A Hybrid Approach to University Course Timetabling Using Reinforcement Learning and Genetic Algorithm

A Special Problem by

Sean Francis N. Ballais
2015-04562
BS Computer Science

Presented to the Faculty of the
Division of Natural Sciences and Mathematics

In Partial Fulfillment of the Requirements
For the Degree of
Bachelor of Science in Computer Science

University of the Philippines Visayas
TACLOBAN COLLEGE
Tacloban City

Month Year

This special problem, entitled "**A HYBRID APPROACH TO UNIVERSITY COURSE TIMETABLING USING REINFORCEMENT LEARNING AND GENETIC ALGORITHM**", prepared and submitted by **SEAN FRANCIS N. BALLAIS**, in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** is hereby accepted.

PROF. VICTOR M. ROMERO II
Special Problem Adviser

Accepted as partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**.

DR. EULITO V. CASAS JR.
Chair, DNSM

Acknowledgements

First of all I would like to thank the Lord for his guidance during the course of my research and for making this thesis possible. I would like to thank my family who served as my inspiration, and never failed to support me all throughout my studies. Thanks to ...

Abstract

Although the abstract is the first thing that appears in the thesis, it is best written last after you have written your conclusion. It should contain spell out your thesis problem and describe your solution clearly.

Make sure your abstract fits in one page !

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A common activity in universities and all academic institutions is course timetabling. This activity is a necessity in academic institutions as this is where classes are scheduled for the incoming semester. In some instances, even entire courses are scheduled for at least four years [**?**]. Performing timetabling manually is tedious and schedules conflicts may not immediately be determined [**?**]. As such, automating this procedure will be beneficial to academic institutions. This problem of automating course scheduling is actually formally known as the University Course Timetabling Problem (UTP) [**?**][**?**]. Many works have already proposed different approaches to automate the timetabling process. Some of the algorithms that were utilized by proposed appraoches include great deluge [**?**][**?**], machine learning [**?**], and genetic algorithms [**?**][**?**][**?**]. Their results produce optimal timetables, or near to the optimal, at least for the case they were originally intended for. Applying their approaches to other institutions *may* require some tweaking to fit in with the new environment they are going to be used in. There is no one approach that works for all cases as is. Due to this, there is a considerable amount of research that is done and is being done in further refining and discovering methods for automatic timetabling.

1.1 The University Course Timetabling Problem

The University Course Timetabling (UCT) Problem is a problem where l lectures are being placed into a t timeslots and r rooms in such a way that it will result in a feasible timetable [?]. It was proven by Cooper, T. and Kingston, J. that this problem is an NP-complete problem which means it is difficult to produce feasible timetables [?]. Identifying whether a timetable is feasible or not is determined by two types of constraints: (a) hard constraints, and (b) soft constraints

Hard constraints are constraints that are not supposed to be violated. Violation of these constraints immediately leads infeasibility of a potential. Some proposed approaches immediately discard timetables that are infeasible [?][?], while others would give high penalty costs for each hard constraint violated [?]. The set hard constraints may differ from one approach to another. However, a common hard constraint that can be observed across approaches is that a student, and, naturally, a teacher as well, cannot attend two or more classes in the same timeslot [?][?]. This is obvious as a person cannot be in two or more places at the same time. These hard constraints are what all constructed timetables must satisfy to be considered as a potential solution.

The counterpart to hard constraints are soft constraints. Unlike, hard constraints, soft constraints are *technically* optional for timetables to satisfy. However, finding the best timetables for a given problem require minimization of the number of soft constraints being violated [?]. The set soft constraints differ from one case to another. Another case might require soft constraints that are not present in another case. For example, one approach [?] has a soft constraint where students should not have only one class in a single day, while another [?] does not and instead does not allow having

more than three lectures of the same class schedule adjacently on the same day. Many approaches do not consider hard constraints in computing the fitness value of a timetable. Instead, many only use the soft constraints as part of their fitness function [?][?][?].

