Radboud University Nijmegen


Institute of Computing and Information Sciences

# Towards Improving the Performance of Model Driven Optimisation

From Domain Models to Low-Level Encodings and Back

Master Thesis Software Science

*Supervisor:*
dr. Daniel Strüber

*Author:*
Lars van Arragon

*Second Supervisor:*
dr. CDN (Diego) Damasceno

*Second reader:*
dr. Nils Jansen

December 2021

# Abstract

The field of Model Driven Optimisation combines the two fields of Model Driven Development and Search Based Software Engineering. Models are used as first class citizens that represent the optimisation problem within Search Based Software Engineering upon which the optimisation algorithms are applied. A meta model is used to describe a modelling language which is used as ruleset to define the initial population of the search space. Exploration operators, which are ideally also models, and evaluation functions are subsequently used to explore the space. During the exploration of the search space the models that are usually big and complex are copied often. This slows down the performance of an optimisation algorithm significantly. There exist multiple tools that implement Model Driven Optimisation, we focus on MDEOptimiser. Within this thesis we provide the first steps towards improving the performance of Model Driven Optimisation techniques by contributing an encoding of these models that does not replace, but compliments them. We include a formal framework for expressing what such an encoding looks like, together with a Java library that implements these ideas in the context of MDEOptimiser. Additionally, we evaluate the improvements to the performance and offer meaningful discussion on the results that includes pointers to where future research can be conducted.

# Contents

# 1 Introduction

In the field of Model Driven Optimisation [28] the two fields of Model Driven Engineering [21] and Search Based Software Engineering [13] are combined. Model Driven Optimisation uses models to represent the optimisation problem and directly applies the search space exploration and fitness evaluation to them. There exist several tools that apply this concept like MDEOptimiser [4], MoMOT [7], Viatra-DSE [1], and FitnessStudio [23].

Within Model Driven Optimisation, a user defines a meta model that describes a modelling language which all models in the population follow. They also specify the space exploration and fitness evaluation applied to the models. During the execution of the optimisation algorithm the population is frequently copied as the algorithm evolves and evaluates them. In practice, these models are big data structures with a lot of information which makes the copying of them slow. While Model Driven Optimisation has great benefits to expressiveness and usability, the slow performance on big models is one of the main drawbacks of this approach.

This thesis contains the first steps towards improving the performance of Model Driven Optimisation using low-level encodings as the basis upon which the optimisation algorithms are applied. To alleviate the drawback we attempt to encode the model in such a way that we do not completely replace the model, but so that we can use them together and they complimentary of each other. We focus on the MDEOptimiser with its accompanying case studies as basis to evaluate our contributions.

We include a formal framework for expressing what an encoding looks like for any given model instance and meta model. Based on this framework we implemented a Java library for encoding any meta model with model instance that is expressed within the EMF framework. With this library we conduct several experiments to validate that the performance of the copying during the algorithm is indeed improved. We also contribute a meaningful discussion on the results of these experiments and provide pointers to where future research can be conducted.

The first section will contain the background information surrounding the topic of model based optimisation techniques. We introduce two case studies upon which we have done experiments. Additionally we describe the definitions and assumptions we use throughout the thesis. After the background information we introduce the research questions within the thesis that we aim to answer. We follow this up by explaining the concept of the encoding both formally and implemented within a Java library. To then verify that these concepts work we describe a methodology for doing experiments on the case studies to measure the performance of both the encoding based and model based approaches. We then introduce the implementation of these experiments, after which we evaluate the results. We then discuss the results, introduced concept and the problems that arose during the thesis. Lastly we conclude by summarising our contributions, the questions we answered and the questions we raised. Together with some pointers for further research to develop this field.

All of the code written for this thesis including the figures containing the results can be found on GitHub[1][2].

[1] `https://github.com/larsvanarragon/mde_optimiser-hilo/tree/nobitset`
[2] `https://github.com/larsvanarragon/mde_optimiser/tree/encoding`

# 2 Background

In the following sections we will explain the background of the concepts used within this thesis. We will include some explanations of the case studies we use to measure certain aspects of our contributions.

## 2.1 Model Driven Optimisation

The concept of Model Driven Optimisation (MDO) stems from the combination of Model Driven Engineering (MDE) and Search Based Software Engineering (SBSE). Its idea is that models can be used as a declarative problem formulation for the complex problem domains within software engineering [28], and can then be subsequently used as the solution representation in search based optimisation techniques. This idea enables users to approach the problem with domain-specific knowledge instead of a low-level technical encoding that replaces the models.

### 2.1.1 Model Driven Engineering

Model Driven Engineering (MDE) is a paradigm where models are not just used for documentation but are first class citizens within the space of software artifacts [21]. A tutorial on the subject in [3], notes that models directly represent their subject to allow for a more direct coupling of problems to solutions. The information within these models is intended to be used by some tooling that is able to bridge the gap between the domain concepts and implementation technologies. An example of this idea is a tool that translates models adhering to the UML standard to executable Java code.

### 2.1.2 Search Based Software Engineering

The goal of Search Based Software Engineering is to reformulate Software Engineering problems as search based optimisation problems [12] and address them using meta-heuristic techniques. At its core, SBSE is comprised of two ingredients: a representation of the problem at hand and the definition of a fitness function which operates on this representation [13]. The problem at hand then becomes an optimisation problem where the software engineer intends to maximize the result of the fitness function by searching within the realm of possible solutions for optimal and near-optimal candidates. For practicality search based algorithms usually include mechanisms for generating new candidate solutions from existing solutions and a way to apply the fitness function to all solutions. At the start of such an algorithm an initial set of candidate solutions is predefined.

As an example suppose the question: 'What is the smallest set of test cases that cover all branches in this program?'. This is called the budgeted maximum coverage problem where we attempt to minimize the budget [16]. A simple representation of this problem would be a bit vector in which each bit represents a test case and is used within the set. The fitness function can then evaluate the solutions based on the amount of used test cases and how much the selected test cases cover all branches in the program. The initial set of candidate solutions can be a randomly generated set of bit vectors. An example for the mechanism for creating new candidate solutions would be to take the fittest candidates and mutate them slightly while also keeping a few of the current fit candidates.

One concept that is relevant in this thesis is a Pareto front. When we have a multi-objective optimisation algorithm where the population is evaluated using multiple fitness functions we do not simply have one 'best' candidate solution. In this case we have an entire front of good solutions that balance the multiple objectives in different ways. This concept is called the Pareto front. In general, a Pareto front contains candidate solutions that contain a 'best' balance. For further reading on Pareto fronts we refer to [26]. It can be the case that we have two Pareto fronts where one of them has better solutions than all or part of the solutions in the other. This concept is what we call how much a Pareto front A dominates another Pareto front B. We can quantify this using the Hypervolume indicator, which indicates in percantages how much one Pareto front dominates another. For further reading on Hypervolume indicators we refer to [8].

### 2.1.3 Combination

The work of Zschaler and Mandow in [28] proposes that instead of creating a separate secondary encoding for the problems, the models and meta-models should directly be used as input for the optimisation algorithms. This way the optimisation algorithms can make use of all the information present in these models. They state that current encodings of the models do not preserve well-formedness and have difficulty ensuring locality.

Applying the entire optimisation algorithm to a model means that this model is copied and transformed many times resulting in a high number of computations. These models are usually complex data structures which contain a lot of information, of which some of this information is static in its nature. In the example given in [28] of the Zoo DSL we can see that the objects do not change, but only their relationships with each other. It is reasonable to imagine that the optimisation could be done more efficiently by using an encoding to focus on the part that needs to be optimised while storing static information elsewhere. By this we mean that we do not aim to create an encoding that completely replaces the model. We aim to contribute an encoding that brings all the relevant information for optimising the model to a low level. The model then compliments the encoding by providing it the information which remains static or does not need to be encoded. This also ensures that improvements to the need of repair, well-formedness and locality as mentioned in [28] are preserved.

### 2.1.4 MDEOptimiser

In order to obtain a qualitative comparison for the contributions of this thesis a concrete tool in which to implement it should be picked. For this purpose we have picked the tool MDEOptimiser introduced in [4]. The MDEOptimiser tool follows a model based approach and is built atop the Eclipse Modelling Framework [22] [27], the Henshin model transformation language [24] [18] [2], and the MOEA evolutionary search framework [10] [11]. The reasoning for this choice is that the tool has shown a better performance than another tool with a similar, but slightly different encoding [14]. To obtain this result they have measured the performance of MDEOptimiser using several case studies. Another reason is that MDEOptimiser is a follow up paper from [28] where they introduced the idea of combining MDE and SBSE. The case studies also form an excellent starting point for implementing ideas for the improvements.

There exist several other tools that also facilitate model driven optimisation. We briefly mention these tools here with relevant references. They can be used to further deepen the reader's knowledge of what is available within the domain of model driven optimisation.

- Marrying Search-based Optimization and Model Transformation Technology (MoMOT) [7]
- VIATRA Design Space Exploration Framework (VIATRA-DSE) [1], [20]
- FitnessStudio [23]

### 2.1.5 Eclipse Modelling Framework

The Eclipse Modelling Framework (EMF) is a modelling framework and code generation facility for building Java application based on structured data models. Within EMF a user can describe class models from which Java code is generated. These models can be stored statically in files and be retrieved during runtime. The EMF is used within MDEOptimiser to define meta models and model instances for the optimisation problems a user has. Such a class diagram is an example of a meta model, and the concrete classes that follow this diagram is an example of a model instance.

### 2.1.6 MOEA Framework

The MOEA evolutionary search framework allows a user to define an optimisation problem with its corresponding fitness functions, evolution operators and evolution algorithm. For our purposes we only have to implement the optimisation problem, define its population and how it evolves. We leave the fitness functions to the case studies provided by MDEOptimiser and the evolution algorithm to those implemented by MOEA. There are three important classes we have to instantiate; namely, the AbstractProblem, Variable and Variation classes. The Variable class represents a single member of the population, and a population has many such variables. To mutate these variables the Variation class is used. This class describes what happens to a non-empty array of parents when they evolve. A single run of the framework can concatenate such Variations to describe complex evolution behavior. Lastly, the AbstractProblem class describes how to evaluate Variables and how to instantiate new Variables when creating a new population.

### 2.1.7 Henshin

Within MDEOptimiser the Henshin model transformation language is used to define the exploration operator for the search based algorithm. Such a model transformation is called a Henshin rule and is in itself a model. The model contains a pattern that has to be matched within the model. It also describes what happens to the pattern when the Henshin rule is applied. This can be done by changing references, creating and deleting objects and even altering attributes within the objects. The Henshin rules are described using the aforementioned EMF and are applied to the models by MDEOptimiser instantiating them within MOEA.

### 2.1.8 Case Studies

As mentioned we will use the case studies as they are specified in [4]. Namely, we will use the case studies Next Release Problem (NRP) and Class Responsibility Assignment (CRA). Both of these case studies are modelled within MDEOptimiser using all the tools mentioned above.

**Next Release Problem**

The essence of the Next Release Problem is that we have several software artifacts that have a cost of realisation and an importance for a customer. These customers also have a differing importance. Now the question becomes what artifacts do we realize for the next release? We want to minimise the cost of the realisation and maximize the customer satisfaction. This problem can be modeled as seen in figure 1. In this case, NRP can have several solutions which contain some software artifacts to be realised. However, in MDEOptimiser every member of the population has one solution for NRP.

