of the genetic

[1]The best relative to what **TutOER** has seen.

algorithm, decisions are made on relatively few evaluations. Furthermore, the order in which the students participate is approximately random. Given that most if not all OER sequences will perform bad with at least a few students. It could happen that by chance a sequence would get those few students at the beginning. A different sequence might get those few students at the end, and will perform better on average at the beginning in comparison. Due to the directed search of the genetic algorithm, it might take some time before a sequence is granted enough opportunities to correct the quality estimate.

The parameters of the genetic algorithm were set to stimulate quick convergence. The underlying argument was that it was unclear how many participants would be available to support the **TutOER** system in showing the desired effect. As a result, the populations very quickly only contained very few unique sequences. This severly limited the exploration opportunities. In part, the parameters setting the population size and number of episodes were responsible for this. However, an additional factor was the implementation of parent selection. The roulette wheel requires the fitness values to be normalized between 0 and 1. Normalization was done using a straightforward approach of dividing each fitness value by the maximum fitness value in the selection. Negative fitness values were set to zero and thus had no chance of being selected. It seemed like a sensible idea at the time to only select from sequences that actually caused some improvement. However, based on the observed variance of the fitness values, this might have been responsible for the elimination of good candidates.

The problem of premature elimination is of course also an issue for a normal teacher evaluating educational material. In fact, this is true for every one-armed bandid situation where there is a trade-off between exploration and exploitation in a noisy environment. The solution appears straightforward, but it comes with a cost. The solution is to evaluate sequences more often before making a selection. The parameter that influences that directly is the number of episodes. That parameter was set to a low value for the experiment, which also becomes clear from the results. Raising it would however also inevitably raise the regret built-up from sequences that turn out to be bad. A more indirect parameter is the number of individuals in the population. A larger population would increase the chance of any individual to be selected. In particular if that would be combined with a different approach to parent selection that deals with negative fitness differently. More diversity in the population would slow down the convergence of the genetic algorithm. However, due to UCB-1 selection, the exploration versus exploitation trade-off would be better managed.

From the data collected in this thesis, it is not possible to know if the **TutOER** system did indeed find the optimal sequences. In other words, it is unknown how bad the fitness estimates are. The author proposes an additional benchmarking experiment to retrieve that information. A simple approach would be to evaluate all sequences a certain number of times. Based on the large variance in the observed fitness values, each sequence would need to be evaluated at least fifty times. That would however require 2000 participants for both the low and high student group. Lowering the variance could perhaps be done with a more elaborate set of questions for the benchmark, which would at least be less sensitive to random mistakes of participants. Unfortunately, the main source of the variance is likely to be the heterogeneous collection of participants in one student group. Ultimately, students should be differentiated into more different groups as to create more homogeneous collections of participants. However, that would require to redo the experiment described in this thesis with the new student groups.

Apart from comparison with an extensive benchmark, it is also interesting to compare this approach with one that only uses UCB. The genetic algorithm now serves as a “smart” filter for UCB with the purpose of speeding up the search. The rationale behind this is that unlike UCB, the genetic algorithm generalizes implicitly by selection and cross-over. In particular because a sequence can be represented by chromosome in a natural manner, which would allow for meaningful direction of the search. As a result some areas of the search space do not need to be explored. This is of course only beneficial if the genetic algorithm is effective. It therefore makes sense to run the same experiment with a setup where only UCB selection is used to pick sequences. The regret will in the beginning almost certainly be much higher, but it may prove to be more effective afterwards.

In conclusion, this thesis presented and tested a possible approach to the incorperation of learning impact in assessment of OER quality. Although many lessons can be learned, the results of the simulation and the experiment show that the principle works, in spite of a limited and mostly diverse collection of participants. The author recommends the field concerning OER quality to search for improvements of and alternatives for this approach. It is essential for the feasibillity of truly open collections of OER to take into account the impact an OER has on learning. After all, that is why we want to have open educational resources in the first place.

Bibliography

[1] Ad Aerts, David Smits, Natalia Stash, and Paul De Bra. Aha! version 2.0, more adaptation flexibility for authors. In *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, volume 1, pages 240–246, 2002.

[2] Sarab Al-Muhaideb and Mohamed El Bachir Menai. Evolutionary computation approaches to the curriculum sequencing problem. *Natural Computing*, 10(2):891–920, 2011.

[3] Daniel Ewell Atkins, John Seely Brown, and Allen L Hammond. *A review of the open educational resources (OER) movement: Achievements, challenges, and new opportunities*. Creative common, 2007.

[4] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[5] Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):pp. 35–39, 1901.

[6] Peter Brusilovsky and Nicola Henze. Open Corpus Adaptive Educational Hypermedia. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 671–696. Springer Berlin Heidelberg, 2007.

[7] Peter Brusilovsky and Christoph Peylo. Adaptive and intelligent web-based educational systems. *International Journal of Artificial Intelligence in Education*, 13(2):159–172, 2003.

[8] Peter Brusilovsky and Julita Vassileva. Course sequencing techniques for large-scale web-based education. *International Journal of Continuing Engineering Education and Life Long Learning*, 13(1):75–94, 2003.

[9] Peter Brusilovsky, Elmar Schwarz, and Gerhard Weber. ELM-ART: An intelligent tutoring system on World Wide Web. In *Intelligent tutoring systems*, pages 261–269. Springer, 1996.

[10] Peter L Brusilovsky. A framework for intelligent knowledge sequencing and task sequencing. In *Intelligent tutoring systems*, pages 499–506. Springer, 1992.

[11] Jet Bussemaker. Kamerbrief over digitalisering van het hoger onderwijs. Kamerbrief, Ministerie van Onderwijs, Cultuur en Wetenschap, 1 2014. URL `http://www.rijksoverheid.nl/documenten-en-publicaties/kamerstukken/2014/01/08/kamerbrief-over-digitalisering-van-het-hoger-onderwijs.html`.

[12] A. F. Camilleri, U. D. Ehlers, and J. Pawlowski. State of the Art Review of Quality Issues related to Open Educational Resources (OER). JRC Scientific and policy reports, European Commission, 2014. URL `http://is.jrc.ec.europa.eu/pages/EAP/documents/201405JRC88304.pdf`.

[13] Cristian Cechinel, Salvador Sánchez-Alonso, and Miguel Ángel Sicilia. Empirical analysis of errors on human-generated learning objects metadata. In *Metadata and semantic research*, pages 60–70. Springer, 2009.

[14] Cristian Cechinel, Salvador Sánchez-Alonso, and Elena García-Barriocanal. Statistical profiles of highly-rated learning objects. *Computers & Education*, 57(1):1255–1269, 2011.

[15] Cristian Cechinel, Sandro da Silva Camargo, Xavier Ochoa, MA Sicilia, and S Sanchez-Alonso. Populating learning object repositories with hidden internal quality information. In *Proceedings of the 2nd Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2012). Manouselis, N., Draschler, H., Verber, K., and Santos, OC (Eds.). Published by CEUR Workshop Proceedings*, volume 896, pages 11–22, 2012.

[16] Chih-Ming Chen. Intelligent web-based learning system with personalized learning path guidance. *Computers & Education*, 51(2):787–814, 2008.

[17] Chih-Ming Chen. Ontology-based concept map for planning a personalised learning path. *British Journal of Educational Technology*, 40(6):1028–1058, 2009.

[18] Min Chi, Kurt VanLehn, and Diane Litman. Do micro-level tutorial decisions matter: Applying reinforcement learning to induce pedagogical tutorial tactics. In *Intelligent Tutoring Systems*, pages 224–234. Springer, 2010.

[19] Doug Clow. An overview of learning analytics. *Teaching in Higher Education*, 18(6):683–695, 2013. doi: 10.1080/13562517.2013.827653.

[20] Betty Collis and Allard Strijker. Technology and human issues in reusing learning objects. *Journal of interactive media in education*, 2004(1), 2004.

[21] Luis de Marcos, JJ Martinez, JA Gutierrez, et al. Competency-based learning object sequencing using particle swarms. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, volume 2, pages 111–116. IEEE, 2007.

[22] Luis De-Marcos, José-Javier Martínez, José Antonio Gutierrez, Roberto Barchino, and José María Gutiérrez. A new sequencing method in web-based education. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 3219–3225. IEEE, 2009.

[23] Erik Duval. Learnrank: Towards a real quality measure for learning. In *Handbook on quality and standardisation in E-learning*, pages 457–463. Springer, 2006.

