中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

基于 DXF 格式的板材优化下料和刀具路径算法的研究

作者姓名：毛良献

指导教师：王品 研究员 中国科学院沈阳计算技术研究所

学位类别：工学硕士

学科专业：计算机应用技术

培养单位：中国科学院沈阳计算技术研究所

2019 年 6 月

Research of Cutting Stock and Tool Path Algorithm Based on DXF Format

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Science in Engineering

in Technology of Computer Application

By

Mao Liangxian

Supervisor： Researcher Wang Pin

Shenyang Institute of Computing Technology，Chinese Academy of Sciences

June 2019

摘 要

现今，计算机科学技术的发展势头凶猛，对周边传统企业带来了巨大压力，同时也给他们带来了机遇。传统企业想方设法利用科学技术这把利剑，来改进生产设备，提高材料利用率，以创造企业利润。在工业制造领域，一直存在着两个重要的问题。一个是板材排样问题，另一个是切割路径问题。板材排样问题的有效解决能极大提高材料的利用率和生产效率，而切割路径问题的解决是数控自动化加工的重要基础，对实现机械自动化有重要的意义。本文以 DXF 文件的读取与解析作为研究的切入点，重点研究了排样问题和切割路径问题，并提出了相关的解决思路和算法实现。通过对算法的测试分析，论证了算法的可行性。

对于 DXF 文件的读取与解析，采用了表驱动的处理模型。针对 DXF 文件中图元存储的不一致性，做了优化排序处理，并实现数据的转化与存储，为后续的板材排样问题和切割路径问题提供了数据支持。

在板材排样问题中，先介绍了一些计算几何的基础算法，接着叙述了基于矩形件的各种排样算法，如 BL 算法，下台阶算法等。最后提出了一种基于 NFP 和遗传算法的排样思路。其中 NFP 结构用于不同部件的不重叠的排放，而遗传算法用于全局优化。

在切割路径问题中，先介绍了一些求解算法，如最近邻算法和基于二叉树优化的最近邻算法，并分析了它们的优缺点。接下来，把切割路径问题转化为 TSP 问题，利用改进的蚁群算法对 TSP 问题求解，取得较优的结果。

关键词：板材排样和切割路径，DXF，NFP，遗传算法，蚁群算法

Abstract

Nowadays, the development trend of computer science and technology is rapid, which brings pressure and opportunities to the surrounding traditional enterprises. Traditional enterprises try their best effort to improve production equipment, improve the utilization of materials and make profits by using the power of science and technology. Two important problems have existed in the field of industrial manufacturing for many years.One is the nesting problem, the other is the cutting path problem.Effective solution of nesting problem can improve material utilization and production efficiency greatly.The solution of cutting path problem is an important foundation of the automatic of NC machining and it is of great significance to realize the automation of machining.In this paper, DXF file's parse is the starting point of the study, and focus on the nest and cutting path problem, and present relevant solutions and implementation of algorithm.Through the test and analysis of the algorithm, the feasibility of the algorithm is demonstrated.

For DXF files parsing, a table-driven model is adopted. Because of inconsistency storage of primitives in DXF files, an optimized method is proposed and the data conversion is achieved, which provides support for nest and cutting path problem.

In the problem of nest problem, some basic algorithms of computational geometry are introduced.Then, various nesting algorithms based on rectangular parts, such as BL algorithm and walk down algorithm, are described. Finally, a new nesting method based on NFP and genetic algorithm is proposed. The construction of NFP is used for non-overlapping placement of different components, while genetic algorithm is used for global optimization.

When talking about cutting path problem, some algorithms are introduced. Such as nearest neighbor algorithm and optimal nearest neighbor algorithm based on binary tree, whose advantages and disadvantages are analyzed. Next, we transform the cutting path problem into TSP problem, and use an improved ant colony algorithm to solve the TSP problem.

Key Words: Packing and Cutting Problem, DXF, No-Fit Polygon, Genetic Algorithm, Ant Colony Algorithm

目 录

图例索引

表格索引

第 1 章 引言

1.1 课题研究的背景及意义

现今，计算机科学技术的发展势头凶猛，对周边传统企业带来了压力的同时，也带来了机遇。一些传统企业不得不冒着风险进行技术革新，赶上当今的科技步伐。物联网，云计算，大数据，人工智能，深度学习，3D 打印等等无不刺激着传统行业。他们想方设法的利用这些工具(技术)来发明最好最高效的生产工具，以此让原料的利用率达到最大，让工业废料极大减低，让企业利润极大上升。对于传统行业中的工业制造业来说也是一样。当前，工业制造的根本原则就是减低企业的生产成本，提高企业的生产效率，提高企业在业界的知名度，从而达到企业利润的快速增长。数控制造是工业制造行业的一个典型行业。在数控行业，原料费用是企业花销中的一个重要组成部分。那么，提高原料的使用率显然是减低企业费用的一种途径。因此，如何提高原料的使用率成为了企业时时刻刻都在思考的问题。本论文把以板材(泛指)作为基础的数控切割领域作为研究领域，把板材的利用率以及数控切割的效率问题作为研究对象，提出可行的实现方案。

板材的利用率问题也可以表述为板材排样问题。板材排样问题是这样描述的：将不同数量和形状的部件(零件)布局在一张或多张的板材上，使得板材的利用率最大化。板材排样在工业领域的应用很多，尤其需要将大板材分割成多个不同形状的小部件的行业，如造船业、航空航天等。目前在多数的企业中，大部分的排样工作采用人工手段，依赖工人的生产经验，这不仅导致工人的工作量大，同时工人的工作效率和材料的利用率也不得保证。针对这种状况，许多企业都希望借助计算机辅助设计(CAD)技术，寻找高效、通用的板材排样系统。而在板材排样中，最重要的就是排样算法的实现，这也决定了板材排样系统的优劣。

另一个需要考虑的是切割路径问题。切割路径问题是这样描述的：控制刀具的移动，使得在切割过程中，缩短空行程和切割时间。在如今的工业制造领域中，切割技术仍旧占据一个重要的地位。切割不仅需要考虑刀具的损耗问题、寿命问题，同时也要考虑被切割物件的质量问题，切割时间问题等。其中，刀具路径问题是一个相对重要的问题。在切割的工艺规划中，需要根据零件的设计图纸确定

零件的加工顺序，即路径切割问题。切割路径问题包括图元的整合和最短切割路径的寻找。并且路径的走刀顺序对板材的切割质量有很大的关系，切割路径的优化对切割时间(效率)有很大的改善。

针对上述提出的排样问题和切割路径问题，本课题进行了相关的技术研究。由于板材的选材问题而言，可以有不同的选择，例如皮革、钢材、木材等。而刀具的类型也有多种多样，例如有铣刀、激光、电火花等。为了研究的方便性，在下面出现的板材都采用一般化处理，也就是认为板材是一个二维的平面，材质不确定。而刀具也做一般化处理，即对切割所用的刀具不考虑其实际的材质和损耗，仅把其当做一个可以做切割的物件。通过这样的一般化处理，那么本文提出的排样算法和切割路径算法是一种理想的情况，在实际应用的使用需要做其他的修正处理。总之，研究排样问题和切割路径问题对企业而言就由十分重要的意义。

1.2 国内外排样技术研究现状

排样问题，按照维度的不同，可以有多种形式。通常可以分为 3 类排样问题，也就是 3 种不同的维度，即一维的排样，二维的排样和三维的排样。而维度超过 3 维的问题很少见，但是也是存在，只不过一般有其他问题转化而来。一维的排样可以看做是对给定长度的材料进行切割，这类问题也经常引用为段带排样问题。在这种问题中，只需要考虑的就是材料的长度。材料的宽度一般是不需要考虑或者是固定的。典型的一维排样问题如钢条的切割。

许多二维的排样问题也可以转化为一维问题，只需要忽略被切割材料的宽度，因此材料的长度在被转化后就是我们所关注的。在二维的排样问题中，我们会从二维的板材上切割二维的部件。这些部件可以是规则的矩形，也可以是不规则的。二维排样中需要就需要考虑材料的宽度了。典型的问题如钢板的切割问题，玻璃切割等。最后对于三维排样问题，也就是在有限的三维空间中封装若干个三维的部件。装箱问题是其中一种。对于本文而言，着重研究的是二维排样。对于二维排样而言，又可以细分为两类：(1)对矩形件的排样，(2)对不规则形状的排样。在第一类问题中，所有需要排样的部件都是规则的矩形件。针对这种问题，已经存在许多的算法，常见的有 BL 算法、下台阶算法等。而对于第二类问题，由于需要排样的部件不规则，通常采取的方法就是矩形包络法。这样就转化为第一类问题而可以求解。但是，矩形件的排样问题是个复杂度很大的问题。若按 n 个矩

行件的进行排样，矩形件的排放种类可以达到 n！中，并且每个矩形件可以横着排，或者竖着排，或者斜着排等等。那么这 n 个矩形件可以达到的排放方式有 $n!*(T_1*T_2*...*T_n)$ 种，T_i 表示的是某个矩形件的排放形式。可以看出，若要遍历求解是非常耗时的。

国内外有不少的专家学者在排样问题中做出了很多研究。最早可以追溯到 20 世纪 40 年代，Brooks 等(1940)就发表了一篇论文，讨论如何将一块矩形板材分割成多个矩形部件。接下来的一段时间没有再出版任何关于板材排样的论文。直到 20 世纪 50 年代，Dantzig(1951)和 Kantorovich(1951)提出了排样问题和线性规划之间的理论关系。之后，Paull(1956)， Eisemann(1957)， Vajda(1958)利用线性规划技术将切割卷筒纸问题转化为矩形布局问题，并得出一个受限的解决方案。到 20 世纪 60 年代，Glimore 和 Gomory(1961)利用线性规划，首次得到一维排样问题的最优解。接下来，Glimore 和 Gomory(1965)也提出了解决二维排样问题的方法。他们的方法也是利用了线性规划技术，可是基于该模型的维度过大(列数过多)，解决问题的能力有限。后续有许多学者提出了动态规划算法、树搜索方法等。在 80 年代，Coffman(1980)等给出了两个新的算法，Next Fit Decreasing Height(NFDH)和 First Fit Decreasing Height(FFDH)算法，这两个都对矩形部件进行排样，按高度顺序排列，并把他们放置在各行中。到了 20 世纪 90 年代，为了满足板材材料的利用率和切割效率的要求，排样问题的启发式搜索算法成为了人们的研究热点。Jain 等(1992)提出了模拟退火算法、Blazewicz 等(1993)提出了 tabu 搜索算法、kroger(1995)提出了遗传算法。这三种算法在一些问题的求解中可以得出较好的可行解。当前，国外的研究的热点主要在于各种算法之间的结合使用。

国内研究套料算法开始于 20 世纪 80 年代。早期大多研究矩形件的排样算法，研究的热点主要在于动态规划、线性规划、遗传算法和模拟退火算法。到如今，国内研究出现了一些成果。张青等人(2017)为解决婚纱冲印的人工排样效率低，利用率差等问题，提出了一种专家模板的照片自动化排版方法，比人工排样高出 4.3%，但工作效率极大改善。罗强等人(2018)提出复合评价因子对矩形进行评价，选择较优的矩形先进行排样。并合理使用遗传算子，改善算法的局部搜索能力，提高了矩形件的排样的质量。扈少华等人(2018)构造了一个基于五块结构的矩形

件套裁方式，通过实验表明，该方法能有效提高板材利用。

总而言之，对于矩形件的排样问题，国内已经取得许多理论成果，但对于不规则件的排样大多数是将不规则件按一定的规则处理为矩形件，并将问题转化为矩形件的排样问题。由于排样问题是一个 NP 完全为题，无论是矩形件或者不规则件的，仍然没有一个公认的最有效的排样算法。现行的套料软件也没能完全取代各行业传统人工套料的方式。因此，优化套料算法依然是一个热点。

1.3 国内外切割路径技术研究现状

切割路径问题，在制造领域也是一个着重研究的方向。对于排样完成的部件，找到一个最优的切割路径来减少切割时间是很必要的。因此，切割路径的算法效率对于工业的生产的重要作用不言而喻。切割路径技术的发展及其在各个领域的应用形成了不同的切割方法，如水射流切割、电火花切割和绳锯切割等。切割路径技术是各种数控切割系统需要解决的共同问题，需要解决的问题主要包括图元的有序化处理、加工方向的判断、轮廓关系的判断、以及全局切割路径的优化方面的问题。

国外研究切割路径问题的时间相对较早。早在 1995 年，Fischetti 等人(1995)研究了对称的广义旅行商问题(GTSP)。其中有一个变体称为 E-GTSP，而这是切割路径问题的一般化描述形式。他们通过使用线性规划模型求解 GTSP 和 E-GTSP 问题，并研究了先对应的 polytopes 的表面结构。在 1997 年，Fischetti 等人(1997)又利用了分支界定算法来求解对称的广义旅行商问题，实验规模达到了 442 个节点。在文中，也描述了一些启发式算法。Korotaeva 等人(1997)使用了动态规划来求解路由问题。接下来，Snyder，L.V.和 Daskin，M.S.(2006)使用了启发式算法，该算法把遗传算法和局部优化的路由启发算法结合起来。最后论文测试了 41 个标准问题，论证了在绝大多数问题中能找到最优解，且计算时间稳定在 10 秒之内。Silberholz，J 和 Golden，B(2007)同样使用了遗传算法的变体来求解 GTSP 问题，而 Chentsov A.G.和 Chentsov A.A.(2013)利用动态规划技术求解路由问题。Arango Serna，M.D.和 Serna Uran，C.A.(2015)提出了一种 memetic 算法来求解 TSP 问题，Petunin AA 等人(2015)则结合动态规划和启发式算法来求解 CNC 的路由问题，同样将其推广到了 GTSP 问题。

国内的研究大致开始于 21 世纪初。早在 2005，高伟增等人(2005)利用了遗

传算法求解加工路径，对 75 个零件进行了排样处理并仿真计算，得出较优的计算结果。在俞武嘉等人(2006)的刀具路径优化排布论文中，同样适用了基于遗传算法的一种优化排布方法。李泳等人(2007)研究的复杂轮廓激光切割路径算法中，利用图论原理将加工路径归结为广义旅行商问题，将轮廓位置关系构造成树形结构，提出了从内到外遍历树形结构的优化算法。曹德列(2012)在对不规则图形的切割研究中，主要利用了数字图像相关的知识，提取图像中的轮廓并转化为 DXF 格式。

从国内外的研究现状可以看出，切割路径的优化一直是制造行业的持续关注的问题。通过将这些问题归结为 GTSP 问题，并绝大多数都使用遗传算法来求解。

1.4 本文的主要工作内容

本课题提出解决排样问题的一种排样算法和解决切割路径问题的一种切割路径算法。各章的主要的如下：

第一章是引言，主要介绍本课题的研究背景和意义，以及课题相关技术国内外的研究现状。

第二章是 DXF 文件格式分析。DXF 文件作为 CAD 的一种常用数据交换格式而广泛的使用。在这章中，主要分析 DXF 文件的文本格式和其数据的提取过程，为后续的排样和路径切割提供数据支持。数据的提取包括图形的有序化处理，顺逆方向的判断等。

第三章是排样算法。排样算法有两个子算法构成。两个子算法分别是 NFP 构造算法和遗传算法。NFP 构造算法解决了不同部件的不重叠的排放，而遗传算法作为一种启发式的优化算法，对排样结果进行优化处理。同时，介绍了一些其他算法的优缺点，例如 BL 算法，下台阶算法等。最后，我们对 NFP 算法进行的验证，分析了其可行性。

第四章是切割路径算法。切割路径算法也分为两个过程。首先是如何将切割路径问题转化为一个 TSP 问题。接着利用改进的蚁群算法对 TSP 问题求解。最后，同样对蚁群算法进行了验证处理，分析了其可行性。

第五章是实验结果。通过上面几章的算法设计、实现以及验证，在本节给出相关的实验实例。

第六章是论文的工作总结和展望。

第 2 章 DXF 文件格式分析

2.1 DXF格式介绍

DXF 的全称是 Drawing Interchange Format，由 Autodest 公司开发的用于 AutoCAD 与其他软件之间进行数据交换的文件格式。DXF 是一种开放的矢量数据格式，有文本和二进制格式。因为文本格式比起二进制格式更加通用，所以术语 DXF File 引用的是文本格式的 DXF（ASCII Drawing Interchange Format File），而术语 Binary DXF File 引用二进制格式。本课题以文本格式作为研究对象，下文介绍的格式都是文本相关的格式描述。

