Automated Mechanism Design

Introducing Reduced Operator-Space Evolution

J.A. Westra

Master of Science Thesis

Delft University of Technology

Biomechanical Engineering (BmechE)

Automated Mechanism Design

Introducing Reduced Operator-Space Evolution

Master of Science Thesis

J.A. Westra

May 24, 2021

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

Abstract

Previous research has shown automated robotic mechanism design to be both deceptive (prone to local minima) and rife with linkage problems (having highly interdependent parameters). This results in a barrier to optimization that is unable to be breached by simply applying more iterations and computational power.

The research also indicates that a graph structure model of the robot in combination with an evolutionary algorithm yields useful robotic mechanisms for a limited set of simple problems. This thesis expands on this pre-existing representation by introducing an indirect model that can be used to include both controllers, motors and other new elements in the representation.

Besides this extension of the mechanism model, a framework for the automated design optimization task itself is introduced. This thesis shows an equivalence between an operator based representation of the mechanisms and the graph based representation. These operators represent modifications on the mechanism structure and/or parameters. By recognizing the operators as paths in this model a graph of the search space itself can be constructed. In this graph the vertices are mechanisms and the edges are operators.

Using the the operator-mechanism equivalence it is shown that designing an optimization algorithm is equivalent to (1) choosing how vertices in the space are grouped together. (2) choosing how the vertices of this search space are connected beforehand by either implicitly or explicitly picking operators and projecting onto their corresponding domain. (3) picking which of the connected paths to traverse based on accumulated information at runtime.

This represents a framework that allows the accumulation of knowledge about optimization algorithms acting within it by defining a set of meta-heuristics. With these it is possible to make informed choices to build better optimization algorithms.

To show the effectiveness of the framework a novel quality diversity algorithm is developed, Reduced Operator-Space Evolution (ROSE), which uses the insights mentioned above to generate a large diversity of well performing mechanisms simultaneously on a representative pick-and-place task. This confirms the theoretical results about the effect of the operator-mechanism equivalence and locality properties. A new step forward to breaching the barrier to optimization.

Alongside this thesis a performant simulation and analysis Python library for mechanisms was developed called PyMechs. The library visualizes the framework and handles mechanism simulation and evaluation, as well as implements ROSE.

It is available at https://github.com/kooswestra/pymechs.

Table of Contents

Preface

This document is part of the graduation project for the master BioMechanical Design (BMD) at the Delft University of Technology (TU Delft) in The Netherlands. The choice to do a project on evolutionary robotics requires some explanation.

I have always been interested in robotics. Having been inspired, like so many others, by the possibilities presented in literature, movies and art. Robotics lies on the crossroads between computers and mechanics, artificial intelligence and motion control. Making it a true multidisciplinary engineering field. It is the interface between computer software and the real, physical world.

During my studies I encountered the work of Michael Schmidt and Hod Lipson where through the application of genetic algorithms they were able to automatically discover and describe natural laws. The fact that it was possible to have computers generate such creative results inspired me, leading me down the rabbithole of evolutionary computation and computational design.

I knew then that I wanted to do my thesis on an application of artificial intelligence to physical systems. Through discussions with friends I came across dr. ir. W.J. Wolfslag, who suggested a continuation of the work of ir. P.R. Kuppens on evolutionary robot design. I picked up this chance to combine my two core interests immediately. The result is the document you see before you now.

J.A. Westra
Delft, University of Technology,
May 24, 2021

Chapter 1

Introduction

1-1 Embodiment

Robot design is a complex task consisting of many different subproblems and spanning multiple engineering fields. To make this complex, multi-disciplinary task manageable to engineers the problem is split into different tasks and each element considered separately, often even by different teams. This results in robots where each subsystem, such as the mechanical structure, control loop, electronics, motors, high-level planning and more is in principle designed and acting independently.

Yet in the natural world which inspires robotics there is no such separation; each system evolved to act upon others in advantageous ways. This synergy allows a large reduction in both energy usage and system complexity; the mechanical and physical structures allow pre-computation of control and sensory information, natural stability properties reduce control effort, mechanical leverage effects reduce the required muscle force, and many more such advantageous relations. Such synergy is known as *embodiment* and is ubiquitous in the natural world [1–3]. A schematic of embodiment is shown in Figure 1.1. Embodiment effectively reduces muscular and cognitive load.

A good example is the data structuring morphology in the eye of a fly. The front of the eye has more facets than the side [4]. This irregularity naturally compensates for the effect of motion parallax: the effect that objects to the front appear to move slower than objects on the side. The placement of more facets to the front results in a similar overall signal amplitude for the same velocity as on the side: the natural layout of the eye provides the compensation [5].

The properties of embodiment would be a great benefit to the artificial world of robotics as well. With possible advantages such as lower energy usage, less material cost, increased stability and lower motor torques it comes as no surprise there is active research into discovering and consequently leveraging such synergy between systems. For example by exploiting natural dynamics to achieve lower energy usage of a robot arm [6], using natural kinematic properties to create walking robots [7,8] and many other applications such as e.g. [3,9–12].

Contrary to the usual design methods designing embodiment requires an "all at once approach" of the relevant systems, which is more costly in terms of engineering time and resources

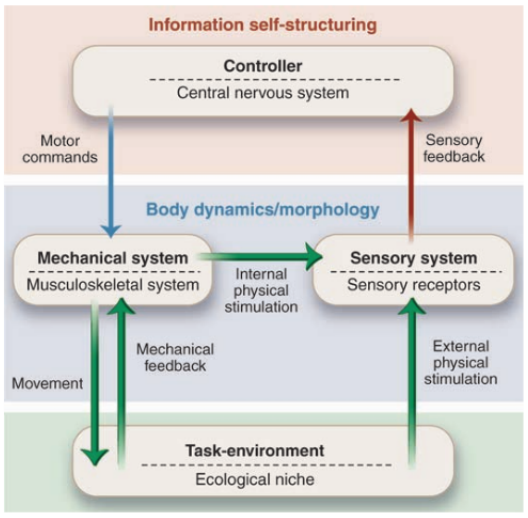

Figure 1.1: A schematic of embodiment in a biological system. The mechanical system acts as a pre-processor of sensory information through internal physical stimulation. Figure from [2].

than traditional robot design. Any synergy put into the design relies to a large part on the engineer's intuition and deeper understanding of all the systems involved. It quickly becomes overwhelmingly complex and time consuming as every system involved needs to be considered and understood simultaneously. This limits the current level of embodiment that can be obtained.

To create robots that leverage embodiment reliably, a process to discover and apply these synergies is required. If we can offload the need for a creative "eureka moment" the cognitive load on engineers is reduced, the complexity of the problem becomes manageable again.

We can achieve this by providing automated design tools. Using the continuously increasing processor power to automatically discover novel synergy, significant improvements to robotic systems could be achieved while simultaneously reducing the engineering effort required to do so. Great success has been had in various fronts using this approach, including deep learning, circuit design, controller synthesis, as well as many other fields [13].

Automated design routines often find designs that are nowhere near what a human designer would think of, yet are far more effective. A good example of this is the cantilever truss shown in Figure 1.2. An irregular design found by an evolutionary algorithm based automated design routine managed to be two hundred times more effective in reducing vibrations than a traditional design [14].

Yet the performance of robot automated robot design algorithms has thus far been relatively stagnant. Despite a significant increase of available processing power which sparked great success in for example deep learning [15], automated robot design has yet to make great strides; it is clear that significant improvements of the performance of existing algorithms can't be achieved by simply throwing more processing power at the problem due to the poorly behaved, infinite and discontinuous search space [16]. This suggests a different approach is necessary, which invites a more detailed look at the automated design problem.

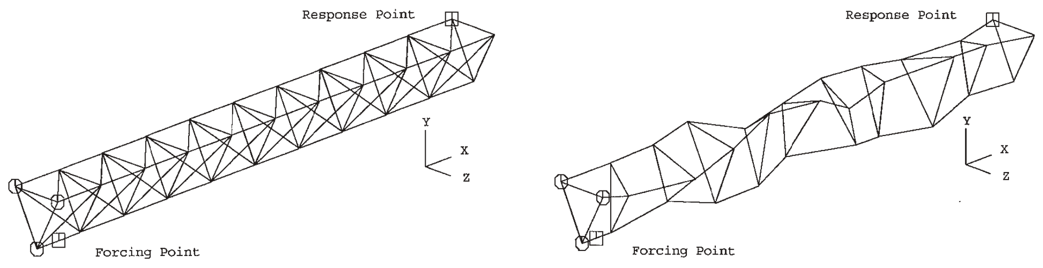

Figure 1.2: A cantilever truss for use in space, the goal was to design a shape that would minimize vibration transfer, while maintaining integrity and strength. On the left is a traditional design, on the right is a solution found by an automated design algorithm. Its highly irregular shape is two hundred times more effective at reducing vibrations than the traditional one. Image from [14].

1-2 Computational design

A design task can be thought of as a search through the space of all possible designs for a design that optimally satisfies design requirements such as effectiveness, cost and aesthetics. Engineers are implicitly doing a form optimization when they are creating and refining a design, even if they haven't made the actual parameters, goals and constraints explicit. By reasoning about function, calculating fitting parameter values and similar tasks they are exploring the space of possible designs for a best fit. It follows that, given a design task with sufficiently quantifiable system parameters and requirements, a design task can be modeled as an optimization problem [17].

The general design space is infinite, reflecting that any design is possible. To make the problems tractable we must bound the problem by parametrization, which imposes a modelling structure. A successful parametrization is a model that breaks the design problem down to fundamental elements, while ignoring superfluous details. Clearly, designing a model and parametrization that is aligned with the problem is important. Extending the model to be more general is likely to yield a better optimum, but that comes with the trade-off that the resulting optimization problem becomes harder to solve [18].

The ideal models for the system represent the Pareto front of complexity and descriptiveness for the problem they are applied to, a front of optimal trade-off between the two. Good models for the problem represent approximations of these unknown “truly optimal” ideal models. Figure 1.3 visualizes what this means in a concrete sense: approximating the circles using a model that only admits squares uses a lot more parameters than strictly necessary while introducing model error. This results in high complexity with comparatively low descriptiveness. If a circle model is adopted an obviously much less extensive description is required that simultaneously achieves better accuracy.

In this trivial case the truly optimal model is the description of a circle but in general such models not so easy to find. Note that when using a circle description squares can no longer be well described. Trying to account for this by being more general and modeling both circles and squares simultaneously results in an optimization problem that is harder to solve instead;

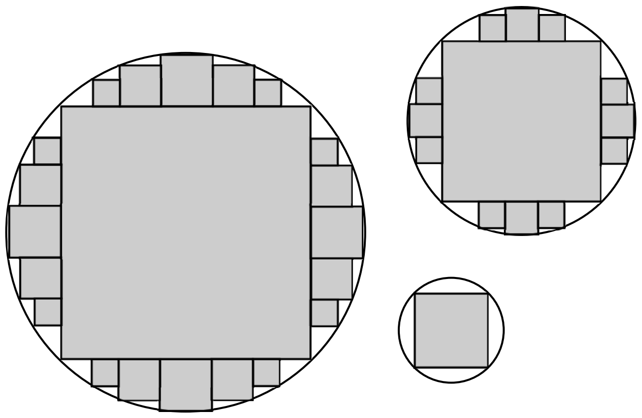

Figure 1.3: Using squares as a model parametrization for circular shapes is poorly aligned: infinitely many squares are required to fill a single circle. When using a circle model instead this system goes from infinite parameters to only three per circle: (radius r, x and y position of the center).

the no free lunch theorem in action [18].

We can identify three necessary components to cast the design problem as an optimization problem:

Model: A model parametrization of the candidate designs is required. The model needs to be as aligned with the problem as possible to reduce the amount of free parameters and thus complexity of the optimization problem while simultaneously capturing as much of the design as possible. A good representation shapes the search space such that a good solution becomes easier to find. The model ideally encapsulates tangible design choices.

Objective: The design requirements need to be quantified in an objective function, a mathematical measure which directly relates the degree of performance of a design with respect to the requirements to a numerical (fitness) score.

Solver: A solver is required to find the optimal design. The solver is a routine that explores the design space admitted by the model parametrization for an optimal solution on the objective. The type of solver that is effective is highly dependent on the relation between the model and the objective.

The combination of the representation and objective function fundamentally shape the search space of the problem. The shape of the search space in turn has a very large role in the quality of the designs that can be automatically obtained. In general there is no explicit relation known between the parameters and the performance on the objective function, requiring a simulation, and thus an optimization method that can deal with the fact that no explicit relation and gradient is known. The usual result is a *black box* optimization problem and compatible solver[1].

[1] Although an explicit gradient and solver can be available in some cases, notably when designing neural networks backpropagation is one such solution.

Evolutionary algorithms are well suited to solving the kinds of optimization problems that arise when applying automated design. Usually the model used for the design has a combination of discrete (structure) and continuous (parameters) elements: it is a mixed-integer optimization problem. The mechanisms in particular are part of that group, as the structure of the mechanism is discrete (e.g. the number of links and the way they are connected) while the parameters are numeric (e.g. the positions, masses, spring constants). The objective function for complex problems also tends to be multi-modal: it has lots of local minima. The stochastic element of evolutionary algorithms allow them to search these local minima to find a better solution, while the gradient-free approach of evolutionary algorithms allow them to deal with the discrete and continuous parameters simultaneously.

1-3 Linkage, Deception & Redundancy

It seems at first glance that the problem is easily solved using the automated design formula: develop a model for robots, model the objective and solve the resulting optimization problem using an evolutionary algorithm. Yet this answer and many variations on it has been tried before [19–26] and in practice the results have been limited, especially when active control comes into the mix [27]. From literature we can identify three fundamental problems that automated robot and mechanism design systems run into:

Linkage Evolutionary algorithms naturally require that sub-solutions can be found that represent parts of the final solution, i.e. each building block has to have a consistent impact on the objective by itself [28]. However, when studying the mechanism representation structure it is clear that combining building blocks generates an entirely new mechanism with completely different behavior, and thus fitness, than the parents as shown in Figure 1.5. When applying a controller this problem becomes even more pronounced: as shown in Figure 1.4 the controller is only able to act through the mechanism. This indirection creates a *linkage* between the controller and mechanism. As a result the mechanism elements can fundamentally not be considered independently during optimization [27, 29].

Deception The search space has many local minima: it is *deceptive*, which means that it is not possible to reach the objective by making incremental improvements with respect to the objective [30].

Redundancy The search space has many mechanisms that have a different representation yet are functionally identical [19], resulting in large amounts of redundancy and corresponding wasted evaluations leading to large computational inefficiency in the exploration of the design space by the solver.

The linkage and deception problems have proven to be highly resistant to simply throwing more processing power at the problem [16]. Yet solving them is paramount to generate better automated designs and eventually reach embodiment for robots.

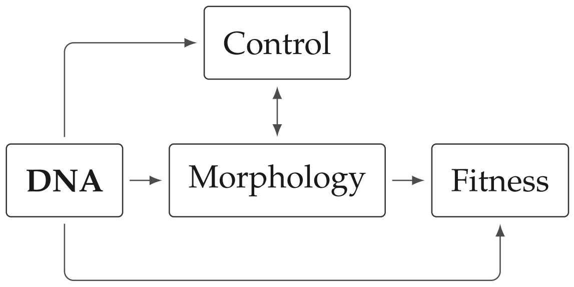

Figure 1.4: The natural structure of a robot allows the controller to only influence the fitness through the mechanism structure.Note that the DNA can also have a direct influence on the fitness through a complexity and/or diversity score.

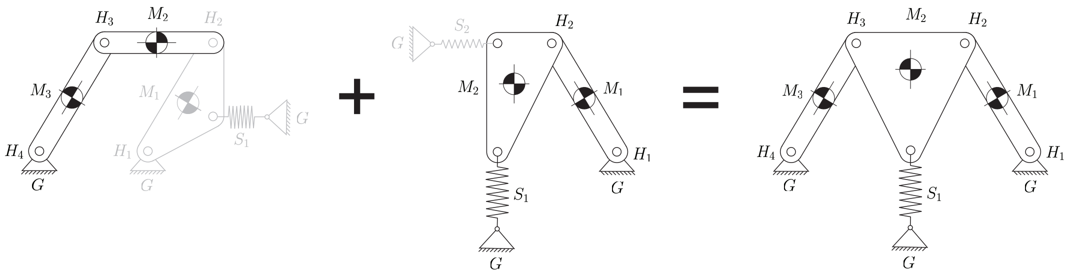

Figure 1.5: Crossover creates mechanisms that are significantly different in behavior from either of their parents. Image from [19].

1-3-1 Mapping issues to computational design functions

As automated mechanism design is a complex problem it is not immediately clear how these three issues arise and relate to the different elements of automated mechanism design. To this end we can break automated mechanism down into manageable subfunctions and pinpoint exactly where, what problem occurs and why. We can map the three issues described by previous research to different steps in this breakdown. This is illustrated in Figure 1.6.

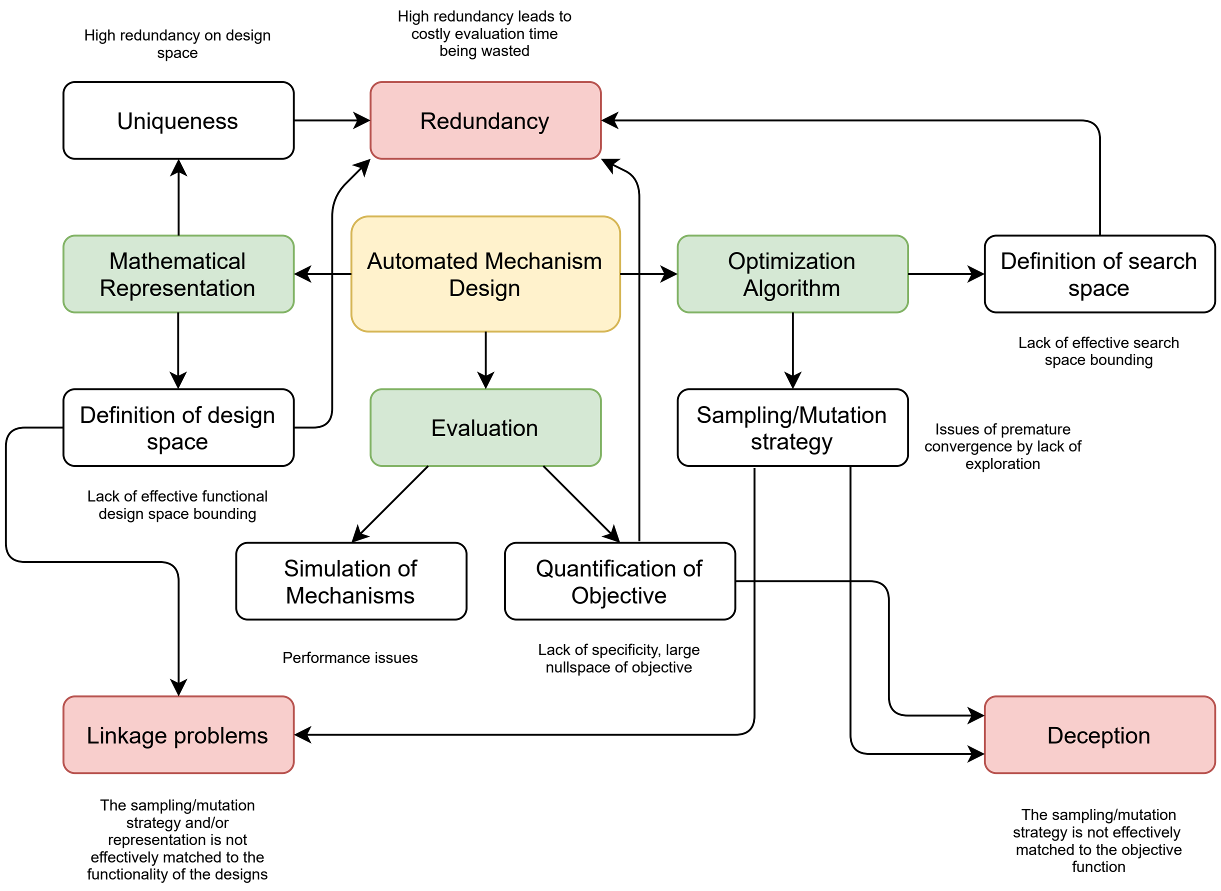
High redundancy on design space
High redundancy leads to costly evaluation time being wasted
Uniqueness
Redundancy
Mathematical Representation
Automated Mechanism Design
Optimization Algorithm
Definition of search space
Lack of effective search space bounding
Definition of design space
Evaluation
Sampling/Mutation strategy
Issues of premature convergence by lack of exploration
Lack of effective functional design space bounding
Simulation of Mechanisms
Quantification of Objective
Performance issues
Lack of specificity, large nullspace of objective
Linkage problems
Deception
The sampling/mutation strategy and/or representation is not effectively matched to the functionality of the designs
The sampling/mutation strategy is not effectively matched to the objective function

Figure 1.6: The breakdown of automated mechanism design as based on the computational design method. The breakdown indicates the areas of issue dealt with by this thesis as the white blocks. It is also indicated how these areas relate to the three known issues. Note that for clarity other subfunctions that are not directly treated are not shown.

As seen in the figure the three issues relate to different elements of the automated mechanism design recipe:

Linkage Linkage problems are coupled to the definition of the design space and sampling and/or mutation strategy of the optimization algorithm. The representation is not matched to the actual functionality of the design that is represented which means that the individual building blocks of a design are highly coupled with respect to the overall functionality of the design. This requires an investigation into the mutation strategy of the evolutionary algorithm and the design space of the representation. This will be treated in Chapters 3 and 4 leading to the introduction of operator-representation duality and the ROSE algorithm which decouples the definition of the design space from the sampling/mutation strategy.

Deception Deception is coupled to the sampling and/or mutation strategy and the quantification of the objective, if the sampling strategy is ineffective for the shape of the objective function the optimization strategy will be ineffective. For example having a gradient descent strategy on an objective function with many local minima. This requires an investigation into the quantification of the objective function and again the mutation strategy. This leads to the definition of the parameter mapping function, parameter invariants and parameter bounding in Chapter 3, and the injection of novelty and the concept of operator-space projection in the ROSE algorithm in Chapter 4.

Redundancy Redundancy is coupled to the uniqueness of the representation, the design space, the search space and the objective. An isomorphism strategy from prior art is used to reduce redundancy as will be explained in Chapter 2, parameter invariants are also introduced in Chapter 3 to reduce redundancy. Finally, in order to deal with the significant redundancy still present a high-performance simulator is developed in Chapter 5, which attacks the redundancy problem from the angle of reducing evaluation time, thereby making wasted evaluations significantly less costly in real time.

1-4 Representative practical problem

Although these problems are not limited to the domain of automated robot design, we select it as a representative task. The methodology of Reduced Operator-Space Evolution (ROSE) developed in this thesis is not limited to automated robot design specifically and can be generalized to other domains such as machine learning, electrical circuits, etc.

But to generate results and solve them we need a representative automated mechanism design problem that specifically has them. To that end this thesis will focus on 2D mechanisms, applied to a pick-and-place task. The developed methodology can in principle be extended to three dimensions, but staying in two allows significantly reduced computation time and complexity while the discovered principles will still hold for both 2D and 3D.

Pick-and-place tasks are a commonplace task in robotics, extending from warehouses management to industrial robots in factories, with for example Amazon having over 200,000 robots working in its warehouses [31]. The strict constraints around the task definition, combined with their practical usefulness, make them an excellent choice for more theoretical research purposes [32]. They provide easily identifiable markers of performance and success, and task complexity can be scaled relatively easily.

Because the pick-and-place robots found by the algorithm will be actuated using motors with control both the linkage and deception problems appear in this otherwise straightforward task. It is of particular interest if the algorithm will find mechanisms and controllers that effectively reduce energy usage, perhaps even discovering embodiment.

1-5 Research goals

The goal is to design an evolutionary algorithm that is able to quickly generate concept designs for actuated mechanisms. In this case the specific test cases are pick-and-place tasks.

Research on the evolution of mechanisms and mechanical systems in combination with active control indicates that there is a boundary of complexity that algorithms are unable to break, no matter the amount of computational power applied to solve the problem [16, 27, 33]. A survey of existing literature, including [19–21, 23, 24] and others[2], indicates this boundary lies below the complexity of the pick and place task. In order to achieve our goal we have to extend the functionality of, and improve on the existing evolutionary algorithms.

The complexity boundary suggests there is a structural problem within the optimization landscape; the search space itself exhibits a very high level of irregularity. This sabotages the effectiveness of evolutionary algorithms as the underlying assumption that successful building blocks combine to a successful whole fails; the search space has significant linkage problems [30]. There is some indication that in that case abandoning the classic objective function itself and instead using Novelty Search will lead to better results [35, 36].

However, in optimization there is no such thing as a free lunch [18], improving the algorithm always requires the injection of *domain knowledge* into the workings of the algorithm. In the case of Novelty Search this is a distance-, or novelty measure for mechanisms. The definition of this measure has significant impact on the behavior of the algorithm, and consequently has to be well understood to be effectively applied.

Note that the danger in adding knowledge is that while it has to be specific enough to provide benefits to the optimization algorithm, it simultaneously has to be general enough to not narrow the scope of the search such that it excludes good or inventive solutions. Yet so far there exists no way to quantify what is “good” knowledge and what is “bad” knowledge to add to the automated mechanism system.

The first subgoal of this thesis is to generalize the graph representation of mechanisms to allow any type of element, including general active controlled motors. [Chapter 3]

The second subgoal is to analyze the mechanism design space and the workings of the automated design algorithm from a more theoretical point of view, in order to generate a framework in which to study the behavior of optimization algorithms in automated mechanism design. It is shown that the mechanisms can be represented by a series of operators in parallel to the graph representation. By splitting the mechanism representation into a structural graph and parameter function it is possible to define the concept of a mechanism space with distance measure. A novel quality diversity evolutionary algorithm based on abstract operators called Reduced Operator-Space Evolution is derived from the structure of this mechanism space. [Chapter 4].

The third subgoal is to create the necessary software to computationally explore the mechanism space, by simulating, evaluating and analyzing both the mechanisms and search space based on the representation of both presented in the preceding parts. The result is the open-source Python library PyMechs, implemented in C++. [Chapter 5]

[2]See also the literature survey accompanying this thesis [34]

Chapter 2

Prior Art

This chapter first introduces the necessary theoretical background on computational design, evolutionary algorithms, linkage problems and deception in more detail in Sections 2-1 and 2-2 for those unfamiliar with them. Section 2-3 discusses the prior art representation used as a base design.