1.2 Timetabling in The University of the Philippines Visayas Tacloban College

The University of the Philippines Visayas Tacloban College is, at the time of writing, a college under the administration of the University of the Philippines Visayas, a constituent of the University of the Philippines System. The college consists of four divisions: Division of Natural Sciences and Mathematics, Division of Humanities, Division of Social Sciences, and Division of Management. Through these divisions, the college offers eight undegraduates degrees: BS Computer Science, BS Biology, BA Social Sciences in Political Science, BA Social Sciences in Economics, BA Psychology, BA Communication Arts, BS Accountancy, and BS Management. Each of these degrees offer classes that are required to be attended and completed by the students undertaking the degree. These classes last from one hour to one and a half hours per session. A session of a lecture-only classes is allocated 1 hour and 30 minutes, while sessions for PE classes and lecture classes with laboratory sessions are given just 1 hour. A class can only have one session per half-week or week if scheduled on a Wednesday. A half-week consists of the first or last two days of a weekday. Classes retain their usual scheduling if scheduled on a half-week but are given twice their usual timeslots when scheduled on a Wednesday. When a class is scheduled in the first half-week, then its schedule is mirrored in the next half-week on its counterpart

day. For example, if a class session is scheduled on a Monday at 1PM, then another session of the class is scheduled on Thursday at 1PM. A class is usually assigned a room that is reserved to the division the class is offered by. As such, the chances that two classes from different divisions will share the same room is unlikely. Two or more instances (sections in the terms of the college) of the same class may be offered by the division depending on the necessity but may be taught by different teachers and in different rooms and, obviously, time slots. Scheduling these classes are typically done manually. The classes of each division are scheduled by a the division chair. Teachers may specify their preferred teaching times and the division chair tries his/her best to accomodate their preferences as much as possible. For our timetabling model, we can determine the following hard constraints basing from the aforementioned setup:

- No two classes scheduled in the same timeslot can have the same students and teachers.
- Each room can only accomodate one class per timeslot.

We can also formulate the following soft constraints:

- A class must be scheduled in rooms that are reserved for the division the class is offered by.
- As much as possible, no classes should be scheduled on the timeslots, 7AM to 8:30AM, and 5:30PM to 7PM.
- No class must be scheduled in the timeslot unpreferable to the teacher assigned to the class.

Computing the objective function will be based on these soft constraints. Each soft constraint will be given a weight which will represent the constraint priority in terms of being satisfied and to help steer the timetable towards a schedule that is more satisfactory to students and teachers. It should be noted that the third soft constraint will cause the timetabling model to not guarantee complete optimality of timetables, with complete optimality being the state in which no soft constraints are ever violated.

1.3 Algorithms Used In Approaches For Solving the University Course Timetabling Problem

Great Deluge

Many previous works dealing with course timetabling utilize an optimizing heuristic (or derivatives of it) called **Great Deluge**. Great Deluge was introduced by Gunter Dueck, and is a heuristic that is similar to Hill Climbing and Simulated Annealing. To understand how it works, imagine that you are in a point in some area with mountainous terrain. This area constitutes your solution space, with higher points in the area having higher values and, thus, having a better solution. The initial point you are located in in the area represents the initial solution that is generated. Imagine as well that it is raining endlessly and the water level W is continuously rising at a constant rate R_w, where R_w. W can start at any value that is greater than 0. Assuming that we are attempting to maximize some function Q, which evaluates a solution based on some criteria, your goal is to locate the relatively highest point in the location. This point can be thought of as the local optimum. Locating the highest point involves walking around the area that is not below the current water level. This

will force you to walk to a higher and higher point since W is constantly rising. Once you are no longer able to proceed to a higher level, it means that you are now in a local optimum. Going outside the analogy and back to a technical perspective, this local optimum would now be the *relatively* best solution for your problem. Every "walk" or move to a higher point is nuanced relative to the analogy. A single walk means construction of a new solution S_{new} with basis on the current solution $S_{current}$. If $Q(S_{new}) \geq W$ [**?**], then we accept S_{new} as the new current solution and "walk" towards it, and we increase W by R_w. Otherwise, we simply generate a new S_{new}. These walks are performed until $Q(S_{new})$ is not greater than $Q(S_{curr})$ for a long time or we have reached the maximum number of moves/iterations [**?**]. Great Deluge can also be adapted to minimize Q. Instead of W increasing, it will be decreasing by the same rate. A solution S will now be accepted if $Q(S) \leq W$. Conversely, the algorithm will stop when $Q(S_{new})$ is not lesser than $Q(S_{curr})$. The second stopping condition for the algorithm still applies in this case [**?**][**?**][**?**]. This minimization case is the one adapted by Great Deluge-based works that focus on course timetabling.