本质上，一个 DXF 文件是多对键值对组合。这里的键称为组代码，用于描述值的类型和值的用途(有些组代码对应的值的作用是确定的)。组代码（键）和值组成键值对，多组键值对生成一个 SECTION，多组 SECTION 构成一个 DXF 文件。在后文描述键值对时，使用如下符号“<GroupCode，Value>”来表示一个键值对。一个 DXF 文件由多个 SECTION 构成。每个 SECTION 以键值对<0，SECTION>开头，以键值对<0，ENDSEC>结尾。对于一个特定的 SECTION，里面存储的是特定的键值对。如有的 SECTION 用于存储制图有关的全局信息，有的 SECTION 用于记录针对应用的类信息等等。在 DXF 格式规范中，定义了 7 个 SECTION，分别是：HEADER，CLASSES，TABLES，BLOCKS，ENTITIES OBJECTS，THUMBNAILIMAGE。其中 HEADER SECTION 包含了包含了绘图的基本信息，以及 DXF 文件格式的版本号和系统变量等。CLASSES SECTION 包含应用特定的类信息，这些类实例可以出现在 BLOCKS，ENTITIES，OBJECTS 中。TABLES SECTION 存储了多个符号表，有 APPID，BLOCK_RECORD，DIMSTYLE，LAYER，LTYPE，STYLE，UCS，VIEW，VPORT。BLOCKS SECTION 存储了块的定义以及组成块引用的绘制实体。ENTITIES SECTION 包含图形化对象（ENTITY）和块引用。OBJECTS SECTION 由非图形化对象组成。THUMBNAILIMAGE SECTION 存储制图的预览图像数据。这里需要特别注意一下 ENTITIES 和 OBJECTS 的区别。在 DXF 格式规范中，对象（OBJECT）和实体（ENTITY）的性质是不一样的。对象的作用是存储一些不可以被显示的信息，例

如字典数据；而实体存储的是可以被制图软件所需要的绘制信息，例如线段的起点和终点，圆的圆心和半径等。而实体(ENTITY)可以存储在 BLOCKS SECTION(作为块引用的绘制实体)以及 ENTITIES SECTION(只作为图形化对象)。

2.2 DXF格式分析

前文说到 DXF 文件有多个 SECTION 组成，每个 SECTION 都存储不同的信息。下面以最简单的 HEADER SECTION 为例，分析一下 DXF 文件的基本格式。

0
SECTION
2
HEADER
9
$<variable>
<group code>
<value>
0
ENDSEC

图 2.1 HEADER SECTION 示例

Figure 2.1 HEADER SECTION sample

图 2.1 格式分析：(1)每行只存储一个数据。若一行存储键(组代码)，下一行就存储相对应的值。下下一行又是一个键，再是对应的值等等。键值对采用分行存储策略，对数据处理软件(CAD)的读取和存储数据都很便捷。(2)SECTION 以〈0，SECTION〉开头，以〈0，ENDSEC〉结尾，这在前文也描述了一下。在〈0，SECTION〉之后，出现了〈2，HEADER〉。这个键值对用于表明这是哪种 SECTION，这里就是 HEADER 类型的 SECTION。对于其他情况，可以是〈2，CLASSES〉，〈2，TABLES〉等。因为 HEADER SECTION 的作用是存储系统变量，故接下来是一些变量的定义。组代码 9 表示对应的值的类型是字符串类型，用途是定义一个变量名．如在下面这个例子中：

9

$ACADVER

1

AC1021

图 2.2 HEADER SECTION 中变量定义

Figure 2.2 variable definition in HEADER SECTION

这里的 9 表示$ACADVER 是一个字符串，同时也说明该字符串表示的是一个变量名。这里的 1 表示 AC1021 是一个字符串，同时也说明该字符串是 AutoCAD（或 DXF）的版本号。这两个键值对组合在一起就定义了 $ACADVER=AC1021。

更多的关于组代码和组代码对应的值的含义，以及每个 SECTION 的详细构成请参考 AutoCAD 2000 DXFReference。
(https：//www.autodesk.com/techpubs/autocad/acad2000/dxf/index.htm)

2.3 DXF文件解析

本课题所关注的是闭合多边形，故在数据提取和解析中，只关注多边形部分。在 DXF 文件中多边形的表示用的是多段线（POLYLINE）和线段（LINE）。下面介绍如何读取 DXF 文件，并从中获取多段线（POLYLINE）和线段（LINE）的相关信息。对于 DXF 这种文件格式，可以利用表驱动方式来读取。解析过程如图所示。

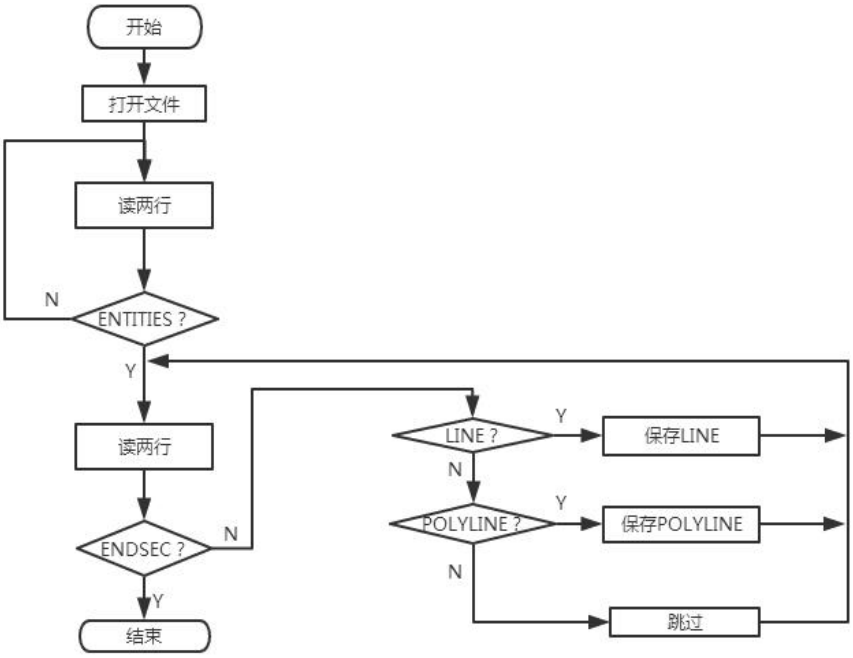

图 2.3 DXF 文件读取的流程图

Figure 2.3 flow chart of reading DXF file

如图所示，先判断我们是否到达了 ENTITIES SECTION。若没有，那么继续读取下一行。思考这里为什么没有考虑 BLOCKS SECTION?回顾先前的介绍，实体(ENTITY)是可以存储在两个 SECTION 中，分别是 ENTITIES SECTION 和 BLOCKS SECTION。而这里却忽略了 BLOCKS?这个与 DXF 文件的生成有关。如果我们在制图过程中，没有使用到块或块的引用，那么在 BLOCKS 中也就不存在实体(ENTITY)，也就不必要分析 BLOCKS SECTION。若我们使用了块(块引用)，那么必须需要分析 BLOCKS SECTION。不然就会缺少图形化的对象。这里我们做了简化处理，故只分析 ENTITIES SECTION(因为在制图中没有使用块引用)。在 ENITITES 中，我们可以获取到所有的图形化对象。同样，本文做了简化，只提取线段(LINE)和多段线(POLYLINE)的相关的数据。

数据的提取采用的是表驱动的模型。因此需要写一个处理程序来去读取并解析 DXF 文件，而 DXF 文件的读取和解析是一件及其繁琐和枯燥的工作。现在已存在很多 DXF 文件的解析库，如 dxflib、libdxfrw 等。本文采用 dxflib 作为解析工具。

Dxflib 是一个 C++解析库，采用了表驱动的设计模式。源码实现中包含了大

量的回调函数，来处理不同的组代码。

在 Dxflib 中，已定义了一个非抽象类 DL_CreationAdapter。该类继承于抽象基类 DL_CreationInterface，并做了默认的实现，故该类中包含了全部可用于重载的函数。用户只需要继承该类，并重写相应的函数，注册到库中即可。当 dxflib 在解析到相应的实体(或块引用)的时候就会调用用户已注册的函数。按照这个思路，用户代码只需要在重载的函数中做相应的逻辑处理，即可获取所需要的信息。由于本文只考虑了闭合多边形。那么只需重载其中的 3 个函数，分别是 DL_CreationAdapter::addLine，DL_CreationAdapter::addPolyline 和 DL_CreationAdapter::addVertex。通过这三个函数，就可以拿到 DXF 文件中每个多边形的相应的数据。

为了与下文的描述相吻合，同时也为了简化复杂度，这里给出的多边形定义非常简单，只是用于存储数据。多边形 Polygon 的定义如下：

```
Struct Point{
    Double x;
    Double y;
}
Typedef Struct Polygon{
    Struct Point points[];
} Polygon;
```

其中 points 是一个 Point 的数组，其中的顶点按逆时针排序。

接下来对于文件的读取和解析操作，只需要修改三个函数的逻辑操作即可。(1)DL_CreationAdapter::addLine 这个函数在每次被调用的时候表示已读取到一条线段。可以将线段存储到一个临时集合内。最后需要做的就是将线段拼接成一个一个多边形。这里为什么需要把线段临时存储，最后才一个个拼接，而不是一下子就能读取一个完整的多边形，然后直接存储到 Polygon 中，再读第二个呢？因为这与 DXF 的存储形式有关，它并没有将一个多边形的边都存储在一起。也就是说，在 DXF 制图过程中，图元的元素是先绘制的先存储。故多边形的边在文件中是分散存储的(龚清洪，2006)。因此，需要临时存储全部的边，最后统一生成一个一个多边形(图元)。(2)DL_CreationAdapter::addPolyline 和 DL_CreationAdapter::addVertex 是一对函数。前一个函数获取多段线的顶点个数，并判断该多段线是否闭合(本文涉及的多边形均闭合)；后一个函数会利用前一个

函数的找到的顶点数，同时在遇到一个顶点时，记录该顶点。因此当顶点数达到指定数目后，就可以得到一个多边形了。若前一个函数再次被调用，就会更新多段线(多边形)的顶点数，那么接下去调用第二个函数就会读取新的多边形。

通过前面三个函数的共同合作，可以把 DXF 中的(闭合)多边形都存入 Polygon 结构体中，对后续的排样算法和切割路径算法提供必要的数据支持。

2.4 本章小结

本章首先讨论了 DXF 文件的格式。接着提出了 DXF 文件解析的基本思路是表驱动方式。然后，借助于已有的库，实现了数据的提取和转化，为下文的排样和路径切割提供了数据支持。

第 3 章　板材优化下料算法

板材优化下料算法，也称排样算法，主要解决的问题就是把各种二维的小部件布局到一个矩形(也可以是其他形状)板材上，使得板材的利用率较高。本章在描述具体的算法之前，先介绍一些基本的几何算法。接着，介绍已存在的一些排样算法，如 BL 算法(韩志仁 等，2011)，下台阶算法(沈志荣，2014)，最低水平线算法(周家智，2018)等，评价这些算法的优缺点。然后，提出一种改进的排样算法。算法主要由两部分构成。一是二维零件的放置问题，即二维零件怎么放置，才不会导致他们互相重叠。这里采用了非拟合多边形算法。该算法的实现过程会详细的叙述。二是整体排样的优化过程。因为第一步得到的是部件与部件的之间的不重叠放置方式，因此这一步需要所有的部件均不重叠的布局在板材上，完成排样。且这一步需要优化布局方式，使得矩形板材的利用率较高。优化算法采用的是遗传算法。最后，会给出实验的测试结果，论证排样算法的可行性。本章的结构安排如下，3.2 节介绍几何算法，3.3 节介绍已有的一些排样算法，3.4 节介绍非拟合多边形的构造算法，3.5 节介绍遗传算法，3.6 节为算法实现。

3.1　基本的几何算法

排样算法，尤其是非拟合多边形构造(NFP)算法，需要用到大量的几何知识。现已存在的许多几何算法库，如 ACIS，Parasolid，OpenCASCADE 等。由于本文只涉及二维的基础几何算法，大规模的算法库的引入是不必要的。下面对一些常见的几何算法(O’Rourke J，1998)做一些说明。

3.1.1　面积计算

对于多边形的面积求解，一种方法是把该多边形分解成多个三角形，然后对三角形的面积求和即可。

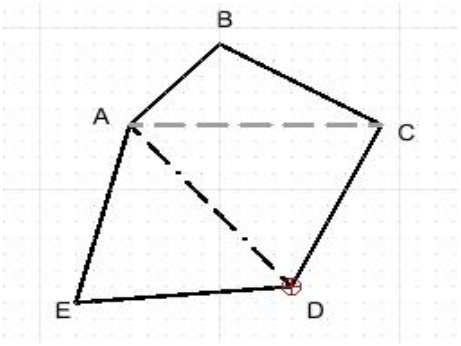

图 3.1　一个多边形

Figure 3.1 A Polygon

图 3.1 中多边形的面积

$$S_{ABCDE} = S_{ABC} + S_{ACD} + S_{ADE} \quad \ldots(3.1)$$

对于三角形面积的求值，可以使用向量方式。如三角形 ABC 的面积为

$$S_{ABC} = \frac{1}{2} * \left| \overrightarrow{AB} \times \overrightarrow{AC} \right| \quad \ldots(3.2)$$

需要注意的是，该公式求解的面积值一定为正。这是因为对 AB 与 AC 叉乘后的值取了绝对值。那对于凹多边形而言，这种方式依旧适用, 只是不能取绝对值。

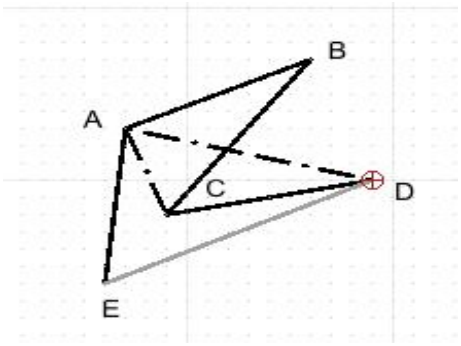

图 3.2 另一个多边形

Figure 3.2 Another Polygon

对于图 3.2 的所示的凹多边形，依旧可以利用上述的方法。此时多边形 ABCDE 的面积为

$$S_{ABCDE} = S_{ABC} + S_{ACD} + S_{ADE} \quad \ldots(3.3)$$

但是对于三角形 ABC，ACD，ADE，采用不加绝对值的求值公式，如下

$$S_{ABC} = \frac{1}{2} * \left(\overrightarrow{AB} * \overrightarrow{AC} \right) \quad \text{... (3.4)}$$

$$S_{ACD} = \frac{1}{2} * \left(\overrightarrow{AC} \times \overrightarrow{AD} \right) \quad \text{... (3.5)}$$

$$S_{ADE} = \frac{1}{2} * \left(\overrightarrow{AD} * \overrightarrow{AE} \right) \quad \text{... (3.6)}$$

从这 3 个公式中可以看出，若S_{ABC}的有向面积为正(其实按右手系规则来说，是负数)，那么S_{ACD}的有向面积就是负的，而S_{ADE}是正的，最终S_{ABCDE}的有向面积是负的。如果我们不是以 A 作为分割三角形的原点，而是任选一点，这个结论同样成立，并可以简化计算。最终的算法如下：

```
Func PolygonArea(Point *p，int num)
    if num < 3
        Return 0;
    Sum = 0;
    p[num+1] = p[1];
    For i = 1 to n
        Sum += p[i].x * p[i+1].y - p[i].y * p[i+1].x;
    Return sum;
```

3.1.2 点在多边形内

判断点是否在多边形内的最常用方法是射线法。对于给定的一点 P，我们向 x 轴的反方向做射线 L。由于多边形是有界的，那么射线 L 的左端(-∞，P_y)一定在多边形外部。现考虑沿着射线 L 从无穷远处开始自左向右移动，遇到与多边形相交的第一个交点时，进入到多边形的内部，遇到第二个交点时，则离开多边形。因此，可以利用交点的个数来判断点 P 是否在多边形内。若射线 L 与多边形的交点个数是奇数，那么 P 点在多边形内；若射线 L 与多边形的交点个数是偶数，那么 P 点在多边形外。还要一种情况，P 点可能出现在多边形的边(顶点)上，这种情况也认定为 P 点在多边形外。考虑下图的情况：