2-1 The computational design equation

The automated design optimization problem described in Section 1-2 can be mathematically defined as finding the optimal parameters $\mathbf{p}$ of the chosen model representation $\hat{y}(\mathbf{p}, \mathbf{u})$, given inputs $\mathbf{u}$, such that the error $f(y, \hat{y}, \mathbf{p})$ between the specified desired behavior $y(\mathbf{u})$ and model behavior $\hat{y}(\mathbf{p}, \mathbf{u})$ is minimized. Additional penalties can be applied based on the parameter values $\mathbf{p}$, such as a complexity penalty so that generally the function f is itself also directly dependent on $\mathbf{p}$. Additionally, constraints $g(\mathbf{p})$ may be present on the parameters of the model. This results in the following optimization problem for the optimal design parameters:

$$\mathbf{p} = \arg\min_{\mathbf{p}} f(y, \hat{y}, \mathbf{p}) \quad \text{subject to} \quad g(\mathbf{p}) \leq 0 \tag{2.1}$$

Traditionally this would be solved with gradient descent, as is often done in Machine Learning problems [15]. However to do this, the relation between the parameters of the model $\mathbf{p}$ and the model behavior $\hat{y}$ has to be known so that $\hat{y}$ can be substituted in the objective function f. This makes f an explicit function of $\mathbf{p}$ so that the gradient of the objective function with respect to the parameters $\frac{\partial f}{\partial \mathbf{p}}$ can be obtained.

In automated design there is in most cases no explicit relation known and model behavior can only be determined through simulation i.e. sampling. This complicates the problem considerably, especially if the sampling computation cost is high. The resulting black box optimization problem is usually solved with evolutionary algorithms [17].

2-2 Evolutionary Algorithms

2-2-1 The canonical evolutionary algorithm

Evolutionary algorithms [13] are a class of optimization methods that mimic natural evolution as described by Darwin [37] in order to find an optimal solution. They are particularly useful when there is no explicit relation known between the objective function and the optimization parameters, as they use sampling as the evaluation method for individuals. It follows that no explicit gradient is required. Additionally they are also very effective when the search domain is poorly behaved, containing many local optima and discontinuities; this is because the stochastic nature of the variational operators and the diversity maintained in the population lets the algorithm explore the search space in more directions than just the one of immediately increasing fitness.

Since both these characteristics are generally the case for the modeling problem presented by automated design as equation 2.1, evolutionary algorithms present a natural and popular solver choice [13].

Evolutionary algorithms work by initially generating a population of possible candidate solutions. These are evaluated using a fitness function. This fitness function is usually the objective function of the optimization problem, but can be augmented in order to boost the performance of the algorithm as is done in e.g. Quality Diversity algorithms [35, 38, 39]. Individuals are selected from the population to reproduce based on their fitness value, with the goal of propagating successful building blocks. During this reproduction step variational operators are applied, such as mutation and crossover. This injects new genetic material into the population and mixes up existing genes which allows the algorithm to effectively explore the search space.

- Mutation is the analog of biological mutations, by inserting a random (small) change in the representation of the individual new information is injected into the population.
- Crossover is the analog of natural sexual reproduction, where the representation (DNA) of 2 individuals is mixed together into a new child individual. Mixing the elements of both parents together can combine successful building blocks.

Poorly performing offspring are discarded, successful offspring are promoted to the population, possibly replacing a worse performing existing individual. These steps can be applied to the entire population at a time, creating a fresh generation every iteration, but they can also be applied per individual or any combination of individuals. A schematic of the canonical evolutionary algorithm is shown in Figure 2.1. The result is that the population as a whole moves in the direction of better fitness.

2-2-2 Schema theorem

Perhaps the most fundamental theoretical result that explains why evolutionary algorithms work is Holland's schema theorem [28]. It shows that genetic algorithms accumulate successful building blocks called schemata over the generations[1]. These schemata are fractions of the genotype, that describe some part of the system under consideration.

[1]To stay concise the exact mathematical details and proofs are omitted here. They are explained thoroughly however in many different papers and books such as [25, 28, 40]

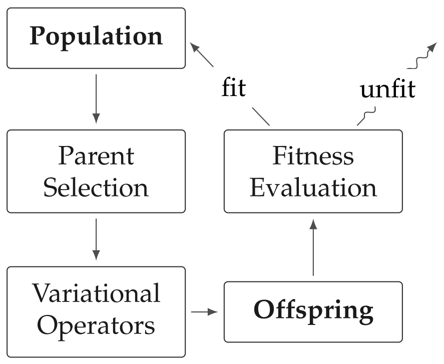

Figure 2.1: The canonical evolutionary algorithm. Individuals are selected as parents from the population and transformed by a variational operator (analogous to mutations and sexual reproduction) to generate offspring. This offspring is evaluated. Fit offspring gets promoted back in the population, unfit offspring gets discarded.

As an example, let's take a simple genetic encoding consisting of just 4 bits describing some sort of system. Schemata in this 4 bit world for example are (where the * denotes a wildcard, it could either be a 1 or 0):

$$\begin{bmatrix}1 & * & * & *\end{bmatrix} \qquad \begin{bmatrix}* & 0 & 1 & *\end{bmatrix} \tag{2.2}$$

Where individuals are for example:

$$\begin{bmatrix}\mathbf{1} & 0 & 0 & 1\end{bmatrix} \qquad \begin{bmatrix}\mathbf{1} & \mathbf{0} & \mathbf{1} & 0\end{bmatrix} \qquad \begin{bmatrix}0 & 1 & 0 & 1\end{bmatrix} \tag{2.3}$$

In this case individual 1 contains schemata 1, individual 2 contains both and individual 3 contains neither.

The schema theorem proves mathematically for a simple genetic algorithm that schemata which have higher individual fitness, and thus provide some evolutionary benefit to individuals, tend to accumulate in the population in an evolutionary algorithm over the generations. The schemata also increase in complexity, increasingly focusing the search in the hyperplanes in the search space spanned by these schemata [28].

As more of these successful building blocks accumulate the algorithm converges to areas of higher overall fitness for all the members in the population, as these members are increasingly made up of successful schemata. This explains how evolutionary algorithms can approximate optimal solutions.

It follows that the success of objective-based evolutionary algorithms lies in the assumption that *the building blocks of a successful solution, are by themselves partially successful at solving the problem.* As the building blocks need to be beneficial on the objective to be selected for by the algorithm. If this is not the case, the search is focused on the wrong hyperplanes of the search space and, for complex problems, is likely not to find the solution at all. This is known as deception [30, 41].

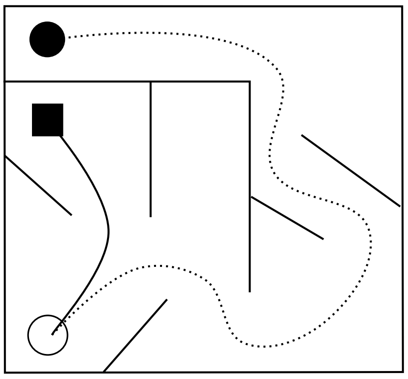

Figure 2.2: A maze domain problem to illustrate deception. Using naive fitness maximization will lead to the algorithm getting stuck in the position denoted by the square

2-2-3 Deception

In order to illustrate deception we can take the simple toy optimization problem where the representation is given by a bitstring of size 4. Let's assume the globally optimal solution is given by:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.4}$$

It is a deceptive problem when the schemata of the optimal solution $\begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$ all result in individually low fitness. While the schemata of a globally poor solution all perform relatively well on the problem.

We can make the toy problem intentionally deceptive by shaping the fitness function as an example. If we assign the bitstring $\begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix}$ a poor fitness score and give otherwise completely wrong solutions $\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ a good score we generate deception. This occurs because according to the schema theorem the schema $\begin{bmatrix} 1 & 0 & 0 & * \end{bmatrix}$ is propagated in the population rather than $\begin{bmatrix} 0 & 1 & 1 & * \end{bmatrix}$. Yet where $\begin{bmatrix} 0 & 1 & 1 & * \end{bmatrix}$ contains three out of four elements of the optimal solution, $\begin{bmatrix} 1 & 0 & 0 & * \end{bmatrix}$ contains none. Clearly the search will in that case be focused on the wrong area of the search space, because a single mutation or crossover event can shape the solution from the worst to the best.

The stochastic nature of evolutionary algorithms generally allow them to deal with these cases to a certain level. In the toy problem example the odds of finding the optimum just by random generation is already 1/16. Yet, in the real world problem of mechanism design the odds of randomly stumbling on the solution in a usual given population size becomes astronomically small. When the complexity of the required combined successful blocks is too great to discover randomly, an evolutionary algorithm will still struggle to overcome it.

An example of deception occurring in an evolutionary algorithm is illustrated in Figure 2.2. In this problem controllers are evolved to guide the robot through a maze from the open circle to the closed circle. Using the naive optimization technique of minimizing the distance to the objective the algorithm will get stuck in the dead-end of the maze denoted by the square instead of finding the true (dotted) path. The problem is deceptive with respect to the objective function.

2-2-4 linkage problems

Linkage problems occur when successful building blocks are disrupted by the variational operators. This is more likely to occur in complex problems as they tend to have more complex building blocks. It is a result of the success of one element of the representation being tightly coupled to the success of another element.

While deception is a property of the objective, linkage problems are a property of the used variational operators and representation. This means that with proper choice of representation, linkage problems can be avoided. In practice this means identifying any linkages a priori, which generally is a hard problem in it's own right [42]. If the problem is multi-objective this becomes even more complicated, as linkages might occur with respect to one of the objectives but not the others and vice versa [43].

Linkage problems occur in practice when evolving the morphology of mechanisms for a specific desired behavior, removing even one connection in a four bar linkage for example will completely disrupt the overall behavior of the entire system. The system as a whole produces resultant behavior which can not be easily decomposed into fundamental individual building blocks.

2-2-5 Novelty Search

Novelty search is a variant of evolutionary algorithm that, rather than maximizing the performance with regard to a fixed objective, is driven by the search for behavioral novelty [36, 41] i.e. it is driven by creating an as large variety of candidate solutions as possible. The greater diversity has been shown to reduce the threat of deception [36]. The behavior is compared to that of the current population and an archive of past behavior. A higher likelihood of reproducing is obtained by doing something new, rather than maximizing a criterion. This aggressively encourages exploration of the behavior space. In fact, the algorithm can be considered as exploration only, albeit in an intelligent way.

Novelty search works by taking an existing evolutionary algorithm and replacing the objective function by a *novelty metric.* Defining this novelty metric requires careful planning, as it determines the behavior space through which the algorithm will search. Defining which behaviors are considered novel and distinct, and how to explicitly quantify this, becomes one of the key design problems. It is known as *behavior characterization* [44].

However, there are some drawbacks. An archive of past individuals needs to be maintained and compared to, which can grow to be very large. Additionally, if the search space is *unbounded* there is always new novelty to be found by going somewhere meaningless. This means boundaries (such as the outer edges of a maze in a pathfinding problem) have to be defined a priori.

Novelty search has been applied to generate virtual creatures [26]. The algorithm Lehman et. al. presented was able to create a variety of functional and distinct walking creatures in a single run.

2-2-6 Quality Diversity

Quality diversity algorithms [35] aim to generate a repertoire of high quality, diverse solutions to a problem in a single run. A quality diversity algorithm takes the diversity generating power of novelty search, and combines it with a direct objective based approach to give more direction to the search. They represent some of the newest developments in the field of evolutionary computation [29,38,45,46]. It is interesting to note that they have shown to be more resistant to deception and linkage problems in the search space while maintaining performance on the objective, at the cost of taking more function evaluations.

Instead of looking for the best overall individual in the search space, quality diversity algorithms split the search space into different regions and look for the best individuals for each separate region. The result is that, rather than returning a single most fit solution, the algorithm generates a *QD collection*; a collection of the best performers across the entire space divided along a measure known as the characterization.

It follows that in order to use a quality diversity algorithm, a method to divide the search space is required. Generally a behavior characterization (A metric for the difference between two behaviors $\hat{y}(\mathbf{p},\mathbf{u})$) is used [44] but other divisions are possible such as along parameters [38, 47]. The performance of the algorithm is significantly influenced by this choice of characterization: clearly it is important to align this characterization with the design problem similar to the way the model should be aligned with the problem. The division of the search space has to make sense.

A great benefit over classic optimization algorithms is that the result of a quality diversity run is a set of independent varying designs, rather than a single optimal one. A designer could pick any one of the solutions that satisfies other objectives that are not represented in the objective function. This is a great benefit to automated design problems as often not all requirements can be encapsulated in the objective function, such as for example aesthetics.

2-3 Encoding Mechanisms

2-3-1 Modeling mechanisms

A model representation and parametrization of mechanisms is fundamental to apply automated design to the design of mechanisms. There have been quite a few attempts over the years at an unambiguous mathematical description of mechanisms, such as Denavit-Hartenberg parameters [48], screw-theory based descriptions, graph-theory based representations and more. But these representations were developed to provide an easy way to model the kinematics of mechanisms, not necessarily to be conducive for automated design.

As explained in Section 2-2 the representation should ideally consist of small but effective and general building blocks called schemata to be processed by the evolutionary algorithm, yet these building blocks have proven hard to identify exactly, as indicated by the numerous different competing representations and approaches [16,21,23,27].

However, success has been achieved using a graph based representation for mechanisms for automated design using an evolutionary algorithm as solver [19]. It has as particular benefits that identical mechanism structures can be easily identified and many mechanism properties can be related to useful fundamental graph properties [25,49].

2-3-2 Graph representation of mechanisms

In graph-based mechanism modeling the mechanism structure is represented by the ordered pair of an undirected, rooted, labeled graph G and corresponding parameter map P [19,20,25,50].

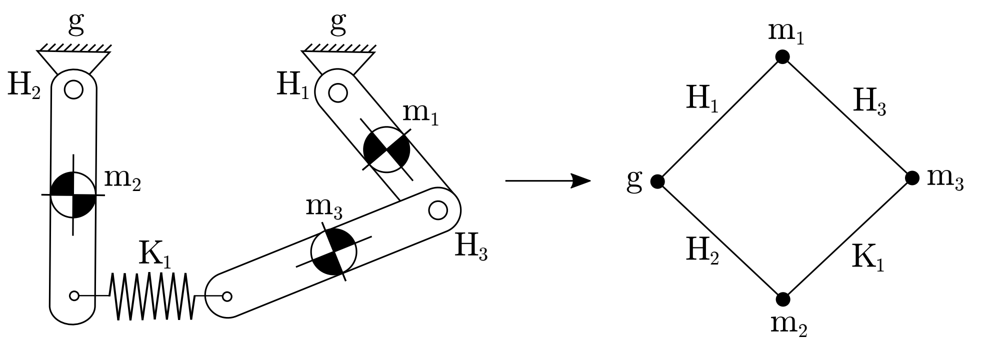

Figure 2.3: A mechanism encoding example, the mechanism on the left can be encoded as the labeled graph shown on the right. While multiple elements can have a ground connection at different positions the ground is still represented by a single vertex in the representation.

The graph defines the discrete elements, such as elements, connections and their types. The parameters represent the numerical elements such as weight, size and spring constants. An example of a mechanism encoded as a graph G is shown in Figure 2.3. Formally an undirected, labeled graph is defined as the ordered triple $G = (V, E, \Phi)$ where

- V is a set of vertices, also known as nodes, e.g. Figure 2.3 the vertices are the masses m_n and ground g.
- E is a set of edges, which are unordered pairs of vertices: an edge always connects two distinct vertices, e.g. in Figure 2.3 the edges are the hinge H_n and spring connections K_n.
- $\Psi : E \to \big\{\{x, y\} \mid (x, y) \in V^2 \wedge x \neq y\big\}$ is an incidence function which maps every edge to an unordered pair of vertices, e.g. in Figure 2.3 the edge K_1 is mapped to the vertices m_2 and m_3.

All mechanisms graphs are rooted because a single vertex, in this case the ground g, is both fixed and unique. This is considered the *root vertex* of the graph. To complete the representation a list of numerical parameters P corresponding to each of the labels has to be given.

The connections in the graph are specified by the incidence function Φ which maps every edge to a pair of vertices. As incidence function we use an incidence matrix: an incidence matrix is a Boolean matrix where the columns represent edges and the rows represent masses. If a mass is connected to an edge, the value at that row, column combination is set to 1. All other entries are set to 0. It follows that columns can only contain two nonzero values, while rows can contain multiple. The incidence matrix belonging to the mechanism of Figure 2.3 is shown in equation 2.5:

$$\Psi = \begin{bmatrix} & H_1 & H_2 & H_3 & K_1 \\ \mathbf{g} & 1 & 1 & 0 & 0 \\ m_1 & 1 & 0 & 0 & 1 \\ m_2 & 0 & 1 & 1 & 0 \\ m_3 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.5}$$

Each label carries its own parameters as defined by the parameter list.

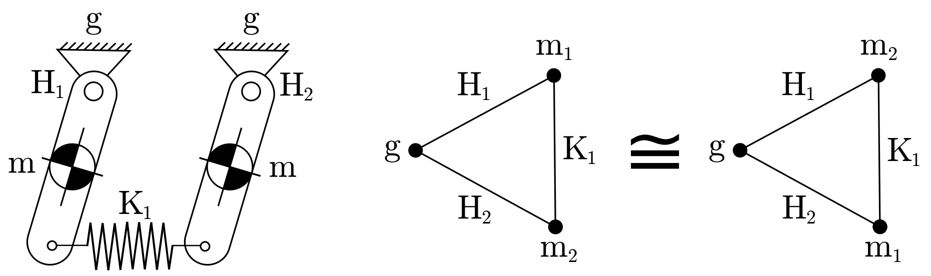

Figure 2.4: Two graphs describing the same simple mechanism consisting of two pendulums connected to the ground and to each other by a spring. The graph on the left can be obtained from the graph on the right by exchanging the numbering of the masses m_1 and m_2: the graphs are *isomorphic.*

2-4 Related graph properties

Several graph properties are closely related to properties of the mechanisms represented by them and will be used to refine the encoding. This section gives a short, qualitative overview of these. For a more detailed treatment see e.g. [49].

2-4-1 Graph isomorphism

Representing the structure of mechanisms with labeled graphs is intuitive but has a distinct pitfall: the resulting encoding is not unique. Multiple different labeled graphs can result in identical mechanisms. This follows from the fact that the elements are interchangeable; the order of numbering is irrelevant, only the label *type* matters. Figure 2.4 illustrates this problem for a simple mechanism structure.

The two masses are not initially numbered but to assign parameters we need to label the graph such that the vertices are distinct. However, there is more than one way to label the masses as there is no distinct starting point available. Depending on whether the left or right mass is picked as the starting point m_1 a different graph representation and incidence matrix is obtained.

From the graphs shown in Figure 2.4 we can determine the incidence matrices as:

$$\begin{bmatrix} & H_1 & H_2 & K_1 \\ \mathbf{g} & 1 & 1 & 0 \\ m_1 & 1 & 0 & 1 \\ m_2 & 0 & 1 & 1 \end{bmatrix} \cong \begin{bmatrix} & H_1 & H_2 & K_1 \\ \mathbf{g} & 1 & 1 & 0 \\ m_1 & 0 & 1 & 1 \\ m_2 & 1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

While the incidence matrices and graphs are not identical, they actually represent the same mechanism. It's still possible to say that the two graphs are closely related because if the numbering is discarded, they are in fact identical. They share the same elements and connections. In that case the graphs are considered *isomorphic.* (Denoted by $\cong$).

Two graphs are considered isomorphic if they can be transformed into each other by moving around the labeling but keeping all connections and elements intact.

By checking the mechanism graphs for isomorphisms the effective search space can be significantly reduced [25].

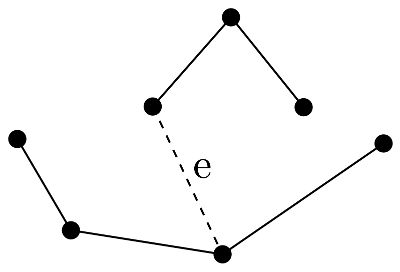

Figure 2.5: This graph is fully connected since a path exists between every vertex. However, if the edge denoted by e is removed this is clearly no longer the case: it becomes disconnected.

2-4-2 Graph connectivity

Another quirk of the representation is that, in principle, mechanisms can be defined that are disconnected. It's possible to define elements that are entirely free floating, mechanisms that consist of several different disconnected sections. Even two unrelated complete mechanisms can technically be defined within a single representation.

The connectedness of mechanisms is defined by another important graph property: graph connectivity. A graph is considered connected if it has at least one vertex and there exists a path between every pair of vertices in the graph. Figure 2.5 illustrates this concept.

It can be checked if a graph is connected by traversing the graph, following all the edges from the starting vertex, and marking any traversed vertices as visited. If no more edges are available to traverse and all vertices have been marked the graph is considered connected. By enforcing that a graph that represents a mechanism *has* to be fully connected loose ends and disconnected elements can be prevented.

2-5 Conclusion

In this section an explanation was given of the issues of automated mechanism design that prior techniques ran into. Significant strides have been made to reduce the complexity of the search space [19, 20, 50]. Graph isomorphism can be used to significantly reduce the search space. However, graph isomorphism checks are very computationally expensive and should be performed as rarely as possible to keep performance up.

Novelty search and quality diversity algorithms have been successful by injecting the necessary exploration to avoid deceptive attractors [26,35,38,51]. However, these approaches require strict boundaries on the search space to prevent the algorithm from generating functionally meaningless novel solutions.

None of the approaches have thus far solved the underlying linkage problems in the mechanism design search space.

Chapter 3

Mechanism model Extensions

This section will explain the modifications to the mechanism representation structure. The original structure is summarized in Section 2-3-2. These are necessary because the prior graph-based mechanism model fails to satisfy the following requirements:

- The mechanism representation should have clear, pre-defined points of interaction with the environment. Requiring the introduction of the end-effector element. This is necessary to reduce redundancy.
- The mechanism representation model should include active motors to facilitate possible control elements. This requires the introduction of an active motor element and controller map. This is necessary to make the representation generalizable.
- The mechanism representation should allow parameters to be defined arbitrarily, i.e. possibly indirectly through an external function. This requires the introduction of the parameter map. This is needed in particular to functionally decouple the representation of the parameters from the representation of the discrete elements.

3-1 Additional elements

While the graph-based representation of mechanisms has proven to be effective [25], it lacks the proper extensions to deal with motors and control. In addition, a pick-and-place robot usually has a single end-effector, to properly model a robot the end-effector should be included in the model. This requires the inclusion of newly defined end-effector, motor and controller elements to the mechanism representation.

3-1-1 The end-effector

Robots and mechanisms often have fixed points of interaction with their environment, either a gripper, linkages to external systems or other such connections. To model these kinds of relations the end-effector element is introduced to the representation.

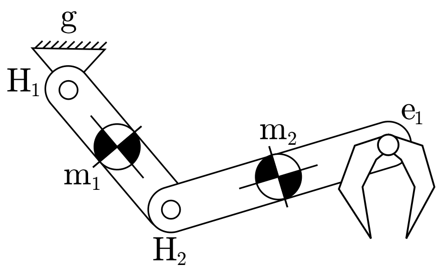

Figure 3.1: A mechanism with added end-effector e_1, the end-effector is visualized as a gripper and modeled as a point mass.

The end-effector (e), like the ground (g), is a unique vertex that can be added to the structure of the mechanism. It represents the ability of the mechanism to manipulate the world around it. To account for connections with varying degrees of freedom a fully constrained connection type (F) is introduced alongside the end-effector.

The end-effector behaves like a variable mass point-mass. Picking up an object can be simulated by adding the mass of the object to that of the end-effector, external forces can also be applied to the end-effector to model the interaction with the environment. Like the ground vertex is always the first, the end-effector is always the last. This aligns with the concept that a mechanism structure is defined by the (parallel) chains of elements between the ground and the end-effector.

3-1-2 Motors

The motor element is a special element in the mechanism representation structure as it is able to add and extract mechanical energy, it is a non-conservative element. It provides the connection bridge between the controller and mechanism representation structures. A controller must be attached to the motor, which applies a (possibly state-feedback) control torque to the mechanism. This controller representation is independent from the mechanism representation, but can be correlated to the mechanism structure.

A controller in the mechanism representation is considered a black box function, in which the mechanism state and time is provided as input and a control torque results as output. If no controller is attached a motor will be treated as a passive hinge.The separation of representations facilitates a co-evolutionary approach to the optimization problem.

3-1-3 Controllers

As controllers are part of more complex mechanism structures it is important to keep them in mind as well. In order not to inherently limit the possibilities of controller encodings these are considered as a separate attachment.

Controllers don't fit neatly into the rest of the mechanism representation since the controller itself can technically be considered a parameter of the mechanism representation. I.e. to fully define a controlled mechanism the parameter set contains not just real-valued vectors defining the mechanism element parameters and positions but also controller objects, which have their own respective independent representations. The result is a nested structure.

A controller is defined as a function that maps the time t, states $\mathbf{x}$ and velocities $\dot{\mathbf{x}}$ of the mechanism to a control output torque τ:

$$C\left(t, \mathbf{x}, \dot{\mathbf{x}}\right) = \tau \tag{3.1}$$

A mechanism M can be assigned an ordered set of controllers $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ to result in a controlled mechanism $M_C = \{M, \mathcal{C}\}$. Where the number of controllers is equal to the amount of motors present in the mechanism.

The controller representation itself is purposely left unspecified in the general model of the controlled mechanism representation. Many different competing evolutionary controller models exist [10,52–55] and the optimal control strategy choice is ultimately highly problem-dependent.

3-2 Parameter map

3-2-1 Parameter List

The parameter list is the general structure used for a single mechanism definition, the parameter function is defined as a list lookup of uncorrelated parameters. This can be viewed as a trivial case of a parameter assignment function: rather than generating a set of parameters the set is completely predefined.

Each label in the list fundamentally corresponds to an element type, which has a known predefined structure of parameters. The position of the element is always represented by the first two parameters, except for springs which define two connections and no constraint. For the elements in the representation this leads to the following definitions from prior art [19]:

- **Ground (g):** The ground g is the root vertex and has no corresponding parameters (positions are encoded in edge connection locations)
- **Mass (m):** $\begin{bmatrix} x & y & m \end{bmatrix}$, the mass of the element, as well as the position of the center of mass at $t = 0$. This position is only used in case the position is not already fully defined by the connection elements.
- **Hinge (H):** $\begin{bmatrix} x & y \end{bmatrix}$, the x and y position of the hinge at $t = 0$
- **Spring (K):** $\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & l_0 & k \end{bmatrix}$, the two connection positions at $t = 0$, the zero length l_0 and the spring constant k of the spring

And the new elements introduced here:

- **Torsion Spring (T):** $\begin{bmatrix} x & y & \theta_0 & k \end{bmatrix}$, the position of the hinge, the zero angle θ_0 and the spring constant k.
- **Motor (M):** $\begin{bmatrix} x & y & C_n \end{bmatrix}$, the position of the hinge, and the number of the assigned controller C_n.
- **Fixed connection (F):** $\begin{bmatrix} x & y \end{bmatrix}$, the position of the fully constrained connection.
- **End-Effector (e):** $\begin{bmatrix} m \end{bmatrix}$, The end effector e is the second fixed vertex element, which only has a mass parameter (the position is fixed through the connection elements)

Other elements are possible in this representation by defining them on the graph structure and defining the required parameters to describe these new elements.