[24] Erik Duval and David Wiley. Guest Editorial: Open Educational Resources. *IEEE Transactions on Learning Technologies*, 3(2):83–84, 2010. ISSN 1939-1382.

[25] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag, Berlin, 2007.

[26] John Hilton III, David Wiley, Jared Stein, and Aaron Johnson. The four 'R's of openness and ALMS analysis: frameworks for open educational resources. *Open Learning*, 25(1):37–44, 2010.

[27] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Bradford Books. MIT Press, 1992. ISBN 9780262581110.

[28] Anna Hovakimyan, Siranush Sargsyan, and Sergey Barkhoudaryan. Genetic algorithm and the problem of getting knowledge in e-learning systems. In *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*, pages 336–339. IEEE, 2004.

[29] Mu-Jung Huang, Hwa-Shan Huang, and Mu-Yen Chen. Constructing a personalized e-learning system based on genetic algorithm and case-based reasoning approach. *Expert Systems with Applications*, 33(3):551–564, 2007.

[30] Jan Hylén, Dirk van Damme, Fred Mulder, and Susan D'Antoni. Open Educational Resources: Analysis of Responses to the OECD Country Questionnaire. Technical report, OECD, 06 2012.

[31] Larry Johnson, Samantha Adams, Malcolm Cummins, Victoria Estrada, Alex Freeman, and Holly Ludgate. The NMC horizon report: 2013 higher education edition. Technical report, New Media Consortium, 2013.

[32] Robin H Kay and Liesel Knaack. Evaluating the learning in learning objects. *Open Learning*, 22(1):5–28, 2007.

[33] Kenneth R Koedinger, Emma Brunskill, Ryan SI d Baker, Elizabeth A McLaughlin, and John Stamper. New potentials for data-driven intelligent tutoring system development and optimization. *AI Magazine*, 34(3), 2013.

[34] Milos Kravcik, Marcus Specht, and Reinhard Oppermann. Evaluation of winds authoring environment. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 166–175. Springer, 2004.

[35] Michael EN Majerus. Industrial melanism in the peppered moth, biston betularia: an excellent teaching example of darwinian evolution in action. *Evolution: Education and Outreach*, 2(1): 63–74, 2009.

[36] Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel, and Rob Koper. Recommender systems in technology enhanced learning. In *Recommender systems handbook*, pages 387–415. Springer, 2011.

[37] Julie A Marsh, John F Pane, and Laura S Hamilton. Making sense of data-driven decision making in education. Technical report, Rand Corporation, 2006.

[38] Worthy N Martin, Jens Lienig, and James P Cohoon. Island (migration) models: evolutionary algorithms based on punctuated equilibria. *Handbook of evolutionary computation*, 6(3), 1997.

[39] Rory McGreal. Learning objects: A practical definition. *International Journal of Instructional Technology and Distance Learning (IJITDL)*, 9(1), 2004.

[40] Mariusz Nowostawski and Riccardo Poli. Parallel genetic algorithm taxonomy. In *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, pages 88–92. IEEE, 1999.

[41] Xavier Ochoa and Erik Duval. Use of contextualized attention metadata for ranking and recommending learning objects. In *Proceedings of the 1st international workshop on Contextualized attention metadata: collecting, managing and exploiting of rich usage information*, pages 9–16. ACM, 2006.

[42] Xavier Ochoa and Erik Duval. Quantitative analysis of learning object repositories. *Learning Technologies, IEEE Transactions on*, 2(3):226–238, 2009.

[43] Xavier Ochoa and Erik Duval. Automatic evaluation of metadata quality in digital repositories. *International Journal on Digital Libraries*, 10(2-3):67–91, 2009.

[44] OECD. Giving Knowledge for Free: The Emergence of Open Educational Resources. Technical report, OECD, 2007. URL `http://www.oecd.org/edu/ceri/givingknowledgeforfreetheemergenceofopeneducationalresources.htm`.

[45] Clark Quinn and Samantha Hobbs. Learning objects and instruction components. *Educational Technology & Society*, 3(2):13–20, 2000.

[46] Azough Samia and Bellafkih Mostafa. Re-use of resources for adapted formation to the learner. In *Computational Intelligence and Intelligent Informatics, 2007. ISCIII'07. International Symposium on*, pages 213–217. IEEE, 2007.

[47] Kazuya Seki, Tatsunori Matsui, and Toshio Okamoto. An adaptive sequencing method of the learning objects for the e-learning environment. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 88(3):54–71, 2005.

[48] Miguel A Sicilia, Elena Garcia, Carmen Pagés, and Jose J Martinez. Complete metadata records in learning object repositories: some evidence and requirements. *International Journal of Learning Technology*, 1(4):411–424, 2005.

[49] James P Spillane. Data in practice: Conceptualizing the data-based decision-making phenomena. *American Journal of Education*, 118(2):113–141, 2012.

[50] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[51] Alice Tani, Leonardo Candela, and Donatella Castelli. Dealing with metadata quality: The legacy of digital library efforts. *Information Processing & Management*, 49(6):1194–1205, 2013.

[52] UNESCO. Forum on the impact of Open Courseware for higher education in developing countries. Final report, UNESCO, 2002. URL `http://unesdoc.unesco.org/images/0012/001285/128515e.pdf`.

[53] Kurt Vanlehn. The behavior of tutoring systems. *International journal of artificial intelligence in education*, 16(3):227–265, 2006.

[54] Katrien Verbert, Xavier Ochoa, Michael Derntl, Martin Wolpers, Abelardo Pardo, and Erik Duval. Semi-automatic assembly of learning resources. *Computers & Education*, 59(4): 1257–1272, 2012.

[55] Jeffrey C Wayman. Involving teachers in data-driven decision making: Using computer data systems to support teacher inquiry and reflection. *Journal of Education for Students Placed at Risk*, 10(3):295–308, 2005.

[56] Martin Weller. Big and little OER. *2010 Proceedings. Barcelona: UOC, OU, BYU. [Accessed: 25/05/14].*, 2010. URL `http://hdl.handle.net/10609/4851`.

[57] Darrell Whitley, Soraya Rana, and Robert B Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–48, 1999.

[58] David Wiley, T.J. Bliss, and Mary McEwen. Open Educational Resources: A Review of the Literature. In J. Michael Spector, M. David Merrill, Jan Elen, and M. J. Bishop, editors, *Handbook of Research on Educational Communications and Technology*, pages 781–789. Springer New York, 2014. ISBN 978-1-4614-3184-8. doi: 10.1007/978-1-4614-3185-5_63. URL `http://dx.doi.org/10.1007/978-1-4614-3185-5_63`.

[59] Nicolai van der Woert, Ria Jacobi, and Hester Jelgerhuis, editors. *2014 Open Education Trend Report*, 3 2014. URL `http://www.surf.nl/trendreportopeneducation2014`.

[60] Li Yuan and Stephen Powell. MOOCs and open education: Implications for higher education. *Cetis White Paper*, 2013.

[61] Robert Zemsky and William F Massy. Thwarted innovation. *What happened to e-learning and why, A final report for the Weather station Project of the Learning Alliance at the University of Pennsylvania in cooperation with the Thomson Corporation, Pennsylvania*, 2004.