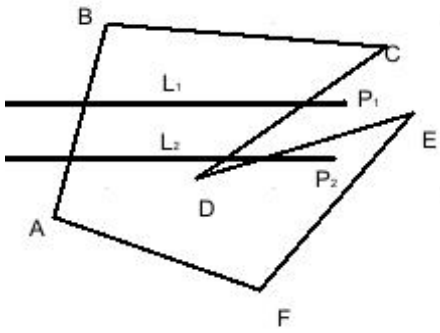

图 3.3 判断点与多边形的位置关系

Figure 3.3 Position relationship between point and Polygon

在图 3.3 中，P_1与多边形 ABCDEF 的交点有 2 个，为偶数个，故P_1在多边形外；而P_2与多边形的交点个数有 3 个，为奇数个，故P_2在多边形内。还可以出现如下的一些特殊情况。

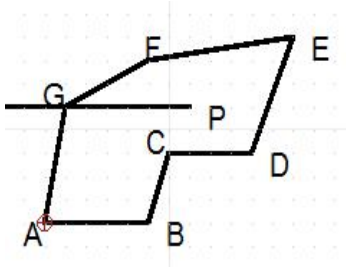

图 3.4 点与多边形的关系 – 特例 1

Figure 3.4 position relationship between point and polygon - case 1

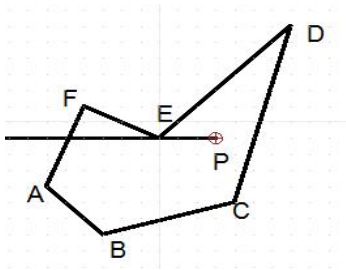

图 3.5 点与多边形的关系 – 特例 2

Figure 3.5 position relationship between point and polygon - case 2

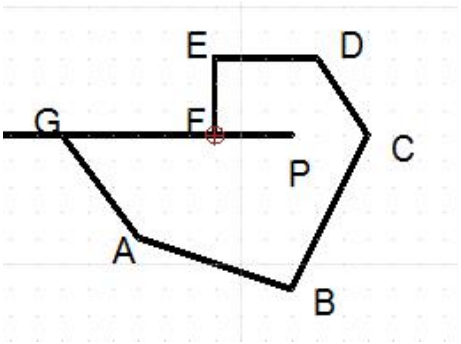

图 3.6 点与多边形的位置关系 – 特例 3

Figure 3.6 position relationship between point and polygon - case 3

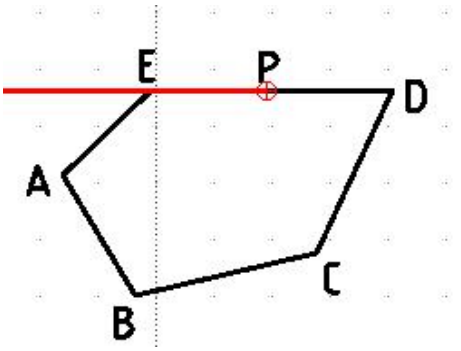

图 3.7 点与多边形的位置关系 – 特例 4

Figure 3.7 position relationship between point and polygon - case 4

图 3.4-3.7 介绍了 4 种特殊情况，在图 3.4 中，射线 L 与多边形的顶点相交。这种情况中计算的交点数只能为 1，不能为 2；在图 3.5 中，不能对射线 L 与多边形顶点 E 的交点计数，故这种情况中也只统计为一个交点。在图 3.6 和图 3.7 中，射线 L 与多边形的一边重合，那么该边因忽略不计。最后的算法如下：

```
Func PointIsInPolygon(Point* p，int num，Point point)
        Int i，j;
        Bool c = false;
        For (i = 0，j = num - 1; i < num; j = i++)
            if( ( ((p[i].y <= point.y) && (point.y < p[j].y)) ||
                ((p[j].y <= point.y) && (point.y < p[i].y))   )   &&
                (point.x < (p[j].x - p[i].x) * (point.y - p[i].y) / (p[j].y - p[i].y) +
                p[i].x)){
                    c = !c;
                }
```

```
return c;
```

3.1.3 多边形的矩形边框

计算一个多边形的最小矩形边框，只需要计算多边形的最低点/最高点的 y 值，最左点/最右点的 x 值即可。算法如下：

```
Func getBoundRect(Point *p，int num)
    Double x_max = DOUBLE_MIN，x_min = DOUBLE_MAX;
    Double y_min = DOUBLE_MAX，y_max = DOUBLE_MIN;
        for i = 1 to num：
            if p[i].x > x_max
                X_max = p[i].x;
            if p[i].x < x_min
                X_min = p[i].x;
            If p[i].y > y_max
                Y_max = p[i].y;
            If p[i].y < y_min
                Y_min = p[i].y;
    Return {{x_min， y_min}，{x_max， y_max}};
```

3.1.4 点的旋转

给定一个旋转点 P 和一个旋转中心 0 以及一个旋转角度α，将 P 点绕 0 逆时针选择α度。算法如下：

```
Func rotatePoint(Point p， Point o， double a)
    Point p2;
    p2.x = p.x*cos(a) - p.y * sin(a) + o.x
    P2.y = p.y*cos(a) + p.x * sin(a) + o.y
    Return p2;
```

3.1.5 多边形旋转

多边形旋转本质上就是对多边形的每个顶点执行旋转操作。算法如下：

```
Func RotatePolygon(Point *p， int num， Point o， double a)
    Polygon p2;
    For i = 1 to num
        p2[i] = rotatePoint(p[i]， o， a);
    Return P2;
```

3.1.6 线段交叉

判断两条线段 AB 和 CD 是否相交。若相交，求出交点。两条线段的位置关系可以分为下面几类：有部分重合、不重合但有交点、不相交。算法有两个步骤组成。(1)快速测试 (2)交叉判断。

(1)快速测试。就是以 AB 作为一个矩形 R1 的对角线，以 CD 作为另一矩形 R2 的对角线。如果 R1 与 R2 不相交，那么 AB 与 CD 一定不相交。

(2)交叉实验。如果两线段相交，那么他们必须是相互交叉对方。如果 AB 与 CD 交叉，那么$\overrightarrow{CA}$和$\overrightarrow{CB}$位于$\overrightarrow{CD}$的两侧，即

$$(\overrightarrow{CA} \text{ x } \overrightarrow{CD}) * (\overrightarrow{CB} \text{ x } \overrightarrow{CD}) < 0 \quad \text{... (3.7)}$$

同理，如果 CD 与 AB 交叉，那么$\overrightarrow{AC}$和$\overrightarrow{AD}$位于$\overrightarrow{AB}$的两侧，即

$$\left(\overrightarrow{AC} \times \overrightarrow{AB}\right) * \left(\overrightarrow{AD} \times \overrightarrow{AB}\right) < 0 \quad \text{... (3.8)}$$

当确认两个向量相交后，就可以进行交点的计算。求交点可以利用平面几何的方法，列方程式完成。但对于斜率为 0 的情况，需要特殊考虑。如后文所示，使用向量法求解可以避免这种情况。设交点 P 的坐标为(x_0， y_0)，线段 AB 的坐标为(x_1， y_1)和(x_2， y_2)，线段 CD 的坐标为(x_3， y_3)和(x_4， y_4)。可以得出下列方程。

$$x_0 - x_1 = k_1(x_2 - x_1) \quad \text{... (3.9)}$$

$$y_0 - y_1 = k_1(y_2 - y_1) \quad \text{... (3.10)}$$

$$x_0 - x_3 = k_2(x_4 - x_3) \quad \text{... (3.11)}$$

$$x_0 - x_4 = k_2(y_4 - y_3) \quad \text{... (3.12)}$$

消除 k1 和 k2，得到如下方程

$$x_0(y_2 - y_1) - x_1(y_2 - y_1) = y_0(x_2 - x_1) - y_1(x_2 - x_1) \quad \text{... (3.13)}$$

$$x_0(y_4 - y_3) - x_3(y_4 - y_3) = y_0(x_4 - x_3) - y_3(x_4 - x_3) \quad \text{... (3.14)}$$

求解 x0 和 y0，得到如下结果

$$\text{t}_1 = (y_2 - y_1) * x_1 + (x_1 - x_2) * y_1 \quad \text{... (3.15)}$$

$$t_2 = (y_4 - y_3) * x_3 + (x_3 - x_4) * y_3 \quad \text{... (3.16)}$$

$$\text{M} = (x_2 - x_1) * (y_4 - y_3) - (x_4 - x_3) * (y_2 - y_1) \quad \text{... (3.17)}$$

$$\text{X} = t_2 * (x_2 - x_1) - t_1 * (x_4 - x_3) \quad \text{... (3.18)}$$

$$Y=t_2*(y_2-y_1)-t_1*(y_4-y_3) \quad \text{...(3.19)}$$

$$x_0=\frac{X}{M}, y_0=\frac{Y}{M} \quad \text{...(3.20)}$$

通过前面的计算，可以得出如下的算法。

(1) 判断 AB 和 CD 对应的矩形是否交叉

```
Func isRectCross(Point A，  Point B，  Point C，  Point D)
    Bool ret = min(A.x，  B.x) <= max(C.x，  D.x) &&
               Min(C.x，  D.x) <= max(A.x，  B.x) &&
               Min(A.y，  B.y) <= max(C.y，  D.y) &&
               Min(C.y，  D.y) <= max(A.y，  B.y)
    Return ret;
```

(2) 交叉判断

```
Func isLineCross(Point A，  Point B，  Point C，  Point D)
    long line1，  line2;
    Line1 = A.x * (C.y - B.y) + B.x * (A.y - C.y) + A.x * (B.y - A.y);
    Line2 = A.x * (D.y - B.y) + B.x * (A.y - D.y) + D.x * (B.y - A.y);
    If(((line1 ^ line2) > = 0) && !(line1 == 0 && line2 == 0))
        Return false;
    Line1 = C.x * (A.y - D.y) + D.x * (C.y - A.y) + A.x * (D.y - C.y);
    Line2 = C.x * (B.y - D.y) + D.x * (C.y - B.y) + B.x * (D.y - C.y);
    If(((line1^ line2) >= 0) && !(line1 == 0 && line2 == 0))
        Return false;
    Return true;
```

(3) 计算交点

```
Func getCrossPoint(Point A, Point B, Point C, Point D)
    If !isRectCross(A,B,C, D)
        Return {DOUBLE_MIN, DOUBLE_MIN};
    If !isLineCross(A, B, C, D)
        Return {DOUBLE_MIN，  DOUBLE_MIN};
    Long x y, t1, t2;
    T1 = (D.x - C.x) * (A.y - B.y) - (B.x - A.x) * (C.y - D.y);
    T2 = (A.y - C.y) * (B.x - A.x) * (D.x - C.x) + C.x * (D.y - C.y) * (B.x -
    A.x) - A.x * (B.y - A.y) * (D.x - C.x);
    X = T2 / T1;
    T1 = (A.x - B.x) * (D.y - C.y) - (B.y - A.y) * (C.x - D.x)
    T2 = B.y * (A.x - B.x) * (D.y - C.y) + (D.x - B.x) * (D.y - C.y) * (A.y -
    B.y) - D.y * (C.x - D.x) * (B.y - A.y);
    Y = T2 / T1;
    Return {X , Y};
```

3.1.7 多边形交叉

多边形的交叉判定是基于线段之间的交叉判断。也就是对任意两个多边形 A 和 B，只要 A 和 B 中的任意一对边相互交叉，那么就说明 A 和 B 交叉了。算法如下：

```
Func isPolygonCross(Point *A， int a， Point *B， int b)
    For i = 1 to a
        For j = 1 to b
            If isLineCross(A[(i-1)%a+1]， A[i%a+1]， B[(j-1)% b+1]，
        B[j%b+1])
                Return true;
        Return false;
```

3.1.8 线段(向量)正规划(规范化)

向量规范的目的就是让向量的长度变为 1。算法如下：

```
Func normalize(Vector *a)
    Double length = sqrt(a.x * a.x + a.y * a.y);
    Return Vector(a.x / length， a.y / length);
```

3.1.9 判断线段是否在多边形内

要判断线段是否在多边形内，首先需要判定线段的两个端点都在多边形内。这个可以通过 3.2.2 节的算法来判断。同时，如果线段与多边形的边有交点，且交点不在两线段的端点，这时可以确定线段一定有一部分在多边形外。因此，要让线段出现在多边形内，必须同时满足两个条件(1)线段的端点在多边形内；(2)线段与多边形的所有边都不存在交点在端点的情况。

若线段与多边形的交点出现在线段的端点，这种情况可以接受。但是，如果多边形的某个顶点和线段的端点相交，还需要判断相邻交点之间的线段是否在多边形内部。因此，可以先求出所有和线段相交的多边形顶点，然后按照 X-Y 坐标排序，这样相邻的两个点就是在线段上相邻的两个交点。可以证明任意相邻两点的中点若在多边形内，那么线段一定在多边形内。在实际的算法设计中，并不需要计算所有的交点。首先应判断线段与多边形的边是否有不再端点的交点，若存在，那么线段一定在多边形外。若线段与多边形的每一条边都不存在在端点的交点，那么线段与多边形的交点一定是线段的端点或者多边形的顶点，此时只需判断交点是否出现在线段上。算法如下：

```
Func LineIsInPolygon(Point A，Point B，Polygon P)
    If AB 的端点有出现在多边形外
```

```
    Return false;
初始化 pointSet 为空；
For P 的每条边 e
    If AB 的某个端点在 e 上
        将该端点放入 pointSet 中；
    Else if 边 e 的某个端点在线段 AB 上
        将该端点放入到 pointSet 中；
    Else if 边 e 和线段 AB 相交
        Return false；
将 pointSet 中的点按 X-Y 排序；
For 对于 PointSet 中的两个相邻的点 Pi， Pi+1
    If Pi 与 Pi+1 的中点不在多边形内
            Return false;
Return true;
```

3.2 BL算法、下台阶算法与最低水平线算法

本节介绍一下 3 种常见的排样算法，分别是 BL 算法，下台阶排样算法，基于最低水平线的排样算法。这 3 种算法的研究对象一般是矩形部件。而对于非规则部件，则将其转化为矩形部件，然后进行排样处理。

3.2.1 BL 算法

BL 算法是 Bottom-Left 算法的简称，这是一种对矩形件排样的处理方法。算法的总体思路：将矩形件从板材的右上角开始，先向下移动，当碰到其他部件而不能再向下移动时，就一直往左移动；若又碰到其他部件而不能再向左移动时，则向下移动。按照这种处理方式，可以让每一个放置的部件尽可能接近左下角。在图 3.8 中演示 BL 算法。

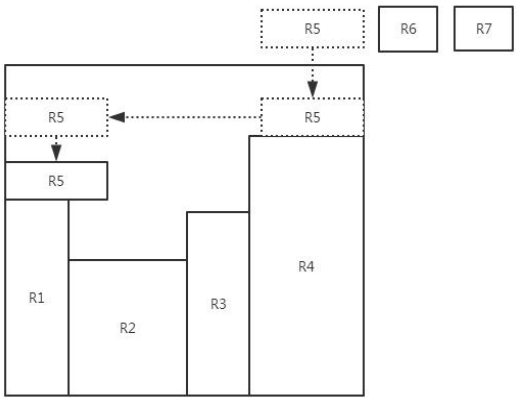

图 3.8 BL 算法演示

Figure 3.8 BL algorithm demo

3.2.2 下台阶排样算法

与 BL 算法不同的是，下台阶排样算法是这样的。首先矩形件还是从板材的右上角开始，先向下移动，若碰到其他部件而不能向下移动时，再朝左移动。在朝左移动过程中，发现可以向下移动的使用，就向下移动。若向下移动碰到了其他已放置的部件而不能再向下移动时，则朝左移动. . . 。图 3.9 演示该下台阶算法的过程。

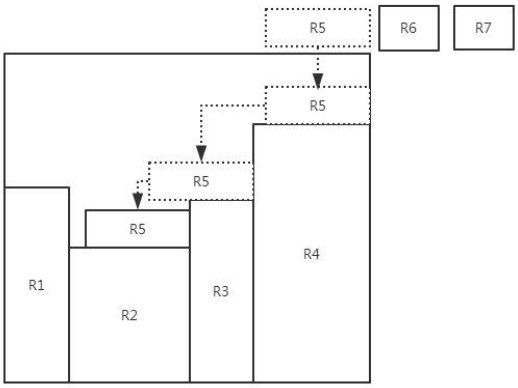

图 3.9 下台阶算法演示

Figure 3.9 algorithm of walk down the steps demo

3.2.3 基于最低水平线的排样算法

基于最低水平线的排样算法可以认为是对 BL 算法和下台阶算法的优化。该算法的核心思想是确定全局最低的水平线以及不同水平线处的宽度值，使得矩形部件排放在最低水平线上(或靠近最低水平线的水平线上)。通过对该算法的简单描述，可以知道在算法实现过程中，不同高度的水平线的更新(合并，添加，删除)以及不同水平线的宽度更新都是很重要的。图 3.10 简单的描述了最低水平线算法的处理过程。

