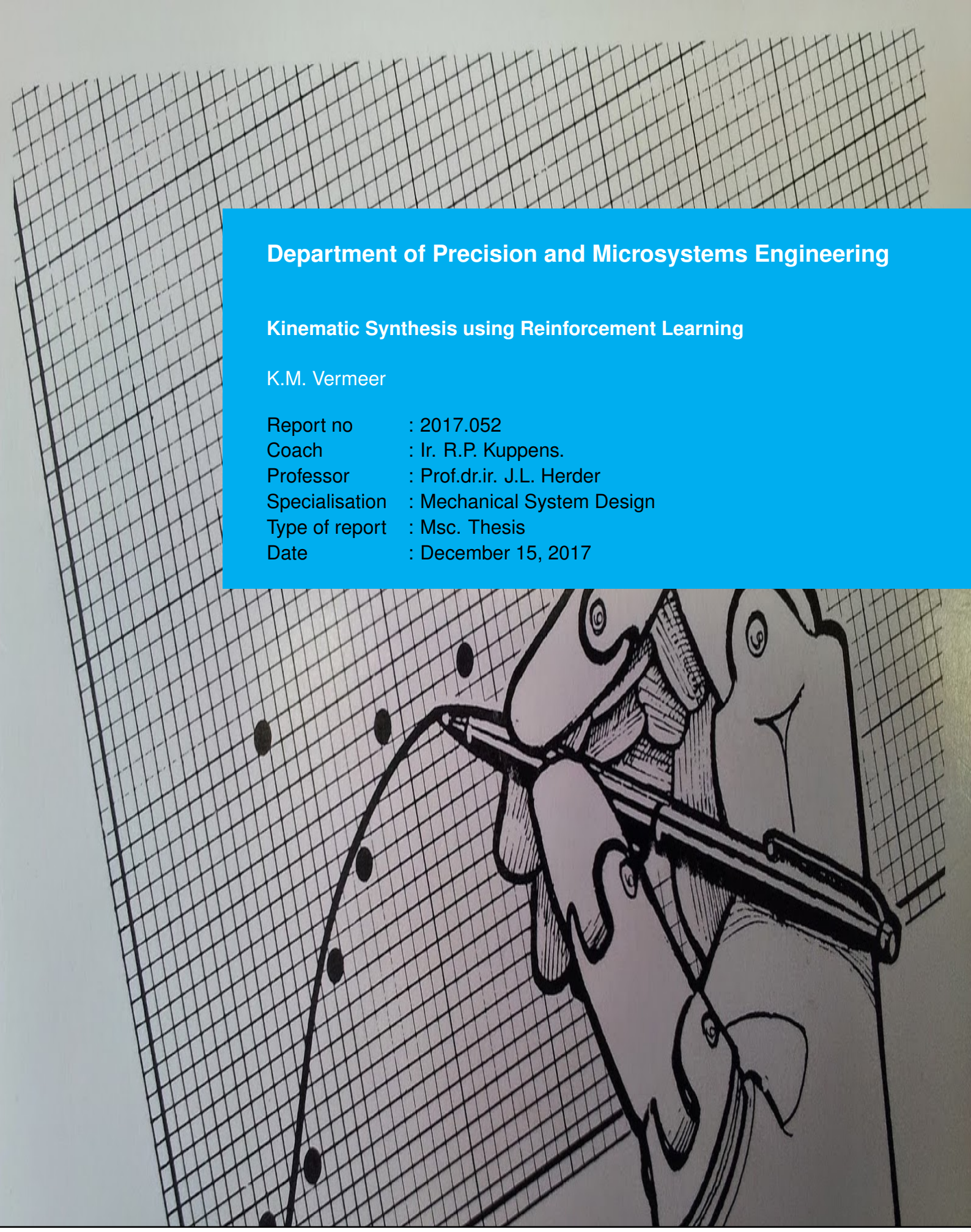


## Department of Precision and Microsystems Engineering

# Kinematic Synthesis using Reinforcement Learning

K.M. Vermeer


| | |
|---|---|
| Report no | : 2017.052 |
| Coach | : Ir. R.P. Kuppens. |
| Professor | : Prof.dr.ir. J.L. Herder |
| Specialisation | : Mechanical System Design |
| Type of report | : Msc. Thesis |
| Date | : December 15, 2017 |


TU Delft Delft University of Technology

Challenge the future

# Kinematic Synthesis using Reinforcement Learning

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

K.M. Vermeer

December 15, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

.

# Abstract

Advanced tools such as machine learning are slowly finding their way into the modern scientist's toolbox . In the design of mechanical systems however hardly any machine learning applications are being used. Research into the viability of such an application is therefore necessary.

We have performed such research, using a specific type of machine learning, known as reinforcement learning, for the synthesis of kinematic mechanisms. Reinforcement learning is an experience-based learning strategy which has proven particularly successful in learning to play games, like chess, blackjack or Go. In this research it is shown that the sequentially alternating nature of game-playing between actions and reward can also be observed in mechanism design by posing design challenges in a game-like format. We have used a decision-tree based mechanism representation developed by Lipson [1] to create such a game-like world in which mechanisms can be designed. To train an actor to navigate this game-like world both Monte Carlo and Temporal Difference learning have been applied, in combination with a neural network as nonlinear value function approximator. Moreover a kinematic simulator and scoring modules have been implemented to evaluate synthesized mechanisms.

We demonstrated the successful implementation of the framework and learning algorithm by synthesizing mechanism for two separate path tracing objectives: straight lines and figure-eights. A set of recommended algorithm settings was extracted from a parameter sweep and grid search including a total of 560 test runs. Straight line mechanisms were obtained with a fixed maximum number of 10 nodes, drawing lines with aspect ratios up to 1:1168. Additionally a mechanism was synthesized capable of drawing figure-eight patterns.

We conclude that the use of reinforcement learning in the context of mechanical system design is viable. More specifically using the presented method kinematic synthesis for path tracing objectives can be performed. The current research cultivates the land for future efforts to bridge the gap between the challenges faced by mechanical system design groups and the advancing solutions developed by the computer sciences.

# Table of Contents

# Preface

Computers, at some point in the future, might be able to pass the Turing test and conceive of thoughts equally or more complex than those of humans. Does that mean they are alive? Does that mean a computer can have a character, a will, a soul? And in that case how do we, humans, differ from computers, except for our physical appearance? Such unanswered questions have fueled my fascination with machine learning over the last few years.

It was under the influence of this fascination that I found myself looking for a thesis assignment that could merge my mechanical engineering background with this newborn interest in machine learning. During an exploratory session with Phd students about possible thesis assignments I met Reinier, who explained he had only recently graduated on the synthesis of mechanisms using evolutionary algorithms. Together we decided to use my literature study to research the possibilities of using generative approaches in the synthesis of compliant mechanisms.

During the literature phase of my research I became enthusiastic about a specific learning method known as reinforcement learning, even though this concept had never been applied to anything remotely connected to mechanical engineering. After discussing with Reinier and Just (the chairing professor) I decided to focus on the application of this learning method to mechanical system design. After a lot of reading, experimenting and waiting an algorithm came about that actually showed some first signs of success. A series of iterations culminated into the results presented in this report.

As it turned out, I found the whole process of doing research, designing an algorithm and actually developing and testing it more enjoyable than anticipated. This enjoyment was not in the least place fueled by my supervisor Reinier, whom I cannot thank enough for his enthusiasm, flexibility and academic guidance. I have come to consider Reinier a friend next to being my supervisor: a worthwhile combination.

Finally I would like to say thanks to my chairing professor Just Herder. His trust in and support of this research have been vital. Moreover his questions forced me to zoom out, away from the algorithms, papers and scripts and to see the context of this thesis and my place in the annals of science.

Delft, University of Technology — K.M. Vermeer
December 15, 2017

# Chapter 1

# Introduction

The ability of mankind to analyze and solve mechanical problems has driven technology throughout the ages, from the invention of the combustion engine to more recent advantages in micro-electromechanical systems (MEMS). Fundamental to these successes are well-executed design processes. As the boundaries of technology are pushed towards higher levels of complexity, the challenge of designing the incorporating systems has grown accordingly. Fortunately, mechanical system design does not stand alone in facing the challenges modern times have brought about.

Throughout several research fields a parallel movement towards machine learning can be witnessed as researchers' answer to increasingly complex challenges. Machine learning is finding applications by the dozens [2–4] as a potential way to get a grip on increasingly complex research questions. Defined in 1959 by Arthur Samuel [5] as the "field of study that gives computers the ability to learn without being explicitly programmed", machine learning has the potential to achieve (super)human levels of artificial understanding in complex manners like performing real-time translation [6], mastering helicopter aerobatics [7], coloring black-and white images [8] or beating the world champion in Go [9], an ancient Chinese war game featuring $10^{170}$ possible board positions. For comparison Fig. 1-1 shows the relative levels of performance machine learning can achieve on 49 different games with respect to human level capabilities. Its ability to understand complex matters on a level equal to or beyond human capacity makes machine learning a potentially powerful aid to researchers and engineers in dealing with modern-day's complex challenges.

This brings about the question whether machine learning's potential can also be applied to mechanical system design. Design robots could assist human designers by exploring the solution space and generating inspiring designs, based on design goals and constraints described by the human designer. This type of design process, in which designer input is limited to the goal and constraints of the design challenge, is coined generative design. Currently, generative design has been mostly applied to the design of structures, in which topology optimization [11] and evolutionary algorithms [12] are proving their value. The trend towards increased complexity urges researchers to look further, exposing mechanism design to the potential benefits it may reap from advancements in machine learning.

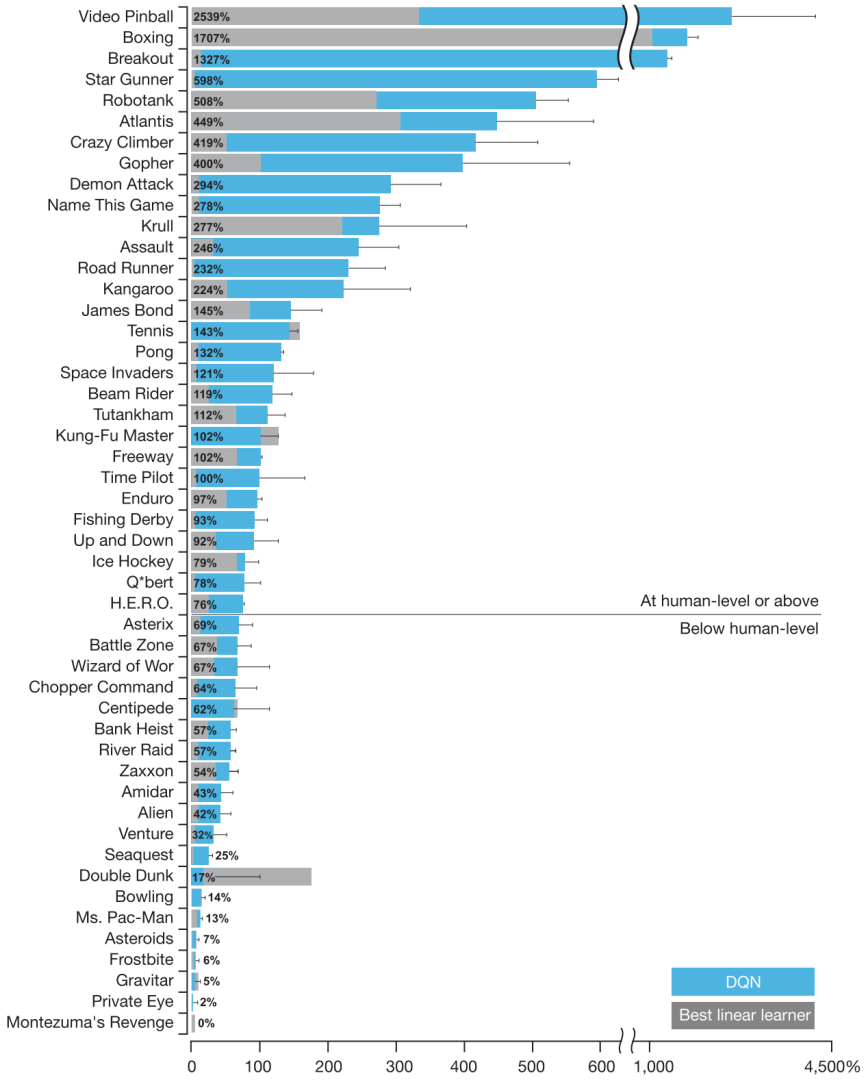



**Figure 1-1:** Relative performance on 49 different games by two types of machine learning with respect to a professional level human player (100%). Image reproduced from [10].

The approaches used in game playing or robot learning may serve as inspiration, but are unfortunately not easily transfered to mechanism design. Most notably, the examples given can all be formalized into a numerical description and/or sets of rules: one's position in a game environment, a robot's state et cetera. Mechanism design, however, is not strictly bound to a given set of rules or a limited solution space, making it a difficult subject to treat in a numeric sense without limiting design freedom. Furthermore, the experience and feeling of respected mechanism designers cannot be expressed in terms of bits and bytes and therefore cannot serve as reference to learn from. This makes mechanism design an atypical application for machine learning. Bridging the gap towards mechanism design is therefore a challenging but vital endeavour in an effort to tap into new applications of machine learning.

To support this cause the current thesis work initialized with a literature survey, exploring the possibilities of generatively synthesizing mechanisms. A summary of this survey is presented hereafter. From this summary the current thesis subject is extracted, which will be presented in the subsequent section. Finally the contents of the report will be discussed in this chapter's final paragraph.

## Literature survey

The survey considered using generative synthesis methods for the design of a specific type of mechanisms featuring compliant members. Such compliant mechanisms (CMs) yield advantages in terms of production cost and performance [13] over conventional mechanisms. Because of the coupled force-deflection behavior in CMs however, designing them is challenging. Computerized help could offer relieve. The goal of this literature survey was to separately inquire into three important prerequisites for a generative CM design approach culminating into a comprehensive table (Table 1-5), indicating potentially fruitful and symbiotic combinations as well as incompatible setups. The three mentioned prerequisites are :

- Numeric mechanism representation: a formal description of a mechanism
- Optimization algorithm: a smart search strategy for finding the optimal design
- Compliance modeling: a way to model and evaluate the behavior of CMs

A compact summary of the review is given in this introduction, although the full survey is available in appendix G. The approach used to compare different methods within each prerequisite will be treated here briefly. An extensive description of the method used to combine these results, alongside more detailed descriptions of the individual evaluation procedures, are also treated in the appendix (mechanism representations: appendix G-3, optimization algorithms: appendix G-4, compliance modeling: appendix G-5).

### Numeric mechanism representation

Balakrishnan and Honavar [14] formulated nine characteristics by which to evaluate neural networks. Kuppens [15] showed the wider applicability of this framework, also covering mechanism representations. A selection of these characteristics has served as the basis for evaluating mechanisms representation methods. They are related to three core values:

- Creative Freedom: the ability to support any type of possible design outcome without imposing design limits. Creative Freedom is required to uncouple generative design from our preconceived notions about CM so as to avoid biasing the generative process.
- Comprehensiveness: the ability to take all aspects of mechanism design into account. Only when topology, size and shape are described can a representation be comprehensive.
- Optimization perspective: the ability of a representation method to be deployed in combination with an optimization or learning algorithm.

Let us denote the total universe of mechanisms that may be described by a certain representation as the solution space. Some representations rely on the discretization of this solution space. In general, this means the total solution space for mechanisms is replaced by a grid-type solution space, with distinct nodes and connections. Examples of such representations are the building block approach [16–18] or the truss-based ground structure representation [19], which was extended with a set of optimizable parameters by [20].

Another important segment of the discretized solutions relies on density-based representations [21,22]. Used extensively in topology optimization, density based representations use an entirely discretized solution space in which each sub-domain is assigned a density value. Material is distributed over the solution space's continuum structure by assigning each sub-domain (square) a binary or continuous density value.

Sticking with the realm of topology optimization, one may also adopt the notion of level-set methods introduced by Sethian and Wiegmann [23]. In level-set methods a function's iso-line is used to describe material distribution. Material distribution can be changed by adapting the iso-line level or changing the level-set function. A structured review of research into this subject was given by Van Dijk [24].

Alternatively, Lipson [1] proposed a decision-tree representation to describe a series of operations resulting in a unique kinematic structure. Lipson distinguished two different operators: $\boldsymbol{T}$ and $\boldsymbol{D}$, describing a mechanism as an initial design and a set of subsequent operations.

Finally graph theory has been used prolifically in generative mechanism design [15, 19, 25–29]. Graph descriptions can be easily extended to include all sorts of extra information besides its topological content. It is however prone to isomorphisms: for each unique mechanism several graphs exist describing the mechanism correctly.

All presented mechanisms have been scored on their merits with respect to each of the three core values. The weighted sum of these scores has led to an overall score and classification of each representation. The results are shown in Table 1-1.

## Optimization algorithm

Numerous optimization algorithms exist, ranging from topology optimization to advanced machine learning concepts. Again, a set of potential solutions has been evaluated based on three core values. These values are:

- Creative Freedom: the ability to support any type of possible design outcome without imposing design limits.

- Optimization speed: the ability to quickly converge towards the right solution.
- Design compatibility: the ability of an optimization algorithm to optimize for something as complex as mechanism designs.

In the case of generative mechanism design, objective functions are typically subject to discrete variables and therefore non-differentiable. This limits the scope of usable algorithms severely, explaining why in literature only a limited number of algorithms has been found.

The most frequently used optimization techniques in generative CM design is Topology Optimization (TO) . TO as introduced by Bendsoe [30] in 1988, determines the optimal topology and shape of a structure for a specific objective. Most TO routines use a density-based approaches like the homogenization [30] and SIMP methods [31]. These methods often use a Finite Element Method (FEM) to evaluate a fine mesh of elements. By using a large number of elements a continuum is approximated. This makes TO compatible with gradient-based optimization algorithms after all.

Alternatively evolutionary algorithms (EAs) have been applied to cope with non-differentiable objectives. EAs were first introduced by Holland [32] and emulate nature' s evolutionary forces of selection, combination and mutation to fulfill their objectives in all sorts of engineering fields [1, 33]. EAs rely on simulation-based fitness evaluation and a set of stochastic rules rather than on direct gradient information.

As a third alternative, machine learning (ML) approaches have been researched. The field of ML can be split into three parts: supervised learning, unsupervised learning and reinforcement learning. The former two types of ML rely on large datasets to learn. The latter type, reinforcement learning, was described by Kaelbling et al. [34] as behavioral learning by trial-and-error interactions with a dynamic environment, ergo using experience instead of datasets. Large datsets concerning mechanisms are not readily available and learning from human-designed examples would defeat the purpose of computer-aided generative design.

Table 1-2 shows the evaluation results for the optimization algorithms:

## Compliance modeling

Modeling compliant structures and mechanisms involves numerous challenges. First of all designing CMs requires considering multiple objectives [35] since the resulting designs should

**Table 1-1:** Scores and classifications per representation: Building blocks, truss-based ground structures, Zhou's [20] approach, density-based approach, Lipson's [1] decision tree, level-set methods, graphs and extended graphs.

| Value | BB | Truss | Zhou | Density | DT | LSM | Ext.graph | Weight |
|---|---|---|---|---|---|---|---|---|
| CF | -3 | -3 | 2 | 2 | 2 | 3 | 3 | 3 |
| Comp. | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 2 |
| Optim. | 2 | 3 | -1 | 0 | 1 | -2 | -3 | 1 |
| Score | -7 | -4 | 9 | 10 | 13 | 13 | 14 | |
| Class | - | - | +/- | + | + | + | + | |

**Table 1-2:** Overview of characteristics optimization algorithms. TO = topology optimization, EA = evolutionary algorithms, SL = supervised learning, USL = unsupervised learning and RL = reinforcement learning.

| Characteristic | TO | EA | SL | USL | RL | Weight |
|---|---|---|---|---|---|---|
| Creative freedom | ✓/× | ✓ | × | × | ✓ | 3 |
| Optimization speed | ✓ /× | × | - | - | ✓/× | 1 |
| Design compatibility | ✓ | ✓ | × | × | ✓/× | 2 |
| Score | 8 | 10 | -5 | -5 | 9 | |
| Class | +/- | + | - | - | +/- | |

be both flexible to achieve deflections, and stiff to retain structural integrity whilst withstand reaction forces. Therefore a number of objectives can be considered, like the mechanical advantage [36] or geometric advantage [37]. In 1970, Shield and Prager proposed another alternative using a mechanism's mutual potential energy [38]. For a more thorough analysis of these and other design objectives the reader is redirected to appendix G-5-2.

For simulating the kinematic behavior of a CM, only approximation methods are considered, given that exact solutions only exist for specific shapes whereas a general solution is sought. In literature, two distinctive approaches for the modeling of compliant kinematics are being used extensively: the Finite Element Analysis (FEA) [17,20,37,39–42] and Pseudo-Rigid-Body (PRB) models [43–47].

In FEA a continuous structure is modeled as a collection of discrete elements of finite length. Elements can be of all shapes and sizes, but are usually geometrically simple shapes like beam and bar elements. When performing the analysis, the behavior of each separate element is determined by solving the partial differential equations governing elastic deformation. The equations are solved exactly in a set of precision points, referred to as nodes, and interpolated using shape functions throughout the structure. Within FEA, a major distinction can be made between linear and non- linear FEA. Since compliant mechanism by trait go through large deflections, the use of linear FEA has been deemed infeasible.

In 1996 Howell and Midha [43] proposed a method in which a compliant mechanisms is accurately approximated by an equivalent rigid-body model. This so-called pseudo rigid body (PRB) method introduces all well-known rigid-body mechanism theory into the compliant mechanism design domain. As a result fast and exact computations can be made with respect to the CM's approximate PRB model.

As a result of elaborate evaluation, described in appendix G-5-2, the characteristics of both compliance modeling methods have been determined (Table 1-3).

## Overview

To capture all three subjects in a two dimensional overview, the mechanism representations and compliance modeling methods have been combined as described in appendix G-6. Each of these combinations has been scored in terms of synergy ranging from − for a mismatch to + for synergistic combinations. Intermediate combinations have been scored +/− and

**Table 1-3:** Characteristics of FEA and PRB models for compliance

| Characteristic | FEA | PRB | Weight |
|---|---|---|---|
| Speed | × | ✓ | 1 |
| Accuracy | ✓ | ✓ | 2 |
| Versatility | ✓ | × | 2 |
| Score | 7 | 4 | |
| Class | +/- | +/- | |

incompatibilities have been denoted with a ∘. All fourteen pairs have subsequently been combined with the selection of optimization algorithms and received a similar synergy score. In parallel, the individual scores, based on the core-values, should be taken into account to determine which combinations of solutions are both synergistic and endorse the core values. The resulting table is shown in Table 1-4.

Derived from the overview in Table 1-4 come three potentially synergistic solutions, which are tabulated in Table 1-5.

By dividing the large challenge of generative design into three subproblems, solutions for these subproblems have been researched without being distracted by the overarching problem. As a result, solutions applied in other research fields have been introduced in the context of the current research and have shown to be promising. Interestingly, two out of the three promising configurations break precedent on several issues. Most daringly, the use of level-set methods and reinforcement learning in mechanism synthesis are both unheard of, even though their resulting evaluation scores are high. Also, by elucidating the method used and argumentations applied to the scoring procedure this table can be extended at any time in future work.

## Thesis subject

Based on the propositions resulting from the literature study, a solution direction has been selected for further development. Because of its unique nature and great compatibility, we have chosen to pursue a generative design algorithm combining Lipson's [1] decision tree with reinforcement learning. Additionally, the current focus on reinforcement learning in the scientific community, by for instance the DeepMind lab [9, 48], has added to the expectation that a thesis on this subject may result in a valuable scientific contribution. The main goal of this thesis is to asses the possibility of using reinforcement learning as an effective tool in designing mechanisms. Unfortunately incorporating compliance modeling would have stretched past the limits of a thesis assignment. Therefore, the scope has been limited to the design of two-dimensional kinematic linkages. A path-tracing objective has been selected as running example. Three subgoals have been defined to address the general objective:

a **Posing the design problem:** Mechanisms will be represented by Lipson's decision tree method [1]. During learning, an agent is required to choose the best operation given the current design. A series of decisions results in a sequence of operations, which in turn lead to a design. The first goal is therefore to create a framework that can interpret decision trees, take in new operations and determine fitness with respect to the design goal.

**Table 1-4:** Overview table of mechanism representations, compliance modeling and optimization methods and their mutual synergies. Scores from individual analysis based on core values are provided. Representations: BB = Building Block method, Truss is Truss-based ground structures, Zhou is Zhou's [20] method using a flexible truss-based ground structure description, Density = density-based approach, DT = Lispon's [1] decision tree description, LSM = level-set method, Ext. graph = Extended graph representation, using labels to enhance comprehensiveness of graphs. Sub-zero scoring optimization algorithms and mechanism representations have been grayed out.

| Representations | | Compliance | | Optimization | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | +/- | + | - | - | +/- |
| Method | Class | Method | Synergy | TO | EA | SL | USL | RL |
| BB | - | FEA | ○ | ○ | + | ○ | ○ | ○ |
| | | PRB | ○ | ○ | + | ○ | ○ | ○ |
| Truss | - | FEA | + | + | +/- | ○ | ○ | ○ |
| | | PRB | +/- | - | +/- | ○ | ○ | ○ |
| Zhou | +/- | FEA | + | + | +/- | ○ | ○ | ○ |
| | | PRB | +/- | - | +/- | ○ | ○ | ○ |
| Density | + | FEA | + | + | - | ○ | ○ | ○ |
| | | PRB | - | - | - | ○ | ○ | ○ |
| DT | + | FEA | + | ○ | + | ○ | ○ | + |
| | | PRB | +/- | ○ | + | ○ | ○ | + |
| LSM | + | FEA | + | + | - | ○ | ○ | ○ |
| | | PRB | - | - | - | ○ | ○ | ○ |
| Ext. graph | + | FEA | +/- | ○ | + | ○ | ○ | ○ |
| | | PRB | + | ○ | + | ○ | ○ | ○ |

**Table 1-5:** Three most promising solution combinations extracted from Table 1-4.

| Mechanism representation | Compliance modeling | Optimization or learning method |
|---|---|---|
| Level-set method | Finite element analysis | Topology optimization |
| Decision tree method | Pseudo-rigid body | EA or RL |
| Extended graph method | Pseudo-rigid body | Evolutionary algorithm |

b **Creating a learning algorithm:** Once the environment represented by the framework from the previous subgoal is in place, a learning agent can start to interact with it. Therefore the second goal is to implement an actor that is able to learn within the bounds of the given framework.

c **Demonstrate the algorithm:** Once a learning algorithm is created it should be demonstrated on an example problem. Once learning has been demonstrated, the framework can be used to assess a second learning problem and demonstrate its general applicability.

## Contents of report

The means and ends to these (sub)goals are condensed into a scientific paper presented hereafter. Subsequently a series of closing remarks is given in chapter 3. For readers interested in a level of detail beyond the paper's scope, extended work is presented in the appendices. The included subjects are shown in Table 1-6, including the paper sections they relate to (if applicable):

**Table 1-6:** Overview of appendix subjects. Paper sections are referred to for the applicable subjects.

| Appendix | Subject | Paper section |
|---|---|---|
| A | A background study on the subject of machine learning | - |
| B | Derivation of backpropagation algorithm | II-C-1 |
| C | Elaboration on neural network implementation | II-C-1 |
| D | Elaboration on Markov Decision Process framework | II & III |
| E | Elaboration on sensitivity methods | III-E & IV-D |
| F | Overview of experimental results | IV |
| G | Literature survey | - |

# Kinematic Synthesis using Reinforcement Learning

Kaz M. Vermeer, Reinier P. Kuppens, Just L. Herder

***Abstract*—The presented research explores the possibility of applying feature-based reinforcement learning to the synthesis of two-dimensional kinematic mechanisms. As a running example the classic challenge of designing a straight-line mechanism is adopted: a mechanism capable of tracing a straight line as part of its trajectory. This paper presents a basic framework, consisting of elements such as mechanism representations, kinematic simulations and learning algorithms, as well as some of the resulting mechanisms and a comparative benchmark to prior art. A sensitivity analysis is performed to analyze the neural network's considerations with respect to the selected features. Finally the general applicability of the algorithm is tested and validated by adopting a second design goal.**

## I. Introduction

MACHINE learning has quickly gained popularity in a wide variety of research fields since the early 21st century [1]–[4]. In the synthesis of mechanical systems however hardly any machine learning applications are being used. Research into such an application is therefore necessary to expose mechanism synthesis to the potential benefits it may reap from advancements in machine learning.

Over the past years research has been conducted on the synthesis of mechanisms by means of topology optimization [5]. In most cases, the optimization process starts with a fully connected homogeneous structure from which pieces of material are gradually eliminated until a mechanism is found that fits the needs of the designer. The prerequisite of having a fully connected starting structure however limits its applications. Research involving constructive approaches, adding elements rather then eliminating them, has thus far mainly relied on evolutionary algorithms [6]–[9]. These algorithms initiate a quasi-random population of individual designs and perform a series of stochastic selections, combinations and mutations to evolve the population into a group of functioning designs.

None of these efforts however apply reinforcement learning (RL), an experience-based learning concept currently researched in other fields [10]–[12]. This method is capable of coping with levels of complexity evolutionary algorithms cannot accommodate [13]. RL is specifically well-suited for game-like situations [14] in which decisions have to be made and rewards may be obtained. Application of RL onto mechanism design therefore requires the development of an inventive new framework in which design challenges are posed in a game-like fashion. This research presents just that in an attempt to bridge the gap between the challenges faced by the field of modern mechanical system design and the advancing solutions developed by the computer sciences.

The present paper offers a method implementing RL in kinematic synthesis using a decision-tree-based mechanism representation and nonlinear value function approximation. To demonstrate this method the challenge of designing a straight-line mechanism has been adopted. This has resulted in a series of successful straight-line mechanisms being synthesized using RL. The applied method has been developed independently of the adopted design goal and should therefore generalize to other goals as well. To demonstrate this ability a second design goal has been introduced. Using the same features and settings, the algorithm obtained similarly successful results on the second design goal.

The method used in this research is presented in section II after which the experiment is described in section III. The results from the experiment are presented in section IV and discussed in section V. Finally the conclusion is given in section VI.

## II. Method

### A. Mechanism representation

In order to synthesize, interpret, evaluate and manipulate mechanism designs, a numeric system, i.e. a computer, has to be able to communicate about such objects. This requires a befitting language, or in formal terms a numeric representation of designs. Additionally the representation should be convertible into the visual domain easily for purposes of human inspection. Literature study [15] provides an overview of methods for such representations and of algorithms for manipulation and evaluation. This study also indicates that one specific representation method exists that forms a strong synergistic combination with reinforcement learning. This representation was first introduced in 2008 by Hod Lipson [6].

Lipson [6] proposed a decision tree representation to describe a series of decisions resulting in a unique kinematic mechanism. In each decision one of two operations is performed: $\boldsymbol{T}$ or $\boldsymbol{D}$. Given a kinematic system of interconnected links and nodes, both operators act on a specified target link when executed. The operators introduce a new node and connect it to the the target link's endpoints. In case of a $\boldsymbol{T}$ operator, the target link is subsequently replaced by a connecting link between the new node and its nearest non-connecting neighbor. A $\boldsymbol{D}$ operator simply leaves the target link untouched. Using Fig. 1 one may compare the different results of these operators. Both $\boldsymbol{T}$ and $\boldsymbol{D}$ have the particularly useful property of conserving a mechanism's number of degrees of freedom (DOF). This means these operators are intrinsically restricting the design space to include only feasible domain. This method therefore prevents efforts being poured into a forlorn search through infeasible designs. This is in contrast to most other representations, like graphs, which do allow over- and under-constraint mechanisms to occur, as was the case in [7].

Given an initial mechanism as starting point, a series of $\boldsymbol{T}$ and $\boldsymbol{D}$ operators fully describes a new design. The sequence

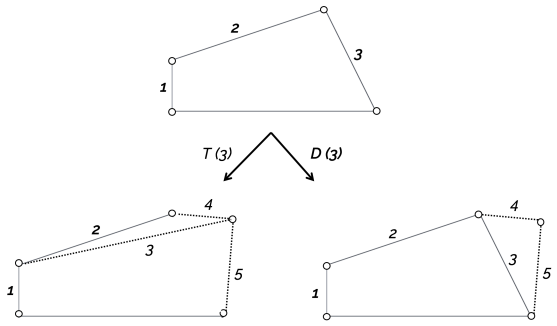



Fig. 1: Example of different outcomes for $\boldsymbol{D}$ and $\boldsymbol{T}$ operations on link number 3.

of operators can be represented as a decision tree, depicted in Fig. 2. In Lipson's implementation [6], the operators are accompanied by an integer, indicating the target link, and a local coordinate, describing the placement of the new node relative to the target link's position.

### B. Kinematic simulation

In the presented framework all generated mechanisms have to be automatically evaluated for performance. To evaluate a mechanism's path-tracing performance in terms of accuracy, kinematic analysis is required. Since no a priori knowledge of the synthesized mechanisms is available, a robust analysis method is needed which can handle any conceivable mechanism within the representation's domain. Therefore numeric simulation is used. Fig. 3 shows the initial configuration of an example mechanism. The bottom two nodes are both fixed. The curved arrow indicates the input link. During simulation the input link is rotated a full circle in 300 equally spaced steps by prescribing its position. After each perturbation the positions of the free nodes are determined by solving the position problem. Finally, the infeasible domain is removed and a smooth trajectory extracted. Applying this treatment to the mechanism presented in Fig. 3 leads to the curved trajectory shown by the dashed blue line.

*1) Solving the position problem:* Since the position problem has to be solved tens of millions of times whilst learning, a simple, general and computationally cheap method is desired. Moreover the method should be able to cope with and detect infeasible domain. Avilés et al.'s method [16] presents just that, involving a reduced form stiffness matrix known as the geometric stiffness matrix $[g]$ for fast computation and an elastic potential error-function to detect and cope with infeasible domain. In this method mechanisms are modeled using rod-type finite elements with revolute joints on both ends, allowing only axial deformation. In each time step the position problem is solved numerically by determining and minimizing the error function: the system's elastic potential $V$. The error function for a linkage with a total of $b$ links of length $L_e$ equals:

$$V(x) = 1/2 \sum_{e=1}^{b} (l_e(x) - L_e)^2 = 1/2 \sum_{e=1}^{b} \left( \sqrt{x^T [\bar{g}]_e x} - L_e \right)^2 \quad (1)$$

in which $l_e(x)$ represents the length of element $e$ as a function of the nodal coordinates in $x$. During iteration, $l_e$ might not be equal to $L_e$, introducing elastic potential and thereby error. As in Avilés et al.'s [16] approach Newton's second-order method [17] is adopted to solve the optimization problem, minimizing $V$ with respect to $x$. Iterative updates of $x$ continue until the error $V$ drops below a predefined tolerance of $1 \cdot 10^{-3}$ or the iteration count exceeds 100, after which the position problem is considered solved.

*2) Detecting infeasible domain and singularities:* Depending on the geometry of a mechanism, the input link may not be able to turn a full circle. During numeric analysis, such infeasible domain can be detected by high levels of elastic potential. By removing trajectory points with high elastic potential only the feasible domain remains. In the limit cases, i.e. mechanisms with dimensions just slightly preventing it from entering infeasible domain, singularities can occur. In such cases somewhere along the trajectory two or more links become aligned resulting in locking behavior, effectively losing one degree of freedom. Such an unstable mechanism position results in large gradients of the potential energy. Accordingly, the gradient and Hessian matrices in Newton's second-order method become ill-conditioned. To deal with the resulting large gradients an adaptive learning rate is employed, preventing overshoots whilst searching for an equilibrium position. This adaptive learning rate $\alpha$ is cut in half whenever the error is rising instead of falling. No updates of $x$ are performed in this case until the error ceases to rise.

### C. Learning approach

Reinforcement learning (RL) is described by Kaelbling et al. [14] as behavioral learning by "trial-and-error interactions with a dynamic environment". RL can be applied when no data is available a priori, but instead a reward signal is received after each experienced data point. Hence a machine can learn by taking actions, assessing the resulting rewards and adapting its decision policy accordingly. Through iteration, the machine can learn to take the actions that maximize the total reward.

As explained by Sutton and Barto [18], in RL the learner is named the *actor*. The actor gains experience through taking *actions* that lead to interactions with its *environment*. After every interaction, the actor is faced with a new situation, or *state*, from which it may perform the next action. As part of the interactions, the actor may receive *rewards.* The actor's goal is to choose actions such that it maximizes the total reward. In a more formal manner, one can say that for every time-step $t$ the actor is in state $S_t$. After taking action $A_t$, it ends up in state $S_{t+1}$ with probability $P_{S_t A_t}(S_{t+1})$ and receives a reward $R_{t+1}$. A schematic of this process is shown in Fig. 4. Over time, the actor's total return $G_t$ equals the sum of all future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (2)$$

A discount factor $\gamma$ is introduced as a means to promote fast results. This procedure is governed by the five-element tuple $(S, A, P, \gamma, R)$. Decision processes described by such a tuple

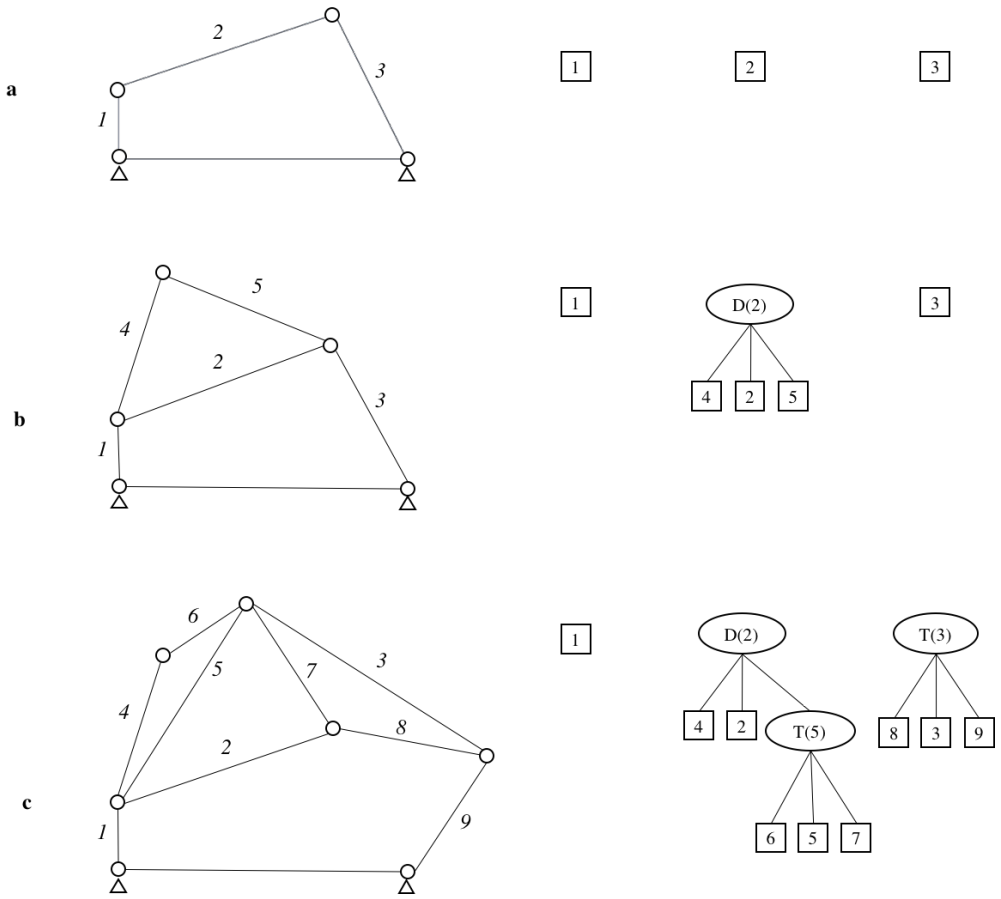



Fig. 2: Example of a decision tree. The initial mechanism in (a) features three operable links: 1, 2 and 3. A $\boldsymbol{D}$ operation is performed on link number 2, resulting in mechanism (b) and its corresponding decision tree. Subsequently, two new decisions are made by performing $\boldsymbol{T}$ operators on both links 3 and 5, resulting in mechanism (c). This tree and its corresponding mechanism may continue to grow as more decisions are being made. Image inspired by Fig. 3 in [6].

are sometimes referred to as Markov Decision Processes (MDP) [19], referring to their adherence to the Markovian property [20] and inclusion of both deterministic and stochastic elements. Using the tuple's elements, one can define the value function $V_\pi(s)$ as the expected return $G$ of being in state $s$ and following policy $\pi$ until termination [18]:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[ G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \end{aligned} \tag{3}$$

Since one is usually concerned with choosing the right action, it is common practice [18], [19] to include both the state and the action as variables of the value-function, leading to the state-action value-function $Q$:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi \left[ G_t | S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \end{aligned} \tag{4}$$

For a known MDP, the fixed point theorem [21] shows that $V_\pi$ can always be found in a recursive fashion. In reality however, the MDP is never fully known and $V$ or $Q$ can only be estimated. Several algorithms exist that can be used to improve an estimation of $V$ or $Q$ after each completed series of states and actions also known as an *episode*. To improve efficiency, one can bootstrap by updating after each time-step instead of after each episode. The former method is referred to as Monte Carlo learning (MC), whereas the latter is known as Temporal Difference learning (TD). Both methods are described in depth by Sutton and Barto [18].