A

Database

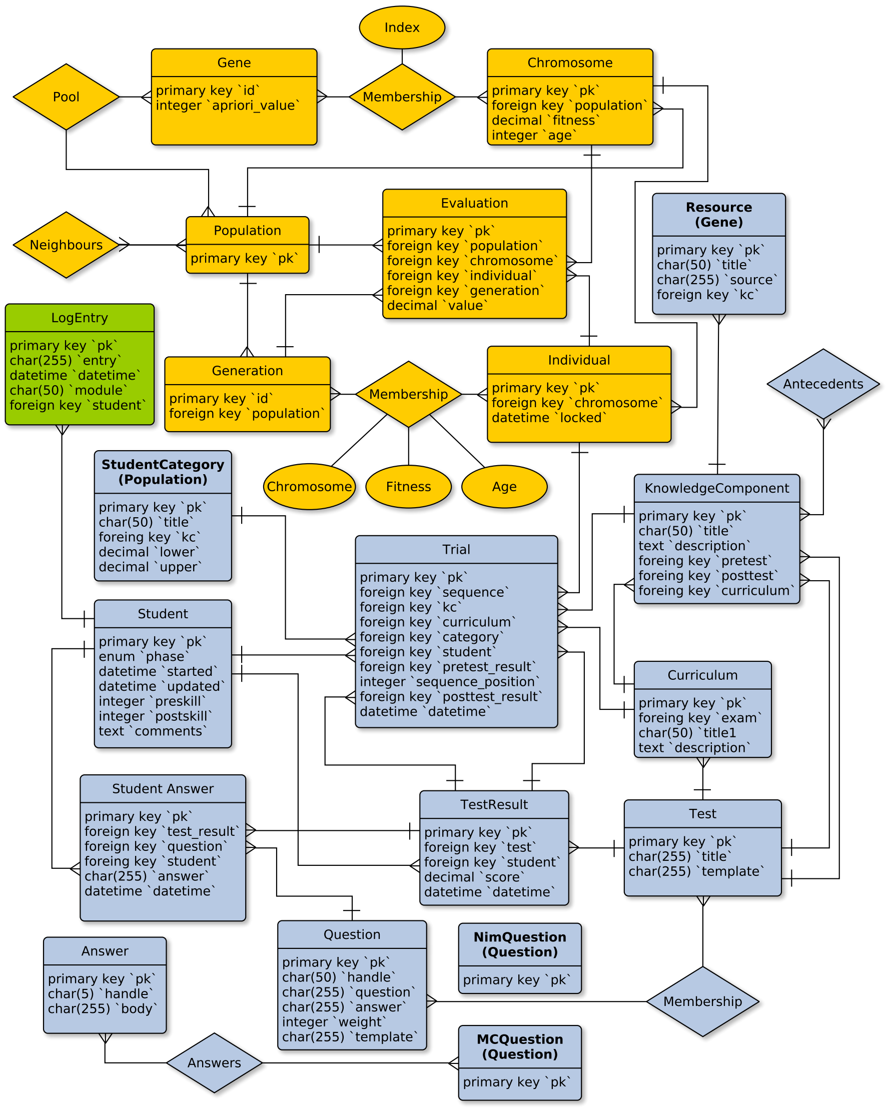


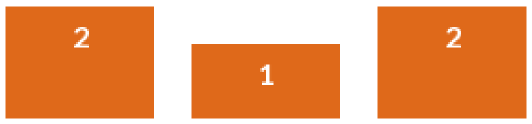
Figure A.1: Entity relationship schema of the database

B

Nim Course Material

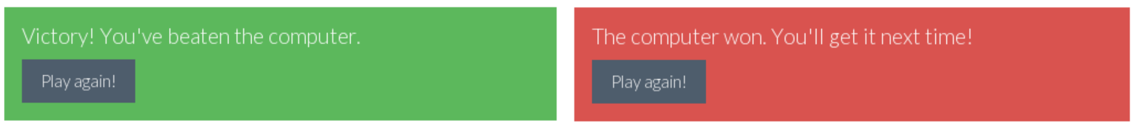
B.1 Interactive Nim Exercises

Resources can contain a script that allows the student to play the nim game. The interface depicts each stack as an orange box of which the height is linked to the number of objects on that stack. The student can click on one of these orange boxes to be prompted for the number of objects to remove. Only valid input is allowed in this prompt, meaning that a student is also not capable of taking nothing or even taking more objects than exist on the stack. The visualisation for the nim game with stacks of two, one and two objects is as follows.



After the student made a move, a computer player automatically makes a counter move. This continues until one of the two players won. The computer player will always attempt to make the optimal move using the nim-sum operator. If there are multiple moves possible given the winning strategy, the computer player picks the first. When faced with an unwinnable situation, the computer selects a stack at random and either removes all objects or all but one, depending on the parity of the number of stacks and the number of objects left on the stack. The strategy in an unwinnable situation is not part of the standard nim-sum strategy, but was added to force a student to play a perfect game throughout.

When the nim game is over, the students receives feedback about the outcome of the game and is presented with an opportunity to play another randomly generated game. The feedback is shown as follows.


Victory! You've beaten the computer.

The computer won. You'll get it next time!

B.2 Rules of the game

B.2.1 Resource 1

This resource is shown in Figure B.1 contains the following text: *Nim is a game in which two players take turns removing objects from distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap. The normal game is between two players and played with three heaps of any number of objects. The*

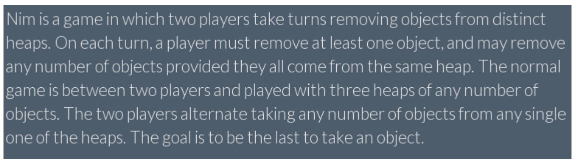

Nim is a game in which two players take turns removing objects from distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap. The normal game is between two players and played with three heaps of any number of objects. The two players alternate taking any number of objects from any single one of the heaps. The goal is to be the last to take an object.

Figure B.1: ***Resource 1*** *covers the rules of the game*

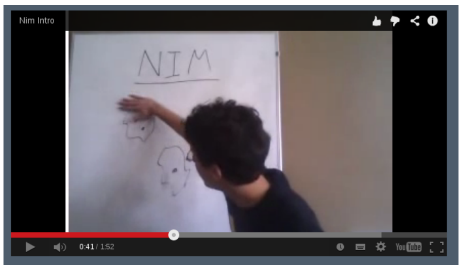


Figure B.2: ***Resource 2*** *covers the rules of the game*

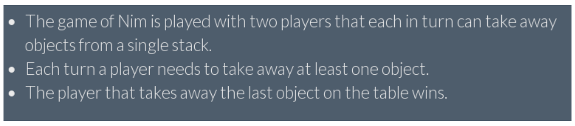

- The game of Nim is played with two players that each in turn can take away objects from a single stack.
- Each turn a player needs to take away at least one object.
- The player that takes away the last object on the table wins.

Figure B.3: ***Resource 3*** *covers the rules of the game*

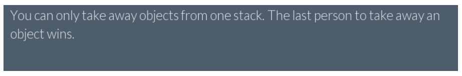

You can only take away objects from one stack. The last person to take away an object wins.

Figure B.4: ***Resource 4*** *covers the rules of the game*

two players alternate taking any number of objects from any single one of the heaps. The goal is to be the last to take an object.

B.2.2 Resource 2

The resource shown in Figure B.2 plays the Youtube video `zEpXIoF2nwk` made by Paul Gafni, which explains the rules by showing an example game. It is a video clip from a larger collection of nim lessons created by the same author. The clip is displayed in an embedded Youtube player.

B.2.3 Resource 3

The third resource that covers the rules of the game is shown in Figure B.3 and contains a textual explanation in three points. One, *The game of Nim is played with two players that each in turn can take away objects from a single stack.* Two, *Each turn a player needs to take away at least one object.* Three, *The player that takes away the last object on the table wins.*

B.2.4 Resource 4

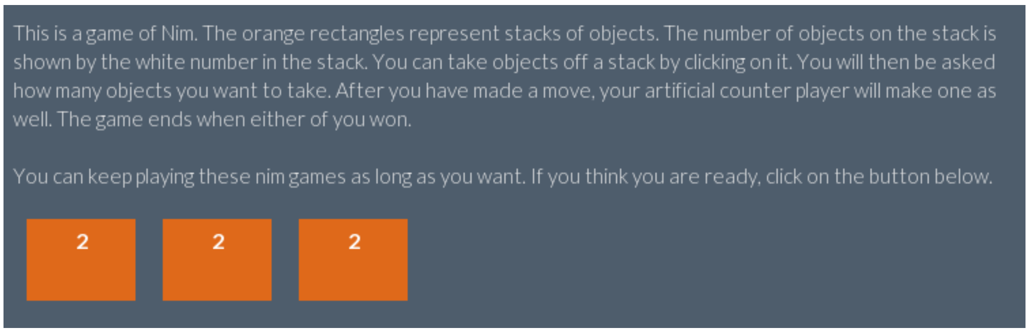
The last resource that covers the rules of the game is shown in Figure B.4 and contains a very brief textual explanation. *You can only take away objects from one stack. The last person to take away an object wins.*