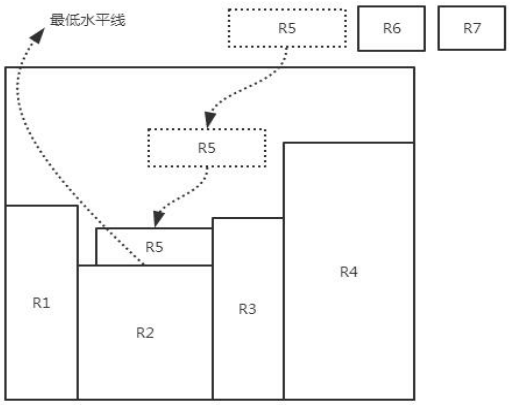

图 3.10　最低水平线算法演示

Figure 3.10 algorithm of minimum horizontal line

3.3　非拟合多边形构造算法

在 3.3 节中，我们介绍了 3 种排样算法的实现。它们都是基于矩形件的排样，对于不规则部件，排样的结果不是很理想。为了解决这种情况，本节实现了一种针对非规则部件(对于矩形部件一定有效)的排样子算法，在 3.6 节会实现完整的排样处理。

3.3.1　非拟合多边形

非拟合多边形(No-fit Polygon，NFP)的构造方式，原理是环绕的方法来产生 NFP，该算法是对 Burke，E.K.等人(2007)提出的算法的一种改进实现。比起传统的利用三角函数的重叠测试(Dowsland K.A and Dowsland W.B，1992；Art R.C，1996；Bennell J.A 等.，2001；Agarwal P.K 等.，2002)，实现该算法简单，同时具有时间复杂度较低的优点。

算法的原理如下，输入参数是两个二维多边形，形状可以是不规则的。假定其中一个多边形(记为 A)保持不动，另一个多边形(记为 B)绕 A 运动。这样当 B 环绕 A 运动一周后，可以生成一个非拟合多边形(下面简称为 NFP)。

假定多边形 A 为固定多边形，B 为环绕多边形。算法实现过程分为 4 步(1)首先需要 A 与 B 接触；(2)移动 B；(3)找到下一个起点；(4)将平移向量合并。下面分节讨论。

3.3.1.1 多边形 A 与 B 的接触

在 B 绕 A 运动之前，需要把 B 放置在 A 的边上，保证 B 接触 A，但又和 A 不相交。然后，可以对 B 沿某方向平移，使之一直是和 A 保持接触。

例如，图 3.11 给出的多边形 A 和 B，同时标记了 A 的最低点 A_minY 和 B 的最高点 B_maxY。

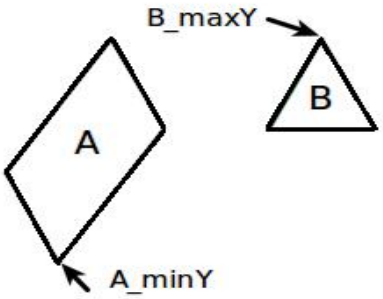

图 3.11 多边形 A 和 B

Figure 3.11 polygon A and B

那么，可以让 B 的最高点与 A 的最低点接触，这样 A 与 B 是一定接触，但不相交(也可以有其他方法，例如 B 最左点和 A 的最右点接触，等等)。平移向量

$$\vec{T} = A\_\min Y - B\_\max Y \quad \text{...(3.21)}$$

那么 B 按 T 平移后，如图 3.12 所示，B 与 A 接触并且不交叉。此时，B 所在位置是 B 绕 A 运动的开始位置，并记录起点和 B 的参考点，保证 B 在绕 A 平移过程中可以回到初始位置。设 B 的参考点和起点都为 A_minY。那么，当 B 的参考点再次平移到起点(A_minY)时，就可以确定 B 绕 A 一周。

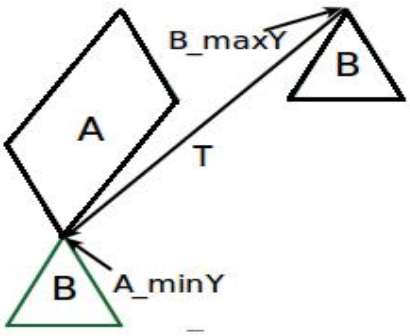

图 3.12　多边形 B 沿 T 向量平移

Figure 3.12 polygon B translate using T vector

3.3.1.2　移动 B

在 A 与 B 接触后，需要对 B 做平移操作。这个过程细分为以下 5 步。

(1)找出接触边对

当前 A 和 B 接触，通过遍历 A 和 B 的全部边，找出所有接触点。继而确定全部接触边对。如图 3 所示，找到 4 组接触边对，分别是<a1，b1>，<a1，b3>，<a2，b1>，<a2，b3>。

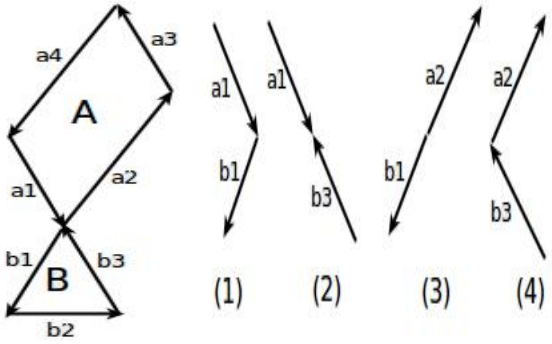

图 3.13　接触点及接触边对

Figure 3.13 Touch point and touch pair

(2)从接触边对中找出平移向量

在(1)中，找到了 4 组接触边对。接触边对的作用是：一组接触边对能生成一个平移向量，故能得到全部的平移向量。例如，在图 3.13 的 4 组边对中，1 号边对<a1，b1>可以生成一个平移向量$\vec{T}=-\vec{b1}$，$\vec{T}$是由边 b1 生成的。那为什么是

由边 b1 生成，而非 a1 生成？算法假定平移向量是接触点到边的终点，故 a1 生成的是空向量。$\vec{T}$ 是 $\overrightarrow{b1}$ 的反向，这是认为 B 必须绕 A 做逆时针运动。那么 2 号边对<a1，b3>就不能生成平移向量。因为接触点是 a1(也是 b3)的终点。而 3 号边对<a2，b1>(a2 与 b1 是平行关系)可以生成平移向量 $\vec{T}=\overrightarrow{a2}$ (或者 $\vec{T}=-\overrightarrow{b1}$)。也就是说，在这种情况下可以选择任何一边来生成平移向量。4 号边对<a2，b3>中，$\vec{T}=\overrightarrow{a2}$。综上可以得出如下表 3.1 所示的对应关系。

表 3.1 多边形 A 和 B 的接触边及对应的平移向量

Table 3.1 touch pairs and corresponding translation vector of polygon A and B

接触点		移动边(来自 B)相对固定边(来自 A)的位置	平移向量
固定边 a(来自 A)	移动边 b(来自 B)		
起点	起点	左边	$-\vec{b}$ ($\vec{b}$ 取反)
起点	起点	右边	$\vec{a}$
起点	终点	左边	无
起点	终点	右边	$\vec{a}$
终点	起点	左边	无
终点	起点	$-\vec{b}$	$-\vec{b}$
终点	终点	任意	无
		平行	$\vec{a}$ 或 $-\vec{b}$

(3) 删除不可行的平移向量

对(2)中找到的所有平移向量，需要依次判断多边形 B 沿这些向量移动会不会与 A 交叉。若有交叉，那么该向量就是不可行的。按如下方式思考：因为 B 绕 A 运动，那么在(1)中的接触边对中来自 B 的边是运动的。也就是说，我们可以对每一个平移向量，依次针对每一组接触边对，判断来自 B 的边按该向量平移，是否与来自 A 的边相交。若有一组接触边相交，那么就可以判断该平移向量是不可行的，直接舍弃。

那么怎么验证接触边对中来自 B 的边按平移向量平移，会不会与来自 A 的

接触边交叉。对于每一组接触边对而言，可以计算出弧度区间，使得来自 B 的边沿区间中的某个弧度方向平移，不会和来自 A 的边相交。故只需要判断平移向量的弧度方向是否在该区间内即可。例如，对(1)中的四组接触边对，可以确定相应的弧度区间(用圆弧表示可行的弧度区间，如图 3.14 所示。

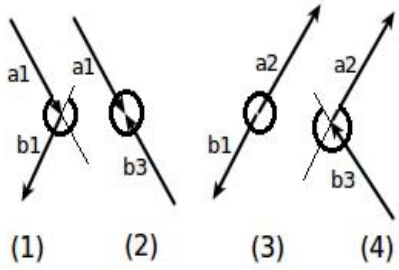

图 3.14　接触边对的可行弧度区间

Figure 3.14 feasible radius interval of touch pair

在(2)中，找到的平移向量有$-\overrightarrow{b1}$和$\overrightarrow{a2}$。对照图 3.14 中的 4 组接触边对的弧度区间和$-\overrightarrow{b1}$(或$\overrightarrow{a2}$)的弧度方向，发现$-\overrightarrow{b1}$(或$\overrightarrow{a2}$)的弧度方向均在每一组的弧度区间内，故$-\overrightarrow{b1}$和$\overrightarrow{a2}$都是可行的平移向量。

(4) 修剪可行的平移向量

通过(3)的过滤，找到了全部可行的平移向量。那么接下来，需要判断每一个可行的平移向量是否能全部应用。

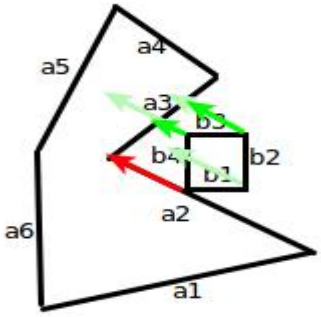

图 3.15　多边形沿平移向量平移，交叉

Figure 3.15 polygon moves along the translation vector and crosses

在图 3.15 中，A 是一个不规则多边形，B 是一个矩形。红色的是一个可行的平移向量，浅绿色是红色的平移向量平移到 B 的每个顶点之后的情况。此时 B 沿着红色的平移向量平移，A 与 B 必然交叉。从而，必须对红色向量进行修剪。图中得到修剪后的是绿色的平移向量。看到有两个绿色向量，但取得是较短的那个。同时，需要注意的是，B 按修剪后的平移向量平移，一定不会与 A 相交？

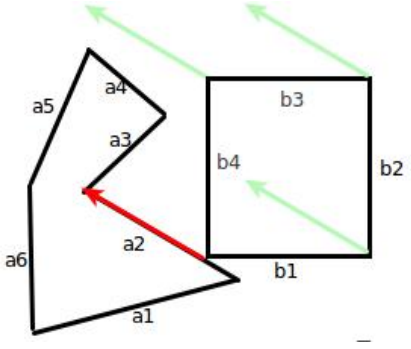

图 3.16　平移向量的起点放置在 B 的每个顶点

Figure 3.16 put start point of the translation vector at each vertex of B

在图 3.16 中，A 不变，B 是较大的矩形，红色向量仍是可行的平移向量。将红色向量平移到 B 的每个顶点，发现并不需要修剪平移向量。但是按红色向量平移后，如图 3.17 所示，A 与 B 相交。这种情况下，就需要考虑将可行的平移向量的终点放置

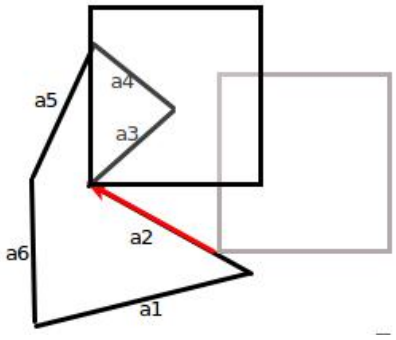

图 3.17　多边形 B 沿平移向量平移，　与 A 交叉

Figure 3.17 polygon B moves along the translation vector and crosses with polygon A

到 A 的每个顶点，再修剪平移向量。如图 3.18 所示。

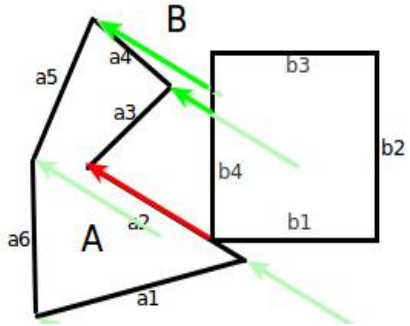

图 3.18　平移向量的修剪

Figure 3.18 trim the translate vector

在图 3.18 中，将平移向量的每个终点平移到 A 的每个顶点，判断与 B 的交叉性，最终得到修剪后的平移向量(绿色)。然后将两种情况的最小值作为最终的可行平移向量。对每一个可行的平移向量执行上述过程之后，就能得到全部修剪后的平移向量，然后按从长到短排序。接下来只需要按最长的修剪后的可行平移向量平移 B 就可以了。

值得指出的是，在 Burke，E.K 等人(2007)的论文中，直接使用修剪后的最长平移向量进行平移。但是，在实验中发现该平移向量不一定是能用的。也就是说，按此平移向量平移后，B 在下一次平移之前就不一定与 A 接触了。

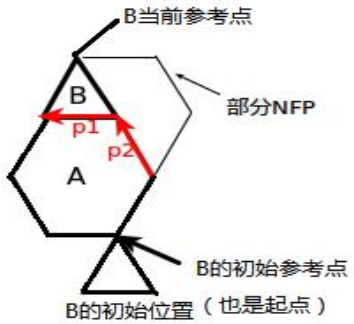

图 3.19　B 沿平移向量 $\overrightarrow{p1}$ (或 $\overrightarrow{p2}$)平移情况分析

Figure 3.19 analysis of polygon B translating along vector $\overrightarrow{p1}$ or $\overrightarrow{p2}$

在图 3. 19 中，找到了两个可行的平移向量 $\overrightarrow{p1}$ 和 $\overrightarrow{p2}$，$\overrightarrow{p2}$ 的长度比 $\overrightarrow{p1}$ 的长，并

且 $\overrightarrow{p1}$ 和 $\overrightarrow{p2}$ 均是不需要修剪的。当 B 按 $\overrightarrow{p2}$ 平移之后，A 与 B 就不在接触，那么在下一轮找接触点时，就不可能找到，也就无法继续执行。因此，在此种情况下，需要抛弃 $\overrightarrow{p2}$，选择 $\overrightarrow{p1}$。

(5) 移动多边形 B

通过前面的步骤，得到了修剪后最长的可行平移向量。接下来，B 按这个向量平移即可。但需要注意两点：(1)判断 B 的参考点是否回到了起点。若回到起点，得到一个 NFP。(2)判断 B 的参考点是否越过起点。这个是对 Burke，E.K. 等人(2007)论文的一个补充。也就是此时 B 的平移距离过长，超过了起点，如图 3.20 所示。

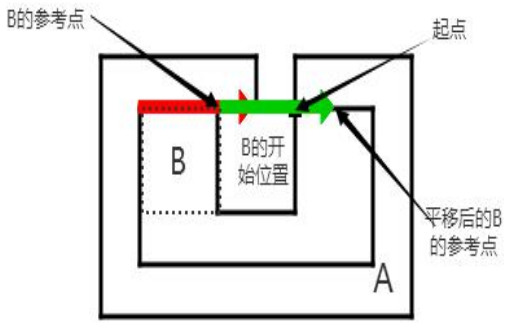

图 3.20 B 的参考点沿平移向量平移越过起点

Figure 3.20 polygon B's reference point gets across the start point

When moving along the translate vector

在图 3.20 中，红色的是可行的平移向量，绿色的表示 B 的参考点的平移。由图知，当 B 的参考点沿平移向量平移后，越过了起点。本来可以生成一个 NFP，现在就会无限循环下去，因为 B 的参考点不会移东到起点了。在这种情况下，需要缩短平移向量。