Genetic Algorithms

Great Deluge is a metaheuristic [**?**] that has been used for solving course timetabling problems [**?**][**?**][**?**]. Aside from Great Deluge, another metaheuristic that has seen use in the course timetabling problem and its variations is the Genetic Algorithm. The genetic algorithm is an optimization algorithm whose behaviour is based on how nature works, particularly on how reproduction works at a genetic level. Many implementations of the genetic algorithm for course timetabling represents the problem by having each gene in the genetic representation of the timetabling be a two-element tuple which consists of the class and the agents that will partake in that said class

[?][?][?]. Some use another representation. In the algorithm, an initial population is first generated. This population does not necessarily contain the locally optimal solution but it is where the relatively best solution will be obtained from. This initial population is referred to as the first generation. From this population, a certain number of individuals will be randomly selected to be bred with one another and be the parents of the next generation. Selection of individuals is dependent on an individual's fitness, with the most fit individuals usually being selected. Calculating this fitness is dependent on the problem. In the context of course timetabling, fitness is based on the constraints that have been violated by the current solution/timetable generated [?][?][?][?][?][?][?]. The breeding process involves selection of parents and having genes from the parents crossover and/or mutate to produce new offspring. Crossover is when genes from both parents are combined to produce an offspring. On the other hand, mutation is done by changing a random gene from either parent to create an offspring. Determining which genes from either parent to apply onto the offspring and which to mutate is dependent on implementation. Once a population of new generations is established, the cycle repeats. This continuous reproduction of generations eventually produces solutions "moves" towards the optimal solution [?]. Despite being able to produce feasible solutions for university timetabling, it should be noted that using a genetic algorithm approach may require more time executing compared to other approaches due to its population-based property. When compared to simulated annealing, another approach for university timetabling, the genetic algorithm takes more time executing [?]. However, approaches utilizing the genetic algorithm can see an improvement in execution time when they utilize graphics processing units (GPUs). The work of Yousef, A., Salama, C., Jad, M., El-Gafy, T.,

Matar, M., and Habashi, S. showed that it is possible to speed up genetic algorithms using GPUs. In their work, they accelerated the computation of the fitness function. Their experiments show that execution speed can be improved by up to 59 times in very large problem instances and by 280% overall when utilizing the GPU [**?**].

Chapter 2

Review of Related Literature

The polygonal bounding area building placement problem (PBABPP) deals with the arrangement of buildings within a polygonally-shaped area. As far as the authors know, no previous research has been conducted for the problem. More so with the fact that this research also takes into account areas prone to natural hazards, namely flooding and landslides. Nevertheless, PBABPP is still an extension of the facility layout problems (FLP), which delves with determining the placement of various assets in a facility. Many techniques utilized in solving FLP instances can be adapted in PBABPP. As such, this literature review will mostly consist of prior works that attempt to solve facility layout problems. Majority of the FLP works included here utilize approximation methods since this work uses one. FLP researches are also easy to find thanks to the fact that it is NP-Hard. Being NP-Hard resulted in numerous researches being done for the field [1].

All research works in the literature produce layouts with varying degrees of fitness and performance. The process of layout generation differ from proposal to proposal based on the specific FLP instance they are working on. Various techniques are used to solve the FLP. Exact methods have been used, but stochastic-based methods like

local search and population-based evolutionary algorithms are popular.

The instance of the facility layout problem that is most related to this work is the unequal area static facility layout problem (UA-SFLP). The works dealing with the UA-SFLP (and even the unequal-area dynamic facility layout problem) use a rectangle to mark the bounds of the area where assets can be placed. This is unlike the problem we are solving here where a polygonal area is used instead.

Chapter 3

Statement of the Problem

Raster images are composed of a pixel matrix. This makes their data representation simple. However, this reduces the amount of detail and quality they have. This limitation is clearer when scaling raster images. This motivates the use of an alternative form of representing images. One form is vector images, which uses mathematical equations to represent an image. The process of converting a raster image to a vector image is called *vectorization*. Semi-structured imagery, such as those used in graphic designs, is one of the classes of images that would benefit from vectorization. This process will allow such images to be easily scaled without sacrificing quality nor detail.

There have been numerous works that tackles image vectorization for semi-structured images. Many of these works primarily use a curve optimization algorithm such as variants of NEWUOA and conjugate gradient. A machine learning approach has been used in one of the works, but as a preprocessing step only [**?**]. Deep learning have been used to solve problems in multiple domains such as natural language processing (NLP), object detection, and playing board games. No other known work has applied deep learning as a core step in image vectorization. This study will deal with such application.