B.3 Intuition

B.3.1 Resource 5

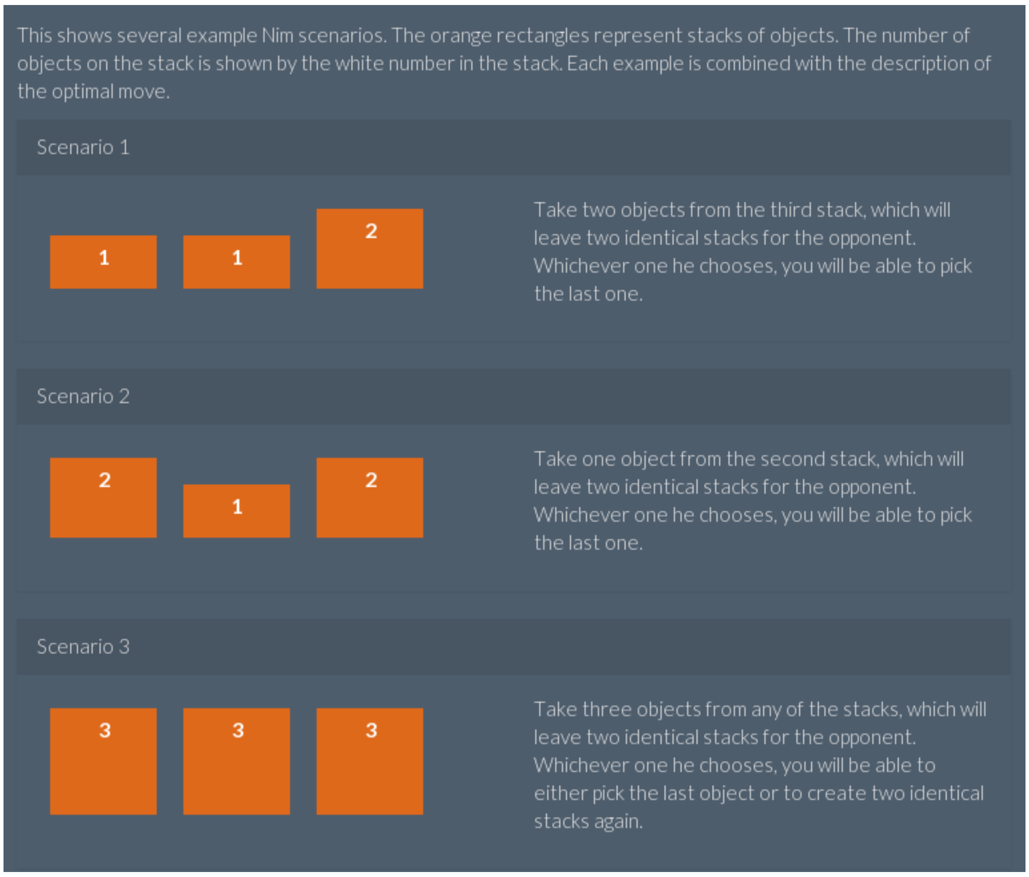
Figure B.5 shows the resource that allows the student to play the nim game using the script defined in SectionB.1. The configuration of stacks should be simple enough for most students to be able to imagine the consequences of an action. The winning sequence of moves is typically three moves long for the student. The configuration is generated randomly in such a way that each stack contains between one and three objects, with two stacks having the identical number of objects. The latter requirements ensures that the game is winnable by the student and that a more advanced nim-sum strategy is not yet necessary.

The nim game interface is also explained in the resource with an introduction.


This is a game of Nim. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. You can take objects off a stack by clicking on it. You will then be asked how many objects you want to take. After you have made a move, your artificial counter player will make one as well. The game ends when either of you won.

You can keep playing these nim games as long as you want. If you think you are ready, click on the button below.

Figure B.5: ***Resource 5*** *covers the intuition behind a winning strategy for the game nim*


This shows several example Nim scenarios. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. Each example is combined with the description of the optimal move.

Scenario 1

Take two objects from the third stack, which will leave two identical stacks for the opponent. Whichever one he chooses, you will be able to pick the last one.

Scenario 2

Take one object from the second stack, which will leave two identical stacks for the opponent. Whichever one he chooses, you will be able to pick the last one.

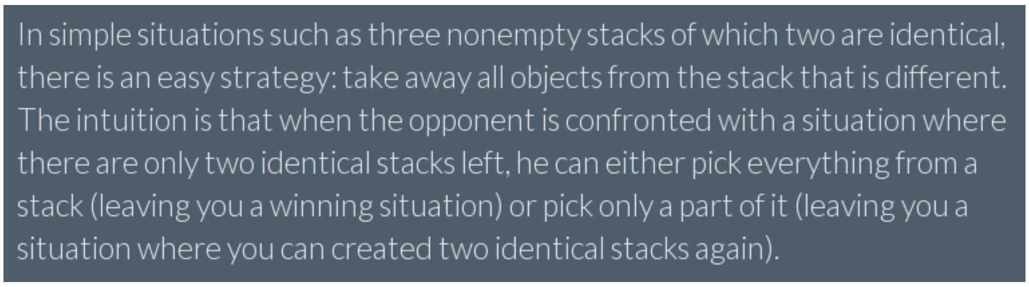
Scenario 3

Take three objects from any of the stacks, which will leave two identical stacks for the opponent. Whichever one he chooses, you will be able to either pick the last object or to create two identical stacks again.

Figure B.6: ***Resource 6*** *covers the intuition behind a winning strategy for the game nim*

This is a game of Nim. The orange rectangles represent stacks of objects. The number of objects on the stack is shown by the white number in the stack. You can take objects off a stack by clicking on it. You will then be asked how many objects you want to take. After you have made a move, your artificial counter player will make one as well. The game ends when either of you won.

You can keep playing these nim games as long as you want. If you think you are ready, click on the button below.



In simple situations such as three nonempty stacks of which two are identical, there is an easy strategy: take away all objects from the stack that is different. The intuition is that when the opponent is confronted with a situation where there are only two identical stacks left, he can either pick everything from a stack (leaving you a winning situation) or pick only a part of it (leaving you a situation where you can created two identical stacks again).

Figure B.7: ***Resource 7*** *covers the intuition behind a winning strategy for the game nim*

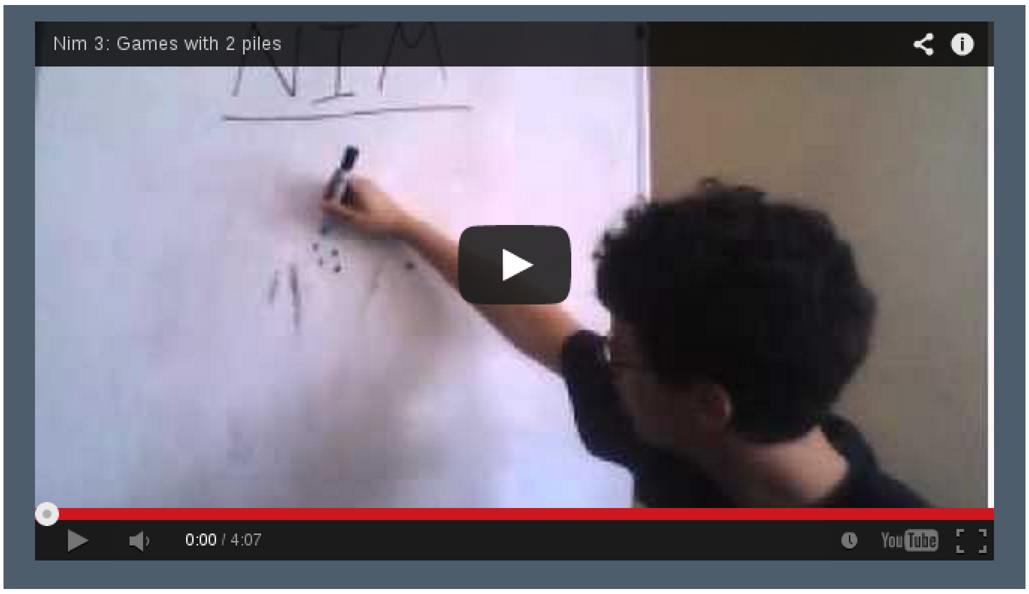


Figure B.8: ***Resource 8*** *covers the intuition behind a winning strategy for the game nim*

B.3.2 Resource 6

A resource shown in Figure B.6 that describes the optimal actions for three example scenarios. The described actions are all aimed at achieving a situation with two identical stacks on the table for the other player. Regardless of what the other player takes away, you can mimic the action for the other stack and end up taking the last object. The first scenario depicts two equal stacks and one bigger stack. The second scenario depicts two equal stacks and one smaller stack. The last scenario depicts three equal stacks.