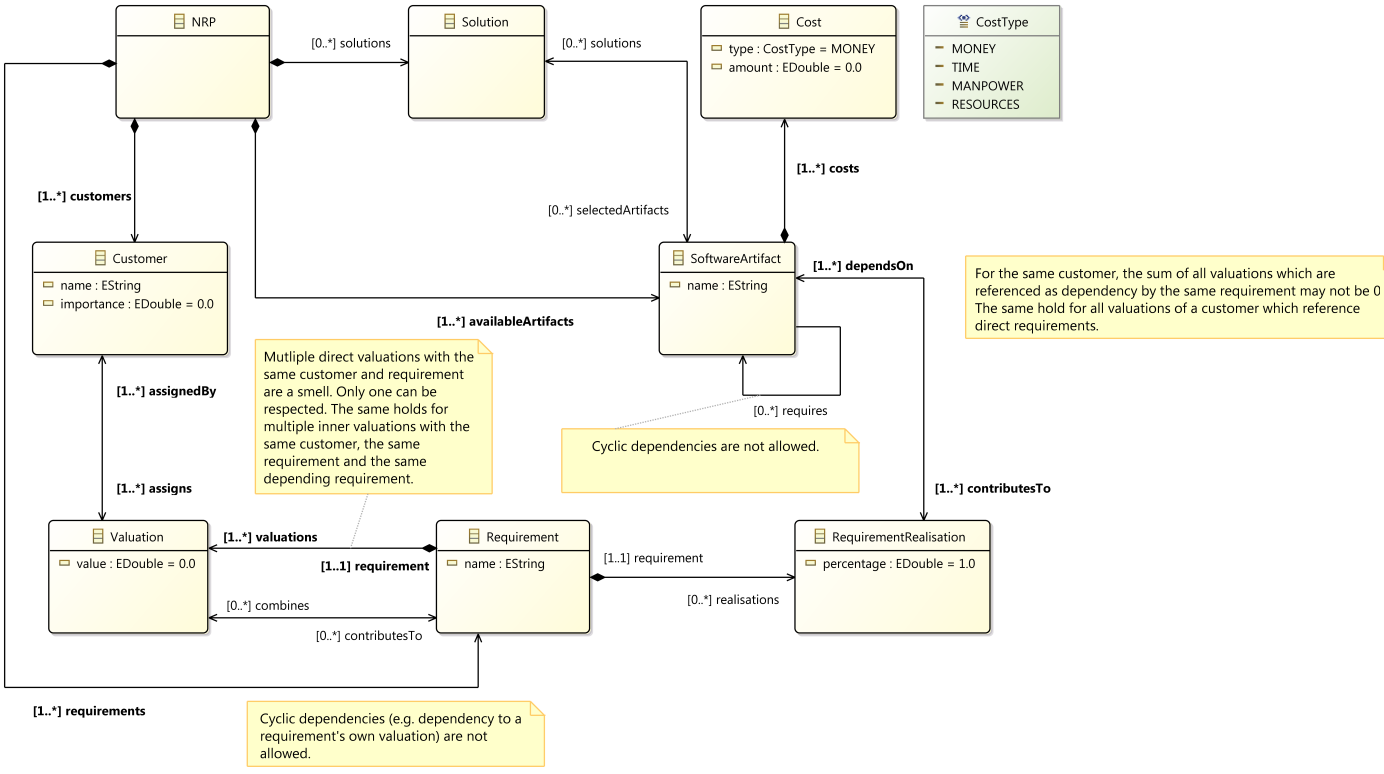


Figure 1: Meta model for NRP [4]

**Class Responsibility Assignment**

The Class Responsibility Assignment problem is a case study on how to transform a procedural program with attributes (i.e. variables) and methods to an object oriented program, while maintaining good cohesion and coupling. These attributes and methods are dependant on each other which should be taken into account within the solution. The meta model for this problem is given in [4] as follows, and is shown in figure 2.

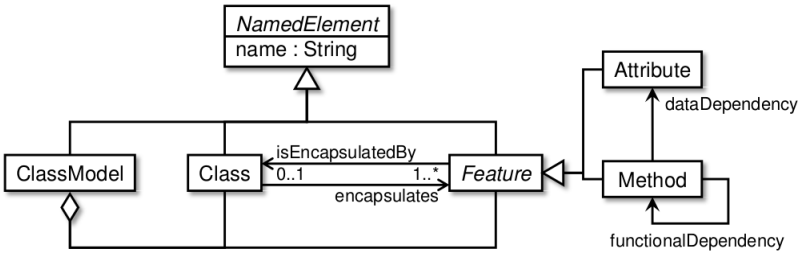


Figure 2: Meta model for CRA [4]

## 2.2 Definitions

In this section we will shortly go over the terminology used in this thesis and explain what we mean by them. We rely on the definitions from [21] by using them and deriving new definitions based on them.

- **Model**, a model is 'a system that helps to define and to give answers of the system under study without the need to consider it directly' [21].
- **Meta model**, a meta model is 'a model that defines the structure of a modelling language' [21].
- **Model instance / Dynamic model**, a dynamic model or model instance is a model that conforms to a certain meta model.
- **Exploration operators**, operators used within an optimisation technique to explore the solution space. This includes mutation and crossover operators.

The relation of the meta models to models and a system is described in figure 3. This shows that a modelling language is described by one or more meta models and that the models are elements of this language. That is to say, the language can be seen as a set which contains all valid models.

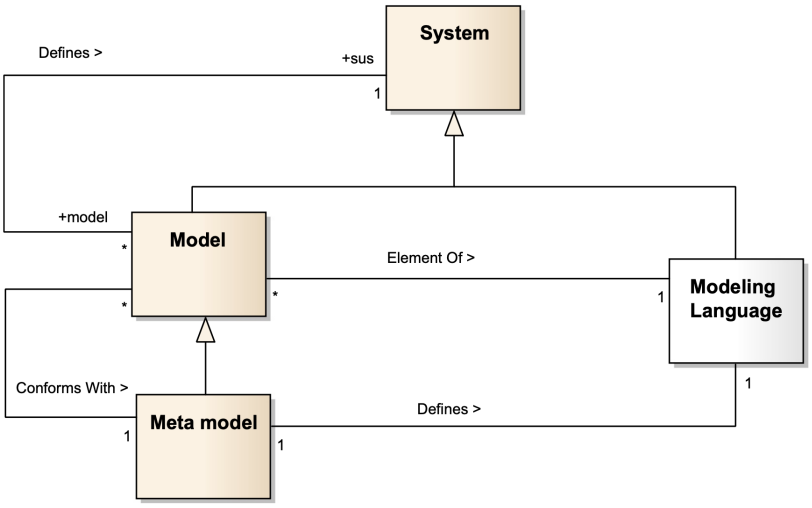


Figure 3: Relationships between model and meta model [21]

Models can be described very abstractly, [19] says that a model is an abstraction of a system that allows predictions and inferences. They also more formally describe the concepts of a model and a meta model. Most notably, they suppose that, in the spirit of work from [15], a formal model of a system already exists. As an example they provide a structured set $(S, r_S)$, where $S$ is some set of elements and $r_S$ denotes their relationships. We will use these ideas in this thesis as well to provide a more structural approach to these models. Additionally, in the spirit of [15] we will not provide any formal definition of the meaning of these models, just their structure. We assume that there are rules that give the models their meaning, but will leave their definitions implicit.

Most modeling languages, especially those considered in the context of model driven engineering and optimisation, follow a 'boxes-and-lines' paradigm. For the notation we will now introduce, we assume that any model we encounter can be seen as such. Which means that we can see any model as a graph where the vertices are the 'boxes' and the edges are the 'lines'. Usually, these models contain more information than just boxes and lines but the structure is all we need.

**Definition 2.1.** *Let $L$ be a set of labels. A **meta model** is a directed graph $(V, E)$ that allows self loops where $V = \{ (l, P) \mid l \in L$ and $P$ is a predicate that denotes whether an object conforms to this vertex $\}$ and $E = \{ (v_{from}, v_{to}, l, Q) \mid (v_{from}, v_{to}) \in V \times V, l \in L$, and $Q$ is a predicate that denotes whether all relationships conform to this edge $\}$. Vertices of a **meta model** are called classes, edges are called references or relations and the set of labels is called the type names.*

The predicates in this definition dictate what the modelling language looks like that is generated by this meta model. In this context these predicates are the constraints and rules on the classes and relations. We leave very little restrictions on the meta model as we only care about whether all objects within a model instance conform to the meta model. The labels are used to give names to the boxes and lines. In the context of [19] the vertices represent the classes and the edges are the relationships/references.

Recall the meta model for NRP in figure 1. Let us apply this definition to this example where we consider only the Solution and SoftwareArtifact classes. We are left with three references solutions, selectedArtifacts and requires. The rules of a UML class model are quite complex, so we will again leave these as predicates of which we implicitly understand how they work. The formal instantiation of this small NRP meta model is:

$$
\begin{aligned}
V = \{&(Solution, S), (SoftwareArtifact, SA)\} \\
E = \{&((Solution, S), (SoftwareArtifact, SA), selectedArtifacts, SEL), \\
&((SoftwareArtifact, SA), (Solution, S), solutions, SOL), \\
&((SoftwareArtifact, SA), (SoftwareArtifact, SA), requires, REQ)\}
\end{aligned}
$$

This way of representing the model does not encompass all of the information present in a UML class model. For example we have no obvious way of representing that a SoftwareArtifact has a name. We choose here to represent all the extra information as the implicitly defined predicates within the meta model. This moves the problem to the model instances and further builds upon this idea that a meta model simply defines a modelling language to which an instance can conform. We are now ready to formally explain what it means for a model to be a model instance.

**Definition 2.2.** *Suppose a* ***meta model*** $(C, R)$*. We call a directed graph* $(V, E)$ *that allows self loops a* ***model instance*** *of* $(C, R)$ *when:*

(1) $\forall v \in V(\exists(l, P) \in C(P(v)))$
(2) $\forall(v_{from}, v_{to}) \in E(\exists((x, X), (y, Y), l, Q) \in R(X(v_{from}) \wedge Y(v_{to})))$
(3) $\forall(x, X), (y, Y), l, Q) \in R(Q(E))$

Again, we do not make explicit what information is present within this graph, because we do not require this information for the encoding. The point of these two definitions is to enable reasoning about the structure of meta models and model instances. Intuitively we have that a graph is a model instance of a meta model if all of the vertices are compliant with the modelling rules of at least one class in the meta model. That for all of the edges in the graph they also are defined as such in the meta model. And that all edges in the graph comply with the modelling rules for each edge in the meta model. Compliance to these modelling rules simply means that the modelling language described by a meta model accepts the model instance as valid.

In order to easily show that an edge or vertex within a model instance conforms to the meta model we overload the operator $\in$. Suppose we have a meta model $(C, R)$ with a corresponding model instance $(V, E)$, we define that for any $c \in C$, $r \in R$, $v \in V$, and $e \in E$:

$$v \in c = v \in (l, P) = P(v) \tag{1}$$

$$e \in r = (e_{from}, e_{to}) \in ((x, X), (y, Y), l, Q) = X(e_{from}) \wedge Y(e_{to}) \tag{2}$$

This overloading automatically also gives us definitions for $\notin$. In addition to this abbreviation we also textually refer to $v \in c$ as an object in its respective class.

Note that this abbreviation only tackles (1) and (2), we do not check whether the rules defined by the edges apply to the entire edge set, we assume that since $(V, E)$ is already a model instance of $(C, R)$ this holds. From here on we will usually refer to any vertex or edge within the meta model with an uppercase symbol and those within the model instance with a lowercase. This way

of defining the conformance to the meta model also enables us to quantify over these models. In other words, we can say that $\forall x \in X$ means all $x$ that conform to $X$, or all object instances that conform to the class.

We can illustrate the definition of a model being an instance of a meta model with the example model instance in figure 4 for the aforementioned small NRP meta model.