3-2-2 Parameter function

The parameters of the mechanism determine the mechanism behavior to a significant degree, as exemplified by the large amount of 4-bar linkage mechanisms in existence [56]. Every 4-bar linkage has the same graph structure so clearly it are the relations of the parameters that explain the behavioral differences between these mechanisms.

There also exists an underlying structure within these parameters; effective motor parameters depend on the mass and length parameters, torsion spring constants on the moment of inertia. Many such relations exist.

As a result rather than the absolute parameters it is the relative relation of the parameters with respect to each other that ultimately determines the mechanism behavior. A simple example to illustrate this concept would be to imagine changing the units from Metric to Imperial, while the numeric values change the mechanism behavior remains exactly the same. This property can be shown from Newton's laws of motion [57].

Simply describing the parameters as a list is not able to capture all these properties. Following the realization that the parameters are both correlated and subject to constraints the following definition is proposed:

Definition 1. *Given a labeled graph G, a parameter assignment $P : G \to \{\bigcup_k \{\mathbf{p}_i\} \mid \mathbf{p}_i \in \mathbb{R}^n \wedge k = |G|\}$ is a map that, to each vertex and edge, assigns a corresponding real-valued vector of numerical parameters.*

Applying P to a labeled graph structure G results in an ordered set of parameter vectors $\{\mathbf{p}_1, \mathbf{p}_2..., \mathbf{p}_k\}$ where the total amount of parameter vectors k is equal to the sum of the amount of vertices and edges.

By defining parameter assignment as a function, rather than more weakly tying the numerical parameters to a list, it is possible to define a parameter generation function that explicitly takes the parameter correlations on the mechanism into account. The set of all possible parameters can be mapped to a space, the domain of possible parameter assignments. This domain represents the set of spaces of possible mechanisms corresponding to each mechanism graph structure.

In a general sense P is an *indirect encoding* (a function which generates parameters encoded in a generator function rather than drawing from a list) which has shown to be effective for neural networks [58, 59]. Additionally the parameter function can be split off from the graph representation, resulting in a structure representation given by the graph G, and a parameter representation given by $P(G)$.

3-2-3 Parameter invariants

Mechanism parameters can be transformed by a *behavior preserving transformation*: one that scales, rotates and otherwise influences the parameters but does not change the behavior. These are transformations on the parameters such that the mechanism fitness does not change. I.e. these transformations lie in the nullspace of the gradient of the objective function. To be more specific, it is possible to specify operations for which the behavior in question is *invariant*.

An interesting consequence is that the parameter assignment function can be designed to be aware of these invariants. This allows a reduction in search complexity, as mechanisms that are different only in parameter scale are treated as identical. The domain of P is contracted such that any regions that represent parameter invariants of the objective function are reduced to single points.

To makes this a bit more concrete we can look at the example of finding a particular resonance frequency for the simple mass-spring system in Figure 3.2.

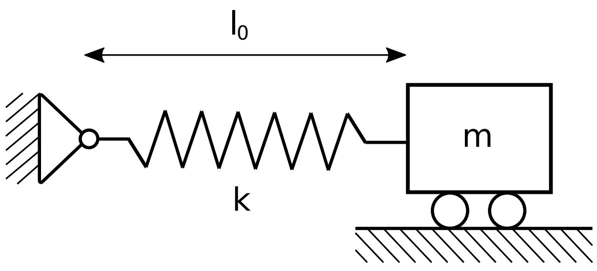

Figure 3.2: A simple Mass-Spring system described by the set of parameters m, k, l_0

We can derive the resonance frequency of this system as $\omega = \sqrt{\frac{k}{m}}$. Clearly any transformation of k, m and l_0 that preserves the ratio $\frac{k}{m}$ lies in the nullspace of the gradient of the objective function. We can simplify the complexity of this problem by looking for the ratio $r = \frac{k}{m}$ instead of all parameters k, m and l_0, a reduction from three to one dimensions that is achieved by contracting the search space using the parameter function.

Doing this for the mechanism parameter function is not as trivial as forces, inertia, mass, lengths all interact to produce the final behavior of the dynamics of the mechanism but this is already very useful to for example fix an end-effector starting position or to correct for trajectory scaling. This significantly reduces search space redundancy, one of the major problems of automated mechanism design.

3-2-4 Constraints and correlations

Constraints on the parameter assignment function provide boundaries for the parameters and are implemented as rules on P. They explicitly modify the domain of possible solutions.

The parameter assignment function is considered correlated if for some element of the assignment function depends on the values at other edges or vertices. While in a way a trivial observation, existing correlations provide valuable information that can be provided as input for an optimization algorithm: Assume a correlation exists, when adding or removing an element to the mechanism a correlation rule h could then look like:

$$P_{n+1} = h\left(P_n\right) \tag{3.2}$$

i.e. when an element is added or removed the new parameters (which have a different size) are some function of the old parameters. For example a rule such that, when adding another mass and spring to the mass-spring damper of Figure 3.2, the new parameters are such that the first eigenfrequency of the new system is identical to the eigenfrequency of the single spring-mass system.

3-2-5 Effect on the evolutionary algorithm

As per the chain rule the gradient of the objective function with respect to the parameters, while keeping the structure fixed, is now defined as:

$$\frac{\partial f}{\partial \mathbf{p}} = \frac{\partial f}{\partial P}\frac{\partial P}{\partial \mathbf{p}} \tag{3.3}$$

e.g. the gradient is projected along the directions spanned by the parameter function. In an evolutionary algorithm we normally provide a change in parameters $\Delta \mathbf{p}$ and observe the change in fitness Δf. By using a parameter map function to generate this change in parameters we ensure the effect is always along $\frac{\partial f}{\partial P}$, such that desired properties will be maintained.

This leads to the important result of the introduction of the parameter map function: different parameter functions can be used as input for the creation of operators, each of which can represent possible sampling strategies that explore the objective function along directions that maintain certain desired behavioral properties. By doing this, multiple parameter invariants can be explored simultaneously.

3-3 Representation example

To summarize, the representation structure $M = \{G, P, \mathcal{C}\}$ defines an actuated mechanism:

1. The labeled graph structure $G = \{E, V\}$, with the unique end-effector and ground vertices. Including the edge labels of the connections.

2. The parameter assignment map P, which provides either the parameter set explicitly or defines a function to generate that set.

3. The controller set $\mathcal{C} = \{C_1, C_2, ..., C_n\}$, if applicable. The controller has it's own nested representation which is separate from the mechanism.

To illustrate the complete representation of a mechanism and controller we can use the example mechanism shown in Figure 3.3: a spring-connected double pendulum that is driven by a motor with a PD controller at the base with reference signal θ_{ref} and input state θ_1, which is the angle of rotation of the mass element m_1 in Figure 3.3.

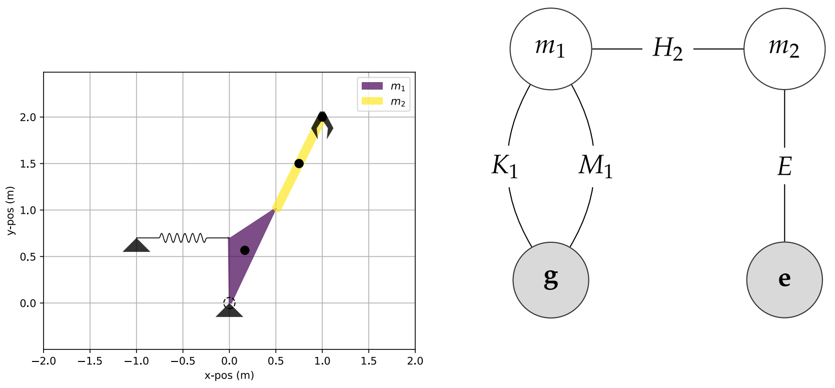

Figure 3.3: The motorized mechanism (left) and it's graph representation (right), the ground and end-effector are a unique vertices. The motor is represented by the dotted circle, the ground locations by the black triangles and the end-effector by the gripper.

From the graph structure in Figure 3.3 we can deduce that it has the following incidence matrix:

$$\Psi = \left[\begin{array}{ccccc} & E & M_1 & H_1 & K_1 \\ \mathbf{g} & 0 & 1 & 0 & 1 \\ m_1 & 0 & 1 & 1 & 0 \\ m_2 & 1 & 0 & 1 & 1 \\ \mathbf{e} & 1 & 0 & 0 & 0 \end{array}\right] \tag{3.4}$$

The mechanism has the following parameter set, representing the connection locations and parameters:

$$P = \left\{\begin{aligned} m_1 &= \begin{bmatrix}1 & 0 & 0\end{bmatrix} \\ m_2 &= \begin{bmatrix}1 & 0 & 0\end{bmatrix} \\ E &= \begin{bmatrix}1 & 2 & 0.1\end{bmatrix} \\ M_1 &= \begin{bmatrix}0 & 0 & 1\end{bmatrix} \\ H_1 &= \begin{bmatrix}0.5 & 1\end{bmatrix} \\ K_1 &= \begin{bmatrix}-1 & 0.7 & 0 & 0.7 & 1.6 & 10\end{bmatrix} \end{aligned}\right\} \tag{3.5}$$

The motor is assigned controller 1, which has its own independent representation which is separate from the representation presented here.

3-4 Conclusion

This section introduced additional elements which extend the prior graph representation of mechanisms. The motor element allows the addition of controlled motors, while the end-effector is introduced as a pre-defined point of interaction used by the objective function.

This section also introduced the concept of the parameter map function. The parameters can be derived from functions, which represent invariants in the search space. These invariants can be captured by using parameter functions. By sampling the gradient of the objective along the directions of these parameter functions which describe invariants the overall search complexity of the evolutionary algorithm is reduced. This can be achieved in practice by using these parameter functions as input for operator generation rules in an evolutionary algorithm, which reduces both deception and search space redundancy.

Chapter 4

ROSE: Reduced Operator Space Evolution

This section will explain the core concepts of reduced operator space evolution. First operator-representation duality is shown, which uses the decoupling of the parameter function from the representation introduced in chapter 3, then the mechanism space is constructed based on this duality. Using the newly defined mechanism space a new evolutionary algorithm is introduced.

This novel algorithm is necessary in order to solve the linkage problems that have plagued automated mechanism and robot design algorithms. In particular, it decouples the mechanism representation from the optimization algorithm sampling strategy through the use of abstract operators.

4-1 Introduction

In order to solve the linkage and deception problems we need to build an understanding of how the algorithm operates and how these problems manifest in the search for an optimal mechanism design.

Section 3-2 shows that the set of parameter functions has an associated domain of possible mechanism designs. By applying a set of constraints on these parameters we can bound and contract this domain, and thus the possible mechanism designs corresponding to that structure which reduces deception. Adding effective correlations such as done for the mass-spring system lets us map the problem to a lower-dimensional space as well while dealing with linkage problems effectively.

This inspires an investigation into how these insights can be applied to the structural (discrete part) of the computational design optimization problem as well.

These obtained insights are used to develop an effective evolutionary algorithm.

4-1-1 Fitness function

Ideally we want to explore the fitness space in a structured way, as this is the value we desire to optimize. Yet in practice it is only the mechanisms which we can influence directly through

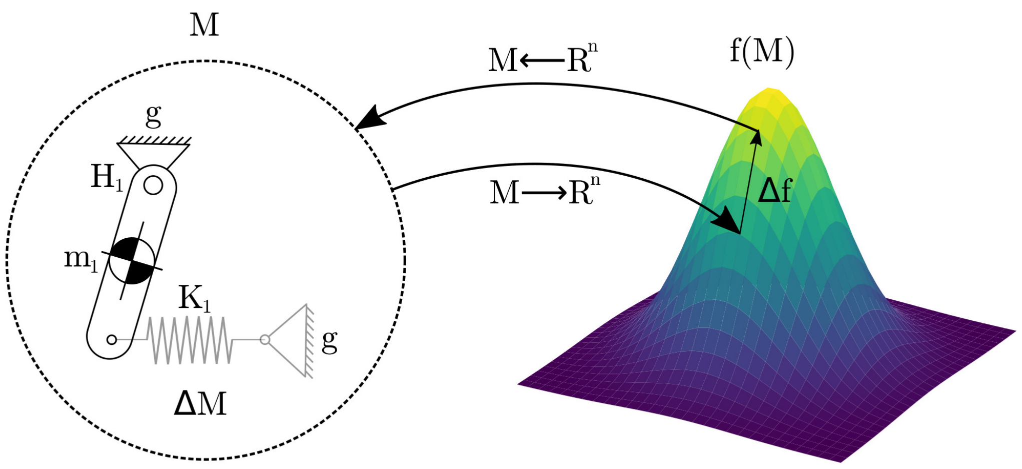

Figure 4.1: A mechanism corresponds to an n-dimensional fitness value in some subset of the reals $\Omega \subset \mathbb{R}^n$ as mapped by the objective function $f : \mathcal{M} \to \Omega$. The mapping from mechanism to fitness is obtained through simulation. The resulting question that defines any optimization algorithm is then: how do we relate a desired change in fitness Δf back to a change ΔM on a mechanism? This is nontrivial as the inverse mapping $f^{-1} : \Omega \to \mathcal{M}$ is unknown a priori and not unique.

operators. A direct relation is not known, as the fitness function requires simulation to be evaluated; it can only be sampled. The question that arises: How can we map a desired transformation in fitness space to a transformation on a mechanism, doing essentially the inverse of the evaluation step. The intuition behind this concept is illustrated in Figure 4.1.

Evaluating f requires simulation, so it follows that while $f(M)$ is known exactly it's gradient $\frac{\partial f}{\partial M}$ is unknown. In fact, since the mechanism structure representation is discrete the gradient only exists locally as a function of the parameter function P when the structure G is kept fixed. Yet the gradient of the mechanism in fitness space, but expressed in mechanism space coordinates. Or more simply, how fitness function evolves with respect to the mechanism parameters, is precisely the knowledge need to move in the direction of better fitness.

So while $\frac{\partial f}{\partial M}$ isn't available we need to use some estimate of it, either implicitly or explicitly, to move the mechanisms in the direction of better fitness i.e. we need it in order to construct an optimization algorithm. We can however *sample* the fitness function.

Using the sampled fitness values we can construct an a posteriori estimate $\hat{f}(M)$ of the fitness function $f(M)$. These estimates can guide the optimization algorithm.

4-1-2 Operators

In order to study an optimization algorithm in the context of mechanisms it is necessary to look at ΔM from Figure 4.1 in some more detail. It is clear what Δf is since it is a real valued number or vector:

$$\Delta f = f(M_2) - f(M_1) \tag{4.1}$$

What is ΔM? We might try just applying the above expression but for mechanisms which results in:

$$\Delta M = M_2 - M_1 \tag{4.2}$$

but this actually has no meaning, since the minus operation for mechanisms is not defined.

We know that the new mechanism can obtained from the old one by applying some operation φ which acts on the graph representation and parameters of the mechanism representation: by adding the spring K_1 and connecting it to the ground g and to the link m_1 we obtain M_2 from M_1. This is visualized in Figure 4.2. Using this insight we can define operators on mechanisms

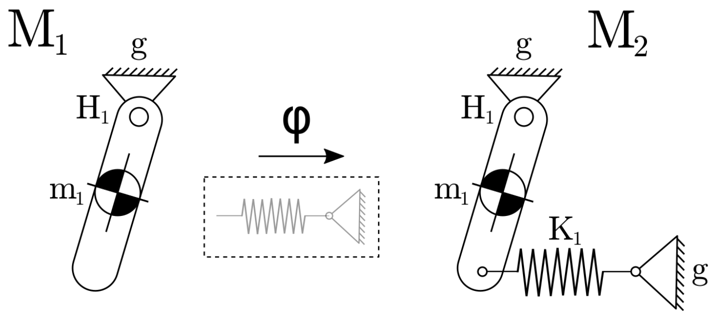

Figure 4.2: A general operation on a mechanism can itself be visualized as a section of mechanism to be added or subtracted from the structure, combined with possible parametric changes to any of the shared elements.

in the following way:

Definition 2. *An operator $\varphi : \mathcal{M} \to \mathcal{M}$ is a transformation that maps a mechanism to another mechanism*

Armed with this definition we can now determine the meaning of ΔM:

$$\Delta M = [\varphi : \varphi(M_1) = M_2] \tag{4.3}$$

In words: The difference ΔM between two mechanisms M_1 and M_2 is given by the operator φ which maps M_1 to M_2 such that $M_2 = \varphi(M_1)$. This gives a nice interpretation to the question posed by Figure 4.1: Finding the mechanism with increased fitness means finding the right operator to apply.

As shown in chapter 3 a mechanism is represented as the combination of a labeled graph defining the structure, and a list of parameters of the elements in the graph. The graph structure is discrete, while the parameters are continuous. As a consequence operators are also described as labeled graph structures, fundamentally they describe pieces of mechanisms to be connected to or removed from the mechanisms. In the special case where the structure of the mechanism remains the same the operators reduce down to parametric maps, which can be represented in a much simpler way.

Inverse operators also exist for every operator i.e. $\varphi^{-1} \circ \varphi = \varphi_I$ where φ_I is the identity operator: $\varphi_I(M) = M$. A trivial example is the operation of adding a link, where the operation to remove that link is its immediate inverse.

4-1-3 Operator representation

To utilize the operators in practice we need to be more specific about their representation. An operator φ can be represented as a chain of possible parameterized operations on the mechanism representation, of which we can identify the following fundamental base types for a mechanism exploiting the fact that they are described as labeled graphs.

- $\{\varphi_G\}$ The set of canonical graph operators for labeled graphs [49]: add a node, remove a node, add an edge, remove an edge, relabel a node, relabel an edge.

- φ_p The parameter point mutation

We can use this operator base to define the general structure of a mechanism operator. Note that the order of the operators is important. For example we can define an operator φ_l that adds a link with a hinge connection in the following way: first add a link (vertex), add the parameters for the link, add a hinge connection for that link (edge) and then add the parameters for that hinge connection:

$$\varphi_l = \varphi_p^2 \left(\varphi_e \left(\varphi_p^3 \left(\varphi_v \left(\ldots\right)\right)\right)\right) = \varphi_p^2 \circ \varphi_e \circ \varphi_p^3 \circ \varphi_v \tag{4.4}$$

The symbol $\circ$ represents the ordered composition of the operators as shown in equation 4.4. For notational simplicity the superscript on the parameter mutation indicates the number of chained parameter mutations. This composition of operators can be visualized as a chain of base operations as shown in Figure 4.3.

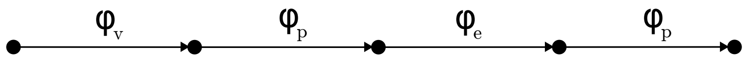

Figure 4.3: A chain of base operators defines the operator to add a link φ_l, each vertex in this chain represents an in-between mechanism representation.

A note on crossover

Crossover in an evolutionary algorithm is the recombination of sections of the representation between two members of the population, essentially mixing both representations together. In contrast to generating a mutator operation which is only a function of a single mechanism a crossover operator generator is a function of two mechanism representations.

It is possible to represent any crossover event as a general operator so it is not necessary to explicitly include them here from a theoretical viewpoint. Studying crossover operators is however of great interest for further research.

While crossover has been shown to be possible for mechanisms [19] it's effectiveness suffers from the linkage problems, the behavior of the parent mechanisms that makes them successful is not preserved. A good example of this is shown in Figure 1.5. For this reason only mutations are considered here.

4-1-4 Complexity measure

We have obtained a description for ΔM, but still no ability to quantify and compare these differences explicitly. Logically an operator that does a lot of modifications should result in a larger ΔM than one that only does a few. By applying concepts from information theory we can quantify this more specifically.

To quantify the complexity of a mechanism I propose the following definition:

Definition 3. *The complexity measure $c : \mathcal{M} \to \mathbb{R}_{\geq 0}$ is a function that for each mechanism assigns a positive real valued number. It is equal to the self-entropy of the mechanisms representation*

The self-entropy [60] is given by:

$$c(M) = -\ln(p(M)) \tag{4.5}$$

Where $p(M)$ is the probability of this particular mechanism randomly occurring on it's representation.

In other words: the complexity of the mechanism represents the log-inverse of the odds of the mechanism to occur when randomly generating mechanisms. This definition of complexity has an important corollary: *changing the base of the representation can increase or decrease the complexity value of a mechanism.* This happens because modifying the base can exclude or include possible mechanism structures, modifying the probability $p(M)$.

We can obtain an explicit form by calculating the probabilities. The probability of a mechanism occurring randomly is given by the multiplication of the probability of each of it's elements:

$$p(M) = \prod_i p(M_i) \tag{4.6}$$

The complexity of the mechanism is given as the negative logarithm of the probability, using equation 4.6 for the probability of the mechanism:

$$\begin{aligned} c(M) &= -\ln(p(M)) \\ &= -\ln\left(\prod_i p(M_i)\right) \\ &= -\sum_i \ln(p(M_i)) \end{aligned} \tag{4.7}$$

This means that the complexity of a mechanism is the sum of the complexity of it's parts. The exact probability $p(M_i)$ of an element occurring technically depends on factors such as the bit-depth of the representation. But since it is a constant multiplication factor that is identical to every parameter and every mechanism, we can filter it out. We care about the about the comparative complexity between mechanisms rather than the exact numerical value.

If we assume p_f is the probability of randomly drawing exactly one value for the parameter from a uniform distribution[1], which is a shared constant factor between all elements. Then an element's probability is determined by it's degrees of freedom n, as given by table 4.1.

$$p(M_i) = \prod_n p_f = (p_f)^n \tag{4.8}$$

if a parameter has n degrees of freedom then $(p_f)^n$ is the probability of drawing that parameter. This results in the following equation for the mechanism complexity:

$$c(M) = -\sum_i \ln\left(p_f^{n_i}\right) = -\ln(p_f)\sum_i n_i \tag{4.9}$$

[1]When using double-precision floating-point representation and every number is equally likely $p_f \approx 2^{-64}$

Element	**Degrees of freedom**
Ground	0
End-effector	0
Mass	3
Hinge	2
Spring	6
Torsion spring	4
Motor	2*

Table 4.1: The amount of parameters required to fully describe particular elements are the degrees of freedom of the element. Note that for the degrees of freedom we only care that the connection exists, not what two masses it connects: the connection is guaranteed to always be between two masses that already exist in the structure and these are already counted. The motor is a special case, it has two degrees of freedom but also an associated controller object. The controller has it's own representation and degrees of freedom which adds more complexity to the mechanism. If no controller is added, a motor simply acts like a hinge in every way.

We don't care about the value of p_f, but note that it is by definition smaller than one, so that $\ln(p_f) < 0$. We can drop this negative constant and cancel out the minus signs to get a simplified complexity equation for the mechanism:

$$\hat{c}(M) = \sum_i (n_i) \tag{4.10}$$

The mechanism complexity from definition 6 is, apart from a constant factor $-\ln(p_f)$, equal to the sum of the degrees of freedom of it's elements *if* every possible parameter permutation is assumed to be equally likely.

From the definition of mechanism complexity we can also infer that there is a minimal representable mechanism. This minimal mechanism is defined as:

Definition 4. *The minimal mechanism $\hat{M} \in \mathcal{M}$ is defined as the (possibly not unique) mechanism with the lowest complexity score, such that for every $M \in \mathcal{M}$ on that representation it follows that $c\left(\hat{M}\right) \leq c(M)$.*

In an identical way we can define the complexity of an operator and the minimal operator. Note that the minimal operator is a walk where all vertices are distinct, known as a path [49]. If any vertex appears twice in a walk this means that somewhere along the line two base operators acted as each others inverse and canceled out, which means that a shorter walk is possible and such an operator is not minimal.

4-1-5 Distance metric

Using the complexity measure we can finally put a number on ΔM, this number can actually be interpreted as a (pseudo) distance metric. From the mathematical definition of a metric space [61] we obtain the properties that are required in order for a distance metric:

The mechanism distance metric is defined as a function $d : \mathcal{M}^2 \longrightarrow \mathbb{R}_{\geq 0}$ that maps two mechanisms into a positive real valued number such that:

- The distance from a mechanism to another mechanism is than zero $d(M_i, M_j) > 0,\ M_i \neq M_j$
- The distance from a mechanism to itself is exactly equal to zero $d(M_i, M_j) = 0,\ M_i = M_j$.
- The distance satisfies the triangle inequality $d(M_i, M_j) + d(M_j, M_k) \geq d(M_i, M_k)$
- The distance satisfies $d(M_i, M_j) = d(M_j, M_i)$.

Note that we can identify the following relation between M_i and M_j:

$$M_i = \hat{\varphi}_i^j(M_j), \quad \hat{\varphi}_i^j \in \Phi \tag{4.11}$$

Where $\hat{\varphi}_i^j$ is the minimum complexity operator which, when applied to mechanism M_j, results in mechanism M_i. We can use this relation to define a distance measure in mechanism space:

$$d(M_i, M_j) = c\left(\hat{\varphi_i^j}\right) \tag{4.12}$$

Where $c(\varphi)$ is the complexity of the operator φ, again given by the self-entropy of the operator representation. Intuitively it scales such that mechanisms which require more operations to turn into eachother are further apart than mechanisms which are closer. It satisfies the relation $d(M_i, M_j) = 0,\ M_i = M_j$ as $\hat{\varphi_i^j} = \varphi_I$ if $M_i = M_j$ and φ_I by definition has a complexity of zero as it is an empty operator. Similarly it satisfies $d(M_i, M_j) > 0,\ M_i \neq M_j$ since $c\left(\hat{\varphi_i^j}\right) = 0$ if, and only if $M_i = M_j$.

The triangle inequality also holds, the minimum operator represents the shortest possible chain of operations between two mechanisms. Any vertices not on this path require additional operations to reach and as such $d(M_i, M_j) + d(M_j, M_k) \geq d(M_i, M_k)$. This follows from the close relation to the edit distance of graph spaces and proof is given in e.g. [ref].

That leaves the final requirement $d(M_i, M_j) = d(M_j, M_i)$. This requirement holds if $c(\varphi) = c\left(\varphi^{-1}\right)$ for all $\varphi \in \Phi$, which is not inherently the case for all possible operator sets: it is easy to imagine a destruction operator with no parameters which takes any existing mechanism and returns an empty mechanism. This means that when designing an operator some care should be taken that the requirement $c(\varphi) = c\left(\varphi^{-1}\right)$ holds when using the distance metric in a strict sense. In practice this distinction is of lesser importance and the resulting pseudo-metric is still very useful. The base set of section 4-1-3 satisfies this requirement.