Chapter 4

Objectives

This study is primarily aims to apply deep learning via artificial neural networks to the problem of vectorizing semi-structured imagery. However, there are still some key objectives that this study seeks to accomplish:

1. To develop an approach that considers Gestalt psychology.
2. To evaluate the effectiveness of using deep learning for image vectorization.
3. To evaluate the accuracy of results obtained from the proposed approach to the target raster inputs.
4. To evaluate and compare the results of the proposed approach to that of previous semi-structured image vectorization methods.
5. To evaluate and compare the speed of the proposed approach compared to previous semi-structured image vectorization methods.

Chapter 5

Proposed Methodology

The proposed methodology will be based on the frameworks proposed by Hoshyari, S., et. al. [**?**], Yang, M., et. al. [**?**], Xiong, X., et. al. [**?**], and Laube, P., et. al. [**?**]. Additionally, human perception will also be taken account. Thus, the Gestalt psychology principles of accuracy, simplicity, continuity, and closure, as taken from Hoshyari, S., et. al., will be taken into account as well.

5.1 Image Preprocessing

Before a raster input image can be fitted with curves, it must preprocessed to simplify the vectorization procedure and align it with .

Corner Detection

The first step in the approach is detecting the corners in the input image. This is an important step as it will allow us to enforce the simplicity principle in Gestalt psychology and make sure the resulting vectorization be C^0 continuous should the raster input be as such.

This stage will be based from the corner detection classifier of the work by Hoshyari, S., et. al. [**?**]. A random forest classifier is used in the said work. Supervised learning is used as corners are manually annotated. Annotated corners will not be specific pixels. Rather, corners will be between at least two pixels. Training data is available publicly provided by the researchers. However, such data is limited only to quantized data (i.e. aliased data). The target raster input is expected to be anti-aliased data. As such, the training data will have to be built from scratch to support anti-aliased data.

Region Segmentation

In line again with the simplicity principle, the input image must be divided into regions. This will result in simpler curves being used in the final vectorization output. The additional benefit of segmenting the input into multiple distinct regions is the possibility for parallelism to be used in the vectorization approach. Since each region is distinct and independent from one another, multiple regions can be vectorized at the same time. Thus, speeding up the vectorization process.

Due to the nature of semi-structured imagery where each region will only contain a single colour, a scanline-based approach can be used in this stage. The scanline algorithm will be based off of the scanline algorithm used for boundary pixel detection in the work of Xiong, X., et. al..

For each line l_i, where $0 \leq i < h \mid h$ is the height of input image, in the raster input, each pixel p_i will be assigned to a pixel set r_{ij}, where j is the index to a set in l_i. Each pixel set will contain horizontally adjacent pixels that have the same or near-same colours. We include pixels whose colours are within a certain threshold k from the colour of the pixels that is most prominent in the set. This threshold is

necessary due to the fact that certain parts of a regions may contain an anti-aliased pixel. Each l_i will have a pixel set vector r_i containing all pixel sets of l_i:

$$r_i = (r_{i0}, r_{i1}, \ldots, r_{i(j-1)}, r_{ij})$$

Consequently, a region vector r will contain all pixel set vectors.

$$r = (r_0, \ldots, r_i)$$

Note that there is an opportunity to utilize parallelism, as shown in the work of Xiong, X., et. al. [**?**], during this step due to the independent nature of every line in the raster input.

Once we obtain all the pixel sets for each line, we iterate through r, $(h-2)$ times. For each iteration, we process r_i and r_{i+1}. If there are any r_{ij_α} whose pixels are vertically adjacent and have the same colour (or within k) to another pixel set $r_{(i+1)j_\beta}$, then those two pixel sets are merged into one. By the end of the iterations, we have obtained a set of regions that we can individually vectorize.

5.2 Curve Approximation

The core step of the proposed approach is curve approximation. This stage fits curves to the region boundaries of the raster input. This stage is based on the work by Laube, P., et. al. on curve approximation on point sequences using deep learning [**?**].