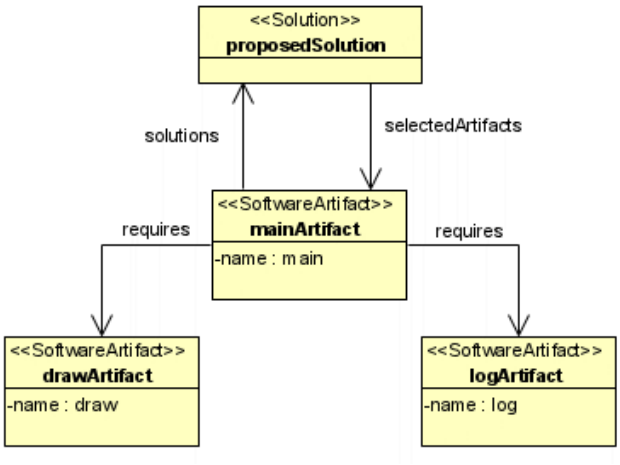


Figure 4: Example model instance of the small NRP meta model

This model instance contains only 1 solution called proposedSolution, with three artifacts: mainArtifact, drawArtifact, and logArtifact. We can view this example as a graph with 4 vertices and 4 edges:

$$V = \{proposedSolution, mainArtifact, drawArtifact, logArtifact\}$$
$$E = \{(proposedSolution, mainArtifact), (mainArtifact, proposedSolution),$$
$$(mainArtifact, drawArtifact), (mainArtifact, logArtifact)\}$$

We have annotated figure 4 with what these edges and vertices conform to in the meta model. In reality, these edges and vertices would contain more information to determine that this in fact holds. For now we just assume that the relevant predicates hold for the relevant vertices and edges, meaning that this example is a model instance for the small NRP meta model.

# 3 Research Questions / Concerns

As stated in the earlier sections the main drawbacks of current model driven optimisation tools is the frequent copying of these models during each evolution step. The main question of this thesis is:

**Q1** Can we use an encoding instead of a model to copy a population of a model driven optimisation problem faster?

As noted, we focus in this thesis on the model driven optimisation tool MDEOptimiser, but the same concern also applies to other tools in this domain (e.g., MoMOT, Viatra-DSE, FitnessStudio). We use the tools and libraries within MDEOptimiser as basis to perform experiments and tests.

A problem we can run into is that while we gain time during the copying, we lose it again at other steps in the algorithm. To this end we can wonder whether we can easily apply the encoding within an existing tool and immediately see improvements. This gives rise to the following question:

**Q2** Does using an encoding reduce the overall execution time of the model driven optimisation algorithm?

When we find a positive answer to either of these questions we know that we have at least reached a partial improvement to the performance.

# 4 Concept

Within this section we will on a more formal level explain the concept of the encoding. Firstly, we explain the initial concept that is the idea of the encoding. After that we establish several assumptions and restrictions that we have surrounding the encoding. Taking all these things into account, we propose the formal encoding. After that we describe how to implement this in a model driven optimisation technique. Lastly we will describe a Java library we implemented based on these concepts.

## 4.1 Initial Concept

Within this section we will explain the conceptual approach taken towards creating an encoding for any model driven optimisation technique like for example MDEOptimiser. In this thesis we focus on implementing the encoding for MDEOptimiser, but the core concept can be extrapolated to any model driven optimisation technique. This means that, as we mentioned, any modelling technique that follows the boxes-and-lines paradigm can be encoded using the concepts defined in this thesis.

The idea at its core is to look at each relation within the meta model and construct a bit vector for each object that has one or many of these references to the other objects. A bit being 1 then shows that there is a reference between them. For this approach we assume that any one way relation in the meta model can be seen as a one-to-many relationship. In case of a one-to-one relation there will be only one bit in the vector that is set to 1. A many-to-many relation in this case is not one way, and is seen as two one-to-many relationships.


Figure 5: A simple meta model

Figure 5 represents a very simple meta model. To obtain an encoding from this meta model we have that for a class $X$ we look at all of the edges $(v_{from}, v_{to}, l, Q) \in E$ where $v_{from} = X$. We can then construct a bit vector $[b_1, \cdots, b_n]$ for each $x \in X$ where $n$ is the amount of $y \in Y$. Meaning that if $b_i = 1$, then $x$ is related to $y_i$, else $x$ is not related to $y_i$.

With this way of encoding all the relations within any given meta model we need to be able to figure out what the actual object instances are we point to using such a $y_i$. At this point it is sufficient to just assume these instances are stored somewhere and we have a way of retrieving them.

In a given model driven optimisation problem we then have these bit vectors be the population upon which the exploration operators are applied. Meaning that the encoding is the part that is copied and evolved, not the model itself anymore. This should then give us a better performance for the optimisation problem.

## 4.2 Assumptions

For this idea to work we make several assumptions based on the information that is provided by the optimisation problem, and the models used within. As this work is based on MDEOptimiser, we take the information that this tool combined with EMF provides. These assumptions are:

- We have access to both the information from the meta model and model instances of any given problem. Specifically, we need to be able to extract the meta relation information for any given object. The definitions we gave have already assumed that we can somehow access this information. Standard modeling platforms such as EMF provide this information via a suitable API.
- For any model driven optimisation problem, we have that applying its exploration operators never introduces violations of the modelling language specified by the meta model. This is in one part needed for being able to view any one way relation in the meta model as a one-to-many relation. And also to be able to guarantee that when we evolve a model instance of a meta model it stays a model instance.
- Lastly, we also assume that we have some way of identifying, storing, and retrieving the objects in the model instances. This means that next to having access to the relevant information, we also have access to code/infrastructure that can store the object instances and if necessary generate identifiers for them.

## 4.3 Restrictions

As a starting point we only look at optimisation problems that do not have exploration operators that alter the attributes of an object. Having this restriction means we can ignore the increased complexity that is created by allowing this. Multiple encodings usually share references to some object instances which means that if we were to alter the attributes in these objects we would have to deal with potentially keeping track of each variant of these objects for every member of the population. Suppose we were to solve this by saving a copy of each variant, we would then quickly lose the benefits of encoding the model this way, as we still copy the actual objects a lot, and have to store the slight variations of them as well. It is reasonable to assume that the exploration operators do not alter the attributes of the classes in the model as none of the example problems provided as part of the MDEOptimiser project do [4]. We do acknowledge that for completeness it is preferable to support this, but this will be left as future work.

## 4.4 Proposed Encoding

In this section we formally propose the encoding to be used within model driven optimisation. We build upon the ideas described in the initial concept and concretely define how the model instances are retrieved and stored.

The first idea for the encoding was the creation of a bit vector for each vital one-to-many relationship. When taking into account that exploration operators can add and delete objects within the model instance this is no longer viable. Addition and deletion of objects implies that we have to scale the bit vector for each added object and keep track of its index. It is more complicated to delete objects as it is vital to preserve the indexes of the objects to be able to later retrieve them from the encoding. Another argument for not using bit vector is that when we want to retrieve the actual objects back from the encoding we would have to loop through the entire vector and check which index contain a 1.

For the encoding we discern between two types of relations here. The abstract- or meta-relationship, that is the edge in the meta model. And the relation instance, which is the edge in the model instance. The abstract relationship describes the allowance of a reference between one class and another, whereas the relationship instances is such an actual reference.

We concluded that a bit vector is not a viable way of encoding the model. Instead, we opted for a identifier based approach to the encoding. The basic premise is that we assume that we can create or generate an identifier for each object instance in the encoding. Using these identifiers we can then easily link both the abstract relation to its instances and the relation instance to its related objects. Intuitively we can see the encoding as two nested relationships. The deepest relationship is that between the object instances and its corresponding references. We in essence collect all of the identifiers for any objects referenced by an object. The highest relationship sorts all these object instances according to what abstract relationship they belong in the meta model. Recall the small NRP example, one abstract relationship we have here is the 'requires' relationship between SoftwareArtifact and Solution. Within the model instance in figure 4 we can see two instances of this abstract relationship, namely between mainArtifact and drawArtifact, and between mainArtifact and logArtifact. These references both start at mainArtifact so the identifiers for drawArtifact and logArtifact will be related to the identifier of mainArtifact.

### 4.4.1 Formal Definition

Within this section we introduce the formal definition of the encoding. We define a mapping from a meta model $(C, R)$ and its model instance $(V, E)$ to an encoding $\mathcal{P}(R_A, \mathcal{P}(R_I, \mathcal{P}(L)))$. Here we have that $R_A$ is a set of labels that denote the abstract relations within the meta model, $R_I$ is a set of labels that denote the relation instances within the model instance and $\mathcal{P}(L)$ is the powerset of labels. The powersets in the encoding denote that we have several abstract relationships who in turn have several relationship instances. All specific label sets like $R_A$ and $R_I$ have that $R_I \subseteq L$ and $R_A \subseteq L$. The usage of these names is just for convenience.

To enable the encoding to link between label representations of the object instances we assume that there is a bijective function $\mathbf{id} : V \to L$ from which we can obtain an identifier for any object instance $v \in V$. With its corresponding inverse which we call $\mathbf{obj} : L \to V$ which links any identifier to its object. Lastly we also assume that we have an operator $+$ which concatenates any two $k, l \in L$ labels. To start of this definition we need to gather all of the referenced objects of any object within the model instance as identifiers and relate them to the identifier of the object that references them.

**Definition 4.1.** *Suppose a graph $(V, E)$ with a vertex $v \in V$, a set of relation labels $R_I$ and a set of labels $L$. We call $r_I : (R_I, \mathcal{P}(L))$ a **relation instance** of $v$ if $r_I = (id(v), \{\ id(w') \mid (w, w') \in E \wedge w = v\ \})$*

Using definition 4.1 we can now obtain all of the **relation instances** for any given graph.

**Definition 4.2.** *Suppose a graph $(V, E)$. We define $\textbf{\textit{RelInst}}(V, E)$ as all of the relation instances for $(V, E)$. Namely, we define $\textbf{\textit{RelInst}}(V, E) = \{\ r_I \mid v \in V \wedge r_I$ is a **relation instance** of $v\}$.*

This gives us a set containing all relation instances for the graph. We now want to obtain all relation instances for any given abstract relation. For this we restrict the relation instances to a relation in the corresponding meta model.

**Definition 4.3.** *Suppose a **meta model** $(C, R)$ and a corresponding **model instance** $(V, E)$. For an arbitrary $r \in R$ we define $\textbf{\textit{RelInstRestr}}(V, E, r)$ as all of the relation instances for $(V, E)$ restricted to $r$. Namely, we define $\textbf{\textit{RelInstRestr}}(V, E, r) = \{\ (v, W/U) \mid (v, W) \in \textbf{\textit{RelInst}}(V, E) \wedge U = \{\ w \mid w \in W \wedge (\textbf{\textit{obj}}(v), \textbf{\textit{obj}}(w)) \in E \wedge (\textbf{\textit{obj}}(v), \textbf{\textit{obj}}(w)) \notin r\} \wedge W/U \neq \emptyset\}$.*

In essence, this definition removes all of the edges from **RelInst** that are not an instance of the meta relation $r$. Note that we are using the overloaded $\in$ operator defined in section 2.2. Using this restriction we can now define what an encoded abstract relation looks like.

**Definition 4.4.** *Suppose a graph $(V, E)$ which is a **model instance** of a **meta model** $(C, R)$ with a relation $r = ((c, P), (d, Q), l, Q) \in R$. We call $r_A : (R_A, \textbf{\textit{RelInstRestr}}(V, E, r))$ an **encoded relationship** of $r$ if $r_A = (l + id(c) + id(d), \{\ r_I \mid r_I \in \textbf{\textit{RelInstRestr}}(V, E, r)\})$*

We can now construct the formal encoding for any given meta model $(C, R)$ with a corresponding model instance $(V, E)$. For every $r \in R$ we can construct an **encoded relationship** and take a set of all of them.