Remember from definition 6 that the complexity measure is dependent on the set of base operators Φ selected. This has an important implication: *By selecting a different set of base operators, mechanisms can be moved closer or further away from each other.*

4-2 Dual representation using operators

It is possible to use the operators to represent mechanisms in the form of deltas on some predefined mechanism structure by applying equation 4.3.

Mapping the mechanisms onto an operator domain has some distinct benefits, it lets us analyze the resulting space, map evolutionary algorithms and reinforcement learning algorithms to pseudo-random walks on this graph and it lets us bound the search space in easily understood ways.

To begin we need two definitions:

Definition 5. *The* domain Ω^{Φ} *of a set of operators* Φ *is defined as the subset of mechanisms* $\Omega^{\Phi} \subseteq \mathcal{M}$ *where, for any pair of mechanisms* $M_i, M_j \in \Omega^{\Phi}$*, there exists an operator* $\varphi_i^j \in \Phi$ *such that* $M_i = \varphi_i^j (M_j)$.

Definition 6. *Given an operator* $\varphi \in \Phi$ *the associated minimal operator* $\hat{\varphi} \in \Phi$ *is defined as* $\hat{\varphi} = \arg\min c(\varphi)$, subject to $\hat{\varphi}(M) = \varphi(M)$, *where* $c(\varphi)$ *is the complexity of the operator.*

The domain Ω^{Φ} corresponding to a set of operators Φ can be visualized as the slice of the larger set of all mechanisms that is interconnected by that particular set of operators. This is closely related to the notion of graph connectivity from section 2-4-2: if we map all mechanisms as vertices to an infinite graph where the operators Φ are the edges there will be a single connected embedded subgraph, the vertices of this subgraph span Ω^{Φ}.

Lemma 1. *Given a set of operators* Φ *with domain* $\Omega^{\Phi} \subseteq \mathcal{M}$*, any mechanism* $M \in \Omega^{\Phi}$ *can be represented by the unordered pair* $\{\varphi_m, M_s\}$ *of a fixed seed mechanism* $M_s \in \Omega^{\Phi}$ *and corresponding representational operator* $\varphi_m \in \Phi$

Proof. The result follows immediately from the definition of the domain Ω^{Φ} by specifying that $M_j = M_s$. □

Lemma 2. *A mechanism* M *that can be represented by an operator and seed mechanism pair* $M(\varphi_m, M_s)$ *can be* minimally *represented by a minimal operator and minimal seed mechanism pair* $M\left(\hat{\varphi_m}, \hat{M_s}\right)$

Proof. By definition a minimal operator $\hat{\varphi}$ exists for every operator φ if $\min c(\varphi)$ exists. From the definition of $c(\varphi)$ we note this is the case for every $\varphi \in \Phi$. As by Lemma 1 M_s can be chosen freely we can simply specify it to be equal to $\hat{M_s}$ □

The resulting structure is a type of labeled graph known as a directed walk [49]. An example of this parallel encoding is shown in Figure 4.4. This parallel encoding is important since it is these walks that the evolutionary algorithm explores in an immediate sense. This structure allows us to gain intuition on deception in the operator based representation: The optimization problem is deceptive if there exists no path $\varphi_M^S \in \Phi$ from the starting mechanism, to the optimal mechanism, for which the vertices are traversed on order of increasing fitness.

This has an interesting result, changing the operator base will change this path, which directly affects deception in the search space.

4-3 Mechanism space

4-3-1 Definition

The set of possible mechanisms $\mathcal{M}$ that can be encoded by a specific representation can be considered a space. Where a mechanism represents a single unique point in this space, and operations φ on them represent transformations.

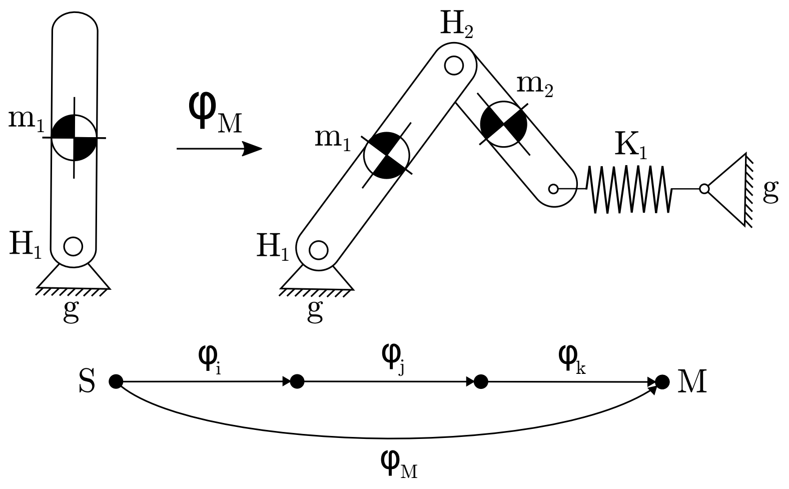

Figure 4.4: The mechanism M (right) can be represented by the operator φ_M on a base mechanism S (left): $M = \varphi_M(S)$. The operator φ_M can be expanded into any suitable choice of operator base. In this example $\varphi_M = \varphi_k \circ \varphi_j \circ \varphi_i$ where φ_i adds the mass m_2, φ_j adds the hinge connection H_2 and φ_k adds the spring connection K_1.

The algorithm is effectively searching through this mechanism space, which contains all possible mechanism solutions given this specific encoding. Evaluation of a mechanism is the mapping of a point in mechanism space onto a corresponding point of the fitness space, which is spanned by the fitness values of all possible mechanisms for this specific problem. There is a single, but not necessarily unique, fitness value associated with every point in mechanism space. As it is likely that the encoding does not capture every possible mechanism, only a subset of the fitness space is explored.

An encoding provides a base for the mechanisms and allows them to be described with a coordinate system (parameters), but the mechanism itself exists independently of the encoding; the mechanism can be transformed from one representation to the other[2] without changing it.

The set of all mechanisms in combination with the distance measure defined in section can be viewed as a space. This leads to the following definition:

Definition 7. *Mechanism space $\mathbb{M} = \{\mathcal{M}, d\}$ is a metric space which consists of the set of all mechanisms $\mathcal{M}$ and the mechanism distance measure $d : \mathcal{M}^2 \longrightarrow \mathbb{R}_{\geq 0}$.*

4-3-2 Structural representation

A mechanism consists of both a structural and parametric element, by setting the mechanism structure fixed we obtain the following special case for operators:

Theorem 1. *Let P_i be the parameter function belonging to mechanism $M_i \in \mathcal{M}$, and P_j the parameter function belonging to mechanism $M_j \in \mathcal{M}$. If M_i is* structurally identical *to M_j, i.e. their graphs G_i, G_j are isomorphic, the operator $\varphi : M_i \to M_j$ can be reduced to a bijective numerical mapping $\varphi_P^n : \mathbb{R}^n \to \mathbb{R}^n$.*

[2]Of course keeping in mind that the mechanism must exist within the intersection of the space spanned by both representations.

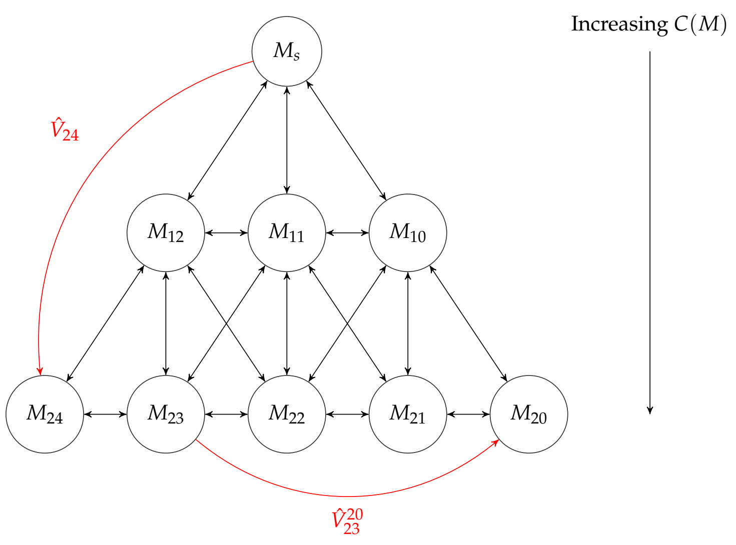

Figure 4.5: A slice of mechanism space defined as a rooted graph structure starting at the seed mechanism M_s, the edges represent the base operators used to represent transformations in this space. The vertices represent fundamental structural mechanism types. The operators transform one mechanism into another. The amount of 'hops' required to go from one mechanism to another, weighted by the complexity of those hops, is the mechanism distance $d(M_i, M_j)$.

This result is particularly notable since operators in general are *not* bijective.

By separating the parameter assignment function P from the mechanism representation the parameter mutations φ_p can be ignored. By additionally specifying a complexity bound η such that $c(M) < \eta$ for all $M \in \mathbb{M}$ the space becomes finite in size. It is possible to construct and visualize that part of the mechanism space itself as a graph structure and gain insight in what it looks like.

As the minimal seed mechanism $\hat{M}_s$ is by definition the minimum complexity mechanism allowed by the representation, any operations on it can only increase the complexity. Setting the minimal seed mechanism at the root of the graph, each node represents an allowed structural variation of a mechanism (i.e. independent of the parameter values) while the edges represent possible operators that are applied on the mechanism. This is visualized by Figure 4.5. The depth of the graph is the level of complexity of the node as given by the complexity function $c(M)$ up to η.

The structural operators represent paths from mechanism nodes to other nodes in this representation, while the parameter variations are internal to the nodes in this graph. A theoretical maximum fitness value for each of these nodes exists, but in practice only an estimate of this fitness value can be obtained.

The nodes are natural to the space, not to the operators or the algorithm, the nodes always exist. Structural operators provide the paths between the nodes, showing the possible traversal options for an optimization algorithm. The shortest path traversal between two mechanisms M_i and M_j is equal to the mechanism distance $d(M_i, M_j)$, which can be clearly visually identified in this graph structure.

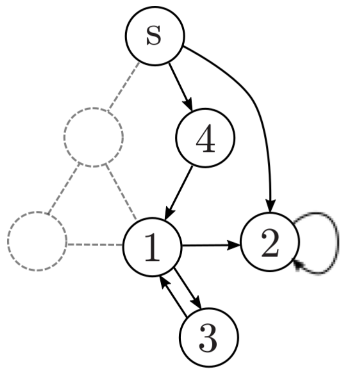

Figure 4.6: A simple evolutionary algorithm can be visualized as a walk through mechanism space. At every vertex surrounding vertices are sampled randomly, the vertex showing the most promise will become the new base for further sampling. Dotted nodes and paths are unexplored. Arrows represent applied operators. Clearly an evolutionary algorithm represents a stochastic exploration of the mechanism space graph, resulting in a subgraph of that space.

4-3-3 Operator space projection

We can exploit the dual representation of mechanisms and operators presented in section 4-1-3 to reduce the search complexity. by formulating a good choice of operator base Φ the mechanisms can be projected from their representation onto an operator domain $\Omega^{\Phi} \subseteq \mathcal{M}$ with desirable properties.

Projecting a problem onto a simpler parametrization leads to an easier optimization algorithm since (1) less parameters need to be found and (2) deception is reduced. If the projection preserves all the desired solutions this can be achieved without loss.

Nodes which are not connected and parameter values that are not reachable will be ignored completely by the optimization algorithm. Additionally, shortcuts in the mechanism space can reduce deception. Figure 4.7 illustrates how introducing these shortcuts can simplify the optimization problem.

A run of an optimization algorithm is a path through this space, where it attempts to guess the maximum fitness at each node it passes through, and picks nodes to 'survive' based on their fitness value.Because the graph nodes and their maximum fitness values are *persistent* when using the same objective every run of the algorithm explores the same set of nodes, even when using a different set of operators $\Phi^* \neq \Phi$. This makes it possible to compare two different algorithms and their effectiveness and fundamentally allows the accumulation of knowledge about mechanism space. This provides a useful set of meta-heuristics for algorithm analysis.

The building blocks processed by the evolutionary algorithm can be identified as the blocks preserved by operators. This provides a fundamental link between the schema theorem and operator structure. Operators introduce higher levels of abstraction and it is these higher abstractions that handle the linkage between the different elements in the mechanism representation. Because the operators handle the linkage problems and not the representation multiple differing linkage issues in the representation can be handled simultaneously.

This is a generalization of using parameter map functions to generate operators as in Section 3-2 to to the mixed continuous-discrete case. Rather than capturing just parametric invariants for numeric parameters, structural invariants such as connectivity can be captured as well.

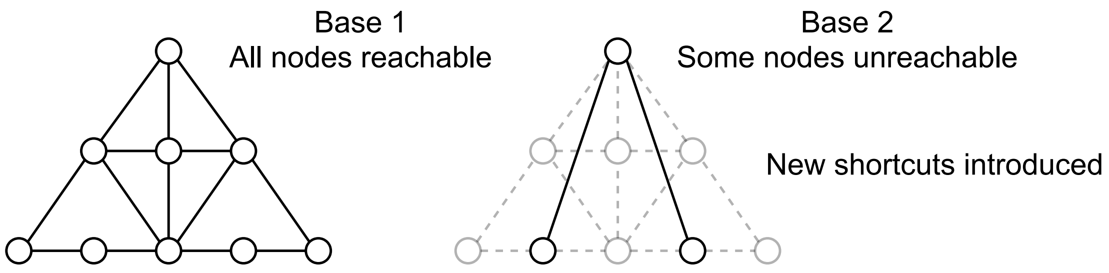

Figure 4.7: A change in operator base can be visualized as a change to the way the vertices (representing mechanisms) are connected. If the domain of the new operator base is smaller than before there will be vertices that become disconnected. The change in base will also modify the distance between the vertices. Both these properties can be highly beneficial for optimization as they fundamentally change the behavior of the evolutionary algorithm.

4-4 Reduced Operator Space Evolution (ROSE)

4-4-1 Motivation

We can draw an important conclusion from the structure of the search space derived in the previous sections: the *operators* define the behavior of the evolutionary algorithm and the preserved schemata. The actual representation of the mechanisms is only relevant to the building blocks processed by the algorithm insofar the representation informs the operators.

Let's illustrate how we can use this to our benefit with a simple example. Assume we have a beam structure such as represented in Figure 4.8: a beam that has to connect two points. During previous design operations the beam itself is already constructed, using a model that takes each separate link into account. But, there is a flaw: the end-point position is off by just a bit. To obtain the optimal solution the optimization algorithm will have to modify the parameters of every element such that the end-point is placed exactly where we want it to be while preserving the relative structure, a clear linkage problem.

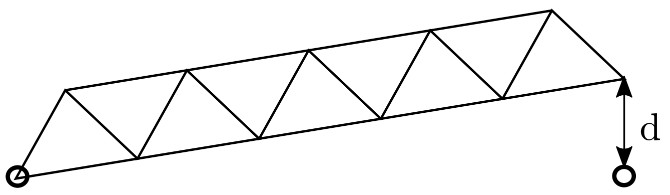

Figure 4.8: The beam structure is off by just a little bit as the endpoint is a vertical distance d from the desired endpoint, it has to be scaled and rotated as a whole to be just right.

The effect when using point mutations is dramatic: every single element will have to be modified individually, yet doing this will compromise the structure of the beam. Because performance on the other objectives also has to be maintained the result is that the algorithm will struggle to find the right solution because it tries to get there by modifying each individual link one coordinate at a time. The model representation is not conducive for this particular optimization task.

Yet something different occurs if we include a new operator to the base set Φ, an abstract operator that modifies every coordinate parameter in the system by a scaling factor s and rotation around the ground connection θ. This operator can be defined on the base as the combination of a large amount of point mutations, identical to the amount of links:

$$\varphi_T(s,\theta) = \bigodot_i \varphi_p \tag{4.13}$$

Where the composition $\bigodot$ is over all the parameters of the mechanism, in any order.

The complexity of this operator is only 2, as the operation can be fully described by two parameters. A much simpler operator compared to the complexity of 22 for the full combination of point mutations that it contains. The operator has such a comparatively simple description as it is generated from a parameter function which captures a fundamental parameter invariant, the relative beam structure.

We can visualize what happens in the mechanism space graph structure as shown in Figure 4.7. If we introduce this abstracted operator the correct solution is suddenly only one operation removed from the current solution, the abstract operator introduced a *shortcut* in the search space that can be used by the evolutionary algorithm; this shortcut introduced a path of increasing fitness from the current structure to the optimal solution. In order to achieve this we did not actually have to fundamentally modify the representation of the beam at all. It is fully contained within the abstract operator of equation 4.13.

In the light of this insight I propose a novel algorithm based around the operators, their representation and operator constraints. An algorithm that exploits the dual representation of mechanisms: Reduced Operator-Space Evolution (ROSE).

Instead of the usual point mutations and crossover design operators that represent actual design choices, ignoring the underlying representation except for the eventual implementation. These higher level operators represent combinations of the base operators. This set of operators is defined by a set of constraints on the full possibilities given by the base operators by only allowing specific combinations. As a consequence this abstract set is always some subset of the possible operators spanned by the original base. This leads to an overall reduction in operator- and algorithm complexity.

The constraints can be used to shape the search space, *because any mechanism can be represented using operators, constraints on the mechanism can be represented as constraints on those operators.* This provides an easy way to implement prior knowledge by shaping the domain Ω^Φ with the operator set. It is for example possible to select operators such that any mutated mechanism will always have one degree of freedom [50, 62]. Such an approach could inform a possible implementation of ROSE.

The operators represent a connection map of the mechanism space nodes. I.e. the nodes of the mechanism space, the structures, are independent of the choice of operators. But picking operators defines the set of edges of that space, it determines how the mechanism structures are connected and these connections ultimately determine how an optimization algorithm in this space behaves.

4-4-2 Description

The algorithm can be divided into three phases, initialization, local fitness estimation, and search space exploration.

Initially the algorithm spreads out randomly from a seed mechanism using the operators, this mechanism *can* be the minimal mechanism, but could just as well be a mechanism already close to the desired design. The path from the seed mechanism to the initial structural nodes is mapped out.

The nodes are optimized using a local optimization strategy, in this case an evolutionary algorithm. This leads to a fitness estimate for every explored node. This is similar to speciation, where the species are determined as the unique structures.

Search space exploration occurs when the local populations are considered to be sufficiently optimized. A *generator function* generates operators to apply to the most successful nodes that connect to new structural nodes, these nodes are checked for isomorphism with existing nodes to prevent duplicates. If a new node is a duplicate the old node will simply be reactivated. The worst performing nodes of the original population are deactivated so the algorithm maintains the same amount of active nodes at all times.

The end result is a ranked map of the explored section of mechanism space. A graph where each explored node contains the best discovered performing mechanism for that particular structure, the edges of that graph represent the operator paths and provide relational information.

We can identify the following two requirements on top of the usual evolutionary algorithm requirements:

- Operator set
- Operator generator function

Note that the representation model is no longer important to the effectiveness of the algorithm, as long as the operators can process the model.

4-4-3 Operator generator function

To use the abstract operator set in an evolutionary algorithm a random generator function is required. It is a function that, given a mechanism, generates an appropriate random operator from the defined set. To implement such a function an operator will be generated in accordance to a set of predefined rules. These rules enforce the parametric constraints as well.

This puts hard bounds on the possible set of resulting operators rather than on the mechanisms explicitly. Because the operator and mechanism structure are so tightly coupled this also results in implicit constraints on the mechanism search space, many nodes are made unreachable by the algorithm because the paths to them are removed. This removes the need to explicitly check for connectedness, groundedness and other necessary properties that are required for the mechanism to be functional which eat up additional computation time.

By default all paths are closed off, picking operators means choosing which paths to open up. Because the operators are higher level their effect is easier to visualize and design. It is also possible to avoid linkage problems by picking the right combinations of operators, as the linkage in the mechanism dna can be reflected as linkage in the operator set. As is illustrated by the scaling operator of the beam structure example.

4-4-4 Relation to deception, linkage and redundancy

Reduced Operator-Space Evolution can solve the fundamental problems in the system in the following ways:

- Linkage: Linkage can be taken into account by the operators, but it is not forced into the model itself. This means that using a single model a problem can be solved where for one of the objectives the linkage problem occurs, while for others it does not.
- Deception: Deception can be countered by the right choice of operators, abstract operators can create paths of increasing fitness to the solution. Additionally, by always having a large amount of active nodes and locally optimizing these it is also much less likely to get stuck in a local minimum. Every node represents a variation, and the large amount of nodes means large variance is maintained throughout optimization which is shown to be beneficial to guard against deception [41].
- Redundancy: Redundancy can be avoided in parameter space by using parameter functions to contract those areas of the parameter space that the behavior is invariant to. Additionally, the search space can be effectively bounded by maintaining desired properties such as connectivity through the operators as well by making sure these properties are structural invariants of the used operators.

4-5 Conclusion

A parallel representation of mechanisms using operators was developed, this provides more insight in the workings of the optimization algorithm and, because it is persistent, allows an in-depth definition and analysis of the problems encountered applying automated design using a graph-based representation.

This analysis led to the design of a novel operator based evolutionary algorithm called Reduced Operator-Space Evolution (ROSE) to solve the automated design optimization problem while bypassing the linkage and deception problems. This is achieved by designing operators which preserve useful properties of the mechanisms, such as the relative structure, connectivity or degrees of freedom. These properties are invariant to those particular operators. The search space is projected onto the domain of these operators, which is effective in dealing with linkage problems represented by the invariant property.

By splitting the search space further into parametric and structural optimization and injecting novelty into the search, a map of the entire search space is built and evaluated. This leads to an overall collection of solutions ranked on the objective and mapped across the space. This way meta-heuristic information is obtained and a quality diversity collection is created.

Chapter 5

PyMechs: The Mechanism Library

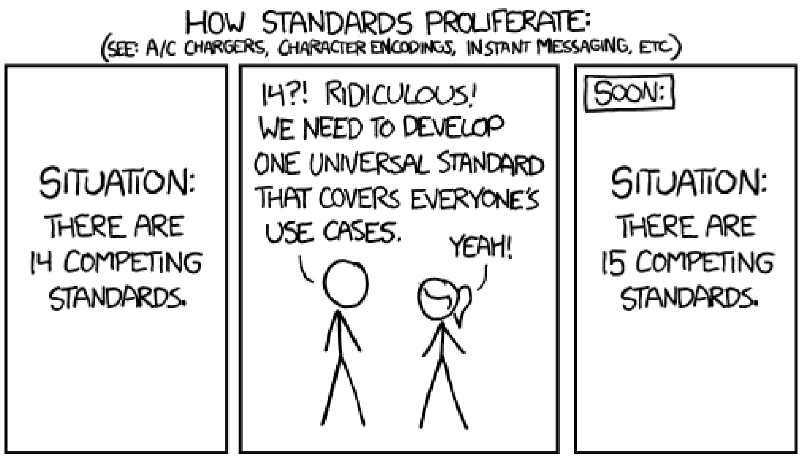
HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Figure 5.1: How standards proliferate, from xkcd.com (CC BY-NC 2.5 license) [63]

The choice to develop another new planar multibody simulation engine among many different existing implementations requires some explanation, after all it is of no use to reinvent the wheel as aptly illustrated by xkcd in Figure 5.1. Initially I investigated the option to use existing simulation libraries as well as matlab implementations developed by previous students. Takeaways

Yet while existing planar engines such as Box2d are fast and fully featured, they generally use approximation methods to obtain a result which can cause problems as the energy present in the system will not be constant. They have overhead that will not be used such as collision detection, and creating an interface adaptor for the mechanism representation and controllers to an existing library to get the right data in and out is also a time consuming process with no guarantee of payoff.

Existing matlab implementations turned out to instead not be fully featured enough, were hard to extend and incompatible with other libraries and software such as python graph-tools. Due to the nature of matlab the simulations were also slow and the lack of proper object oriented programming in matlab meant the code is very hard to maintain and not particularly modular. This limits the potential significantly.

By developing a custom library the library can be deeply tied to both the mechanism representation and the evolutionary algorithm.

Finally, developing this library has proven a valuable learning experience.

5-1 Design goals

As it is expected that the evolutionary algorithm will require many calls to the simulator it is essential that this simulator is computationally efficient, multi-threadable and accurate. Additionally, the code should be easily possible to extend to mechanism design with new types of components. Finally, each element should be modifiable independently of the others so many different controller types, evolutionary strategies and mechanism types can be tested with only minor modifications to the same core library.

From these requirements it follows that a modular approach would be the most successful. Where the heavy lifting is done by a c++ physics solver to obtain optimal performance, while the analysis, algorithm design and visualization is done from python to provide it's usability.

5-2 Object-based representation

This section provides a quick overview of the core objects of the library and their main purpose. Note that every object has a direct relation to the description of mechanism space given by chapter 4. To keep this section concise only the objects that provide a python interface are mentioned. Detailed documentation is available at https://github.com/kooswestra/pymechs.

5-2-1 DNA

The Dna class is the software form of the graph representation of the mechanisms. It stores the labeled graph structure and the set of parameters. In addition to being a data structure it provides helper functions to determine if the dna is valid according to specific criteria. The dna is checked on construction to be valid as a representation, but note that dna of an invalid mechanism structure (e.g. containing disconnected elements) may be defined as long as the representation of that structure is complete and internally consistent.

5-2-2 Mechanisms

The Mechanism class is the core object of the library. Where dna objects represent the genotype, mechanism class objects represent the phenotype. When constructing a mechanism a dna object has to be provided, the mechanism takes ownership of the dna, applies mechanism validity checks such as connectivity, and stores it. A translation is made from the encoding to obtain the links and connections that make up the structure. As a final step the initial conditions and equations of motion of the mechanism structure are assembled.

The resulting mechanism object is a container that stores all the individual elements such as links and hinges but also the mechanism state and equations of motion. It presents methods that can be called for analysis, simulation and animation.

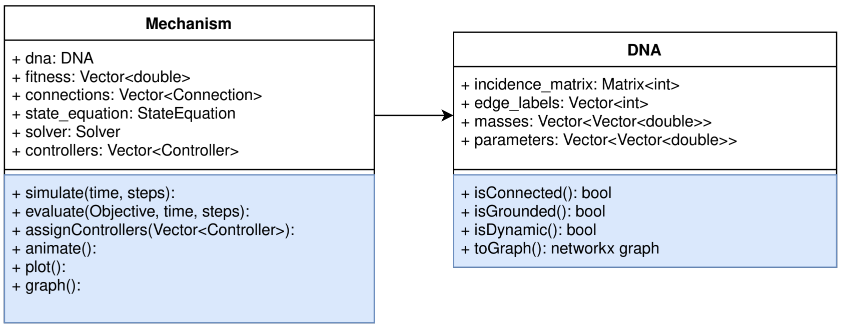
Mechanism
+ dna: DNA
+ fitness: Vector<double>
+ connections: Vector<Connection>
+ state_equation: StateEquation
+ solver: Solver
+ controllers: Vector<Controller>
+ simulate(time, steps):
+ evaluate(Objective, time, steps):
+ assignControllers(Vector<Controller>):
+ animate():
+ plot():
+ graph():
DNA
+ incidence_matrix: Matrix<int>
+ edge_labels: Vector<int>
+ masses: Vector<Vector<double>>
+ parameters: Vector<Vector<double>>
+ isConnected(): bool
+ isGrounded(): bool
+ isDynamic(): bool
+ toGraph(): networkx graph

Figure 5.2: Structure of the python accessible mechanism and DNA classes

Note that once constructed the structure of the mechanism object is not modifiable. This is in keeping with the idea that it should only be possible to get a new mechanism when obtaining a new dna object. It is however possible to assign a different controller object to a mechanism motor after construction.