B.3.3 Resource 7

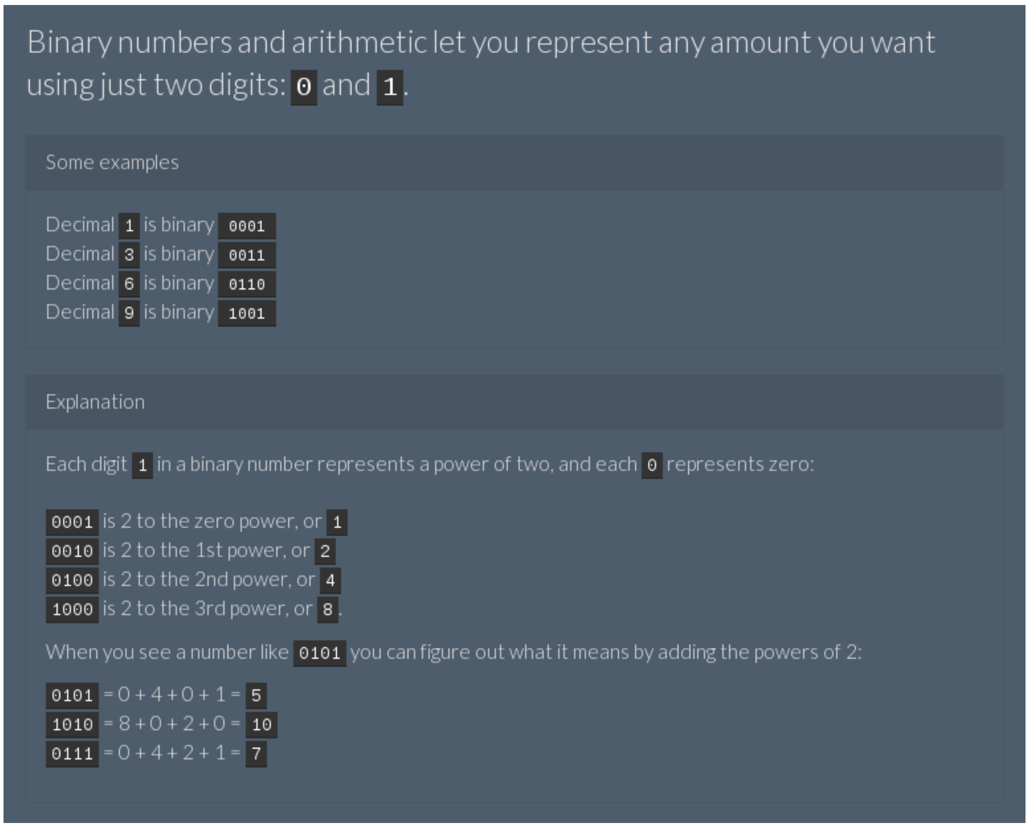
Figure B.7 shows a resource that gives a textual explanation of the intuitive strategy of enforcing two equal stacks on the opponent. This is formulated as follows.

In simple situations such as three nonempty stacks of which two are identical, there is an easy strategy: take away all objects from the stack that is different. The intuition is that when the opponent is confronted with a situation where there are only two identical stacks left, he can either pick everything from a stack (leaving you a winning situation) or pick only a part of it (leaving you a situation where you can created two identical stacks again).

B.3.4 Resource 8

The resource shown in Figure B.8 has an embedded video player which shows a video explanation made by Paul Gafni[1]. The video demonstrates in a walkthrough of the game the kind of reasoning a player must use to decide what move to make.

B.4 Binary numbers



Binary numbers and arithmetic let you represent any amount you want using just two digits: `0` and `1`.

Some examples

Decimal `1` is binary `0001`
Decimal `3` is binary `0011`
Decimal `6` is binary `0110`
Decimal `9` is binary `1001`

Explanation

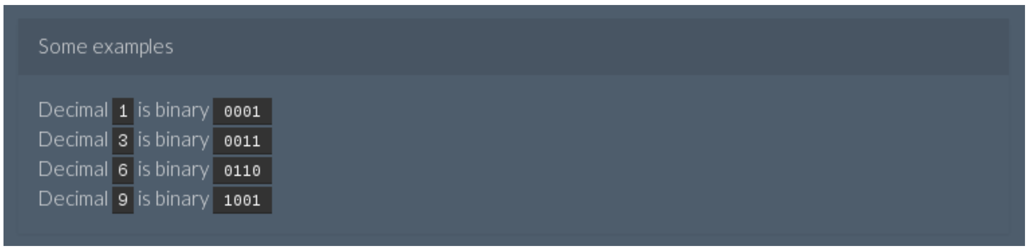
Each digit `1` in a binary number represents a power of two, and each `0` represents zero:

`0001` is 2 to the zero power, or `1`
`0010` is 2 to the 1st power, or `2`
`0100` is 2 to the 2nd power, or `4`
`1000` is 2 to the 3rd power, or `8`.

When you see a number like `0101` you can figure out what it means by adding the powers of 2:

`0101` = 0 + 4 + 0 + 1 = `5`
`1010` = 8 + 0 + 2 + 0 = `10`
`0111` = 0 + 4 + 2 + 1 = `7`

Figure B.9: ***Resource 9*** *covers the conversion of binary numbers to their decimal form.*



Some examples

Decimal `1` is binary `0001`
Decimal `3` is binary `0011`
Decimal `6` is binary `0110`
Decimal `9` is binary `1001`

Figure B.10: ***Resource 10*** *covers the conversion of binary numbers to their decimal form.*

B.4.1 Resource 9

Figure B.9 shows a resource that lists four example conversions between decimal and binary numbers, describes the meaning of having a `1` in a particular position of a binary number and walks through the conversion of three binary numbers to their decimal counterparts. Throughout these

[1]Source: `https://www.youtube.com/channel/UC3EadMDqZmJVJ2f43QSzIRQ`

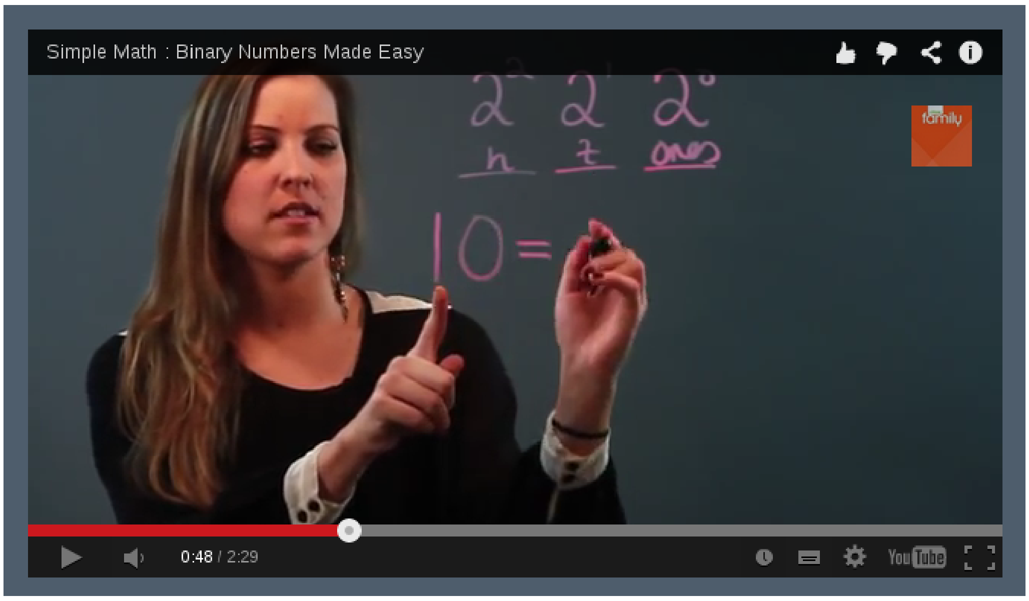


Figure B.11: ***Resource 11*** *covers the conversion of binary numbers to their decimal form.*

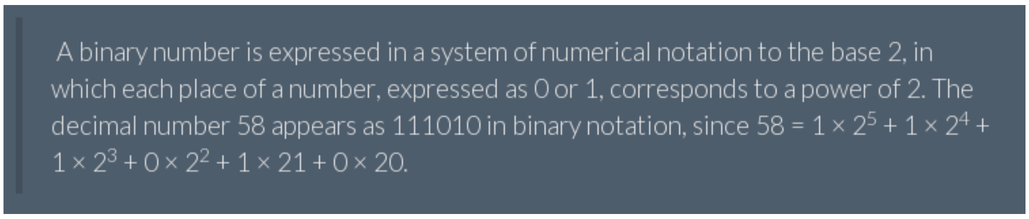

A binary number is expressed in a system of numerical notation to the base 2, in which each place of a number, expressed as 0 or 1, corresponds to a power of 2. The decimal number 58 appears as 111010 in binary notation, since 58 = $1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 21 + 0 \times 20$.