**Definition 4.5.** *Suppose a graph $(V, E)$ which is a **model instance** of a **meta model** $(C, R)$. We define an **encoding** of $(C, R)$ and $(V, E)$ to be $\{\ r_A \mid r \in R \wedge r_A$ is an **encoded relationship** of $r\}$*

**Example**

Let's apply the abovementioned definitions to our small NRP example. Recall that we have a small NRP meta model $(C, R)$ and a small NRP model instance $(V, E)$ with:

$$
\begin{aligned}
C &= \{(Solution, S), (SoftwareArtifact, SA)\} \\
R &= \{((Solution, S), (SoftwareArtifact, SA), selectedArtifacts, SEL), \\
&\quad ((SoftwareArtifact, SA), (Solution, S), solutions, SOL), \\
&\quad ((SoftwareArtifact, SA), (SoftwareArtifact, SA), requires, REQ)\} \\
V &= \{proposedSolution, mainArtifact, drawArtifact, logArtifact\} \\
E &= \{(proposedSolution, mainArtifact), (mainArtifact, proposedSolution), \\
&\quad (mainArtifact, drawArtifact), (mainArtifact, logArtifact)\}
\end{aligned}
$$

First, we look at what a **relation instance** looks like for some vertex. This small example only has two relation instances, as there are only two classes that have outgoing relations. We show them for mainArtifact and proposedSolution:

$$
\begin{aligned}
r_{I\,mainArtifact} &= (id(mainArtifact), \{id(proposedSolution), id(drawArtifact), id(logArtifact)\}) \\
r_{I\,proposedSolution} &= (id(proposedSolution), \{id(mainArtifact)\})
\end{aligned}
$$

Taking both $r_{I\,mainArtifact}$ and $r_{I\,proposedSolution}$ as a set now already is $\mathbf{RelInst}(V, E)$. We can now look at what all restricted relations look like for $(V, E)$, but we will focus only on those of mainArtifact as here we can restrict to different abstract relations.

We can either restrict to $r_0 = ((SoftwareArtifact, SA), (Solution, S), solutions, SOL)$ which gives:

$$\mathbf{RelInstRestr}(V, E, r_0) = \{(id(mainArtifact), \{id(proposedSolution)\})\}$$

Or we can restrict to $r_1 = ((SoftwareArtifact, SA), (SoftwareArtifact, SA), requires, REQ)$, which gives:

$$\mathbf{RelInstRestr}(V, E, r_1) = \{(id(mainArtifact), \{id(drawArtifact), id(logArtifact)\})\}$$

This enables us to encode one of these restricted relationship instances. Let's pick $r_0$, we have then that $r_A$ is an **encoded relationship** for $r_0$ if:

$$r_A = (solutions{+}id(SoftwareArtifact, SA){+}id(Solution, S), \{(id(mainArtifact), \{id(proposedSolution)\})\})$$

The entire **encoding** of $(C, R)$ and $(V, E)$ can now be instantiated leading to the following set:

$$
\begin{aligned}
&\{(solutions + id(SoftwareArtifact, SA) + id(Solution, S), \{(id(mainArtifact), \{id(proposedSolution)\})\}), \\
&\ (requires + id(SoftwareArtifact, SA) + id(SoftwareArtifact, SA), \\
&\qquad\qquad \{(id(mainArtifact), \{id(drawArtifact), id(logArtifact)\})\}), \\
&\ (selectedArtifacts + id(Solution, S) + id(SoftwareArtifact, SA), \{(id(proposedSolution), \{id(mainArtifact)\})\})\}
\end{aligned}
$$

### 4.4.2 Application within Model Driven Optimisation

Using the above definitions, we have encoded a meta model with its model instance. However, this is not yet enough to actually use the encoding within any given MDO technique. To enable this, we have to consider two things. One, how do we evolve the encoding instead of the model? Two, how do we evaluate the encoding with the fitness function? Both of these questions concern aspects of the process that the user inputs. The fitness function and the rules for evolving the models. We do not want a user of an MDO technique to consider the encoding. That would defeat the entire purpose of using models as a way of representing the population. The goal of this thesis is to make sure the user of such an MDO technique does not even realise it is being encoded.

One way to enable the seamless integration of the encoding is to intercept relevant function calls to the model and supply the information based on the encoding. To do this it is important that we add an extra ingredient to the encoding. A bidirectional map of the $\mathbf{id} : V \to L$ and $\mathbf{obj} : L \to V$. We call this bidirectional map the 'Repository', and it implements the storage of all relevant model instances. To use the Repository we fill it with the initial objects in the model instance during the encoding phase. Then, whenever an exploration operator adds an object instance to the model, we instantiate the object, generate an identifier for it, and add it to the Repository. If an exploration operator removes an object from the model, the Repository should remove the relevant mapping to save space.

With the addition of this Repository it is now trivial to intercept functions on the model. From the context of the function we can retrieve the relevant abstract relationship. We can then, based on the object on which it is called, retrieve the relevant identifiers it is related to within the encoding, and return the list of actual objects back to caller. We will illustrate this idea using the small NRP model example. Recall figure 1 and 4. A logical function every Solution has is to get its selected artifacts. The context gives us the identifier for this encoded relationship. Namely, $selectedArtifacts + id(Solution, S) + id(SoftwareArtifact, SA)$. With this we can then obtain all relationship instances for this abstract relationship. Now suppose we call this function on the proposedSolution. We can then invoke $id(proposedSolution)$ to lookup the relevant set of identifiers that this Solution references. We then simply return a new set that contains the actual object instances for the set of identifiers.

This way both the fitness functions and exploration operators can still treat the population as models, while they are actually using the encoding. Ideally, the generation of this interception should be done automatically. This was out of scope for the thesis and will thus be left as future work.

### 4.4.3 Java Library

We implemented all of the aforementioned concepts as a Java library for the EMF framework. The library currently only supports the automatic generation of the encoding and the automatic lookup of related objects. Within the library there are three important classes that facilitate all the functionality for the encoding to be applied in any situation. These are the classes Converter, Encoding, and Repository, as we can see in the class diagram in figure 6. The ModelLoader class is utility for loading in the files.

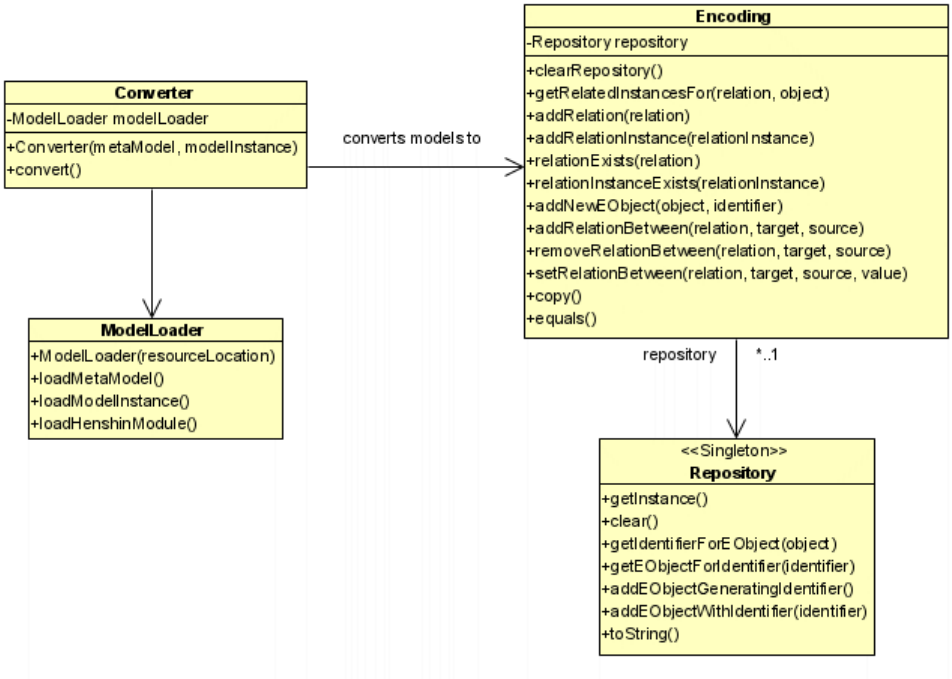


Figure 6: Encoding Java library class model

The converter exposes an interface that accepts any meta model ecore file and a model instance xmi file and converts it to an encoding. The Repository class is implemented as described in the previous section, it contains the bidirectional map between object instances and identifiers. It supplies functionality for the *id* and *obj* functions. The Encoding class itself contains as data a HashMap that has as keys all of the abstract relationships within the meta model and as values another HashMap which contains the **RelInstRestr** for that abstract relationship.

A user of this library should only ever have to work with the Converter.convert() function and the lookup function from the Encoding class. This gives a two layered approach where the model exists within the code to guide the user to not construct violations of the models within their implementation. The encoding should now be used to do the heavy lifting of the optimisation algorithm, namely the evolution and copying.

Within the library we also support the addition of new objects to the repository as described within the previous section. However, we do not support the removal of objects from the repository, as efficient garbage collection is a big challenge in itself. Therefore, we left it out of the scope of this thesis.

The interception of the calls on the models is not supported by the Java library. There are several options to enable this, one of them is a wrapper that is generated with the generation of the actual code for the models by EMF. The wrapper then functions as the bridge between the model and the encoding and should as such be context aware, which is possible because it is generated based

on the model. Another option is to intercept the calls to the models using AspectJ. This enables a user to redirect a call to the model to the encoding using Aspect Oriented Programming (AOP). We will not describe AspectJ in detail within this thesis. Instead, we direct readers to [17] as a reference to get started with AOP and AspectJ. Since AspectJ does not require us to interfere with the code generation of EMF and gives us relative easy access to the interfaces we need this is what we chose to implement the interception of the calls within the experiments we performed.

# 5 Evaluation methodology

To validate the concepts we introduced in section 4 we use the case studies described earlier. We'll apply one aspect of the concept to each case study to validate the concepts in a step-by-step manner. First we apply the initial concept as mentioned within 4.1 to the NRP problem, as this is a relatively simple problem with only one edge that is changed in the model. Then we apply the entirety of the concept in the CRA case study. In both cases we measured the time for completion of the optimisation based on the size of the population and the amount of evaluations. We also use boxplots (for further details refer to [9]) to represent the data obtained from the experiments in both NRP and CRA. These boxplots show the average of the data, where the majority of the data points lie, and where the outliers lie. Lastly, we apply t testing on the fitness results in both experiments to determine that there is no statistical difference between the experiments that use the encoding and the experiments that use the standard models [6]. As critical value we take the standard 5% that is used by most statisticians. In all cases we want to not reject the null hypothesis, as we want there to be no statistical significance between the experiments that use the encoding and the experiments that do not. For further reading we refer to [25] and [6]

For NRP we kept the setup simple. We only have two variants of the experiment, one with the crucial edge encoded according to the initial concept described in section 4.1, and one that just uses the model. The way that the experiments now obtain the results is significantly different as the exploration operators had to be adapted. To make sure that the results from these experiment only differ within the representation of the data we checked whether the results from both runs are comparative. We ran the algorithm for both variants on a large population for a long time. Combining the Pareto fronts of these results and using this to calculate the hyper volume for each subsequent run of the algorithm. On these values we then apply t testing to see whether it is reasonable to assume that these results came from the same methods. For the data we executed several experiments with varying population size and amount of evaluations to see whether these variables would change the outcome. We executed them for each variant, one with the use of the encoding and one without. During these experiments we only kept track of the final fitness, the overall time consumption, and the hypervolumes of each population. The point of the experiments is to enable answering our first research question (RQ1). Since the setup is so similar, we can fully focus on whether using an encoding improves the performance or not.