The StateEquation class is an internal helper class that is always part of a mechanism object and represents the dynamics model of the equations of the mechanism. It translates the elements and connection logic of the mechanism into the mathematical objects that are required to solve for the trajectory, such as the constraint Jacobian or the equations of motion.

An overview of the structure of the mechanism class is shown in Figure 5.2.

5-2-3 Operators

Operators are defined by the Mutator[1] class. They represent instructions acting on dna object. An operator can be generated and then applied to a dna object to construct a new, modified, dna object. They have their own internal representation given by the series of base operators and corresponding parameters as described in section 4-1-3.

Several wrappers around the mutator class are provided that implement specific types of operations that can be easily identified, such as adding a link. These represent higher level abstractions of base operators as described in section 4-3-3. It is possible to implement new operators in a similar way. A complete list of the implemented operators and an implementation example is given in the appendix. These operators also implement a generation function, which allows the random generation of the operators for use with an evolutionary algorithm. I.e. randomly add a link, mutate a random parameter, etc.

5-2-4 Controllers

The controller class provides an interface to connect controllers to the mechanism motors, it is a virtual class and as such an implementation has to be chosen or provided. Any controller model can inherit from this class and will work with the library given that it implements the controlOutput method, which is the function given by equation 3.1. Both a PID controller model and the feedforward controller model used later on in section ... are provided as example implementations.

[1]I have opted to use the term Mutator for the class name rather than Operator because the latter has too much overlap with the protected *operator* keyword in c++. Mutator is an unambiguous alternative.

5-2-5 Objectives

Similar to the controllers the objective class is an interface to calculate the fitness function. An implementation has to be provided, the objective functions for the tests used in this thesis are included with the package.

5-2-6 Search graph

As shown in section 4-3 an automated mechanism design algorithm can be represented by a rooted, directed graph structure. Where the nodes represent mechanism structural variations and the edges represent structural operators. This graph structure is implemented as the FitnessGraph class. It provides a method to visualize the graph structure and stores the explored portion of mechanism space.

In line with the theoretical structure of the mechanism space, isomorphic mechanism graphs are considered identical structures and are grouped together in a single node. Inside this node a local population exists which can be modified by appropriate parametric operators. This local structure is implemented in the FitnessNode class.

To allow exactly reproducible results the evolution section of the library has a function to set the seed of the random generator used for the algorithms, parameters and other random numbers by the evolution.

5-3 Physics engine

In order to simulate a mechanism the equations of motion of the mechanism have to be determined and numerically solved. Fundamentally the equations of motion for solid bodies are given by the Newton-Euler equations, where for simplicity the angular accelerations and acting moments are added to the state and force vectors:

$$\mathbf{M}\ddot{\mathbf{x}} = \sum \mathbf{f} \tag{5.1}$$

Where $\mathbf{x} = \begin{bmatrix} x_1 & y_1 & \theta_1 & ... & x_k & y_k & \theta_k \end{bmatrix}^T$ is a vector of the states of each of the k solid body links present in the system. The equation is subject to a set of holonomic constraints $\mathbf{D}_n\left(\mathbf{x}\right)$ resulting in internal forces that are included in $\mathbf{f}$. To determine the reaction forces we can exploit the fact that the reaction forces are unable to add energy to the system. By applying the principle of virtual power we can add these reaction forces to the equations of motion using lagrange multipliers, which leads to the following relation:

$$\mathbf{M}\ddot{\mathbf{x}} + \frac{\partial \mathbf{D}}{\partial \mathbf{x}}^T \boldsymbol{\lambda} = \sum \mathbf{f} \tag{5.2}$$

However, this adds the vector of n unknown variables $\boldsymbol{\lambda}$ representing the magnitude of the reaction forces, with $\frac{\partial \mathbf{D}}{\partial \mathbf{x}}^T$ representing the direction of those forces; this leaves the equations of motion undefined as we now have k equations for $n + k$ variables. However, the constraints define k additional equations that have to be satisfied. We can add the constraints on the accelerations to the equation system in order to solve for $\ddot{\mathbf{x}}$.

Using the chain rule for $\mathbf{D}(\mathbf{x})$ and noting that the constraints added by mechanism elements are not directly dependent on time (scleronomic) we can calculate the first derivative of the constraint equations with respect to time as:

$$\begin{aligned}\frac{\partial \mathbf{D}}{\partial t} &= \frac{\partial \mathbf{D}}{\partial \mathbf{x}}\frac{d\mathbf{x}}{dt}\\ &= \frac{\partial \mathbf{D}}{\partial \mathbf{x}}\dot{\mathbf{x}} = 0\end{aligned}$$

Additionally, by noting that $\frac{\partial^2 \mathbf{D}}{\partial t \partial \mathbf{x}} = \frac{\partial^2 \mathbf{D}}{\partial \mathbf{x} \partial t}$ if the constraint equations in question have continuous 2nd derivatives[2]we can calculate the second derivative as:

$$\begin{aligned}\frac{\partial^2 \mathbf{D}}{\partial t^2} &= \frac{\partial}{\partial t}\left(\frac{\partial \mathbf{D}}{\partial \mathbf{x}}\dot{\mathbf{x}}\right)\\ &= \frac{\partial \mathbf{D}}{\partial \mathbf{x}}\frac{\partial \dot{\mathbf{x}}}{\partial t} + \frac{\partial^2 \mathbf{D}}{\partial t \partial \mathbf{x}}\dot{\mathbf{x}}\\ &= \frac{\partial \mathbf{D}}{\partial \mathbf{x}}\ddot{\mathbf{x}} + \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial \mathbf{D}}{\partial t}\right)\dot{\mathbf{x}}\\ &= \frac{\partial \mathbf{D}}{\partial \mathbf{x}}\ddot{\mathbf{x}} + \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial \mathbf{D}}{\partial \mathbf{x}}\dot{\mathbf{x}}\right)\dot{\mathbf{x}} = 0 \qquad (5.3)\end{aligned}$$

Isolating the terms containing $\ddot{\mathbf{x}}$ results in the following set of k equations which represent constraints on the accelerations:

$$\frac{\partial \mathbf{D}}{\partial \mathbf{x}}\ddot{\mathbf{x}} = -\left(\frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial \mathbf{D}}{\partial \mathbf{x}}\dot{\mathbf{x}}\right)\dot{\mathbf{x}}\right) \qquad (5.4)$$

Combining equation 5.2 with equation A.4 results in the following system of $n+k$ variables in $n+k$ equations for the equations of motion:

$$\begin{bmatrix}\mathbf{M} & \mathbf{J}^T\\ \mathbf{J} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{x}}\\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\sum \mathbf{f}\\ -\frac{\partial(\mathbf{J}\dot{\mathbf{x}})}{\partial \mathbf{x}}\dot{\mathbf{x}}\end{bmatrix} \qquad (5.5)$$

Where for simplicity $\mathbf{J} = \frac{\partial \mathbf{D}}{\partial \mathbf{x}}$ is defined as the Jacobian of the constraints. In order to solve this equation numerically we can recast the equation as a first order D.E. by setting $\mathbf{v} = \dot{\mathbf{x}}$ resulting in the following first order algebraic differential equation:

$$\begin{bmatrix}\dot{\mathbf{x}}\\ \dot{\mathbf{v}}\\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \begin{bmatrix}\mathbf{M} & \mathbf{J}^T\\ \mathbf{J} & \mathbf{0}\end{bmatrix}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{v}\\ \sum \mathbf{f}\\ -\frac{\partial(\mathbf{J}\mathbf{v})}{\partial \mathbf{x}}\mathbf{v}\end{bmatrix} \qquad (5.6)$$

We can reduce the form of this equation to significantly increase the computational efficiency: note that the equation of motion given by equation 5.5 has a block structure where $\mathbf{M}$ specifically is *not* time dependent, as the mass is assumed constant throughout. As a consequence it follows that $\mathbf{M}^{-1}$ can be precomputed, saving valuable computation time. In order to achieve

[2]This is a consequence of not allowing collisions, a collision constraint does have a discontinuity and special care has to be taken in that case.

this we can solve the algebraic part of the algebraic differential equation by decomposing the blockmatrix inversion into inversions of the submatrices to obtain explicit equations for $\boldsymbol{\lambda}$ and $\ddot{\mathbf{x}}$.

By applying a LDU decomposition on the left hand side matrix, and noting that $\mathbf{M}$ is invertible we obtain the following:

$$\begin{bmatrix}\mathbf{M} & \mathbf{J}^T \\ \mathbf{J} & \mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathbf{I} & 0 \\ \mathbf{J}\mathbf{M}^{-1} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{M} & 0 \\ 0 & -\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{M}^{-1}\mathbf{J}^T \\ 0 & \mathbf{I}\end{bmatrix} \tag{5.7}$$

This decomposition can easily be inverted to solve equation 5.5 in terms of the inverse of the Jacobian and mass matrices:

$$\begin{bmatrix}\ddot{\mathbf{x}} \\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\mathbf{I} & -\mathbf{M}^{-1}\mathbf{J}^T \\ 0 & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{M}^{-1} & 0 \\ 0 & -\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{I} & 0 \\ -\mathbf{J}\mathbf{M}^{-1} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\sum \mathbf{f} \\ -\frac{\partial(\mathbf{J}\dot{\mathbf{x}})}{\partial \mathbf{x}}\dot{\mathbf{x}}\end{bmatrix} \tag{5.8}$$

Writing out the terms results in the following set of equations:

$$\begin{bmatrix}\ddot{\mathbf{x}} \\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{J}^T\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\mathbf{J}\mathbf{M}^{-1} & \mathbf{M}^{-1}\mathbf{J}^T\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1} \\ \left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\mathbf{J}\mathbf{M}^{-1} & \left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\end{bmatrix}\begin{bmatrix}\sum \mathbf{f} \\ -\frac{\partial(\mathbf{J}\dot{\mathbf{x}})}{\partial \mathbf{x}}\dot{\mathbf{x}}\end{bmatrix} \tag{5.9}$$

The bottom row of which gives the independent solution for $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} = \left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\left(\mathbf{J}\mathbf{M}^{-1}\sum \mathbf{f} - \frac{\partial\left(\mathbf{J}\dot{\mathbf{x}}\right)}{\partial \mathbf{x}}\dot{\mathbf{x}}\right) \tag{5.10}$$

And the top row the independent solution for $\ddot{\mathbf{x}}$:

$$\begin{aligned}\ddot{\mathbf{x}} &= \mathbf{M}^{-1}\left(\sum \mathbf{f} - \mathbf{J}^T\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)^{-1}\left(\mathbf{J}\mathbf{M}^{-1}\sum \mathbf{f} - \frac{\partial\left(\mathbf{J}\dot{\mathbf{x}}\right)}{\partial \mathbf{x}}\dot{\mathbf{x}}\right)\right) \\ &= \mathbf{M}^{-1}\left(\sum \mathbf{f} - \mathbf{J}^T\boldsymbol{\lambda}\right)\end{aligned} \tag{5.11}$$

From these equations we can immediately discern some useful properties:

- $\mathbf{M}$ is time independent and diagonal, as a consequence so is $\mathbf{M}^{-1}$, this means it is only necessary to calculate it once for every mechanism simulation instead of for every time-step. Since $\mathbf{M}$ is by definition a nonsingular diagonal matrix the inverse is trivial to compute.
- Solving equation 5.11 really only requires solving equation 5.10 and applying a few matrix multiplications. The result is a set of equations of which the linear system to be solved with a costly matrix decomposition only has size n, in comparison to the complete $2k+n$ system when solving equation 5.6. As solving a linear system scales computationally as $\mathcal{O}\left(n^3\right)$ this represents a significant performance gain when running numerical simulations.
- The matrix $\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)$ is positive definite, square and symmetric by construction since the mass of an element can only ever be positive and larger than zero. This means a Cholesky decomposition is always possible and the symmetry of this matrix can be exploited to speed up computation further.

- For a solution to exist $\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)$ has to be nonsingular. This can be understood in simpler terms as the fact that the mechanism links should remain distinguishable at all times: identical links in identical positions become indistinguishable, they can be freely interchanged with no change to the system, causing a singularity in $\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right)$.
- It is the reaction forces that are solved numerically each time step. I.e. the simulation has the following structure: Algebraically solve for the reaction forces at $\mathbf{x}_n$, subtract these from the external forces $\mathbf{f}$ at $\mathbf{x}_n$, estimate $\mathbf{x}_{n+1}$ using the total force vector and trivial $\mathbf{M}^{-1}$ and integrating, then refine the estimate of $\mathbf{x}_{n+1}$ by solving $\mathbf{D}\left(\mathbf{x}_{n+1}\right) = 0$.

5-3-1 Assembling the equations of motion

In order to solve equation 5.11 for an arbitrary general mechanism we need to assemble $\mathbf{M}$, $\mathbf{D}$, $\mathbf{J}$, $\sum \mathbf{f}$ and $\frac{\partial J(\boldsymbol{D})\dot{\mathbf{x}}}{\partial \mathbf{x}}\dot{\mathbf{x}}$ automatically from the mechanism representation given by chapter 2-3.

We can directly determine from the Newton-Euler equations that the overall state of the system can be defined in terms of the states of the K separate links $\mathbf{x}_i = [x_i, y_i, \theta_i]^T$ as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_K \end{bmatrix} \tag{5.12}$$

For link L_i with state $\mathbf{x}_i$ the mass matrix and it's inverse are given by:

$$\mathbf{M}_i = \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & I_i \end{bmatrix}, \quad \mathbf{M}_i^{-1} = \begin{bmatrix} 1/m_i & 0 & 0 \\ 0 & 1/m_i & 0 \\ 0 & 0 & 1/I_i \end{bmatrix} \tag{5.13}$$

The complete mass matrix and inverse mass matrix of the mechanism can be obtained by placing the mass matrices of each of the K links present in the mechanism along the diagonal:

$$\mathbf{M} = \begin{bmatrix} M_1 & 0 & 0 & 0 \\ 0 & M_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & M_K \end{bmatrix}, \quad \mathbf{M}^{-1} = \begin{bmatrix} M_1^{-1} & 0 & 0 & 0 \\ 0 & M_2^{-1} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & M_K^{-1} \end{bmatrix} \tag{5.14}$$

The force vector can easily be constructed by elementwise concatenation of the forces affecting each of the K links:

$$\sum \mathbf{f} = \begin{bmatrix} \sum \mathbf{f}_1 \\ \sum \mathbf{f}_2 \\ \vdots \\ \sum \mathbf{f}_K \end{bmatrix} \tag{5.15}$$

While we don't know what the full state looks like as the mechanism structure is variable, we can already define the constraint equation for the n^{th} constraint inducing edge element as a function of the states of the two connected links:

$$\mathbf{D}_n\left(\mathbf{x}\right) = \mathbf{D}_n\left(\mathbf{x}_i, \mathbf{x}_j\right) = 0 \tag{5.16}$$

Every constraint equation has to be equal to zero, consequently we find the full set of constraint equations by concatenation:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_N \end{bmatrix} = 0 \tag{5.17}$$

We can also immediately calculate two derivatives for this constraint, the ones in terms of the states of the connected elements. These are given by:

$$\mathbf{J}_n^i = \frac{\partial \mathbf{D}_n}{\partial \mathbf{x}_i}, \quad \mathbf{J}_n^i = \frac{\partial \mathbf{D}_n}{\partial \mathbf{x}_j} \tag{5.18}$$

because $\mathbf{D}_n$ is by definition only a function of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ it follows that $\frac{\partial \mathbf{D}_n}{\partial \mathbf{x}_k} = \mathbf{0}$ for $k \neq i, j$. Using this relation the actual Jacobian for the nth constraint for a system with K states can be obtained from the two relations of equation 5.18 as:

$$\mathbf{J}_n = \begin{bmatrix} \mathbf{0}_{1..i-1} & \mathbf{J}_n^i & \mathbf{0}_{i+1...j-1} & \mathbf{J}_n^j & \mathbf{0}_{j+1...K} \end{bmatrix} \tag{5.19}$$

Where $\mathbf{J}_i^n$ is the part of the Jacobian corresponding to $\mathbf{x}_i$ and $\mathbf{J}_n^j$ corresponding to $\mathbf{x}_j$, $\mathbf{0}_{i...j}$ represents a zero matrix of size 2 by $3\,(i-j)$. The full Jacobian of the entire system with N constraints and K states can then be obtained by the concatenation of each of the N Jacobians given by equation 5.19:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \vdots \\ \mathbf{J}_N \end{bmatrix} \tag{5.20}$$

Finally we need to assemble the Coriolis terms $\frac{\partial (\mathbf{J}\dot{\mathbf{x}})}{\partial \mathbf{x}}\dot{\mathbf{x}}$. Applying 5.19 results in:

$$\frac{\partial\left(\mathbf{J}\dot{\mathbf{x}}\right)}{\partial \mathbf{x}}\dot{\mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}\left(\begin{bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \vdots \\ \mathbf{J}_N \end{bmatrix}\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \vdots \\ \dot{\mathbf{x}}_K \end{bmatrix}\right)\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \vdots \\ \dot{\mathbf{x}}_K \end{bmatrix} = \frac{\partial}{\partial \mathbf{x}}\left(\begin{bmatrix} \mathbf{J}_1^i\dot{\mathbf{x}}_{i1} + \mathbf{J}_1^j\dot{\mathbf{x}}_{j1} \\ \mathbf{J}_2^i\dot{\mathbf{x}}_{i2} + \mathbf{J}_2^j\dot{\mathbf{x}}_{j2} \\ \vdots \\ \mathbf{J}_N^i\dot{\mathbf{x}}_{iN} + \mathbf{J}_N^j\dot{\mathbf{x}}_{jN} \end{bmatrix}\right)\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \vdots \\ \dot{\mathbf{x}}_K \end{bmatrix} \tag{5.21}$$

Evidently only the states of the connected elements have a nonzero contribution to $\mathbf{J}_n\dot{\mathbf{x}}$. Additionally, for the nth constraint it follows from equation 5.16 that:

$$\frac{\partial \mathbf{J}_n\dot{\mathbf{x}}}{\partial \mathbf{x}_k} = \mathbf{0} \text{ for } k \neq i, j \tag{5.22}$$

By combining this property with equation 5.21 we obtain a way to assemble the Coriolis term for the entire system as a concatenation of terms belonging to the separate constraints:

$$\frac{\partial\left(\mathbf{J}\dot{\mathbf{x}}\right)}{\partial \mathbf{x}}\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \vdots \\ \mathbf{C}_N \end{bmatrix} \tag{5.23}$$

Where:

$$\mathbf{C}_n = \frac{\partial \left(\mathbf{J}_n \dot{\mathbf{x}}_{i,j}\right)}{\partial \mathbf{x}_{i,j}} \dot{\mathbf{x}}_{i,j} \tag{5.24}$$

In conclusion: we can assemble the equations of motion for the complete mechanism in an elementwise fashion, if, for each element, we have at least the following definitions available:

For each *link:*

- The equation for the moment of inertia around the center of mass of the link.

For each *constraint inducing* element:

- The constraint (equation 5.16).
- The Jacobian of the constraint w.r.t. the states of the connected masses (equation 5.19).
- The Coriolis terms belonging to the constraint, in terms of the state of the connected elements.

For each *force inducing* element:

- The force as a function of the state of the connected elements.

An example that shows how to define these equations for a rotational hinge joint which connects two masses is given in appendix A-1. The moment of inertia for a rod is given by $\frac{mL^2}{12}$, the inertia equation for more complex polygonal elements is derived in appendix A-2.

5-3-2 Numerical solver

The equations of motion given by equation 5.11 are solved for each timestep by using a Cholesky decomposition implemented by the Eigen library [64]. This particular method was chosen because it is computationally efficient, while more accurate than iterative solvers.

The result is integrated using a *fixed-step explicit* Runge-Kutta 4 method. This method is chosen since it has much better stability and accuracy properties than for example fixed step Euler integration at identical computational cost [65], while requiring no implicit solution for $\mathbf{v}_{n+1}$. A fixed time step makes mechanism trajectories easily comparable; the scale of each simulation is identical. The method is given by (for a fixed step size h, state vector $\mathbf{x}$, time t and first order O.D.E. $\dot{\mathbf{x}} = f(t, \mathbf{x})$):

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \\ t_{n+1} &= t_n + h \end{aligned} \tag{5.25}$$

where:

$$\begin{aligned} \mathbf{k}_1 &= f\left(t_n, \mathbf{x}_n\right) \\ \mathbf{k}_2 &= f\left(t_n + \frac{h}{2}, \mathbf{x}_n + \frac{\mathbf{k}_1}{2}\right) \\ \mathbf{k}_3 &= f\left(t_n + \frac{h}{2}, \mathbf{x}_n + \frac{\mathbf{k}_2}{2}\right) \\ \mathbf{k}_4 &= f\left(t_n + h, \mathbf{x}_n + \mathbf{k}_3\right) \end{aligned} \tag{5.26}$$

After the solution is obtained the error on the constraints is checked using equation 5.17, if the error is too large as defined by the desired precision ε the state is corrected by taking a step of Newton's method to find $\hat{\mathbf{x}}_{n+1} \approx \mathbf{x}_{n+1}$ such that $\mathbf{D}\left(\hat{\mathbf{x}}_{n+1}\right) \leq \varepsilon$:

$$\hat{\mathbf{x}}_{n+1} = \mathbf{x}_{n+1} - \mathbf{J}^{+}\mathbf{D}\left(\mathbf{x}_{n+1}\right) \tag{5.27}$$

Where the pseudoinverse $\mathbf{J}^{+} = \mathbf{J}^{T}\left(\mathbf{J}\mathbf{J}^{T}\right)^{-1}$ is used, as the Jacobian of the constraints is not invertible.

5-3-3 Numerical stability

$\mathbf{J}^{+}$ exists only if the rows of $\mathbf{J}$ are independent [66]. This implies that the constraints on the velocities must be independent. When $\mathbf{J}\mathbf{J}^{T}$ becomes (nearly) singular the solver becomes unstable and the mechanism simulation stops and sets a failure flag. Note that increasing the step size generally resolves this issue unless it is a true singular state of the system, which can happen in mechanisms exhibiting exact symmetry.

A practical result of explicit integration is that the simulator is not particularly suitable for stiff mechanisms which result in a stiff differential equation [65]. A simple temporary workaround for stiff mechanisms is to decrease the step size if the simulation fails. Creating a more stable numerical integrator based on an iterative solver method to perform implicit integration would solve the stability problems for stiff mechanisms but such a solver is non-trivial.

It is also important to mention that the solver is not optimized for sparse matrices, which appear when building very large mechanisms.

However, both very large and stiff mechanisms are not considered to be suitable for the task goals discussed in this thesis and as such solving these edge cases is not part of the current simulator design goals. I invite anyone with the interest reading this to implement a sparse and/or implicit physics solver.

5-4 Additional functionality

Not every mechanism in the mechanism space is functional, some will fall apart immediately because they lack enough connections, some will fall into infinity because they lack a ground connection, others cannot move. We can significantly reduce computation by discarding these without having to do a costly evaluation by simulation them. the library applies these checks on every mechanism that is constructed.

5-4-1 Sanity checks

Dna objects are checked for validity on construction. If the parameters, labels and incidence matrix do not match up or are simply invalid an exception is raised. The same is done for operator objects.

If negative parameters are provided for a parameter that can only be positive, such as the spring constant of a spring, the constructor will use the absolute value of that parameter by default. It is also possible to disable this behavior and raise an exception instead.

To prevent the algorithm spending costly evaluations on mechanisms which can be deemed invalid each mechanism can be checked for the following properties on construction

- **Grounded:** Returns true if every element of the mechanism is connected with the ground in a constrained way. It checks if the graph with spring edges removed is fully connected. This prevents mechanisms with dangling or floating links.
- **Connected:** Returns true if every element of the mechanism is connected, excluding the ground. This prevents mechanisms that consist of many different separate parts that do not connect. It checks if the graph with ground node removed is fully connected.
- **Dynamic:** It is checked if the resulting mechanism has at least a single degree of freedom available.

Trying to generate a mechanism that does not satisfy these three requirements will raise an exception. These checks are enabled by default but can be disabled so it is still possible to create such mechanisms if desired.

Other general warnings and exceptions with helpful feedback will usually[3] pop up if a user will try to do something that is not possible. For example when trying to animate a mechanism without simulating it first or trying to assign a controller to a mechanism with no motors.

5-4-2 API

A method is provided to convert dna objects of the mechanism library to graph-tool [67] graph objects as well as networkx [68] graph objects. These external Python libraries can then be used to determine useful graph properties such as isomorphism and edit distance. The dna graphs can also be plotted using the graph-tool toolbox. The full possibilities of these external libraries are given by their respective documentations.

5-4-3 Storage

Methods are provided to store and read mechanisms, operators, and evolutionary space graphs into json files on disk. This lets users easily store and load interesting mechanisms they or the algorithm discovered.

5-5 Program flow

A script using the library can generally be broken down into two different types

- Construction and simulation of singular actuated mechanisms.
- The evolutionary algorithm, which uses the simulations and manages entire populations of mechanisms.

Note that the evolutionary algorithm embeds the mechanisms.

The modular structure of the code is visualized in the schematic shown in Figure 5.3. A mechanism is constructed once and many of it's properties are precomputed, this logic is only processed once. This provides a big benefit over the previous implementations, which required many comparisons and memory allocations every run to rebuild the equations of motion equations for each time step of the simulation.

[3]Not every possible issue is guaranteed to be handled, .