The $Q$-function becomes more and more accurate by learning, which can be exploited to make sound decisions and therefore maximize reward $R_t$. Always choosing the action that maximizes expected reward in $t+1$ is known as a greedy policy. Such a policy however is short-sighted and likely to get stuck in local optima. To prevent such behavior a policy called $\epsilon$-greedy [18], [22], [23] is adopted. This policy introduces randomly selected actions with a probability of $\epsilon$ to encourage exploration of the action space. In the current application $\epsilon$ is initialized as high as 0.5 and diminishes throughout the learning process.

For the current application TD learning is chosen, since this method is found to generally converge faster then its MC counterpart [18]. The simplest and best-known TD algorithm

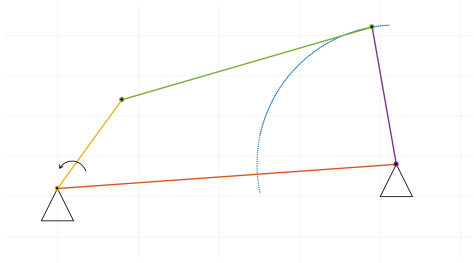


Fig. 3: The starting point: a basic four-bar mechanism with two grounded nodes. The arrow indicates the input link. The top link, often named coupler link, transfers the input motion to output link. The blue dashed line depicts the output link's path, moving back and forth along the same trajectory.

is SARSA, its pseudocode shown in algorithm 1. In each time-step $t$ an error term between the true and expected reward is determined. Afterwards the approximation of $Q(S_t, A_t)$ is adjusted in the direction of the true value by a small step of size $\nu$.

Initialization;
  $Q \leftarrow$ arbitrary;
**foreach** *Episode* **do**
  Initialize $S$;
  Choose $A$ from $S$ using $\epsilon$-greedy policy and $Q$;
  **foreach** *Step in the episode* **do**
    Take action $A$, receive $R$ and $S'$;
    Choose action $A'$ from $S'$ using $\epsilon$-greedy policy and $Q$;
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \nu\,[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$;
    $S \leftarrow S'$ ; $A \leftarrow A'$;
  **end**
**end**

**Algorithm 1:** SARSA using TD learning from [18]

*1) Value function approximation:* In the SARSA algorithm $Q(S, A)$ is calculated and adjusted. In a practical sense, this means $Q$ should be an entity that can take a state and action as input and produce an expectation as output. Several embodiments exists for such an entity: closed-form mathematical functions, lookup-tables and (non)linear function approximation.

Closed-form mathematical functions, used to calculate $Q$ as $f(S, A)$, are not a fitting choice as they generally are not compatible with state-descriptions and discrete actions as inputs. Secondly, iterative updates of closed-form functions can only materialize by means of parameter adjustments. Selecting the right parameters to include in the function however requires a priori knowledge of the desired function approximator, which is unavailable.

Secondly, one could represent $Q(S, A)$ in a tabular fashion, keeping track of a separate $Q$ value for each unique state-action pair [18]. For specific applications this method can be effective [24]. However, besides the obvious memory limitation to small state and action spaces, the tabular approach also fails in what is known as the generalization of states [25]: the ability to

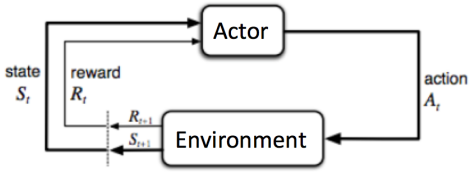



Fig. 4: Schematic showing the cyclical life of an RL actor. Image reproduced from [18].

use experience gained in state-action pair $(S, A)$ for decision making in closely related state-action pairs. In a table each state-action pair is treated as a perfectly separated individual, such that experience from visiting state-action pair $(S, A)$ is only valuable for future visits to that exact same pair. Neighboring pairs however, with states and actions much alike $S$ and $A$, may also benefit from the $(S, A)$ experience. Tabular representation cannot facilitate such cross-border sharing of experience.

Fortunately a third solution exists that does: estimating $Q(S, A)$ by using linear function approximation. A set of features $f(S, A)$ can be obtained and combined linearly using weights $w$ to approximate $Q(S, A)$:

$$Q(S, A) = w_1 f_1 + w_2 f_2 + ... + w_n f_n = w_i f_i \quad , \qquad (5)$$

adopting Einstein's summation notation for a total of $n$ weights and features. A suitable feature list serves as a summary of the current state-action pair, containing only the information significant for decision making. A unique mapping exists from a state-action pair to its feature list. Comparable state-action pairs however may result in similar or identical features. Therefore linear function approximation allows for the generalization of states, in contrast to the tabular approach. Experience gained in state-action pair $(S, A)$ is valuable for visits to comparable state-action pairs $\hat{S}, \hat{A}$ too, as long as their feature values $f(S, A)$ and $f(\hat{S}, \hat{A})$ are similar. During learning, the weights $w_i$ from equation 5 are updated according to the update rule in algorithm 1. We used linear function approximation in the current research as a first step and produced working mechanisms. Linear combinations are however unable to detect complex nonlinear relations between features or groups of features and the outputted value.

To cope with such complex relations an alternative solution is used in this research, replacing the linear combination with a neural network (NN). Using the feature values as inputs, the NN's single hidden layer and output node are used to approximate $Q(S, A)$. A schematic of the NN is shown in Fig. 5. After each time-step backpropagation is performed: the error calculated in algorithm 1 is propagated backwards through the NN to determine the gradient of each weight with respect to the error. The weights are thereafter updated according to the acquired gradient information [26]. The used NN features bias terms in both the input and hidden layer and is subject to $L_1$ and $L_2$ regularization [27].

*2) Feature selection:* The feature set used to approximate $Q(S, A)$ should reflect on the current state $S$ and action $A$. This reflection should distinguishably identify unique $(S, A)$-pairs whilst labeling highly correlated pairs correspondingly. Whilst

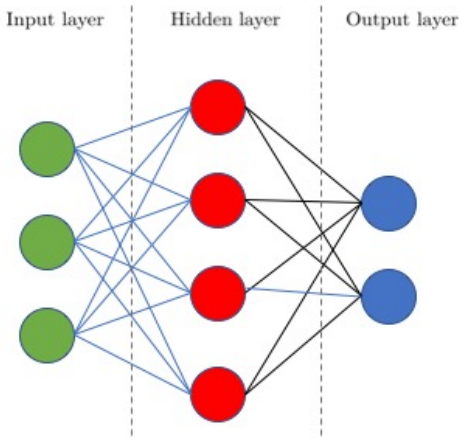



Fig. 5: Example of neural network architecture with a single hidden layer. In the proposed optimal settings the input, hidden and output layers feature respectively 314, 236 and 1 node(s) for mechanisms consisting of up to 16 links.

the former is a requirement to serve as a basis for decisions, the latter facilitates feature sets to generalize over $(S, A)$-pairs.

A major downside of using feature-based value-function approximation is the requirement for hand-crafted features. This requirement introduces a dependence on designer-insight in a context further dominated by artificial intelligence. This is particularly troublesome since the relation between the NN's input and output is in itself the subject of study. This relation is therefore not known a priori and hence cannot be used to determine which features are of significant value. To circumvent the issue of selection a wide variety of features have been used in this research. Table I presents an overview of all features. Feature selection has been trusted to the NN itself by means of regularization [28]. This self-selecting character contributes to the network's general applicability by adapting the features used to the design goal it is faced with. Hence, the initial features have been hand-crafted entirely independent from the adopted design goal.

Listed as 'Graph characteristics of $S_{t+1}$' in Table I is a large number of features based on graph theoretical characteristics. Included are each node's degree, betweenness, closeness, page rank, eigenvector, laplacian eigenvector, eccentricity and cluster coefficient as well as the graph's density, mean distance, efficiency, number of spanning trees, spectral gap, Fiedler value, radius, diameter, number of cycles, longest cycle length and shortest cycle length. All features in Table I have been scaled to fit the NN's recommended operating range between -1 and 1.

## III. Experiment

In this section we will first elaborate on the selected kinematic challenge and introduce a restriction imposed on the algorithm. Then two design goals are presented by means of which the effectiveness of the algorithm can be demonstrated. Thirdly a number of design variables and algorithmic variations will be introduced, for which optimal values have to be established using a parameter sweep and grid search. Finally a sensitivity analysis is proposed to gain insight in the relevance of the selected features.

TABLE I: Overview of all included features, their value type and number of values in each feature. Some feature sizes are dependent on the maximum number of nodes $N$ or links $L$. The total number of features equals $||f|| = 7L + 14N + \binom{N}{2} + 17$. Since the transformation probabilities in this application are uniform, ergo the outcome of action $a$ is always a known new state $S_{t+1}$, numerous features based on $S_{t+1}$ are included. For mechanisms featuring 10 nodes, 314 features are used.

| Feature | Value type | Entries |
|---|---|---|
| Operator selection ($\boldsymbol{T}$ or $\boldsymbol{D}$ ) | Boolean | 2 |
| Link is selected as target | Boolean | $L$ |
| Existence of target link | Boolean | 1 |
| Number of connections to target link | Integer | 1 |
| Number of active links in $S_{t+1}$ | Integer | 1 |
| Number of active nodes in $S_{t+1}$ | Integer | 1 |
| Link is active in $S_{t+1}$ | Boolean | $L$ |
| Node is active in $S_{t+1}$ | Boolean | $N$ |
| Relative link lengths in $S_{t+1}$ | Real | $L$ |
| Relative node angles in $S_{t+1}$ | Real | $N$ |
| Relative nodal positions in $S_{t+1}$ | Real | $2N$ |
| Shortest path betwn. each nodal pair in $S_{t+1}$ | Real | $\binom{N}{2}$ |
| Graph characteristics of $S_{t+1}$ | Miscellaneous | $8N + 11$ |
| Node is part of center in $S_{t+1}$ | Boolean | $N$ |
| Node is part of periphery in $S_{t+1}$ | Boolean | $N$ |
| Link is connected to ground in $S_{t+1}$ | Boolean | $L$ |
| Link is part of longest cycle in $S_{t+1}$ | Boolean | $L$ |
| Link is part of shortest cycle in $S_{t+1}$ | Boolean | $L$ |
| Link is part of min. spanning tree in $S_{t+1}$ | Boolean | $L$ |

### A. A game of mechanism design

Every episode starts with the same initial mechanism shown in Fig. 3. Subsequently the actor is allowed to choose an operator ($\boldsymbol{T}$ or $\boldsymbol{D}$ ) and a link number to operate on at each time-step $t$. Choices are made by performing an exhaustive sweep over all possible actions $A_t$ and $\epsilon$-greedily selecting the action with the highest predicted state-action value $Q(S_t, A_t)$. Taking an action leads to a new state $S_{t+1}$. The actor receives a reward $R_{t+1}$ for the new design described by $S_{t+1}$, calculated by the scoring module. This process repeats $n$ times until the terminal state is reached and the episode ends. $n$ equals 6 by default, but can be changed as a design variable.

### B. Restrictions and simplifications

Reinforcement learning theory shows great potential for complex learning problems, as demonstrated by others using deep neural networks [29] and actor-critic models for continuous action spaces [23], [30]. However before indulging in such advanced applications the viability of RL in mechanism design should be demonstrated, as this research attempts to do. Therefore complexity in the RL algorithm is deliberately avoided. To this end a restriction in the mechanism design game is put into place: the operators' accompanying local coordinates are not included in learning. Instead, new nodes are placed automatically such that they form an isosceles triangle with their respective target link. The resulting triangle's height is set equal to the target link's length and the new node is preferable placed in the periphery of the existing mechanism.

### C. Design goal

The current research takes on one of the oldest and best known problems in kinematic synthesis: the straight-line problem [31]. This problem serves as a useful example since several solutions to it have already been devised [32], [33]. Moreover straight-lines mechanisms have served as design a challenge for computerized synthesis before [6], [7]. This design goal therefore facilitates comparative benchmarking to prior art.

A module has been developed in order to determine a mechanism's straight-lines tracing score. The module analyses the trajectory of each node by determining the straightest section of each trajectory and fitting a minimum-surface box around the section. The aspect ratio of this box serves as a measure of the section's straightness. The final score is the product of the aspect ratio and the section's length, divided by 100 for scaling purposes.

The current method was developed independently from the design goal, and should therefore be able to synthesize other types of tracing mechanisms as well. To demonstrate this a second scoring module has been developed, pursuing to design figure-eights-tracing mechanisms. Because of the circular nature of a figure-eight, any feasible tracing mechanism should facilitate a complete circular input motion of $2\pi$ radians. Therefore mechanisms resulting in any form of infeasible domain are immediately disregarded by the scoring module. After disregarding such mechanisms, the module finds the remaining trajectories' principle axes and fits a scaled figure-eight on each one. An intermediary performance score is established per trajectory as the reciprocal of the normalized mean shortest distance between the trajectory points and the figure-eight. Finally the trajectories are searched for the characteristic figure-eight center crossing. The final score per trajectory is established based on the intermediate score and whether or not such a crossing is present. Figure 9b visualizes the scoring module's procedure for one trajectory. The lines between the blue dashed trajectory and the magenta figure-eight scatter plot represent the closest euclidean distance between the trajectories. The star is plotted to mark a detected crossing.

### D. Variations in settings and algorithms

During the experiment a number of settings with regard to the NN has been varied. These so-called hyper-parameters are the hidden layer type, learning rate $\nu$, number of hidden nodes, the $L_1$ and $L_2$ regularization rates and the selected training algorithm. Additionally tests have been performed using dropout [34] in both the input and hidden layer, which did not increase performance. Hence dropout has been omitted in further testing and experiments. Besides the hyper-parameters two other design variables have been varied: the decay rate for $\epsilon$ (exploration factor) and the level of penalty for infeasible actions and designs. The decay rates $\beta_1$ and $\beta_2$ used in the Adam algorithm are conform recommended settings [35] and kept constant throughout the experiments.

During the experiment each specific setting has been varied over its attainable values, whilst the remaining parameters have been kept constant according to the basic settings, both shown in Table II. Parallel to this parameter sweep a grid-search is conducted by varying the learning algorithm and reward scheme, resulting in four different setups (Table III). The variations A,B, C and D learn either by adopting the TD method, or by following an MC procedure. Moreover they adopt either one of two reward-schemes, accumulating rewards throughout an episode or relying solely on the episode's final reward.

TABLE II: Basic settings used during parameter sweep. In each experiment these settings are adopted by all but one parameter, which is the swept parameter. The third column lists the values included in the sweep. LReLU refers to Leaky ReLU, SGD to Stochastic Gradient Descent and SGDNM to Stochastic Gradient Descent with Nesterov Momentum.

| Parameter | Basic value | Sweep values |
|---|---|---|
| Learning rate $\nu$ | $1 \cdot 10^{-1}$ | $[10^{-3}, 10^{-2}, 10^{0}]$ |
| Hidden layer style | Sigmoid | [ReLU, LReLU, tanh [36]] |
| Hidden layer size | $\frac{3}{m4}$ of input layer size | $[0.25, 0.5, 4, 16]$ |
| Regularization | $\lambda_{L1} = 0$, $\lambda_{L2} = 0$ | $[10^{-4}, 10^{-3}, 10^{-2}]$ |
| Training algorithm | Adam [35] | [SGD, SGDNM [37]] |
| Decay rates | $\beta_1 = 0.90$, $\beta_2 = 0.99$ | [-] |
| Infeasibility penalty | 5 | $[0, 1, 25]$ |
| Exploration factor | 100 | $[30, 300]$ |

TABLE III: Combinations of algorithm choices (TD or MC) and reward structure (accumulated or end-only) are visited in a grid search. The possible combinations are denoted as A,B, C and D.

| | Temporal Difference | Monte Carlo |
|---|---|---|
| End-only reward | A | B |
| Accumulated reward | C | D |

### E. Sensitivity analysis

The use of a neural network obscures the input-output relation between the features and the $Q$-value estimation. In order to gain an understanding of the added value of each feature, several sensitivity analyses have been performed. The relative importance of the features was firstly established using the weights method, introduced by Garson [38]. This method is based on partitioning the hidden-to-output weights of each hidden node into components associated with the input-to-hidden weights [39]. Using basic arithmetic operations this method leads to a relative importance score of each feature. Secondly, Lek's [40] profiling method has been applied. In this method one feature value is kept fixed while all other inputs step from their minimum to their maximum value in a set amount of steps. The median of the resulting $Q(S, A)$-value is saved. This process is repeated throughout a range of values for the fixed feature, and subsequently repeated for all features. The results are used to compose a profile graph for each feature, indicating how the output varies with respect to the feature's value. Thirdly, a very basic approach has been used by simply taking the mean absolute input-to-hidden weight for each feature. Finally a finite-difference sensitivity analysis has been performed. During finite-difference analysis features are exposed one-by-one to a tiny increase $\delta$ and a tiny decrease

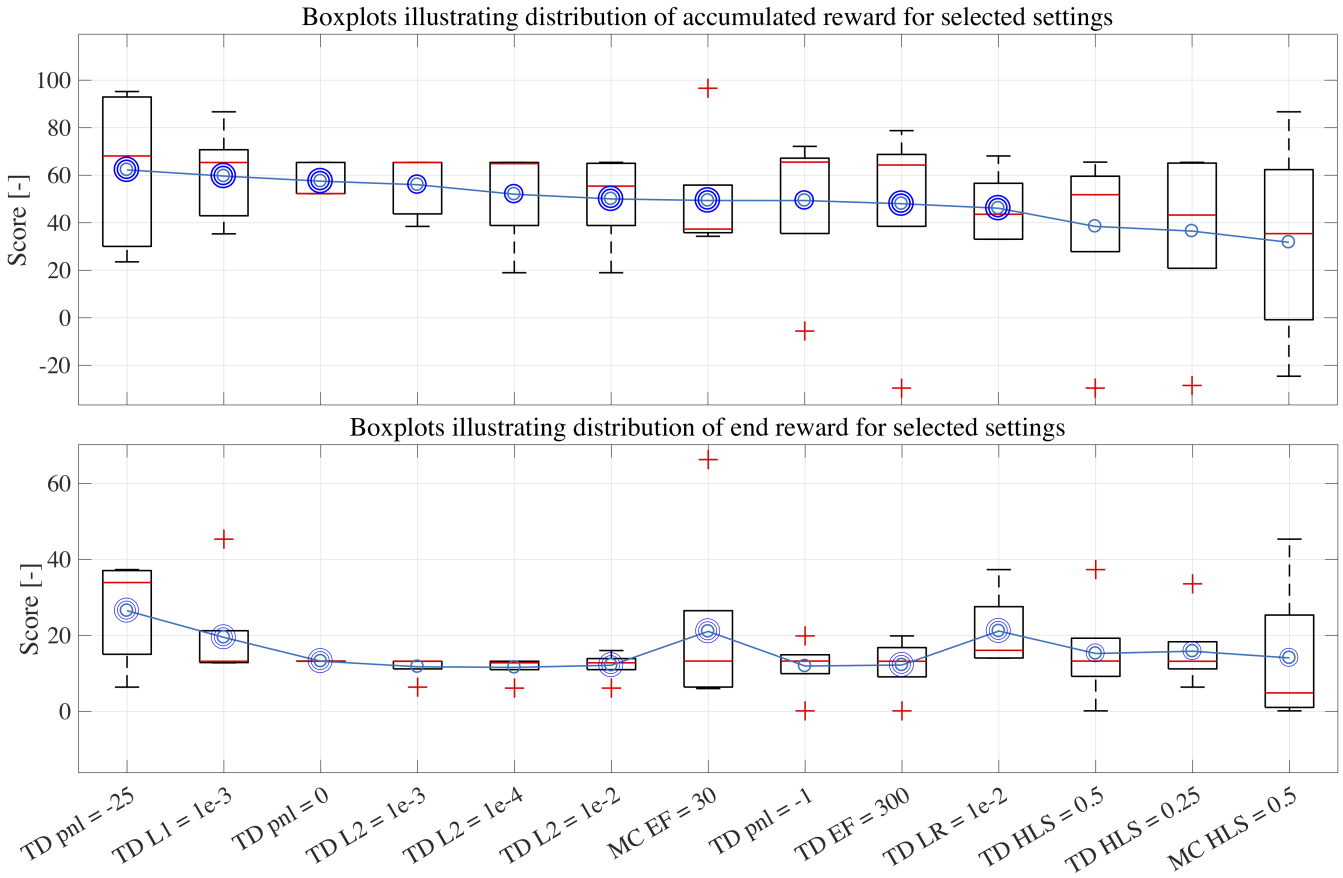



Fig. 6: Boxplot showing the reward distribution for settings that are present in at least one of the top-tens pertaining to the highest accumulated reward (top) or end reward (bottom). The setting abbreviations are TD for Temporal Difference and MC for Monte Carlo learning. Furthermore pnl = negative reward penalty, L1 = $L_1$ regularization parameter, L2 = $L_2$ regularization parameter, EF = Exploration factor, LR = Learning rate $\nu$, HLS = hidden layer size. The distributions are based on experiments in fivefold. Results are marked as outliers when deviating more than $2\sigma$ from the mean, illustrated by the red plus-signs. The mean values are represented by the circles. Triple circle markers indicate settings present in both top-tens. Double circle markers represent top-ten settings unique to the figure's corresponding reward scheme. Single circle markers represent settings scoring below top-ten.

$-\delta$, after which the NN output $O$ is evaluated. The feature sensitivity is calculated as:

$$v_f = \frac{O^+ - O_-}{2\delta}, \tag{6}$$

where $O^+$ refers to the output after a positive perturbation and $O^-$ after a negative one.

## IV. Results

### A. Results from parameter sweep

With a parameter sweep counting 28 design variations and a grid-search of 4 algorithm-reward combinations a total of 112 unique setups has been tested in fivefold for 5000 episodes. The results were averaged per setup in order to limit the influence of outliers. The reward distribution in the top-ten settings of both reward schemes is shown in Fig. 6. In both cases the highest median and mean is reached by the TD algorithm, although MC in combination with a low Exploration Factor (EF), equal to high levels of exploration, shows the ability to produce high-scoring but unrepeatable individuals. Combining the best scoring settings with the basic setup from Table II, recommended settings are extracted and shown in Table IV. Fig. 7 demonstrates convergence of one specific test run by means of a semilogarithmic plot of the mean-squared error propagation. The error converges towards zero with values stabilizing around $\sqrt{10^{-3}}$ after after some 2500 episodes. Convergence showed to deteriorate for longer games featuring more than 6 operations per episode.

### B. Resulting structures

The developed algorithm has synthesized a collection of functional designs, of which three well-performing individuals are shown in Fig. 8. The outer two structures (figures 8a and 8c) were limited to a maximum of eight nodes, whereas the structure in Fig. 8b features ten nodes. The latter design, drawing the straightest line with an aspect ratio of 1:1168, resembles Hoecken's design [41], characterized by the large structure on top of the coupler link. Interestingly, the other two

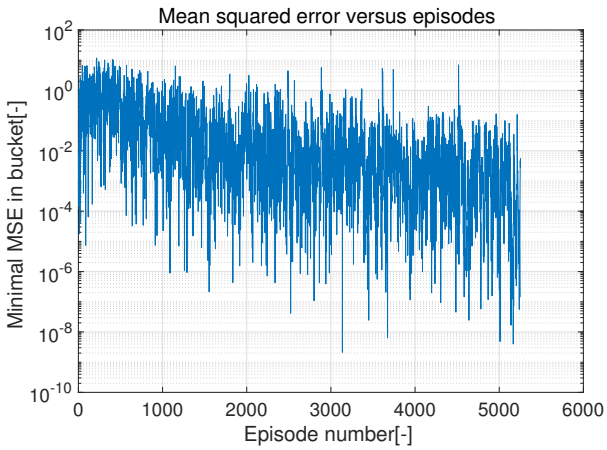



Fig. 7: Semilogarithmic plot showing the propagation of the mean squared error through a series of episodes. Random events, caused by exploration, result in high error spikes. Therefore, the minimum errors per bucket of 20 error-values are shown in this figure.

TABLE IV: Recommended settings after performing an extensive search through 112 unique setups.

| Parameter | Value |
|---|---|
| Algorithm | SARSA |
| Reward scheme | Reward after each time-step |
| Learning rate $\nu$ | $1 \cdot 10^{-2}$ |
| Hidden layer style | Sigmoid |
| Hidden layer size | $\frac{3}{4}$ of input layer size |
| Regularization | $\lambda_{L1} = 1 \cdot 10^{-3}$, $\lambda_{L2} = 1 \cdot 10^{-3}$ |
| Training algorithm | Adam |
| Decay rates | $\beta_1 = 0.90$, $\beta_2 = 0.99$ |
| Infeasibility penalty | 25 |
| Exploration factor | 30 |

structures use different approaches leading to straight lines with aspect ratios of approximately 1:410. More specifically, the structure in Fig. 8a contains a diamond shaped subset, of which the green top node passes through a straight section. While following the radial motion of the input link, the diamond folds in and out, counteracting movement perpendicular to the straight section. The same technique was used by Peaucellier in 1873 [32] in his perfect straight-line mechanism.

### C. Benchmarking

In an earlier attempt to synthesize straight-line mechanisms Lipson [6] adopted a comparable scoring method to the one presented in this paper. This allows for a quantitative comparison between the current RL approach and Lipson's evolutionary algorithm (EA). In his paper [6] Lipson describes most of the resulting structures exceeded aspect ratios of 1:1000, with outliers as high as 1:28340. The results in Fig. 8 show aspect ratios of respectively 1:415, 1:1168 and 1:406. Therefore the straightness in the current results do not challenge the levels of straightness achieved by Lipson. Computational times are difficult to compare because of technological advances in the last ten years, but do remain within the same order of magnitude, i.e. approximately 10 hours.

The contrast between these results is mostly due to three discrepancies in the comparison: the restriction introduced in this paper, differences between the objective functions used and the issue of numerical inaccuracy during kinematic simulation. First of all a restriction was introduced in section III-B. More specifically the relative coordinates introduced by Lipson were omitted and replaced by an automatic choice of location for new nodes. Hence the current solution is not as free in its design choices as Lipson's, therefore ruling out a large number of possible configurations. Secondly the current solution evaluates a trajectory on both straightness and length, whereas Lipson's solution adheres to the aspect ratio only. Thus a discrepancy exists between the algorithms' optimization goals, complicating the comparison. Thirdly minuscule inaccuracies in the calculated trajectories, through numeric simulation, may disturb an otherwise high aspect ratio. To test this hypothesis the current numeric simulator was used to determine the aspect ratio of a Peaucellier machine's trajectory, which is known to be perfectly straight. Upon analyzing the trajectory an aspect ratio of 1:1947 was obtained. This indicates the simulator's accuracy is insufficient to measure aspect ratios in this extreme region.

Because of these three reasons no convincing verdict can be given about the optimal algorithm for kinematic synthesis.

### D. Sensitivity analysis

The use of four different sensitivity analyses, as described in III-E, has resulted in four slightly different results. There are however five features that belong to the ten most sensitive features in all analyses. These are shown, in arbitrary order, in Table V.

TABLE V: Most sensitive features with respect to $Q$.

| Feature | Relation to $Q$ |
|---|---|
| Existence of target link | Positive |
| Link 6 is selected as target | Positive |
| Link 4 is selected as target | Negative |
| Link 7 is part of longest cycle | Positive |
| Relative vertical position of node 4 | Negative |

The order of merit between these features varies over the different analysis methods and is therefore not included. Nonetheless, the table yields interesting results. The strong positive relation found between $Q$ and the existence of the target link is rather trivial, since actions on non-existing links lead directly to an infeasibility penalty. The next two however are peculiar: apparently operating on link 6 leads to increased results, whereas operations on link 4 are negatively related to the expected reward. These sensitivities have no obvious explanation: while both Fig. 8a and Fig.8b are the result of an operation on link 6 , they started out with an operation on link 4.

### E. Generality of algorithm

The learning algorithm has been tested using a second design goal as presented in III-C, utilizing the same settings

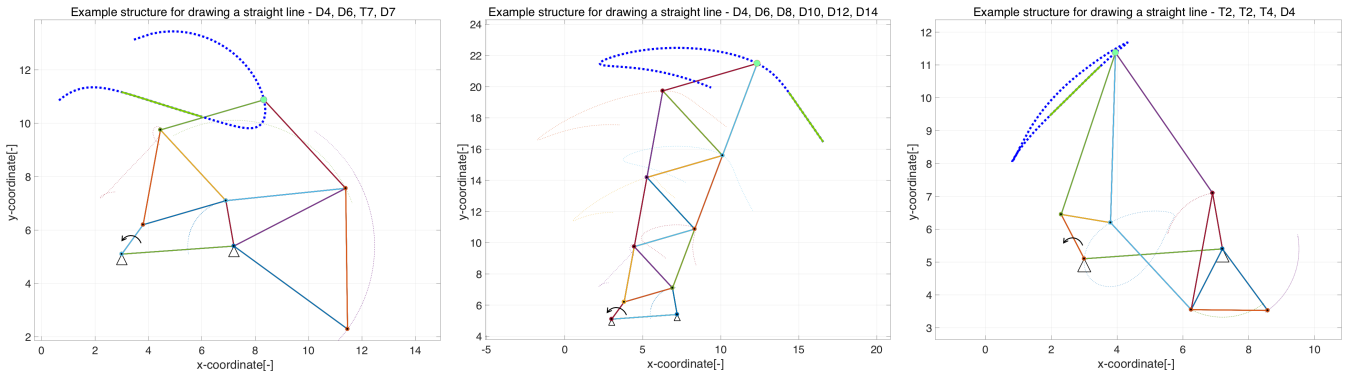



Fig. 8: Three structures as created by the algorithm. The thick, dark blue dashed lines represent the trajectories featuring a straight section, highlighted by a full green line. The large green nodes represent each structure's straight-line tracing node. The aspect ratios of the straight sections are (a) 1:415, (b) 1:1168 and (c) 1:406.

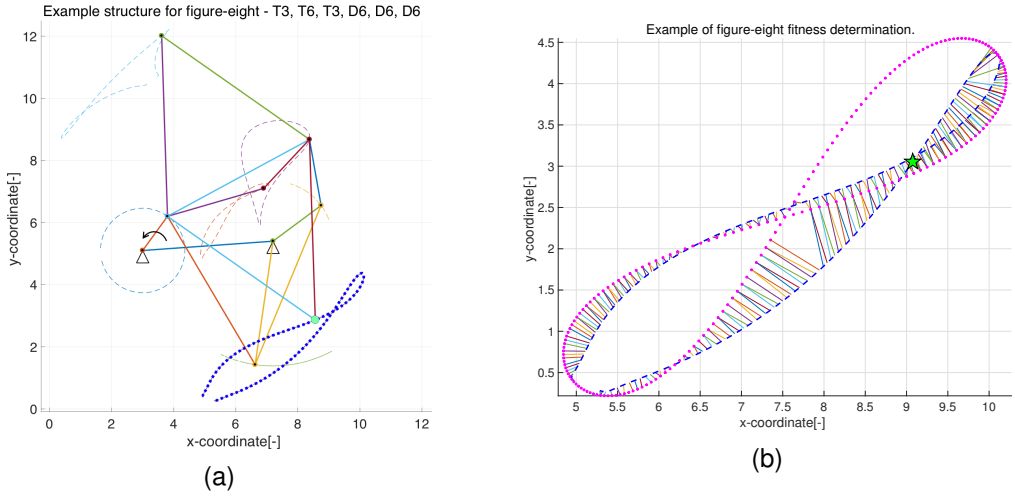



Fig. 9: Example of the application of the proposed algorithm adopting a figure-eight design goal. Figure (a) shows the resulting structure, in which the green node (lower-right corner) traces a blue thick dashed figure-eight pattern. Figure (b) shows how the trajectory is compared to a true figure-eight. The colored lines display the shortest distance between each trajectory point and the true figure-eight. In this case, the mean euclidean distance equals 0.203. The green star indicates a detected crossing.

and features used to synthesize straight-line mechanisms. The resulting design is shown in Fig. 9a. The depicted mechanism clearly traces a figure-eight, albeit curved. Even though the imposed restrictions do not allow for the creation of a perfect figure-eight trajectory, the results demonstrate the algorithm's capability of adapting to other design goals.

I proof I prove

## V. Discussion

The results presented in this paper prove that RL can be applied to synthesize straight-line mechanisms. Therefore this demonstration broadens the mechanical system design spectrum of tools and gives future researchers an extra angle to consider whilst choosing a fitting synthesis technique. Moreover, by successfully applying the presented method on a second design goal, the method's general applicability to mechanism design has been indicated. The current research may serve as a stimulant for the engineering world to start adopting machine learning into its range of design tools.

Zooming out, one sees how this paper opens the door to a new area of application for the heavily researched field of RL. DeepMind's David Silver [42] recently said: "Now we can start to tackle some of the most challenging and impactful problems of humanity", after having presented a breakthrough in RL. Such an announcement indicates a movement in the RL community towards the application of its skills to other research fields. By demonstrating the viability of RL in mechanical engineering this research removes a barrier for scientists like Silver to apply their craft to mechanical system design.

Extensions on the current results could be made by removing the imposed restrictions on Hod Lipsons's [6] $\boldsymbol{T}$ and $\boldsymbol{D}$ operators. A first step would be to extend the actor's freedom in choosing the location of new nodes. A paramount improvement however could be made by adopting a learning algorithm capable of coping with a continuous action space. Such algorithms

(policy gradient [29], actor-critic models [23], [30]) do not use an `argmax` operator to perform greedy action selection, but instead estimate a stochastic reward distribution over the action space and select an action accordingly, decoupling computational time from the size of the action space. The resulting algorithm could search a continuous design space from which it may obtain designs that achieve more accurate performance. Secondly the value function approximator could be improved by further researching the ideal set of features. The presented result from the sensitivity analyses has introduced only limited insight in the inner workings of the network. The effect of individually important features may have been outweighed by a combined effort from less important features. Also, $Q$ might be very sensitive to certain groups of features moving together. Unfortunately both of these phenomena cannot be detected by the applied analysis methods. As a first step towards feature selection however, the presented methods can be used to remove obsolete features, increasing the algorithm's efficiency. Besides the issues concerning feature selection, the neural network's layer depth may be extended. Such a deep neural network would be able to detect more complex relations between sets of features and the expected reward, leading to increased algorithm performance. Finally some post-processing steps may be developed to clean up the algorithm's results. For example, post-processing could remove obsolete triangles or adapt dimensions to improve aesthetics and producability of the designs. The resulting software may be directly applicable to the design of mechanisms for specific kinematic goals, whether it be tracing trajectories, amplifying motion or other kinematic challenges.

## VI. Conclusion

The current research demonstrates the added value of reinforcement learning in mechanism design, which has thus far been uncultivated ground. In this paper a method is presented by which kinematic synthesis can be molded into a game-like process compatible with reinforcement learning algorithms. Building on previous research by Lipson [6] the presented process starts out with a basic four-bar mechanism on which a series of operations can be performed. After each operation the resulting design's performance is evaluated and compared to the design goal. This experience serves as input for the reinforcement learning algorithm whilst slowly converging towards a reliable state-action value-function approximation.

This research has explored the use of value-function approximation by a neural network to predict the performance of a kinematic mechanism. Herein fertile groundwork is laid by proving this concept, although proper feature selection and experimentation with the neural network architecture may further enhance the presented results.

By means of an extensive parameter search a list of recommended algorithm and hyper-parameter settings has been devised and a series of straight-line mechanisms has been successfully and independently produced by the developed algorithm. Although not drawing as perfectly straight as Peaucellier's exact solution or Lipson's designs, the resulting designs are unmistakably straight-line mechanisms and in specific cases adopt the working principles behind Peaucellier's [32] and Hoecken's [41] mechanisms.

Quantitative benchmarking showed room for improvement on the part of the design accuracy of the proposed algorithm. However the imposed restriction on the algorithmic design freedom impairs the fairness of comparison, as did small inaccuracies of the obtained kinematic simulation results.

Finally, using a second design goal, a demonstration was given of the algorithm's adaptability. The successful handling of a second design goal indicates the algorithm's potential as a general solution for a variety of kinematic synthesis challenges.

In future work the applied SARSA algorithm can be upgraded to an actor-critic or policy gradient model, effectively alleviating the necessity of the current restrictive measures on the nodal placement and thereby increasing the algorithms design freedom. Combined with further work on feature selection and neural network architecture, reinforcement learning should be regarded as a promising means for the synthesis of mechanical systems.

## References


[1] P. Dollar, Z. Tu, and S. Belongie, "Supervised learning of edges and object boundaries," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 1964–1971.

[2] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 871–882, 2013.

[3] Q.-H. Ye, L.-X. Qin, M. Forgues, P. He, J. W. Kim, A. C. Peng, R. Simon, Y. Li, A. I. Robles, Y. Chen *et al.*, "Predicting hepatitis b virus–positive metastatic hepatocellular carcinomas using gene expression profiling and supervised machine learning," *Nature medicine*, vol. 9, no. 4, pp. 416–423, 2003.

[4] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, "The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 3238–3243.

[5] L. Yin and G. K. Ananthasuresh, "Design of Distributed Compliant Mechanisms," *Mechanics Based Design of Structures and Machines*, vol. 31, no. 2, pp. 151–179, 2003.

[6] H. Lipson, "Evolutionary synthesis of kinematic mechanisms," *AI EDAM*, vol. 22, no. 03, pp. 195–205, aug 2008.