With CRA we took a different approach, as we wanted to now answer the second research question (RQ2). We turned the formal encoding into a Java library so that we can automatically generate an encoding from any given meta model and model instance. This allowed us to apply it to any given case study using MDEOptimiser. We chose CRA because this case study has five model instances, each with a different size ranging from small to large. This enables us to see whether the encoding works for different sizes of model instances.

In this case study we chose to use the mutation operators given by [4]. That way we don't have to deal with crossover operators that can have multiple parents. This makes implementing and running the encoding for CRA easier as we only had to change some edges and add some objects. It essentially doesn't really matter what type of exploration operators are used, because we are only interested in whether using the encoding this way is beneficial with respect to the performance of the algorithm.

For the measuring of CRA we took the same style of approach of doing several experiments with the encoding as representation and several with the model as representation. This time however, we used the Java library we created as the encoding. We again measured the time it took for the algorithm to run for different sized model instances with varying population size and amount of evaluations. However, this time we also measured the relevant inner steps within the algorithm to compare them. This is relevant now, as we want to look at whether using the encoding in this way already increases the overall performance. It enables us to more deeply analyze where the usage of the encoding is faster, and where it might be slower. We measured the following steps individually:

- copying a member of the population
- applying the pattern matching to a model/encoding
- applying the mutation to a member of the population
- evaluating the fitness of a member of the population

The full mutation of a member of the population is comprised from the application of the pattern matching of the Henshin rule, and applying the corresponding rule to a member of the population.

Additionally, we also kept track of the fitness of members of the population for both variants. We again used t testing to verify that the best fitness found for each experiment come from the same underlying algorithm. This time by applying the test to all of the best fitnesses found for each experiment with a model instance.

# 6 Implementation

This section contains the implementation details of the experiments done for NRP and CRA. The results that are produced by the experiments are saved to a file, which is later read by a Python script that turns the data into comprehensible graphs.

## 6.1 Next Release Problem

As mentioned in the methodology we first implemented the idea for the encoding in the context of the Next Release Problem. This means we implemented the encoding as a vector of bits.

Recall the meta model for NRP in figure 1, in this figure we can observe that the crucial relation for this problem lies between Solution and SoftwareArtifact. The core of the NRP is just simply adding a software artifact to the solution or not, meaning if we encode this relation as a bit vector, we can simply evolve this vector instead of the whole model. The rest of this model will always stay the same for each member of the population.

We use the Java implementation of the NRP provided in [4], on which we implemented the MOEA framework. As we need to evaluate two variants we implemented two problems that represent them. The encoded problem was quite straightforward as we could simply use the provided BinaryVariable given by MOEA. For the model problem however, we had to implement our own custom variable.

For simplicity we have chosen to ignore the exploration operators provided in [4], and use the standard operators for evolving a bit vector provided by the MOEA framework, namely the HUX and BitFlip operators which will be introduced shortly. The point of this experiment is to check whether using an encoding like this gives a performance boost when everything is the same but the way the data is stored.

**The HUX operator**, also called the half uniform crossover operator, is an operator which accepts two parents and then compares their vectors. For parents $p$ and $q$ with vectors $p_1, \cdots, p_n$ and $q_1, \cdots, q_n$ we have that if $p_i \neq q_i$ then their values are swapped. In a usual case there is only a chance that this swap happens, but in the case of this experiment this chance is 100%.

**The BitFlip operator** accepts one parent and has for each bit a 1% chance to flip its value to the opposite.

In this experiment we apply the HUX operator first and then apply the BitFlip on the resulting children. We let the MOEA framework handle the operators for the encoded variant of the problem, and we implement these operators for the model variant ourselves by instantiating a Variation class. This implementation should behave exactly the same as the standard implementations from MOEA.

We use two model instances for this experiment, a smaller one with 5 customers, 25 requirements, and 63 software artifacts. And a bigger one with 25 customers, 50 requirements, and 203 software artifacts. For each model instance we run the experiment on two variables; namely, the population size and amount of evaluations. Per combination of these variables we run the experiment several times.

To intercept the calls to the model we use AspectJ to redirect them to obtain the information from our bit vector. This is quite simple to implement as the only calls that we have to intercept are

the getters in the SoftwareArtifact and Solution classes that reference each other.

## 6.2 Class Responsibility Assignment

For the implementation of CRA we used the Java implementation provided by [4]. As stated in the methodology, we use the Java library we created as the representation for our encoding. To ensure the encoding is used instead of the model we used AspectJ again to intercept the calls to the model and redirect them to the encoding. This time however, the interception is more difficult. As we needed to implement it for every getter within the CRA model. Performing this task is quite tedious as the encoding guides the implementation and little effort is required to create the advices. Ideally, this part should be automated, which we will not do in this thesis. We will leave the automation as future work.

We wanted to stay as close as possible to the implementation MDEOptimiser uses. We copied the way MDEOptimiser calls MOEA, and then filled in our custom AbstractProblem, Variable and Variation that are to be used during the optimisation algorithm. This time we did use the exploration operators as provided by [4]. Like MDEOptimiser, we rely on Henshin for the pattern matching of the rule to the model. To enable this we implemented a stub for the graph that Henshin expects that is filled with the information of the encoding. We then tell Henshin to pattern match to that graph, which calls the relevant getters in the model. Henshin then gives us back a list of changes that should happen to the model, which we apply to the encoding instead.

In order to measure the performance we recorded the time it took for each of the actions as described in the methodology to complete as well as the overall time. Since these steps get executed multiple times throughout the algorithm we simply take the average of each of these times per experiment. To measure it for MDEOptimiser we add measurements as hard coded pieces within the corresponding classes in MDEOptimiser.

We use five model instances within this experiment that are also provided by [4]. These model instances range from only 9 features with 14 dependencies to 160 features and 600 dependencies.

# 7 Results

In this section we will show and discuss the results gotten from the experiments we conducted. Firstly, we will discuss the results for the NRP case study. Then we will move on to the results for the CRA case study.

## 7.1 Next Release Problem

For the experiments using the Next Release Problem we tried to get an answer to question 'Can we use an encoding instead of a model to copy a population of a model driven optimisation problem faster?'. As mentioned in the methodology we kept the setup simple such that the experiments only differ in the representation of the data. We also wanted to see whether the population size or the amount of evaluations had an impact on the performance. As mentioned in the methodology we run the experiments with the population size and the amount of evaluations as variables. We define a start value, an end value and an increment value for the experiment and then run it multiple times for each possible combination. The value we record per experiment is the amount of time it takes for the algorithm to complete, and its hypervolume with respect to the previously computed reference pareto front.

Within NRP we performed the experiment on a model instance with 5 customers, 25 requirements and 63 software artifacts. We executed this experiment with a start value for the population size of 500, an end value of 2000, and an increment of 500. The amount of evaluations started at 10000, ends at 30000 and has an increment of 10000. From running this experiment it was clear that the population size of an optimisation algorithm does not nearly impact the performance of it as much as the amount of evaluation does. We illustrate this in figure 7 and figure 8. In figure 7 we take 10000 evaluations statically and vary between all four possible population sizes. Whereas in figure 8 we take a static population size of 500 and vary between the three amounts of evaluations. In the figures the 'enc' and 'mod' prefixes denote whether it was an experiment with the encoding or the model as data representation. The p and e suffixes denote the population size and amount of evaluations.

Like we mentioned within the methodology, we use boxplots to display the data. As a short reminder the box contains all values between the 25th and 75th percentile, where the orange line is the median, the lines outside the box contain the rest of the values, and the circles are the outliers. For a more in depth explanation of boxplots refer to [9]. A boxplot that represents data for the encoding will always have a light blue background, while a boxplot that represent data for MDEOptimiser will have a beige background. Additionally, except for figure 7 and 8 where the encoding results are on the left and the MDEOptimiser results are on the right, the results in each figure alternate starting with the encoding on the left.

Since the population size does not have as much impact on the time taken as the amount of evaluations, we combine the values of the experiments to now only vary over the amount of evaluations for the following figures. In the raw data the varying population size is still visible.

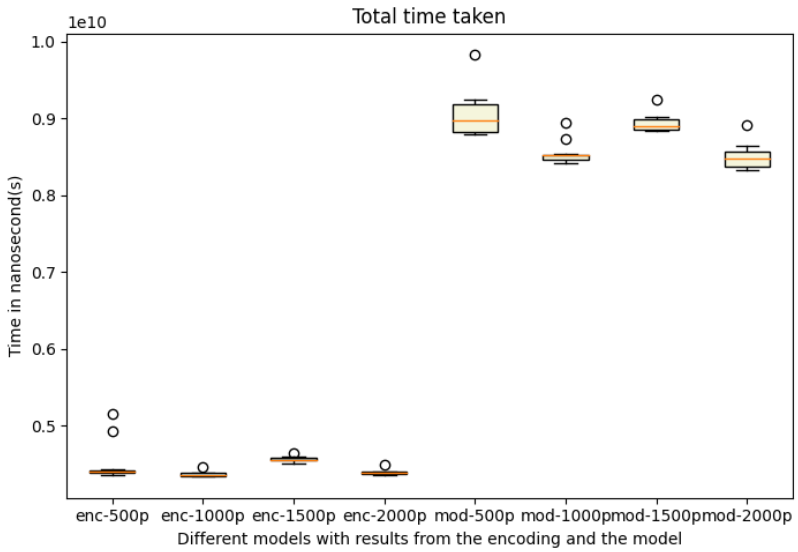


Figure 7: NRP with 5 customers, 25 requirements, and 63 software artifacts varying on the population size

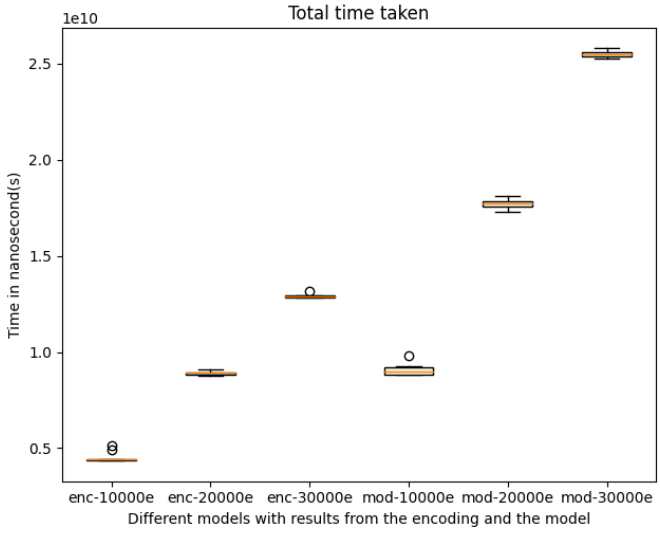


Figure 8: NRP with 5 customers, 25 requirements, and 63 software artifacts varying on the amount of evaluations

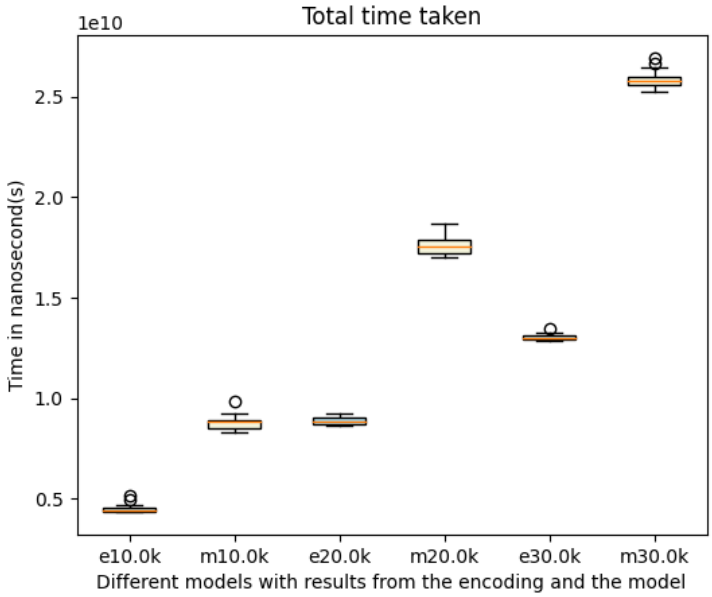


Figure 9: NRP with 5 customers, 25 requirements, and 63 software artifacts with combined population size values

Figure 9 shows all the values we obtained for the experiment on this model instance. Alternating between the encoding variant and model variant on the x-axis, with the numbers now representing the amount of evaluations (i.e. $10.0k$ mean ten thousand evaluations). As we can see the encoding variant greatly outclasses the model variant. In this experiment we observed an average speedup of 1.98 with a minimum speedup of 1.79 and a maximum speedup of 2.09.