3.3.1.3 找到下一个起点

在上一节中，生成了一个完整的外部 NFP，是 B 对 A 的外部环绕。那是否存在着其他 NFP 呢？也就是 A 的某些边在 B 的第一次环绕时未遍历到，那这次可以从这些边生成其他的 NFP。这里涉及到起点搜索。在 3.4.1.1 节中，我们假

定的起点是 A 的最低点。而在这里，我们需要找到其他起点，以便 B 能从这些起点出发，得到其他的 NFP。

(1) 起点搜索算法

在寻找外部 NFP 过程中，A 中的有些边可能没有遍历到。故可以从这些边中寻找起点。假设 a 是 A 中一条未遍历的边，试着在 a 中找到一个可行的起点。算法过程如下：依次让 B 的每个顶点平移到 a 的起点，判断：(1)如果 A 此时与 B 没有相交，那么 a 的起点是一个可行的起点。可再次运用 3.3.1.2 节中介绍的方法。(2)如果 A 此时与 B 有交叉，那么需要让 B 沿着 a 移动，直到平移到一个的不相交位置，或者到达 a 的终点(这就说明在 a 上不可能找到起点)。

现在，假定 A 与 B 相交，然后 B 沿 a 平移寻找可行的起点。首先，可以快速判断一下，B 沿着 a 移动是否一定存在交叉。方法如下：因为此时的接触点是 a 的起点，并且 B 中有两条边也接触到这个顶点。那只要判断一下，B 的两条接触边是否至少有一条边在 a 的左侧。若成立，那么 B 沿着 a 移动一定会与 A 相交。

这样，就需要判断 A 中其他未遍历的边了。如果通过上述判断，就需要修剪 a 向量。过程如下：当前的平移向量$\vec{T}$=a 的起点—>a 的终点。同样，利用(4)中介绍的修剪方法修剪$\vec{T}$，得到$\vec{T}'$。那么 B 按$\vec{T}'$平移，同时更新接触点和 B 的参考点。再次运用 1.2 节中介绍的方法，找出剩余的平移向量。这里要注意，就是对 a 进行标记，表示 a 被遍历过了。那么在下一次的搜索中就不会遍历边 a 了。

(2) 平移向量合并生成 NFP

通过 3.4.1.1 节到 3.4.1.3 节的计算，找到所需要的全部平移向量。现在，就需要对他们合并生成 NFP。每一组平移向量都对应一个起点和 B 的参考点。那对 B 的参考点执行一组平移操作，就可以生成一个 NFP。

3.3.2 案例介绍

下面举一些例子，表明算法的运行情况。其中左图是多边形的初始位置，右图是平移的开始位置、起点和生成的 NFP。

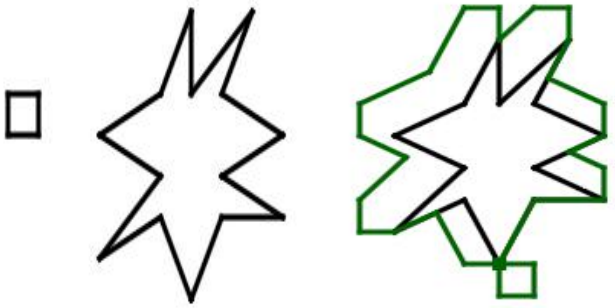

图 3.21 一个外部的 NFP - 样例 1

Figure 3.21 a outer NFP - case 1

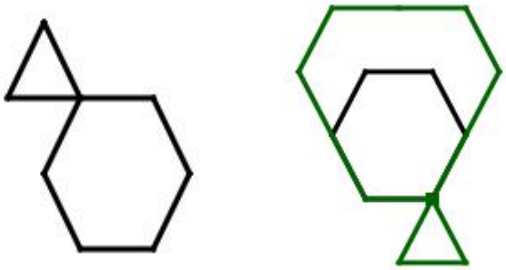

图 3.22 一个外部的 NFP - 样例 2

Figure 3.22 a outer NFP - case 2

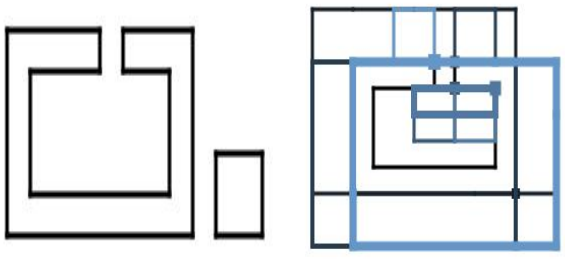

图 3.23 内部和外部的 NFP - 样例 3

Figure 3.23 inner and outer NFP - case 3

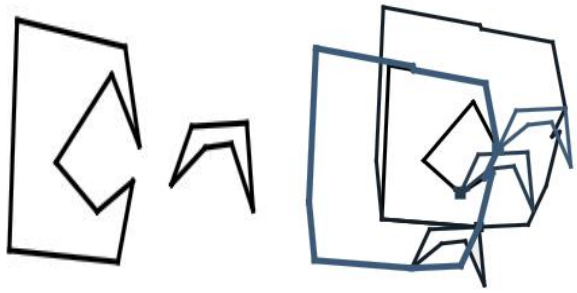

图 3.24　内部和外部的 NFP - 样例 4

Figure 3.24 inner and outer NFP - case 4

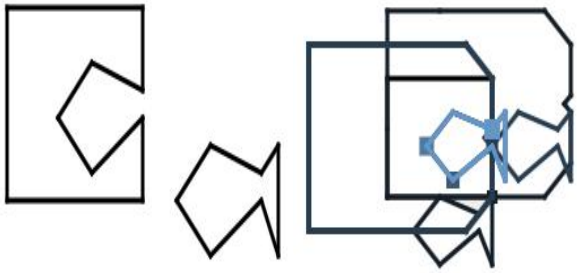

图 3.25 互锁和外部的 NFP - 样例 5

Figure 3.25 interlock and outer NFP - case 5

3.3.3　NFP 算法测试

根据 Burke，E.K.等人(2007)的论文和来自 the Association of the European Operational Research Societies 的板材排样工作小组的数据集(https://www.euro-online.org/websites/esicup/data-sets/)，执行下列的测试。实验中使用的机器为 Intel Core i5-3230M@2.6GHz，8G 内存。表格的形式参考了 Burke，E.K.等人(2007)的论文。

表 3.2 对不同数据集的测试结果

Table 3.2 test result for different data sets

A	B	C	D	E	F	G	H
数据集	不同形状数	旋转度	由旋转生成的形状数	总共的逻辑形状数	NFP 个数	生成时间(s)	每秒的 NFP 数
Terashima 1	540	0	1	540	291600	386.67	754
Terashima 2	480	0	1	480	230400	320.00	720
Albano180	8	180	2	16	256	0.21	1219
Albano90	8	90	4	32	1024	0.53	1932
Dagli	10	90	4	40	1600	0.70	2286
Dighe1	16	90	4	64	4096	1.06	3864
Dighe2	10	90	4	40	1600	0.45	3556
Fu	12	90	4	48	2304	0.15	15360
Jakobs1	25	90	4	100	10000	3.52	2841
Jakobs2	25	90	4	100	10000	3.47	2882
Mao	9	90	4	36	1296	1.32	982
Marques	8	90	4	32	1024	0.65	1575
Poly1a	15	90	4	60	3600	1.00	3600
Poly2a	15	90	4	60	3600	0.99	3636
Poly3a	15	90	4	60	3600	0.8	4500
Poly4a	15	90	4	60	3600	1.09	3303
Poly5a	15	90	4	60	14400	1.02	3529
Poly2b	30	90	4	120	32400	5.64	2553
Poly3b	45	90	4	180	57600	20.50	1580
Poly4b	60	90	4	240	90000	55.48	1038
Poly5b	75	90	4	300	256	125.32	718
Shapes	4	90	4	16	16	0.10	2560
Shapes0	4	0	1	4	4	0.01	1600
Shapes1	4	180	2	8	8	0.01	6400
Shirts	8	180	2	16	16	0.1	2560
Swim	10	180	2	20	20	4.97	80
Trousers	17	180	2	34	34	0.53	2181

通过测试，表明该方法具有一定的高效性。与先前的一些借助于三角函数的方法相比，该算法的实现简单，高效，且不需要对每一种特殊情况做特殊的考虑。利用起点搜索过程，对某些传统方法无法解决的互锁，洞等情况，能够成功解决。

对板材排样系统的设计与实现由一定的借鉴意义。

3.4 遗传算法

3.4.1 遗传算法的概念

遗传算法(Genetic algorithm，GA)是受自然选择过程启发的一种元启发算法，属于进化算法中的一类。对优化和搜索问题能产生高质量的解的关键是依赖于仿生操作，如变异、交叉和选择。在 1960 年，基于达尔文的进化论，John Holland 介绍了遗传算法，而 David E. Goldberg 对其进行了扩展。

下面对遗传算法涉及的一些术语做一些说明。在遗传算法中，针对一个优化问题的候选解(个体)的集合(种群)会向更优的解集合发展。每一个候选解都有一些属性(如染色体)，这些通常能会变异和被修改。传统上，解一般是通过二进制编码成 01 字符串，但是其他的编码方案也是可行的。进化通常开始于一个随机生成的总群，它的每一次迭代(生成下一代)称作一次进化。在每一次进化中，都需要计算各个个体的拟合值，而拟合值表示的是被求解问题的目标函数。于是，拟合度越好的个体会被“选择”并且它的染色体会被修改(重新结合或随机性的变异)而用于生成下一代。新的下一代(新的候选解集)会用于算法的下一次迭代中。一般情况下，当一个最大数目的迭代次数到达或者一个合适的拟合值出现，算法终止。因此，对于一个典型的遗传算法，需要两部分(1)问题域的基因表示(2)一个用于评估的拟合函数。在进化过程中，存在着交叉和变异。

其中，交叉是父代的个体相互间的染色体重新组合，然后遗传给下一代。而变异则是染色体在交叉重组过程中，出现某些基因会在极低的概率下的发生改变，这样后代的性状会与父代有一定的差别，可能会比父代更优秀，也可能差于父代。

3.4.2 遗传算法的执行过程

遗传算法是从问题的一个解集(种群)开始的。种群由个体组成，个体代表着一个具体的解。一个解包含多个部分，每个部分都可以由一个基因来表示。因此个体由基因构成。当初带总群产生之后，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的后代总群。在每一代中，根据群体中个体的适应度大小来选择优秀个体，并借助于自然遗传学的遗传算子进行组合交叉和变异，生成代表新的解集的种群(下一代)。

这个过程将导致种群像自然进化一样，后代的种群比前代更加适应环境，末代种群中的最优个体可以作为问题最优近似解。下图是遗传算法的流程图。

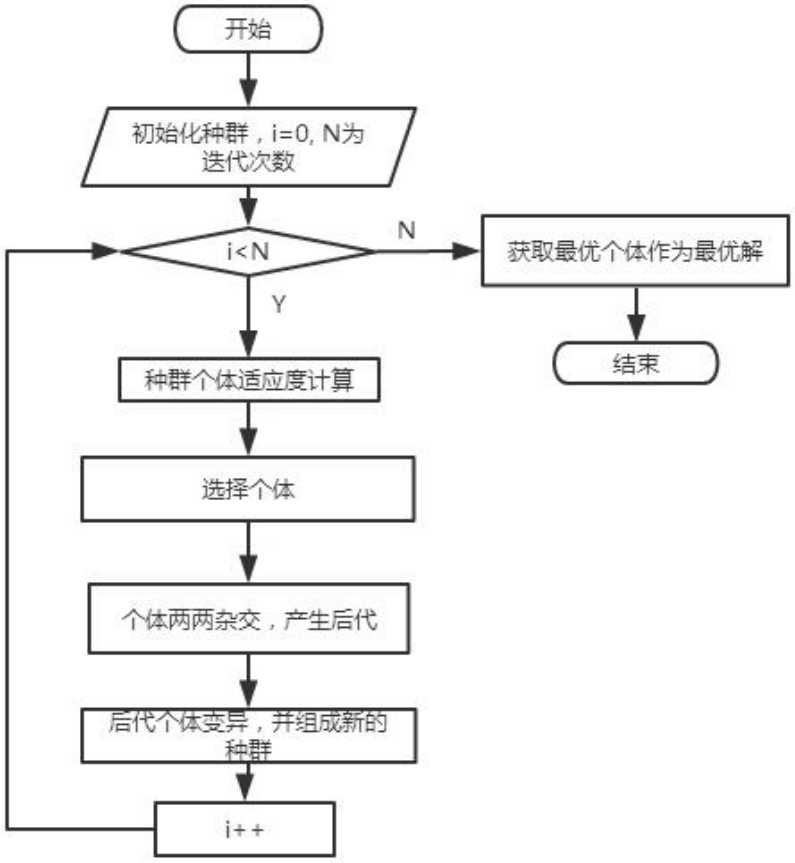

图 3.26　遗传算法流程图

Figure 3.26 flow chart of genetic algorithm

如图 3.26 所示，在初始化时有一个初始总群(若只有一个初始个体，则需要通过个体的变异、交叉来生成初始总群)和一个总群迭代次数。当总群迭代次数超过 N 时，就选择最优的个体作为解。在每次迭代中，首先需要对总群中的每个个体计算其拟合度，然后两两杂交，生成后代，后代可能会变异。后代的集合为一个新的总群，作为下次迭代的父代总群。

针对板材排样问题而言，可以对个体和种群按如下方式来解读。

(1)个体

一个个体表示一个解。每个个体包含一个部件集合和每个部件的旋转角度。在部件集合中含有全部需要被放置的部件(多边形)。因此，问题的关键在于对每

个个体计算其拟合度，也就是对全部部件的一个排放，确定其良好度。因为在遗传算法的求解过程中，会依赖部件之间的位置关系(NFP)，故可以采取一些优化措施。例如对于每一对部件(A，B)，可以构造出一个缓存(键值对集合)，防止重复计算。缓存的构造过程：利用多边形 A 和多边形 B 的各自 ID，以及他们的旋转角度和 B 是否嵌入 A 内来生成键，而值是他们生成的 NFP。

(2)个体间的杂交

对于每个个体的表示而言，即为一个多边形的放置列表以及他们相对应的旋转角度。那么个体间的杂交可以按如下方式执行。任意选择两个个体，如

个体 1：[P_{11}， P_{12}， P_{13}， ...， P_{1n}]，[R_{11}， R_{12}， R_{13}， ...， R_{1n}]；

个体 2：[P_{21}， P_{22}， P_{23}， ...， P_{2n}]，[R_{21}， R_{22}， R_{23}， ...， R_{2n}]；

其中 P_{ij} 表示各个部件(多边形)，R_{ij} 表示多边形的旋转角度。i 为个体编号，j 为一个个体中多边形的编号。然后随机选定一个值 $t \in [1,n]$，则生成的杂交体为

杂交体 1：[P_{11}，P_{12}，...，P_{1t}，$p_{2,t+1}$，...，P_{2n}]，[R_{11}，R_{12}，...，R_{1t}，$R_{2,t+1}$，...，R_{2n}]；

杂交体 2：[P_{21}，P_{22}，...，P_{2t}，$p_{1,t+1}$，...，P_{1n}]，[R_{21}，R_{22}，..，R_{2t}，$R_{1,t+1}$，...，R_{1n}]；

杂交体 1 和杂交体 2 当做子代个体，并做变异处理。

(3)个体的变异

个体的变异过程指的是对个体中的多边形的排列顺序进行交换，或者对多边形的旋转角度修改。

(4)个体的拟合值计算

对于每个个体，需要计算其拟合值，拟合值的确定包含下面几个参数。(1)已放置的多边形的矩形宽度和板件的宽度的比值；(2)不能放入到板件的多边形个数。(3)每执行一轮放置(由于在一轮放置中不一定能全部放置，故需要多轮)就对拟合值递增。通过这些步骤，可以确定一个个体的拟合值，也就得到了一个解。

(5)群体

对于每个群体而言，其个体的数量可以进行配置。同样，也要配置遗传算法的迭代次数。因此，如果迭代的次数不超过群体的总个体数，那么每次迭代计算都只计算同一个总群中的不同个体(那些还没有确定拟合值的个体)，求其拟合值。如果配置的迭代次数超过了种群的总个体数，那么当前群体中的每个个体都会被计算。同时，还需要生成下一代的种群。下一代种群的生成，首先是利用当前群体中的个体互相杂交，生成下一代个体，同时下一代个体会有一定的突变率。这样，重复执行，使得下一代的个体总数等于当代的个体总数，那么下一代种群就确定了。对下一代继续执行迭代计算，直到达到迭代次数或者下一代的个体全都被计算，若出现后一种情况，就需要再生成下一代...。