Figure B.12: ***Resource 12*** *covers the conversion of binary numbers to their decimal form.*

different presentations a student could find the decimal numbers one through ten and their binary counterparts.

B.4.2 Resource 10

The resource shown in Figure B.10 lists four examples of decimal numbers and their binary counterparts. The decimal numbers listed were one, three, six and nine.

B.4.3 Resource 11

A resource that shows a video made by Marija Kero of eHow[2] is shown in Figure B.11. The video shows how to calculate the conversion of a binary number to its decimal form, by demonstrating this for two examples. The video is displayed without any description.

B.4.4 Resource 12

The resource shown in Figure B.12 contains an adapted version of the definition of a binary number given by an online dictionary[3]. The adaptation contains two mistakes. The last two powers of two do not have a superscript display of the power, making it potentially confusing. This was however only discovered till late in the experiment by the author. The late discovery combined with the notion that these type of mistakes are very common in any large collection of educational material resulted in the decision to leave it like this.

[2]Source: `https://www.youtube.com/user/eHowFamily`
[3]`http://dictionary.reference.com/browse/binary`

A binary number is expressed in a system of numerical notation to the base 2, in which each place of a number, expressed as 0 or 1, corresponds to a power of 2. The decimal number 58 appears as 111010 in binary notation, since $58 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 21 + 0 \times 20$

B.5 Nim-Sum

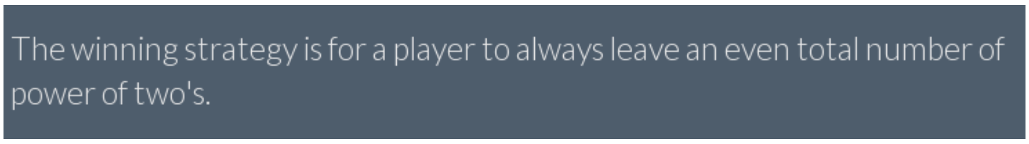
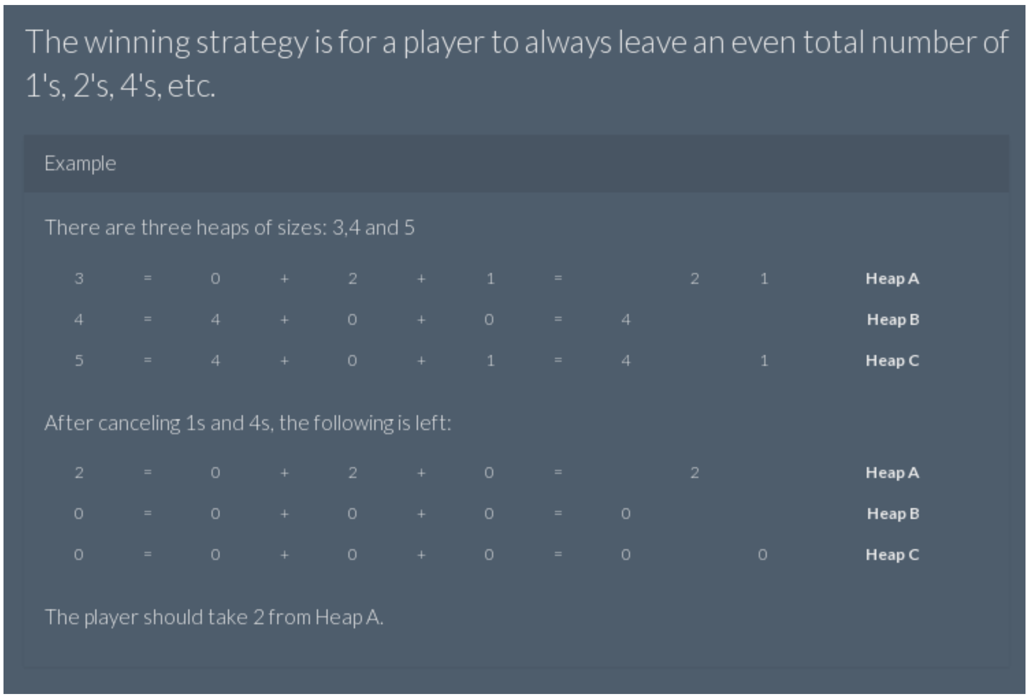

The winning strategy is for a player to always leave an even total number of power of two's.

Figure B.13: ***Resource 13*** *covers the nim-sum by means of pair cancelling*


The winning strategy is for a player to always leave an even total number of 1's, 2's, 4's, etc.

Example

There are three heaps of sizes: 3,4 and 5

3	=	0	+	2	+	1	=		2	1	**Heap A**
4	=	4	+	0	+	0	=	4			**Heap B**
5	=	4	+	0	+	1	=	4		1	**Heap C**

After canceling 1s and 4s, the following is left:

2	=	0	+	2	+	0	=		2		**Heap A**
0	=	0	+	0	+	0	=	0			**Heap B**
0	=	0	+	0	+	0	=	0		0	**Heap C**

The player should take 2 from Heap A.

Figure B.14: ***Resource 14*** *covers the nim-sum by means of pair cancelling*

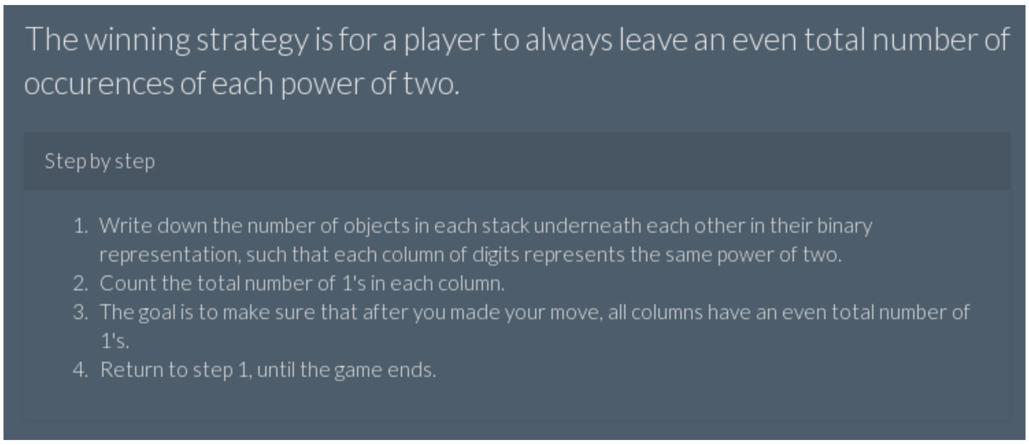
B.5.1 Resource 13

The resource shown in Figure B.13 contains an extremely brief textual explanation of pair cancelling, in the following words.

The winning strategy is for a player to always leave an even total number of power of two's.

B.5.2 Resource 14

Figure B.14 shows a resource that describes pair cancelling in one line and then applies that in an example. The stacks in the example contain three, four and five objects. The resource only shows the first next move the player should make, which is take 2 objects from the first stack.