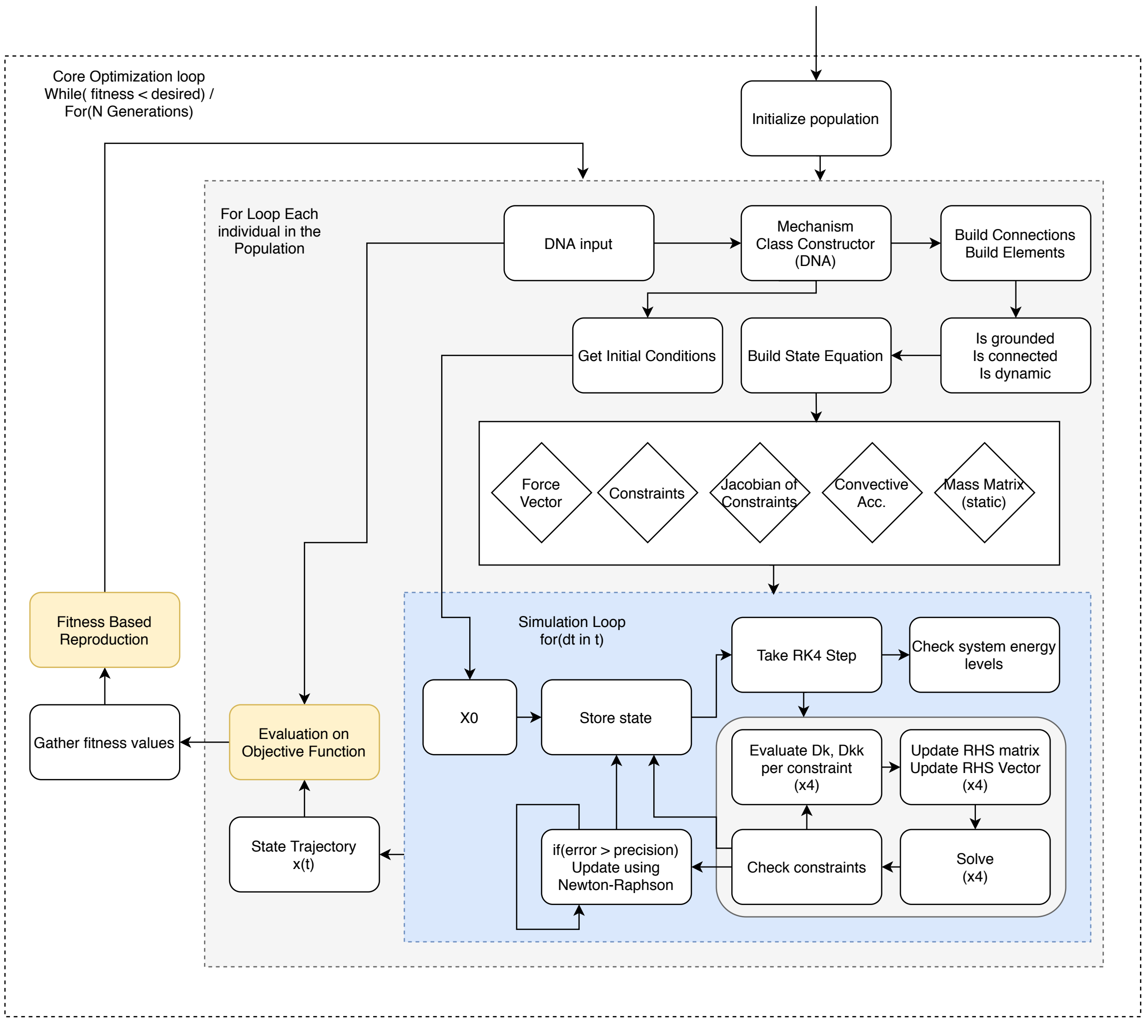
Core Optimization loop
While(fitness < desired) /
For(N Generations)
Initialize population
For Loop Each individual in the Population
DNA input
Mechanism Class Constructor (DNA)
Build Connections Build Elements
Get Initial Conditions
Build State Equation
Is grounded Is connected Is dynamic
Force Vector
Constraints
Jacobian of Constraints
Convective Acc.
Mass Matrix (static)
Simulation Loop for(dt in t)
Fitness Based Reproduction
Take RK4 Step
Check system energy levels
X0
Store state
Gather fitness values
Evaluation on Objective Function
Evaluate Dk, Dkk per constraint (x4)
Update RHS matrix Update RHS Vector (x4)
State Trajectory x(t)
if(error > precision) Update using Newton-Raphson
Check constraints
Solve (x4)

Figure 5.3: Schematic of the program structure of the simulation of a mechanism.

5-6 A few notes on computational optimization

Because the library had to be particularly fast for relatively small mechanisms it has been optimized using extensive profiling.

Some of the used practices include:

- Preallocation of memory, only relevant dynamic elements of matrices and vectors are updated. In some cases only these elements are stored.
- Precomputation of sines and cosines of the state that are reused by different elements
- Minimization of copy operations and comparisons
- Minimization the amount the Jacobians and constraints have to be calculated

5-7 Conclusion

The goal was to design and build a modular library suitable for the evolution, simulation and analysis of planar mechanisms. Both functionality and performance has been adequately determined using testing and profiling. Python bindings are made available to increase the ease of use and a set of example scripts is included with the package. . The full library is available as a python wheel package and source code is available at [gitlab link] or on request from the author of this thesis.

Chapter 6

Results

6-1 Algorithm and simulator evaluation

6-1-1 Objectives

It is shown in Chapter 4 that the mechanisms can also be represented by a series of operators and mapped onto a metric space. A novel quality diversity evolutionary algorithm based on abstract operators is derived from the structure of this mechanism space: ROSE. This algorithm is used to computationally design mechanisms for a pick-and-place task.

In Chapter 5 the necessary software to computationally explore the mechanisms and search space was described and developed. High performance was required and this is put to the test in a simulation performance benchmark in Section 6-2.

ROSE provides meta-heuristics to analyze the algorithm run, these are visualized and analyzed in Section 6-3. since ROSE can be classified as a Quality Diversity algorithm [35] the properties of the resulting solution space are analyzed in Section 6-3-1.

Finally the ROSE algorithm is also compared to previous iterations of automated mechanism design algorithms in Section 6-4.

These questions will be treated separately using the example application of a 2D pick-and-place problem.

6-1-2 Methods

6-1-2-1 Mechanism simulation benchmark

The double pendulum shown in Figure 6.1 is simulated for 10 seconds in 200 timesteps using a naive Python implementation of the equations of motion that uses Numpy [69] and then compared to both multi- and single-threaded simulation of that same double pendulum by the mechanism library presented alongside this thesis.

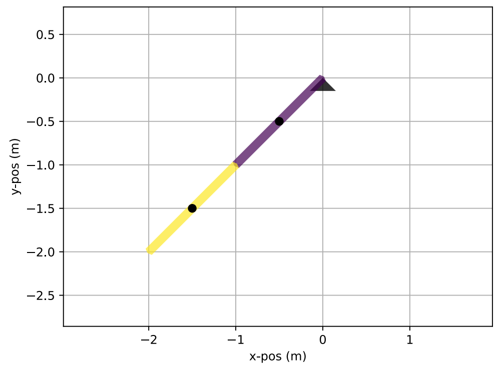

Figure 6.1: The double pendulum without end-effector that is used to generate the profile results, shown here in it's initial position.

6-1-2-2 Pick-and-place task

The automated design task is designed around a basic pick and place task, a common and representative practical robotics application [32]. Because the algorithm could otherwise always find simple single pendulum solutions by moving the ground the original ground connection is kept fixed at $\begin{bmatrix}0 & 0\end{bmatrix}$. This should result in more interesting solutions as a more complex mechanism is required to satisfy the task.

The pick and place task is visualized in Figure 6.2. The task itself is time dependent resulting in a dynamic objective: the optimal solution would be a stable limit cycle. This is of particular interest as the generated mechanisms should not only have to have the right kinematics, but also tune the parameters such that their periods match the specified task time exactly.

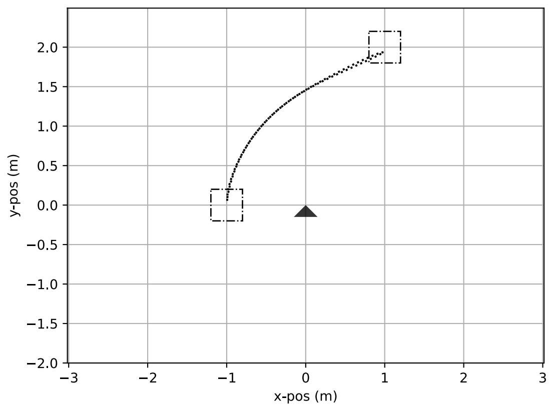

Figure 6.2: Visualization of the task, the positions where the boxes should be picked up on the top and put down at the bottom are represented by the striped-dotted boxes. The fixed ground at $\begin{bmatrix}0 & 0\end{bmatrix}$ is also shown. An example end-effector trajectory for the full cycle is illustrated as the dotted path.

The position error is defined by a time-based closed-loop trajectory reference based on a Poincaré map; be at place $\mathbf{x}_r$ with velocity zero at time $\mathrm{d}t_r$, at the end of the simulation the mechanism should be back at the starting position:

$$E_r = \sum_n \|\mathbf{x}(t_e) - \mathbf{x}_r\|^2 - \|\dot{\mathbf{x}}(t_e)\|^2 \tag{6.1}$$

A complexity penalty is applied to pressure towards simpler mechanisms, this has the effect of the algorithm reducing redundant elements:

$$E_c = c(M) \tag{6.2}$$

While the mechanism search space is technically unbounded (see chapter 4), the mechanism complexity fitness puts soft limits on the parts of the mechanism space that are explored. This soft cap helps in practice to avoid the unbounded novelty trap [36].

A total distance traveled penalty is also used, which is the total traveled distance minus the minimum distance required for the task to be completed.

$$E_d = \left\| \int_0^{t_e} \mathbf{x}(t)\, dt - L_0 \right\|^1 \tag{6.3}$$

A side-effect is that the average velocity of the mechanism is also soft capped, as a large average velocity will lead to a high distance traveled for the end-effector. An optimal mechanism on this task will have a closed end-effector trajectory such that $\mathbf{x}(0) = \mathbf{x}(t_e)$ and $\dot{\mathbf{x}}(0) = \dot{\mathbf{x}}(t_e) = \mathbf{0}$, which highly increases the likelihood of this mechanism trajectory being quasi-stable as well. Although there is no stability guarantee beyond t_e because only the end-effector trajectory is tracked, not the entire mechanism trajectory.

The final objective function is a weighted rms sum of these values, the weights are applied such that they scale the fitness values to the same degree of contribution, otherwise one fitness contribution will dominate over the others. As each of the contributions define an error the negative is taken as the fitness value. This results in the total fitness equation to be maximized:

$$E = -\sqrt{(\lambda_r E_r)^2 + (\lambda_c E_c)^2 + (\lambda_d E_d)^2} \tag{6.4}$$

Scaling weights were heuristically determined from test runs as $\lambda_r = 1$, $\lambda_r = 0.1$ and $\lambda_r = 0.03$ resulting in an approximately spherical distribution. These weights are used in all further tests except where specifically noted.

6-1-2-3 Algorithm seed

Every algorithm run uses the same seed mechanism: the single pendulum grabber mechanism shown in Figure 6.3. In addition every run uses the same parameters as well as the same random number generator seed 0. This provides consistency across runs and platforms and makes it easier to reproduce the results using the scripts accompanying this thesis which are also provided as examples in the mechanism library.

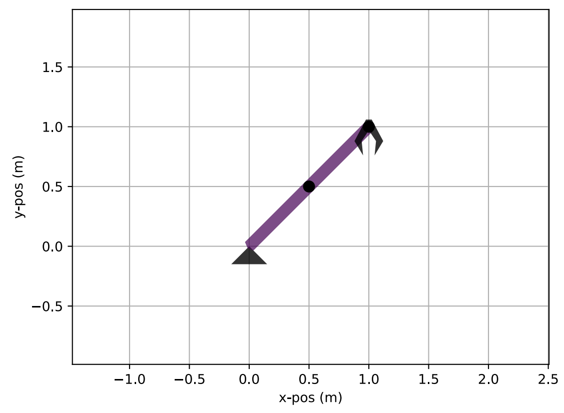

Figure 6.3: The seed mechanism M_s for all tests is a single pendulum with end-effector at $\begin{bmatrix}1 & 1\end{bmatrix}$ and a fixed ground connection at $\begin{bmatrix}0 & 0\end{bmatrix}$ which is initially a hinge H_1.

6-1-2-4 ROSE operator set

Compounded operators were defined for use with ROSE. The compound operators are combinations of the fundamental base operators; an abstracted operator set Φ with corresponding domain Ω^{Φ} as described in section 4-3-3. These operators and their generation function provide the core of the evolutionary algorithm. The following combinations were used:

Add Link

Adds a mass, but also always adds a hinge connection for that mass to another non-ground link. Preserves connectivity. It is defined on the base operator set as:

$$\varphi_L^+ = \varphi_p^5 \circ \varphi_e^+ \circ \varphi_v^+ \tag{6.5}$$

Remove Link

Only removes a link if it has no more than one constraint connection which prevents removal of middle out links. It is defined on the base operator set as:

$$\varphi_L^- = \varphi_p^5 \circ \varphi_e^- \circ \varphi_v^- \tag{6.6}$$

Add Connection

Adds a connection with label and appropriate parameters. Never add a connection such that the mechanism becomes fully constrained. Additional ground connections are only possible as springs as these are non-constrained. It is defined on the base operator set as:

$$\varphi_C^+ = \varphi_p^n \circ \varphi_e^+ \tag{6.7}$$

Remove Connection

Only removes connections if doing so keeps the mechanism grounded and connected. It is defined on the base operator set as:

$$\varphi_C^- = \varphi_p^n \circ \varphi_e^- \tag{6.8}$$

Where n is the number of parameters of the connection.

Relabel

When relabeling, preserves the connection location. It is defined on the base operator set as:

$$\varphi_R = \varphi_p^{n-2} \circ \varphi_r \tag{6.9}$$

Where n is again the number of parameters of the connection.

Transform

Transformation operator, that scales, rotates and moves the mechanism as a whole. It acts on the positions of all elements. It is defined on the base operator set as:

$$\varphi_T = \bigodot_M \varphi_p^2 \tag{6.10}$$

Where the composition $\bigodot$ is over M, all the elements of the mechanism, in any order.

Move End-Effector

Operator that specifically moves the end-effector to another link while preserving it's position. It is defined on the base operator set as:

$$\varphi_E = \varphi_e^+ \circ \varphi_e^- \tag{6.11}$$

Mutate Parameters

Mutates the parameters of an element all at once by adding or subtracting the existing parameters with new random values. Every mutation operator there is a 50% chance another mutation is created, modifying the mechanism further. It is defined on the base operator set as:

$$\varphi_M = \varphi_p^k \tag{6.12}$$

Where k is a random number from a geometric distribution.

6-2 Simulator performance

First the simulator is put to the test. Due to the high required number of simulations performance was of the essence when developing the simulator. To illustrate the performance gained by the library a comparison test has been set up in Python. All the timings for the tests in this section and the following sections are generated on an Intel core i5 9300H CPU with 16Gb DDR4 RAM. The CPU has 4 cores and 8 logical processors using hyper-threading, resulting in 8 effectively usable threads by the library.

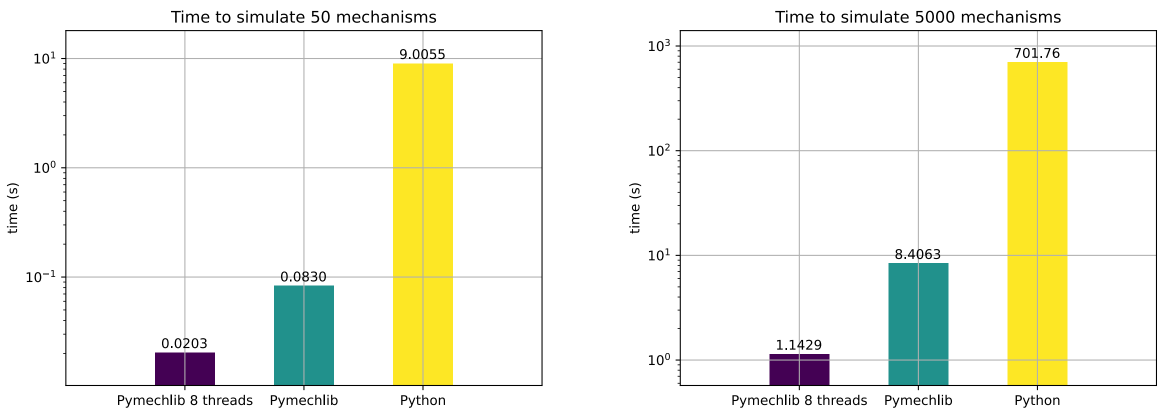

Figure 6.4: Simulation timing results averaged over 10 runs. Using the multi-thread implementation PyMechs is approximately 600 times as fast on simulation than the Python implementation when simulating 5000 mechanisms.

There is a clear difference in performance, the mechanism library PyMechs is on the order of 450 times faster in the case of 50 mechanisms and over 600 times as fast in the case of 5000 mechanisms. The PyMechs implementation manages 5000 simulations in less time than the Python implementation needs for 50 simulations.

Using more cores beats single threaded simulation by just over a factor 4 in the case of 50 mechanisms. However when simulating 5000 mechanisms the result is in an increase in performance much closer to the theoretically optimal factor of 8. As the amount of simulations goes up the time per mechanism goes down. This effect is caused by the inherent overhead of the Python interface.

6-3 ROSE meta-heuristics

A run of the ROSE algorithm for the 2D pick-and-place problem results in a rooted graph structure as described in section 4-3. The exploration of the projection presented in section 4-3-3 can be visualized and analyzed because of the close integration with graph-tool [67]. This structure where the vertices represent mechanism structures and the edges represent operators is shown in Figure 6.5. The red vertex is the seed mechanism as shown in Figure 6.3.

A first glance shows that many different paths between mechanisms are present, the graph is densely interconnected as can be expected from the quite general operator base that is used.

The shortest paths to the 40 best performing mechanisms are represented as bold edges in the image. These paths are the corresponding minimal representational operators $\hat{\varphi}_M$ of these mechanisms as represented on the base operator set Φ.

Of note is the presence of loops, some operators connect a node to itself. This is the result of symmetric mechanism structures where for example moving the end-effector to another link actually results in an isomorphic mechanism structure.

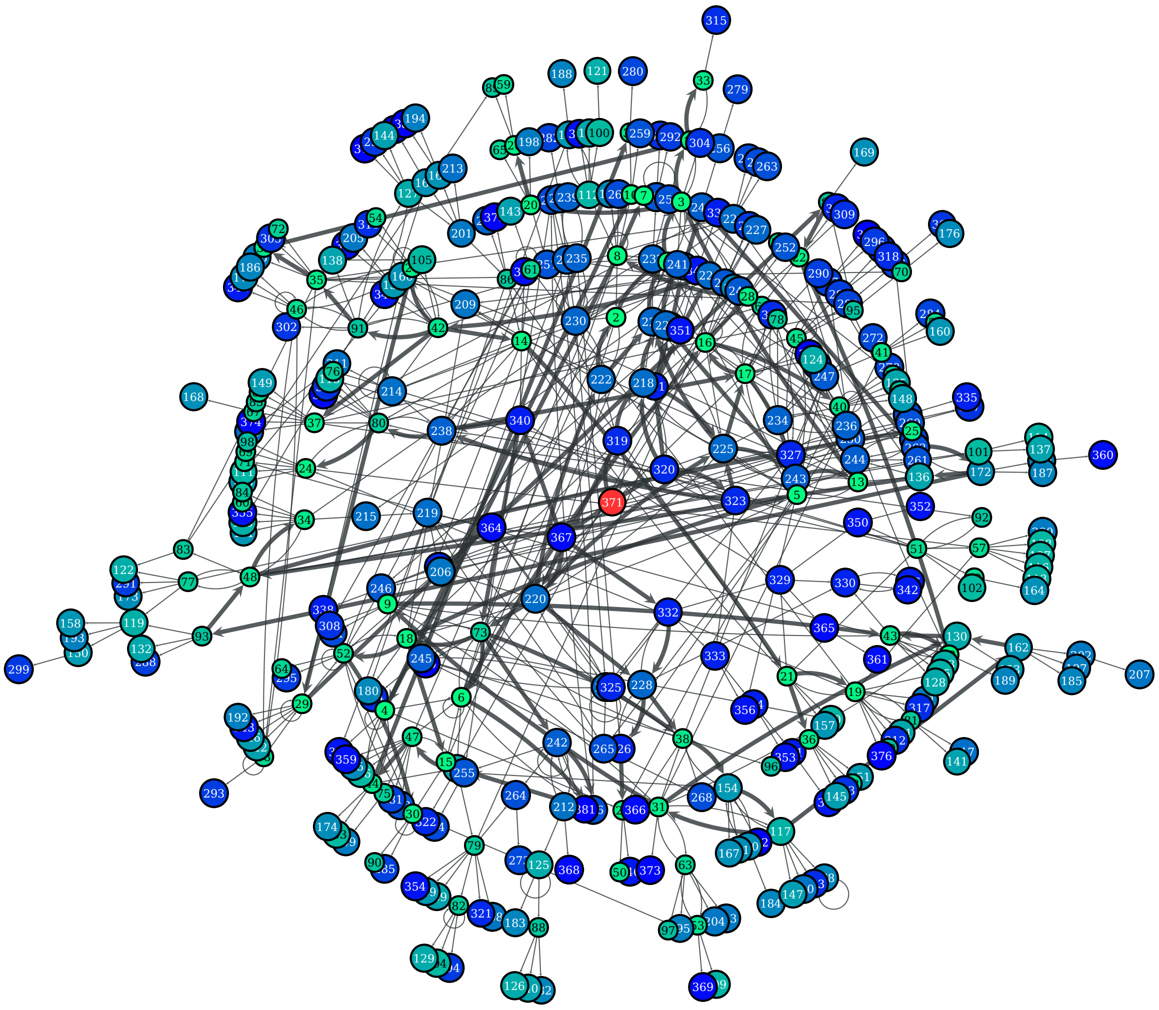

Figure 6.5: A visualization of the stochastic exploration of the domain Ω^Φ as generated by a ROSE run. Every node represents a unique mechanism structure and every arrow represents an operator from the generation set that is applied to that node, connecting it to neighboring nodes. The nodes are ranked by order of the fitness of their best member and sorted radially by their complexity from the seed mechanism. The original seed mechanism structure is denoted in red. Clearly the graph is densely interconnected. The minimum paths connecting the 40 best nodes $\hat{\varphi}_M$ are printed as bold lines.

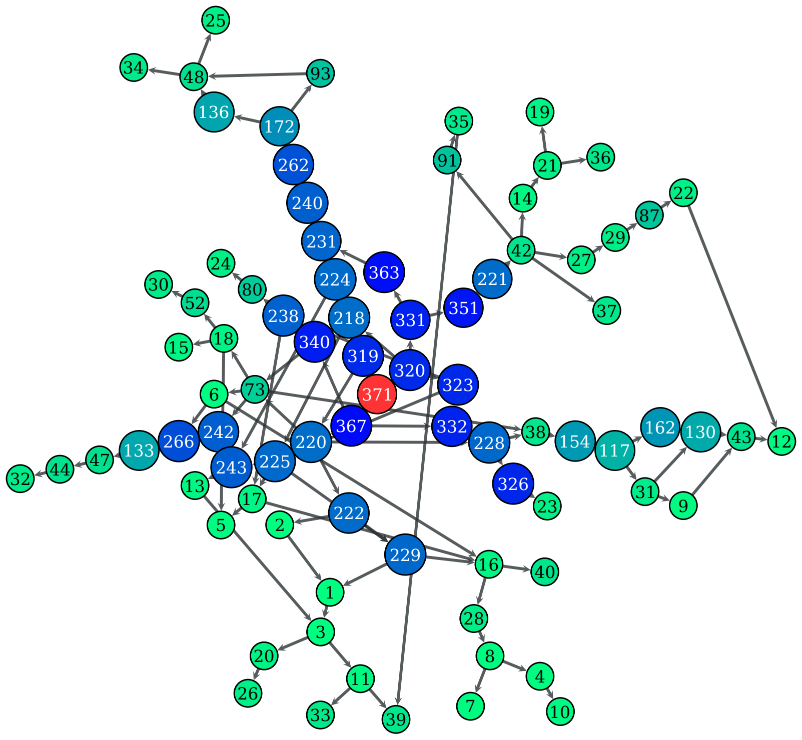

Figure 6.6: The same exploration of mechanism space but reduced to the minimum paths $\hat{\varphi}_M$ connecting the 40 most optimal mechanism structures (The bold lines from Figure 6.5).

Figure 6.6 shows the same exploration but projected onto the domain of the minimal representational operators of the 40 best performing mechanisms. A clear reduction of complexity occurs which suggests that the used operator base Φ can be refined further to one that admits a smaller domain without reducing the quality of the resultant solutions. This shows that the ideal mechanism structures for this problem are embedded within a significantly reduced subset $\Omega^* \subset \Omega^\Phi$ of the already reduced space Ω^Φ used by the ROSE algorithm. The clusters of good solutions visible in this graph represent groupings of highly similar solution strategies. It is also visible that some sections of the good solution subspace require the algorithm to pass through a significant amount of poorly ranked nodes, indicating the presence of deception.

The behavior of the fitness over time of the vertices is shown in Figure 6.7. While the maximum fitness eventually remains stagnant the average fitness of active vertices keeps fluctuating with an upward trend. This shows that the algorithm is still exploring other promising paths for possible breakthroughs.

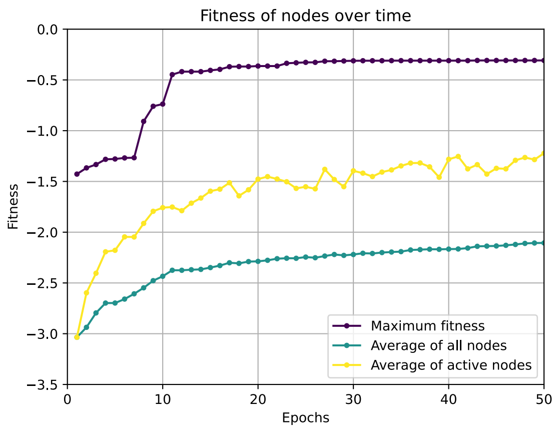

Figure 6.7: Fitness of the explored mechanism space vertices from Figure 6.5 over time.

6-3-1 Solution diversity

Due to the ROSE algorithm processing many prospective nodes at once and keeping the successful nodes active in the graph the end-result is not just a single optimal mechanism, but a listing of the best mechanism of every vertex. As each vertex represents a unique structural variation the result of an algorithm run is a diversity of possible solutions. *ROSE provides an estimate of the maximum fitness of every single vertex present in it's structure.* This represents a great diversity of solutions each with varying fitness values on the objective function given by equation 6.4, each vertex represents different combinations of properties such as complexity and position error.

Some of the resultant varying strategies are shown in Figure 6.8, representing the best mechanism for their particular structural vertex in mechanism space. Note the usage of a counterweight link to balance against the weight of the end-effector for the mechanisms with rank 6.