Next, we ran two experiments on a bigger model instance with 25 customers, 50 requirements, and 203 software artifacts. Figures 10 and 11 show their results. For figure 11 we executed this experiment with a start value for the population size of 200, an end value of 1000 and an increment of 200. The amount of evaluations started at 1000, ends at 10000 and has an increment of 1000. Since we combined the results for the population size they are not visible within either figure. For figure 10 we executed this experiment with a static value for the population size of 1000. The amount of evaluations started at 10000, ends at 50000 and has an increment of 10000.

In these figures we can again see that the model variant of the experiment is significantly slower than the encoding variant. We can even draw a line through each variant in the figures and observe them separating linearly. For the experiment corresponding to figure 10 we observed an average speedup of 1.96 with a minimum speedup of 1.79 and a maximum speedup of 2.04. Furthermore, for the experiment corresponding to figure 11 we observed an average speedup of 1.93 with a minimum speedup of 1.41 and a maximum speedup of 2.27.

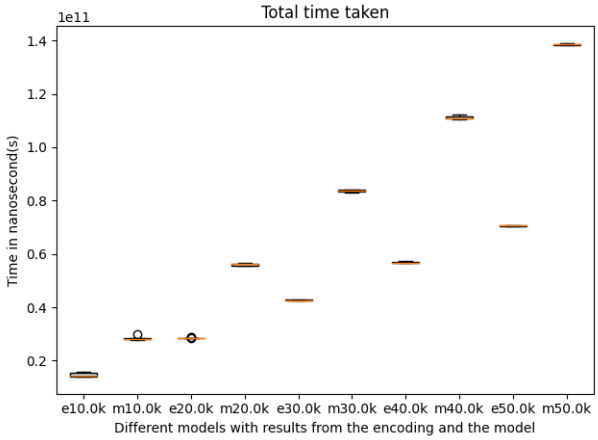


Figure 10: NRP with 25 customers, 50 requirements, and 203 software artifacts varying on the amount of evaluations

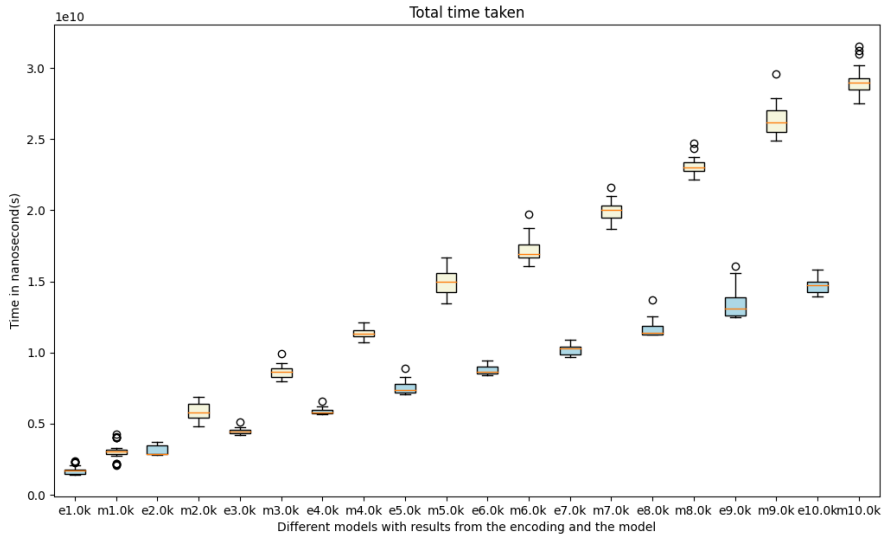


Figure 11: NRP with 25 customers, 50 requirements, and 203 software artifacts varying on the amount of evaluations

For all three experiments we calculated the hypervolumes for each variant with respect to a previously calculated pareto front of the model instance. This pareto front is the combination of the result of running both experiments with a large population size and amount of evaluations. We then applied t testing with as null hypothesis that both samples have identical average (expected) values. The p-value for all three experiments was not smaller than the 5% that we use to disprove the null hypothesis. In fact, the p-values were significantly larger than 5% in each experiment. This means that we can conclude that the values obtained during the experiments where obtained from the same algorithm.

## 7.2 Class Responsibility Assignment

With the experiments that use the Class Responsibility Assignment case study we tried to find an answer to the second question: 'Does using an encoding reduce the overall execution time of the model driven optimisation algorithm?'. Within the experiments for CRA we do not vary over the population size and the amount of evaluations, as we established in the NRP experiments that the speedup does not change with them. We instead take vary over the five different model instances supplied by the case study which range from 9 features with 14 dependencies to 160 features and 600 dependencies. For the population size we have chosen 40 with 500 evaluations per member of the population (i.e. 20000 total evaluations). Like we mentioned in the methodology, we measure the total time for the experiments to complete and the average time per experiment it took to complete: copying a member, applying the mutation, evaluating a member, and applying the pattern matching. For all of the figures within this section the x-axis will contain boxplots per model instance for each variant of the experiments. As an example with 'A-Encoding' we mean the experiments that use the encoding on the model instance A.

Initially, this looks to be quite promising for the encoding. The figures 12 and 13 show that the encoding experiments completed on average faster for the model instances A and B. For model instance C the encoding experiments where on average as fast as MDEOptimiser, but had greater variance among the values. However, as soon as we look at the big model instances D and E we see the opposite picture. The encoding is no longer faster than MDEOptimiser and for instance E it is significantly slower. To further understand a possible cause for these results we measured the time it took an experiment to complete several steps.

We started with looking at the average time it took to copy members of the population. The results for these experiments are shown in figures 14 and 15. From these results we can observe that, like in the NRP case study, the encoding beats out MDEOptimiser for every model instance. Having an average speedup of 2.36 for the biggest model instance E. This means that we have to look at the other steps within the algorithm that interact with the encoding.

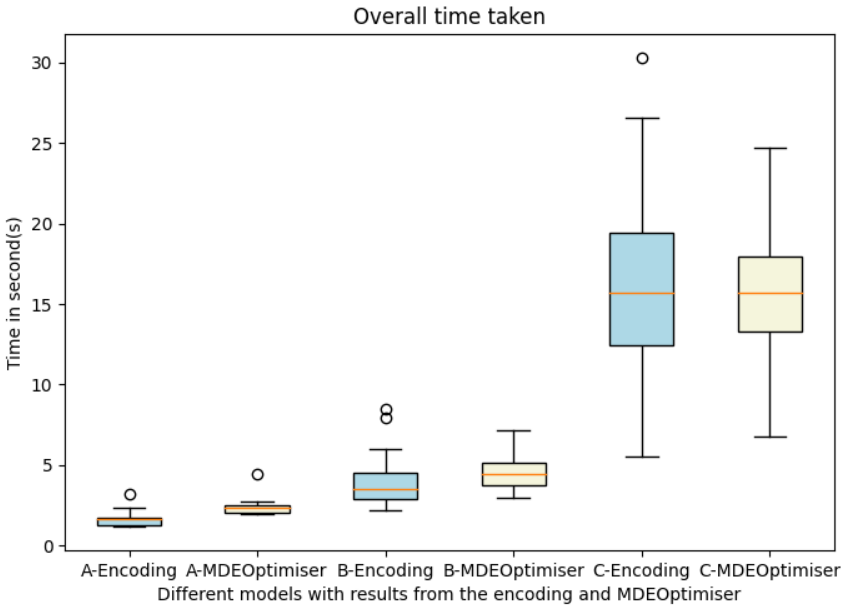


Figure 12: CRA overall time until completion for model instances A, B, and C

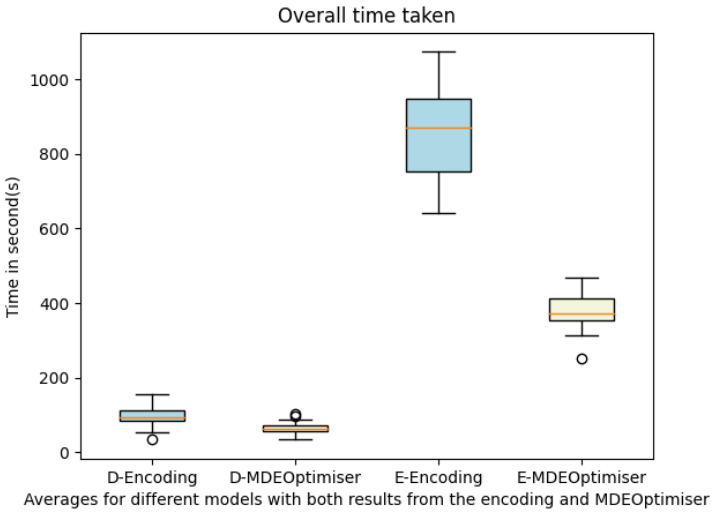


Figure 13: CRA overall time until completion for model instances D and E

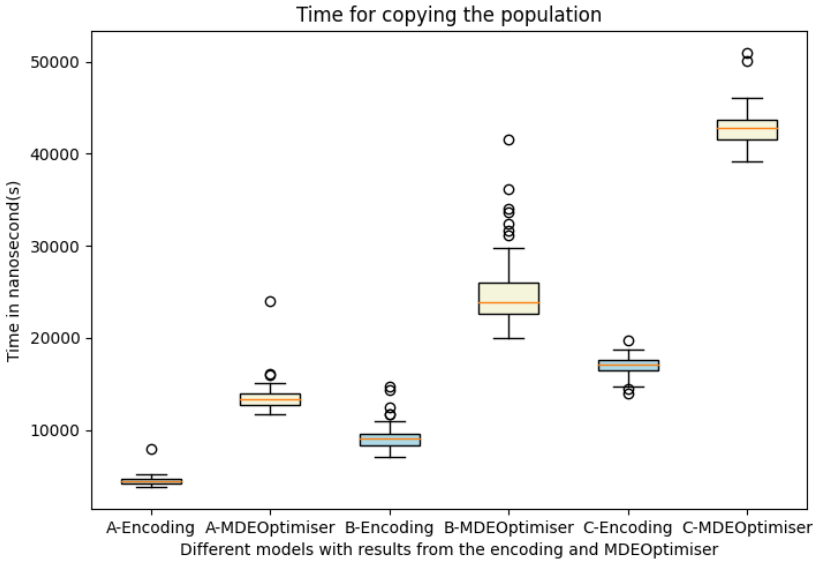


Figure 14: CRA average time to copy a member of the population for model instances A, B, and C

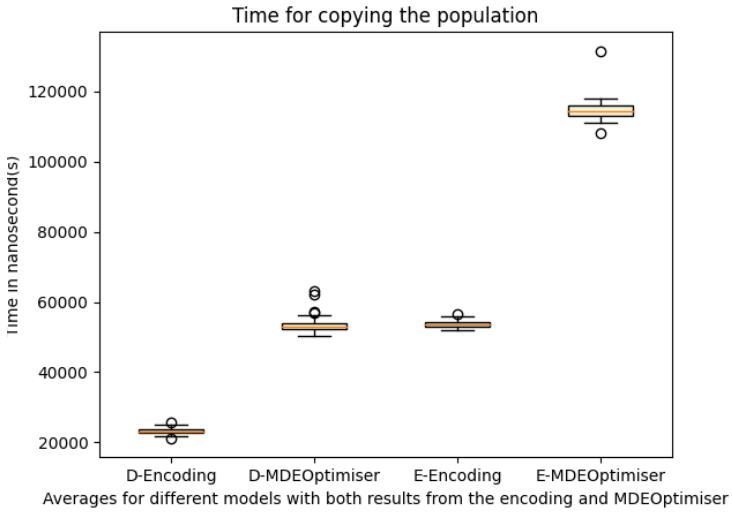


Figure 15: CRA average time to copy a member of the population for model instances D and E

Let's first look at the average time it takes for the experiments to mutate a member of the population. These results are shown in figures 16 and 17. What we can observe from these results is that the encoding performs worse when applying the found match. However, the time we lose during this step lies in the thousands of nanoseconds. Overall we should still be faster considering that during the copying step the encoding is tens of thousands of nanoseconds faster than MDEOptimiser and both steps happen the same amount of times.