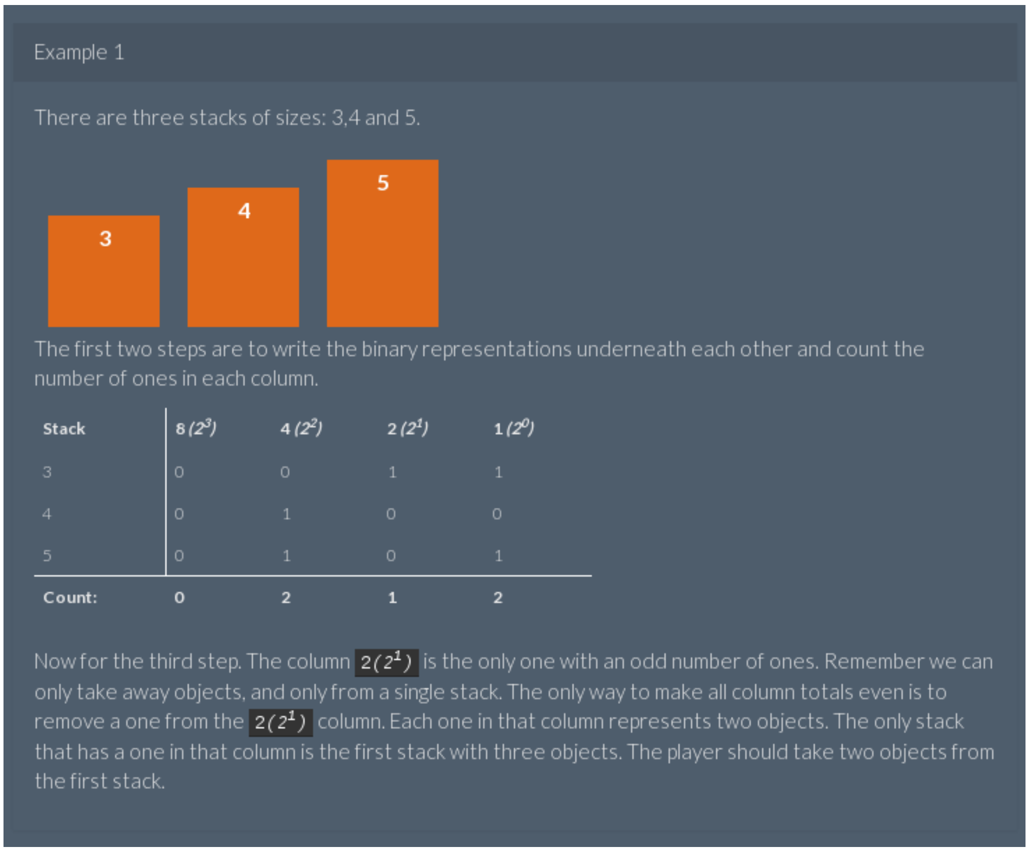


The winning strategy is for a player to always leave an even total number of occurences of each power of two.

Step by step

1. Write down the number of objects in each stack underneath each other in their binary representation, such that each column of digits represents the same power of two.
2. Count the total number of 1's in each column.
3. The goal is to make sure that after you made your move, all columns have an even total number of 1's.
4. Return to step 1, until the game ends.

Figure B.15: ***Resource 15*** *covers the nim-sum by means of pair cancelling. This screenshot is the first part of three screenshots of this resource.*



Example 1

There are three stacks of sizes: 3,4 and 5.

The first two steps are to write the binary representations underneath each other and count the number of ones in each column.

Stack	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
Count:	0	2	1	2

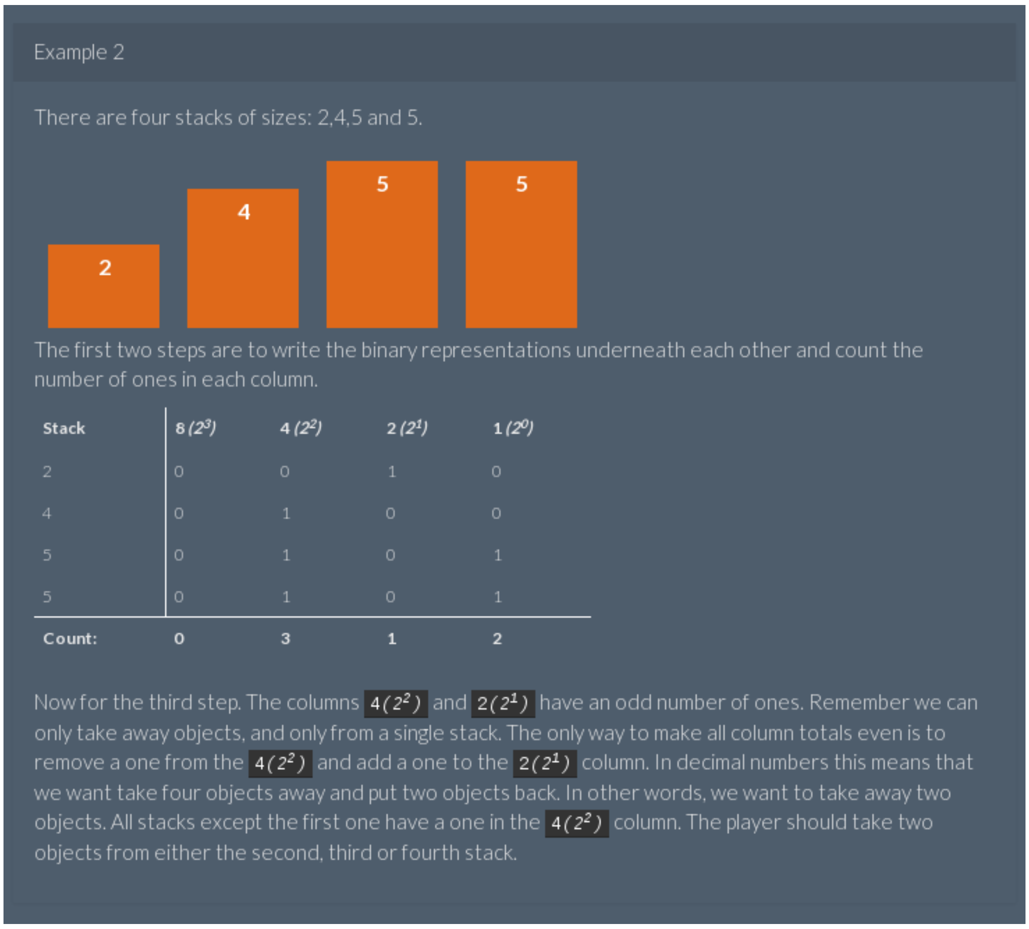
Now for the third step. The column `2(2^1)` is the only one with an odd number of ones. Remember we can only take away objects, and only from a single stack. The only way to make all column totals even is to remove a one from the `2(2^1)` column. Each one in that column represents two objects. The only stack that has a one in that column is the first stack with three objects. The player should take two objects from the first stack.

Figure B.16: ***Resource 15*** *covers the nim-sum by means of pair cancelling. This screenshot is the second part of three screenshots of this resource.*

B.5.3 Resource 15

Figure B.15, Figure B.16 and Figure B.17 are screenshots of a resource that gives the following step-by-step description of how to apply pair cancelling.

1. Write down the number of objects in each stack underneath each other in their binary



Example 2

There are four stacks of sizes: 2,4,5 and 5.

The first two steps are to write the binary representations underneath each other and count the number of ones in each column.

Stack	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
2	0	0	1	0
4	0	1	0	0
5	0	1	0	1
5	0	1	0	1
Count:	**0**	**3**	**1**	**2**

Now for the third step. The columns `4(2²)` and `2(2¹)` have an odd number of ones. Remember we can only take away objects, and only from a single stack. The only way to make all column totals even is to remove a one from the `4(2²)` and add a one to the `2(2¹)` column. In decimal numbers this means that we want take four objects away and put two objects back. In other words, we want to take away two objects. All stacks except the first one have a one in the `4(2²)` column. The player should take two objects from either the second, third or fourth stack.

*Figure B.17: **Resource 15** covers the nim-sum by means of pair cancelling. This screenshot is the last part of three screenshots of this resource.*

representation, such that each column of digits represents the same power of two.