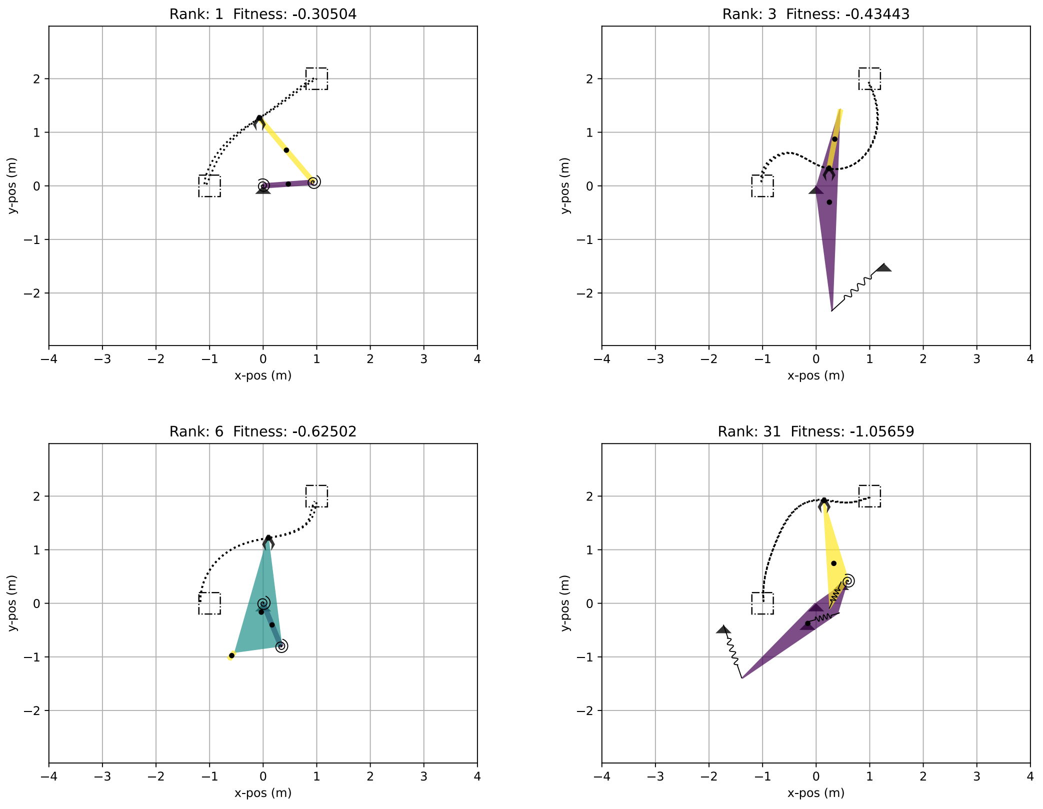

Figure 6.8: Some of the diverse strategies found by the ROSE 40 algorithm to solve the pick and place task. Ranked by their performance on the objective.

6-3-2 Multi-objective analysis

An effect of the high diversity of solution strategies is that we can analyze the resultant mechanism space graph structure as if it were a true multi-objective optimization result.

To study this further we can analyze the passive mechanism pick and place results from section 6-3-1. The fitness of each node separated out into complexity $\lambda_c E_c$ and position error $\lambda_r E_r$ is shown in Figure 6.9. Despite not actively solving as a multi-objective optimizer the algorithm still generates a front of solutions on the different variables of the objective function. Notably a significant amount of mechanisms satisfies the minimum requirement of overall fitness > -1. The log plot clearly indicates an inverse correlation between complexity and minimum position penalty: to achieve a smaller position penalty, the mechanisms has to grow in complexity and vice-versa.

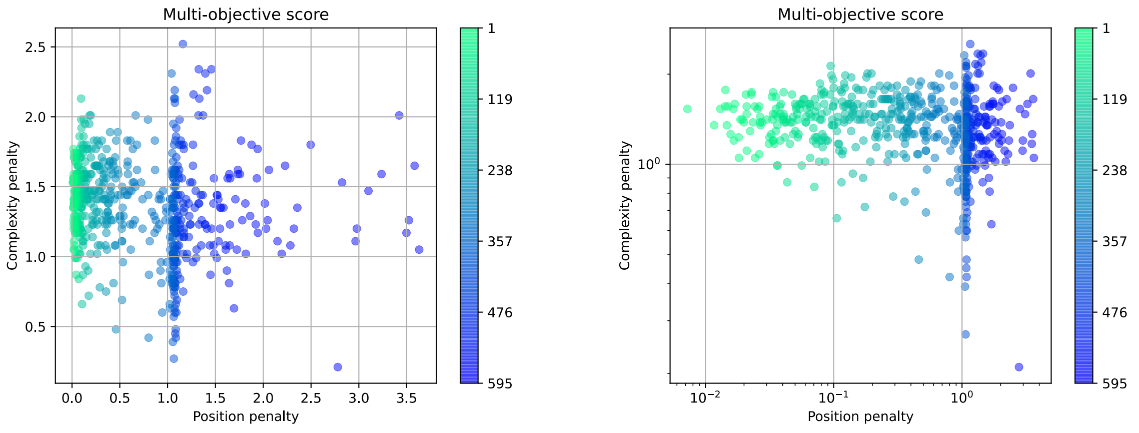

Figure 6.9: The complexity versus position penalty for the best mechanisms of the explored mechanisms space nodes plotted linearly (left) and plotted as log-log (right). The color of the node represents the mechanisms rank in the ROSE graph, with green being the best and blue being the worst.

Of particular note is the line-shaped cluster of solutions visible at a position penalty of approximately 1, which looks like a soft boundary on the fitness. Two mechanisms from this line are shown in Figure 6.11.

The line represents single-pendulum mechanism solutions, which have a theoretically optimal position error of exactly 1. This boundary represents a deceptive attractor in the search space, which the algorithm manages to bypass by constantly injecting more diversity in structural possibilities. This is also visible in the violin plot of Figure 6.13, the 10 island algorithm in particular tends to get stuck in this particular attractor.

The fitness of each node separated out into weighted excess distance traveled penalty $\lambda_d E_d$ and weighted position penalty $\lambda_r E_r$ is shown in Figure 6.10. The log-log plot shows there is quite a bit of variance still between the best performing mechanisms on these two penalties. It also indicates that in this case there is no clear trade-off. Contrary to the complexity and position trade-off it is possible to reduce the position penalty and traveled distance penalty simultaneously with the best performing mechanisms doing both.

Again we can identify visible clusters in this plot, but in this case it is not a line but two distinct clusters, the difference is explained by the difference between clockwise and anti-clockwise rotating single spring-pendulum mechanisms as shown in Figure 6.11. Contrary to Figure 6.9 there is no clear trade-off between these penalties. Instead, mechanisms with small position penalty also tend to have small traveled distance penalty.

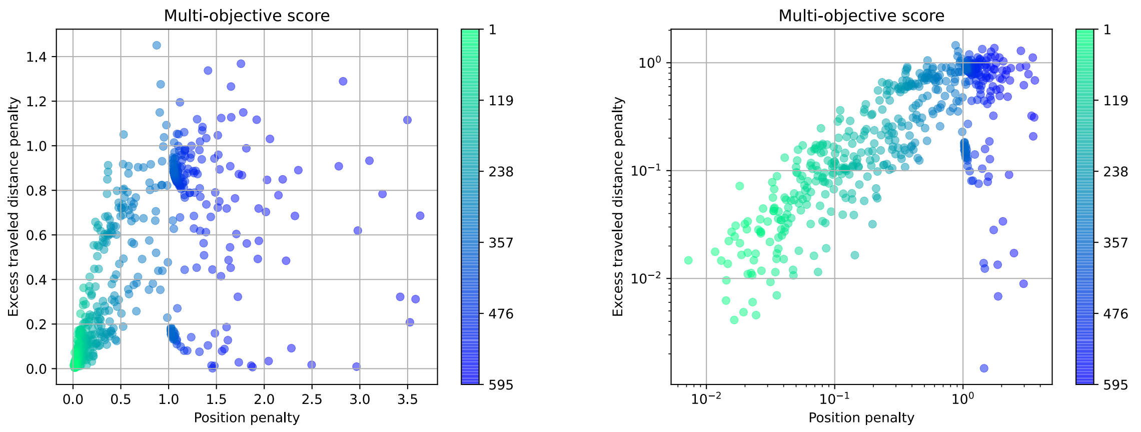

Figure 6.10: The traveled distance versus position penalty for the best mechanisms of the explored mechanisms space nodes plotted linearly (left) and plotted as log-log (right).

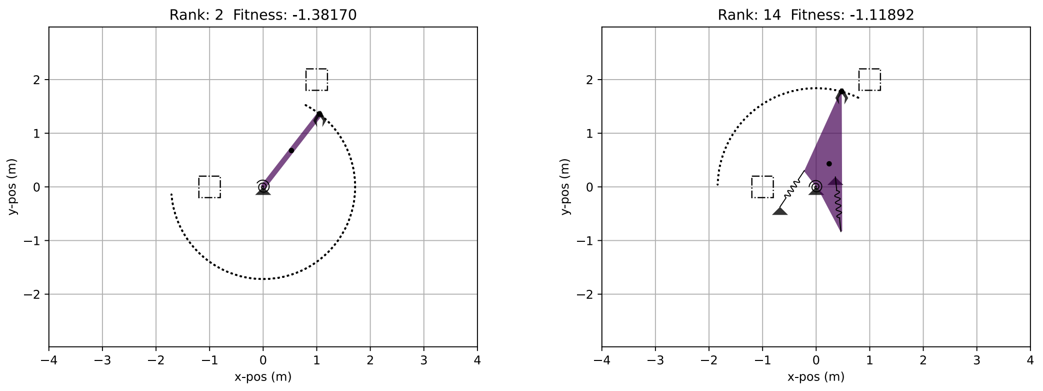

Figure 6.11: Two mechanism structures that have a position error of approximately 1. Both structures employ a single rotating pendulum structure for their end-effector. The right has better overall fitness as it manages to reduce the excess traveled distance error by rotating counter-clockwise rather than clockwise around the ground. Ranked on complexity penalty.

6-4 Comparison to prior art

The initial test is a comparison of the performance of operator guided evolution versus the performance of a more classic island model [70] approach as used previously to automatically design mechanisms [19]. This island model still fits within the operator approach: by specifying the operators accessible for local node optimization as the full set, rather than just the parametric operators, and by specifying an exchange operator which exchanges the best operators of the nodes as is done in the island model, a version of the island model is obtained within the framework of operator guided evolution. This version does not exploit any of the structure present in mechanism space however.

Finally a comparison is made with classic evolution with no islands or other diversity maintenance tricks. The PyMechs multi-thread implementation presented with this thesis is used to generate and analyze all results. The parameters displayed table 6.1 were used for the tests. The higher initial spread for the Island models is to account for the differences in initial population generation of those algorithms [70].

	ROSE 40	ROSE 10	Island 40	Island 10
Parameter	**Value**	**Value**	**Value**	**Value**
Epochs	50	50	50	50
Generations	32	32	32	32
Active leaves/Islands	40	10	40	10
Population	32	32	32	32
Initial spread	5	5	10	10

Table 6.1: The parameter values used for the algorithm comparison results.

First the ability to generate passive mechanisms is tested for these algorithms. The fitness vs the amount of used simulations is plotted in Figure 6.12. Comparing the median optimal fitness over 50 separate runs shows that the ROSE algorithm is much more likely to converge to an effective solution than either of the island model runs. The ROSE algorithm even achieves a 100% convergence rate on this test when using 40 active nodes.

Using statistical non-parametric mapping, a form of statistical parametric mapping [71] we can compare the resulting data over time. A non-parametric approach was used because the data is not normally distributed. The result is displayed as the bars on the top of Figure 6.12.

Even though for the initial epochs there is no significant ($p < 0.05$) difference between the 40 node ROSE algorithm and the Island model algorithm after approximately 10 epochs the ROSE algorithm gains a decisive lead over the Island algorithm. After 50 epochs the confidence of ROSE being better than Islands grows to $p < 0.001$. This clearly improved performance in the long run can be attributed to the ROSE algorithm continually injecting more diversity into its solution population making it much less likely to get trapped in local minima.

This effect is also qualitatively evident in the violin plot of the final fitness values shown in Figure 6.13, which illustrates the distribution of the maximum fitness found by the algorithm between different runs. There are clusters at lower fitness values visible for both Island algorithms and they have a larger variance of fitness endpoints because they get stuck in the deceptive attractor. Both ROSE versions have a higher median fitness and a tighter distribution than the Island algorithms, indicating that they converge to better solutions and do so more reliably.

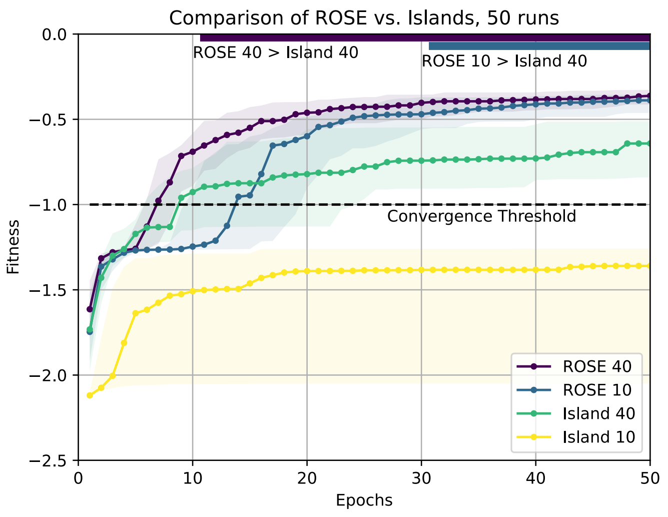

Figure 6.12: A comparison of the median optimal fitness obtained from 50 separate runs, the shaded area represents the range between the 25th and 75th percentile optimal solution respectively. The bars on top indicate $p < 0.05$ confidence intervals as obtained by applying a pairwise SnPM t-test.

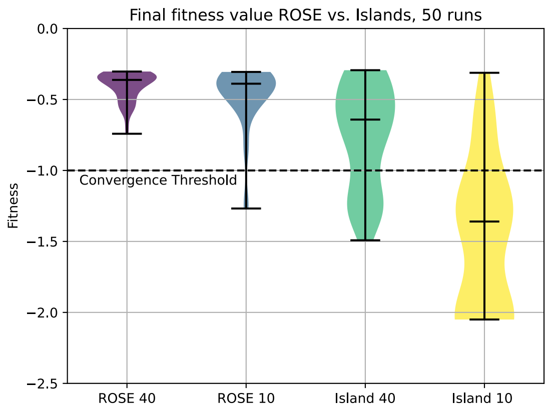

Figure 6.13: A violin plot illustrates the distribution of the best mechanism fitness after 50 epochs. The top and bottom bars represent the maximum and minimum value of the distribution respectively while the middle bar represents the median fitness value.

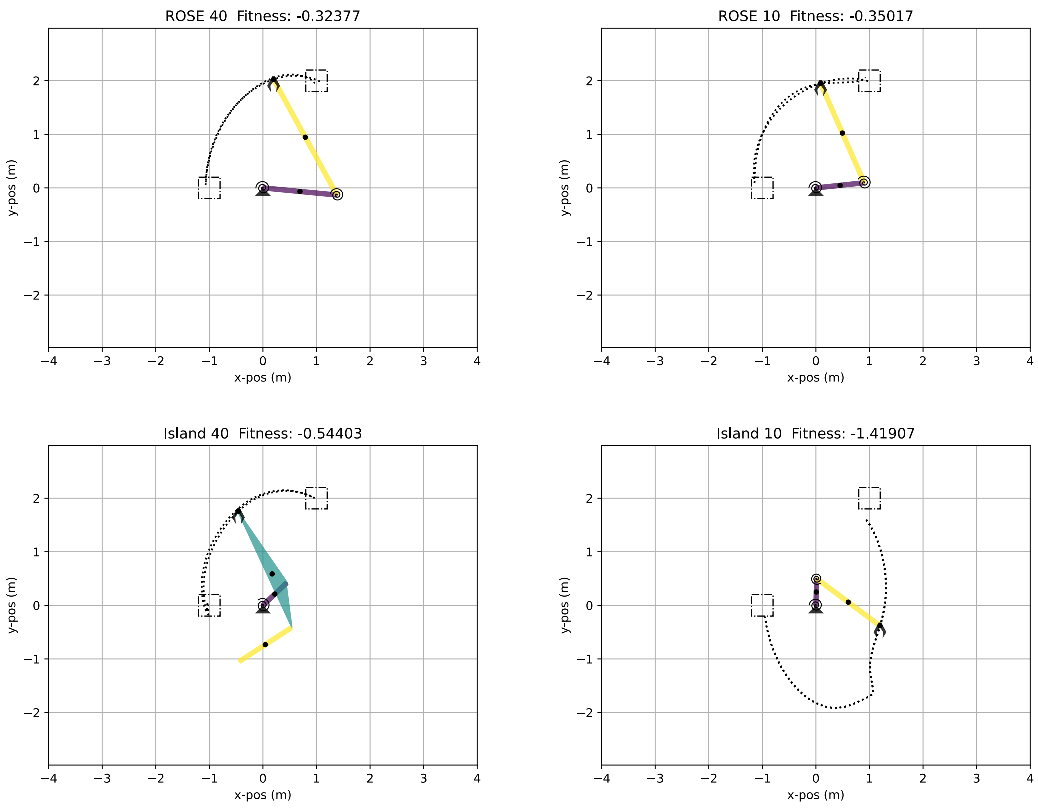
ROSE 40 Fitness: -0.32377
ROSE 10 Fitness: -0.35017
Island 40 Fitness: -0.54403
Island 10 Fitness: -1.41907
y-pos (m)
x-pos (m)

Figure 6.14: The best pick-and-place mechanisms of ROSE 40 (top left) ROSE 10 (top right) Island 40 (bottom left) and Island 10 (bottom right) after 50 epochs.

The best resulting mechanisms of all four algorithms for the first run, which uses seed 0, are displayed in Figure 6.14. Both ROSE algorithms converged on the same structure, a spring-loaded double pendulum that moves counter-clockwise. Island 40 found a solution that uses a counterweight to balance against gravity instead, while Island 10 found a solution with the same structure as the two ROSE solutions but moving clockwise rather than counterclockwise resulting in a much larger overall path and lower position accuracy.

Chapter 7

Discussion

7-1 Summary

The objective was to build a system that could aid designers of mechanical systems and mechanisms to better exploit natural dynamics and embodiment in their design process. From further analysis this goal could be split up into two fundamental tasks:

- The development of an algorithm that achieves automatic generation of mechanisms with desired natural dynamics and kinematics.
- The creation of a software package for the construction, visualization and simulation of mechanisms. Of which the simulator in particular is required to be very fast due to the needs of the computational design algorithms.

The development of the algorithm followed from the mathematical analysis of the search space introduced in Chapter 4, which leads up to the creation of the Reduced Operator-Space Evolution (ROSE) algorithm. A Quality Diversity algorithm which exploits the dual representation of mechanisms introduced in Section 4-2 and the resultant metric space introduced in Section 4-3.

The requirement for a software package led to the creation of PyMechs. A fully documented mechanism simulation, visualization and computation library in Python. With API's for further graph analysis using graph-tool [67] or networkx [68], and the possibility for the addition of active controlled elements in the structure through the provided controller API.

As shown in Figure 6.1 the PyMechs library simulator is up to 600 times as fast as a Python simulator based on Numpy, representing an order of magnitude of performance gain over previous iterations. It also provides visualizations and animations of the mechanisms, such as in e.g. Figures 6.8 and 6.14. This satisfies the design goal of building a performant simulation and visualization package.

The ROSE algorithm developed in this thesis manages to produce viable pick-and-place mechanisms that satisfy the task criteria as shown in Figure 6.8. Satisfying the design goal of a viable computational design system. It outperforms the previously used Island model [25] in the number of simulations required to find a solution, in the quality of the resulting solutions and in the diversity of the solutions as shown by Figures 6.12 and 6.13.

The combination of all the ROSE algorithm and the PyMechs simulator results in a system where an entire set of high-quality solutions to a pick-and-place problem is found in a matter of minutes, compared to the days required for a single solution by the system that inspired this thesis [25].

Of particular benefit is that it generates a large variety of viable solution strategies which makes it more effective as a design aid. An engineer can study the resulting mechanisms and decide which one to use or be inspired by based on any criteria of their choosing.

The ROSE algorithm and mechanism space framework also provides a set of meta-heuristics to give deeper insight into the structure of the search space and the behavior of the algorithm during a run such as shown in Figures 6.5 and 6.10. This allows designers to create their own task-specific operator generator rules and analyze the results in a structured way.

In conclusion the objective to design an automated mechanism design aid was achieved.

7-2 Results

The underlying representation of the mechanisms has no effect on the performance of the ROSE algorithm when using operator based evolution, because the operators *become* the representation used by the algorithm as shown by the duality presented in Section 4-2.

ROSE uses the mechanism space model to generate a large variety of effective results. This can usually be expected to be relatively costly in terms of computation because it maintains a lot of diversity. In practice however it turns out to actually be less costly than the Island model, while obtaining better results as shown by Figure 6.12. It appears that for this problem exploration, e.g. diversity, is more important than exploitation. This matches well with previous adjacent research by e.g. [26, 33, 35] and supports the notion that the search space is highly deceptive.

A deceptive attractor is clearly visible in the analysis of Figure 6.9, which matches the clusters around -1.25 in the endpoint fitness violin plot of Figure 6.13 presented by the Island runs. Clearly the Island algorithms tend to often get stuck in this attractor while ROSE manages to deal with it effectively. As can be seen in Figure 6.7 the ROSE algorithm is still exploring other promising paths for possible breakthroughs even when apparently stuck on a local maximum, which explains how it is able to escape.

Using a more basic island model, but maintaining the complex operators results in greatly improved performance as well as shown in section 6-4. There clearly is great benefit to using abstract operators even with random vertex groupings. The benefit of obtaining the resultant graph structure like Figure 6.5 for analysis is lost however when using just a simple island model. An Island model without the complex operators fails to converge entirely.

7-3 The three problems

7-3-1 Linkage problems

Linkage problems occur because of the disruption of behavior when a mechanism changes structure or parameters. These linkages can effectively be taken into account in ROSE by implementing an operator for them. In the proof of concept operator set presented in Section 6-1-2-4 the transformation operator for example deals with the linkage of mechanism parameters. By applying it the mechanism can be scaled and rotated as a whole. Taking this linkage into account already results in significantly improved performance.

Other linkages can be dealt with simultaneously in ROSE by creating more such operator generation rules, for example to deal with the mechanism control and morphology linkage explained in the introduction.

7-3-2 Deception

Deception represents the presence of solutions in the search space that are powerful local maxima which are particularly hard to escape.

The version of ROSE used for the results in chapter 6 constitutes a proof of concept. It shows that by utilizing the theoretical analysis it is possible to straightforwardly define meaningful heuristics on operator space through operator generation rules of an operator set Φ. The corresponding domain Ω^Φ can then be analyzed such as is done in Section 6-3.

In the meta-heuristics analysis a deceptive attractor is clearly visible in the fitness space of Figure 6.9. This is further reinforced by analyzing the domain in Figure 6.6, which shows that despite the use of abstract operators the algorithm still has to pass through several low performing structures to reach the high performing mechanism structures. When no operators would be used the number of poorly performing nodes in between would be even larger, since the current domain excludes for example disconnected mechanisms. By introducing shortcuts around these with operators while introducing more diversity the ROSE algorithm manages to deal with it very effectively. This is especially clear in the violin plot of Figure 6.13.

Mechanism and operator complexity are closely related, it has been shown in chapter 4 that a mechanism M can be represented as an associated operator φ_M. Solution complexity is also closely related to the difficulty of the optimization problem, a complex solution is harder to find on a given representation than a simpler one in much the same sense that a small maze is easier to navigate than a large one.

This leads to the interesting conclusion that it is possible to lower the difficulty of the optimization problem: by reducing the complexity of the mechanism on it's operator representation. This can be achieved by for example picking the right combinations of operators as a set of base operators for the search. By projecting onto the domain of the right set of operators deception can in theory be bypassed altogether.

7-3-3 Redundancy

Redundancy is the lack of uniqueness of the representation, the design space, the search space and the objective. An isomorphism strategy from prior art was used to condense the structural nodes in the ROSE algorithm. The parameter invariants 3 are used to reduce redundancy, in particular the scaling operator which maintains the relative mechanism structure. The high-performance simulator reduced the cost of simulation as well, making redundancy a less important problem.

7-4 Drawbacks

Keeping track of the space exploration graph requires memory, the usage of which can in theory grow to be very large. For any of the tests the size of the graph even with no optimization of memory usage at all is reasonable (<200 Mb) even for very long algorithm runs that explore most of the effective search space. But, due to the penalty on the complexity the total search space is bounded to be relatively small for most test problems. More advanced optimization problems might run into memory issues when highly complex graph structures are explored. Dividing and bounding the search space effectively for Quality Diversity algorithms such as ROSE is an ongoing research topic [35].

Generating a new vertex requires comparing it to every vertex already in existence to check if it is isomorphic to an existing structure, as the amount of vertices grows this comparison scales poorly. To avoid a lot of very expensive isomorphism computations the algorithm already checks if the complexity is equal, and only then checks for isomorphism.

In the practical tests the cost of adding new nodes is very small compared to the cost of simulation, not in small part because this only happens every epoch. In different problems with very computationally cheap evaluations and lots of possible nodes, the epoch step of the algorithm can quickly become comparatively computationally expensive.

The mechanism distance is only used implicitly and the mapping of the mechanism space is stochastic. The result is that it can actually be possible for nodes that are very far away from each other in the fitness space graph to actually be really close to each other in mechanism space. This means that the graphs drawn by the algorithm can occasionally be misleading, it is important to keep this in mind.

Many of these problems can be avoided by picking another type of vertex grouping, as the current grouping is itself a heuristic overlaid on top of the infinite mechanism space structure as explained in Section 4-3. When exploring larger, more complex spaces the groupings themselves could also be made larger and more complex.

Another solution is obtained by picking the right operator set to reduce the space complexity further, already many possible mechanism structure nodes are discarded as these are unviable. When dealing with more complex problems using the right operators with the right domain is likely to be even more effective.

7-5 Further research suggestions

7-5-1 Operator generation

The operators defined in Section 6-1-2-4 used to generate the results are just one among many different possible sets. However it is clear that through careful study of the mechanisms it is possible to create many more effective operators. For example ones that preserve the degrees of freedom of the mechanism.

It is possible to convert existing mechanisms from a database of successful existing mechanisms into the representation defined in this thesis. It should be possible to find operators that are more effective than the base operators in solving the automated design problem. This could be done for example by minimizing the total complexity of all operators used to represent the successful mechanisms.

The mechanism space admits an entirely different optimization approach as well. As shown in Section 4-3 the search space of automated mechanism design can be represented by a graph. The possible paths through this graph can be interpreted as a decision tree. A different approach could be to let the algorithm explore the operators as a set of decisions with corresponding expected fitness results using a reinforcement learning approach.