We move on to the evaluation of the members of the population. The results for this step are visible in figures 18 and 19. These results immediately show that the encoding is slower when evaluating a member of the population. This slowdown becomes even more noticeable when the size of the model increases, which is the inverse of what we aimed to achieve. The difference in results here is also not just a small amount of time like with the mutation. It is more than tens of thousands of nanoseconds slower than MDEOptimiser.

The same observation can be made for the results of the matching step within the algorithm. These results are the time it took for a Henshin rule to be pattern matched to a model and are shown in the figures 20 and 21. The difference here is the most extreme. We observed on average a more than 10 million nanoseconds difference in model instance E between the encoding and MDEOptimiser. This matching is done during the algorithm as often as the copying of a member. When a match isn't found for a rule this matching is repeated up to 3 times. These slowdowns add up and they are what is reflected within the overall results.

For all five model instances we applied t testing to calculate the p-value between the best fitness values of the encoding and MDEOptimiser. As null hypothesis we had that both samples have identical average (expected) values. We found that for the model instances of which we had a lot of data points (A, B, and C) the p-values where higher than those of the ones for which we had less data points (D and E). However, all p-values were significantly larger than the 5% that we use to disprove the null hypothesis. From this we can conclude that the values obtained during the experiments came from the same algorithm.

In conclusion, the evaluation, matching, and mutation of members of the population is slower in the experiments that implement the encoding than in MDEOptimiser itself. Whereas the copying of the members of the population is still faster when using the encoding. We will further discuss these results in the next section.

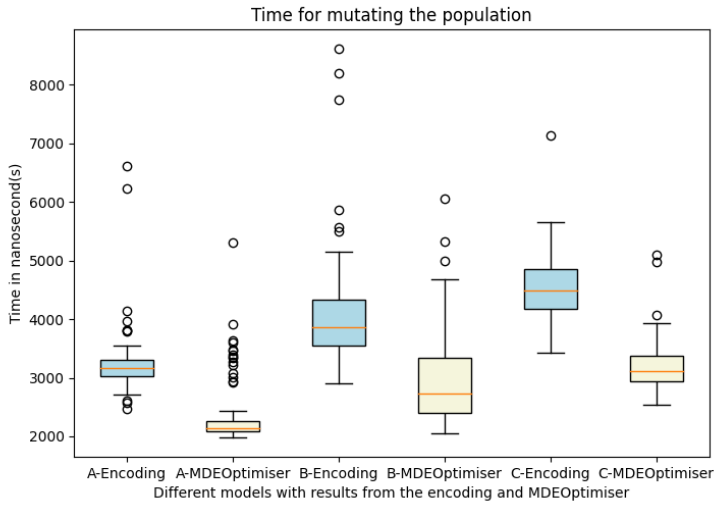


Figure 16: CRA average time to mutate a member of the population for model instances A, B, and C

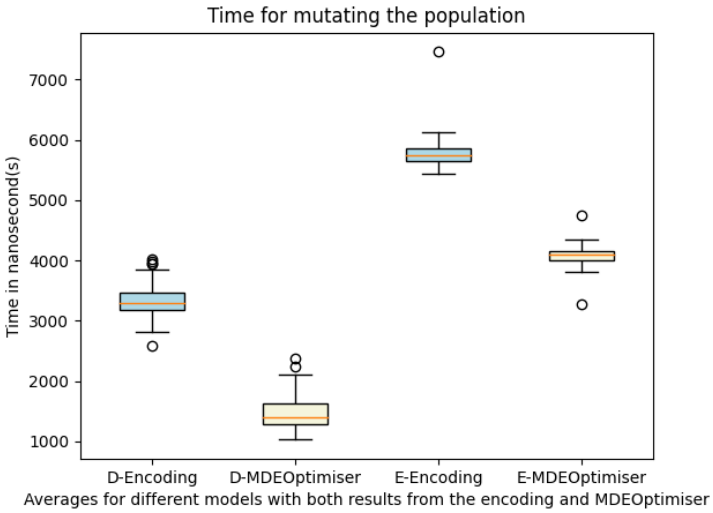


Figure 17: CRA average time to mutate a member of the population for model instances D and E

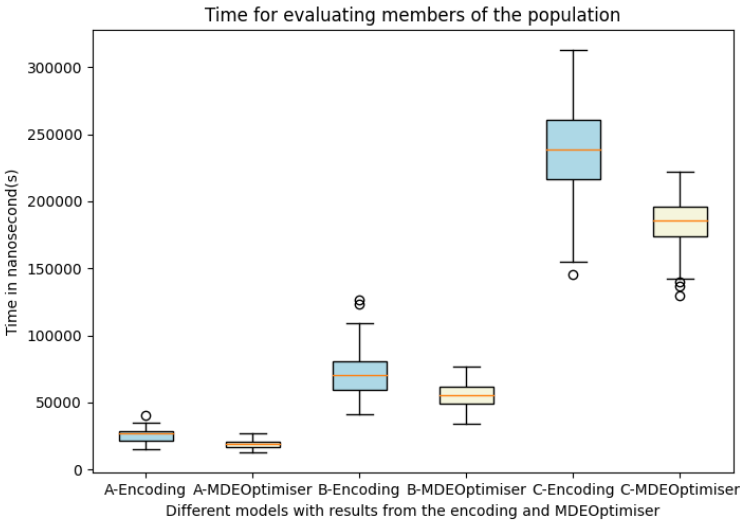


Figure 18: CRA average time to evaluate a member of the population for model instances A, B, and C

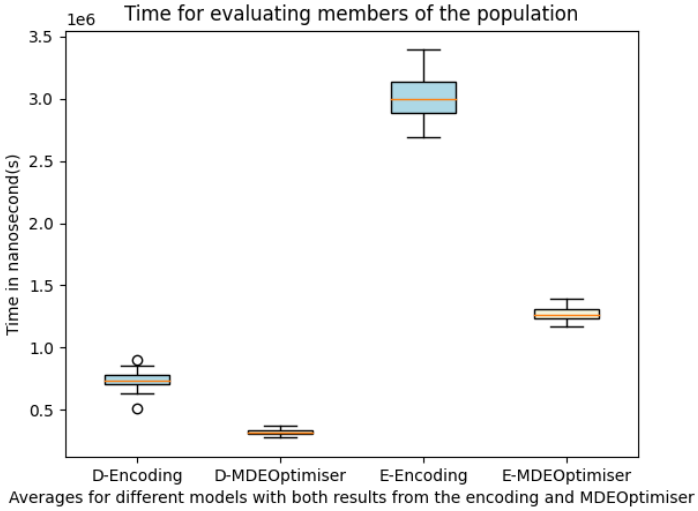


Figure 19: CRA average time to evaluate a member of the population for model instances D and E

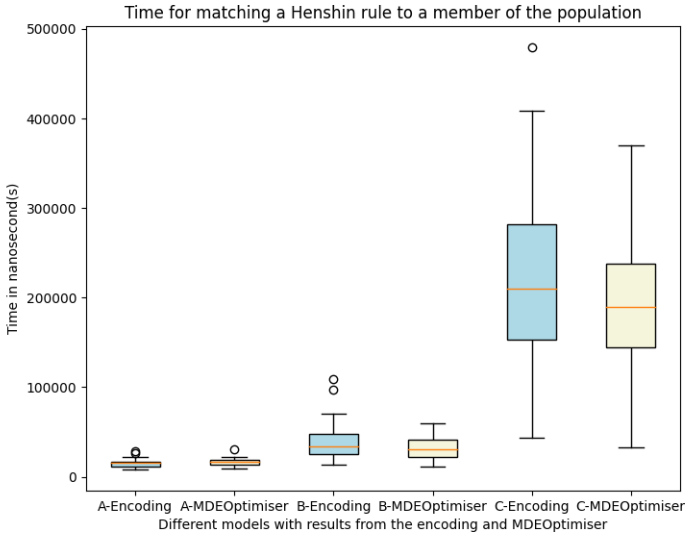


Figure 20: CRA average time to match a member of the population to a Henshin rule for model instances A, B, and C

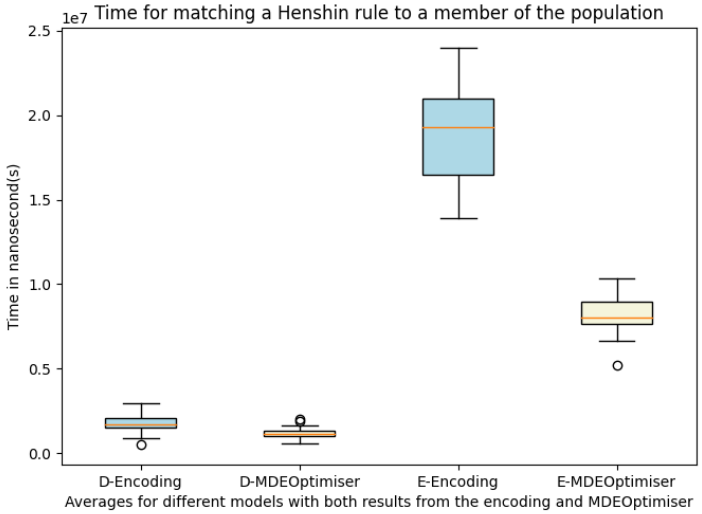


Figure 21: CRA average time to match a member of the population to a Henshin rule for model instances D and E

# 8 Discussion

This section will contain a discussion on the evaluated results of the experiments performed in this thesis.

In the case of the NRP, we see that the results show that in both cases of the model instance using an encoding as the representation is a lot quicker than using the model. We have however only measured the total execution time, which means that at this stage it is too early to say that this is definitely due to the encoding, and not just because we use a different Variation for each variant. In hindsight, it would have been better to also measure the individual execution times for both the evolution of a member of the population, and the copying of a member of the population. However, since we have controlled this experiment very thoroughly such that there are little other variables at play, we can at least say with some certainty that it makes the algorithm faster. This is made fully certain when we also take the results for the CRA case into account. In this case we see that the usage of the encoding also significantly improves the performance for the copying of the members of the population.