[7] P. Kuppens, "Automated Robot Design With Artificial Evolution," Msc. Thesis, Delft University of Technology, 2016.

[8] H. Zhou and K.-L. Ting, "Topological Synthesis of Compliant Mechanisms Using Spanning Tree Theory," *Journal of Mechanical Design*, vol. 127, no. 4, p. 753, 2005.

[9] A. Saxena, "Synthesis of Compliant Mechanisms for Path Generation using Genetic Algorithm," *Journal of Mechanical Design*, vol. 127, no. May 2010, p. 745, 2005.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[11] L. Okdinawati, T. M. Simatupang, and Y. Sunitiyoso, "Multi-agent reinforcement learning for collaborative transportation management (ctm)," in *Agent-Based Approaches in Economics and Social Complex Systems IX*. Springer, 2017, pp. 123–136.

[12] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[13] V. Rieser, D.T. Robinson, D. Murray-Rust, M. Rounsevell, "A comparison of genetic algorithms and reinforcement learning for optimising sustainable forest management," in *Proceedings of the 11th International conference on GeoComputation*, 2011.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[15] K. M. Vermeer, "Literature Survey: Automated Design of Compliant Mechanisms," Literature Survey, Delft University of Technology, 2017.

[16] R. Avilés, A. Hernández, E. Amezua, and O. Altuzarra, "Kinematic analysis of linkages based in finite elements and the geometric stiffness matrix," *Mechanism and Machine Theory*, vol. 43, no. 8, pp. 964–983, 2008.

[17] R. Battiti, "First-and second-order methods for learning: between steepest descent and newtons method," *Neural computation*, vol. 4, no. 2, pp. 141–166, 1992.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learn: An Introduction*, 2nd ed. The MIT Press, 2012.

[19] F. S. Melo and M. I. Ribeiro, "Q-Learning with Linear Function Approximation," in *Learning Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 1, pp. 308–322.

[20] A. A. Markov, "The theory of algorithms," 1953.

[21] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fund. Math*, vol. 3, no. 1, pp. 133–181, 1922.

[22] C. Szepesvári, "Algorithms for Reinforcement Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[23] H. van Hasselt and M. A. Wiering, "Reinforcement Learning in Continuous Action Spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, no. Adprl. IEEE, apr 2007, pp. 272–279.

[24] A. Perez-Uribe and E. Sanchez, "Blackjack as a test bed for learning strategies in neural networks," in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, vol. 3, no. 9. IEEE, nov 2009, pp. 2022–2027.

[25] O. Abul, F. Polat, and R. Alhajj, "Multiagent reinforcement learning using function approximation," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 485–497, 2000.

[26] R. Hecht-Nielsen *et al.*, "Theory of the backpropagation neural network." *Neural Networks*, vol. 1, no. Supplement-1, pp. 445–448, 1988.

[27] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," *Twenty-first international conference on Machine learning - ICML '04*, p. 78, 2004.

[28] J. Z. Kolter and A. Y. Ng, "Regularization and Feature Selection in Least-Squares Temporal Difference Learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 521—-528.

[29] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[31] V. Tarabarin, Z. Tarabarina, and D. Chirkina, "Designing, Analysis and Computer Modeling of Straight-Line Mechanisms," in *Explorations in the History of Machines and Mechanisms*, 2012, pp. 551–563.

[32] C. Peaucellier, "Note sur une question de geometrie de compas," *Nouv. Ann. der Math*, vol. 12, pp. 71–81, 1873.

[33] E. A. Dijksman, *Approximate straight-line mechanisms through four-bar linkages*. Editions de l'Académie de la Republique Socialiste de Roumanie, 1972.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[35] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations*, San Diego, 2015.

[36] S. Y. Fei-Fei Li, Justin Johnson, "Lecture 6: Training neural networks, part 1 - stanford university cs231n," Lecture slides, 2017, available on http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf , visited on November 22nd 2017, Stanford University.

[37] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[38] D. G. Garson, "Interpreting neural network connection weights," *AI Expert*, vol. 6, no. 7, pp. 47–51, 1991.

[39] M. Gevrey, I. Dimopoulos, and S. Lek, "Review and comparison of methods to study the contribution of variables in artificial neural network models," *Ecological modelling*, vol. 160, no. 3, pp. 249–264, 2003.

[40] S. Lek, A. Belaud, P. Baran, I. Dimopoulos, and M. Delacoste, "Role of some environmental variables in trout abundance models using neural networks," *Aquatic Living Resources*, vol. 9, no. 1, pp. 23–29, 1996.

[41] S. Lu, D. Zlatanov, X. Ding, and R. Molfino, "A new family of deployable mechanisms based on the hoekens linkage," *Mechanism and Machine Theory*, vol. 73, pp. 130–153, 2014.

[42] DeepMind YouTube Channel, "Alphago zero: Discovering new knowledge," 2017.

# Chapter 3

# Closing remarks

In the introduction a journey was presented, initiating with a broad interest in the automation of mechanism design and turning into a research goal pertaining to the synthesis of compliant mechanisms. The subsequent literature study contained research into three important and individually treated subjects, consciously looking for separate solutions to each sub-problem. As a result solutions applied in distant research fields have been embraced in the context of mechanism synthesis, leading to unexpected and exciting possibilities. The literature survey therefore served as an important stepping stone towards the thesis subject and final results.

After extracting a thesis subject from the recommendations in the literature survey, three goals have been introduced. The first goal answers to the challenge of fitting the seemingly incompatible worlds of mechanism design and reinforcement learning together. To achieve this goal a framework is presented in which mechanism design is casted in a game-like mold. Played as a game, kinematic synthesis fits right into the niche of reinforcement learning problems. The presented paper clearly demonstrates the viability of this concept by performing successful kinematic synthesis using the framework presented.

Secondly a learning algorithm is required in order to be able to learn within the bounds of the developed framework. Using the SARSA algorithm (algorithm 3 in appendix A-2-2), Temporal Difference Learning has been implemented, optimizing the neural network weights of a value-function approximator.

Finally the viability and succesfull implementation of the developed algorithm has been demonstrated on two different design goals. Using a second design goal not only the viability but also the general applicability of the developed algorithm has been demonstrated. To summarize:

- ✓ Posing the design problem
- ✓ Creating a learning algorithm
- ✓ Demonstrate the algorithm

By achieving all imposed goals, this thesis and the current research may therefore be considered a success.

For those readers interested in a level of detail beyond the scope of the presented paper a number of appendices is included in this thesis report (Appendix B to F), each supporting a different section of the paper as the titles indicate. Additionally a thorough overview of the field of machine learning is included in appendix A, as well as the full literature review in appendix G. It would however be a shame to merely look inwards to this thesis, whereas it is the outside world that may benefit from this research. Let us therefore zoom out and reflect on the relevance of this thesis work within the scientific context.

The current research has demonstrated the value machine learning, more specifically reinforcement learning, may have on mechanism synthesis. From literature study it was shown that by 'forgetting' the mechanical engineering context of this research, machine learning is rather an obvious choice for the subgoal of optimization or learning. Perhaps the presented demonstration is a first step towards loosing the need to 'forget', as it shows why machine learning and mechanical engineering can actually be united quite easily. The adoption of machine learning by the mechanical engineering community serves its best interests as smart design tools will be of great supportive value to the design of advanced mechanisms.

The presented research could induce a likewise movement in the opposite direction: having proven its viability, specialized machine learning researchers may now enter the mechanical domain to apply their knowledge and skill. In a recent video presentation [49] David Silver, DeepMind's lead reinforcement learning researcher, said: "Now we can start to tackle some of the most challenging and impactful problems of humanity". He spoke these words after presenting a breakthrough in reinforcement learning, indicating his eagerness to start applying artificial intelligence's capabilities to other research fields. By demonstrating the viability of reinforcement learning in mechanical engineering, the way is paved for researchers like Silver to explore the "impactful" domain mechanical system design yields.

Further research may extend the current work by adopting advanced algorithms, like policy gradients [50], or actor-critic models [51,52]. Such algorithms do not rely on an $\epsilon$-greedy policy and therefore do not require an exhaustive search through all possible actions. Instead, these algorithms produce a probability distribution over all actions, which can easily cope with a continuous action space. Adopting such an action space the restrictions imposed on Lipson's operators in the current research are removed. The resulting algorithm would have full freedom of placing new nodes wherever it sees fit, greatly enhancing its creative capabilities.

If such and other improvements could be made, as proposed in the paper, the result would be a pragmatic design tool for everyday use. Such a tool could serve mechanical engineers by generating a number of functioning mechanisms as a starting point for complex design challenges. Additionally, this synthesis tool could be used as a turnkey application for the start-to-end design of simple mechanisms. Such an application would make engineering and designing accessible for the majority of people who have not had formal engineering training. Such a development would strongly support the ongoing 3D printer-induced trend towards decentralized manufacturing.

Of course one can only speculate about the future impact of machine learning on mechanism design. However, the results rendered by this research outline an artificial-intelligence aided future in which engineering innovations will emanate from joint efforts of both man and, increasingly so, machine.

# Appendix A

# Background: Machine Learning

The field of Machine Learning (ML) originates from computer-science and has started to develop in the 1950s with the first neural network, built by Minsky and Edmonds, and Arthur Samuel's checkers playing machine [5], demonstrated on television in 1956 (figure G-11). In those early days of the field, computer power was still limited and the true potential of many breakthrough articles did not become apparent until decades later. Slow progress and disappointing results amounted to severe criticism in and on the Artificial Intelligence (AI) community. Starting in the 1970s, a period of limited funding and progress initiated, which has later been coined the "AI Winter", lasting until the start of the 20th century. In the last decades, computational power has grown exponentially and concepts like deep learning have become feasible. With increased feasibility, AI has become a highly active research field as well as an industry standard in data-driven companies.

**Figure A-1:** A 1956 television demonstration of Arthur Samuel's Checkers program, played against by Robert Nealey on the IBM 7094. Nealey lost from this milestone program. Image courtesy to IBM.

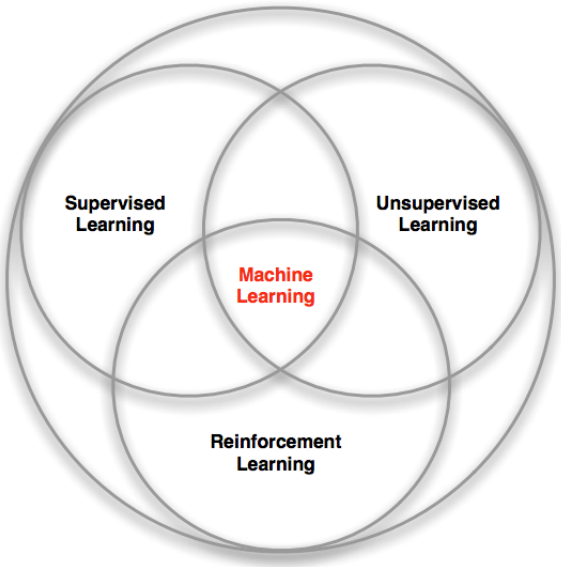



**Figure A-2:** Schematic showing the three categories that ML comprises of. Image reproduced from [53].

## A-1 Machine Learning categories

Within ML a distinction is made between several different concepts. These concepts can be generally divided into three categories, shown in figure A-2. These categories are supervised, unsupervised and reinforcement learning. To get a fundamental understanding of the ML field, each of these categories will be discussed briefly.

### A-1-1 Supervised Learning

In Machine Learning, most applications are based on learning by experience. In such applications a machine trains by internalizing many examples, consisting of inputs and their according outputs. During training, the machine develops a model of the relation between inputs and outputs it witnesses in the examples. Subsequently, the model can be used to predict outputs for new input data.

This type of training, requiring a large dataset of data with known inputs and corresponding outputs, is called supervised learning. Such complete datasets are known as labeled data. Examples of supervised learning are ample in image detection [54], traffic-flow predictions [55] and medical predictions [56]. The most important algorithms in supervised learning are Support Vector Machines (SVM), Neural Networks (NN), Gaussian Process Regression (GPR), Random Forests and linear classifiers like regression models.

Generally, a labeled data set is split in two parts: a training set and a testing set. During supervised learning, the algorithm is trained on the training set and subsequently tested on the test set. Since the test true outputs are known, the algorithm's accuracy can be evaluated.

### A-1-2 Unsupervised Learning

As the name indicates, this second type of ML is similar to the first category, except for the loss of data labels, or supervision. Jain [57] distinguishes unsupervised learning by the absence of category information, or labels, in the target data set. Even with the limited remaining information, unsupervised learning algorithms are able to detect patterns and describe hidden structures in the data. Therefore, unsupervised learning is mainly used for data clustering and anomaly detection. Data clustering alone is being applied to a broad range of topics [57], ranging from character recognition in handwriting [58] to customer segmentation for marketing purposes [59]. A well-known example of an unsupervised leaning algorithm is the K-means clustering algorithm, which aims to segment a dataset in such a way that every data-point is considered to be part of the cluster with the nearest mean.

### A-1-3 Reinforcement Learning

Reinforcement learning (RL) is described by Kaelbling et al. [34] as behavioral learning by trial-and-error interactions with a dynamic environment. Sutton [60] gave an even more formal definition: 'Reinforcement learning is the learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal'. RL can be applied when no labeled data is available, but outputs can be given a measure of success. In such a case, a machine can learn by taking actions, assessing the results and adapting its perceptions accordingly. Through iteration, the machine can learn to take the right actions resulting in the highest rewards.

Rewards play an important role in reinforcement learning. The machine's goal is to maximize its expected cumulative reward over (pseudo)time. Machine's therefore take into consideration the effect of their current move on their cumulative reward by making predictions of future rewards [61]. Instantaneous rewards are usually valued more than possible future rewards, which is referred to in literature as the problem of delayed rewards [62]. The trial-and-error search and delayed rewards are the two most distinguishing features of reinforcement learning [60] and make it suitable for sequential decision making.

Because of its distinctive character, RL finds applications in different types of fields than (un)supervised learning. Applications range from learning a robot to walk [63] to playing ATARI games [64] and minimizing elevator waiting time [65].

### A-1-4 ML and generative design

Crudely, one can say that both supervised and unsupervised learning require large datasets to train on, whereas reinforcement learning relies on experience gained from interaction with its environment. When designing mechanisms, reliance on datasets can be consider bad practice for two reasons:

- Datasets are not readily available. Atlases, containing large numbers of mechanisms, have been developed [66], but performance indicators are still to be determined for every single design. As such, creating a sufficiently large labeled dataset of mechanisms and their performance is an arduous if not impossible task. Also, performance labels are

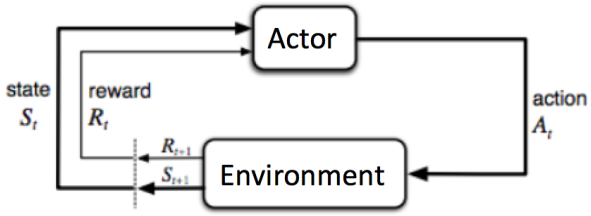



**Figure A-3:** Schematic showing the cyclical life of an RL actor. Image reproduced from [67].

dependent on the formulation of the goal, and therefore new labels are required for every new design goal.

- The necessity of using ML to perform design task is born out of limitations in human design capacity. It would therefore be unwise to train a machine on designs resulting from the same limited human designer. Doing so decreases the chance of generative design algorithms to come up with truly original designs, which can be seen as a goal in its own right.

If reliance on datasets is not considered valid, both supervised and unsupervised learning are ruled out, leaving only Reinforcement Learning as a valid option. As mentioned, RL requires an environment to interact with in order to gain experience and learn. For mechanism design environment interaction can be regarded as prototyping: materializing a design in order to assess its performance. Of course, physical models are not mandatory in this type of prototyping: using computational simulations performance indicators can be gained. By feeding these indicators back into the RL algorithm as a reward signal, all prerequisites are in place to create a generative design algorithm.

## A-2 Reinforcement Learning

In RL, the learner is named the *actor*. The actor gains experience through taking *actions* that lead to interactions with its *environment*. After every interaction, the actor is faced with a new situation, or *state*, from which it may perform the next action. As part of the interactions, the actor may receive *rewards*. The actor's goal is to choose actions such that it maximizes the total reward. In a more formal manner, one can say that for every timestep $t$ the actor is in state $S_t$. After taking action $A_t$, it ends up in state $S_{t+1}$ and receives a reward $R_{t+1}$. Over time, the actor's total return $G_t$ equals the sum of all future rewards up to the final time-step $\tau$:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_\tau \tag{A-1}$$

In most applications of RL, it is desirable to gain these rewards as fast as possible. Therefore, a discount factor $\gamma$ is introduced, ranging between 0 and 1. $\gamma$ can be seen as an interest rate, making rewards earned now more valuable than rewards earned in the future. Taking the $\gamma$ into account, equation A-1 becomes:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{A-2}$$

In each time step, the agent has to choose an action based on the state its in. To do so, the agent implements a mapping $\pi_t$: $S \rightarrow A$. This mapping is called the *policy* and $\pi_t(a|s)$ describes the probability of taking action $a$ at timestep $t$ in state $s$.

### A-2-1 Markov Decission Processes

RL problems are usually posed in the form of a Markov Decision Process (MDP) [68]. MDP's are denoted with a tuple $(S, A, P, \gamma, R)$ containing five elements:

- $S$: a set of all possible states in which the actor may be.
- $A$: a set of actions the actor can take.
- $P$: the state transition probabilities. $P_{sa}(s')$ gives the probability of transitioning to state $s'$ by performing action $a$ in state $s$.
- $\gamma$: the discount rate for future rewards.
- $R$: the reward function.

Furthermore, MDPs describe stochastic processes that comply with the Markov property of being memoryless. This means that probability distributions of future states are only dependent upon the current state and not on any previous states.

In an MDP, the actor starts out in a state $s_0$ and takes an action $a_0$. It transitions to state $s_1$ under probability $P_{s_0 a_0}$. From this new state, it can take a new action and keep repeating this cycle. In each state, the actor receives a reward $R(s)$. The expected accumulative reward is subject to devaluation over time and is described by equation A-2.

In order to maximize the total reward, RL algorithms need to be able to estimate the value of $G_t$ give a certain state or action. This estimation serves as an input in determining whether certain states are desirable to be in, or which actions will lead to the highest rewards. These estimations are called *Value Functions*. Given a certain policy $/pi_t$, the state-value function $V_{/pi_t}(s)$ equals the expected reward of being in state $s$ and following policy $/pi_t$ until termination:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right] = \mathbb{E}_{\pi_t} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \tag{A-3}$$

Since one is often concerned with choosing the right action, it is common practice to include the current choice of action the value function as well, leading to the state-action-value function:

$$Q_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_{\pi_t} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \tag{A-4}$$

Both value functions have a recursive character:

$$V_\pi(s) = \mathbb{E}_{\pi_t} \left[G_t | S_t = s\right] = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \tag{A-5}$$

Taking into account the state-transition probabilities $P$, one can write equation A-3 as:

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s') \tag{A-6}$$

This is the *Bellman equation*, which considers all possible new states $s'$ given the current state $s$ and the probability distribution according to policy $\pi$. The value function $V_\pi$ is the unique solution to its Bellman functions, and can be found using the recursive character of equation G-6. The right-hand side of equation G-6 can be seen as a mathematical operator $T^\pi$, mapping $V^\pi(s')$ to $V^\pi(s)$. The fixed point theorem [69] shows that $V^\pi$ can be found by iteratively applying operator $T^\pi$ to a initial value function $V$. For a more exhaustive derivation of the Bellman equations, the reader is referred to [67, 68, 70] .

In RL, the objective is to find a policy $\pi^*$ that maximizes the value function $V(s)$ by prescribing the optimal action $a$ for each state $s$, resulting in $V^*(s)$. Again, $V^*(s)$ can be derived by applying $T^*$ iteratively to an initial value function $V(s)$. Once $V^*(s)$ is known, $\pi^*$ can be computed. This method is coined value iteration [70].

Alternatively, one can apply a random initial policy $\pi$ and iteratively update the policy until the optimum is found: policy iteration. In practice, value iteration is most commonly applied [68].

### A-2-2 Model-free MDPs

Both value iteration and policy iteration are methods that can solve a known MDP. However, a mechanism design challenge typically does not have a known MDP: the actor does not know how its environment works and has to find out on the go. This type of problem is called *model-free*, referring to the fact that no accurate model of the environment is given. In fact, most real-life applications of RL are based on model-free algorithms and such algorithms have become the main reason why RL is being used. Without fully knowing the MDP, it is impossible to solve it exactly. It is however possible to estimate the value function of an unknown MDP. Two of the most important algorithms for doing so are presented here: Monte Carlo learning and Temporal Difference learning.

#### Monte Carlo Learning

In most RL cases, one can distinguish *steps* from *episodes*. A step is simply going from $t$ to $t+1$ by taking an action $a$ and transitioning from $s$ to $s'$. In many cases however the number of steps is limited by the environment. Take for instance the game Mario: an actor playing a level of Mario can keep performing actions until it reaches a terminal state: either reaching the end of the level or dying. Such a streak of actions is called an episode. During an episode, the actor visits numerous states and may even visit certain states multiple times. In Monte Carlo (MC) methods, value estimates and policies are updated after every full episode.

The value of a state is the total expected return starting from that specific state. Therefore, the simplest way of estimating this value per state is to simply average the results of all episodes in which this state is visited. Sutton and Barto [67] captured the algorithm in the following pseudo-code:

By following Algorithm 1, one can estimate the value function for all states $s$ given the policy $\pi$. However, in practice one is most often interested in finding the optimal policy $\pi*$ for which the reward is maximized. To find out which action $a$ to take in a given state $s$, one can use the state-action-value function $Q(s, a)$. Using a greedy policy, the actor always chooses the

**Algorithm 1:** Monte Carlo policy evaluation from [67]

```
Initialization;
    π ← policy to be evaluated;
    V ← an arbitrary state-value function;
    Returns(s) ← empty list for all s;
while not converged do
    Generate an episode under policy π;
    foreach s do
        G ← return following the first occurance of s;
        Append G to Returns(s);
        V(s) ← average(Returns(s))
```

action $a$ with the highest state-action-value, simply be trying out al actions and perceiving the resulting state-action-values. Selecting the action with the highest state-action-value equals the following policy:

$$\pi(s) = \underset{a}{\text{argmax}} Q(s, a) \tag{A-7}$$

If one can find a good estimate for the state-action-value function, the optimal policy follows out of equation A-7. As a result of using $Q(s, a)$ instead of $V(s)$, the algorithm will have to assign rewards to state-action pairs, instead of over states. The state-action-value of a state-action pair equals the average reward of all episodes in which the state-action pair was visited.

**Algorithm 2:** Monte Carlo control from [67]

```
Initialization;
    π ← arbitrary;
    Q ← arbitrary;
    Returns(s, a) ← empty list for all s and a;
while not converged do
    Play an episode under policy π ;
    foreach s, a pair in the episode do
        G ← return following the first occurance of s, a;
        Append G to Returns(s, a);
        Q(s, a) ← average(Returns(s, a))
    foreach s in the episode do
        π(s) = argmax_a Q(s, a)
```

Convergence for a MC control algorithm along the lines of 2 is considered inevitable [67], but has not been proven formally.

### Temporal Difference Learning

In Monte Carlo learning, the policy is not affected until after a full episode has been completed. A more efficient way of learning is introduced in the Temporal Difference (TD) Learning method, which can update policies before terminating the episode. The practice of updating estimates before ending the episode can significantly increase an algorithm's convergence rate.

Since TD methods do not wait until the end of the episode, they cannot use the return $G$ to update. Instead, TD relies on the recurrent character of the value function (equation A-5) to establish the update. The following holds true for the recurrent value function $V(s)$:

$$V(S_t) = R_{t+1} + \gamma V(S_{t+1}) \tag{A-8}$$

Since $V(s)$ is estimated in model-free RL, equation A-8 will not hold as long as the true function is not found. After each time step, the error between the two sides of the equation can be determined and $V(S_t)$ can be adjusted towards equilibrium. The update is usually performed according to equation A-9, using a step size parameter $\alpha$ to stabilize the algorithm.

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{A-9}$$

Note that the only 'ground truth' in equation A-9 is $R_{t+1}$, since it comes directly from the environment. This makes the right hand side of equation A-8 slightly more trustworthy than the left hand side, hence the updates adjust the left hand side value $V(S_t)$. Overall, this slowly converges towards the true value function $V(s)$.

The practice of updating estimates on the basis of other estimates is called bootstrapping. The heavy reliance on estimates gives a feeling of insecurity and intuitively gives rise to questions about divergence. Fortunately, Sutton already proved the convergence of the basic TD algorithm in 1988 [71]. His proof was extended for the general case of TD algorithms in 1992 by Dayan [72].

The previous section has only considered prediction methods, focusing on the value function $V(s)$. In a similar fashion to MC learning this can be extended towards TD control by replacing the state-value function by the state-action-value function $Q(s, a)$. This means the update rule (equation A-9) becomes:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{A-10}$$

So to perform this update, the information in tuple $< S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} >$ is required. The algorithm that performs this type of update is therefore coined *Sarsa*. The outline of the Sarsa algorithm, as proposed in [67], is shown in algorithm 3.

### TD($\lambda$): the middle ground

In the previous section, it was shown that TD learning updates after every step based on the return $R_{t+1}$ in equation A-9. Of course it would also be possible to update after two steps, such that the update becomes:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) - V(S_t)\right] \tag{A-11}$$

**Algorithm 3:** Sarsa using TD learning from [67]

Initialization;
    $Q \leftarrow$ arbitrary;
**foreach** *Episode* **do**
    Initialize $S$;
    Choose $A$ from $S$ using $Q$ and a greedy policy;
    **foreach** *Step in the episode* **do**
        Take action $A$, receive $R$ and $S'$;
        Choose action $A'$ from $S'$ using $Q$ and a greedy policy;
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$;
        $S \leftarrow S'$ ; $A \leftarrow A'$;

In principle it is possible to update after any number of steps $n$, as long as $n$ does not exceed the total number of steps in the episode. In the extreme case, where $n$ equals the total number of steps in the episode, the resulting algorithms is exactly the same as MC learning. This concept is called $n$-step TD learning. Extending on this concept, one may also combine information from two different $n$-step updates. For example, one might update using the average of the updates from taking one step, two steps and four steps. Taking the extreme case, one can combine the updates from all possible step-sizes using a weighting function $\lambda$.

This type of update, TD($\lambda$) can smoothly transition between basic TD, therefore also called TD(0), learning and MC learning, equivalent to TD(1).

### A-2-3 Exploration versus exploitation

During a learning process, two conflicting interests occur. First of all, the agent prefers to choose takens it has taken before and that it knows will lead to a strong reward. On the other hand, the only way to discover such rewarding actions is by trying them out a first time. The result is a vicious circle in which the actor has to *exploit* its experience in order to choose the right actions, but simultaneously needs to *explore* the possibilities to build strong experience. Pursuing either one of these tasks while neglecting the other will never result in success. The actor will have to balance exploration and exploitation in a way that progressively favors the best action without overlooking potentially rewarding undiscovered actions.

The balance between exploration and exploitation has to be struck whilst processing reward signals and choosing new actions. In the previous sections the actions were chosen greedily with respect to $Q(s, a)$. This type of greedy selection is purely exploititive and does no leave any room for exploration. Therefore, equation A-7 is usually adapted to include a stochastic element. An effective and popular means to introduce exploration in the action selection is by adopting an $\epsilon$-greedy policy. Such a policy introduces a stochastic element that chooses the greedy action most of the time, but prefers a random action every now and then. $\epsilon$ represents the probability of selecting a random action. This makes $\pi$ a stochastic policy where the

chance of taking action $a$ given state $s$ equals:

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon, & \text{if } \underset{a}{\text{argmax}} Q(s,a). \\ \frac{\epsilon}{m}, & \text{otherwise.} \end{cases} \tag{A-12}$$

.

## A-3 Determining Q-values

The Monte Carlo algorithm described in 2 features a state-action value $Q(s,a)$. If a certain state-action-pair $(s_i, a_i)$ is visited during an episode the resulting reward $G$ from that episode is added to the state-action-pair's list of returns. The mean value of this list is assigned to $Q(s_i, a_i)$. This approach, coined 'table-lookup', requires keeping track of reward lists for all state-action-pairs. There are two major downsides to this approach.

### A-3-1 Table-lookup: curse of dimensionality

If we denote the total number of possibly reachable states by $n_s$ and the total number of possible actions by $n_a$, the lookup approach requires $n_s$ x $n_a$ separate Q-values to be stored and updated throughout the learning process. Taking, for instance, a simplified version of black jack. In this game a dealer can be showing 10 different playing cards, and the sum of the player's hand can vary from 12 to 21, assuming hands under 12 automatically call for a hit. This results in 100 different states (10 by 10). With only two possible actions this means Q-values can be stored in a 10-by-10-by-2 3D matrix, consisting of 200 cells. Playing a few thousand games should be sufficient to get a reasonable feel for the value of each state-action-pair.

Unfortunately, it turns out that many real-life applications of reinforcement learning are not as easily tabulated as a game of black jack. Staying in the gaming environment, a match of backgammon can feature as many as $10^{20}$ state, whereas the board game Go even features $10^{170}$ unique states [53]. Tabulating state-spaces of such a size would require enormous amounts of memory to store the Q-values, let alone the amount of games required to have visited all state-action-pairs sufficiently often. Finally many problems exists for which a continuous state-space should be considered, as is the case with autonomous vehicles. Therefore, table-lookup approaches are limited to problems with discrete state spaces. These problems are the result of what is called the curse of dimensionality: more-dimensional spaces quickly growing to infeasible sizes.

### A-3-2 Table-lookup: lack of generality

A second drawback of using lookup-tables can be found in their lack of generality: each state-action-pair is considered an independent situation. Therefore, neighboring or related state-action-pairs do not learn from the rewards gained by their peers. A common example given to demonstrate this lack of generality can be found in a game of Pac-man. Consider the two states in figure A-4. The only difference between the states is the presence of a white

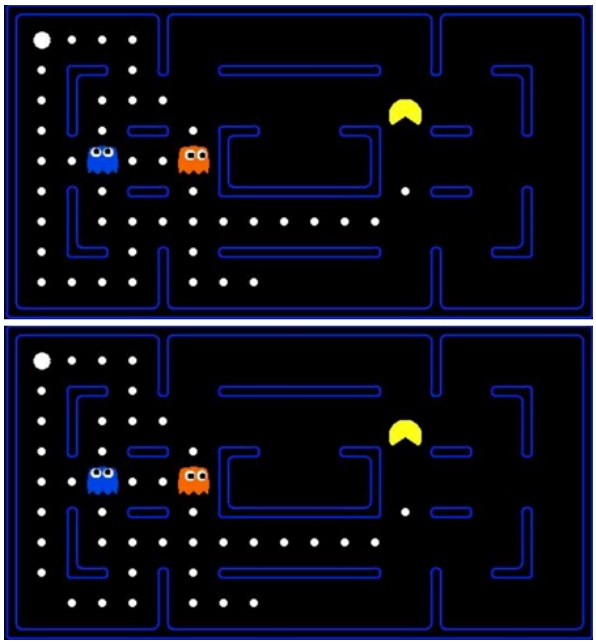


**Figure A-4:** Two very similar, but slightly different states in a game of Pac-man. The only difference is the piece of food in the lower-left corner being present or not. As these two states are different, they will be handled as entirely independent situations when using table lookup.

food dot in the lower-left corner. Looking at these images, the states are clearly related, and learnings from either one of the two states are most likely directly applicable in the other state. Using table-lookup however, both states refer to a different cell in $Q(s, a)$ and are therefore treated as entirely independent individual entries.

## A-4 Value function approximation

The presented downsides of table-lookup limits its applicability to only a niche type of problems. The large state-space examples have however been accomplished by RL (Backgammon using TD learning [73], Ms. Pac-Man [74] and in an impressive breakthrough paper Go [48]). Indeed none of these three used a table-lookup approach. Instead they al relied on using a set of hand-crafted features to describe the current game state. These feature values can be used as input for a function approximator that determines $Q$. In the simplest form this can be a linear combination of features, in more advanced examples like [48] a deep neural network was used. Using this approach, $Q$-values no longer need to be stored. Also, the state-space is no longer required to be discrete, as function approximators can generally work well with continuous inputs.

Instead, the features have to be calculated every time $Q$ requires evaluation. Also, the goal of training is no longer to establish a trustworthy lookup table with state-action-values for all $(s, a)$ coordinates, but to find the right parameters for the function approximator to generate an accurate value of $Q(s, a)$ given the feature values stemming from $(s, a)$.

To elucidate the concept using features, let us take another look at the Pac-Man example shown in figure A-4. Sensible features might be:

- Distance to nearest ghost
- Presence of food in each possible direction (booleans)
- Presence of walls in each possible direction (booleans)
- ...

Strikingly, these features would have equal values for both examples in figure A-4. The function approximator does not distinguish between the two situations in the figure, and therefore the results from the top situation fully generalize to the bottom one. Using value function approximation, experience from every single state-action-pair is valuable for a whole set of similar state-action-pairs, even unseen states.

### A-4-1 Drawbacks of value function approximation

Unfortunately, value function approximation introduces a number of downsides. First of all the approximations can only be as good as the quality of the input features. Therefore features should be hand-picked, requiring a significant amount of insight in the problem at hand from the engineer implementing the algorithm.

Secondly, accurate value function approximation can become a very difficult, if not impossible task, when the relation between features and the true state-action-value is non-linear. A method for dealing with such non-linear relations is given next, but such methods do significantly increase the complexity of the resulting algorithm.

## A-5 Function approximation methods

A large number of function approximation method are available throughout literature, although in machine learning the most common examples are linear feature combinations and neural networks. After a short introduction on linear feature combination approximators, a thorough overview of neural networks will be given.

### A-5-1 Linear feature combinations

Having hand-crafted a set of features, the feature values can be stored in a feature vector $\phi$. This vector from the input of any function approximator one might use. In the simplest case a linear combination of the features is used to approximate $Q$:

$$\hat{Q}(s,a) = w_i \phi_i(s,a) \tag{A-13}$$

with weight vector $w$. Einstein's summation convention is adopted, effectively turning equation A-13 into the dot product between vectors $\phi$ and $w$. During learning, weights in $w$ are updated such that $\hat{Q}$ approaches $Q$. Defining the approximation error as $\delta = Q - \hat{Q}$, the weight update can be determined as:

$$\Delta w = \alpha \delta \frac{\partial \delta}{\partial w} \tag{A-14}$$

In which $\alpha$ represents the update step size. The partial derivative of the error with respect to the weights is dependent on $\hat{Q}$ only:

$$\frac{\partial \delta}{\partial w} = \frac{\partial \hat{Q}}{\partial w} = \phi \tag{A-15}$$

and substituting the result back to A-14 results in:

$$\Delta w = \alpha \delta \phi \tag{A-16}$$

During learning, the real $Q$ is unknown (hence the effort to approximate it). However during training rewards are received. The state-action values should reflect these rewards since for each time step:

$$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) = Q(s_t, a_t) \tag{A-17}$$

The left-hand-side of the equation is referred to as the target. In TD-learning (algorithm 3) the difference between the target and approximated value is checked after each time-step and the resulting error used to update the weights of the linear function approximator:

$$\Delta w = \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(s_t, a_t)]\phi \tag{A-18}$$

For different algorithms, like the Monte Carlo algorithm, the target is specified differently, but the same principles adhere.

## A-5-2 Neural networks

Another more complex approach to function approximation is known as a neural network (NN). NNs are inspired by nature's example: the nervous system. A nervous system consists of neurons that are interconnected via axons and together form a complex network. Signals propagate through the nervous system by delivering a small impulse to a neuron. If the charge is large enough, the neuron is activated and emits a new impulse. Using this domino-type interaction, enhanced by some of nature's cleverest tricks: Myelin sheaths, the nervous system is able to transmit signals at a great speed through a complex network.

### Forward pass

Inspired on nature NNs were developed to perform complex function approximation. An exemplary NN is shown in figure A-5a. The colored dots are the neurons, the blue and black lines are the axons. Each vertical row of dots is referred to as a layer. The first layer is always the input layer, the last layer is always the output layer. Currently, there is only one layer in between: the hidden layer. The total number of layers can be varied by adding more hidden layers. When all parameters are correct, when the neural network has been trained previously, a function can be approximated by performing what is known as a forward pass. In such a forward pass the function features are loaded onto the input layer. This implies the input layer is of the same size as the feature vector $\phi$. The feature values are passed forwards towards the hidden layer, through the blue lines. The lines have weights attached to them, and during transportation each feature is multiplied by the weight value. The weighted sum then enters node $j$ as $x_i$. In the node the value $x_i$ is put through an activation function, also known as

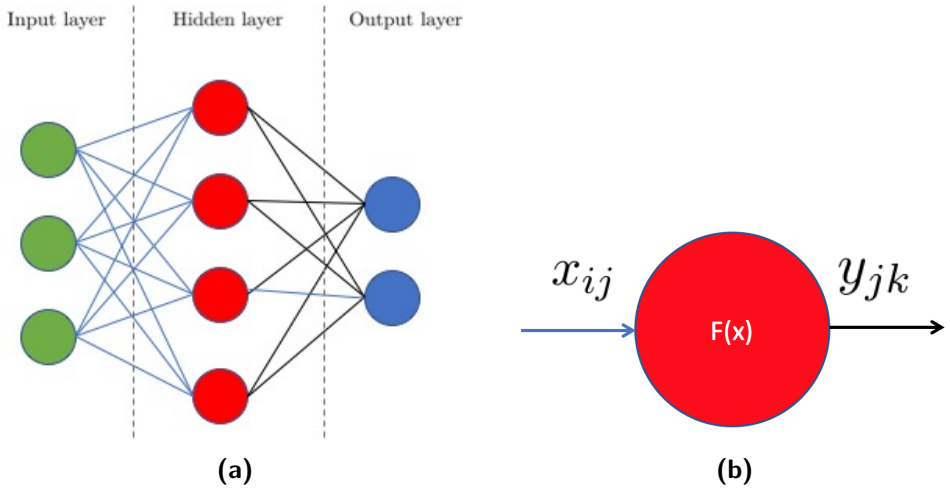



**Figure A-5:** (a): An example of a neural network. The example features an input layer with 3 nodes, a hidden layer with 4 nodes and an output layer of 2 nodes. (b):An example of a neural network's hidden layer's neuron $j$. Values from the preceding layer are sent to neuron $j$. The value from node $i$ in the preceding layer is referred to as $x_i$. The weighted sum of all input values $x$ is taken and serves as input for activation function $f(x)$. The output from $f(x)$ is node $j$'s output: $yjk$, $k$ indicating the output node it is attached to.

squashing function. Such a function 'squashes' the input value to fit a value at either one end of an extreme spectrum. As an example a famous activation function is plotted in figure A-6. In a full forward pass, the data is loaded into the input layer and subsequently moved through all layers, following the procedure as just described, until the values at the output layer are known.