2. Count the total number of 1's in each column.

3. The goal is to make sure that after you made your move, all columns have an even total number of 1's.

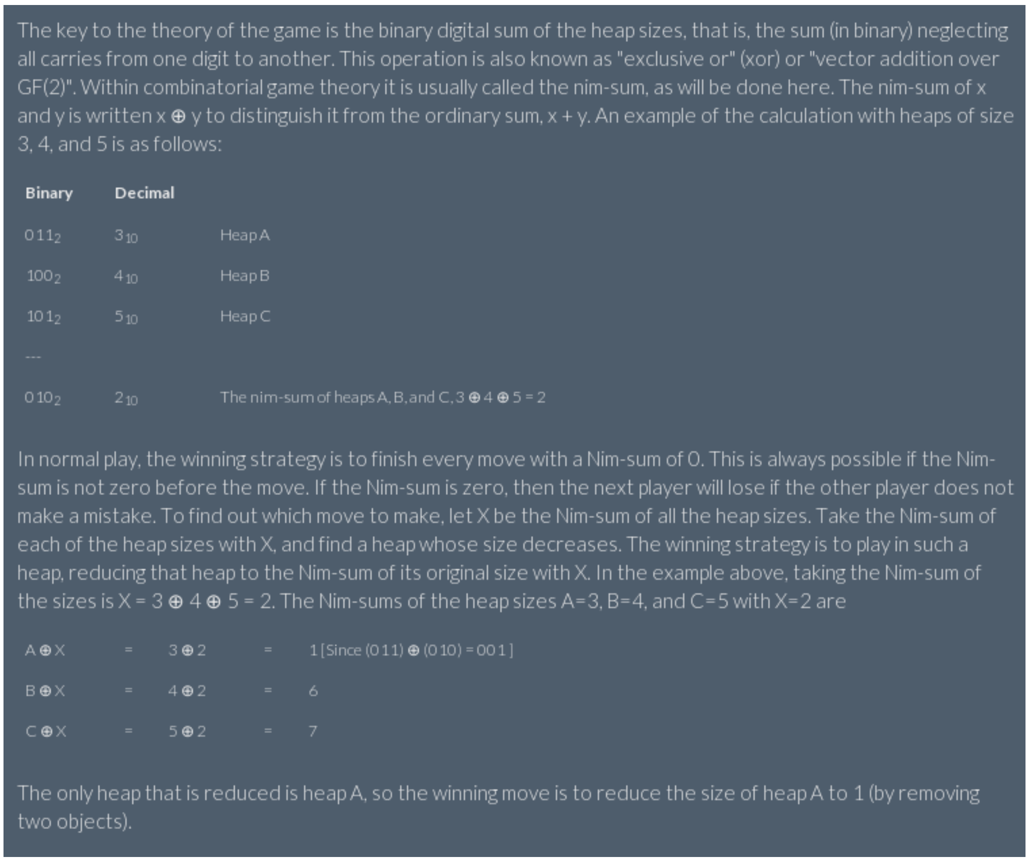
4. Return to step 1, until the game ends.

Furthermore it provides an explicit walkthrough of each of these steps for two examples. One containing three stacks of three, four and five objects, and one containing four stacks of two, four, five and five objects.

B.5.4 Resource 16

This last resource for the Nim-Sum knowledge component takes a different approach where the nim-sum is explained using the application of the binary XOR operation, which essentially is what the nim-sum operation really is. The description originates from the English Wikipedia article[4] about Nim. Figure B.18 shows a screenshot of this resource.

[4]`http://en.wikipedia.org/wiki/Nim`



The key to the theory of the game is the binary digital sum of the heap sizes, that is, the sum (in binary) neglecting all carries from one digit to another. This operation is also known as "exclusive or" (xor) or "vector addition over GF(2)". Within combinatorial game theory it is usually called the nim-sum, as will be done here. The nim-sum of x and y is written x ⊕ y to distinguish it from the ordinary sum, x + y. An example of the calculation with heaps of size 3, 4, and 5 is as follows:

Binary	**Decimal**	
011_2	3_{10}	Heap A
100_2	4_{10}	Heap B
101_2	5_{10}	Heap C

010_2	2_{10}	The nim-sum of heaps A, B, and C, 3 ⊕ 4 ⊕ 5 = 2

In normal play, the winning strategy is to finish every move with a Nim-sum of 0. This is always possible if the Nim-sum is not zero before the move. If the Nim-sum is zero, then the next player will lose if the other player does not make a mistake. To find out which move to make, let X be the Nim-sum of all the heap sizes. Take the Nim-sum of each of the heap sizes with X, and find a heap whose size decreases. The winning strategy is to play in such a heap, reducing that heap to the Nim-sum of its original size with X. In the example above, taking the Nim-sum of the sizes is X = 3 ⊕ 4 ⊕ 5 = 2. The Nim-sums of the heap sizes A=3, B=4, and C=5 with X=2 are

A ⊕ X	=	3 ⊕ 2	=	1 [Since (011) ⊕ (010) = 001]
B ⊕ X	=	4 ⊕ 2	=	6
C ⊕ X	=	5 ⊕ 2	=	7

The only heap that is reduced is heap A, so the winning move is to reduce the size of heap A to 1 (by removing two objects).

Figure B.18: ***Resource 16*** *covers the nim-sum by means of XOR calculations*

C

Nim Test Questions

C.1 Rules of Nim

1. Can you take objects from more than one stack?
   - I have no idea
   - No, you can only take one object at a time
   - No, you can only take objects from a single stack
   - Yes, you are allowed to do that
   - Yes, but only if there are not enough objects on one stack
2. Bob and John are playing the normal version of a Nim game. John takes the last object on table. Who won?
   - I have no idea.
   - Bob won
   - John won
   - Nobody won yet
   - That depens on whether it was John's second turn
3. How many objects are you allowed to take away from a stack?
   - I have no idea.
   - Exactly one object.
   - You have to take all the objects of that stack that you chose.
   - You have to take at least one object.

C.2 Intuition

1. On the table are three stacks. The first stack is empty, the second stack has two objects and the third stack has one object. What is the best move to make?
   - I have no idea.

- Take one object from the second stack
- Take two objects from the second stack
- Take one object from the third stack

2. On the table are three stacks. The first stack has two objects. The second stack has two objects. The third stack has one object. What is the best move to make?
   - I have no idea.
   - Take one object from the first stack
   - Take two objects from the first stack
   - Take one object from the second stack
   - Take two objects from the second stack
   - Take one object from the third stack
3. On the table are three stacks. All stacks have two objects. What is the best move to make?
   - I have no idea.
   - Take one object from any single stack
   - Take two objects from any single stack

C.3 Binary numbers

1. What is the binary representation of the decimal number 10?
   - I have no idea.
   - 0010
   - 1000
   - 1010
   - 1111111111
   - 0000000010
2. What is the decimal representation of the binary number 1000?
   - I have no idea.
   - 1
   - 4
   - 8
   - 1000
3. Which number is bigger, the binary number 1001 or the decimal number 1001?
   - I have no idea.
   - The binary number is bigger.
   - The decimal number is bigger.
   - They are equal.

C.4 Nim-Sum

1. On the table are three stacks. The first stack has 10 objects. The second stack has 8 objects. The third stack has 6 objects. From which stack do you take objects when making an optimal move?
   - I have no idea.
   - The first stack.
   - The second stack.
   - The third stack.
   - Any stack will do.
2. On the table are three stacks. The first stack has 10 objects. The second stack has 8 objects. The third stack has 6 objects. How many objects will you take from a stack when making an optimal move?
   - I have no idea.
   - Two objects.
   - Four objects.
   - Six objects.
   - Eight objects.
3. On the table are five stacks. The first stack has 9 objects, the second 8, the third 7, the fourth 6 and the fift stack has 5 objects. Which move will you make?
   - I have no idea.
   - Nine objects from the first stack.
   - Eight objects from the second stack.
   - Seven objects from the third stack.
   - Six objects from the fourth stack.
   - Five objects from the fift stack.

D

Simulation Convergence Data

Table D.1: Simulation convergence results of ten repetitions in the normal environment

Setup	1	2	3	4	5	6	7	8	9	10
pop5ep5	79	19	34	20	19	17	19	24	19	64
pop5ep10	19	29	19	39	49	19	19	19	19	99
pop5ep20	29	769	29	89	89	89	29	33	69	29
pop10ep10	25	29	19	29	25	39	19	31	29	19
pop10ep20	49	37	29	35	29	29	39	32	29	34
pop20ep20	39	29	35	32	43	41	35	41	33	43
el0mu05	19	27	23	19	25	27	19	27	29	19
el2mu25	29	22	31	19	47	69	53	33	31	23
el0mu25	30	23	40	29	25	21	32	39	41	30
el1mu05	27	23	27	23	23	19	27	27	35	19
el1mu25	22	33	23	30	40	27	29	29	21	19
el2mu05	23	32	27	23	21	23	19	27	19	27

Table D.2: Simulation convergence results of ten repetitions in the noisy environment

Setup	1	2	3	4	5	6	7	8	9	10
pop5ep10	19	19	19	139	19	19	69	23	19	23
pop10ep10	19	19	19	19	29	19	29	27	29	19
pop5ep20	49	29	649	49	91	69	33	33	29	29
pop10ep20	69	37	29	29	29	37	29	38	49	35
pop20ep20	36	51	29	43	49	55	41	57	41	41
pop5ep5	94	24	19	24	37	23	19	26	19	25
el0mu05	31	27	27	19	23	29	27	19	27	79
el2mu25	25	19	61	29	59	29	27	29	19	29
el0mu25	39	29	29	29	38	19	25	29	28	27
el1mu05	29	31	31	27	26	21	40	37	40	33
el1mu25	29	27	31	25	29	26	27	27	29	37
el2mu05	25	29	29	19	28	19	26	27	19	29