3.5 实现

在 3.4 节中，介绍了 NFP 算法的实现过程。在 3.5 节中，介绍了遗传算法并把遗传算法的思想和本文需要讨论的排样问题结合起来。在本节中，将合并 NFP 算法和遗传算法，实现排样处理。

算法的总体过程如下：

(1)全局参数的初始化，分别是

1 需要确定板材的宽和高。

2 确定需要排样的部件。

3 一些配置信息的设置。

4 迭代次数。

5 构造一个 NFP 缓存(参见 3.5)，用来存放键值对。

(2)排样之前的操作

每次排样操作之前，需要对全部的部件(多边形)设置旋转度。开始时，我们在第一步中得到的是一个部件集合，它可以看做是唯一的一个个体，只是该个体中的部件(多边形，也可以称为基因)的旋转度(基因的表现形式)都是 0 度，也就是不旋转。那么在进行正常的排样操作之前，就需要先生成一个种群。第一个种群中的每个个体可以通过第一个个体(也就是那个唯一的个体)变异得到，且种群中的每个个体的拟合度均未计算。现在，已得到一个初始种群。在每次迭代中，会对种群中的一个个体计算拟合度。因此当迭代次数超过种群的个体总数时，需要利用当前种群生成下一代种群。而下一代种群的个体生成是通过父代的杂交以

及自身的变异可以得到。最后，得到的每个个体都有不一样的特性。在排样中的体现是，每个个体中各个部件的旋转度不同以及部件的排列顺序不同。经过指定次数的迭代，就可以获取最优的个体，对该个体中的部件做排样处理，就可以得到最佳排样。

(3)个体的排样处理

对下文算法中出现的符号给出一些说明。部件用 part 表示，部件集用 parts 表示，板材用 bin 表示，部件与板材生成的 NFP 用 partBinNFP 表示，拟合度用 fitness 表示，部件对应的平移向量用 vector 表示，平移向量的集合用 placement 表示。一个 placement 表示的是一块板材，而 placements 代表多个 placement，表示需要用多个相同的板材(因为有时一个板材不一定能放下全部的部件)。已放置的部件集合用 placed 表示。还有一些符号如 combinedNFP，finalNFP 等，由上下文来判断。

该算法过程的输入是 parts，输出是 placements 和 fitness。

每次迭代的执行过程如下：

While(parts 非空)

1.0 fitness += 1

2.0 For PART in parts

2.1 得到 PART 与 bin 的 partBinNFP

2.2 若 PART 是第一个放入 bin 的，将该 PART 放入 bin 最左下方，这个动作会生成该 PART 对应的 vector，把该 vector 放入到 placement 中，然后 PART 加入 placed 中，回到 2 继续 for 循环

2.3 for part in placed

2.3.1 与当前的 PART 生成 NFP(这里用的是缓存)

2.3.2 将 NFP 放入到一个临时集合 S 中

2.4 对 S 中的全部 NFP 求 Union 操作，生成 combinedNFP.

2.5 将 CombinedNFP 与 partBinNFP 执行 Difference 操作，生成 FinalNFP

2.6 for part in FinalNFP

2.6.1 利用该 part 和 placed 中的每个 part 生成最小矩形边框，

从而得到最小面积 minArea 与最小宽度 minWidth；该 part 与当前的 PART 可生成一平移向量 vector

2.6.2 利用 2.6.1 中的 minArea，minWidth 更新相关变量

2.7 对 2.6.1 中生成的 vector，将其加入到 placement 中, 当前 PART 加入到 placed 中

3 更新 fitness，fitness += minWidth，删除 parts 中那些已放入 placed 的 part

4 如果当前的 placement 非空，则将该 placement 放入到 placements 中，否则退出 while 循环

While 循环之外：更新 fitness，fitness += 还剩余的未放置 part 数。最后返回 placements 和 fitness。

注意：在上述算法的 2.4 和 2.5 中，我们提到了对 NFP 的一些操作。例如对 NFP 执行 Union 和 Difference 操作。这些操作会对给定的多个多边形进行求交，求差等等。在实现过程中，利用了 clipper 库。Clipper 库是一个开源的自由软件，它的主要作用就是对多边形和线段执行偏移操作(offsetting)和裁剪操作(clipper)-交叉(intersection)，并(union)，差(difference)和异或(exclusive-OR)。具体可以参考 clipper 网址(http://www.angusj.com/delphi/clipper.php)。

利用上述给出的算法，得到了 parts 对应的放置方式 placements，以及该放置方式的拟合度。然后，经过多次迭代，每次迭代都会生成一个较优解。通过比较不同的解，得到最优解。在第 5 章会描述算法的执行效果。

3.6 本章小结

本章研究的主体对象是排样算法的实现。首先介绍了一些常用的几何算法实现。接下来，讨论了几种排样算法的实现，分别是 BL 算法，下台阶算法和最低水平线算法，比较了他们各自的优缺点。最后，介绍了基于 NFP 和遗传算法的排样算法。NFP 算法的作用是将部件不重叠的排放，而遗传算法是整体的优化过程。结合这两个过程，最终的排样算法具有较优的性质。并且，通过测试论证，该算法确实是可行的。

第 4 章　刀具路径切割算法

4.1　问题求解

通过第 3 章的的论述，实现了板材优化下料算法。接下去，需要找到一个板材切割路径，使得切割走过的路径长度最短。已经存在一些朴素的算法实现了切割路径，如最近邻算法，基于二叉树优化的最近邻算法(沈志荣，2014)。下面简要分析这两个算法的优缺点。

对于最近邻算法，其基本思想如下：首先，找到一个离机床原点最近的部件。这个部件被标记为第一个被切割部件。接下来，需要寻找第二个被切割部件。对于最邻近算法就是寻找离第一个部件最近的部件。然后，依次进行直到全部的部件都被遍历完。这种方式需要考虑如下情况：当遍历到某个未被遍历的部件 A 时，下一步需要寻找离他最近的部件 B 。若部件 B 是已经遍历的，那么这时还需要找到离 A 第二近的部件 C 。若 C 又已遍历，那么就需要找到离 A 第三近的部件 D ...，按照这种方式持续下去。通过上述简单的描述，可以知道该算法需要对每个部件维护一个最近队列。也就是针对每一个部件，生成一个由近到远构成的队列。

算法的思想很简单，而且队列的生成可以在初始化部分就完成。但是，该算法的适用性不强。也就是说当板材上的部件数较少时，可以有较优的性能。而数量一旦达到上百上千时，性能就急剧下降。还有，除了性能上的差异，还存在致命的缺点。即不能产生较优解。因为算法每次都寻找最近的部件，这样，会产生局部最优，而全局特性差。例如对于下图所示的多边形排放，如果采用最近邻

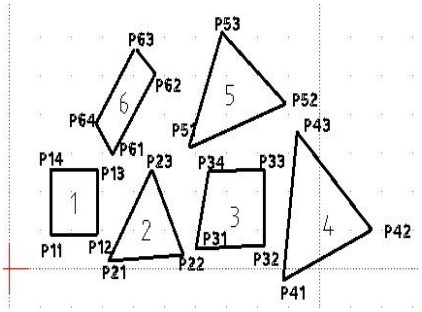

图 4.1 一些多边形

Figure 4.1 some polygons

算法，可以得出“最佳的”切割路径为 p11->p21->p31->p41->p52->p62->p11。如下图所示。

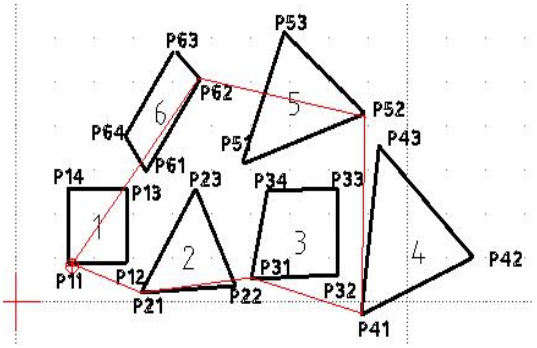

图 4.2 最近邻算法得到的切割路径

Figure 4.2 a cutting path using nearest neighbor algorithm

然而，该路径并不是最优的。下图展示了另一个切割路径。

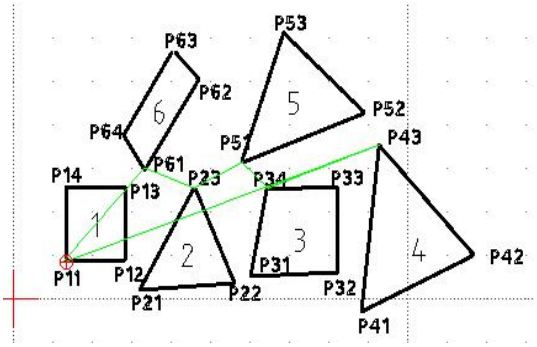

图 4.3 另一个切割路径

Figure 4.3 another cutting path

图 4.3 的切割路径 P11->P61->P23->p51->P34->P43->P11 明显比最邻近算法产生的路径来的短。因此针对最近邻算法的不足，在沈志荣(2014)论文中提出了一种基于二叉树的最近邻优化算法。该算法对最近邻算法进行了改进处理。大致的算法思路如下：在寻找下一个部件的时候，该算法并不是直接取最近的那一个。而是取最邻近的和次邻近的。然后通过一个判断来决定哪个可以作为下一个部件的候选者。这样，每次都是选取两个，若其中已存在遍历过的，那就选第三 / 四... 邻近的。在算法实现的时候，通过模拟一个二叉树来寻找最优路径。二叉树的根是第一个被选中的部件 A，而它的两个儿子是最邻近部件 B 和次邻近部件 C 。而

B 的两个儿子分别是与 B 最邻近的部件 D 和次邻近的部件 E (注意这里 D 和 E 的选取需要排除 A， B， C，因为 A， B， C 是已遍历过的)。

按照这种方式可以构造生成一颗二叉树。我们求最优解时只需要从根遍历到每个叶子节点就可以找出最优解。但是，如果部件数目过多，那么二叉树的深度过深，需要的存储空间也就过大，搜索时间也就很长。为了解决这种问题，作者使用了一点技巧。即当超过某一层(可以是 10 层)后，只寻找离该部件最近的部件(当然是未被遍历的)，而不在考虑次近的部件。

对于上诉的两种算法，本质上而言都是一种贪心的思想。一般都会得到局部最优解，但是对全局而言就不怎么好了。

本文提出了一种算法思路，基于的是蚁群算法和 TSP 求解模型。在接下来的论述中，在 5.2 节先描述 TSP 问题，在 5.3 节描述切割路径问题是如何转换为 TSP 问题，接着在 5.4 节介绍蚁群算法和求解过程。5.5 节测试算法的运行效率。第 6 章中给出算法在路径切割的上的运行结果。

4.2 TSP问题

TSP(Traveling Salesman Problem)问题，又称旅行商问题。描述的是：有一个旅行商，他要走遍 n 个城市，并且每个城市当且仅当经过一次，最终要回到最初所在城市。现在要解决的问题是，给出一个寻走方案，使得说走过的路程最短。

TSP 问题本身是一个经典的 NP 问题，也就是说我们无法找到一个多项式的精确解。对于一个具体的 TSP 问题，采用的一般是传统的方法，例如分支定界法，动态规划法，贪心法。而这些算法对于较小的复杂度(城市数较少)时，可以精确而快速的求解。一旦问题的规模较大(城市数多达上百上千)时，这些方法的计算过程会消耗大量的时间和内存空间。

因此，提出了一些近似算法来求解 TSP 问题。典型的算法如蚁群算法(张伟等，2011)，遗传算法，模拟退火算法等。这类算法都一致采用了启发式思想，故计算结果不一定是最优的，但是可以大量的降低空间和时间上的消耗。本文提出了一种改进算法，对传统的蚁群算法做了一些修改，并通过测试算法，证明其有一定的使用价值。在 5.4 节会对蚁群算法做具体的讨论。接下去，先看看切割路径问题是怎么转换为 TSP 问题的。

4.3 问题转换

切割路径问题证明转换为 TSP 问题。比较一下这两个问题的相同和不同之处。对与切割路径问题而言，路径是由一些部件(多边形)来描述的。每个部件都是一个多边形(闭合的)，而多边形具有多个顶点。若用数学语言来描述，可得

$$S=\{V_1,V_2,....,V_i,...,V_n\} \quad ...(4.1)$$

$$V_i=\{V_{i1},V_{i2},...,V_{ij},...,V_{i|V_i|}\} \quad ...(4.2)$$

$$1\le i\le n,\ 1\le j\le |V_i|$$

这里 n 表示的是多边形个数，S 表示部件集合(全部的多边形)。

其中V_i表示第 i 个顶点集(第 i 个多边形)，$|V_i|$表示该顶点集包含的顶点数。V_{ij}表示顶点集V_i中的一个顶点。那么最短路径可以表示为如下等式：

$$\begin{aligned} &D=\min\{\sum_{i\ne j}|V_{ik}-V_{jt}|\},\\ &V_{ik}\in V_i(1\le k\le |V_i|),V_{jt}\in V_j(1\le t\le |V_j|) \end{aligned} \quad ...(4.3)$$

这里的$|V_{ik}-V_{jt}|$表示的各个顶点之间的距离，D 表示最短路径。与切割路径问题不同的是，TSP 问题可以用如下的数学等式描述。

$$S=\{V_1,V_2,...,V_i,...,V_n\}\quad 1\le i\le n \quad ...(4.4)$$

其中V_i表示的每个城市，S 表示城市的集合。那么需要求解的最短距离为：

$$\begin{aligned} &D=\min\{\sum_{i\ne j}|V_i-V_j|\},\\ &1\le i\le n,1\le j\le n \end{aligned} \quad ...(4.5)$$

其中V_i-V_j表示城市之间的距离。

与路径切割问题相比，可以看出 TSP 问题中的V_i直接取自 S，而不是取自 S 中的某个元素的元素。因此，可以对路径切割问题做如下类似 TSP 问题的描述：有一个旅行商，他要遍历多个国家。每个国家都有多个城市。但是，他只需要遍历各个国家中的某一个城市(剩余的城市不需要遍历)，最后要返回初始城市。问他如何选择一条路径，使得能依次遍历每个国家(一个国家中选择一个城市遍历即可)，最后回到初始城市且走过的路程最短。

比较 TSP 问题和切割路径问题，将切割路径问题转化为 TSP 问题的关键就是从每个国家中选择一个“代表城市”来代表该“国家”。那怎么选择呢？如果采取暴力求解，可以生成组合数为$\prod_{i=1}^{n}|V_i|$，这个是无法想象的。因此，本文采用随机算法。算法的思想如下：首先在第一次选择过程中，对于每一个多边形，都随机选中一个顶点来“代表”该多边形。然后标记这些顶点为已选择过。在下一轮的选择中，可以选择这样的一个顶点(1)它没有被标记过(2)与上一次被选中的顶点相距最远。这样对于每一次选择，都能生成一个 TSP 问题，然后利用蚁群算法求解。在 5.4 节中，介绍蚁群算法的思想以及求解过程。在 5.5 节中测试本文实现的蚁群算法对 TSP 问题求解的效果。

4.4 蚁群算法

4.4.1 蚁群算法的简介

蚁群系统(Ant System)是有意大利学者 Dorigo Maniezzo 等(1996)于 20 世纪 90 年代首先提出来的。他们在研究蚂蚁觅食的过程中，发现单只蚂蚁的觅食行为比较简单。但根据观察，一群蚂蚁在觅食的时候总能够找到离食物最近的路线。这其中的原理是什么？

通过研究发现，每只蚂蚁在觅食过程中，会在经过的路途中释放一种“信息素”物质，其他蚂蚁可以觉察到这种物质。因此，该物质可能会对蚂蚁的觅食行为产生影响。

当某些路径上经过的蚂蚁越多的时候，这些路径上的”信息素”浓度就越高，那么其他蚂蚁选择这些路径的概率可能就越大，从而这些路径上的信息素浓度又会相应的增大。当然，路径上的“信息素”浓度会随着时间的流逝而逐渐的衰减。