### back propagation

Input values can be forwarded to calculate the neural network's outputs. Let us imagine the neural network is not yet trained and after moving through a forward pass, the network's

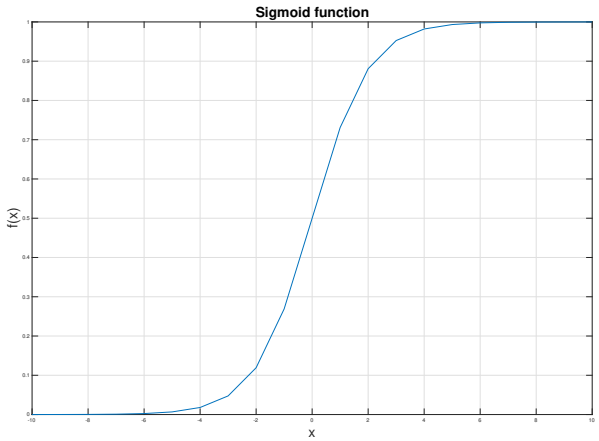



**Figure A-6:** The sigmoid function squashes all values towards 0 or 1.

outputs do not match the correct values. In such a case an error $E$ is present in the output, indicating that the network's weights are incorrect. In correspondence with most optimization techniques the best way of adapting the weights would be to find the derivative of the error with respect to each individual weight, such that all weights can be updated in the direction that minimizes the error. Using this approach, known as steepest descent, the correct weights can be obtained after cycling through numerous iterations. Obtaining this derivative information is possible through a process called back propagation, in which the error is propagated through the network in reverse direction, relying heavily on the chain rule of differentiation to come to the desired derivatives. An algebraic derivation of the back propagation process is given in appendix B.

# Appendix B

# Addendum to paper Section II - C - 1: Derivation of backpropagation algorithm

The backpropagation procedure can be derived using a series of chain-rule governed steps [75]. An example network is shown in figures B-1a and B-1b, to which will be referred in the following derivation. Extending this derivation to larger networks is however possible. We will refer to the $k$ input values as $p_k$. For the $j$ hidden layer nodes the weighted and summed inputs are defined as $x_j$ and the outputs as $h_j$. The $i$ output nodes are fed a weighted sum of hidden layer outputs $s_i$ and output values $y_i$. Weights between the input and hidden layer are referred to as $w_{kj}$. The error $E$ is the result of a cost or loss function:

$$E = \frac{1}{2}\sum_i (\hat{y}_i - y_i)^2 \tag{B-1}$$

in which $\hat{y}$ indicates the desired output value and $y$ the network's outputs. The desired weight update following steepest descent can be expressed like:

$$w_{ab} = w_{ab} - \alpha \frac{\partial E}{\partial w_{ab}} \tag{B-2}$$

with stepsize $\alpha$ for each layer $ab$. Looking at the hidden-to-output weights first, one can state that:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ji}} \tag{B-3}$$

where

$$\frac{\partial E}{\partial y_i} = -(\hat{y}_i - y_i) = y_i - \hat{y}_i \tag{B-4}$$

$$\frac{\partial y_i}{\partial s_i} = f'(y_i) \tag{B-5}$$

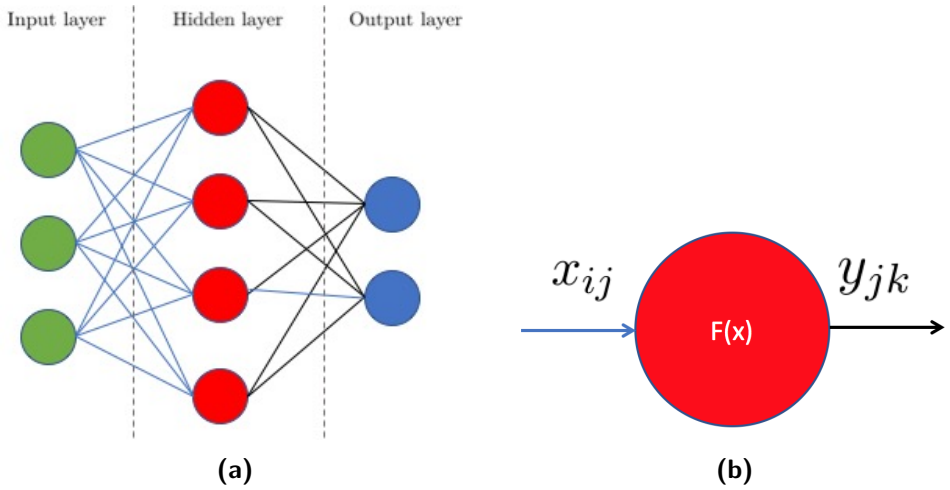



**Figure B-1:** (a): An example of a neural network. The example features an input layer with 3 nodes, a hidden layer with 4 nodes and an output layer of 2 nodes. (b):An example of a neural network's hidden layer's neuron $j$. Values from the preceding layer are sent to neuron $j$. The value from node $i$ in the preceding layer is referred to as $x_i$. The weighted sum of all input values $x$ is taken and serves as input for activation function $f(x)$. The output from $f(x)$ is node $j$'s output: $yjk$, $k$ indicating the output node it is attached to.

$$\frac{\partial s_i}{\partial w_{ji}} = h_j \tag{B-6}$$

such that

$$\frac{\partial E}{\partial w_{ji}} = (y_i - \hat{y}_i) f'(y_i) h_j \tag{B-7}$$

Note that the derivative of the transfer function $f(x)$ was used to determine $^{\partial y_i}/_{\partial s_i}$. It useful to denote $\delta_i$ as

$$\delta_i = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i} = (y_i - \hat{y}_i) f'(y_i) \tag{B-8}$$

such that

$$\frac{\partial E}{\partial w_{ji}} = \delta_i h_j \tag{B-9}$$

For the other layer of weighs, from inputs to hidden the derivation is slightly more complex since the error now has to pass through a series of layers and weights before it can be related to the weights:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial x_j}\frac{\partial x_j}{\partial w_{kj}} \tag{B-10}$$

The derivative of the error with respect to the hidden node-outputs can be determined as a chain-rule

$$\frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial h_j} = \delta_i w_j i \tag{B-11}$$

Such that equation B-10 becomes

$$\frac{\partial E}{\partial w_{kj}} = \delta_i w_j i f'(h_j) p_k \tag{B-12}$$

Let us define another $\delta$ as:

$$\delta_k = \delta_i w_j i f'(h_j) \tag{B-13}$$

We can now generalize these equations by stating that for each layer from $b$ to $a$ with input

$$w_{ab} = w_{ab} - \alpha \delta_a y_b \tag{B-14}$$

Wherein $y_b$ represents the outputs of layer $b$; in the example case $h_j$ for the hidden layer and $p_k$ for the input layer. $\delta_a$ is defined as:

$$\delta_a = \begin{cases} (y_a - \hat{y}_a) f'(y_a), & \text{if } a \text{ is the output layer} \\ \delta_b w_b a f'(y_a), & \text{else} \end{cases} \tag{B-15}$$

In which $\delta_b$ represents the delta-term corresponding to the layer $b$, one row ahead in the forward pass from $a$.

# Appendix C

# Addendum to paper section II - C - 1: The Neural Network implementation

Many Neural Network (NN) packages exist that can be easily obtained and used to create and train a NN in just a few lines of code. By using such packages however one abstains from the detailed workings of such a network. For one this means it is impossible to make adjustments in the network implementation itself and secondly the profound understanding gained from self-programming such a network is lost. Therefore the current implementation was developed in `matlab` without the use of any NN packages. In practice this means functions have been developed to perform activation functions, perform a forward-pass, perform back-propagation and implement an effective strategy for updated the NN's weights. These implementations, especially that of backpropagation, is a tedious work and very much error-prone. It is therefore generally advised to test the implementation, which has been done in twofold in the current research: by testing the implementation on a simple supervised learning problem and by performing a derivative check.

## C-1 Supervised learning example

The easiest way of testing the NN implementation is by programing it in a modular fashion, such that the NN module can be decoupled from the learning challenge at hand, and connected to a simple learning problem with a known correct outcome. In this case the choice was made to use the NN implementation module to learn the XOR problem. This is a supervised learning problem in which a simple network is required to learn the behavior of an XOR logical operator which returns a TRUE if, and only if, either one of its two inputs is TRUE while the other is FALSE. An overview of possible inputs and associated output is given in table C-1.

This logical operator is well-suited to serve as test-case for the NN implementation, since one can easily generate a large number of examples with their associated correct outputs, and use these to train the network. If the network's implementation is correct, such training should be completed within a few thousand iterations. During learning the mean squared error has

**Table C-1:** Output table for the XOR operator, stating all possible configurations of its two boolean inputs A and B and the associated correct outputs.

| A | B | Out |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

been tracked. Training was automatically halted once the moving squared average error over the last 50 values, reached a value below $1 \cdot 10^{-3}$. The error propagation has been plotted in figure C-1 for four different training algorithms. As the image clearly shows convergence occurs, which endorses the notion of a correct implementation.

## C-2 Update algorithms

Several update algorithms have been tested before adopting the *Adam* algorithm as was presented in the paper. These algorithms primarily describe how the results from back propagation, e.g. the gradient information, should be used to update the weights of the network. As shown in figure C-1, four different algorithms were implemented and tested: SGD, SGD with momentum, SGD with Nesterov momenteum and Adams. All four will be discussed briefly in this section.

### C-2-1 Stochastic gradient descent

Stochastic gradient descent, or SGD, is the simplest form of weight updates one may image. The gradient of the error with respect to the weights describes how each weight affects the error in the current state, around which is being linearized. Therefore the error can be minimized by moving in opposite direction of this gradient. This direction is called the direction of steepest descent, as it is the fastest way to minimize the error. However keep in mind this gradient is the dependent on the linearization point and therefore only reliable close to this point. To prevent overshooting out of the reliability zone around the linearization point, a stepsize $\mu$ is introduced. The update rule according to SGD equals:

$$\Delta W = -\mu \frac{\partial E}{\partial W} \tag{C-1}$$

### C-2-2 Momentum and Nesterov momentum

The stepsize $\mu$ prevents the SGD approach from overshooting, but by doing so also limits the update speed once a reliable gradient is found. Visually this can be seen in the long and near-linear error descent in figure C-1a. To speed up learning an extra term is therefore proposed in the form of a fraction $m$ of the previous weight update $\Delta w(t-1)$ [76]. This extra

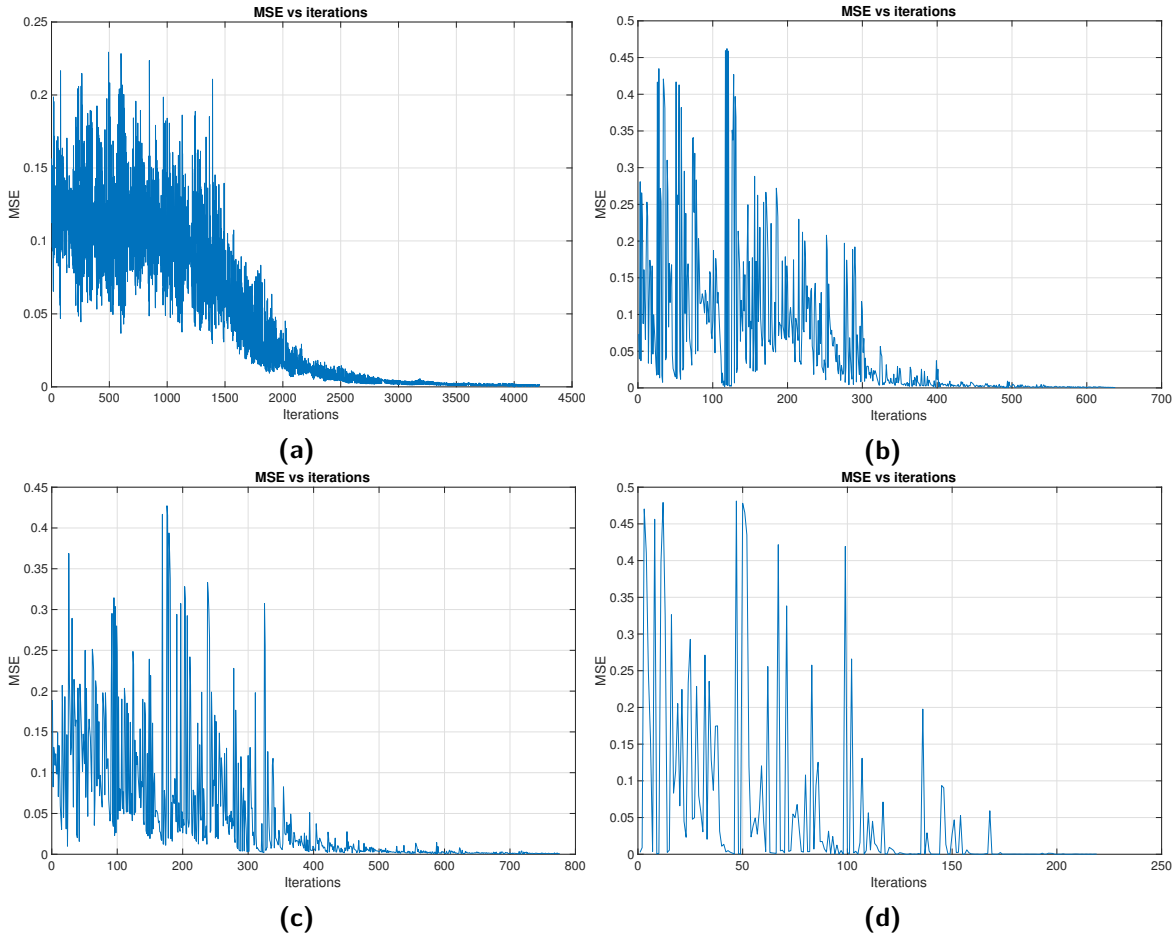



**Figure C-1:** Mean squared error versus iteration count while training NN using four different training algorithms. (a) shows the result of using basic steepest gradient descent (SGD). In (b) a momentum term is added, clearly expediting the learning process. Similar results are obtained by using Nesterov momentum (c). Finally the advanced Adam algorithm (d) shows the highest convergence rates. Convergences occurred respectively after 4222, 639, 777 and 176 iterations.

term is added to the update step, introducing the numerical variant of inertia:

$$\Delta W(t) = -\left(\mu \frac{\partial E}{\partial W(t)} + m\Delta w(t-1)\right) \tag{C-2}$$

Especially when consecutively similar gradients are found the momentum term increases exponentially, thereby quickly picking up speed in the opposite direction of the stable gradient. This means that by adopting momentum, the learning speed will rise whenever relatively stable gradients are found and decrease along more turbulent trajectories. The update step described in equation C-2 relies on gradient in the current time-step $t$ and a momentum term based on the previous time-step $t-1$. Nesterov [77] noted that the momentum term does not depend on the current gradient information, therefore leaving potential for a higher-quality gradient direction by performing the momentum-part of the update before determining the new gradient $\partial E/\partial W(t)$. As the results in figures C-1b and C-1c show, adding the momentum term significantly speeds up the learning process, reaching a converged state after only $1/6^{\text{th}}$ of the required iterations without momentum.

#### Adam

The name Adam is an acronym for Adaptive Moment Estimation [78]. The Adam algorithm makes both the learning rate $\mu$ and the momentum term $m$ parameter-dependent. That is: it uses a momentum style update on all the weights, but adapts the parameters $\mu$ and $m$ based on the update frequency of each parameter. Infrequent parameters receive larger updates whilst frequent parameters' updates are kept small. The adaptiveness of both the learning rate $\mu$ and momentum term $m$ prevents the need for specifying the right rates upfront for all parameters. Figure C-1d shows Adam's fast convergence, reaching convergence a factor 3.5 earlier than both momentum-enhanced SGD examples.

## C-3 Gradient check

The results of back propagation process is the gradient of the error term $E$ with respect to the weights, as is shown in appendix B. Since this process is such tedious work a separate check for the implementation of this section has been performed. The checking procedure consists of determining numeric finite-differences gradients and comparing them to the analytic derivatives resulting from back propagation. These values should be similar. As an example, both gradients are shown in figure C-2. To clarify, the shown values are the derivatives of the error $E$ with respect to the weights connecting the XOR's second input (B) to 30 hidden layer nodes. Gradients of the other weights show similar results. Visually, the two sets of gradients seem to completely overlap. Indeed, the overall mean absolute difference between the two sets of gradients equals $2.1752 \cdot 10^{-5}$, which is close to nothing. Similar to the previous test, these results confirm the correct implementation of the Neural Network.

## C-4 Extension of basic network

Neural Networks have been researched extensively, resulting in a variety of additions to the basic concept, all in pursuit of increasing the network's ability to learn. Several of these

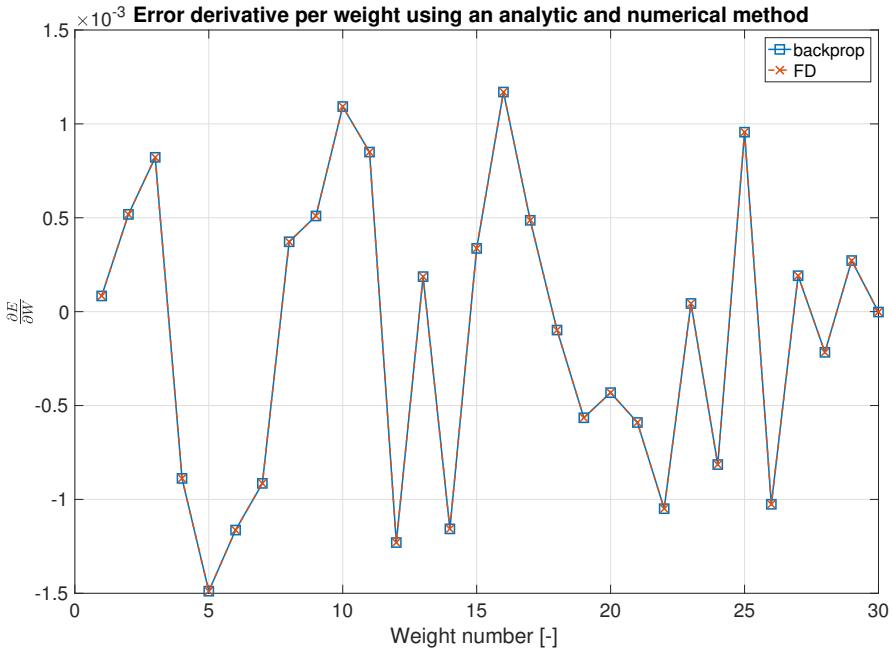



**Figure C-2:** This figure shows the calculated gradients of the error term$E$ with respect to a series of weights. The blue squares represent the analytic result from backpropagation and the red crosses the numeric results from a finite-difference calculation (FD). The results are clearly similar.

additions, for now referred to as *trics*, have ben adopted in the NN used to produce the results in the presented paper. These extensions to the basic neural network will be discussed briefly in this section.

## C-4-1 Regularization

Neural networks can be prone to a scenario known as overfitting. Overfitting means that the network trains in such a way that it scores extremely well at the given training set, but hardly generalizes to new test examples. Overfitting is generally the result of incorporating too much complexity in the network, as is the case when using a polynomial of to high an order when fittgin a small set of data examples. Regularization is a special mechanism that encourages fitting parameters to be small, i.e. penalizes complexity. Regularization terms are added to the network's object function, which subsequently starts optimizing both for the best fit and for the lowest level of complexity. In the solution presented in the paper, two types of regularization are present: $L_1$ and $L_2$ regularization. Additionally, experimentation with dropout, another method of regularization, has been performed.

### $L_1$ and $L_2$ regularization

In $L_1$ regularization a penalty term is added to the objective function. This penalty grows proportionally to the sum of the absolute values of all input parameters. This penalty, sometimes referred to as the sparsity inducing property of $L_1$ regularization [79], encourages the network to push non-contributing input parameters to zero, creating a sparse input

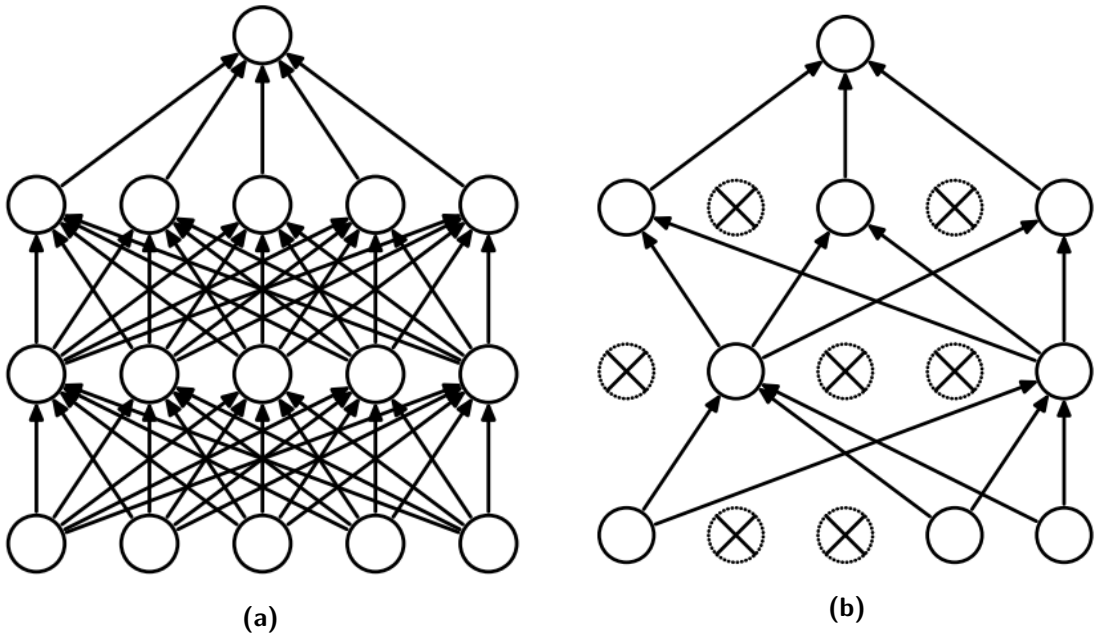



**Figure C-3:** (a) a standard neural network with two hidden layers. (b) a neural network after several nodes have been dropped out. Image courtesy of Srivastava et al. [81].

parameter vector [80]. In practice this results in feature selection: the process of selecting only the important features. Similarly in $L_2$ regularization a penalty term is also added to the objective function. In $L_2$ however this term is proportional to the sum of the squares of the input parameters. By squaring the input parameters the $L_2$ norm is differentiable throughout its whole range, whereas the $L_1$ term is not. This has computational benefits. The two terms both have their merits and may be superimposed, as was done in the current implementation. During a parameter sweep optimal settings for both parameters were found to be $1 \cdot 10^{-3}$ for both mechanisms.

### Dropout

The dropout algorithm, first introduced by Srivastava et al in 2014 [81] attempts to prevent over fitting by combining the predictions of many different large NNs during test time. This is achieved by randomly dropping out nodes and their connecting weights from the network during training. This essentially means a different NN architecture, or thinned network (figure C-3), is used in each training iteration. This prevents the network from overly relying on a single prediction strategy and thereby creates a robust and resilient network. At test time all nodes are active, essentially acting as a large number of parallel thinned networks. The weights are scaled back during test time to counteract the effect of the superimposed thinned networks. Dropout was implemented in the network and tested both with the XOR example and its actual application in the presented paper. Unfortunately results did not improve and performance decreased upon introducing dropout. Whether this is due to a flaw in the algorithm cannot be stated. However it seems more likely that such poor behavior is the result of an unknown implementation error on the part of the author.

# Appendix D

# Addendum to paper sections II and III: Design framework

In section II and III of the presented paper the adopted method and performed experiment are described in a concise manner. A more detailed description of the method and experiment is given in this appendix in the form of a framework, describing how to pose a design challenge in a game-like fashion for reinforcement learning.

As mentioned in appendix section A-1-3, Reinforcement learning (RL) is described by Kaelbling et al. [34] as behavioral learning by trial-and-error interactions with a dynamic environment. Ergo, when applying RL on mechanism design, there should be some sort of environment that can interact with the results of the algorithm's output, such that trial-and-error occurs. Since the output of the learning algorithm should be a design, the interaction with the environment will be an assessment of the generated design, resulting in a score. There are several subjects in need of consideration in order to have an interactive environment that is ready to learn:

1. Mechanism representation
2. Kinematic simulation
3. Scoring
4. States and actions
5. Integration into an interactive environment

In the current chapter, these elements will be elucidated. Together, they form the framework for the rest of this thesis.

## D-1 Addendum to paper section II-A: Mechanism representation

Designs generated by any form of smart algorithm have to be communicated to external sources in order to have any form of application. This can be a human designer, other

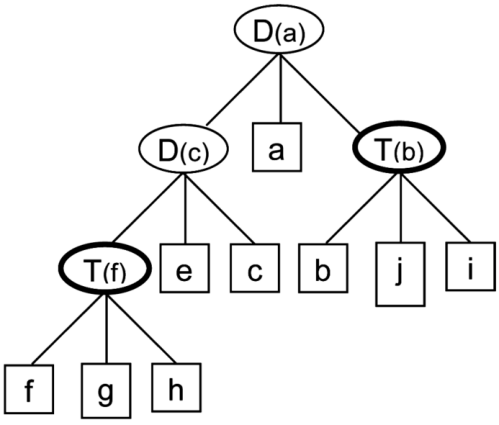



**Figure D-1:** Example of a tree-like flow chart, edited from [1].

pieces of software for post-processing, or even a fabrication center for direct materialization of the generated design. Doing so requires a numeric representation of the design that can be easily visualized. Literature study [82] has shown there are several ways to perform this task. However, in combination with RL the study showed clear advantage for one specific representation style using decision trees as introduced by Hod Lipson in 2008.

### D-1-1 Lipson operators

Lispson [1] proposed a tree representation to describe a series of operations resulting in a unique kinematic chain. When drawing these operations in a schematic manner, the resulting overview resembles a tree as shown in figure G-6. Both the illustrated top-down approach and a more abstract bottom-up approach can be used, although Lipson [1] has only demonstrated the former. Lipson distinguished only two different operators: the $\boldsymbol{T}$ and $\boldsymbol{D}$ operators. Lipson [1]: "The $\boldsymbol{D}$ operator creates a new node and connects it to both the endpoints of a given link, essentially creating a rigid triangular component. The $\boldsymbol{T}$ operator replaces a given link with two links that pass through a newly created node." The illustration in figure D-2 shows how the two operators differentiate when applied on the same link. As the illustrations shows, the $\boldsymbol{D}$ operator is a simple extension, whereas the $\boldsymbol{T}$ operator is more complex and replaces the subjected link with a new one.

The rationale behind these operators stems from the fact that both operators keep the number of degrees of freedom in the system constant. Lipson found most conventional representations (like graphs) would tend to generate over-constraint, deadlocked mechanisms. His tree representation approach allows for optimization algorithms to explore the space of solutions without over or under constraining the mechanisms.

Given a known starting point, a mechanism can be fully described by a series of $\boldsymbol{T}$ and $\boldsymbol{D}$ operators. The exact placement of the new hinges can be described by relative coordinates, passed along with the operators.

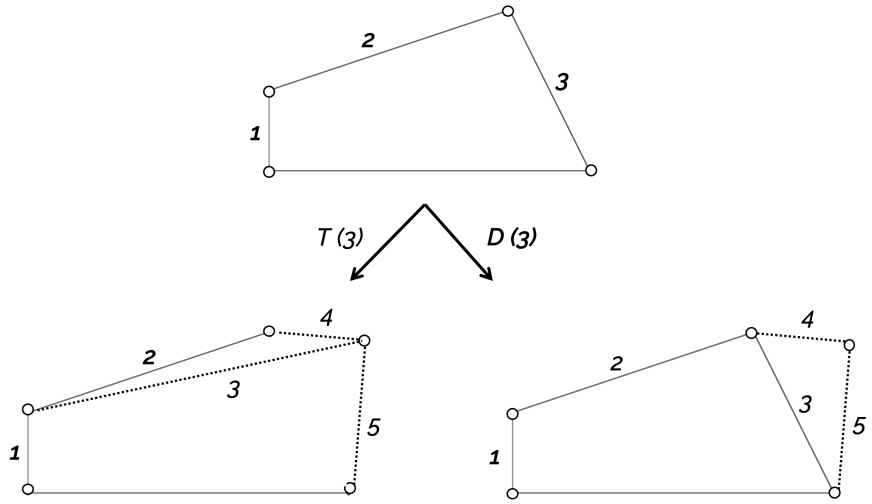



**Figure D-2:** Example of different outcomes for $\boldsymbol{D}$ and $\boldsymbol{T}$ operations on link number 3

## D-2 Addendum to paper section II-B: Kinematic simulation

Generated mechanisms will need to be evaluated in order to interact with it. The type of evaluation required is dependent on the given design goal. This thesis focuses on a path tracing objective, which means the linkage's trajectory should be determined and evaluated. Analytical methods exist to determine the trajectory of linkages, however each method only applies to subset of the total solution space. A more general solution is obtained by determining the trajectories numerically through simulation. In such simulations, the trajectories throughout the linkage's whole range of motion will be determined by subsequently forcing a small rotation on the input link and solving for the position of all other links and hinges.

The starting point of a simulation is an initial configuration, like the one shown in figure D-3. The hinges at the end of the orange bar are both connected to ground and cannot move. The yellow bar is the *input link*, which means its rotation will be prescribed once the simulation begins. During simulation, the input link will be rotated as far as possible in both directions, to discover the full range of motion of the mechanism. The two resulting ranges of motion, one from each rotation direction, are stitched together to form a single smooth trajectory. Applying this treatment to the mechanism presented in figure D-3 leads to the trajectories shown in the dashed lines.

The linkage is modeled using finite elements and simulated numerically using Newton's second-order method. The approach taken to model the elements and solve the positions for each time-step was proposed by Avilés et al. [83]. They proposed using a reduced form of the stiffness matrix called the geometric matrix. Using this matrix results in a simple, general and computationally cheap solution process. As this process will be executed thousands, if not millions, of times during RL the computationally efficiency of this approach is an important merit.

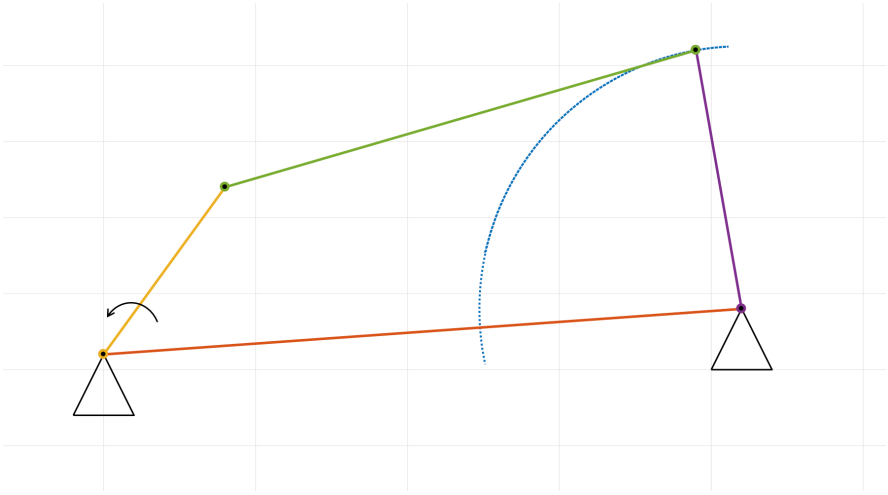


**Figure D-3:** The starting point: a basic four-bar mechanism. The input link is depicted in yellow, the base in orange. The green transmission link transfers the input link's motion to the purple output link. The blue dashes trace the trajectory of the input link's far end. The red dashes trace the output link's path, which moves back and forth along the same trajectory.

## D-2-1 Addendum to paper section II-B-1: FEM model

The linkage is modeled as rod type elements, connecting to each other at their nodes. Rod elements can only be strained in axial direction, since they cannot transfer force in any other direction through the revolute joints that bind them. Therefore, a rod has four degrees of freedom in 2D, two for each node, in global coordinates, or two in local coordinates With surface area $A_e$ and elasticity modulus $E_e$ the stiffness matrix for the element with local coordinates $x^e$ can be expressed in its familiar form:

$$[k_b]_e = \frac{A_e E_e}{L_e} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{D-1}$$

The element's length can be determined by its coordinates $x$ of nodes $A$ and $B$ using simple trigonometry:

$$L_e = \sqrt{(x_{1B} - x_{1A})^2 + (x_{2B} - x_{2A})^2} \tag{D-2}$$

and the angle $\theta_e$ represents the angle of the rod in the global coordinate framework, being zero when fully horizontal and positive for anti-clockwise rotation. Rotation matrix $R$ can be utilized to transform $[k_b]_e$ to global coordinates:

$$[k_b]_e = [R]_e^T [k_b]_e [R]_e \tag{D-3}$$

Which after substitution of the rotation matrix' trigonometric entries leads to the symmetric stiffness matrix:

$$[k_b]_e = k_e \begin{bmatrix} \cos^2\theta_e & \cos\theta_e \sin\theta_e & -\cos^2\theta_e & -\cos\theta_e \sin\theta_e \\ \cos\theta_e \sin\theta_e & \sin^2\theta_e & -\cos\theta_e \sin\theta_e & -\sin^2\theta_e \\ -\cos^2\theta_e & -\cos\theta_e \sin\theta_e & \cos^2\theta_e & \cos\theta_e \sin\theta_e \\ -\cos\theta_e \sin\theta_e & -\sin^2\theta_e & \cos\theta_e \sin\theta_e & \sin^2\theta_e \end{bmatrix} = k_e[g]_e \quad \text{(D-4)}$$

The matrix $[g]_e$ in equation D-4 is the geometric matrix of the element. This matrix is independent of material properties and element lengths, and is only a function of the rod's orientation given by $\theta_e$. The stiffness values $k_e$ are all brought back to unit stiffness, since the kinematics of the linkage are independent of these stiffnesses. The geometric element matrices are assembled into the system's total geometric matrix $[G]$. The element matrices are ordered in $[G]$ according to the element number to which they attend.

Numerically, assembly takes place in two steps. First of all, each element matrix $[g]_e$ is expanded: that is $[g]_e$ is being placed in null-matrix of the same dimensions as $[G]$, on the exact same location it should end up in $[G]$. Avilés et al. [83] refer to expanded matrices by denoting them with a bar: $[\bar{g}]_e$. After summation of the expanded matrices the total geomatric matrix $[G]$ is obtained.

## D-2-2 Addendum to paper section II-B-1: Solving the position problem

After perturbing the input link, the distances between the nodes of the system are no longer equal tot he undeformed length. Finding the new positions of the nodes to satisfy this condition is referred to as the position problem. To solve the position problem numerically, an error function is required. In this case the system's elastic potential $V$ can be used as such. To solve the problem, the elastic potential must be zero. The error function for a linkage with a total of $b$ bars of length $L_e$ equals

$$V(x) = 1/2 \sum_{e=1}^{b} (l_e(x) - L_e)^2 = 1/2 \sum_{e=1}^{b} \left( \sqrt{x^T [\bar{g}]_e x} - L_e \right)^2 \quad \text{(D-5)}$$

in which $l_e(x)$ represents the length of element $e$ as a function of the nodal coordinates in $x$. During iteration, $l_e$ might not be equal to $L_e$, introducing elastic potential and thereby error. The length difference equals deformation $\delta L_e$, which is caused by parallel action forces $a_e$ on end points $A$ and $B$ of the rod. Since unit stiffness applies, one can express the action forces as

$$a_e = \begin{bmatrix} a_A \\ a_B \end{bmatrix} = e\delta L_e \begin{bmatrix} -\cos\theta_e \\ \theta_e - \sin\theta_e \\ \cos\theta_e \\ \sin \end{bmatrix} = \delta L_e h_e \quad \text{(D-6)}$$

in which vector $h_e$ transforms the forces that cause deformation $\delta L_e$ to the orientation of the rod element. It follows that

$$[\bar{g}]_e x = L_e \bar{h}_e \quad \text{(D-7)}$$

and therefore the expanded vector of action forces becomes

$$\bar{a}_e = \frac{\delta L_e}{L_e}[\bar{g}]_e x \tag{D-8}$$

Which can be summed per element to form the system action vector $A$.

To find the minimum of the error function D-5, Avilés et al [83] rely on Newton's second-order method. This requires expending the error function $V$ into a Taylor series around position $x_q$ and truncating the series after the second-order term:

$$V_q(x) = V(x_q) + \nabla V_q^T \nabla x_q + 1/2 \nabla x_q^T [H]_q \nabla x_q \tag{D-9}$$

with $\nabla V_q$ being the gradient vector $V$ in $x_q$, $[H]_q$ the Hessian matrix in $x_q$ and $\nabla x_q$ the nodal displacement vector $x - x_q$. The nodal displacement vector can be retrieved by solving the linear system of equations given by:

$$[H]_q \nabla x_q = -\nabla V_q \tag{D-10}$$

The resulting nodal displacement vector can be used as an update of the nodal coordinates towards a position of lower elastic potential. Using learning rate $\alpha$ the following update may be performed iteratively:

$$x_{q+1} = x_q + \alpha \nabla x_q \tag{D-11}$$

Once $V(x_q)$ reaches a value below a set tolerance $\epsilon$, convergence is achieved.

The values in $\nabla$ and $[H]$ still have to be determined. Using basic algebra, and again referring to Avilés et al [83], one can show that:

$$\nabla V(x) = \frac{\partial V(x)}{\partial x} = \sum_{e=1}^{b} (l_e(x) - L_e) \frac{\partial l_e(x)}{\partial x} = \sum_{e=1}^{b} \delta L_e \frac{[\bar{g}]_e x}{l_e} = \sum_{e=1}^{b} \delta L_e \bar{h}_e = \sum_{e=1}^{b} \bar{a}_e = A \tag{D-12}$$

and

$$[H] = \frac{\partial^2 V}{\partial x^2} = \sum_{e=1}^{b} \left( \bar{g}_e + \delta L_e \, \frac{\partial^2 l_e}{\partial x^2} \right) \tag{D-13}$$

in which the second-order derivative of $l_e$ with respect to $x$ can be represented as

$$\frac{\partial^2 l_e}{\partial x^2} = l_e^{-1} (\bar{z}_e - \bar{g}_e) \tag{D-14}$$

with $\bar{z}_e$ being the expanded form of symmetric matrix $[z]$:

$$[z] = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{D-15}$$

### D-2-3 Addendum to paper section II-B-1: Determining the trajectory

To determine the full trajectory of a linkage the proposed method will be performed several times, each time solving for the position of the free nodes. The free nodes are those nodes not grounded and whose location is not prescribed by the input link. At the start of each time-step the input link is perturbed a little, say $\frac{\pi}{150}$ radians, introducing new potential in the system. The position problem is then solved to find a new equilibrium position. After storing the node locations the process starts over by perturbing the input a little more. This process continues until the link has turned a full circle, or when no satisfying solution can be found. Finally, the mechanism is restored to its original position and solved for a fulll circle in the opposite direction.

### D-2-4 Addendum to paper section II-B-2: Detecting infeasible domain and singularities.

Depending on the geometry of a linkage, the input link may not be able to turn a full circle. In a numerical analysis, such infeasible domain can be detected by high levels of elastic potential. By removing trajectory points with high elastic potential only the feasible domain remains.