Point Sequence Generation

The work of Laube, P., et. al. takes a point sequence as input. As such, for this proposed approach, we must generate point sequences from the region we will be

vectorizing. For every region we ought to vectorize, we treat the center of boundary pixels and the detected corners of each region as a point sequence. However, we must also take into account the fact that certain portions of a region may be a corner where the curves in such segment would have C_0 continuity. As such, the point sequence we generate must take into account corners. This would implore us to take note of the following cases during point sequence generation:

1. For regions with no corners, a random pixel will be selected as both start and end point of the point sequence. The expectation is that there will be no difference in the resulting curve from choosing a different start and end point.

2. For regions with a single corner, the corner point will be selected as the start and end point of the point sequence. This is to ensure that the resulting vector output will have a corner at that point.

3. For regions with two or more corners and assuming n is the number of corners, the point sequence will be divided at those corners into separate point sequences. This will result in $(n + 1)$ new point sequences. Each new point sequence will have their start and end points be the corner points they are adjacent to.

Each point sequence will then be passed to the next stage to be parametrized and have a curve approximated for.

Point Sequence Curve Approximation

The point sequences obtained from the previous step are now to be fitted with curves, specifically B-splines. As provided by the framework by Laube, P., et. al., two neural networks will be used in this stage and some preprocessing will be performed on

the point sequences. The two neural networks are a point parametrization network (PPN), which approximates parametic values to point sequences, and a knot selection network (KSN), which predicts new knot values for knot vector refinement.

Sequence Segmentation

The input point sequence must first be split. This is to ensure that real data and training data match in terms of complexity. Let us define a function $\hat{k}(p)$ that measures the complexity of a point sequence p, where k_i is the curvature at point p_i, given its total curvature:

$$\hat{k}(p) = \sum_{i=0}^{m-1} \frac{(|k_i| + |k_{i+1}|)||p_{i+1} - p_i||_2}{2}$$

A point sequence p is split into point sequence segments $p^s, s = 1, \ldots, r$ at the median, if $\hat{k(p)} > \hat{k_t}$ for a threshold $\hat{k_t}$. This $\hat{k_t}$ will be set, as per the original authors have done, to the 98th percentile of $\hat{k}(\cdot)$ of the training set. This process is performed $r - 1$ times, until each p^s satisfies $\hat{k(p)} > \hat{k_t}$.

Sub/Supersampling and Normalization

To be able to approximate parametric and knot values using the PPN and KSN, the number of points per segment p^s must equal the input size l of the aforementioned networks. As such, all segments p^s are either subsampled or supersampled.

If a segment p^s has a number of points greater than l, then p^s is subsampled. This process involves drawing points in p^s such that the drawn indices i are equally distributed and include the first and last point. If, on the other hand, the number of points in p^s is less than l, then *temporary* points are linearly interpolated between consecutive points p_i^s and p_{i+1}^s. This interpolation is performed until the number of

points equal l.

The sampled segments are then normalized to $\bar{p}^s$, which consists of the points

$$\bar{p}_i^s = \frac{p_i^s - min(p^s)}{max(p^s) - min(p^s)}$$

where $min(p^s)$ and $max(p^s)$ are the minimum and maximum coordinates of p^s respectively.

Parametrization of Point Segments

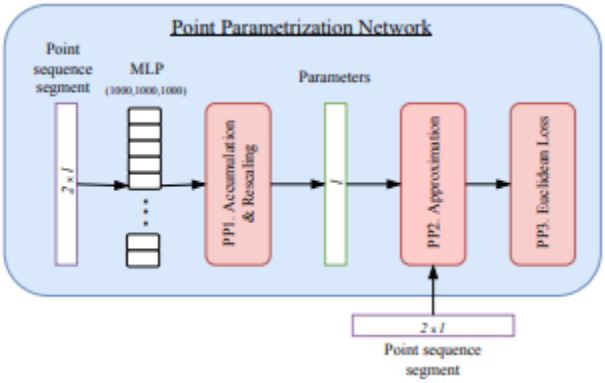

Figure 5.1: The network architecture of the Point Parametrization Network (PPN)

.

The PPN will be responsible for parametrization of point segments. For every $\bar{p}^s$, the PPN generates a parametrization $\bar{t}^s \subset [0, 1]$, which will be rescaled to $[u_{s-1}, u_s]$ and adapted to sampling of $\bar{p}^s$.

In supersampled segments, the parameters t_i^s of temporary points are simply removed from $\bar{t}^s$. In a subsampled p^s, for every point p_i that was removed from the segment, a parameter $\bar{t}_i$ is inserted to $\bar{t}^s$.

$$t_i = t_\alpha^s + (t_\beta^s)\frac{chordlen(p_\alpha^s, p_i)}{chordlen(p_\alpha^s, p_\beta^s)}$$

chordlen is the length of the polygon defined by a point sequence. In the sub-sampled segment, with parameters t^s_α and t^s_β, p^s_α and p^s_β are the closest neighbours of p_i.