Taking these results into account, we can answer RQ1, 'Can we use an encoding instead of a model to copy a population of a model driven optimisation problem faster?'. The answer to this question is 'yes', using an encoding as the basis for the population does indeed speed up the copying of said population. We can conclude this from the results of the NRP experiments in the figures 10, 11, and 9, and this is further confirmed by the results shown in the CRA figures 14 and 15.

However, the same cannot be said for RQ2. When looking at all other times, we see that none of them are actually faster than the model based approach. The worst offender in the CRA case is the pattern matching that is done by Henshin. Now there can be several reasons for why this is the case. One reasonable explanation is that since we use AspectJ to intercept all calls to the model, the overhead that is created due to this makes it impossible to have faster or even comparable times. This makes the most amount of sense in our opinion, AspectJ is known to increase overhead somewhat [5], but that is usually deemed negligible due to the amount of calls typical Java programs have to libraries, which dwarfs the amount of time spent in user code. In our case however, we spent a lot of time in the user code, as we capture any call towards the domain model. It is also the case that a simple getter that returns part of the model is faster than having to gather the right objects from the object repository, as this list needs to be constructed every time. When combining this with the fact that these getters are called a lot, we can easily see how just a slight decrease in performance can lead to seconds or even minutes when running an entire optimisation algorithm. Another explanation could be that using an encoding this way and intercepting calls to the models just isn't viable as it costs more in performance than it gains.

One thing that we also saw during the implementation of the encoding is that if we have the underlying data structure be a bit vector lookups through this vector would take at worst $n$ amount of cycles where $n$ is the amount of objects within a class. This was because in the beginning we did not take into account that Henshin rules can add and remove objects from the model. To alleviate this issue we adapted the encoding to use a set of identifiers instead, which cut the amount of cycles to only those object that are actually related to the object.

The ideas we represent within this thesis can be extrapolated to tools outside of MDEOptimiser. We think that the core concept, the formal encoding, is a useful way of encoding any meta model with model instances. Which can solve, as we show, the performance of copying the population.

The main problem that we see now is that MDEOptimiser and the surrouding tools are not tailored towards using such an encoding. We thought that using AspectJ would not impact the performance as much as it did. However, we can not yet be certain that it is the fault of AspectJ that the performance is worse. It could just be the case that this way of encoding the models only benefits the copying of the population. More research is needed to figure whether we can get MDEOptimiser to a place where the overall performance of the encoded variant is as fast or faster than the model variant.

Another big issue is the usability of the encoding. In an ideal setting we do not want the end user that describes the domain and exploration operators to perfectly tune their models for the encoding. This is to preserve the advantages that a tool like MDEOptimiser introduces. In the CRA experiment we enabled this with the use of AspectJ, as that enabled us to redirect calls to the model to the encoding. This usability issue lies directly in contrast to the next steps to actually integrate the encoding. For that, we want the code to ideally work directly with the encoding when we're talking about the fitness functions, mutations, and pattern matching.

# 9 Conclusion

From these experiments we can draw several conclusions. The first conclusion we draw is that using the encoding within MDEOptimiser does significantly improve the performance of the copying of members of the population. We back this up by the results from both case studies wher we show that the copying is faster. This means we can answer RQ1 with yes.

However, as we see from the results from the CRA experiments, the way we implemented the encoding within MDEOptimiser using AspectJ does not increase the overall performance. This means that we answer RQ2 with no, the current implementation to integrate the encoding in MDEOptimiser does not improve the performance. Within the next section we describe possible future work using a different way of intercepting the calls to the model which might increase the overall performance.

In conclusion, we contribute a formal approach to obtaining an encoding from an arbitrary boxes and lines model. Along with an implementation of this encoding for the EMF framework. We show that this encoding is faster at copying the population over to the next iteration of the optimisation algorithm. The drawback that we also show is that using AspectJ as bridge between the model and the encoding severely decreases the performance of the other steps within MDEOptimiser. Lastly, we discuss possible causes for this, and suggest new angles to research this topic further. Because we do believe that the problems as described in this thesis could be alleviated and the encoding can further increase the performance without sacrificing any of the benefits.

# 10 Future work

This thesis does not contain all of the appropriate answers for the questions proposed, and even introduces new questions that should be looked into. As future work we are of the opinion that the most important aspect to solve is how to apply the Henshin operators, and thus the matching, directly on the encoding, and not intercepting everything with AspectJ. When applying this

improvement on the CRA case we expect that the overhead for the matching will be minimized, which should give equal or better performance than MDEOptimiser.

Another interesting aspect to explore is related to the overhead AspectJ creates. It could be worth altering the way the model code is generated from the models the user creates, if this takes the encoding into account, it is possible to diminish the negative impact of AspectJ even more and maybe even completely stop using it to intercept calls to the model. This should be done carefully as we do not want to lose the benefit of using models to describe the optimisation problem. Additionally, we think it is important that the interception of the calls to the model is generated automatically. This enables easier application of the concepts to new case studies.

As mentioned earlier we do not take into account that a class within the model can have its attributes altered by the exploration operators. For completeness it is preferable that the encoding does also accept such changes. The current way how the Repository handles the actual object instances is not suited for this. Several encodings usually use some shared instances of objects, what happens then if for one encoding an attribute changes? If we copy this object and alter it only for one encoding we again introduce the problem that this thesis aims to solve. This seems to be quite a complex issue that might be worth exploring. One could think of a possible solution where the values of an object instance are also somehow encoded. As these values will always be some combination of basic values like Integers, Strings and Lists.

Lastly it is useful to apply the encoding as introduced in this thesis to more different problems, as we have now only applied it to the CRA problem. Applying it to the other case studies mentioned in [4] will give a better idea on how to further improve the encoding itself, and possibly how to deal with the matching and application of the exploration operators and how to deal with the interception of calls to the model.

# References


[1] Hani Abdeen et al. "Multi-objective optimization in rule-based design space exploration". In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 2014, pp. 289–300.

[2] Thorsten Arendt et al. "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations". In: *Model Driven Engineering Languages and Systems*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 121–135. ISBN: 978-3-642-16145-2.

[3] Jean Bézivin. "Model Driven Engineering: An Emerging Technical Space". In: *Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*. Ed. by Ralf Lämmel, João Saraiva, and Joost Visser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 36–64. ISBN: 978-3-540-46235-4. DOI: 10.1007/11877028_2. URL: https://doi.org/10.1007/11877028_2.

[4] Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. "MDEoptimiser: A Search Based Model Engineering Tool". In: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '18. Copenhagen, Denmark: Association for Computing Machinery, 2018, pp. 12–16. ISBN: 9781450359658. DOI: 10.1145/3270112.3270130. URL: https://doi-org.ru.idm.oclc.org/10.1145/3270112.3270130.

[5] Bruno Dufour et al. "Measuring the Dynamic Behaviour of AspectJ Programs". In: *SIGPLAN Not.* 39.10 (Oct. 2004), pp. 150–169. ISSN: 0362-1340. DOI: 10.1145/1035292.1028990. URL: https://doi.org/10.1145/1035292.1028990.

[6] Javier Fernandez. *The statistical analysis t-test explained for beginners and experts*. 2020. URL: https://towardsdatascience.com/the-statistical-analysis-t-test-explained-for-beginners-and-experts-fd0e358bbb62.

[7] Martin Fleck, Javier Troya, and Manuel Wimmer. "Marrying search-based optimization and model transformation technology". In: *Proc. of NasBASE* (2015), pp. 1–16.

[8] Tobias Friedrich, Christian Horoba, and Frank Neumann. "Multiplicative Approximations and the Hypervolume Indicator". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. GECCO '09. Montreal, Québec, Canada: Association for Computing Machinery, 2009, pp. 571–578. ISBN: 9781605583259. DOI: 10.1145/1569901.1569981. URL: https://doi-org.ru.idm.oclc.org/10.1145/1569901.1569981.

[9] Michael Galarnyk. *Understanding Boxplots*. 2018. URL: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51.

[10] David Hadka. *Beginner's Guide to the MOEA Framework*. 2016.

[11] David Hadka. *MOEA Framework Quick Start*. 2019. URL: https://github.com/MOEAFramework/MOEAFramework/releases/download/v2.13/MOEAFramework-2.13-QuickStart.pdf.

[12] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. "Search based software engineering: A comprehensive analysis and review of trends techniques and applications". In: (2009).

[13] Mark Harman et al. "Search Based Software Engineering: Techniques, Taxonomy, Tutorial". In: *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*. Ed. by Bertrand Meyer and Martin Nordio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–59. ISBN: 978-3-642-25231-0. DOI: 10.1007/978-3-642-25231-0_1. URL: https://doi.org/10.1007/978-3-642-25231-0_1.

[14] Stefan John et al. "Searching for Optimal Models: Comparing Two Encoding Approaches". In: *Journal of Object Technology* 18.3 (July 2019). Ed. by Anthony Anjorin and Regina Hebig. The 12th International Conference on Model Transformations, 6:1–22. ISSN: 1660-1769. DOI: 10.5381/jot.2019.18.3.a6. URL: http://www.jot.fm/contents/issue_2019_03/article6.html.

[15] Roland Kaschek. "A little theory of abstraction." In: Jan. 2004, pp. 75–92.

[16] Samir Khuller, Anna Moss, and Joseph Seffi Naor. "The budgeted maximum coverage problem". In: *Information processing letters* 70.1 (1999), pp. 39–45.

[17] Gregor Kiczales et al. "Getting started with AspectJ". In: *Communications of the ACM* 44.10 (2001), pp. 59–65.

[18] Christian Krause. *Getting Started with Henshin*. URL: https://www.eclipse.org/henshin/examples.php?example=bank.

[19] Thomas Kühne. "Matters of (Meta-) Modeling". In: *Software & Systems Modeling* 5.4 (Dec. 2006), pp. 369–385. ISSN: 1619-1374. DOI: 10.1007/s10270-006-0017-9. URL: https://doi.org/10.1007/s10270-006-0017-9.

[20] András Szabolcs Nagy and Gábor Szárnyas. "Class responsibility assignment case: a VIATRA-DSE solution". In: (2016).

[21] Alberto Rodrigues da Silva. "Model-driven engineering: A survey supported by the unified conceptual model". In: *Computer Languages, Systems & Structures* 43 (2015), pp. 139–155. ISSN: 1477-8424. DOI: https://doi.org/10.1016/j.cl.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1477842415000408.

[22] Dave Steinberg et al. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[23] Daniel Strüber. "Generating efficient mutation operators for search-based model-driven engineering". In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2017, pp. 121–137.

[24] Daniel Strüber et al. "Henshin: A Usability-Focused Framework for EMF Model Transformation Development". In: *Graph Transformation*. Ed. by Juan de Lara and Detlef Plump. Cham: Springer International Publishing, 2017, pp. 196–208. ISBN: 978-3-319-61470-0.

[25] Student. "The probable error of a mean". In: *Biometrika* (1908), pp. 1–25.

[26] David A Van Veldhuizen, Gary B Lamont, et al. "Evolutionary computation and convergence to a pareto front". In: *Late breaking papers at the genetic programming 1998 conference*. Citeseer. 1998, pp. 221–228.

[27] Lars Vogel. *Eclipse Modeling Framework (EMF) - Tutorial*. 2019. URL: https://www.vogella.com/tutorials/EclipseEMF/article.html.

[28] Steffen Zschaler and Lawrence Mandow. “Towards Model-Based Optimisation: Using Domain Knowledge Explicitly”. In: *Software Technologies: Applications and Foundations*. Ed. by Paolo Milazzo, Dániel Varró, and Manuel Wimmer. Cham: Springer International Publishing, 2016, pp. 317–329. ISBN: 978-3-319-50230-4.