As indicated by the results in chapter 6 there are some nodes that have more viability for solutions than others, this is especially clear from Figures 6.5 and 6.6. This is supported by the observation that the structural variance of real world mechanisms is generally limited. With many mechanisms consisting of just four links [56]. Studying what properties make certain structures more viable than others, and refining the set of operators such that only viable ones are explored would likely significantly reduce search difficulty.

Of course filtering too much prematurely might end up preventing novel solutions being discovered. Designing good operators to bound the search space in desired ways would be of interest for future research.

7-5-2 Parameter correlations and hyper-parameters

The parameter function has not been utilized to it's potential in the proof of concept version of ROSE presented in chapter 6. By statistically analyzing existing mechanisms it should be possible to discover useful correlations between their parameters. These can then be used to define rules on new parameter generation.

An interesting possibility for future research would be to let the parameters of the mechanism themselves be a function of some hyper parameter based network or function. This indirect encoding could maintain parameter correlations and behavior better, this approach has shown significant success when applied to the evolution of neural networks [59, 72].

An alternative way to capture these parameter correlations would be to define them on the operators instead and only use parameter mutations that take them into account, this is more flexible with respect to many different objectives.

7-5-3 Controller generation

While the mechanism library presented here allows motors and control through the provided API the consequences of them have not yet been fully explored to their potential and the results have been limited.

A control mutation operator could be developed, one that takes into account both the controller and the mechanism with some simple rules such as stability. This should take care of the inherent linkage problems between mechanisms and controllers. Designing the right operator generation strategy for this is still an open question and would prove an interesting research direction. In particular an operator that maps a controller behavior to a new mechanism topology would be invaluable.

Because mechanisms are constructed as interconnected elements with holonomic constraints, a Port-Hamiltonian approach [73] might prove to be an effective strategy to inform controllers or at a minimum provide constraints of stability.

7-5-4 Other applications

A further look at the method of automated design as described here shows that the method itself is in fact not limited to just mechanisms at all. It is possible to apply it to any problem where an initial model representation of the problem and objective function can be created. By letting the algorithm explore for a better model representation and evaluating and analyzing the steps it took to get to a solution it is possible to continuously learn to improve the search efficiency for that specific domain.

This could be used to for example topologically reduce neural networks, electronic circuits, or any other mixed-integer computational design problem where an underlying structure is present.

7-6 Reflection

A set of simple generation rules can lead to very complex structures and emergent behavior, as shown by Conway's Game of Life [74] and the field of Cellular Automata [75]. Conversely, many resulting complex structures can actually be described by a very simple, short set of rules. A highly condensed representation of a complex system.

The field of Machine Learning which has seen a recent boom fundamentally relies on this property. Training a neural network on a dataset is equivalent to finding the right set of reduced mathematical rules to describe the desired transformation from network input to output [15]. Recent work in GAN's for example has shown that neural networks can capture a set of simple rules to generate eerily realistic human faces [39].

This type of representation and algorithm has analogies to the natural world: a DNA strand does not directly encode the structure of a flower but rather holds a set of instructions which leads to the growth of one, a subtle but important distinction.

This notion was the inspiration for ROSE, rather than rely on the modification of the parameters directly a set of rules leads to the resulting mechanisms. The operators defined in Section 6-1-2-4 are a proof of concept but the resulting algorithm already outperforms existing methods in convergence rate, solution quality and solution diversity as shown in the case study. However, rather than these particular operators it is the mathematical framework and the software tools to analyze it that are the real contribution of this thesis.

Capturing underlying rules of the complex system under consideration, either automatically or by manual investigation, is a goal worthy of further investigation. It is what I believe to be a key step to the creation of improved computational design methods that could eventually capture some of the wonders of the natural world.

Chapter 8

Conclusion

ROSE algorithm

Automated design tasks can be defined as an optimization problem by combining a model representation which maps the desired real world object to a mathematical object (and vice versa) with a model evaluation function which maps this object to a numerical measure of success on the objective. Prior art shows this can be applied to mechanism design using a graph based mechanism representation and an evolutionary algorithm.

However, existing implementations suffer from significant linkage problems, the deceptive search space and design space redundancy. To improve upon these the problem is analyzed in-depth leading to the following:

- The graph model of mechanisms is extended using a parameter map function, which maps the vertices and edges of the graph representation to numerical parameters. It is shown this can be used to define constraints and correlations on the parameters. In particular the correlations can be used effectively generate parametric mutations by defining these as operators. These operators can capture combinations of parameters which combine to overall behavior, by making those combinations invariant to the operator the behavior is preserved by the operator.

- Extra elements are introduced to the mechanism structure, the end-effector, the torsion spring, the fixed connection and the motor. These allow a richer solution space and can better model the desired mechanism solutions.

- The concept of mechanism space is introduced as the possible solution space of planar mechanisms. This is a metric space, where the distance between mechanisms is equal to a word metric where the set of generating elements is the set of operators used. By modifying the operator set the connectivity of this space is modified, which reshapes the search space and the resultant evolutionary algorithm in beneficial ways. By creating operators that preserve certain mechanism properties, such as connectivity, the search space can also be effectively bound. This reduces both redundancy and deception.

- Operators acting on the mechanisms are studied in the context of the mechanism space. It is shown that there is a dual representation of mechanisms using operators. The mechanism space together with a fitness function and set of operators represent a projection of mechanism space onto the domain of operators in which the algorithm explores for an optimal solution. By defining a seed mechanism and setting this as the initial root node the search space is shown to be a rooted, ordered graph where each level of depth represents a change in mechanism complexity, a useful meta-heuristic tool.

These extensions culminated in the development of the Reduced Operator-Space Evolution (ROSE) algorithm. A quality diversity algorithm that relies on operator generation rules and the corresponding projected domain in mechanism space. A proof-of-concept implementation of ROSE using simple operator generation rules is tested against previous state-of-the-art Island models and a clear improvement in both convergence properties and solution quality is shown.

PyMechs

In order to generate and analyze the results, a fully documented planar mechanism simulation and evolution toolbox called PyMechs has been developed for Python based on the representation presented in this thesis. The physics simulator is coded in C++ and has been optimized using both profile guided optimization and mathematical reductions to achieve significant performance gains of up to 6000% compared to previous iterations using Numpy.

To provide analysis tools a visualizer is provided with the library to animate and plot mechanism trajectories and structures. The library can optionally be linked with Graph-Tool [67] or Networkx [68] to easily analyze the mechanism and search-space graph structures.

The software that was developed in order to study mechanisms based on the graph representation is available at https://github.com/kooswestra/pymechs.

Acknowledgments

This document came to be through the support of many people over a long period, owing to a variety of medical issues.

I would like to give special thanks to my supervisor ir. A. Berry for his continued support, in particular in helping me deal with the multitude of setbacks I encountered. His administrative and moral support was invaluable.

I want to thank prof. dr. ir. M. Wisse for his support and willingness to organize a change in supervision, as well as his personal involvement. I want to thank dr. ir. W.J. Wolfslag for helping me get started on the project and motivating me to see it through. I would also like to thank ir. P.R. Kuppens for the many interesting discussions we had on this subject. I want to thank C. Verdier for his support as interim supervisor as well.

Additionally I would like to thank ir. B. van Vliet for his support as study coordinator in helping me clear up my program and getting everything back in order.

Finally I want to thank both my parents and my partner in crime Miara Fraikin for their continued and enduring support.

List of Symbols

Notation	**Description**
$\mathbf{p}$	Parameter vector
$\mathbf{x}$	State vector
$\mathbf{u}$	Input vector
$\hat{y}$	Model
y	Desired ideal model
f	Objective function
M	Mechanism
C	Controller
M_C	Controlled mechanism
G	Graph
V	Vertex set
E	Edge set
Ψ	Incidence function
P	Parameter assignment map
Ω	Domain
$\mathcal{M}$	Set of all mechanisms
φ	Operator
c	Complexity measure
$\hat{c}$	Simplified complexity measure
d	Distance measure
Φ	Set of operators
Ω^{Φ}	Domain corresponding to a set of operators
$\mathbb{M}$	Mechanism space
$\mathbf{M}$	Mass matrix
$\ddot{\mathbf{x}}$	Acceleration vector
$\mathbf{v}$	Velocity vector
$\mathbf{f}$	Force vector
$\mathbf{D}$	Constraint equations
$\boldsymbol{\lambda}$	Lagrange multiplier vector

Appendix A

Dynamics

A-1 Hinge example

In order to illustrate how to actually obtain the required formulas to build the equations of motion in an automated way we can look at a hinge linkage in more detail. A sketch of the linkage is shown in Figure A.1:

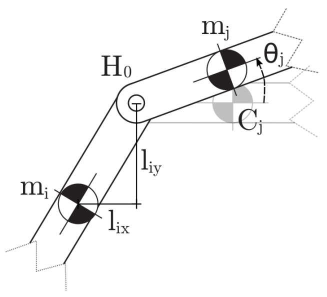

Figure A.1: Sketch of a hinge linkage between two links m_i and m_j, m_i is in it's initial position while m_j is rotated around H_0 compared to the initial position, the initial locations of the centers of mass are given by $\mathbf{c}_i$ and $\mathbf{c}_j$ and the initial coordinates of the hinge linkage by H_0.

From the figure we can deduce the constraints introduced by hinge n as:

$$\mathbf{D}_n = R(\theta_i)\,\mathbf{l}_i + \mathbf{x}_i - R(\theta_j)\,\mathbf{l}_j - \mathbf{x}_j = 0 \tag{A.1}$$

Where $\mathbf{l}_i = \mathbf{c}_i - H_0$ and $\mathbf{l}_j = \mathbf{c}_j - H_0$ are the distance vectors of the centers of mass of the links to the hinge position at the initial state as visualized in the figure. These vectors are then rotated with a 2D rotation matrix $R(\theta)$ to obtain the distance vector at the current state. Writing out all components results in the following set of equations:

$$\mathbf{D}_n = \begin{bmatrix} x_i - x_j + l_{ix}\cos(\theta_i) - l_{iy}\sin(\theta_i) - l_{jx}\cos(\theta_j) + l_{jy}\sin(\theta_j) \\ y_i - y_j + l_{ix}\sin(\theta_i) + l_{iy}\cos(\theta_i) - l_{jx}\sin(\theta_j) - l_{jy}\cos(\theta_j) \end{bmatrix} = \mathbf{0} \tag{A.2}$$

Which defines the constraints for this specific connection. Calculating the Jacobian of constraint n with respect to the state of the connected linkages $\begin{bmatrix} x_i & y_i & \theta_i & x_j & y_j & \theta_j \end{bmatrix}^T$ results in the following expression:

$$\mathbf{J}_n = \begin{bmatrix} 1 & 0 & -l_{ix}\sin(\theta_1) - l_{iy}\cos(\theta_i) & -1 & 0 & l_{jx}\sin(\theta_j) + l_{jy}\cos(\theta_j) \\ 0 & 1 & l_{ix}\cos(\theta_1) - l_{iy}\sin(\theta_i) & 0 & -1 & -l_{jx}\cos(\theta_j) + l_{jy}\sin(\theta_j) \end{bmatrix} \tag{A.3}$$

Calculating the corresponding Coriolis terms yields:

$$\frac{\partial(\mathbf{J}_n\dot{\mathbf{x}})}{\partial\mathbf{x}}\dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta}_i^{\,2}\left(-l_{ix}\cos(\theta_i) + l_{iy}\sin(\theta_i)\right) + \dot{\theta}_j^{\,2}\left(l_{jx}\cos(\theta_j) - l_{jy}\sin(\theta_j)\right) \\ \dot{\theta}_i^{\,2}\left(-l_{ix}\sin(\theta_i) - l_{iy}\cos(\theta_i)\right) + \dot{\theta}_j^{\,2}\left(l_{jx}\sin(\theta_j) + l_{jy}\cos(\theta_j)\right) \end{bmatrix} \tag{A.4}$$

A-2 Polygonal elements

The shape of each mass element is defined by locations of it's connection points. Since masses can have more than two connections, this means that the shape can be a polygon consisting of a number of points equal to the number of connections. These elements are defined as *simple polygons:* shapes defined as the area enclosed by a sequence of points $\mathcal{P}$. Figure A.2 shows an example of such a polygon.

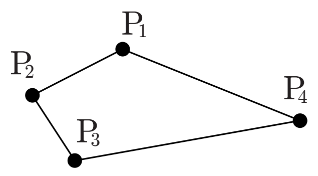

Figure A.2: A simple polygon defined by the sequence of points $\{P_1, P_2, P_3, P_4\}$, note that the polygon is ordered anti-clockwise.

A-2-1 Construction and ordering

To calculate the properties of these planar elements and be able to properly draw them the polygon has to be constructed and ordered anti-clockwise. This is done by calculating the incident angle ϕ_n of each connection point $P_n(x_n, y_n)$ with a point which is inside the polygon area $P_{in}(x_{in}, y_{in})$, as:

$$\phi_n = \arctan\left(\frac{y_n - y_{in}}{x_n - x_{in}}\right) \tag{A.5}$$

And reordering $n \to k$ the connections points by increasing angle such that $\phi_{k+1} > \phi_k$, defining a polygon boundary. The result is that any lines drawn from this point to the polygon vertices can not cross the polygon boundary. We pick the mean of the connection points to by definition be a point inside the polygon area:

$$P_{in} = \frac{1}{n}\sum_n P_n \tag{A.6}$$

This precludes the use of hollow shapes or other shapes that can not be described in this manner. A possible workaround is to manually order the connections before element construction and disable automatic ordering.

A-2-2 Center of mass

Since a constant density ρ of the link element is assumed, the center of mass of the polygonal shape is equal to it's centroid. Which is given by:

$$\begin{aligned} C_x =& \frac{1}{6A} \sum_n (x_{n+1} + x_n)(x_n y_{n+1} - x_{n+1} y_n) \\ C_y =& \frac{1}{6A} \sum_n (y_{n+1} + y_n)(x_n y_{n+1} - x_{n+1} y_n) \end{aligned} \tag{A.7}$$

Where A is the polygon area is given by:

$$A = \frac{1}{2} \sum_n (x_n y_{n+1} - x_{n+1} y_n) \tag{A.8}$$

For both equations the vertices loop around $P_{n+1} = (x_0, y_0)$ i.e. the polygon boundary ends at the some point where it begins.

A-2-3 Moment of inertia

For solid rods the moment of inertia can easily be shown to be equal to $\frac{mL^2}{12}$, but for general polygonal shapes this is a bit more involved. The moment of inertia for a planar elements with constant density $\rho = \frac{m}{A}$ and center of mass at the origin is defined as:

$$I = \rho \iint_A r^2 dA = \rho \iint_A \left(x^2 + y^2\right) \rho dA \tag{A.9}$$

Note that the (anti-clockwise ordered) polygon of the mass element by definition defines a closed boundary around the area in terms of the connecting lines $P_n \rightarrow P_{n+1}$ between it's $n \geq 3$ points $P_n = (x_n, y_n)$. Green's theorem states that it is possible to turn an integral over an area A enclosed by a piecewise smooth, oriented curve C into an integral along the boundary:

$$\iint_A \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) dA = \oint_C (Ldx + Mdy) \tag{A.10}$$

Where the path of integration is anti-clockwise. This means that if we define the functions M and L satisfying $\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} = \left(x^2 + y^2\right)$, we can turn the integral over the polygon area into an integral over the polygon boundary. By setting $M = \frac{1}{3}x^3$ and $L = -\frac{1}{3}y^3$ we satisfy this condition resulting in:

$$I = \rho \oint_C \left(-\frac{1}{3}y^3 dx + \frac{1}{3}x^3 dy\right) \tag{A.11}$$

Where the integral is along the polygon boundary defined by the piecewise connecting lines between the ordered polygon points $C = C_0 \cup C_1 \cup ... \cup C_n$. As this is a line integral the integral along the entire boundary is equal to the sum of the integrals along each individual segment

$$\oint_C f(u)\, du = \sum_n \oint_{C_n} f(u)\, du \tag{A.12}$$

Note that each segment of the curve is a line connecting the point P_n with the point P_{n+1}. We can parameterize each curve section as:

$$C_n = \left(\left(x_{n+1} - x_n\right) u + x_n, \left(y_{n+1} - y_n\right) u + y_n\right), \quad 0 \leq u \leq 1 \tag{A.13}$$

substituting this parameterization into equation A.11 to obtain the contribution of a segment C_n results in:

$$\begin{aligned}\rho \oint_{C_n} \left(-\frac{1}{3}y^3 dx + \frac{1}{3}x^3 dy\right) = &-\frac{\rho}{3}\int_0^1 \left(\left(y_{n+1} - y_n\right) u + y_n\right)^3 \left(x_{n+1} - x_n\right) du \\ &+ \frac{\rho}{3}\int_0^1 \left(\left(x_{n+1} - x_n\right) u + x_n\right)^3 \left(y_{n+1} - y_n\right) du \end{aligned} \tag{A.14}$$

Performing the integral and collecting terms results in the following complete expression for the inertia of a polygonal element, whose center of mass is at the origin:

$$I = \sum_n \frac{\rho}{12} \left[\left(y_{n+1} - y_n\right)\left(x_{n+1} + x_n\right)\left(x_{n+1}^2 + x_n^2\right) - \left(x_{n+1} - x_n\right)\left(y_{n+1} + y_n\right)\left(y_{n+1}^2 + y_n^2\right)\right] \tag{A.15}$$

We can calculate the inertia of any general polygon by shifting the vertices such that the center of mass is at zero and then using equation A.15.

Note that the area of the polygon can be derived by setting M and L such that $\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} = 1$. By defining $M = \frac{1}{2}x$ and $L = -\frac{1}{2}y$ and using the parametrization from equation A.13 we obtain the expression for the polygon area given by equation A.8. It is important to note that it is not possible to substitute $\rho = \frac{m}{A}$ inside the sum of equation A.15, as the contribution of a single section of the curve to the area can be equal to zero.

Bibliography

[1] Rolf Pfeifer, Max Lungarella, and Fumiya Iida. Self-organization, embodiment, and biologically inspired robotics. *science*, 318(5853):1088–1093, 2007.

[2] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence.* MIT press, 2006.

[3] Rolf Pfeifer and Gabriel Gómez. Morphological computation–connecting brain, body, and environment. *Creating brain-like intelligence*, pages 66–83, 2009.

[4] Nicolas Franceschini, Jean-Marc Pichon, and Christian Blanes. From insect vision to robot vision. *Philosophical Transactions of The Royal Society Of London. Series B: Biological Sciences*, 337(1281):283–294, 1992.

[5] Kazunori Hoshino, Fabrizio Mura, and Isao Shimoyama. Design and performance of a micro-sized biomorphic compound eye with a scanning retina. *Journal of Microelectromechanical Systems*, 9(1):32–37, 2000.

[6] Michiel Plooij and Martijn Wisse. A novel spring mechanism to reduce energy consumption of robotic arms. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2901–2908. IEEE, 2012.

[7] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

[8] Roger D Quinn and Roy E Ritzmann. Construction of a hexapod robot with cockroach kinematics benefits both robotics and biology. *Connection Science*, 10(3-4):239–254, 1998.

[9] Wouter Wolfslag, Michiel Plooij, Robert Babuška, and Martijn Wisse. Learning robustly stable open-loop motions for robotic manipulation. *Robotics and Autonomous Systems*, 66:27–34, 2015.

[10] Heiko Hamann, Jürgen Stradner, Thomas Schmickl, and Karl Crailsheim. A hormone-based controller for evolutionary multi-modular robotics: From single modules to gait learning. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[11] Yasuhiro Fukuoka, Hiroshi Kimura, and Avis H Cohen. Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts. *The International Journal of Robotics Research*, 22(3-4):187–202, 2003.

[12] Da-peng Yang, Jing-dong Zhao, Yi-kun Gu, Xin-qing Wang, Nan Li, Li Jiang, Hong Liu, Hai Huang, and Da-wei Zhao. An anthropomorphic robot hand developed based on underactuated mechanism and controlled by emg signals. *Journal of Bionic Engineering*, 6(3):255–263, 2009.

[13] AE Eiben and JE Smith. *Introduction to Evolutionary Computing.* Springer, 2nd edition, 2015.

[14] AJ Keane. The design of a satellite beam with enhanced vibration performance using genetic algorithm techniques. *Journal of the Acoustical Society of America*, 99(4):2599–2603, 1996.

[15] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[16] Agoston E Eiben. Grand challenges for evolutionary robotics. *Frontiers in Robotics and AI*, 1:4, 2014.

[17] David E Goldberg. Genetic algorithms as a computational theory of conceptual design. In *Applications of Artificial Intelligence in Engineering VI*, pages 3–16. Springer, 1991.

[18] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[19] P Reinier Kuppens and Wouter J Wolfslag. A string-based representation and crossover operator for evolutionary design of dynamical mechanisms. *IEEE Robotics and Automation Letters*, 3(3):1600–1607, 2018.

[20] I.C. Staal. Evolutionary mechanisms automated synthesis of robotic mechanisms by an evolutionary algorithm. Master's thesis, TU Delft, 2014.

[21] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[22] Yong-Mo Moon and Sridhar Kota. Automated synthesis of mechanisms using dual-vector algebra. *Mechanism and Machine Theory*, 37(2):143–166, 2002.

[23] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[24] Chris Leger. *Darwin2K: An evolutionary approach to automated design for robotics*, volume 574. Springer Science & Business Media, 2012.

[25] P.R. Kuppens. Automated robot design with artifical evolution. Master's thesis, TU Delft, 2016.

[26] Joel Lehman and Kenneth O Stanley. Evolving a diversity of creatures through novelty search and local competition. In *Proceedings of the genetic and evolutionary computation conference*, 2011.

[27] Anton Bernatskiy and Josh Bongard. Choice of robot morphology can prohibit modular control and disrupt evolution. In *European Conference on Artificial Life (ECAL)*, pages 60–67. Cambridge, MA: MIT Press, September 2017.

[28] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Oxford, England: U Michigan Press, 1975.

[29] Milan Jelisavcic, Rafael Kiesel, Kyrre Glette, Evert Haasdijk, and A. E. Eiben. Benefits of lamarckian evolution for morphologically evolving robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 65–66, New York, NY, USA, 2017. ACM.

[30] D.E. Goldberg. Simple genetic algorithms and the minimal deceptive problem. *Genetic Algorithms and Simulated Annealing, Research Notes in Artifical Intelligence*, pages 71–88, 1987.

[31] Brian Heater. Amazon says it has deployed more than 200,000 robotic drives globally, Jun 2019.

[32] Amazon robotics challenge. https://www.amazonrobotics.com/, 2017.

[33] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Proceedings of the Artificial Life Conference*, pages 226–234, 2016.

[34] Literature Survey. 2017.

[35] Justin K Pugh, Lisa B Soros, Paul A Szerlip, and Kenneth O Stanley. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 967–974. ACM, 2015.

[36] Joel Lehman and Kenneth O Stanley. Abandoning objectives: evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[37] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life.* John Murray, 1859.

[38] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 99–106, New York, NY, USA, 2017. ACM.

[39] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[40] Darrell Whitley and Timothy Starkweather. Genitor ii: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(3):189–214, 1990.

[41] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[42] Lennart Ljung. System identification. *Wiley encyclopedia of electrical and electronics engineering*, pages 1–19, 1999.

[43] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[44] Elliot Meyerson, Joel Lehman, and Risto Miikkulainen. Learning behavior characterizations for novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 149–156, New York, NY, USA, 2016. ACM.

[45] Elliot Meyerson and Risto Miikkulainen. Discovering evolutionary stepping stones through behavior domination. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 139–146, New York, NY, USA, 2017. ACM.

[46] Jonathan C. Brant and Kenneth O. Stanley. Minimal criterion coevolution: A new approach to open-ended search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 67–74, New York, NY, USA, 2017. ACM.

[47] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[48] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, 22:215–221, 1955.

[49] Reinhard Diestel. *Graph Theory.* Springer Graduate Texts in Mathematics. Springer-Verlag, 5th edition, 2017.

[50] Kaz Vermeer, Reinier Kuppens, and Justus Herder. Kinematic synthesis using reinforcement learning. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 51753, page V02AT03A009. American Society of Mechanical Engineers, 2018.

[51] Fabian Fränz, Jan Paredis, and Rico Möckel. On the combination of coevolution and novelty search. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 201–208. IEEE, 2017.

[52] Askhat Diveev, David Kazaryan, and Elena Sofronova. Symbolic regression methods for control system synthesis. In *Control and Automation (MED), 2014 22nd Mediterranean Conference of*, pages 587–592. IEEE, 2014.

[53] P.J. Fleming and R.C. Purshouse. Evolutionary algorithms in control systems engineering: a survey. *Control engineering practice*, 10(11):1223–1241, 2002.

[54] Jose Miguel Atiencia Villagomez, Askhat Diveev, and Elena Sofronova. The network operator method for synthesis of intelligent control system. In *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*, pages 174–179. IEEE, 2012.

[55] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[56] Franklin Day Jones, Holbrook Lynedon Horton, and John A Newell. *Ingenious Mechanisms for Designers and Inventors*, volume 1-4. Industrial Press Inc., 1967.

[57] Russell Charles Hibbeler. *Dynamica*. Pearson Education, 2010.

[58] Kenneth O. Stanley and Jason Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 2009.

[59] Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. Hyperneat-ggp: A hyperneat-based atari general game player. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 217–224, New York, NY, USA, 2012. ACM.

[60] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[61] Mícheál O'Searcoid. *Metric spaces*. Springer Science & Business Media, 2006.

[62] Hod Lipson. How to draw a straight line using a gp: Benchmarking evolutionary design against 19th century kinematic synthesis. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, Seattle, Washington, USA*, volume 26, 2004.

[63] Randall Munroe. Standards. https://xkcd.com/927/.

[64] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[65] William E Boyce, Richard C DiPrima, and Douglas B Meade. *Elementary differential equations*. John Wiley & Sons, 2017.

[66] Eliakim H Moore. On the reciprocal of the general algebraic matrix. *Bull. Am. Math. Soc.*, 26:394–395, 1920.

[67] Tiago P Peixoto. The graph-tool python library. *figshare*, 2014.

[68] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[69] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[70] Darrell Whitley, Soraya Rana, and Robert B Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7(1):33–47, 1999.

[71] William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. *Statistical parametric mapping: the analysis of functional brain images.* Elsevier, 2011.

[72] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, CEC'09, pages 2764–2771, Piscataway, NJ, USA, 2009. IEEE Press.

[73] Arjan Van Der Schaft and Dimitri Jeltsema. Port-hamiltonian systems theory: An introductory overview. *Foundations and Trends in Systems and Control*, 1(2-3):173–378, 2014.

[74] John Conway. The game of life. *Scientific American*, 223(4):4, 1970.

[75] B Chopard and M Droz. *Cellular automata*, volume 1. Springer, 1998.