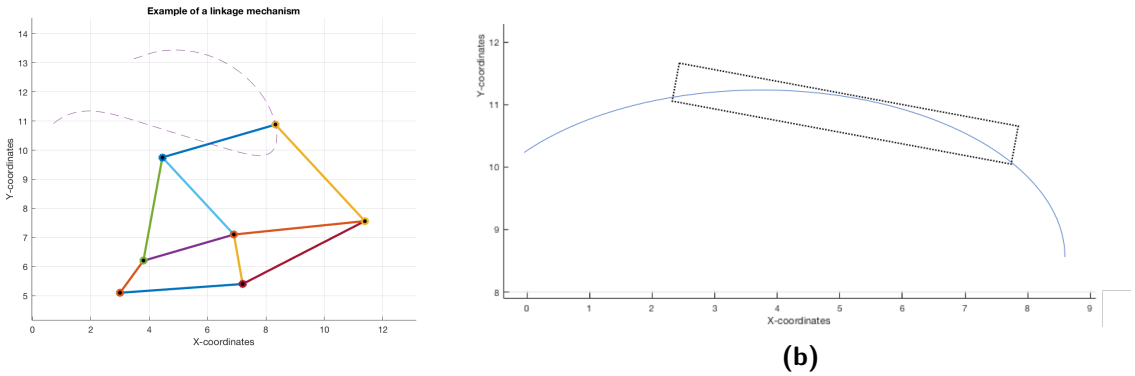
A special case occurs when two (or more) bars of a linkage become exactly aligned, effectively losing one degree of freedom. In such a case the linkage becomes unstable, resulting in large gradients of the potential energy. Accordingly, the matrices in equation D-11 become ill-conditioned. When minimizing the error for such a position the target is easily overshot, as a result of the large gradients. To deal with this an adaptive learning rate is employed, cutting $\alpha$ in half whenever the error is rising instead of falling. No updates of $x_q$ are performed in this case until the error ceases to rise.

## D-3 Addendum to paper section III-C: Design goals and scoring

The evaluation of a linkage takes place in two steps. The first step is finding the trajectory of all nodes, which was discussed in the last paragraph (D-2-2). The second step consists of comparing these trajectories with the desired end-goal and scoring them for fitness. Since such end-goals may vary, the scoring module is a separated and independent module. This means it can be swapped out without influencing the main learning algorithm. An example is giving for the case in which a straight trajectory is desired.

### D-3-1 Scoring for straight lines

The results of this thesis are mainly centered around straight-line-mechanism. To determine the fitness of such a straight-line-mechanism a scoring module is required that can select the straightest section of a trajectory and determine its straightness and length. Figure D-4a shows an example of such a trajectory. The trajectory consists of a number of coordinate points, not necessarily of equal distance to each other. By inspecting the deviation in $x$ and $y$ direction between each trajectory point $i$, the angle of the connecting vector can be determined: $\theta_i$. In




(a) (b)

**Figure D-4:** (a) When the short red link is rotated, each of the hinges in this mechanism start following their respective trajectories. The trajectory belonging to the yellow top hinge is depicted by the purple dashes. (b) Example of a minimum-surface box, fitted around a segment of a random trajectory. The aspect ratio of the shown example is roughly 1:8.

a straight section, all connecting vectors will point in the same direction. Therefore deviations in the direction of the connecting vectors $\Delta\theta_i$ are a characteristic of non-straight sections.

$\Delta\theta_i$ can only be compared fairly when the length of the connecting vectors is equal. Since this is not the case, $\Delta\theta_i$ is divided by the distance between point $i$ and $i-1$. The normalized result $\Delta\bar{\theta}_i$ can be compared for all trajectory points $i$.

Subsequently, the 25% yielding the lowest normalized deviation $\Delta\bar{\theta}_i$ are selected. Of these points, the longest sequence of connected trajectory points is distilled. The resulting sequence is determined as the straightest section in the trajectory. Subsequently a minimum-surface box is shaped along the straight section. The aspect ratio of this box can be used as a measure of straightness: the higher the ratio, the straighter the line. A ratio of 1:1000 is equal to a line that deviates 1 mm over a distance of 1 meter. The straightness score, or aspect ratio, is multiplied by the length of the section to come to the trajectories total score. This score is divided by 100 to prevent higher order-of-magnitude signals in the training procedure and subsequently outputted. If no significant straight section could be detected, or if the mechanism is otherwise found to be infeasible, a fixed negative score is registered. The absolute value of this penalty is set as a design variable at the start of the learning process.

## D-3-2 Scoring for figure-eight trajectories

A second scoring module has been developed to test the general applicability of the learning process. Instead of evaluating trajectories for straightness this module evaluates the trajectories resemblance on a figure-eight shape. Figure-eights are characterized by a closed, circular trajectory and a central crossover point. The trajectory of a perfect figure-eight can be described by a set of two parametric equations:

$$x = \cos(t) \tag{D-16}$$

$$y = \sin(2t) \tag{D-17}$$

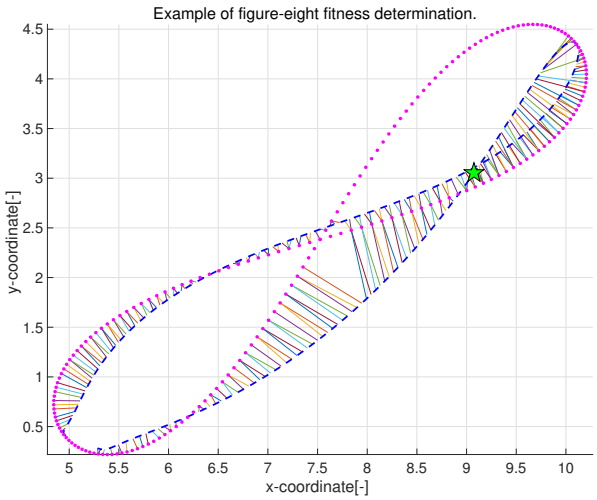



**Figure D-5:** A trajectory is compared to a true figure-eight. The colored lines display the shortest distance between each trajectory point and the true figure-eight. In this case, the mean euclidean distance equals 0.203.

To perform such an evaluation, first of all the analyzed trajectory is checked for connectedness: if the start and end point of the trajectory are significantly far apart, the trajectory is discarded as unfit. Secondly, the length $l$ and orientation $\theta$ of the trajectories principle axis is determined as well as the maximum width around this axis $T$. This axis is defined as the line between the two most distant points the trajectory. Once established, this principle length and orientation are used to generate a figure-eight with the same principle axis:

$$x = \cos(\theta) l \cos(t) - \sin(\theta) T \sin(2t) \tag{D-18}$$

$$y = \sin(\theta) l \cos(t) + \cos(\theta) T \sin(2t) \tag{D-19}$$

Finally the generated figure-eight trajectory is translated such that it matches the mechanism's trajectory. The two trajectories are then checked for resemblance by measuring the euclidian distance between each point on the mechanism's trajectory and its nearest neighbor in the figure-eight trajectory. The resulting distances are averaged after which an intermediate score $p$ is established as the reciprocal of the average euclidean distance. Finally, a separate function is called to check for the characterizing center crossing. If such a crossing exists in the trajectory, the final score equals $1.1p$. If such a crossing does not exist, the final score equals $0.1p$.

## D-4 Addendum to paper section III-A: States and action

As introduced in section A-2, reinforcement learning problems can be formalized as MDPs. An important element of such MDPs are *states* and *actions*. In this section the definition of states and actions will be given for the current application.

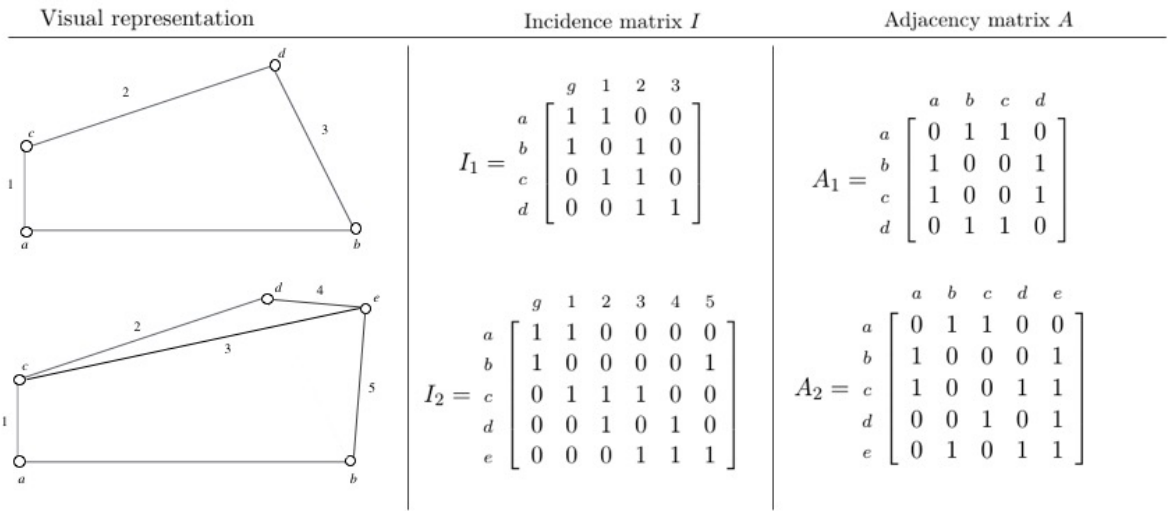



**Figure D-6:** Two example mechanisms represented visually and by means of their incidence and adjacency matrices. The horizontal bar present in both structured is referred to as the 'ground bar' and represented by the $g$ in both incidence matrices $I_1$ and $I_2$.

## D-4-1 State

The state of an MDP has to describe the current situation as accurately as possible. In this case, states refer to mechanisms throughout different steps in their development. Thus, at the start of the generative design process, the state will represent a basic four-bar linkage. After executing actions according to policy $\pi$ the agent finds itself in a new state, representing a more complex mechanism. The state should give an unambiguous description of a mechanism, such that it fully and uniquely describes it.

A full description of a mechanism minimally comprises of two types of information: topology and shape information. Topology information describes how all elements are connected. Matrices offer a great structure to store such information in, as their 2D appearance allows for a convenient 'from-to' display. Figure D-6 shows two example linkages and their respective topology information. Topological relations can be formalized in two different ways:

- Two bars are incident when they share a hinge.
- Two hinges are adjacent when they are connected to the same bar.

The incidence and adjacency relations are captured in logical matrices $I$ and $A$, shown to the right of the structures in figure D-6. Both matrices describe the same topology and therefore only one is required in the state representation. In the current implementation the incidence matrix is chosen, although using the adjacency would have been equally effective.

Both of these matrix representations are widely used in graph theory to describe networks consisting of vertices (bars in this case) and edges (hinges in this case). For more information on graph theory and its applications the reader is referred to [84].

Shape information is most easily stored by keeping track of the hinge positions using a Cartesian coordinate system. The shape information is then an $n$-by-2 matrix, $n$ being the

number of hinges. For convenience, the two sources of information are combined in a sparse state matrix $s$ of fixed size. As the mechanism becomes increasingly complex both $I$ and $H$ increase in size, filling the state matrix $s$ until pre-imposed size limits are reached.

## D-4-2 Actions

Reinforcement learning in its most basic form is a process that learns to find the best state-action-pair for each given state. Taking the action results in a new state, for which the agent may proceed to find the optimal state-action-pair again. The state representation has just been elaborated on, but what about the actions?

The mechanism representation presented in section D-1-1 utilizes two different operators: $T$ and $D$. The action space is therefore limited to these two operations. In Lipson's paper [1] these operators are accompanied by an integer value and a coordinate. The integer value describes on which bar the operator should perform its actions. The coordinate describes the position of the new hinge resulting from the operator.

Without the coordinate this action specification results in $2m$ possible actions, for $m$ available bars. The coordinates can viewed as continuous parameters, implying an infinitely large action space.

### Addendum to paper section III-B: Restricting the action space

A common procedure in reinforcement learning is a full sweep of the action space, searching for the action with the highest resulting reward. This process is only feasible when the total action space is limited and relatively small. Therefore, algorithms featuring such search procedures can only work with discrete action spaces. The Lipson operators however feature a continuous coordinate parameter. To alleviate this seeming mismatch, this relative coordinate is omitted al together and replaced by a deterministic process for placing new nodes. More specifically, new nodes are placed on a line perpendicular to the link subject to operation, crossing exactly through the middle of this link. The node is placed on a distance equal to the length of the corresponding link. The procedure prefers to create new nodes on the peripheral side of the link, but will adjust to place it within the center of the mechanism in case of repeated operations. Operations can therefore only be repeated once, since extra repetitions would lead to double node placements.

# Appendix E

# Addendum to paper section III - E and IV - D: Sensitivity Analysis

In the presented paper, reference is made to a number of different methods for determining the sensitivity of the Neural Network's inputs, the features, with respect to its output, the $Q(s, a)$ estimation. In this addendum to section III - E in the paper, these methods will be further elucidated. Finally some notes are given on the selection of the top five as a result of these methods.

## E-1 Weights method

Garson [85] introduced a method to determine the relative importance of Neural Network inputs known as the weights method, as it only uses the weight values to determine relative importance. In his method, the sum of products of normalized weights is used as a measure of importance. To normalize the weights, each input-to-hidden weight is divided by the sum of all input-to-hidden weights of the hidden node it is connected to. By normalizing Garson partitions the hidden-to-output weights into components associated with each input node. As Garson uses the absolute values of weights, this method does not correct for cancellation of weights due to sign opposition. In Gevery's review of comparison models for NNs [86] a simplified algorithm is proposed giving identical results. It consists of two simple calculations:

$$Q_{ih} = \frac{|W_{ih}|}{\sum_{i=1}^{ni} |W_i h|} \tag{E-1}$$

$$\mathrm{RI} = \frac{\sum_{h=1}^{nh} Q_{ih}}{\sum_{h=1}^{nh} \sum_{i=1}^{ni} Q_{ih}} \tag{E-2}$$

Equation E-1 describes the process of normalizing the input-to-hidden ($ih$) weights for each input $i$ in the total number of inputs $ni$. Equation E-2 describes how the relative importance (RI) can be determined based on these normalization results by summing of each hidden layer

element $h$ and dividing over the sums of each hidden and input node of $Q_{ih}$. The symbol $Q$ in these two equations is not related to the Q-value estimation of $Q(s, a)$.

This method has been later improved upon by Milne [87], Goh [88] and Gedeon [89].

## E-2 Mean weight method

The mean weight method is the most simple method of this list. It simply takes the mean absolute input-to-hidden weight of all features. It uses this value as an importance measurement. After normalizing with respect to the sum of al mean input-to-hidden weights a relative importance measure is obtained. This method, just as the weights method, relies solely on the network's weights, making it independent of certain feature values.

## E-3 Profiling method

In 1996, Lek [90] introduced a different procedure for determining the relation between input features and network outputs coined the profiling method. In his method each input variable, or feature, is studied separately by fixing all other feature values and stepping through a range of values for the selected feature. To do so, a range of possible values is determined for each individual feature and a fixed number of precision points generated. Whilst sweeping to the range of one specific variable, the other variables are subsequently set to their minimum, first quartile, median, third quartile and maximum values. Therefore five values are obtained for each point in the specific variable's range-sweep. The median of these five values is used to create a profile of the researched variable.

Since the feature values used to determine the profiling scores are all generated by the algorithm, one can say that this method is also solely reliant on the network's weights.

## E-4 Finite-difference method

The finite-difference method tests the influence of each feature value on the network's output by asserting a slight perturbation on each feature individually and witnessing the resulting change output. As the paper reads the fine-difference output is determined by a simple formula:

$$v_f = \frac{O^+ - O_-}{2\delta} \tag{E-3}$$

where $O^+$ refers to the output after a positive perturbation and $O^-$ after a negative one. The result $v_f$ can be normalized with respect the sum of all $v_f$ values to obtain a measure of relative importance. One should however take care in adopting these results: where the previous methods were solely reliant on the network's weights this method is not. The finite-difference method approximates the partial derivative of the network's output with respect to each feature. Since the network is a nonlinear function approximator, this derivative is dependent on the feature value itself. To give an example, the $Q(s, a)$ value is plotted against several values of the feature 'no_of_conns', which describes the number of connections to the selected

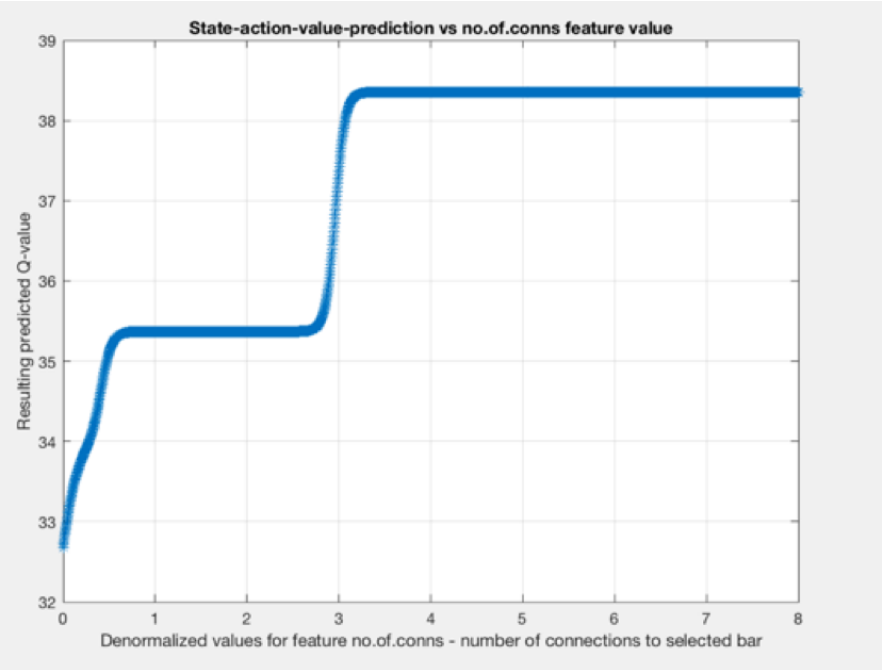



**Figure E-1:** This scatter plot relates the predicted $Q(s, a)$ value to one of the NN's features, whilst keeping all other feature values constant. The figure shows a highly non-linear response. Hence local derivatives do not generalize into the overall dependence of $Q(s, a)$ on the feature value $v_f$.

link are already present. All other feature values were kept constant. As the graph clearly shows the relation between this feature and the $Q(s, a)$ value is very much non-linear. Hence finite-difference derivatives would vary greatly depending on the value of the depicted feature: $v_f$ will be approximately zero when the feature value is 1 or 2, whereas it would be a large positive number when the number of connections equals 3. Therefore, the zoomed-in results from finite-difference analysis cannot be used to give an overall impression of the relation between the feature value and the resulting$Q(s, a)$ approximation.

Moreover some feature values are booleans or are otherwise limited to certain discrete values. In such cases, introducing small perturbations has no practical interpretation.

## E-5 Selecting the top five

The weight, mean weight and finite-difference method all result in relative importance scores for all features, form a which a top-ten can be easily extracted. For the profiling method this is slightly more difficult, since one would need to compare 2-dimensional profiles instead of single scores. Therefore the maximum variation in each profile was used as a measure of importance, which has subsequently be used to determine another top-ten.

In order to select the top five most influential features, as presented in the paper, a series of steps as performed. First of all the finite-difference method was applied to determine the relative importance of every feature four consecutive times, once after each step in a game of

mechanism design. All features present in any one of the finite-difference top-tens pertaining to these four data points are pre-selected as potentially important features. Of these features, 23 in total, the five highest rated individual features, based on the weights, mean weights and profiling method, have been presented and briefly discussed in the paper.

# Appendix F

# Addendum to paper section IV: Overview of parameter sweep results

In the paper a brief overview of the optimal algorithm settings is given. These settings are the result of an extensive grid search and parameter sweep over a total of 560 runs using 112 unique pairs of settings. A complete overview of these results is given in tables F-2 to F-5. The first column indicates the algorithm used (Monte Carlo or Temporal Difference) and the type of reward structure (episodic or end-rewards only). The second column describes which parameter is varied. An explanation of the abbreviations used is shown in Table F-1. The basic settings, which are fixed whilst changing the specific parameters shown in tables F-2 to F-5, are also tabulated in F-1. The third column shows the accumulated episodic reward obtained after the run. The final design's reward is shown in the fourth column. Finally the runtime in hours over all 5000 episodes is tabulated.

Equations F-1 and F-2 describe how parameter ER is related to the $\epsilon$-greedy parameter $\epsilon$:

$$N_0 = \frac{K}{\text{ER}} \tag{F-1}$$

$$\epsilon = \frac{N_0}{N_0 + k} \tag{F-2}$$

where $\epsilon$ is the $\epsilon$-greedy parameter, $K$ the total number of episodes in a run and $k$ the current episode number.

Equation F-3 describes how parameter HLS prescribes the hidden layer size:

$$||h|| = \text{HLS} \cdot \texttt{round}(||f||) \tag{F-3}$$

where $h$ is the vector of hidden nodes and $f$ the vector of all features. The HLS parameter describes the size of the hidden layer with respect to the input layer.

The top-5 high scorers in terms of accumulated episodic reward are:

1. TD - Episodic with Infeasibility penalty -25

**Table F-1:** Explanation of abbreviations used in tables F-2 to F-5 and basic values throughout parameter sweep

| Acronym | Explanation | Basic value |
|---|---|---|
| DR1 | Dropout rate in input layer | 0 |
| DR2 | Dropout rate in hidden layer | 0 |
| L1 | $L_1$ regularization | 0 |
| L2 | $L_2$ regularization | 0 |
| LR | Learning rate $\mu$ | 0.1 |
| ER | Exploration rate in F-1 | 100 |
| HLS | Hidden layer size according to F-3 | 0.75 |
| HLT | Hidden layer type | Sigmoid |
| Penalty | Penalty value for infeasible designs | -5 |
| Trainer | Updating algorithm used during NN training | Adam |

2. TD - Episodic with $L_1 = 0.001$
3. TD - Episodic with Infeasibility penalty 0
4. TD - Episodic with $L_2 = 0.001$
5. TD - Episodic with $L_2 = 0.0001$

The top-5 high scorers in terms of final design reward are:

1. TD - Episodic with Infeasibility penalty -25
2. TD - Episodic with LR = 0.01
3. MC - Episodic with ER = 30
4. TD - Episodic with $L_1 = 0.001$
5. TD - Episodic width HLS = 0.25

Based on these two lists the selection of recommended settings as appears in the paper was chosen. Interestingly the runtime in hours is mostly constant around 8 hours, except for experiments with large hidden layers. In all runs with HLS = 16, that is the hidden layer has 16 times more nodes than features, the runtime exceeds 10 hours. Speed differences between MC and TD cannot be extracted from the tables, since all entries are based on 5000 runs.

An example of the development of the accumulated episodic reward throughout a full run is shown in figure F-1. The figure shows a quick rise in reward throughout the first 500 episodes. The spikey behavior of the graph is a direct result of the $\epsilon$-greedy policy, resulting in random exploration actions that often result in low rewards.

**Table F-2:** Recommended settings after performing an extensive search through 112 unique setups.

| Algo - Reward | Parameter | Value | Total reward | End reward | Runtime [hrs] |
|---|---|---|---|---|---|
| MC - Episodic | DR1 | 0.1 | −18.93 | 2.09 | 6.69 |
| MC - Episodic | DR1 | 0.2 | −17.30 | 2.29 | 6.59 |
| MC - Episodic | DR2 | 0.3 | 7.39 | 4.34 | 7.21 |
| MC - Episodic | DR2 | 0.5 | 30.49 | 5.29 | 7.23 |
| MC - Episodic | L1 | 0.0001 | −2.09 | 3.34 | 8.82 |
| MC - Episodic | L1 | 0.001 | 7.82 | 5.57 | 9.27 |
| MC - Episodic | L1 | 0.01 | 20.73 | 6.59 | 8.92 |
| MC - Episodic | L2 | 0.0001 | 16.79 | 6.37 | 8.78 |
| MC - Episodic | L2 | 0.001 | 19.29 | 6.14 | 9.14 |
| MC - Episodic | L2 | 0.01 | 2.27 | 4.33 | 8.48 |
| MC - Episodic | LR | 0.001 | 2.92 | 3.77 | 8.29 |
| MC - Episodic | LR | 0.01 | 7.09 | 2.89 | 8.73 |
| MC - Episodic | LR | 1 | −18.78 | 1.39 | 7.04 |
| MC - Episodic | ER | 300 | 7.05 | 3.91 | 8.38 |
| MC - Episodic | ER | 30 | 49.36 | 21.06 | 7.95 |
| MC - Episodic | HLS | 0.25 | 19.37 | 5.50 | 9.01 |
| MC - Episodic | HLS | 0.5 | 31.78 | 14.08 | 8.57 |
| MC - Episodic | HLS | 16 | −0.80 | 1.61 | 10.41 |
| MC - Episodic | HLS | 4 | 27.27 | 9.21 | 9.21 |
| MC - Episodic | HLT | ReLU | −1.46 | 1.03 | 7.07 |
| MC - Episodic | HLT | ReLU | −32.81 | 0.14 | 5.44 |
| MC - Episodic | HLT | tanh | −24.58 | 0.14 | 5.96 |
| MC - Episodic | Penalty | -1 | 30.45 | 5.85 | 8.34 |
| MC - Episodic | Penalty | -25 | 22.95 | 6.65 | 8.46 |
| MC - Episodic | Penalty | 0 | 15.55 | 3.39 | 7.87 |
| MC - Episodic | Trainer | Adam | 11.54 | 6.04 | 7.87 |
| MC - Episodic | Trainer | SGD | 9.09 | 5.73 | 7.50 |
| MC - Episodic | Trainer | SGD NM | 12.88 | 4.32 | 8.31 |

**Table F-3:** Recommended settings after performing an extensive search through 112 unique setups.

| Algo - Reward | Parameter | Value | Total reward | End reward | Runtime [hrs] |
|---|---|---|---|---|---|
| MC - End only | DR1 | 0.1 | −5.00 | −0.89 | 7.43 |
| MC - End only | DR1 | 0.2 | −5.00 | 0.14 | 7.44 |
| MC - End only | DR2 | 0.3 | −5.00 | −0.47 | 6.73 |
| MC - End only | DR2 | 0.5 | −5.00 | 0.48 | 6.53 |
| MC - End only | L1 | 0.0001 | 3.68 | 4.71 | 7.23 |
| MC - End only | L1 | 0.001 | 2.14 | 4.20 | 7.39 |
| MC - End only | L1 | 0.01 | 1.01 | 4.41 | 7.42 |
| MC - End only | L2 | 0.0001 | 1.62 | 4.93 | 6.67 |
| MC - End only | L2 | 0.001 | −2.32 | 0.97 | 6.68 |
| MC - End only | L2 | 0.01 | 2.24 | 3.27 | 6.65 |
| MC - End only | LR | 0.001 | 8.32 | 8.32 | 8.99 |
| MC - End only | LR | 0.01 | 7.42 | 9.70 | 9.14 |
| MC - End only | LR | 1 | −5.00 | −1.92 | 7.42 |
| MC - End only | ER | 30 | 8.13 | 8.13 | 8.04 |
| MC - End only | ER | 300 | 8.03 | 11.18 | 7.28 |
| MC - End only | HLS | 0.25 | −0.14 | 2.95 | 7.83 |
| MC - End only | HLS | 0.5 | 3.51 | 3.51 | 7.98 |
| MC - End only | HLS | 16 | −5.00 | 1.09 | 9.25 |
| MC - End only | HLS | 4 | −5.00 | −0.89 | 7.53 |
| MC - End only | HLT | ReLU | −5.00 | 0.14 | 7.34 |
| MC - End only | HLT | ReLU | −5.00 | −0.89 | 6.06 |
| MC - End only | HLT | tanh | −5.00 | 0.14 | 7.25 |
| MC - End only | Penalty | -1 | 1.19 | 0.84 | 8.01 |
| MC - End only | Penalty | -25 | −0.96 | 8.07 | 7.82 |
| MC - End only | Penalty | 0 | 0.00 | −0.86 | 7.72 |
| MC - End only | Trainer | Adam | 6.41 | 7.44 | 7.94 |
| MC - End only | Trainer | SGD | 2.89 | 3.92 | 8.94 |
| MC - End only | Trainer | SGD NM | 4.61 | 4.61 | 8.89 |

**Table F-4:** Recommended settings after performing an extensive search through 112 unique setups.

| Algo - Reward | Parameter | Value | Total reward | End reward | Runtime [hrs] |
|---|---|---|---|---|---|
| TD - Episodic | DR1 | 0.1 | −28.15 | −0.97 | 7.04 |
| TD - Episodic | DR1 | 0.2 | −27.97 | 0.37 | 6.97 |
| TD - Episodic | DR2 | 0.3 | −9.20 | 1.66 | 7.93 |
| TD - Episodic | DR2 | 0.5 | −11.21 | −0.52 | 7.62 |
| TD - Episodic | L1 | 0.0001 | 41.01 | 9.22 | 7.66 |
| TD - Episodic | L1 | 0.001 | 59.65 | 19.48 | 7.74 |
| TD - Episodic | L1 | 0.01 | 33.03 | 8.97 | 7.57 |
| TD - Episodic | L2 | 0.0001 | 52.02 | 11.58 | 7.82 |
| TD - Episodic | L2 | 0.001 | 56.01 | 11.75 | 7.68 |
| TD - Episodic | L2 | 0.01 | 50.03 | 12.15 | 8.02 |
| TD - Episodic | LR | 0.001 | 16.88 | 4.48 | 8.23 |
| TD - Episodic | LR | 0.01 | 46.12 | 21.16 | 8.15 |
| TD - Episodic | LR | 1 | −18.64 | −0.62 | 7.87 |
| TD - End only | DR1 | 0.1 | −3.99 | −0.91 | 8.28 |
| TD - Episodic | ER | 30 | 21.63 | 8.20 | 7.40 |
| TD - Episodic | ER | 300 | 48.01 | 12.21 | 7.77 |
| TD - Episodic | HLS | 0.25 | 36.49 | 15.84 | 7.82 |
| TD - Episodic | HLS | 0.5 | 38.46 | 15.24 | 7.74 |
| TD - Episodic | HLS | 16 | 13.78 | 9.76 | 10.77 |
| TD - Episodic | HLS | 4 | 43.37 | 9.04 | 8.56 |
| TD - Episodic | HLT | ReLU | −29.58 | 0.14 | 6.93 |
| TD - Episodic | HLT | ReLU | −30.93 | 0.36 | 6.28 |
| TD - Episodic | HLT | tanh | −29.58 | 0.14 | 7.01 |
| TD - Episodic | Penalty | -1 | 49.35 | 11.94 | 7.70 |
| TD - Episodic | Penalty | -25 | 62.25 | 26.50 | 7.86 |
| TD - Episodic | Penalty | 0 | 57.50 | 13.22 | 8.04 |
| TD - Episodic | Trainer | Adam | 30.88 | 8.68 | 7.99 |
| TD - Episodic | Trainer | SGD | −2.18 | 2.78 | 7.92 |
| TD - Episodic | Trainer | SGD NM | 39.06 | 8.46 | 7.62 |

**Table F-5:** Recommended settings after performing an extensive search through 112 unique setups.

| Algo - Reward | Parameter | Value | Total reward | End reward | Runtime [hrs] |
|---|---|---|---|---|---|
| TD - End only | DR1 | 0.2 | −5.00 | 0.20 | 8.40 |
| TD - End only | DR2 | 0.3 | −5.00 | −2.95 | 7.24 |
| TD - End only | DR2 | 0.5 | −5.00 | 0.14 | 7.41 |
| TD - End only | L1 | 0.0001 | −5.00 | −1.92 | 7.52 |
| TD - End only | L1 | 0.001 | −5.00 | −1.92 | 7.18 |
| TD - End only | L1 | 0.01 | −3.90 | −0.62 | 7.46 |
| TD - End only | L2 | 0.0001 | −5.00 | −0.56 | 7.56 |
| TD - End only | L2 | 0.001 | −5.00 | −0.69 | 7.63 |
| TD - End only | L2 | 0.01 | −5.00 | −0.89 | 7.29 |
| TD - End only | LR | 0.001 | 7.77 | 8.80 | 7.89 |
| TD - End only | LR | 0.01 | 3.47 | 6.56 | 7.52 |
| TD - End only | LR | 1 | −3.79 | −0.70 | 8.73 |
| TD - End only | ER | 30 | −5.00 | −1.86 | 7.38 |
| TD - End only | ER | 300 | −5.00 | −0.71 | 7.29 |
| TD - End only | HLS | 0.25 | −5.00 | 0.15 | 7.77 |
| TD - End only | HLS | 0.5 | −5.00 | −2.82 | 7.62 |
| TD - End only | HLS | 16 | −5.00 | −0.18 | 11.32 |
| TD - End only | HLS | 4 | −5.00 | −3.86 | 8.13 |
| TD - End only | HLT | ReLU | −5.00 | 0.14 | 8.13 |
| TD - End only | HLT | ReLU | −5.00 | 0.34 | 7.07 |
| TD - End only | HLT | tanh | −5.00 | −0.89 | 8.16 |
| TD - End only | Penalty | -1 | 0.55 | 2.04 | 7.56 |
| TD - End only | Penalty | -25 | −13.81 | 3.11 | 8.05 |
| TD - End only | Penalty | 0 | 0.00 | 0.36 | 7.72 |
| TD - End only | Trainer | Adam | −5.00 | −0.89 | 7.51 |
| TD - End only | Trainer | SGD | 2.21 | 2.21 | 8.61 |
| TD - End only | Trainer | SGD NM | 3.12 | 3.12 | 8.86 |

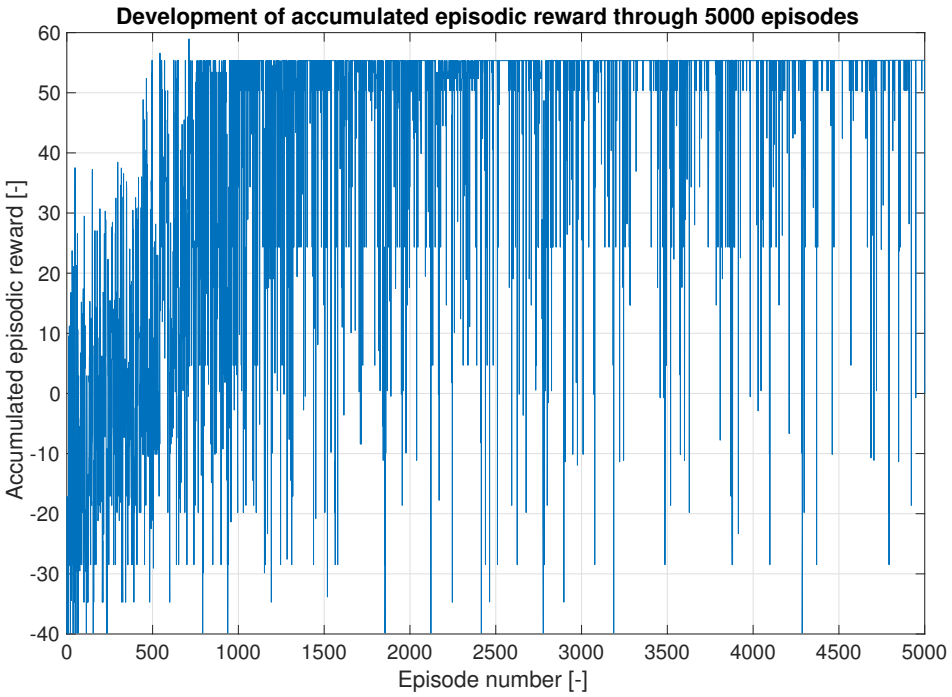



**Figure F-1:** The development of accumulated episodic rewards throughout a 5000 episode run. Due to learning the results rapidly rise, until a plateau is reached. Random actions, evoked by the $\epsilon$-greedy policy, cause the figure's curious behavior.

# Appendix G

# Literature survey

## G-1 Introduction

The ability of mankind to analyze and solve mechanical problems has driven technology throughout the ages, from the invention of the automobile to more recent advancements in micro-electromechanical systems (MEMS). Mechanism design has been long studied in an effort to discover best practices and create comprehensive systematical design methods. Ample methods have been proposed and published, of which some have became industry standards, like Pahl and Beitz' systematic design method [91].

More recently, a lot of research has been performed, most notably by Howell [43,92,93], on the subject of Compliant Mechanisms (CMs). CMs are described by Howell as mechanisms that gain "at least some of their mobility from the deflection of flexible members rather than from movable joints only" [94], or as he more wittingly put it in [95]: "If something bends to do what it is meant to do, then it is compliant." CMs offer advantages over conventional mechanisms, resulting in lowered costs and increased performance. Cost reductions have come about by decreased part count, which results in simplified manufacturing and reduced assembly efforts. Also, the decreased part count results in reduced weight, maintenance and wear, increasing reliability, precision and thereby performance [13]. Finally, CMs inherent capability of storing elastic energy can be exploited to eliminate the use of external components like stroke return springs [96], minimizing complexity and weight.

However, CM design is challenging. The main design challenge is caused by the fact that force and deflection are coupled. Consequently they cannot be considered separately as is done in conventional mechanism design. Fortunately, this has been resolved by the introduction of computer-aided design (CAD). The use of CAD in CM design has resulted a large number of successful applications [97–102].

Computer-aided design still leaves the designer with a large number of decisions to make. Therefore, the success of the final design will be heavily dependent on the designer's insights and design strategy. In an effort to find the overall best design for a given objective, the designer is required to take the whole solution space into account. Since this space is an

inconceivably large, the need has risen to perform the design process in a computerized fashion: generative design.

It is however not straightforward to tackle a design challenge with an algorithm. To do so, one requires three important elements:

- A formal description of a mechanism: mechanism representation.
- A smart search strategy: optimization algorithm
- A way to evaluate the results: a physical mechanism model.

Fortunately, for conventional mechanisms, such prerequisites can be met. As early as 1876, Reuleaux [103] attempted to classify machinery design in order to break down the complexity of mechanism design [16]. In 1966 Freudenstein and Dobrjanskyj introduced a new abstraction of kinematic structures in the form of so-called graphs [104]. After its introduction in the 1960s, graphs have been used extensively for the analysis and synthesis of mechanisms. Optimization algorithms, like topology optimization [30], have been continuously developed and numerical simulations of mechanisms have become easily accessible by Turner's introduction of the finite element analysis (FEA) [105]. With all elements in place, generative design has been demonstrated numerous times for conventional mechanisms [1, 15, 18, 25–27, 37, 106].

However, the algorithms developed for conventional mechanisms rarely apply to CMs or limit themselves to the use of topology optimization [39]. This is unfortunate, since the number of CM applications is steadily rising and so is the need for generative CM design.

Therefore, we will investigate generative mechanism design algorithms within the context of CM. More specifically the goal of this work is to create an overview of potential combinations of mechanism representations, optimization algorithms and compliance models for simulation. This will be most beneficial for mechanism designers developing generative compliant mechanism design algorithms. The overview created in this paper can be used to explore the realm of possibilities, or to narrow down the solution space in the case where one or two of the elements are already in place.

The major difficulty in this subject is the interaction between the three design elements. The elements are coupled and as such should be treated as an integral design solution. Understanding the potential synergies (or incompatibilities) between the elements requires a thorough understanding of each individual element. Also, since CM is still a young field, a lot of existing representation and optimization methods have never been used in a CM context. In this paper, the unique characteristics of compliant mechanisms have to be taken into account whilst exploring such uncultivated ground.

In chapter G-6 the findings from all three pillars are condensed into a single table. Three promising solutions are extracted and discussed in more detail.

## G-2 Method

In the subsequent chapters, numerous mechanism representation methods will be covered alongside optimization strategies and design-for-compliance research. All three of these subjects

represent an entire research field on their own. In this research however, they are combined to find possible solutions towards a single goal: developing a generative design algorithm for planar compliant mechanisms. Core values can be established for every one of these subjects, such that the traits of specific methods can be related to generative design. In each individual chapter, several methods will be evaluated based on the established values.