The initialization of the parametric step requires an initial knot vector. We first define $u_0 = 0$ and $u_n = 1$. For each segment (except the last one), one knot u_i is added.

$$u_i = u_{i-1)} + \frac{chordlen(p^s)}{chordlen(p)}, i = 1, \ldots, r-1$$

This yields a start and end knot for every point sequence segment.

The PPN Architecture The PPN, as stated earlier, takes in an input of segments p, which can be written as $p = (x_0, \ldots, x_{l-1}, y_0, \ldots, y_{l-1})$. The parameter domain is defined as $u_0 = t_0 = 0 and u_n = t_{l-1} = 1$. For a sequence of points p, a parameter vector $t = (t_i)_i$, is defined as $t_i = t_{i-1} + \Delta_{i-1}$. The task of the PPN is to predict missing values $\Delta = (\Delta_0, \ldots, \Delta_{l-2})$ with

$$\Delta_s ubi > 0, i = 0, \ldots, l-2$$

such that $t_0 < t_1$ and $t_{l-2} < t_{l-1}$. We apply a multilayer perceptron (MLP) to the input data p, yielding as output a distribution for parametrization $\Delta^{mlp} = (\Delta^{mlp}_0, \ldots, \Delta^{mlp}_{l-2})$ of size $l-1$.

The PPN further contains additional layers PP1, PP2, and PP3, which will be discussed next.

PP1. Accumulation and Rescaling The output Δ^{mlp} is used to compute a parameter vector t^{mlp} with $t^{mlp}_0 = 0$ and

$$t_i^{mlp} = \sum_{j=0}^{i-1} \Delta_j^{mlp}, i = 1, \ldots, l-1$$

Since t_{l-1}^{mlp} is usually not 1, rescaling t^{mlp} yields the final parameter vector t with

$$t_i = \frac{t_i^{mlp}}{max(t^{mlp})}$$

The MLP layer in the PPN uses a softplus activation function defined to be:

$$f(x) = ln(1 + e^x)$$

PP2. Approximation B-spline curve approximation is included directly into the PPN as a network layer. The input points p and their parameters t are used for an approximation with knot vector $u = (0, 0, 0, 0, 1, 1, 1, 1)$ for $k = 3$. The approximation layer's output $p^{app} = (p_0^{app}, \ldots, p_{l-1}^{app})$ is the approximating B-spline curve evaluated at t.

PP3. Euclidean Loss A loss function, which is a Euclidean loss function, is to be used in the PPN. The Euclidean Loss function is defined as

$$\frac{1}{l} \sum_{i=0}^{l-1} ||p_i - p_i^{app}||_2 \tag{5.2.1}$$

Parametrization Refinement

In some cases, the approximated parametrization of a p^s may have errors. The approximation error of a p^s is computed by using the Hausdorff distance to the input data p.

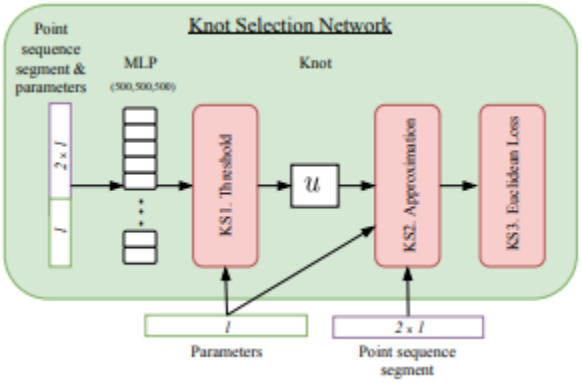

Figure 5.2: The network architecture of the Knot Selection Network (PPN)

.

The segments p^s that have a large approximation error are computed a new knot using the KSN. For $\bar{p}^s$ and $\bar{t}^s$, the KSN generates a new estimated knot $\bar{u}_s \in [0, 1]$. The new knot $\bar{u}_s$ is mapped to the actual knot value range $[u_{s-1}, u_s]$ by

$$\tilde{u}_s = u_{s-1} + \bar{u}_s(u_s - u_{s-1})$$

Instead of $\tilde{u}_s$, the parameter value t_i closest to $\tilde{u}_s$ is inserted into u. u can be further refined until the desired curve approximation error threshold is satisfied.