4.4.2 符号、公式和算法

在介绍蚁群算法之前，先介绍一些符号。这些符号在算法的实现中有重要的作用。

4.4.2.1 符号

蚂蚁的数量用 m 表示。值一般为城市数量的 1.5-2 倍。若值较大，那每条路径上的信息素浓度趋于平均，正反馈作用减弱，导致解的收敛速度变慢。若值较

小，会导致某些路径可能都不会有蚂蚁经过，在迭代多次后，这些路径的信息素浓度会趋于 0，导致快速收敛，解的全局性差。

信息素因子用α表示。该值代表蚂蚁运动中积累的信息量。值的取值范围通常在$[1,4]$之间。该值对算法的影响很大，值得随机选择可能会使搜索结果趋于局部最优，或无法使得算法是随机的。

启发函数因子用β表示。该值表示启发式信息在指导蚁群搜索中的相对重要程度，取值范围通常在$[3,4.5]$之间。该值同样对整体算法的影响较大，若选择不佳，即会使算法趋于局部最优，或导致算法太过于随机而无法找到最优解。

信息素挥发因子用ρ表示。该值反映信息素的挥发速度，1-ρ则是信息素的稳定情况。ρ的取值通常在$[0.2\text{-}0.5]$之间。当ρ值过大时，容易导致局部最优。而ρ值过小时，收敛速度减低，导致最优解的生成需迭代更多次。

信息素常数用Q表示，指单只蚂蚁走完全程所释放的总量。该值的不当选择同样会出现局域最优的结果。

城市数量用 n 表示。

D_{ij}表示城市 i 到城市 j 之间的信息素值。

$\tau_{ij}(t)$表示在时刻 t，城市 i 和 j 之间的信息素浓度。

$p_{ij}^{k}(t)$表示在时刻 t，蚂蚁 k 从城市 i 向城市 j 转移的概率。

$\eta_{ij}(t)$是启发函数，表示蚂蚁从城市 i 转移到城市 j 的期望程度，这里我们取值$\frac{1}{D_{ij}}$。

$Allow_k$代表蚂蚁 k 可以访问城市的集合，初始时刻$Allow_k$有 n-1 个元素，即忽略蚂蚁所在的初始城市。随着迭代的进行，$Allow_k$的值减少，直到为空，表示蚂蚁走完所有城市。

$\Delta\tau_{ij}^{k}$表示在所有蚂蚁遍历完所有城市后，第 k 只蚂蚁对城市 i 与城市 j 之间信息素浓度的贡献值。

$\Delta\tau_{ij}$ 表示在所有蚂蚁遍历完所有城市后， 城市 i 和城市 j 之间信息素浓度的总累加量。

L_k 表示蚂蚁 k 遍历完所有城市后走过的总路程。

4.4.2.2 公式

本小节介绍 3 个公式，分别用于计算 $p_{ij}^k(t)$， $\tau_{ij}(t)$， $\Delta\tau_{ij}^k$。

$$p_{ij}^k(t)=\begin{cases}\dfrac{\left(\tau_{ij}(t)\right)^{\alpha}\bullet\left(\eta_{ij}(t)\right)^{\beta}}{\sum\limits_{s\in Allow_k}\left(\tau_{ij}(t)\right)^{\alpha}\bullet\left(\eta_{ij}(t)\right)^{\beta}} & ,j\in Allow_k\\ 0 & ,j\notin Allow_k\end{cases} \quad \text{... (4.6)}$$

从公式(4.6)中可以看出信息素因子 α 为信息素浓度 $\tau_{ij}(t)$ 的指数，启发函数因子 β 为启发函数 $\eta_{ij}(t)$ 的指数。这样就可以很好理解 α 和 β 的含义，他们分别决定了信息素浓度和转移期望对蚂蚁 k 从城市 i 转移到城市 j 的可能性的贡献程度.

$$\begin{cases}\tau_{ij}(t+1)=(1-\rho)\bullet\tau_{ij}(t)+\Delta\tau_{ij} & ,0<\rho<1\\ \Delta\tau_{ij}=\sum\limits_{t=1}^{m}\Delta\tau_{ij}^t\end{cases} \quad \text{... (4.7)}$$

公式(4.7)反映了信息素浓度的更新规律。可以看到，当所有蚂蚁遍历完所有城市后，信息素浓度在 t+1 时刻的值有两个部分组成。分别是上次所有蚂蚁遍历完所有城市后路径上信息素的残留值（$\tau_{ij}(t)$）及本次所有蚂蚁遍历完所有城市后每条路径上的信息素的增加量（$\Delta\tau_{ij}$）。

$$\Delta\tau_{ij}^k=\begin{cases}\dfrac{Q}{L_k} & ,\text{第k只蚂蚁从城市i走到城市j}\\ 0 & ，\text{否则}\end{cases} \quad \text{... (4.8)}$$

公式(4.8)表示的是某蚂蚁(编号 k)对城市 i 到城市 j 的信息增量的贡献值。直观的看，若蚂蚁 k 走过的路越长，那么沿途上城市与城市之间的信息素浓度的增量就越少。反之，城市与城市之间的信息素的浓度增量会相对增加。结合公式(4.6)和(4.7)看出，信息素浓度累计量大的路径在下次迭代中被选择的概率也越大。这也说明了信息素能用来确定最短路径。关于 $\Delta\tau_{ij}^k$ 的计算还有其他一些模型，

下面会叙述。这里给出的是蚂蚁圈(Ant Cycle System)模型，是求解 TSP 问题常用的一种方法。在下一节叙述用该模型求解的思路。

Dorigo 等人(1996)除了提出蚂蚁圈系统模型之外，还提出了其他两个，分别是(1)蚂蚁数量系统，(2)蚂蚁密度系统。

在蚂蚁圈系统中，所采用的是公式(4.8)的信息素浓度更新规则。从公式(4.8)中可以看出，L_k 表示的是蚂蚁走完一圈的路径值，表明蚂蚁圈系统模型是在全部蚂蚁走完之后，统一更新各个路径上的“信息素”浓度。而蚂蚁数量系统模型和蚂蚁密度系统模型有所不一样。

(1)蚂蚁数量系统

对公式(4.8)进行了修改，采用如下的信息素浓度更新公式

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{d_{ij}} & ,\text{第k只蚂蚁从城市i走到城市j}\\ 0 & ,\text{否则}\end{cases} \quad \ldots (4.9)$$

与公式(4.8)不同的是，这里不用 L_k，而是改用 d_{ij}。表明当一只蚂蚁从城市 i 走到城市 j 时，立刻更新该段上的信息素浓度。若在同一轮迭代中又有一只蚂蚁经过该路径时，所使用的信息素浓度是就是更新后的最新值。这与蚂蚁圈系统模型就不一样了，蚂蚁圈系统模型是在全部蚂蚁走完一圈(迭代完一轮)后，才一次性更新全部路径上的信息素浓度。

(4.7)蚂蚁密度系统

也对公式(4.8)进行了修改，采用如下公式

$$\Delta\tau_{ij}^{k}=\begin{cases}Q & ,\text{第k只蚂蚁从城市i走到城市j}\\ 0 & ,\text{否则}\end{cases} \quad \ldots (4.10)$$

与公式(4.9)相同的是同步更新模式，即蚂蚁每走一步就更新相应路段的信息素浓度。不一样的是这里直接按常数 Q 累计浓度。

对比三种模型，第二种和第三种的更新采用的是局部信息，而第一种采用全局信息，是在所有蚂蚁走完一圈后再全局更新。

4.4.2.3 算法

本文采用蚂蚁圈系统模型作为基础模型，加以改进，提出一种优化算法。以下论述该算法总体思路。首先有 m 只蚂蚁，n 个城市。

(1)将 m 只蚂蚁随机分配到 n 个城市。初始化城市与城市之间的信息素浓度，记录城市与城市之间的距离。

(2)进行 N 次迭代，每次迭代执行如下过程：

(2.1)每只蚂蚁寻找下一个城市。(这里不实时更新蚂蚁走过的城市与城市之间的信息素浓度，最后统一更新)。

(2.2)当所有蚂蚁都走完所有城市之后，执行(2.3)否则对每只蚂蚁继续执行(2.1)。

(2.3)由于每只蚂蚁已走完所有城市，计算出每只蚂蚁所走的总路程。并更新最小路程与相对应的城市路径。

(2.4)利用每只蚂蚁的所走路径，更新城市与城市之间的信息素浓度。

(2.5)最后，把 m 只蚂蚁又随机放到 n 个城市，进入下一次迭代。

以上对蚁群算法的蚂蚁圈模型进行的描述，然而，在实际的算法实现中，还伴随着许多的随机因素，如 m 只蚂蚁的随机分配，α 和 β 值等等。这些因素对每次迭代过程，以及最终的解都有很大的影响。

因此，接下来对蚁群算法进行一些修改，增加改进。

(1)m 只蚂蚁需平均分配到各个城市。这个操作是后续操作的前提条件。

(2)在初始的蚁群算法中，每只蚂蚁在寻找下一个城市的时候，利用的是 $p_{ij}^{k}(t)$ 来确定下一个城市被选中的概率。而算法一般都是利用 Round Robin 算法。该算法实现简单，可是效果却不一定很好。在改进算法中，先计算出在当前城市中的蚂蚁数，然后对每只蚂蚁选择这样的一个城市：先对未遍历的城市由近及远排序，然后采用轮寻的方法，依次对每只蚂蚁选择一个城市。例如，未遍历的城市还有 5 个，当前城市的蚂蚁有 20 只。并对 5 个城市按距离排序(假设 0 号最近，4 号最远)。那么 0、5、10、15 号蚂蚁走 0 号城市(最近的)；1、6、11、16 号蚂蚁走 1 号城市(次近的)等等。注意，这里包括了全部未遍历的城市，另一种做法是选择一部分未遍历的，而不全都选择。我把这种方式称做“第一次选择”，即在初始城市中的蚂蚁选择下一个城市的时候，可以使用这种方式。

(3)下一次迭代开始前的蚂蚁分配。在最初的算法中，利用了直接随机分配。这里的改进如下：因为上一次迭代结束，可以得到一些较优化路径，每个路径都有一个初始城市，那么这些初始城市显然可以在该次迭代前分配到更多的蚂蚁。

方法如下，根据蚂蚁总数和总迭代次数，可以计算出一个值，即在每次迭代中选择多少蚂蚁放在那些较好的初始城市中。例如，蚂蚁总数为 1000，迭代次数为 50 次，较好的初始城市为 6 个(总城市数为 50)，那么我们在第二次迭代中，先取 20 只蚂蚁平均放在该 6 个较好的初始城市，剩下的 980 只蚂蚁还是平均分配到所有的城市中。若第二次迭代后，可以得到 4 个最优城市，那么我们就先选择 40 只蚂蚁平均分配到这个 4 个城市，然后将剩余的 960 只平均分配到所有城市。总而言之，随着迭代的进行，较好的初始城市的初始蚂蚁数会越来越多，较差的初始城市的初始蚂蚁数越来越少。

(4)对(2)的一个补充。在每次迭代的开始时，蚂蚁全都放在初始城市中，我们需要对每只蚂蚁确定下一个城市(第二个城市)。算法实现时可采用(2)中的方法(“第一次选择”)，而在后面的城市选择中(如第三、第四，...个城市)都使用原来的方式。这样也说明了确定下一个城市的选择方式有两种情况，分别是(2)中的方法，或者是原来的方式。再推广一下，可以在每次选择下一个城市时，随机采用一种方式。或者，还有其他的组合方式。

上面说到了 4 个方面，其中第 1 个改进是确定的。每个城市能分配到的蚂蚁数必须是几乎一致的，这对后续的(2)(3)(4)来说都是前提条件。对于第三个改进，表明并不会在每次迭代前都随机分配蚂蚁，而是考虑了上一次迭代的结果来分配蚂蚁，第三个改进是针对每次迭代而言的。最后对于第二和第四个改进，我们采用了贪心的思想，只选择那些较近的城市作为候选对象。同时，由于在一个初始城市中，有多只蚂蚁存在，故可以让这些蚂蚁平均分配到“候选城市”中，这个优化是针对迭代中的“每次”城市选择而言的，粒度相对较细。改进的算法流程图如下所示：

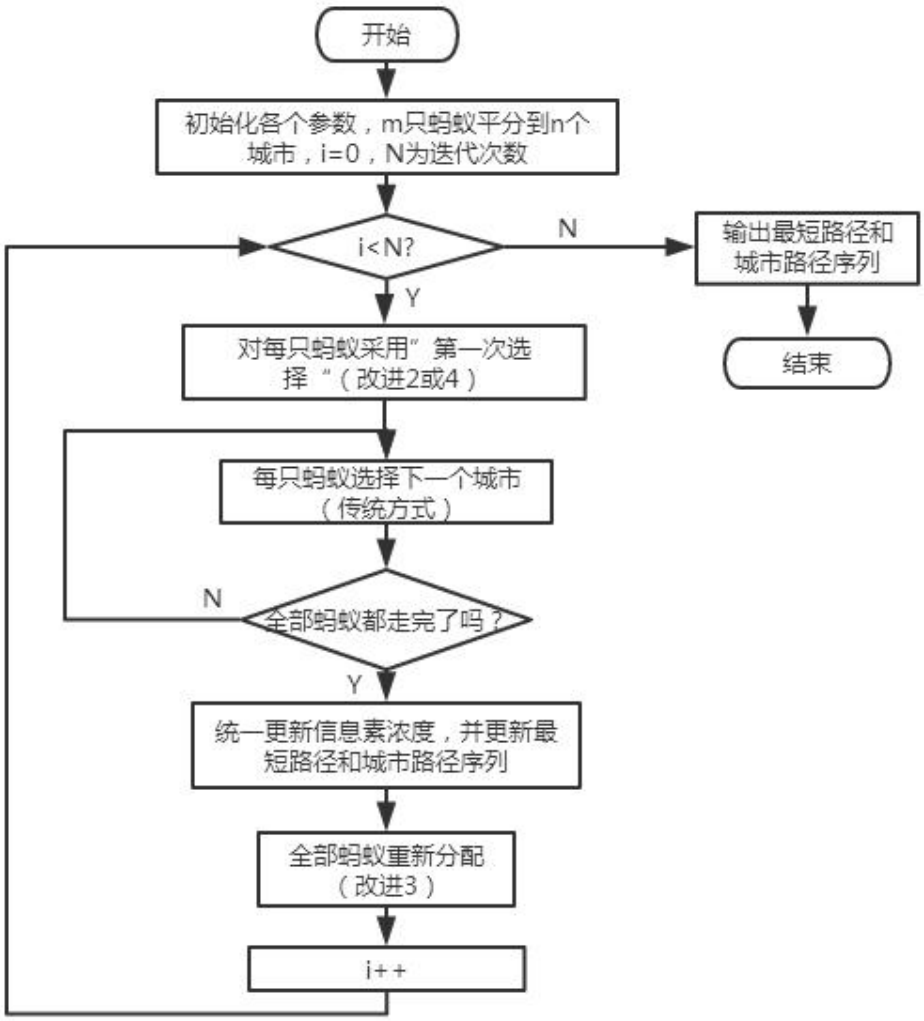

图 4.4 改进的蚁群算法流程图

Figure 4.4 flow chart of improved ant colony algorithm

4.5 算法测试

在 4.4 节中，介绍了蚁群算法的实现。那用其求解 TSP 问题，效果如何，以下测试结果。实验数据来源于(https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/)。TSPLIB 是一个数据库，包含了各种 TSP 问题的实例，如美国 48 个城市的坐标分布等。在 TSPLIB 中，我们用到的是对称的旅行商问题(symmetric TSP)的相关数据，即节点 A 到节点 B 的距离等于节点 B 到节点 A 的距离(对称)。

表 4.1　改进的蚁群算法的测试与比较

Table 4.1 testing and comparison of improved ant colony algorithm

数据集	最优结果(来源网站统计)	测试结果	比较结果(相对最优值的增加百分比)
a280	2579	3078	19.35%
ali535	202339	210248	3.91%
att48	10628	20138	26.44%
att532	27686	37425	20.73%
bayg29	1610	1720	6.83%
Bays29	2020	2457	21.63%
Berlin52	7542	10237	35.73%
Bier127	118282	120743	10.53%
Brazil58	25395	29328	15.49%
Brd14051	469385	492952	6.30%
Brg180	1950	2013	3.23%
Burma14	3323	3579	7.70%
ch130	6110	7213	18.05%
Ch150	6528	8752	34.07%
d198	15780	18432	16.81%
D493	35002	38038	8.67%
D657	48912	52397	7.13%
Gr17	2085	2630	26.14%
Pr226	80369	10238	24.72%
St70	675	680	0.74%
Ts225	126643	139762	10.36%
Pla85900	142382641	142949284	6.02%
Ulysses16	6859	7234	5.47%