In the subsequent chapters, methods will be given a numeric score. Since such numeric scores can give the false impression of precision, the scores will be translated into classes: -, +/- and +. The classes are determined as follows:

$$\text{Class} = \begin{Bmatrix} -, & \text{if Score} < 0, \\ +/-, & \text{if } 0 < \text{Score} < 10, \\ +, & \text{if Score} \geq 10, \end{Bmatrix} \tag{G-1}$$

## G-2-1 Evaluating mechanism representations

In chapter G-3 several mechanism representation methods are introduced and evaluated based on their characteristics. A formal approach to characterizing the methods is applied, based on Balakrishnan and Honavar [14] and explained hereafter. The characteristics are linked to three core values, such that a character-based-evaluation can be performed.

### Mechanism representation characteristics

Balakrishnan and Honavar [14] formulated nine characteristics by which to evaluate neural networks. Kuppens [107] showed the wider applicability of this framework, also covering mechanism representations. Therefore, the same framework will be adopted in the current work. In this framework, a representation method is described by $\mathcal{R}$, the total set of solutions by $\mathcal{S}$ and the representation of a specific solution from $\mathcal{S}$ using method $\mathcal{R}$ by $r_n$ in which the nominator $n$ is used to distinguish between different solutions or different methods.

Balakrishnan and Honavar defined the following nine characteristics:

1. **Completeness:** A representation $\mathcal{R}$ is considered complete when, in principle, all possible solutions $\mathcal{S}$ can be described by $\mathcal{R}$. Or, equivalently, the set of all solutions $\mathcal{S}$ is a subset of all possible descriptions by $\mathcal{R}$. In addition a representation is only considered complete when it describes the full set of parameters involved in mechanism design, from topology to dimensions to shape.

2. **Closure:** A representation $\mathcal{R}$ is considered fully closed when every description results in a feasible solution $\mathcal{S}$. Representations that are not fully closed can be made so by adding additional constraints.

3. **Compactness:** A representation $r_1$ is considered more compact than its alternative $r_2$ when the memory size required to store the descriptions resulting from $r_1$ are smaller than those resulting from $r_2$.

4. **Scalability:** A representation's scalability describes how its compactness is affected by an increase or decrease in solution size, say by varying the number of links in a solution $\mathcal{S}$.

5. **Multiplicity:** A representation can exhibit two types of multiplicity. Firstly, a representation $R$ exhibits multiplicity when a single descriptions can decode to more than one physical solution. Secondly, a representation is considered to exhibit multiplicity when more than one description exists that maps to a single physical solution.

6. **Ontogenetic plasticity:** Whenever a representation $\mathcal{R}$ is dependent on environment variables, it is considered to exhibit ontogenetic plasticity. This can be the case when the representation is subject to learning or an environment-sensitive development process.

7. **Modularity:** A representation $\mathcal{R}$ is considered modular if descriptions resulting from $\mathcal{R}$ are allowed to contain instructions to copy and re-use parts of the description itself. As a result, modular descriptions may re-use pieces of descriptive information, resulting in higher compactness and better scalability.

8. **Redundancy:** A representation $\mathcal{R}$ is considered redundant when it contains redundant information, which can increase the representation's robustness, which may be of use in case of an error-prone translation process from description to solution.

9. **Complexity:** The complexity of a representation $\mathcal{R}$ can be interpreted in multiple way. In one specific interpretation, the complexity describes the amount of computation power required to work with representation $\mathcal{R}$. In another interpretation, the complexity refers to the structural complexity of the possible solutions that can be described by $\mathcal{R}$.

The current research describes representation concepts, not detailed methodologies. Therefore, some characteristics may not be apparent. In chapter G-3, each representation will be characterized by its completeness, closure, multiplicity, modularity and computational complexity. Scalability, ontogenetic plasticity and redundancy can only be determined once a representation $\mathcal{R}$ is fully defined and are therefore omitted from this initial overview.

### Core values

Not all representation methods may be eligible for generative mechanism design. To evaluate the eligibility, one has to know what to look for: which values are important in selecting the right method. On the subject of representation methods, three core values are defined:

- **Creative Freedom:** the ability to support any type of possible design outcome without imposing design limits. It is desired that the representation method allows as much creative freedom as possible, minimizing design limitations and prejudice towards certain solutions on beforehand. Generative design may serve as the initial creative spark after which human designers (or other algorithms) can process the spark and turn it into a feasible final design. Creative Freedom is required to uncouple this spark from our preconceived notions about CM and lead us to new designs. To evaluate Creative Freedom, representation methods will be critically analyzed in search of their design boundaries. These boundaries limit the solution space are therefore a good measure for Creative Freedom.

**Table G-1:** Cross linking representation characteristics with value drivers

| Characteristic | Creative freedom | Comprehensiveness | Fitness for numeric optimization |
|---|---|---|---|
| Completeness | + | + | ○ |
| Closure | ○ | + | ○ |
| Multiplicity | ○ | ○ | - |
| Modularity | - | ○ | + |
| Complexity | + | + | - |

- **Comprehensiveness:** the ability to take all aspects of mechanism design into account. Freudenstein and Maki [108] distinguished between structure and function of a mechanism. A mechanism's structure, or topology, describes the parts a mechanism consists of and how these are connected. The structure is a purely mathematical description, which can be enumerated in an unbiased and systematic manner. Structure however is only one part of the equation: by changing design parameters of the structure (dimensions) several mechanisms with totally different functionality may be created. We therefore do not consider representations comprehensive when they only describe topology. Only when topology, size and shape are described can a representation be comprehensive.
- **Optimization perspective:** the ability of a representation method to be deployed in combination with a numeric optimizer. Unfortunately, some processes may not be optimized as easily as others and may therefore not be suitable for use in a generative design algorithm. Whilst evaluating such processes it is important to acknowledge their incompatibility with (certain) optimization strategies. This value is not simply binary (yes/no), but may also attain intermediate values for methods that do allow for optimization but are non ideal, like computationally heavy processes.

### Evaluation

In chapter G-3 numerous representation methods are introduced. They are subsequently characterized using the list of traits in section G-2-1. In order to come to an evaluation in terms of the core values of section G-2-1, a link between characteristics and values has to be established. The relations between each characteristic and the core values are shown in table G-1. Pluses correspond to positive relations, minus signs to negative relations and circles are put in place for unrelated combinations. At the end of chapter G-3, in section G-3-5 , the values in this table will be used to translate the characteristics of each individual representation method into scores in terms of the core values.

The scores are weighted, summed and classified. The philosophy behind the choice of the chosen weights stems from deliberate consideration. Creative freedom is quintessential in the mission that generative design is supposed to fulfill: the design of mechanisms unconceivable by human effort only. This value has therefore been appreciated with a weight of 3. Optimization perspectives has been given the lowest weight (1), since in none of the representations it can become a fundamental problem. Also, optimization speeds are important in later stages, but remain a nice-to-have in the current research. Comprehensiveness is regarded more valuable,

since a lack of it would indeed cause fundamental problems: it has been granted the middle weight of 2.

## G-2-2 Evaluating optimization algorithms

A similar approach is used in evaluating optimization algorithms. In chapter G-4, five different optimization approaches are introduced and subsequently their application in generative mechanism design is discussed. Each algorithm is evaluated using a set of core values, similar to the representation methods. The core values are however chosen slightly different.

### Core values

On the subject of optimization algorithms, three core values have been established.

- **Creative Freedom:** the ability to support any type of possible design outcome without imposing design limits. This is the same value as mentioned in section G-2-1.
- **Optimization speed:** the ability to quickly converge towards the right solution. A high optimization speed makes the generative mechanism design algorithm practically usable.
- **Design compatibility:** the ability of an optimization algorithm to optimize designs. Optimization algorithms require problems to be posed in a pre-defined way in order for the algorithm to be able to optimize it. Design problems can be posed in numerous ways, which may or may not prove sufficient to work with specific optimization algorithms. Literature is of great help here: algorithms that already have been applied to mechanism synthesis are clearly compatible with design.

### Evaluation

All methods in chapter G-4 are scored directly in terms of core values. The scores are weighted and summed in a similar fashion to the mechanism representation methods. The weights are also chosen in a similar fashion: creative freedom receives the maximum weight of 3, whereas optimization speed is of minimal value (weight 1). Design compatibility is assigned a medium weight of 2, since incompatibility with mechanism design is a severe shortcoming that undermines the algorithm. Using these weights, every algorithm is assigned a score and classified accordingly.

## G-2-3 Evaluating compliance modeling methods

Chapter G-5 dives into methods for evaluating the performance of compliant mechanisms. Similar to the optimization algorithms, the compliance modeling methods are evaluated directly using a set of core values. In a typical optimization process many iteration loops will be cycled through. In these loops compliance modeling will always be performed after synthesis of a mechanism. In other words: the mechanisms design is already there when the analysis is performed. This impacts the core values for compliance modeling.

### Core values

In compliance modeling a slightly adapted set of core values will be used to evaluate the potential of compliance modeling methods within the context of generative mechanism design. These values are as follows:

- **Speed:** the ability to perform a kinematic and dynamic analysis of a compliant mechanism's behavior in a quick manner. As mentioned earlier, compliance modeling is part of an iterative process used by optimization algorithms. It is no exception that thousands or even millions of mechanisms are evaluated in order to find the optimal solution. Therefore, time spent evaluating a mechanism quickly accumulates and should therefore be minimized.

- **Accuracy:** the ability to evaluate compliant mechanism behavior in an accurate way. In the optimization process, the evaluation results stemming from the compliance modeling method will be the optimizer's single source of information. It is therefore imperative that this information is accurate. Upwards from a certain level of accuracy however, optimization results will only be affected negligibly.

- **Versatility:** the ability to analyze performance of compliant mechanisms with every shape and size. One of the main objectives in generative mechanism design is to come up with new mechanisms, presently unknown to mankind. Such mechanisms can only be evaluated properly when the accuracy of the evaluation method is reliable and thus independent of the mechanism's shape. Therefore, the modeling method should be versatile.

### Evaluation

The methods presented in chapter G-5 are evaluated directly according to the just mentioned core values. The selection of compliance modeling methods is the smallest of all three subjects with only two alternatives. Numeric scores are given in a similar fashion to the previous subjects, which are again weighted, summed and classified. The weights in this case are 1 for speed, as speed is more of a convenience then a must-have at this point in generative mechanism design research, and 2 for both accuracy and robustness, since both of these values are more important than speed but circumventable when necessary.

## G-2-4 Combination of results

The results from chapters G-3 to G-5 are accumulated in chapter G-6. Using a table, the scores of all individual methods are shown. Also, all combinations of methods are evaluated in terms of synergistic potential and scored accordingly. The resulting table can be used to select the most promising combinations of representation, optimization and compliance modeling methods.

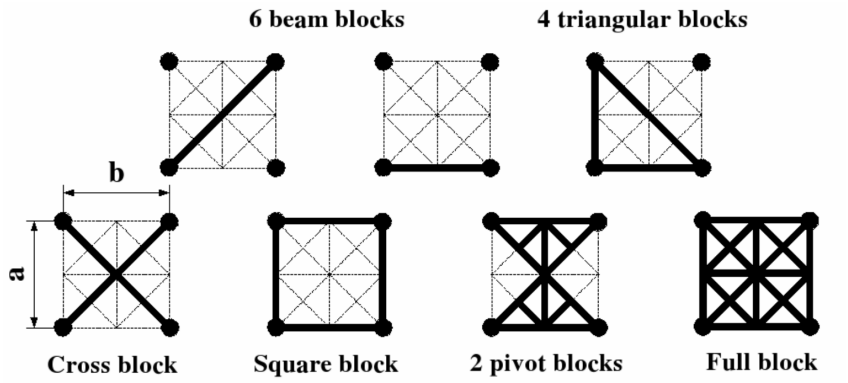



**Figure G-1:** Examples of compliant building blocks, reproduced from [17]

## G-3 Representation of mechanisms in the computer domain

Mechanisms are most easily caught in technical drawings. Unfortunately, such visual representations are not suited for generative mechanism design using numerical analysis. As Kota and Chiou [16] put it in 1992, the required representation "[...]allows for reasoning with the abstract function of the primitive mechanisms without cluttering the reasoning process with unnecessary details." To paraphrase, a representation should describe all the strictly required variables of a mechanism, but nothing more. In this chapter, an overview will be created of known representations of mechanisms within literature. The aim of the overview is to aid algorithm or mechanism designers in choosing the right mechanism representation for their specific purpose. The representations will be evaluated according to the method described in G-2-1: each of the representation methods will be characterized using a set of five characteristics as introduced in section G-2-1 and subsequently evaluated against the values from section G-2-1.

### G-3-1 Representations in discretized solution spaces

Let us denote the total universe of mechanisms that may be described by a certain representation the solution space. Some representations rely on the discretization of this solution space. In general, this means the total solution space for mechanisms is replaced by a grid-type solution space, with distinct nodes and connections. The first example of such a discretized representation is found in the building block approach.

Examples of the building block approach are available in both conventional and compliant mechanism design [16–18]. The building block approach limits the solution space of mechanisms by predefining a set of building block that the resulting mechanism can consist of. Kota [16,18] created a library of existing mechanism defined their functionality. He then proceeds to define the Motion Transformation Matrix (MTM) for each building block. Using this matrix representation, combined with constraint matrices, he was able to formalize the design process for a desired motion.

In 2002, Moon and Kota [109] proposed a mechanism synthesis method using dual-vector algebra. Screws are represented by the product of a dual-number and dual-vector, hence

**Table G-2:** Characteristics of the Building Block representation

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | × | The use of pre-defined building blocks limits the total solution space |
| Closure | ✓ / × | Constraints are required to ensure closure |
| Multiplicity | ✓ | Different descriptions may lead to the same solution |
| Modularity | ✓ | Building blocks form perfect examples of modular elements |
| Complexity | × | The modularity and limited variety of building blocks limits complexity |

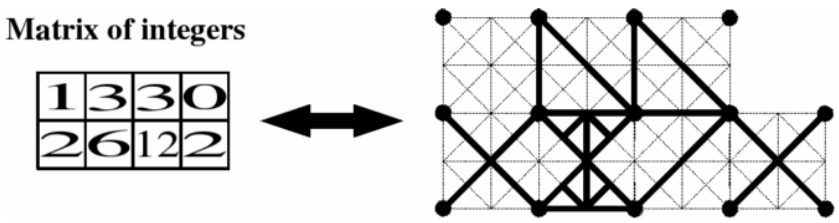



**Figure G-2:** Example of matrix representation and according structure using compliant building blocks, reproduced from [17].

the name dual-vector algebra. Screws are often used in the kinematic analysis of rigid body motions, in which the screw's line represents the orientation and direction of motion and the screw's pitch describes transformations from translational to rotational motion. Moon and Kota use screw theory's dual-number and dual-vector notation to describe a series of basic mechanisms and create a library of building blocks. The subsequent mechanism synthesis follows a similar approach to Kota's earlier work [16], although the resulting motion from combinations of building blocks is analyzed using screw theory instead of MTM matrices.

Kota's approach [16,18,109] used a library of conventional mechanisms to draw building blocks from. A compliant version of this approach was more recently demonstrated by Bernardoni et al, [17], who composed compliant building blocks from a 9-by-9 grid, as shown in G-1. After enumeration of $n$ building blocks with varying configurations, mechanisms can be described by a design matrix of integers in the $[1, n]$ range. Each entry in the matrix represents a 3-by-3 grid in the available design space, to be filled in with the block specified by the integer. Matrix entries representing grids out of the design space are filled with zeros. Empty blocks are also part of the library and are used to model the absence of material. Figure G-2 shows an example of such a design matrix and the according structure next to it.

The building block approach allows for fast optimization, since the stiffness matrix for each block can be determined in advance. After putting the blocks together, the global stiffness matrix can be deduced from its components' matrices with relative ease. As a result, one can model the behavior of a large number of structures in a relatively low-cost and fast manner: a trait especially valuable in combination with optimization algorithms. A characterization of the building block representation is shown in table G-2.

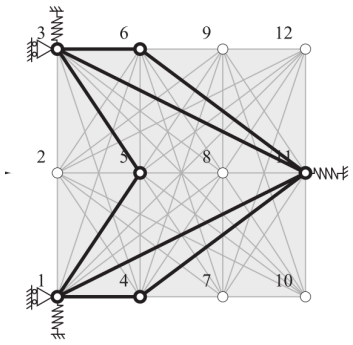



**Figure G-3:** Example of truss-based ground structure with a possible solution, reproduced from [19].

**Table G-3:** Characteristics of the truss-based ground structure representation

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | × | The use of pre-defined building blocks limits the total solution space, the representation only describes shape. |
| Closure | ✓ | Closure can be established by choosing the right building blocks |
| Multiplicity | ✓ | Different descriptions may lead to symmetrically identical structures |
| Modularity | ✓ | Building blocks form perfect examples of modular elements |
| Complexity | × | The modularity and limited variety of building blocks limits complexity |

Kawamoto [19] chose a slightly different path for the solution space's discretization. By using a truss-based ground structure (figure G-3), he limits the solution space to designs that can be represented by the grid. He then continues to use a graph-theoretical enumeration method and combines them with further design constraints to come up with a finite set of possible topologies for the required mechanism. Representing mechanisms on such a grid has similar merits to the building block approach: the total number of possible designs is brought down to a manageable number, allowing for an exhaustive search for the global optimum. When all nodes in the grid are connected to all other nodes, one can speak of a full ground structure, as used in 3D by Frecker et al. [110] . When nodes are only connected to neighboring nodes (as in Kawamoto's example), one may speak of a partial ground structure. Its characteristics are tabulated in table G-3.

The truss-based ground structure representation is not dependent on user-defined building blocks and therefore allows for more creative freedom in the structures it produces. However, this representation only provides topological information. As a result, it is not possible to perform parallel optimization of the topology and other parameters, like beam thickness or density of materials: it lacks in comprehensiveness.

Zhou [20] used a more comprehensive representation, which allows for a limited version of parameter optimization. In his research, spanning tree theory is utilized to detect and remove

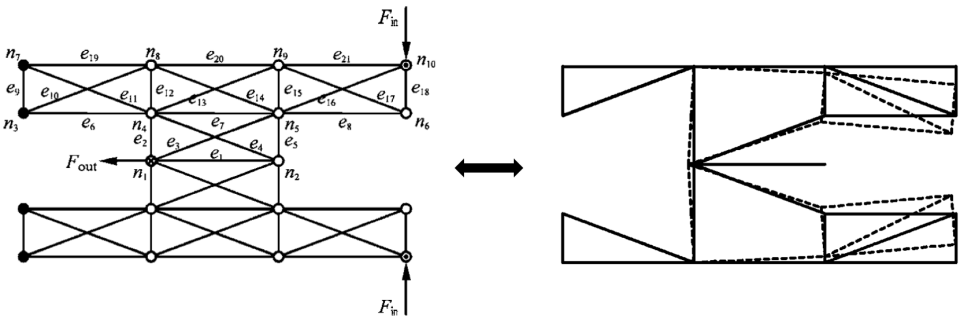



**Figure G-4:** Example of structural universal and resulting mechanism topology, edited from [20].

**Table G-4:** Characteristics of Zhou's [20] representation

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | ✓/× | The use of pre-defined building blocks limits the total solution space. The parameter tuning however increases Completeness. |
| Closure | ✓/× | Constraints are required for closure |
| Multiplicity | ✓ | Different descriptions may lead to symmetrically identical structures |
| Modularity | × | Elements are not repeated in modular fashion |
| Complexity | × | The limited number of variables limits complexity |

any invalid disconnected topologies. He then proceeds to describe the topology in a bit string, specifying beam thickness for each possible beam within the ground structure, referred to by Zhou as the structural universal. Beam thicknesses are specified by a two-bit bit-string, which limits the possible thicknesses to a set of 4 distinct values, of which one must be zero. His representation also allows for the designer to select a number of flexible nodes of which the $x$ and $y$ coordinate can be varied to induce a variety of beam lengths. Again, only four distinct values can be chosen from, as the coordinates have to be described by a two-bit element. Finally, all two-bit elements are concatenated into a $(4n + 4i)$-bit string, with $n$ the number of connections in the structural universal and $i$ the number of flexible nodes.

Zhou's approach allows for parallel optimization of topology and beam parameters. The resulting designs are highly dependent on the designer's input, as a designer still has to determine the possible beam parameters, number and position of flexible nodes and most importantly the shape of the structural universal. Zhou for example uses the structural universal from figure G-4 to create the mechanism right of it. The figure clearly shows how the choice of structural universal has impacted the final design. A characterization of Zhou's approach is given in table G-4. Cao et al. [111] expanded this approach by adding compliant links and joints to the equation. By doing so Zhou's representation's applicability is extended to the design of compliant mechanisms, rigid body mechanisms and hybrid solutions.

A density-based representation has been used extensively in topology optimization and generative mechanism design, for example by Ansola et al. [21], Saxena and Ananthasuresh [22]. Density based representations use an entirely discretized solution space, in which each sub-

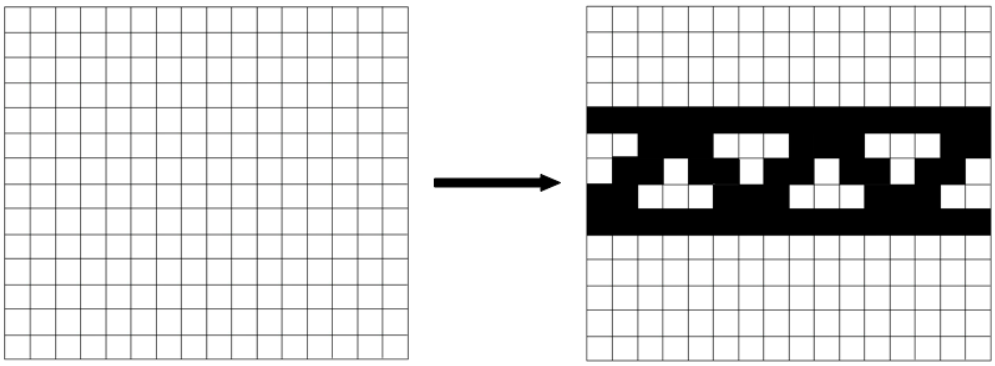


**Figure G-5:** Example of continuum structure and a possible mechanism represented by the structure

**Table G-5:** Characteristics of density-based representations

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | ✓/× | The solution space is limited in size and discretized in nature. This limits the completeness, although by choosing fine meshes it is possible to minimize this limitation. |
| Closure | ✓/× | Constraints are required for closure |
| Multiplicity | × | Descriptions can lead to one, and only one, unique solution |
| Modularity | × | Elements are not repeated in modular fashion |
| Complexity | ✓/× | Complexity may rise as mesh becomes more and more refined |

domain is assigned a density value. An example of such a solution space is shown in figure G-5. Material is distributed over the solution space's continuum structure by assigning each sub-domain (square) a binary density value: 0 or 1. Intermediate density values may also be used, but their physical interpretation is unclear. Research has been performed on the use of such intermediate 'grey' density values, which may result from optimization processes. In such cases Heavyside step functions may be applied to project grey values onto binary density values [112] and intermediate values may be penalized as in the Solid Isotropic Material Penalization (SIMP) method [31]. The visual representation shown in figure G-5 can easily be converted into a matrix or integer string containing densities. Density-based solutions are characterized in table G-5.

By using discretized solution spaces, the total number of solutions decreases from infinite to a manageable number: Kawamoto [19] was even able to find the global optimum by assessing all possible configurations. Secondly, discretized solutions can facilitate the use of building blocks who's stiffness matrices are determined in advance, making performance analysis of the resulting global mechanisms computationally cheap. Since optimization algorithms usually perform such analyses in each one of many iterations, low cost of analysis can be an important advantage of the building block approach.

Unfortunately the discretized representations share a large dependence on designer input, whether it is in the form of building blocks or structural universals. As a result, they do not

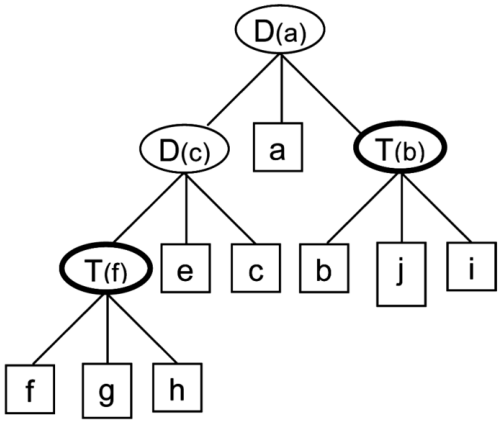



**Figure G-6:** Example of a tree-like flow chart, edited from [1].

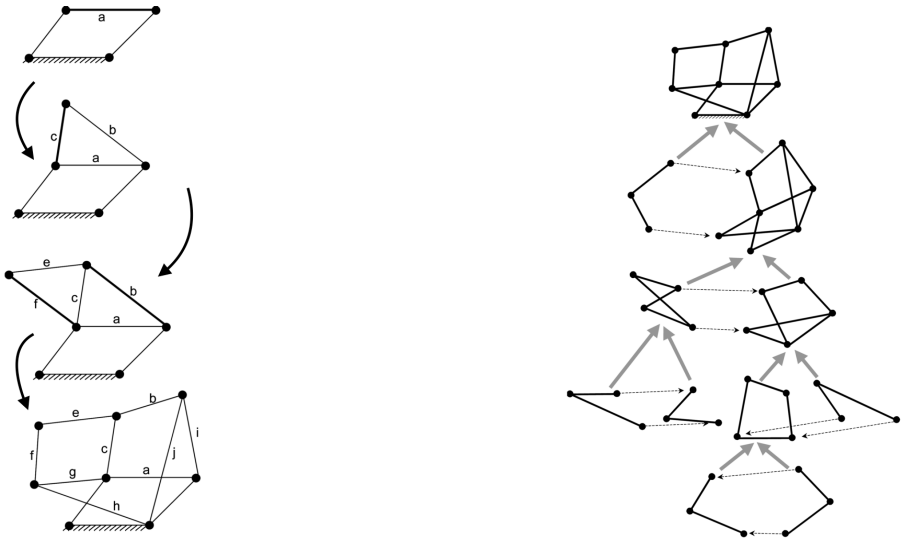
facilitate maximal *creative freedom*. Also, their *comprehensiveness* is not sufficient by default. Except for Zhou's [20] approach, the described representations did not allow the inclusion of parameters other than the mechanism's topology.

## G-3-2 Tree representation

Lispson [1] proposed a tree representation to describe a series of operations resulting in a unique kinematic chain. When drawing these operations in a schematic manner, the resulting overview resembles a tree as shown in figure G-6. Lipson distinguished only two different operators: the $\boldsymbol{T}$ and $\boldsymbol{D}$ operators. Lipson [1]: "The $\boldsymbol{D}$ operator creates a new node and connects it to both the endpoints of a given link, essentially creating a rigid triangular component. The $\boldsymbol{T}$ operator replaces a given link with two links that pass through a newly created node." The tree representation offers optimization algorithms an open design space, in contrast to the representations in section G-3-1.

Lipson proposes two different approaches to the tree representation: top-down and bottom-up. The top-down approach describes a kinematic chain by using a simple and set initial design and prescribing series of $\boldsymbol{T}$ and $\boldsymbol{D}$ operations on members of this mechanism, slowly increasing its complexity. The initial design, referred to as embryonic mechanism can be transformed to an arbitrary mechanism whilst retaining a constant number of degrees of freedom. In the bottom-up approach, the tree's leaves are the starting point. The starting point for this approach is a dyad for each tree at the far end of the tree. These dyads, named atomic building blocks, are hierarchically assembled into a single complex mechanism going up the tree whilst keeping the total number of degrees of freedom constant by merging of point pairs. Figure G-7 shows both variations of the tree representation.

Both approaches keep the total number of degrees of freedom constant throughout the process, which is no coincidence. Lipson found most conventional representations (graphs, see section G-3-4) would tend to generate over-constraint, deadlocked mechanisms. His tree representation was developed around a constant number of degrees of freedom requirement. The resulting




**(a)** Top-down approach. **(b)** Bottom-up approach.

**Figure G-7:** Lipson's top-down (left) and bottom-up (right) approaches describe the same mechanism using different starting points and subsequent sets of $\boldsymbol{T}$ and $\boldsymbol{D}$ operations. Edited from [1].

tree representation allows for optimization algorithms to explore the space of solutions without over or under constraining the mechanisms.

By selecting the embryonic or atomic building block(s) a mechanism can be fully described by a tree of $\boldsymbol{T}$ and $\boldsymbol{D}$ operators and the according local coordinates for the new node. Unfortunately, however, the proposed method only allows for the use of links and rotational hinges. As a result, it can only be used for kinematic chains. Even though kinematic chains can be powerful tools in path-generation, no springs or other higher-level components can be incorporated and no dimensional parameters are available for optimization. This limits the use of the tree representation to kinematic objectives only, limiting its *comprehensiveness* and with that the *creative freedom* this representation offers. A formal characterization is offered in table G-6.

**Table G-6:** Characteristics of the Tree representation

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | × | The operators only allow for a subset of the total solution space to be described. |
| Closure | ✓ | The resulting structure is always feasible |
| Multiplicity | ✓ | Different descriptions may lead to identical structures |
| Modularity | × | Elements are not repeated in modular fashion |
| Complexity | × | The limited number of operators limits complexity, also the constant DOF formulation automatically prevents conceiving of under- or over-constrained mechanisms. |

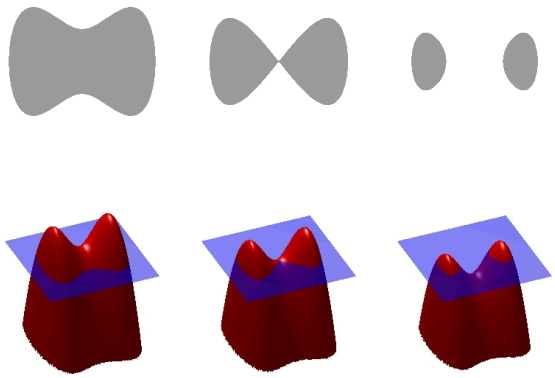


**Figure G-8:** Example of level-set function with iso-line. By adapting the iso-line level, or changing the level-set function, different material distributions may be described. Image courtesy to Wikimedia Commons.

**Table G-7:** Characteristics of level-set methods

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | ✓ | Level set functions can describe, or closely approximate, all sorts of planar mechanisms |
| Closure | ✓/× | Constraints are required to ensure feasible designs |
| Multiplicity | ✓/× | Different combinations of level-set functions and height-levels may cause multiplicity, requiring measures to counteract this |
| Modularity | × | Elements are not repeated in modular fashion |
| Complexity | ✓/× | Level-set functions are simple by default, but reverse-engineering of a level-set function description from a given design is difficult |

### G-3-3 Level-set methods

Introduced by Sethian and Wiegmann [23], level-set methods are alternatives within topology optimization to the density-based approaches mentioned in section G-3-1. In level-set methods, a function's iso-line is used to describe material distribution. Image G-8 shows how an iso-line is used to determine material distribution (the grey portions). Adapting the iso-line level changes the material distribution, as well as changing the level-set function. Even though research into this subject is relatively young, starting in 2000, already a large number of papers has been written on this subject, of which a structured reviews was given by Van Dijk [24].

Using level-set methods, smooth material boundaries can be drawn. By superimposing several local level-set functions, complex iso-lines including holes and multiple pieces can be achieved. Determining the right function for a specific design is however difficult. A characterization overview is given in table G-7

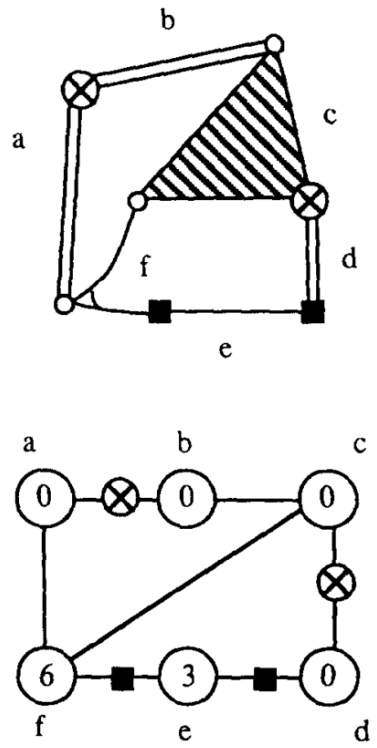



**Figure G-9:** Example of an extended graph and the compliant mechanism it represents, reproduced from [25].

## G-3-4 Graph theory

As mentioned in the introduction, Freudenstein and Dobrjanskyj [104] introduced graph theory to represent kinematic structures in 1966. Since then it has been used prolifically in generative mechanism design [15, 19, 25–29, 37]. Basic graph theory has been extended to include extra information on link and hinge types [25] and matrix and bit-string representations [15] have been developed.

In early graph representations [113] links were represented by edges and joints by vertices, such that a graph would resemble the mechanism it represents. Dobrjanskyj soon changed this method and flipped the convention, representing links by vertices and joints by edges, as present in [114]. The resulting convention results in simpler graphs, but unfortunately also removes the visual resemblance between a graph and the mechanism it represents [29]. As such, both conventions still have their specific uses. For the remainder of this paper, the latter convention is adopted, in which links are represented by vertices and joints by edges.

The basic graph has been extended by the use of edge and vertex labels [15,108] as to distinguish between different types of joints and links. Murphy et al. [25] proposed the extension of graph theory towards the inclusion of compliant members. By using labels, he created the possibility of inserting clamped connections and flexural pivots as joints (edges) and both compliant and stiff members as links (vertices). Each link is labeled with its segment compliance $s_c$, which is determined by the number of variables needed to describe the segment's motion on one end

when the other is fixed. Figure G-9 shows an exemplary graph using both vertex and edge labels to describe a compliant mechanism.

Graphs are systematic representations of mechanical systems and are therefore convenient in the enumeration and synthesis of mechanisms. For computerized synthesis however, a transformation into the numeric domain is required. In literature the means to this end is found in different types of matrices, sometimes transformed even further into bit-strings for DNA-like encoding of genotypes [26].

The most commonly used matrix representation of graphs is found to be the adjacency matrix. Tsai [115] defines the vertex-to-vertex adjacency matrix $A$ for $v$ vertices as:

$$a_{i,j} = \begin{Bmatrix} 1, & \text{if vertex } i \text{ is adjecent to vertex } j, \\ 0, & \text{otherwise (including } i = j), \end{Bmatrix} \tag{G-2}$$

where $a_{i,j}$ denotes the $(i, j)$ element of $v \times v$ matrix $A$. By definition, $A$ is symmetric and has zero diagonal elements. Using the matrix, a fully defined unique graph can be reconstructed. The same can be achieved using an edge-to-edge adjacency matrix. The adjacency matrix is label-dependent, since the position of columns and rows follows from a graph's labeling order. In literature the adjacency matrix has been used extensively, examples of which are found in [25, 27, 106].

Another popular choice of matrix [15, 26] is the incidence matrix. This matrix describes the vertex-to-edge relations, with $v$ vertices and $e$ edges in a $v \times e$ matrix $B$. Again referring to Tsai [115] for a definition of $B$:

$$b_{i,j} = \begin{Bmatrix} 1, & \text{if vertex } i \text{ is an end vertex of edge } j, \\ 0, & \text{otherwise} \end{Bmatrix} \tag{G-3}$$

The incidence matrix is has two non-zero entries in each column, since every edge connects two vertices. Likewise the adjacency matrix, each incidence matrix fully describes a unique graph.

Some authors define their own matrices by adapting an incidence or adjacency matrix. Murphy et al. [25] proposes the use of an extended adjacency matrix in which part of the parameterization is included in the matrix. The zero elements on the diagonal $a(i, i)$ are replaced with the segment compliance $s_c$ of the $i^{\text{th}}$ link. The resulting matrix is referred to as the compliant element matrix (CE). Secondly, Murphy et al. propose to substitute the nonzero non-diagonal elements in the CE matrix for a joint-type indicator integer. The conventions are summarized in G-4.

$$CE_{i,j} = \begin{Bmatrix} s_c(i), & \text{if } i = j, \\ 3, & \text{if sgement } i \text{ and segment } j \text{ are connected with a kinematic pair}, \\ 2, & \text{if segment } i \text{ and segment } j \text{ join are connected with a flexural pivot}, \\ 1, & \text{if segment } i \text{ is clamped to segment } j, \\ 0, & \text{otherwise} \end{Bmatrix} \tag{G-4}$$

Kuppens [15] created another specialized representation using graphs. He proposed to use an incidence matrix and label both axes to determine the edge type (hinge or spring) and vertex

**Table G-8:** Characteristics of the extended graph representation

| Characteristic | Yes/No | Reasoning |
|---|---|---|
| Completeness | ✓ | Using extended notations, a complete representation can be obtained |
| Closure | ✓ | By imposing the right constraints, closure can be ensured |
| Multiplicity | ✓ | Different descriptions may lead to identical structures |
| Modularity | × | Elements are not repeated in modular fashion |
| Complexity | ✓ | Graphs can become very complex and varied |

type (mass or ground). For optimization reasons, Kuppens also proposed to systematically enumerate the possible distributions of the two nonzero entries in the incidence matrix' columns. By doing so, Kuppens was able to describe each unique column configuration with an unique number. The resulting description of a graph consists of a set of integers instead of a full matrix, resolving a lot of dimensionality problems when pairing mechanisms of different sizes and complexities in evolutionary optimization. Staal [26] also uses an incidence matrix, but proposed the use of a directed graph: using 1 and -1 as nonzero entries in each column, she was able to capture the to-and-from direction of the links. No clear advantage over the use of an undirected graph was shown. Both Kuppens and Staal combined topology, represented by a graph, with parameter strings to allow for parallel optimization of both a mechanisms topology and the parameters of its elements.

As mentioned earlier, a graph describes a unique mechanism. This is, however, not true in the other direct: a unique mechanism can be described by several graphs. This behavior is coined isomorphism and forms a challenge in the enumeration of unique mechanisms, as seemingly different graphs depict the same mechanism, although labeled in a different order. Tsai [115] describes several ways of detecting isomorphism. Detection and removal of isomorphic graphs is applied widely [15, 19, 25, 27, 28, 106] as it may reduce the number of feasible designs considerably without reducing *creative freedom*. Reduction factors can in the order of $10^4$ are not unheard of [15].

The use of graph theory is a popular choice in literature and not without a reason. Freudenstein's distinction between a mechanism's structure and function defines the limited possibilities of a simple graph: it solely describes the structure of a kinematic chain. By the extensive use of labeling and the exploitation of matrix characteristics [25] graphs can be extended to contain more functional parameters. When extended notations like those of Kuppens and Staal are considered, the *comprehensiveness* of graphs can be increased to any desired level. In principle, the use of graphs does not require any restrictions from a designer and therefore has complete creative freedom. It must be noted however that creative freedom may be restricted bye prescribing the number of vertices and edges to use. A formal characterization of extended graphs is given in G-8.