The KSN Architecture This stage utilizes a KSN, as mentioned earlier. The KSN predicts a new knot u to the interval $(0, 1)$ for a given segment p and parameters t, of which were previously approximated by the PPN. This network uses an MLP, which transforms the input to a single output value u^{mlp}. The RELU function is used as the activation function for the MLP, except for the output layer where the Sigmoid function is used instead.

Similar to that of the PPN, the KSN has three additional layers: KS1, KS2, and KS3.

KS1. Threshold Layer The new knot u computed has to satisfy the following conditions: $u \in (0, 1)$, $t \cap [0, u] \neq \theta$, and $t \cap [u, 1] \neq \theta$. Satisfying these conditions will will require us to use a threshold layer which maps u^{mlp} to

$$u = \begin{cases} \epsilon & , \text{if } u^{mlp} \leq 0 \\ 1 - \epsilon & , \text{if } u^{mlp} \geq 1 \\ u^{mlp} & , \text{otherwise} \end{cases}$$

Introducing a small $\epsilon = 1e - 5$ makes sure that the knot multiplicity at the end knots stays equal to k.

KS2. Approximation Approximation in the KSN is generally similar to that of PPN. The only different is that of the knot vector. The knot vector in the KSN is defined to be $u = (0, 0, 0, 0, u, 1, 1, 1, 1)$. For backpropagation, the derivative of the B-Spline basis functions with respect to u is required.

KS3. Euclidean Loss The loss function for the KSN is the same as that of the PPN. See 5.2.1.

Network Training

The training of the PPN and KSN will be based from the work of Laube, P., et. al. [**?**]. The input size of the network will be $l = 100$.

The data set generated by the original authors consists of 150,000 curves. This data was synthesized from B-spline curves. Random control points c_i were generated using a normal distribution μ and variance δ to define cubic ($k = 3$) B-spline curves with $(k + 1)$-fold end knots and no interior knots. The y-coordinates are given the configuration: $\delta = 2$ and $\mu = 10$. For the x-coordinates, $\delta = 1$ and $\mu = 10$ are used for

the first control point. All consecutive points have μ increased by $\Delta\mu = 1$. Curves with self-intersections are discarded, because the sequential order of their sampled points is not unique, and point sequences are usually split into subsets at the self-intersections. Smaller δ for the x-coordinates of control points reduces the number of curves with self-intersections. To closely match the target input as much as possible, we also include curves that have been manually fitted to raster images. These curves can be obtained from vector images available online.

For each curve, l points $p = (p_0, \ldots, p_{l-1})$ are sampled. These curves then to have increasing x-coordinates from left to right. As such, index-flipped versions of the point sequences of the dataset are added, resulting in 300,000 point sequences. 20% of the sequences are used as test data in the training process.

The PPN is trained first since the KSN requires point parametrizations t, which is obtained from the PPN. After training, the PP2 and PP3 layers are discarded and PP1 becomes the output layer of the PPN. The parametric values t are computed for the training dataset by applying the PPN and train the KSN on the combined input. After training, KS2 and KS3 are discarded, with KS1 becoming the network output layer. The MLPs of the PPN and KSN will consist of three hidden layers with sizes (1000, 1000, 1000) and (500, 500, 500) respectively. Dropout is applied to the MLP layers. The network is trained using the Adam optimizer.

5.3 Region Colouring and Merging

Once the curves have been approximated, the vectorization of the region will be filled with the colour prominent in the raster version of the region. The regions will be plotted unto their locations in the original raster input. Once all the regions have

been plotted, they will be grouped into a single vectorization. This will now be the vectorization output of the raster image.

Chapter 6

Describing How You Validated Your Approach.

Chapter 7

Stating Your Results and Drawing Insights From Them.

Chapter 8

Summarizing Your Thesis and Drawing Your Conclusions.

Appendix A

What should be in the Appendix

What goes in the appendices? Any material which impedes the smooth development of your presentation, but which is important to justify the results of a thesis. Generally it is material that is of too nitty-gritty a level of detail for inclusion in the main body of the thesis, but which should be available for perusal by the examiners to convince them sufficiently. Examples include program listings, immense tables of data, lengthy mathematical proofs or derivations, etc.

Bibliography

[1] Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007.