## G-3-5 Conclusion

The characterizations of different representations can be compared to one another using the overview in table G-9. The characterizations have been scored systematically with respect to

**Table G-9:** Overview of characteristics per representation. Building blocks, truss-based ground structures, Zhou's [20] approach, density-based approach, Lipson's [1] decision tree, level-set methods and extended graphs.

| Characteristic | BB | Truss | Zhou | Density | DT | LSM | Ext. graph |
|---|---|---|---|---|---|---|---|
| Completeness | × | × | ✓ /× | ✓ /× | × | ✓ | ✓ |
| Closure | ✓ /× | ✓ | ✓ /× | ✓ /× | ✓ | ✓ /× | ✓ |
| Multiplicity | ✓ | ✓ | ✓ | × | ✓ | ✓ /× | ✓ |
| Modularity | ✓ | ✓ | × | × | × | × | × |
| Complexity | × | × | × | ✓ /× | × | ✓ /× | ✓ |

**Table G-10:** Scores and classifications per representation Building blocks, truss-based ground structures, Zhou's [20] approach, density-based approach, Lipson's [1] decision tree, level-set methods, graphs and extended graphs.

| Value | BB | Truss | Zhou | Density | DT | LSM | Ext.graph | Weight |
|---|---|---|---|---|---|---|---|---|
| CF | -3 | -3 | 2 | 2 | 2 | 3 | 3 | 3 |
| Comp. | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 2 |
| Optim. | 2 | 3 | -1 | 0 | 1 | -2 | -3 | 1 |
| Score | -7 | -4 | 9 | 10 | 13 | 13 | 14 | |
| Class | - | - | +/- | + | + | + | + | |

the three core values. The relation between each characteristic and value has already been established in table G-1.

The ✓, × and ✓/× symbols in table G-9 have been replaced by numeric values 2, -1 and 1. Subsequently each representation's characteristics have been multiplied with -1, 0 or 1, according to the -1, ∘ and 1 from table G-1. This results in a score per value, shown in table G-10. In accordance with the method ( G-2-1), weights are assigned to the values and the final score is determined as the weighted sum of value-scores. The numeric scores are classified according to equation G-1. The result show a close top-4 in which density methods, decision trees, level-set methods and extended graphs seem suitable for use in generative design. From the discrete methods, Zhou's approach also receives a relatively positive scores and may be taken into consideration for use in a generative design algorithm.

## G-4 Optimization algorithms

Smart mechanism design, whether for compliant or rigid-body mechanisms, requires an effective algorithm to search for the best design possible. Such algorithms, often referred to as optimization algorithms, are widely spread throughout all sorts of engineering fields. Within the context of generative mechanism design, Frecker et al. [110] and Mankame and Ananthasuresh [116] used variations of a technique called 'Topology Optimization', whereas Bernardoni et al [17] and Kuppens [15] applied an 'Evolutionary Algorithm'. The first sections

of this chapter will go in to the distinctive features of optimization algorithms and introduce a number of algorithms that could be used for generative mechanism design. As an extension, the topic of Machine Learning is covered in the last section and its potential for generative mechanism design discussed. Finally all results are summarized in a comparison table (G-16).

## G-4-1 Optimization algorithms

A vast amount of optimization algorithms have been developed over the past decades, all with their own merits and applications.

An important distinction between the different optimization algorithms can be found in their use of gradient information. Generally in optimization an objective function $f(x)$ is minimized, using the decision variables $x_i$ to do so. Basic optimization algorithms, referred to as Direct Search algorithms by Hooke [117], search an optimum by evaluating the function value $f(x)$ many times and changing search direction every time a local minimum is found. More involved approaches, like the Gradient Descent approach, also incorporate the first derivative of the objective function with respect to the decision variables$x_i$. The gradient is used to determine the optimal search direction for a given point in the objective-space. Algorithms using the first derivative of the objective function are referred to as first-order algorithms. Second-order algorithms also exist [118], which use both the first and the second order derivative of $f(x)$ with respect to $x$, like Newton's method.

The differentiability of the objective function $f(x)$ is important, since it determines the compatibility of certain design problems and algorithms. Objective functions may not be analytically differentiable, in which case computationally expensive numeric gradients are required, or may use discrete variables in which case gradient-based optimization algorithms cannot be used at all. In the case of generative mechanism design, objective functions are typically subject to discrete variables and therefore non-differentiable. This limits the scope of usable algorithms severely, explaining why in literature only two different algorithms can be found.

## G-4-2 Optimization algorithms in Generative Mechanism Design

The most frequently used optimization techniques in generative CM design can be categorized as Topology Optimization (TO) . Topology optimization, as introduced by Bendsoe [30] in 1988, determines the optimal topology and shape of a structure for a specific objective. Most TO routines use a density-based approaches like the homogenization method [30] and SIMP method [31]. In such approaches, a material distribution is determined within a discretized design space. The distribution can be discrete, such that each element is either filled with material or not, or continuous, such that a range of densities can be assigned to each element. The elements are analyzed using FEA, as introduced in section G-5-3. A comprehensive overview of continuous density-based approaches is given by [119]. Alternatively, the level-set approach can be applied. This approach implicitly describes the geometrical material boundaries using the iso-contours of a specified function: the level-set function. An extensive review on this subject is given by Van Dijk [24].

Besides TO, evolutionary algorithms have been applied to mechanism design. Bentley [120] defined evolutionary algorithms (EA) as the overarching definition for genetic algorithms

(GA), evolutionary programming (EP), evolutionary strategies (ES) and genetic programming (GP). GA were first introduced by Holland [32] and emulate nature's evolutionary forces of selection, combination and mutation to fulfill their objectives in all sorts of engineering fields. EA is clearly inspired by nature and uses terms and theory from biology to describe the optimization process. Solutions ('individuals') are formally described by a bit string ('genotype') to encode their phsycial appearance ('fenotype') and are part of a larger group ('population'). The performance of solutions is improved by simulation of nature's mating behavior: new individuals are created by recombination of existing genotypes. A stochastic merit order is applied, providing strong genes with the best chance of reproduction. By introducing random variations in the genotypes, new search directions are opened up for investigation.

An important notion in EA is that of exploitation versus exploration [121]: by selective reproduction the traits of strong individuals are exploited and developed, by random mutation and recombination the solution space is explored. As a metric, diversity is used, since it decreases with exploitation and increases with exploration. Premature convergence to local optima can be prevented by maintaining the right level of diversity in a population [122].

In the next sections, the specific relation between both optimization methods and generative mechanism design will be treated. Both sections conclude by referring to the three core values mentioned in section G-2-2: *creative freedom*, *Optimization speed* and *Design compatibility*. Each section will end with a scoring table. At the end of this chapter, an overview table is presented.

### Topology optimization and generative mechanism design

TO is conventionally used to design structures. It is therefore primarily applied to design for maximum stiffness, minimizing weight in structures. Since TO results in monolithic designs, it was not used in the generative design of conventional, multi-part, mechanisms. However, as compliant mechanisms research developed, Ananthasuresh et al. [123] used TO to design flexible structures by adapting the objective of the optimizer. Maximum stiffness was no longer the goal and replaced by one of the functional objectives presented in section G-5-2. Sigmund subsequently designed compliant grippers with TO using a truss-structure discretization [124] and Larsen et al. [125] used TO to design a compliant mechanism with multiple input and output ports using continuous TO.

Sigmund [36] explains that conventional TO applications often rely on optimality criterion methods to solve the TO problem. Such methods are very efficient when taking into account a large amount of variables to solve simple objective functions under only one constraint. Sigmund however posed his compliant mechanism TO problem using a complex objective with multiple constraints. He therefore resorted to a different optimization algorithm, known as sequential linear programming, or SLP.

Furthermore, Sigmund reports of two TO-related problems specifically important within the context of mechanism design. First of all, TO using square elements is prone to 'checkerboard': optimal solutions showcasing alternated voids and filled elements in a checkered fashion. Both Diaz et al [127] and Jog et al. [128] showed how kinematic constraints on the elements resulted in checkerboard patterns with unrealistic high stiffness. In the context of compliant mechanisms, Yin and Ananthasuresh [11] showed how TO for minimal strain energy also

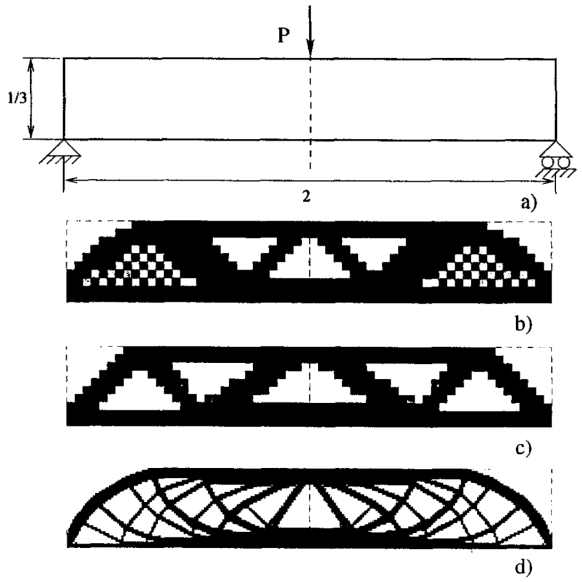



**Figure G-10:** Example of numerical instabilities in TO. The same problem (a) is solved with different resulting solutions. Solution (b) shows a clear checkerboard. Between solutions (c) and (d) only the number of elements was changed, clearly showing the optimizer's mesh-dependency. Image reproduced from [126]

stimulates checkerboard-like behavior in the form of point flexures with artificially low strain energy. Secondly, Sigmund addresses the issue of mesh-dependency: TO results depend on how the solution space is discretized. Both of these issues are depicted in figure G-10. They can both be addressed by regularization. A sensitivity filter was introduced by Sigmund [36] to perform this regularization. Deepak et al. [129] performed a comparison study with five different objective formulations to compare their design performance, numeric efficiency and handling of point flexures.

Looking at the core values, TO does not allow full creative freedom: the discretization limits the design space and mesh-dependency makes the optimization dependent on human input. If, however, computational cost would be no issue and meshes could be made infinitesimally fine, then TO would allow for full creative freedom. Furthermore, optimization speeds in TO are governed by the number of elements in the discretization and quickly drops as finer meshes are used. Finally, TO is perfectly compatible with design objectives, as demonstrated [36, 42].

### Evolutionary algorithms and generative mechanism design

In literature, several examples [1, 15, 17, 26, 28, 40, 106] are available of evolutionary algorithms being applied to mechanism design. In these examples, EAs are applied on a population of randomly initialized mechanism designs. The mechanisms are represented using a genome string, which can take up different forms. Kuppens [15] used a graph representation and flattened it to a string by applying the schema theorem [130]. Lipson's [1] based his genomes on the issued decision tree operators. Bernardoni [17] built the genome out of 2-by-2 matrices containing integers indicating building blocks.

**Table G-11:** Characteristics of Topology Optimization

| Characteristic | Score | Reasoning |
|---|---|---|
| Creative freedom | ✓/× | Topology optimization limits freedom by limiting and discretizing the design space. Furthermore, mesh-dependency causes large dependence on human input. Only with large computational power can these issues be overcome. |
| Optimization speed | ✓/× | TO requires FEA for each iteration. Especially with fine meshes, this induces heavy computational costs and thus low speed. |
| Design compatibility | ✓ | TO is perfectly compatible with generative design as shown in several examples |

**Table G-12:** Characteristics of Evolutionary Algorithms

| Characteristic | Score | Reasoning |
|---|---|---|
| Creative freedom | ✓ | Evolutionary Algorithms are very robust and widely applicable. They therefore do not limit the creative freedom. |
| Optimization speed | × | As a probabilistic method, evolutionary algorithms have to run large numbers of iterations, limiting their convergence speed. |
| Design compatibility | ✓ | EA is perfectly compatible with generative design as shown in several examples |

Parsons [35] points out that EAs are characterized by the fact that they search the design space using a population of solutions instead of performing point-search with only one solution. Furthermore, they rely on the objective function rather than gradients and on probabilistic rules, instead of the traditional deterministic rules [32]. As such, EA is very robust and applicable in many situation: they are in general not limited by discontinuous search spaces, or the non-existence of derivative information. Looking back at the core values of optimization introduced in chapter G-2-2, EA's robustness and versatility allows it to obtain maximal creative freedom. The lack of gradient information and a deterministic approach however can cause low convergence speeds. Finally, its compatibility with design challenges has already been proven [1, 15]. An overview of the scores is shown in Table G-12.

### G-4-3 Machine Learning

The field of Machine Learning originates from computer-science and has started to develop in the 1950s with the first neural network, built by Minsky and Edmonds, and Arthur Samuel's checkers playing machine [5], demonstrated on television in 1956 (figure G-11).

An overview will be given of the three main functional categories that exist within the realm of ML and their applicability to generative mechanism design. These categories are supervised, unsupervised and reinforcement learning.

**Figure G-11:** A 1956 television demonstration of Arthur Samuel's Checkers program, played against by Robert Nealey on the IBM 7094. Nealey lost from this milestone program. Image courtesy to IBM.

### Supervised learning

In Machine Learning, most applications are based on learning by experience. In such applications a machine trains by internalizing many examples, consisting of inputs and their according outputs. During training, the machine develops a model of the relation between inputs and outputs it witnesses in the examples. Subsequently, the model can be used to predict outputs for new input data.

This type of training, requiring a large dataset of data with known inputs and corresponding outputs, is called supervised learning. Such complete datasets are known as labeled data. Examples of supervised learning are ample in image detection [54], traffic-flow predictions [55] and medical predictions [56]. The most important algorithms in supervised learning are Support Vector Machines (SVM), Neural Networks (NN), Gaussian Process Regression (GPR), Random Forests and linear classifiers like regression models.

Generally, a labeled data set is split in two parts: a training set and a testing set. During supervised learning, the algorithm is trained on the training set and subsequently tested on the test set. Since the test true outputs are known, the algorithm's accuracy can be evaluated.

### Supervised learning and generative mechanism design

Supervised learning algorithms are only feasible when large amounts of labeled data can be made available. Within the context of generative mechanism design, such large datasets are not ready available as-is. Theoretically, if large numbers of successful designs and objectives were to be made available in a consistent format, generative mechanism design could be posed as a supervised learning problem. Such a library however is not available as-is and is not expected to be so within reasonable time.

Secondly, mention of *creative freedom* was made in section G-2-1 as a core value for generative design approaches. However, supervised learning promotes learning based on existing examples.

**Table G-13:** Characteristics of Supervised Learning in generative mechanism design

| Characteristic | Score | Reasoning |
|---|---|---|
| Creative freedom | × | Learning from examples limits creative freedom. |
| Optimization speed | - | No fair comparison can be made with regards to optimization speed. |
| Design compatibility | × | Posing generative design as a supervised learning problem is difficult if not impossible. |

It is therefore unlikely, although not impossible, for a supervised learning algorithm to generate creatively new mechanisms when it is trained on human-designed examples. Using supervised learning therefore defeats one of the purposes of computer-aided generative mechanism design. Furthermore, supervised learning is not compatible with generative design and can therefore not be compared in terms of optimization speed. An overview of scores is given in Table G-13.

### Unsupervised learning

Jain [57] distinguishes unsupervised learning by the absence of category information, or labels, in the target data set. Even with the limited remaining information, unsupervised learning algorithms are able to detect patterns and describe hidden structures in the data. Since no labeled data is available, unsupervised learning's accuracy cannot be evaluated quantitatively. Unsupervised learning is mainly used for data clustering and anomaly detection. Data clustering alone is being applied to a broad range of topics [57], ranging from character recognition in handwriting [58] to customer segmentation for marketing purposes [59].

### Unsupervised learning and generative mechanism design

Unsupervised learning, in contrast with its supervised variant, does not suffer from the lack of creative freedom that learning from examples entails. Its main trait however is the unraveling of patterns or structures in large amounts of data, which is unfortunately not available within the context of generative mechanism design. Therefore, unsupervised learning will not be applicable to generative mechanism design in the strict sense.

There may, however, be applications once a design algorithm is up and running and large numbers of potential designs can be generated. In such a case, unsupervised learning may be applicable for clustering the solution space and/or automatically remove anomalies resulting from numeric instabilities.

The inapplicability of unsupervised learning to generative mechanism design makes a fair comparison in terms of creative freedom and optimization speed impossible. Therefore a clear 'no' can be assigned. An overview is given in Table G-14

### Reinforcement learning

Reinforcement learning (RL) is decribed by Kaelbling et al. [34] as behavioral learning by trail-and-error interactions with a dynamic environment. Sutton [60] gave an even more formal

**Table G-14:** Characteristics of Unsupervised Learning in generative mechanism design

| Characteristic | Score | Reasoning |
|---|---|---|
| Creative freedom | × | Learning from examples limits creative freedom. |
| Optimization speed | - | No fair comparison can be made with regards to optimization speed. |
| Design compatibility | × | Posing generative design as a supervised learning problem is difficult if not impossible. |

definition: 'Reinforcement learning is the learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal'. RL can be applied when no labeled data is available, but outputs can be given a measure of success. In such a case, a machine can learn by taking actions, assessing the results and adapting its perceptions accordingly. Through iteration, the machine can learn to take the right actions resulting in the highest rewards.

Rewards play an important role in reinforcement learning. The machine's goal is to maximize its expected cumulative reward over (pseudo)time. Machine's therefore take into consideration the effect of their current move on their cumulative reward by making predictions of future rewards [61]. Instantaneous rewards are usually valued more than possible future rewards, which is referred to in literature as the problem of delayed rewards [62]. The trial-and-error search and delayed rewards are the two most distinguishing features of reinforcement learning [60] and make it suitable for sequential decision making.

Because of its distinctive character, RL finds applications in different types of fields than (un)supervised learning. Applications range from learning a robot to walk [63] to playing ATARI games [64] and minimizing elevator waiting time [65].

#### Formalized description of RL

RL problems are usually posed in the form of a Markov Decision Process (MDP) [68]. MDP's are denoted with a tuple $(S, A, P, \gamma, R)$ containing five elements:

- $S$: a set of all possible states in which the actor may be.
- $A$: a set of actions the actor can take.
- $P$: the state transition probabilities. $P_{sa}(s')$ gives the probability of transitioning to state $s'$ by performing action $a$ in state $s$.
- $\gamma$: the discount rate for future rewards.
- $R$: the reward function.

In an MDP, the actor starts out in a state $s_0$ and takes an action $a_0$. It transitions to state $s_1$ under probability $P_{s_0 a_0}$. From this new state, it can take a new action and keep repeating

this cycle. In each state, the actor receives a reward $R(s)$. The expected accumulative reward is subject to devaluation over time and is described by:

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_3) + ...\right] \tag{G-5}$$

As equation G-5 shows, the rewards from future states are discounted for every time step $t$ with factor $\gamma^t$.

A set of defined state-to-action decisions is called a policy $\pi$. $\pi$ maps $S \rightarrow A$. The expected rewards of policy $\pi$, starting in state $s$ are described by $V^\pi(s)$. $V^\pi(s)$ can be solved [68, 70] for each state $s$ by the set of Bellman equations:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s') \tag{G-6}$$

The right-hand side of equation G-6 can be seen as a mathematical operator $T^\pi$, mapping $V^\pi(s')$ to $V^\pi(s)$. The fixed point theorem [69] shows that $V^\pi$ can be found by iteratively applying operator $T^\pi$ to a initial value function $V$.

In RL, the objective is to find a policy $\pi^*$ that maximizes the value function $V(s)$ by prescribing the optimal action $a$ for each state $s$, resulting in $V^*(s)$. Again, $V^*(s)$ can be derived by applying $T^*$ iteratively to an initial value function $V(s)$. Once $V^*(s)$ is known, $\pi^*$ can be computed. This method is coined value iteration [70].

Alternatively, one can apply a random initial policy $\pi$ and iteratively update the policy until the optimum is found: policy iteration. In practice, value iteration is most commonly applied [68].

In most applications, $P$ and $R$ are not known a-priori. In case of a limited number of possible states and actions it may be sufficient to perform numerous trial runs, and estimate the maximum likelihood state transition values $P_{sa}(s')$ by inspecting how often actions $a$ in state $s$ led to state $s'$. In practice however, the number of states is often continuous and therefore infinite, like the length of a link or a stiffness value. As such, the number of states is inexhaustibly large and trials cannot cover them all. For low-dimensional problems, applying discretization to the state-space may offer solace. Unfortunately, this leads to inaccuracies by assuming the value function $V(s)$ constant over each discretization interval. Secondly, the number of discretization points grows exponential with the state-dimension, making discretization an unpractical solution for states with more than $\approx$2 dimensions. As an alternative to discretization, one may use a model or simulations to perform the state transition according to the transition probabilities. When no such model is available, one may resort to fitted value iteration. In this method $V(s)$ is sampled for a finite number of states, effectively creating labeled data. Using a supervised learning algorithm, a function for $V(s)$ can be approximated.

### Reinforcement learning and generative mechanism design

In generative mechanism design, fully described mechanisms will have to be generated at some point and tested for performance. Such a process fits well with reinforcement learning's approach of trial and error, using the mechanism's performance metric as reward signal. The sequential character of RL, with series of actions and states, could be put to work by seeing

**Table G-15:** Characteristics of Reinforcement Learning in generative mechanism design

| Characteristic | Score | Reasoning |
|---|---|---|
| Creative freedom | ✓ | Trial-and-error learning excludes every form of designer-input. |
| Optimization speed | ✓/× | Reinforcement Learning is known as computationally expensive. |
| Design compatibility | ✓/× | Posing generative mechanism design as a sequential decision making process is uncultivated ground, although not decisively impossible. |

each completed design as one step along a trial-and-error road towards the optimal mechanism. Alternatively, a sequential mechanism representation may be adopted, like Lipson's [1] decision tree approach in order to grow a mechanism whilst rewarding the algorithm with each new decision.

In terms of creative freedom, Reinforcement learning is an excellent choice. By nature, reinforcement learning algorithms develop intuiting and learn all by themselves, without any designer input. This makes them as creatively free a can be. Secondly, reinforcement learning is known a computationally expensive way of learning, and as such convergence speeds will not be high. Finally, the largest challenge for reinforcement learning in generative mechanism design will be its compatibility. Generative mechanism design is usually not a sequential decision making process. Therefore the generative mechanism design might be posed in an unconventional way to fulfill the requirements of reinforcement learning. As no proof points of this approach are available, success can not be guaranteed.

In the past paragraphs, ML has only been introduced briefly. Detailed explanation of ML techniques is beyond the scope of the current research. Interested readers are referred to [131] and [132].

### G-4-4 Overview

In the past sections, several optimization methods were addressed and reviewed within the context of generative mechanism design. For evolutionary algorithms and topology optimization examples from literature have been provided, proving their usefulness within the generative mechanism design realm. An outreach towards machine learning has been provided, from which unseen alternatives may arise. All five optimization methods have been scored on the core values as introduced in section G-2-2. An overview of these scores is given in Table G-16. The scoring convention from chapter G-2-2 has been adopted: 2 points for a ✓, 1 point for a ✓/× and -1 point for a ×. Weights have been provided in the table according to section G-2-2. Missing scores, like the supervised learning score for *Optimization speed*, receive zero points. The scores are classified according to the rules in G-1.

**Table G-16:** Overview of characteristics optimization algorithms. TO = topology optimization, EA = evolutionary algorithms, SL = supervised learning, USL = unsupervised learning and RL = reinforcement learning.

| Characteristic | TO | EA | SL | USL | RL | Weight |
|---|---|---|---|---|---|---|
| Creative freedom | ✓/× | ✓ | × | × | ✓ | 3 |
| Optimization speed | ✓ /× | × | - | - | ✓/× | 1 |
| Design compatibility | ✓ | ✓ | × | × | ✓/× | 2 |
| Score | 8 | 10 | -5 | -5 | 9 | |
| Class | +/- | + | - | - | +/- | |

## G-5 Modeling of compliant behavior

As described in the introduction, the generative design of mechanisms comprises of three major steps: representing a mechanism in the computer domain, determining a mechanism's behavior and finally optimizing the mechanism's design. The previous chapter gave an overview of representation methods: the first step. In this chapter we create an overview of methods for determining mechanism behavior by modeling its kinematics. The chapter starts out with some background on compliant mechanisms, different types of compliant structures and possible optimization objectives. Sebsequently the two major categories of modeling methods will be elaborated upon.

Compliant mechanisms are intrinsically different from conventional mechanisms, most importantly because of their inherent stiffness and resulting force-deflection coupling. In his book about Compliant Mechanisms, Howell [13] identifies several difficulties in designing and analyzing CMs. Mainly, the compliant mechanism's deflection as a result from applied forces is dependent on the location and magnitude of the applied forces. Also, since most compliant mechanisms display large deflections, linear beam does not suffice for describing the behavior of deflecting members. More complex descriptions involving elliptic integrals [133] give accurate results for simple mechanisms, but become very involved for more complex mechanisms. Implicit solutions are therefore adopted, which are introduced in the upcoming sections.

### G-5-1 Distributed versus lumped compliance

One of the major distinctions in compliant mechanisms can be made between distributed and lumped compliance. The distinction is based on a mechanism's effective source of flexibility: a mechanism's flexibility can be distributed along its members, like in bending beams, or it can be sourced in specialized small portions of the mechanism. In the first case, one may speak of distributed compliance, whereas the latter is referred to as lumped compliance (see figure G-12).

In lumped compliance pivot points can be distinguished, which act as flexural hinges. Stresses in such mechanisms are highly localized, limiting the allowed range of motion of such mechanisms [11]. Still, many design synthesis exercises have resulted in such lumped compliance mechanisms.

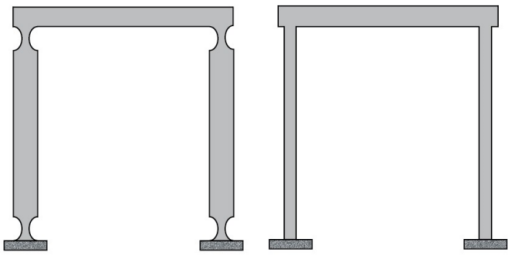



**Figure G-12:** (a) lumped compliance (b) distributed compliance. Reproduced from [39].

Yin and Ananthasuresh [11] explain that most synthesis methods produce lumped compliance mechanisms because of their dependency on flexibility-stiffness formulations. By using these formulations, the strain energy is minimized during the optimization process. Strain energy will be lowest when using revolute joints, since they generate large motion with zero strain energy. Optimization algorithms therefore exploit the revolute joint behavior in order to achieve maximum displacement while minimizing strain energy by lumping all compliance in a number of flexure hinges. Design of flexural pivots has been, and still is, researched extensively [134, 135] and pseudo-rigid-body-models of lumped compliance using torsional springs have been proposed by Howell [92].

Hetrick [41] proposed the use of stress-constraints in order to force solutions to adopt distributed compliance over lumped compliance. Yin and Ananthasuresh [11] proposed an alternative algorithm, introducing a new objective function that appreciates uniform deformation throughout the structure. The algorithm constraints the local relative rotations in the structure, penalizing the use of concentrated compliance.

### G-5-2 Design objectives in compliance

An important notion in compliant mechanism design are the contradictory goals flexibility and stiffness. Flexibility is desired in order to create structures that can achieve large deflections when actuated. Stiffness however is also required to retain structural integrity of the mechanism and withstand reaction forces stemming from its application. Therefore, compliant mechanism design requires consideration of multiple objectives [35].

Flexibility in compliant mechanisms is described by a mechanism's *mechanical advantage*, as proposed by Sigmund [36]:

$$M = \frac{F_o}{F_i} \tag{G-7}$$

This metric characterizes the way a mechanisms translates an input displacement $u_i$, as the result from input force $F_i$, into an output displacement $u_o$ with the resulting output force $F_o$. Mangesha and Lu [37] use the mechanical advantage's geometric counterpart: the geometric advantage or GeoA (G-8) when describing the performance of a specific compliant mechanism.

$$\text{GeoA} = \frac{u_o}{u_i} \tag{G-8}$$

When synthesizing compliant mechanisms, maximization of these advantage metrics can be used as design objective, as in [36] and [37]. In other applications, the input motion is fixed and as such the absolute output motion itself can be optimized for [20], instead of a ratio. In all of these single-objective optimizations, the stiffness objective is implicitly introduced as a constraint during optimization.

Shield and Prager, in 1970, [38], introduced the concept of Mutual Potential Energy (MPE): the potential energy of a structure subjected to two different sets of loadings. Frecker et al. [110] applied MPE to compliant mechanisms: the external forces being the first load and a dummy load in the direction and position of the prescribed displacements being the second. In doing so, the MPE becomes a measure for the structure's flexibility. Ananthasuresh and Kota applied this metric to the synthesis of compliant mechanisms in 1994 [123], while applying constraints to ensure the required structural stiffness.

Ananthasuresh proposed to use the total strain energy (SE) in the system as a measure for a CM's stiffness: the more strain energy in the mechanism the lower its stiffness. Ananthasuresh proposed a multi-criteria optimization method, using a weighted linear combination of the objectives, to find solutions with maxium MPE and minimum SE. Unfortunately, the absolute values of both objectives can be of totally different orders, leaving one of the two to dominate over the other. Finding the right weight factors to balance the objectives has shown to be difficult and cannot be generalized. Frecker et al. [110] therefore propose an alternative in which the ratio between MPE and SE is maximized. Saxena and Anantasuresh [22] extended the ratio with powers $m$ and $n$, as in equation G-9.

$$\frac{\mathrm{MPE}^m}{\mathrm{SE}^n} \tag{G-9}$$

Others, like [40, 116], searched for the optimal CM to for path generation, minimizing the least-squares error function as their objective.

Elastic deformation in compliant mechanisms cause changes to its kinematic characteristics. The mechanical advantage of a CM will however remain relatively constant when a CM's internal strain energy is lowered as shown by Salamon and Midha [136]. Hetrick and Kota [41] therefore proposed an alternative, energy-based approach. By minimizing the internal strain energy during parameter optimization, they provided "near-constant kinematic properties while operating under a broad range of input deflections and external loads."

## G-5-3 Simulating kinematics in optimization context

During optimization, the performance of candidate-mechanisms needs to be evaluated. Thus, its kinematics have to be determined. The kinematics of mechanisms are very difficult to capture in exact analytical expressions. Fortunately, alternative approximate solution methods are available. In literature, two distinctive approaches to the modeling of compliant kinematics are being used extensively: the Finite Element models and Pseudo-Rigid-Body models.

### Finite Element Analysis

In literature, a large portion of all compliance modeling is performed using Finite Element Analysis (FEA), also known as Finite Element Method (FEM) [17, 20, 37, 39–42]. In this

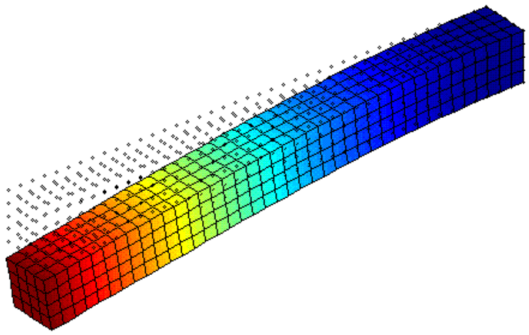


**Figure G-13:** Example of Finite Element Analysis of beam. The beam is meshed in cube-shaped elements. Displacements, stresses and strains are subsequently calculated for each cube and combined. Image courtesy to http://www.codedevelopment.net/3dbeam_brick_sm.html.

method, a continuous structure is modeled as a collection of discrete elements of finite length. Elements can be of all shapes and sizes, but are usually geometrically simple shapes like beam and bar elements. When performing the analysis, the behavior of each separate element is determined by solving the partial differential equations governing elastic deformation. The equations are solved exactly in a set of precision points, referred to as nodes, and interpolated using shape functions throughout the structure. As long as the element sizes are small, a pseudo-continuous motion from loading can be modeled.

### Linear and non-linear FEA

Within Finite Element Analysis, a major distinction can be made between linear and non-linear FEA. The differential equations governing elastic deformation are captured by the deceptively simple looking equation G-10.

$$\mathbf{Kd} = \mathbf{f} \tag{G-10}$$

In this equation, $\mathbf{K}$ is the stiffness matrix, $\mathbf{d}$ the displacement vector and $\mathbf{f}$ the force vector. When forces and stiffnesses are known, the displacement can be calculated using the inverse stiffness matrix $\mathbf{K}^{-1}$. The stiffness matrix can be determined for every separate element. The elemental matrices can be assembled into the object's overall stiffness matrix $\mathbf{K}$. This approach is referred to as the direct stiffness method and is used in most commercial FEM packages. It was first introduced by Turner [105]: Boeing's head of the Structural Dynamics Unit.

In linear analysis, one assumes linear stiffness behavior in the elements. In that case, the direct stiffness method can be applied to find $\mathbf{K}$, which can subsequently be used for the rest of the analysis. As such, $\mathbf{K}$ is constant and only needs to be computed once. This type of linear FEA is computationally inexpensive and will take no more than a few minutes to execute, even for very large shapes [137].

In reality however, several reasons exist for which stiffness values can significantly change throughout analysis: Geometrical nonlinearities may occur when loading results in large

deflections, material nonlinearities may occur when strains surpass the linear elastic region in their stress-strain curve or nonlinear stiffness behavior could result from contact stresses resulting from displacements [138]. In all these cases, the stiffness matrix requires updating throughout the analysis, requiring an iterative approach towards a solution. When this is the case, one speaks of nonlinear FEA. Nonlinear FEA is much more computationally expensive, since the stiffness matrix $\mathbf{K}$ and its inverse $\mathbf{K}^{-1}$ need to be re-calculated in every iteration.

The use of linear FEA may be perfectly justifiable when nonlinear effects are minimal, for instance when no contact stresses can occur and deflections remain relatively small. When designing compliant mechanisms however, large deflections will be present and so may be contact stresses. Therefore, linear FEA can not be used for compliant mechanisms [139]. Mentioning of FEA in this report should always be read as nonlinear FEA, except for when explicitly mentioned otherwise.

### FEA in generative compliant mechanism design

Finite element analysis is used on a wide-scale in combination with truss-based ground structures or cellular micro-structures, usually referred to as topology optimization and first applied to compliant mechanisms by Ananthasuresh [123]. In topology optimization, a compliant mechanism is synthesized on a ground structure and subsequently modeled using FEA. The resulting kinematic behavior forms the basis for a qualitative assessment of the mechanism and is, as such, an important input for the optimization algorithm.

Finite element analysis determines a mechanism's behavior based on elementary deformation theory, usually applied on a large number of elements. Since FEA models loading behavior based on fundamental physical principles, it captures compliant behavior well and requires no upfront knowledge of compliance in order to run a simulation. This makes FEA a versatile method, but also implies high computational cost. Engineers should therefore be aware of the method's limitations and choose parameters, such as element type and size, wisely.

### Pseudo-rigid-body models

As early as 1996, Howell and Midha [43] wrote that 'the design of mechanisms often requires iteration between synthesis and analysis procedures.' Analysis of compliant mechanisms proved difficult and timely, since no exact solution can be found and FEA approximations would take a long time to run. Howell and Midha therefore proposed a new method in which a compliant mechanisms is accurately modeled by an equivalent rigid-body mechanisms. This so-called pseudo rigid body (PRB) method introduces all the well-known rigid-body mechanism theory to be used in a compliant mechanism design context, including exact solutions and low computational costs.

Howell and Midha distinguish between two classes of compliance synthesis: Rigid-Body Replacement synthesis and Synthesis with Compliance.

### Rigid-Body Replacement synthesis: kinematics only

Rigid-Body Replacement synthesis is the simplest of the pseudo-rigid body approaches and suffices when one is only concerned with kinematics. In Rigid-Body Replacement, a rigid-body

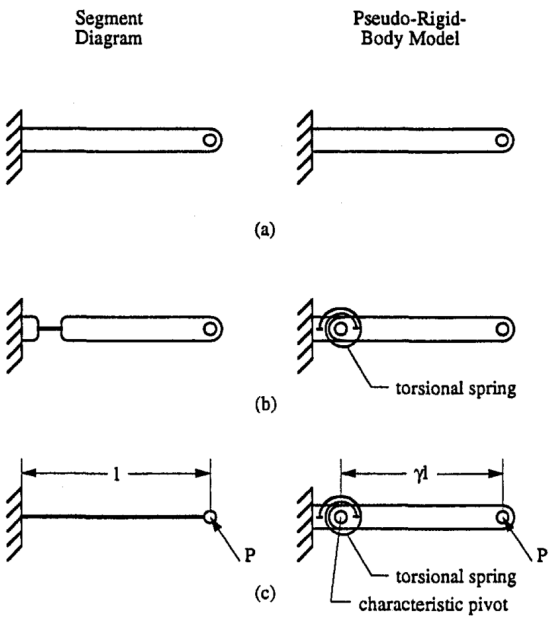



**Figure G-14:** Example of pseudo-rigid-body models of well-known compliant members. (a) a rigid link, (b) lumped compliance, modeled by a torsional spring, (c) continuous compliance modeled by a spring and characteristic pivot. Reproduced from [43].

mechanism synthesis is performed to determine the geometry of the mechanism, after which the structural properties of the flexible members can be chosen to meet the required stress and loading behavior. The biggest challenge is to determine the appropriate pseudo-rigid-body replacement mechanism. Also, validating design feasibility is an important aspect, since mechanisms that are feasible as pseudo-rigid-bodies may become infeasible when translated to a compliant mechanism. A well-known example of such an infeasible compliant design is the inclusion of flexural pivots, modeled by hinges, deflecting more than a full rotation in the rigid-body model. Constraints should be applied during optimization to prevent the arise of such infeasible design.

Rigid-Body Replacement synthesis is the simplest of the pseudo-rigid body approaches and suffices when one is only concerned with kinematics. In this type of synthesis, mechanism geometry is based on a rigid-body design. Rigid members can be substituted for flexible counterparts to meet required stress and loading behavior. The main challenges in this approach are the determination of the appropriate pseudo-rigid-body replacements and the validation of design feasibility. The latter can be seen in a simple example: flexural pivots may be modeled by hinges while the range of motion of a flexural pivot is limited compared to that of a hinge. Therefore, constraints should be applied during optimization to prevent the arise of infeasible design.

#### Synthesis with Compliance: kinematics and dynamics

Synthesis with Compliance is a more advanced pseudo-rigid-body method and takes dynamics into account, as well as kinematics. The inclusion of dynamics enables Synthesis with

Compliance to be used in more sophisticated design tasks, like input/output force amplification or constant torque motion. By taking the energy stored in compliant members into account, desired force, torque and energy characteristics can be specified upfront and designed for. Matthew and Tesar [44] have presented such an extended synthesis method by giving analytical formulations for the synthesis of springs in planar mechanisms in order to provide balancing properties. Roth and Huang [45] presented a method for designing spring elements to match known external forces. Such methods can be utilized in synthesis with compliance. More recently, Pucheta and Cardona [46] applied synthesis with compliance to the design of bistable compliant mechanisms, which can be stated as 'a position synthesis problem where the energy storage characteristics are specified.' Rosenberg et al. [47] use an energy approach to design a PRB mechanism, which translates into a statically balanced compliant mechanism.

#### Pseudo-rigid-body models for lumped and distributed compliance

PRB models can be used to model both lumped and distributed compliance mechanisms. Lumped compliance mechanisms use flexural pivots, which show close resemblance with spring loaded hinges. Therefore, Howell [92] proposed the use of a torsional spring with stiffness $\frac{EI}{l}$ as derived in beam theory [140]. The torsional spring replaces the flexural member and is located at half it's length.

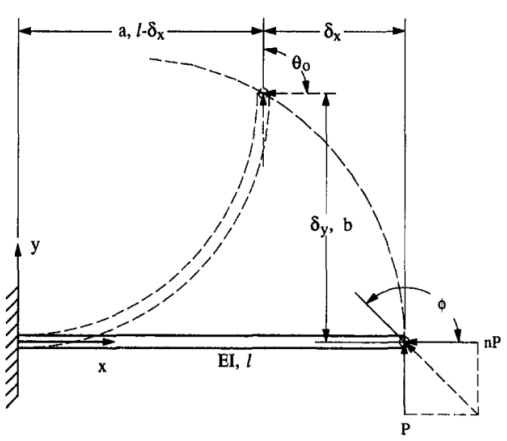
For distributed compliance, more advanced PRB models are proposed by Howell and Midha [93]. In distributed compliance, a flexure bends like an end-loaded large-deflection beam, as depicted in figure G-15a. The PRB model proposed by Howell and Midha is shown in figure G-15b. The nearly-circular path of the large-deflection beam is approximated in the PRB by two rigid links, joined in a spring loaded pivot. The torsional spring yields a non-linear stiffness characteristic to resemble the end-loading. The location of the characteristic pivot is chosen such that the path of large-deflection beam's endpoint is approximated during rotation. The resulting factor between the location of the pivot and the beam length $l$ is referred to as the characteristic radius factor $\gamma$. Together with the spring stiffness, this is the most sensitive design parameter. Howell and Midha [93] have developed an analytical framework for the determination of the optimal design parameters and resulting approximation errors. Su [141] extended Howell's work by proposing a PRB model with three revolute joints, allowing accurate approximation of deflections over 120° with characteristic radi $\gamma_0$=0.1, $\gamma_1$=0.35, $\gamma_2$=0.40 and $\gamma_3$=0.15. Saxena [142] and Yu et al. [143] also extended the PRB model by including end moments besides the end force.

### G-5-4 Comparison between FEA and PRB

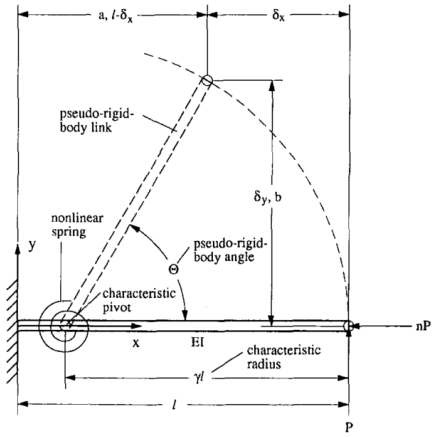
Both FEA and the PRB method can be used to analyse kinematics in a compliant mechanism. There are however pros and cons to both of these methods in regards to the values described in section G-2-3: Speed, Accuracy and Robustness.

#### Speed

FEM analysis loses speed when large numbers of elements are required for accurate modeling. Computational time can be limited by choosing larger elements, or optimizing element size




**(a)** End-loaded large-deflection beam. Reproduced from [93].




**(b)** PRB model of distributed compliance. Reproduced from [93].

across the mechanism, but loss of accuracy may be the result of such an action. Also, lowering the number of elements makes the FEM prone to the exploitation of loopholes like checker-boarding [42]. This mesh-dependent behavior is unwanted [36].

PRB models, on the other hand, are governed by a set of explicit equations and can therefore be solved directly. Direct solutions are much quicker to compute than the implicit type of solution FEA offers.

### Accuracy

Using a FEM, a highly accurate analysis of the kinematics of monolithic compliant structures can be achieved. Especially by choosing small elements, the effects of discretization become negligible and the resulting model highly realistic. Furthermore, the accuracy is not dependent on the design: in theory any compliant monolithic structure can be modeled and analyzed with high accuracy, no matter the design.

Pseudo rigid body models can achieve high accuracies when applied correct. Howell and Midha [93] report path approximation accuracies of over 99.5 % for large deflections (up to 77°). They have provided parametric equations describing the beam end angular deflection and load-deflection curves, as well as tables with solutions [93]. For complex mechanisms however, requiring sequential PRB models, the inaccuracies start to add up. Secondly, high-accuracy PRB models are available for certain elements only, like flexures and flexural pivots. When more exotic monolithic shapes are being designed, accuracies become uncertain, or PRB models might not even be available.

### Versatility

The FEA method is very versatile, as long as computational costs are assumed workable. As mentioned in the previous section, the effects of discretization become negligible as long as elements are chosen sufficiently small. Assuming an infinitely strong computer, one could say

**Table G-17:** Characteristics of FEA and PRB models for compliance

| Characteristic | FEA | PRB | Weight |
|---|---|---|---|
| Speed | × | ✓ | 1 |
| Accuracy | ✓ | ✓ | 2 |
| Versatility | ✓ | × | 2 |
| Score | 7 | 4 | |
| Class | +/- | +/- | |

that using FEM every mechanism can be accurately modeled, independent of shape and size. This is as close to the definition of versatility (section G-2-3) as one can get.

In the pseudo-rigd-body approach however, a library of rigid-body replacements of compliant elements is used to simulate compliant behavior. In case of lumped compliance, with distinct members, such a library is highly useful. This library however limits PRB to the use of pre-conceived sub-structures when modeling a compliant mechanism. When exotic compliant mechanisms, featuring distributed compliance and an organic shape, require analysis, the accuracy of the library's building blocks deters. As such, PRB is less versatile than FEM.

#### Conclusion

In a nutshell, FEM is more generally applicable and accurate, but is computationally very costly. PRB models, on the other hand, are not as accurate in general as FEM, but are computationally a lot more efficient. As such, both methods have their merits and only in combinations with optimization procedures and mechanisms representations will their be a best pick.

The assessment given in the last paragraphs has been summarized and displayed in table G-17. The weights are distributed according to the method described in section G-2-3.

## G-6 Results

The current research has thus far centered around three subjects main subjects that need combination in order to create an operational generative compliant mechanism design algorithm. The three subject discussed are:

- Mechanism representations (chapter G-3)
- Optimization algorithms(chapter G-4
- Compliance modeling (chapter G-5)

For each subject literature has been studied and numerous solution alternatives have been provided. These alternatives have subsequently been subject to evaluation as to determine to what extent they address the core values stipulated in chapter G-2. Especially for the

mechanism representations the core value scores spread out widely. In this final section, the aim is to bring the separated information from chapters G-3 to G-5 together and create an overview of synergies or incompatibilities between solution combinations. Together with the core value scores of the mechanism representations, this overview should indicate the most promising sets of combinations for generative mechanism design algorithms.

## G-6-1 Overview table

To capture all three subject in a two dimensional overview, the mechanism representations and compliance modeling methods have been combined. This results in 14 pairs of representation methods and compliance modeling methods. Each of these combinations has been scored in terms of synergy ranging from $-$ for a mismatch to $+$ for synergistic combinations. Intermediate combinations have been scored $+/-$ and incompatibilities have been denoted with a $\circ$.

All 14 pairs have subsequently been combined with the selection of optimization algorithms presented in chapter G-4. The resulting 70 combinations all received a similar synergy score.

In parallel, the individual scores based on chapter G-2's values should be taken into account to determine which combinations of solutions are both synergistic and endorsing the core values.

The resulting table is shown in table G-18.

Without going in to much detail, the granted synergy scores will be discussed in the following set of short paragraphs. Each paragraph covers one horizontal set of scores: one for each representation method. Supervised and unsupervised learning are not mentioned in these paragraphs, since they are incompatible with all representations for reasons explained in section G-4-3.

The building-block approach uses pre-defined building blocks, of which the kinematics and dynamics are assumed to be independent and known. The kinematics and dynamics of resulting mechanisms are determined by super-positioning principles, which is why two 'not applicable' $\circ$ symbols were inserted. Topology optimization methods (like SIMP) are not applicable, since only building blocks can be selected. Reinforcement learning is not applicable since no sequential decision making is occurs. Evolutionary Algorithms are a fitting choice, since it can handle this type of discrete optimization and building blocks can be arranged in a DNA string.

Truss-based grounds structures consists of numerous individual members, connected at distinct nodes. Applying FEA can be as easy as handling every truss as an element. A pseudo-rigid-body model does not benefit from this characteristic. The combination of FEA and TO is a proven one and therefore assigned the highest value. Using EA is also possible, since the description of a mechanism in a truss-based ground structure can be translated to DNA with discrete optimization possibilities. In combination with PRB.

Zhou's method is an extension of the truss-based ground structure. It therefore receives the same type of scores in terms of synergies. However its score as a representation method is higher than the truss-based ground structure's, mostly because of the increased creative freedom resulting from Zhou's introduction of tunable parameters.

The density method is classically used in topology optimization. It requires a discretized geometrical solution space, which makes it a great fit with FEA. Using the density methods,

**Table G-18:** Overview table of mechanism representations, compliance modeling and optimization methods and their mutual synergies. Scores from individual analysis based on core values are provided. Representations: BB = Building Block method, Truss is Truss-based ground structures, Zhou is Zhou's [20] method using a flexible truss-based ground structure description, Density = density-based approach, DT = Lispon's [1] decision tree description, LSM = level-set method, Ext. graph = Extended graph representation, using labels to enhance comprehensiveness of graphs. Sub-zero scoring optimization algorithms and mechanism representations have been grayed out.

| Representations | | Compliance | | Optimization | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | +/- | + | - | - | +/- |
| Method | Class | Method | Synergy | TO | EA | SL | USL | RL |
| BB | - | FEA | ○ | ○ | + | ○ | ○ | ○ |
| | | PRB | ○ | ○ | + | ○ | ○ | ○ |
| Truss | - | FEA | + | + | +/- | ○ | ○ | ○ |
| | | PRB | +/- | - | +/- | ○ | ○ | ○ |
| Zhou | +/- | FEA | + | + | +/- | ○ | ○ | ○ |
| | | PRB | +/- | - | +/- | ○ | ○ | ○ |
| Density | + | FEA | + | + | - | ○ | ○ | ○ |
| | | PRB | - | - | - | ○ | ○ | ○ |
| DT | + | FEA | + | ○ | + | ○ | ○ | + |
| | | PRB | +/- | ○ | + | ○ | ○ | + |
| LSM | + | FEA | + | + | - | ○ | ○ | ○ |
| | | PRB | - | - | - | ○ | ○ | ○ |
| Ext. graph | + | FEA | +/- | ○ | + | ○ | ○ | ○ |
| | | PRB | + | ○ | + | ○ | ○ | ○ |

semi-organic shapes can be acquired instead of distinct members. This is a desirable option, since it allows for creative freedom, but such organic shapes are difficult to describe with PRB.

The decision tree algorithm is a special character in the family of representations. Mechanisms are described by a string of subsequent decisions that transform a base shape into the desired structure. The resulting mechanism consists of distinct members, making it a good fit with FEA. The representation does not provide any sort of ground structure on which to perform TO, which is therefore incompatible. EA can however be used, as Lispon [1] showed. Interestingly, the decision tree representation is the only method that uses a sequential description, making it the only good fit with reinforcement learning.

Representing a structure as an iso-line in level-set methods stems from the family of topology optimization. It therefore logically fits well with FEA and TO. The resulting shapes from LSM are usually organic, making it difficult to model with PRB.

Finally, ther is the graph representation. Just as with DTs, this representation does not provide any geometrical framework for TO and is therefore incompatible with it. Graphs can be translated into DNA to work with an evolutionary algorithm, as shown by for example Kuppens [15]. Since graphs describe distinct connections and members, they can be easily modeled using PRB. In case of FEA, graphs have to be translated to a physical design, discretized and analyzed: a cumbersome solution.

## G-6-2 Promising combinations for generative design

Representation methods and optimization algorithms with sub-zero core value scores are discarded as viable options for generative mechanism design and grayed out in table G-18. Their synergistic values may still be valuable for use in other applications. From the remaining unmarked rows and columns, several high scoring combinations can be found in table G-18. In this final section, three possible combinations are extracted and their scores elaborated on.

### Option 1: LSM + FEA + TO

The level-set method scored high on Creative Freedom because of its independence of any predefined structure or size. As such, it has limitless freedom in determining a 2D shape for a compliant mechanism. Since such shapes are most likely organic, finding a correct pseudo-rigid-body replacement will be very difficult. Using FEA however, such shapes can be meshed and analyzed although computationally it may be challenging. Level-set representations are already used in topology optimization, which is therefore a strong synergistic optimization strategy. Relating the performance of a mechanism back to the level-set equation, and improving this equation based on the mechanism's behavior will be challenging.

### Option 2: DT+PRB+EA/RL

The decision tree method, developed by Lipson [1], features mechanisms described by a series of $\boldsymbol{T}$ and $\boldsymbol{D}$ operators in combination with some dimensionality parameters attached to them. Since it creates mechanism with distinct members, compliant versions would be most easily modeled using pseudo-rigid-body synthesis. Since each mechanism can be described

with a bit-string (a series of operators and parameters), this method will work well with Evolutionary Algorithms: the bit-string can serve as DNA for the evolutionary processes. Besides evolutionary algorithms, a more innovative approach may be taken by exploiting the 'subsequent decisions character' of this representation in a reinforcement learning optimizer. In such a machine learning approach, a machine learns through trial and error what series of decisions, or operators, lead to an ideal mechanism for a pre-defined design goal. Reinforcement learning has never been applied to this type of algorithm, which makes this both an exciting and a challenging exercise.

#### Option 3: Ext. graph + PRB +EA

Thirdly, the extended graph representation scored highest on the core values, because of its large creative freedom and comprehensibility scores. Graphs describe mechanisms as distinct members, which fits well with the pseudo-rigid-body synthesis model. In literature, graphs have been translated into integer strings [15], which are very well suited for use as DNA in evolutionary approaches. Murphy et al. [25] have shown an example of how graph descriptions can be extended to include compliant members. Alternatively, one could use the extended graphs to directly describe pseudo-rigid-body mechanisms that can be optimized using an evolutionary algorithm and finally translated to a fully compliant design by inversing the pseudo-rigid-body synthesis procedure. In order to find truly great solutions, the EA will have to be carefully conditioned in order to strike the right balance between exploration and exploitation. This is identified as the largest challenge.

## G-7 Discussion

The results from chapters G-3 to G-5 have been condensed into a single table G-18, from which three promising algorithm designs were extracted. Interestingly, two out of the three promising configurations break precedent on several issues. Most daringly, the use of level-set methods and reinforcement learning in mechanism synthesis are both unheard of, even though their resulting evaluation scores are high.

This can be explained by the method used in this research. The method proposes dividing the large challenge of generative design into subproblems regarding mechanism representation, numeric optimization and compliance modeling. Solutions for these subproblems have been researched separately from the greater problem. As a result, solutions applied in other research fields (other large problems) have been introduced in the context of the current research and have shown to be promising. Further investigation will have to prove whether or not these solutions can flourish in the current contextual atmosphere.

The current solutions are the result of thorough literature study. It is however possible that solution directions have been accidentally overlooked. In case of such an event, the method description allows other researcher to complement table G-18 in the spirit of this research with new additions.

Coming to the research results, objective facts were used for subjective argumentation in the evaluation of each and every method. The arguments found their basis in the values described in G-2. The choice of values, even though they were selected based on a foundation

of arguments, largely influence the evaluation decisions. The current method may be extended for use in different contexts by adapting the core values to the needs of the specific application it is being used for.

The results in this research are obtained under certain assumption. Over time however, such assumptions may no longer be valid and therefore the value of this research declines as it ages. One major assumption is that no other and better alternative within each individual research subject is available. In time however, new technologies may be developed that are much more suited for (for instances) optimization of mechanism designs. Fortunately, such new methods can be added to the overview in table G-18 and as such keep the research relevant. Also, computational power has been assumed limited, whereas limits in this specific case seem to fade quickly. If the cost of computational power keeps declining, the negative value associated with it in this research should also decline, in which case several scores will need to be revised.

## G-8 Conclusion

In this literature survey we have taken an in-depth look into a large number of methods and techniques that can be used when developing a generative design algorithm for compliant mechanisms. The methods and techniques have been clustered into three subjects: mechanism representation, compliance modeling and optimization. Individual methods and techniques have been scored within each subject based on a set of core-values for generative design algorithms presented in chapter G-2. These scores have been collected and tabulated in an extensive overview table (table G-18). The table's interior is filled with synergy scores, describing the potential of a certain combination of techniques. The scores should be taken as qualitative assessments of each combination of methods. Designers of generative design algorithms can profit from table G-18 in twofold: the value scores of individual methods may be used to enforce designers in decision making on specific elements of their algorithms, whereas the synergy scores may aid designers in selecting the final algorithm elements to supplement their project.

Also, the method by which table G-18 has been constructed, and the argumentation used to provide scores have been elucidated in this research. As such, the table is not only relevant at the time of writing, but may be kept relevant by adding new evaluations of new technologies in the spirit of the original research.

Besides the overview table (G-18) this research has also led to the extraction of three potential generative design algorithms that break precedent and take the values presented in the method (chapter G-2) into account.

# Bibliography


[1] H. Lipson, "Evolutionary synthesis of kinematic mechanisms," *AI EDAM*, vol. 22, no. 03, pp. 195–205, aug 2008.

[2] R. Carbonneau, K. Laframboise, and R. Vahidov, "Application of machine learning techniques for supply chain demand forecasting," *European Journal of Operational Research*, vol. 184, no. 3, pp. 1140–1154, 2008.

[3] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[4] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," in *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. IEEE, 1999, pp. 371–377.

[5] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[6] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. M. Schwartz, and J. Makhoul, "Fast and robust neural network joint models for statistical machine translation." in *ACL (1)*, 2014, pp. 1370–1380.

[7] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[8] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European Conference on Computer Vision*. Springer, 2016, pp. 649–666.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015.

[11] L. Yin and G. K. Ananthasuresh, "Design of Distributed Compliant Mechanisms," *Mechanics Based Design of Structures and Machines*, vol. 31, no. 2, pp. 151–179, 2003.

[12] Q. He and L. Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 1, pp. 89–99, 2007.

[13] L. L. Howell, *Compliant Mechanisms*. John Wiley & Sons, 2001.

[14] K. Balakrishnan and V. Honavar, "Properties of Genetic Representations of Neural Architectures," *Proc. of the {W}orld {C}ongress on {N}eural {N}etworks ({WCNN}'95)*, pp. 807–813, 1995.

[15] P. Kuppens, "Automated Robot Design With Artificial Evolution," Msc. Thesis, Delft University of Technology, 2016.

[16] S. Kota and S.-j. Chiou, "Conceptual Design of Mechanisms Based on Computational Synthesis and Simulation of Kinematic Building Blocks," *Research in Engineering Design*, vol. 4, pp. 75–87, 1992.

[17] P. Bernardoni, C. Bidard, and C. E. A. Drt, "A new compliant mechanism design methodology based on flexible building blocks," vol. 5383, pp. 244–254, 2004.

[18] S.-j. Chiou and S. Kota, "Automated conceptual design of mechanisms," *Mechanism and Machine Theory*, vol. 34, pp. 467–495, 1999.

[19] A. Kawamoto, M. P. Bendsøe, and O. Sigmund, "Planar articulated mechanism design by graph theoretical enumeration," *Structural and Multidisciplinary Optimization*, vol. 27, no. 4, pp. 295–299, 2004.

[20] H. Zhou and K.-L. Ting, "Topological Synthesis of Compliant Mechanisms Using Spanning Tree Theory," *Journal of Mechanical Design*, vol. 127, no. 4, p. 753, 2005.

[21] R. Ansola, E. Veguería, J. Canales, and J. A. Tárrago, "A simple evolutionary topology optimization procedure for compliant mechanism design," *Finite Elements in Analysis and Design*, vol. 44, no. 1-2, pp. 53–62, 2007.

[22] A. Saxena and G. K. Ananthasuresh, "Topological synthesis of compliant mechanisms using the optimality criteria method," *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pp. 1900–1910, 1998.

[23] J. A. Sethian and A. Wiegmann, "Structural boundary design via level set and immersed interface methods," *Journal of computational physics*, vol. 163, no. 2, pp. 489–528, 2000.

[24] N. P. Van Dijk, K. Maute, M. Langelaar, and F. Van Keulen, "Level-set methods for structural topology optimization: A review," *Structural and Multidisciplinary Optimization*, vol. 48, no. 3, pp. 437–472, 2013.

[25] M. D. Murphy, A. Midha, and L. L. Howell, "The topological synthesis of compliant mechanisms," *Mechanism and Machine Theory*, vol. 31, no. 2, pp. 185–199, 1996.

[26] I. Staal, "Evolutionary mechanisms: Automated synthesis of robotic mechanisms by an Evolutionary Algorithm," Ph.D. dissertation, 2014.

[27] L. I. Duanling, Z. Zhonghai, and C. Guimin, "Structural Synthesis of Compliant Metamorphic Mechanisms Based on Adjacency Matrix Operations," *Chinese Journal of Mechanical Engineering*, vol. 24, no. 4, pp. 522–528, 2011.

[28] E. Kranendonk, "Masters Thesis: A Practical Approach to Evolutionary Algorithm based Automated Mechanism Design."

[29] D. Olson and A. Erdman, "An Algorithm for Automatic Sketching of Planar Kinematic Chains," vol. 107, no. March 1985, pp. 1–6, 1985.

[30] M. P. Bendsoe and N. Kikuchi, "Generating Optimal Topologies in Structural Design Using a Homogenization Method," *Computer Methods in Applied Mechanics and Engineering*, vol. 71, pp. 197–224, 1988.

[31] M. P. Bendsøe, "Optimal shape design as a material distribution problem," *Structural and multidisciplinary optimization*, vol. 1, no. 4, pp. 193–202, 1989.

[32] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artificial intelligence*, vol. 40, no. 1-3, pp. 235–282, 1989.

[33] Y. Rahmat-Samii, "Genetic algorithm (GA) and particle swarm optimization (PSO) in engineering electromagnetics," *Conference Proceedings - ICECom 2003: 17th International Conference on Applied Electromagnetics and Communications*, no. October, pp. 1–5, 2003.

[34] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[35] R. D. Parsons and S. L. Canfield, "Developing genetic programming techniques for the design of compliant mechanisms," *Structural and Multidisciplinary Optimization*, vol. 24, no. 1, pp. 78–86, 2002.

[36] O. Sigmund, "On the Design of Compliant Mechanisms Using Topology Optimization," *Mechanics Based Design of Structures and Machines*, vol. 25, no. 4, pp. 493–524, 1997.

[37] T. E. Menghesha and K.-J. Lu, "Synthesis of Planar Compliant Mechanisms," *Proceedings of the ASME 2007 International Design Engineering Technical Conference*, pp. 1–9, 2007.

[38] R. T. Shield and W. Prager, "Optimal structural design for given deflection," *Zeitschrift für angewandte Mathematik und Physik ZAMP*, vol. 21, no. 4, pp. 513–523, jul 1970.

[39] J. a. Gallego and J. Herder, "Synthesis Methods in Compliant Mechanisms: An Overview," *Volume 7: 33rd Mechanisms and Robotics Conference, Parts A and B*, pp. 193–214, 2009.

[40] A. Saxena, "Synthesis of Compliant Mechanisms for Path Generation using Genetic Algorithm," *Journal of Mechanical Design*, vol. 127, no. May 2010, p. 745, 2005.

[41] J. a. Hetrick and S. Kota, "An Energy Formulation for Parametric Size and Shape Optimization of Compliant Mechanisms," *Journal of Mechanical Design*, vol. 121, no. 2, p. 229, 1999.

[42] G. K. Ananthasuresh, "How Far are Compliant Mechanisms from Rigid-body Mechanisms and Stiff Structures?" in *Advancements in Mechanics*, ser. Mechanisms and Machine Science, V. Kumar, J. Schmiedeler, S. V. Sreenivasan, and H.-J. Su, Eds. Heidelberg: Springer International Publishing, 2013, vol. 14, pp. 83–94.

[43] L. L. Howell, A. Midha, and T. W. Norton, "Evaluation of Equivalent Spring Stiffness for Use in a Pseudo-Rigid-Body Model of Large-Deflection Compliant Mechanisms," vol. 118, no. March, pp. 126–131, 1996.

[44] G. K. Matthew and D. Tesar, "Synthesis of Spring Parameters To Satisfy Specified Energy Levels in Planar Mechanisms." *J Eng Ind Trans ASME*, vol. 99 Ser B, no. 2, pp. 341–346, 1977.

[45] C. Huang and B. Roth, "Dimensional Synthesis of Closed-Loop Linkages to Match Force and Position Specifications," *Journal of Mechanical Design*, vol. 115, no. 2, p. 194, 1993.

[46] M. A. Pucheta and A. Cardona, "Design of bistable compliant mechanisms using precision-position and rigid-body replacement methods," *Mechanism and Machine Theory*, vol. 45, no. 2, pp. 304–326, 2009.

[47] E. Rosenberg, G. Radaelli, and J. Herder, "An energy approach to a 2DOF compliant parallel mechanism with self-guiding statically-balanced straight-line behavior," *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2, no. PARTS A AND B, pp. 455–464, 2010.

[48] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[49] DeepMind YouTube Channel, "Alphago zero: Discovering new knowledge," 2017.

[50] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

[51] H. van Hasselt and M. A. Wiering, "Reinforcement Learning in Continuous Action Spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, no. Adprl. IEEE, apr 2007, pp. 272–279.

[52] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[53] D. Silver, "Reinforcement learning lecture slides - university college london," Lecture slides, 2015, available on http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html , visited on July 12th 2017, University College London.

[54] P. Dollar, Z. Tu, and S. Belongie, "Supervised learning of edges and object boundaries," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 1964–1971.

[55] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 871–882, 2013.

[56] Q.-H. Ye, L.-X. Qin, M. Forgues, P. He, J. W. Kim, A. C. Peng, R. Simon, Y. Li, A. I. Robles, Y. Chen *et al.*, "Predicting hepatitis b virus–positive metastatic hepatocellular carcinomas using gene expression profiling and supervised machine learning," *Nature medicine*, vol. 9, no. 4, pp. 416–423, 2003.

[57] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[58] S. D. Connell and A. K. Jain, "Writer adaptation for online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 329–346, 2002.

[59] P. Arabie, "Cluster analysis in marketing research," *Advanced methods in marketing research*, pp. 160–189, 1994.

[60] R. S. Sutton, "Introduction: The challenge of reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 225–227, 1992.

[61] P. Lison, "An introduction to machine learning," Lecture slides, October 2012, available on http://folk.uio.no/plison/pdfs/talks/machinelearning.pdf , visited on April 5th 2017, from Language Technology Group (LTG) - Department of Informatics - University of Oslo.

[62] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge England, 1989.

[63] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, "The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 3238–3243.

[64] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[65] R. H. Crites and A. G. Barto, "Improving elevator performance using reinforcement learning," *Advances in neural information processing systems*, vol. 8, 1996.

[66] M. Mayourian and F. Freudenstein, "The Development of an Atlas of the Kinematic Structures of Mechanisms," *Journal of Mechanisms Transmissions and Automation in Design*, vol. 106, no. 4, pp. 458–461, 1984.

[67] R. S. Sutton and A. G. Barto, *Reinforcement Learn: An Introduction*, 2nd ed. The MIT Press, 2012.

[68] A. Ng, "Cs229 lecture notes: Reinforcement learning and control," 2016, available from http://cs229.stanford.edu/materials.html, accessed on 25-04-2017.

[69] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fund. Math*, vol. 3, no. 1, pp. 133–181, 1922.

[70] V. Heidrich-Meisner, M. Lauer, C. Igel, and M. A. Riedmiller, "Reinforcement learning in a nutshell." in *ESANN*. Citeseer, 2007, pp. 277–288.

[71] R. S. Sutton, "Learning to Predict by the Method of Temporal Differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[72] P. Dayan, "The convergence of TD(lambda) for general lambda," *Machine Learning*, vol. 8, no. 3-4, pp. 341–362, 1992.

[73] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[74] P. Burrow and S. M. Lucas, "Evolution versus temporal difference learning for learning to play Ms. Pac-Man," *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pp. 53–60, 2009.

[75] R. Hecht-Nielsen *et al.*, "Theory of the backpropagation neural network." *Neural Networks*, vol. 1, no. Supplement-1, pp. 445–448, 1988.

[76] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[77] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[78] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations*, San Diego, 2015.

[79] C. Painter-Wakefield and R. Parr, "L1 regularized linear temporal difference learning," *Technical report: Department of Computer Science, Duke University, Durham, NC, TR-2012–01*, 2012.

[80] J. Z. Kolter and A. Y. Ng, "Regularization and Feature Selection in Least-Squares Temporal Difference Learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 521—-528.

[81] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[82] K. M. Vermeer, "Literature Survey: Automated Design of Compliant Mechanisms," Literature Survey, Delft University of Technology, 2017.

[83] R. Avilés, A. Hernández, E. Amezua, and O. Altuzarra, "Kinematic analysis of linkages based in finite elements and the geometric stiffness matrix," *Mechanism and Machine Theory*, vol. 43, no. 8, pp. 964–983, 2008.

[84] J. L. Gross and J. Yellen, *Graph theory and its applications*. CRC press, 2005.

[85] D. G. Garson, "Interpreting neural network connection weights," *AI Expert*, vol. 6, no. 7, pp. 47–51, 1991.

[86] M. Gevrey, I. Dimopoulos, and S. Lek, "Review and comparison of methods to study the contribution of variables in artificial neural network models," *Ecological modelling*, vol. 160, no. 3, pp. 249–264, 2003.

[87] L. Milne, "Feature selection using neural networks with contribution measures," in *AI-Conference*. World Scientific Publishing, 1995, pp. 571–571.

[88] A. T. C. Goh, "Back-propagation neural networks for modeling complex systems," *Artificial Intelligence in Engineering*, vol. 9, no. 3, pp. 143–151, 1995.

[89] T. D. Gedeon, "Data mining of inputs: analysing magnitude and functional measures," *International Journal of Neural Systems*, vol. 8, no. 02, pp. 209–218, 1997.

[90] S. Lek, A. Belaud, P. Baran, I. Dimopoulos, and M. Delacoste, "Role of some environmental variables in trout abundance models using neural networks," *Aquatic Living Resources*, vol. 9, no. 1, pp. 23–29, 1996.

[91] G. Pahl and W. Beitz, *Engineering design: a systematic approach*. Springer Science & Business Media, 2013.

[92] L. L. Howell and a. Midha, "A Method for the Design of Compliant Mechanisms With Small-Length Flexural Pivots," *Journal of Mechanical Design*, vol. 116, no. 1, p. 280, 1994.

[93] L. L. Howell and A. Midha, "Parametric Deflection Approximations for End-Loaded, Large-Deflection Beams in Compliant Mechanisms," *Journal of Mechanical Design*, vol. 117, no. 1, p. 156, 1995.

[94] L. L. Howell, *Compliant mechanisms*. John Wiley & Sons, 2001.

[95] L. L. Howell, S. P. Magleby, and B. M. Olsen, *Handbook of Compliant Mechanisms*. Chichester: John Wiley & Sons Ltd., 2013.

[96] C. McLarnan, "Optimal synthesis of flexible link mechanisms with large static deflections," *Journal of Engineering for Industry*, p. 521, 1975.

[97] M. I. Frecker, K. M. Powell, and R. Haluck, "Design of a multifunctional compliant instrument for minimally invasive surgery." *Journal of biomechanical engineering*, vol. 127, no. November 2005, pp. 990–993, 2005.

[98] N. Ulrich, R. Paul, and R. Bajcsy, "A medium-complexity compliant end effector," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. IEEE, 1988, pp. 434–436.

[99] D. Shetty, "Compliant gripper for precision robotic assembly," in *Systems Engineering, 1990., IEEE International Conference on*. IEEE, 1990, pp. 335–338.

[100] S. Kota, J. Hetrick, Z. Li, and L. Saggere, "Tailoring unconventional actuators using compliant transmissions: design methods and applications," *IEEE/ASME Transactions on mechatronics*, vol. 4, no. 4, pp. 396–408, 1999.

[101] S. Canfield and M. Frecker, "Topology optimization of compliant mechanical amplifiers for piezoelectric actuators," *Structural and Multidisciplinary Optimization*, vol. 20, no. 4, pp. 269–279, 2000.

[102] S. Kota, J. Hetrick, Z. Li, S. Rodgers, and T. Krygowski, "Synthesizing high-performance compliant stroke amplification systems for mems," in *Micro Electro Mechanical Systems, 2000. MEMS 2000. The Thirteenth Annual International Conference on*. IEEE, 2000, pp. 164–169.

[103] F. Reuleuax, *The Kinematics of Machinery - Outline of a Theory of Machines*, A. Kennedy, Ed. New York: MacMillan, 1876.

[104] F. Freudenstein and L. Dobrjanskyj, *On a theory for the type synthesis of mechanisms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1966, pp. 420–428.

[105] M. J. Turner, "Stiffness and deflection analysis of complex structures," *journal of the Aeronautical Sciences*, 1956.

[106] Y. Liu, "Automated Type Synthesis of Planar Mechanisms Using Numerical Optimization with Genetic Algorithms," vol. 127, no. September 2005, 2004.

[107] P. Kuppens, "Automated mechanism design - literature study," Tech. Rep.

[108] F. Freudenstein and E. R. Maki, "The creation of mechanisms according to kinematic structure and function," *Environment and Planning B: Planning and Design*, vol. 6, no. 4, pp. 375–391, 1979.

[109] Y. M. Moon and S. Kota, "Automated synthesis of mechanisms using dual-vector algebra," *Mechanism and Machine Theory*, vol. 37, no. 2, pp. 143–166, 2002.

[110] M. I. Frecker, G. K. Ananthasuresh, S. Nishiwaki, N. Kikuchi, and S. Kota, "Topological Synthesis of Compliant Mechanisms Using Multi-Criteria Optimization," *Transactions of the ASME. Journal of Applied Mechanics*, vol. 119, no. June 1997, 1997.

[111] L. Cao, A. T. Dolovich, A. L. Schwab, J. L. Herder, and W. C. Zhang, "Toward a Unified Design Approach for Both Compliant Mechanisms and Rigid-Body Mechanisms: Module Optimization," *Journal of Mechanical Design*, vol. 137, no. 12, p. 122301, 2015.

[112] O. Sigmund, "Morphology-based black and white filters for topology optimization," *Structural and Multidisciplinary Optimization*, vol. 33, no. 4, pp. 401–424, 2007.

[113] L. Dobrjanskyj, "Application of graph theory to the structural classification of mechanisms." 1966.

[114] L. Dobrjanskyj and F. Freudenstein, "Some Applications of Graph Theory to the Structural Analysis of Mechanisms," *Journal of Engineering for Industry*, vol. 89, no. 1, p. 153, 1967.

[115] L.-W. Tsai, *Enumeration of Kinematic Structures Fundamentals of Environmental Discharge Modeling*, 2001.

[116] N. D. Mankame and G. K. Ananthasuresh, "Topology optimization for synthesis of contact-aided compliant mechanisms using regularized contact modeling," *Computers and Structures*, vol. 82, no. 15-16, pp. 1267–1290, 2004.

[117] R. Hooke and T. A. Jeeves, ""direct search"solution of numerical and statistical problems," *Journal of the ACM (JACM)*, vol. 8, no. 2, pp. 212–229, 1961.

[118] R. Battiti, "First-and second-order methods for learning: between steepest descent and newton's method," *Neural computation*, vol. 4, no. 2, pp. 141–166, 1992.

[119] H. A. Eschenauer and N. Olhoff, "Topology Optimization of Continuum Structures: A review*," *Applied Mechanics Reviews*, vol. 54, no. 4, pp. 331–390, 2001.

[120] P. J. Bentley and S. Kumar, "Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem," *Genetic and Evolutionary Computation Conference (GECCO '99)*, pp. 35–43, 1999.

[121] A. E. Eiben and C. A. Schippers, "On evolutionary exploration and exploitation," *Fundamenta Informaticae*, vol. 35, no. 1-4, pp. 35–50, 1998.

[122] R. K. Ursem, "Diversity-guided evolutionary algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2002, pp. 462–471.

[123] S. Ananthasuresh, G.K., Kota and Y. Gianchandani, "A Methodical Approach to the Design of Compliant Micromechanisms," *Workshop Technical Digest*, pp. 188–192, 1994.

[124] O. Sigmund, "Some inverse problems in topology design of materials and mechanisms," in *IUTAM symposium on optimization of mechanical systems*. Springer, 1996, pp. 277–284.

[125] U. D. Larsen, O. Sigmund, and S. Bouwstra, "Design and fabrication of compliant micromechanisms and structures with negative Poisson's ratio," *Journal of Microelectromechanical Systems*, vol. 6, no. 2, pp. 99–106, 1997.

[126] O. Sigmund and J. Petersson, "Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima," *Structural Optimization*, vol. 16, no. 1, pp. 68–75, 1998.

[127] A. Diaz and O. Sigmund, "Checkerboard patterns in layout optimization," *Structural optimization*, vol. 10, no. 1, pp. 40–45, 1995.

[128] C. S. Jog and R. B. Haber, "Stability of finite element models for distributed-parameter optimization and topology design," *Computer methods in applied mechanics and engineering*, vol. 130, no. 3-4, pp. 203–226, 1996.

[129] S. R. Deepak, M. Dinesh, D. K. Sahu, and G. K. Ananthasuresh, "A Comparative Study of the Formulations and Benchmark Problems for the Topology Optimization of Compliant Mechanisms," *Journal of Mechanisms and Robotics*, vol. 1, no. 1, p. 011003, 2009.

[130] D. Whitley, "An overview of evolutionary algorithms: practical issues and common pitfalls," *Information and software technology*, vol. 43, no. 14, pp. 817–831, 2001.

[131] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[132] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," 2007.

[133] A. Zhang and G. Chen, "A comprehensive elliptic integral solution to the large deflection problems of thin beams in compliant mechanisms," *Journal of Mechanisms and Robotics*, vol. 5, no. 2, p. 021006, 2013.

[134] N. Lobontiu, J. S. N. Paine, E. Garcia, and M. Goldfarb, "Corner-Filleted Flexure Hinges," *Journal of Mechanical Design*, vol. 123, no. 3, p. 346, 2001.

[135] Y. K. Yong and T. F. Lu, "Comparison of circular flexure hinge design equations and the derivation of empirical stiffness formulations," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, vol. 32, pp. 510–515, 2009.

[136] B. A. Salamon and A. Midha, "An introduction to mechanical advantage in compliant mechanisms," *Journal of Mechanical Design, Transactions of the ASME*, vol. 120, no. 2, pp. 311–315, 1998.

[137] D. S. S. Corp., "Understanding nonlinear analysis," White paper, Tech. Rep., 2006.

[138] P. Wriggers, *Nonlinear finite element methods*. Springer Science & Business Media, 2008.

[139] A. Saxena, "Topology Synthesis of Compliant Mechanisms for Nonlinear Force-Deflection and Curved Path Specifications," *Journal of Mechanical Design*, vol. 123, no. 1, pp. 33–42, 2001.

[140] J. Shigley and C. Mischke, *Mechanical Engineering Design*, ser. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1989.

[141] H. Su, "A pseudorigid-body 3r model for determining large deflection of cantilever beams subject to tip loads," *Journal of Mechanisms and Robotics*, vol. 1, no. 2, pp. 1–9, 2009.

[142] a. Saxena and S. N. Kramer, "A simple and accurate method for determining large deflections in compliant mechanisms subjected to end forces and moments," *Journal of Mechanical Design*, vol. 120, no. 3, pp. 392–400, 1998.

[143] Y.-Q. Yu, L. L. Howell, C. P. Lusk, Y. Yue, and M.-G. He, "Dynamic Modeling of Compliant Mechanisms Based on the Pseudo-Rigid-Body Model," *Journal of Mechanical Design*, vol. 127, no. 4, pp. 760–765, 2005.