通过和最优值进行比较，可以得出如下的结论：本算法的结果比起最优值都要大，但绝大部分的增加率控制在 30%以内，表明算法是可行的。

4.6 本章小结

本章论述的是路径切割问题的求解过程，提出了一种路径切割算法。先介绍了一些算法的实现过程，如最近邻算法，基于二叉树优化的最近邻算法等，并分析了他们的优缺点。接着，把切割路径问题转化为 TSP 问题。同时，介绍了蚁群算法的一些概念，并利用改进的蚁群算法求解 TSP 问题。最后，通过对蚁群算法测试，证明了该算法具有一定的适用性。

第 5 章　实验结果

在前面的章节中，我们依次讨论了 DXF 文件的读取(第二章)，排样算法(第三章)，切割路径算法(第四章)。以下是各个章节的实验结果。

5.1　DXF数据读取

在第三章中，我们介绍了 DXF 的读取算法和相应的多边形存储结构。回顾一下，多边形的存储就是对多边形的顶点按逆时针存储即可。以下演示算法的实现效果。本实验用到的工具为 QCAD。首先在 QCAD 中，画出如下图形。

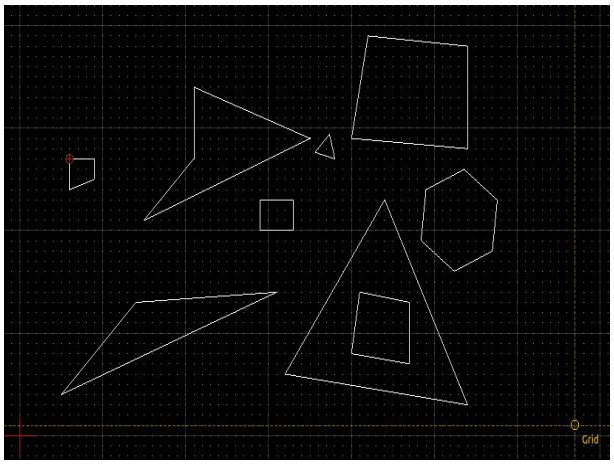

图 5.1　QCAD 生成的一些图形

Figure 5.1 draw some shapes using QCAD

在图 5.1 中，红色的十字表示的是坐标原点，x 轴为水平向右，y 轴垂直向上，是标准的二维笛卡尔坐标系。假设生成的文件为 TEST.dxf。接下来，需从 TEST.dxf 文件中，读取全部的多边形。利用 libdxf 库，可得如下的实验数据。

表 5.1 多边形的顶点数据

Table 5.1 vertices data of polygons

顶点标号	顶点数据
0	(210,340) (350,290) (150,210) (210,270)
1	(540,30) (320,60) (440,230)
2	(140,130) (50,40) (310,140)
3	(410,140) (400,80) (470,70) (470,130)
4	(420,390) (400,290) (540,280) (540,380)
5	(60,270) (60,240) (90,250) (90,270)
6	(290,230) (330,230) (330,200) (290,200)
7	(373.66,293.66) (356.34,276.34) (380,270)
8	(484.019,190.359) (524.019,160.359) (570,180) (575.981,229.641) (535.981,259.641) (490,240)

从这些数据中可以看出，标号为 0、1 和 6 的这 3 个多边形的顶点是按顺时针存储的。因为这个与 DXF 文件的存储格式有关，即 DXF 是先绘制的先存储。那怎么判断多边形顶点是逆时针存储还是顺时针存储的？以及如何都转化为逆时针存储。从 3.2 节的几何算法知识可知，能利用多边形的有向面积来判断。若以右手法则来规定正方向，那么多边形顶点按逆时针排序后得到的有向面积就是正的；而多边形顶点按顺时针排序后的有向面积就是负的。这样处理之后，得到的结果都以逆时针排序。对上述结果修正后，得到如下数据。

表 5.2 修改后多边形的顶点数据

Table 5.2 modified vertices data of polygons

顶点标号	顶点数据
0	(210,270) (150,210) (350,290) (210,340)
1	(440,230) (320,60) (540,30)
2	(140,130) (50,40) (310,140)
3	(410,140) (400,80) (470,70) (470,130)
4	(420,390) (400,290) (540,280) (540,380)
5	(60,270) (60,240) (90,250) (90,270)
6	(290,200) (330,200) (330,230) (290,230)
7	(373.66,293.66) (356.34,276.34) (380,270)
8	(484.019,190.359) (524.019,160.359) (570,180) (575.981,229.641) (535.981,259.641) (490,240)

其实与表 5.1 相比，只是对 0、1 和 6 标号的顶点序反转了，使得是逆时针序。

然后可以将这 9 个多边形存储在 Polygon 结构中。

通过对 DXF 的数据提取和结果的排序，可以得到所有多边形的数据，为后续的板材排样问题和切割路径问题提供实验数据。

5.2 板材优化下料算法

在第三章中，介绍了排样算法的整体实现思路。回顾一下，排样的实现主要依靠了两个子过程。一个是放置过程，另一个是优化过程。放置过程利用的是 NFP 算法，优化过程利用的是遗传算法。在排样算法中，使用的多边形数据结构相对复杂，故接下来先描述具体的多边形结构，然后观察算法的实验结果。

在 5.1 节中获取了原始的多边形数据。然而，这些数据不能立刻用于排样算法。因为在排样过程中，需要标记每一个多边形，以及该多边形是否存在孔洞或者该多边形是否是另一个多边形的孔洞等等。下面给出描述多边形的数据结构：

```
Struct Polygon{
    Point vertics[];
    Point[] holes[];
    Point[] parent;
    Int id;
    Double offsetX，  offsetY;
    Struct Config *config;
    Int rotation;
}
```

比起先前的数据结构描述(见 2.3 节)，这里更加具体化。在当前的结构中，vertics 用于表示多边形本身的顶点，holes 用于描述该多边形(部件)中存在的孔洞(可以存在多个)，parent 表示该多边形若是某个部件的孔洞，那么 parent 指向那个部件(多边形)。id 是该多边形的标识，offsetX 和 offsetY 表示该多边形距原点的偏移值。config 为配置信息，这个后文详述。最后一个 rotation 表示该多边形可能的旋转数。如 rotation 为 4，表示该多边形的旋转度数可以为 0、90、180 和 270。因此旋转度数能用于遗传算法的个体变异和杂交中，这在 3.5.2 节的个体部分有介绍。

特别需要说明一下的是配置信息。在 3.6 节的排样算法实现中，第一步的初始化部分提到了配置信息的初始化。在配置信息中给出一些参数，主要有两类，一是用于遗传算法的参数设置，例如种群个体总数，变异率。另一是设置排样的间距，以及是否能将多边形(部件)放置在某个多边形的孔洞内等。

以下为实验结果。对 5.1 节得到的数据，进过 4 次迭代，每次迭代输出一个排样，对应于如下一个图。其中图 1 具有最小的拟合值(最好的排样效果)。

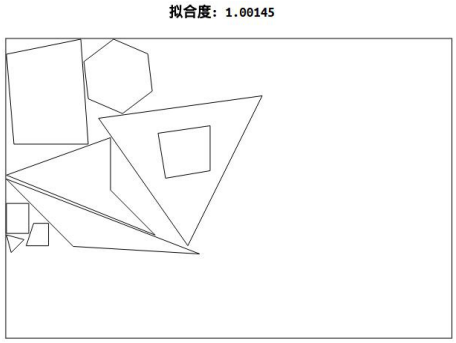

图 5.2 排样结果 1

Figure 5.2 nest case 1

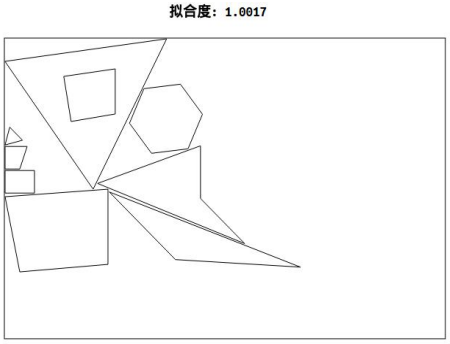

图 5.3 排样结果 2

Figure 5.3 nest case 2

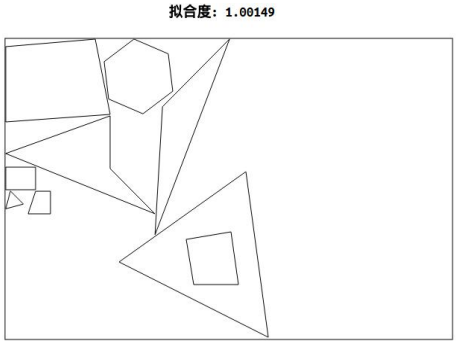

图 5.4 排样结果 3

Figure 5.4 nest case 3

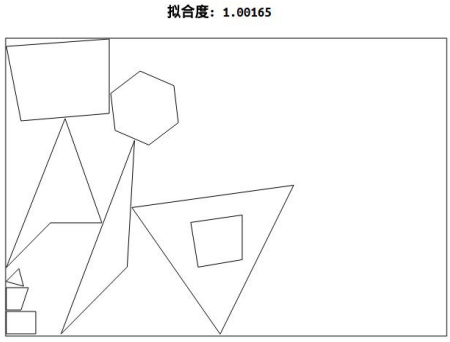

图 5.5 排样结果 4

Figure 5.5 nest case 4

若修改部件间的间距，并重新迭代 4 次，可以得出如下图示。

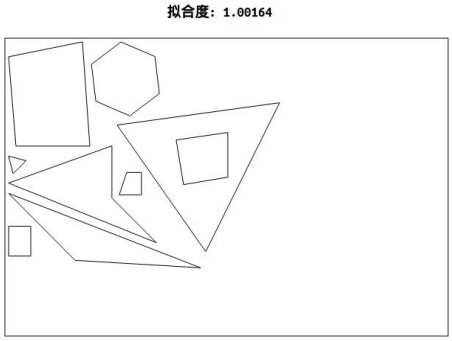

图 5.6 排样结果 5

Figure 5.6 nest case 5

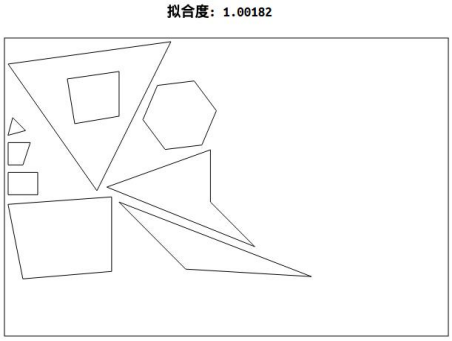

图 5.7 排样结果 6

Figure 5.7 nest case 6

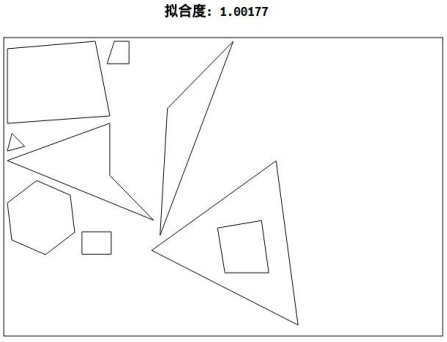

图 5.8　排样结果 7

Figure 5.8 nest case 7

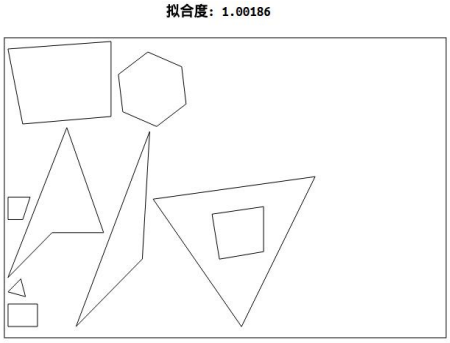

图 5.9　排样结果 8

Figure 5.9 nest case 8

通过改进的排样算法，得到了较优的排样结果。

5.3　刀具路径切割算法

在 5.2 节中，得到了排样后的效果图。在本节需针对该排样，得出最短的切割路径。在 4.4.2.3 节中描述了算法的实现过程。以下还是对 test.dxf 文件以及 5.2 节中的图 5.6 到图 5.9，给出如下的较优最短路径。

以下图示中，红色的点代表切割起点，粉丝的点代表第二个点，其他蓝色的点是后续的点．切割动作从红色的起点开始，绕红色点所在的多边形一周；再到粉色的点，绕粉色点所在的多边形一周；继续到后续的蓝色的点，绕它们所在的多边形一周，最后回到红色的起点。

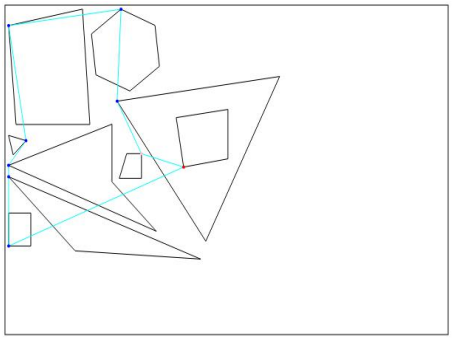

图 5.10 切割路径 1

Figure 5.10 cutting path case 1

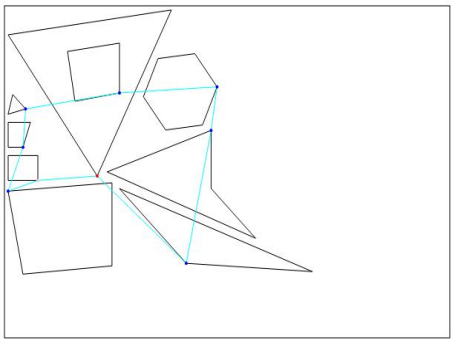

图 5.11 切割路径 2

Figure 5.11 cutting path case 2

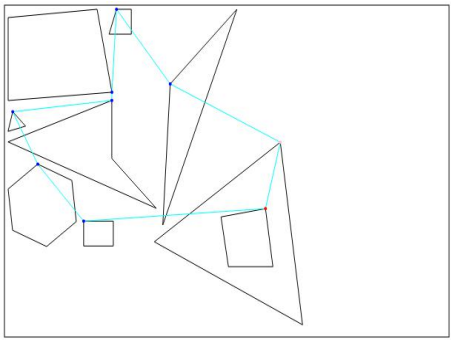

图 5.12 切割路径 3

Figure 5.12 cutting path case 3

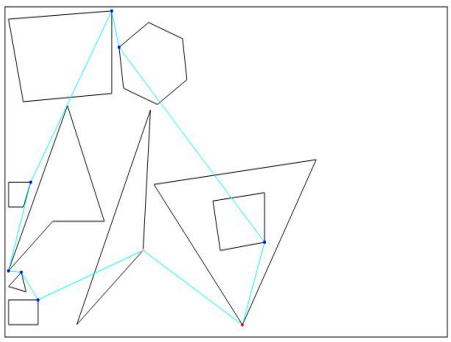

图 5.13 切割路径 4

Figure 5.13 cutting path case 4

表 5.3 切割路径值

Table 5.3 length of cutting path

图标号	切割路径的长度
5.10	830.9580336539182
5.11	1031.1109356961051
5.12	941.3596104452558
5.13	1178.19268962737

表 5.3 给出了相对应图的切割长度。从图 5.10 到图 5.13 中可以看出一些不足之处：(1)对于有孔洞的部件而言，该算法并没有先切割孔洞，在切割外轮廓(图 5.13)。(2)还可以看到当切割某一个带孔洞的部件时，它先把外轮廓给切割了，然后再切割其他的多边形，后续再来切割孔洞(图 5.11)。而这些在实际的情况中，一般而样，对于有孔洞的部件而言，先是切割完孔洞，然后切割自身。

5.4 其他案例

以下给出了一些其他案例，可以看出算法的效果。

5.4.1 板材优化下料算法的案例

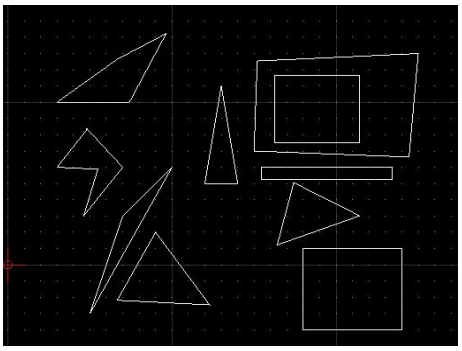

图 5.14　DXF 样例 1

Figure 5.14 DXF sample 1

图 5.15　样例 1 的排样

Figure 5.15 nest for sample 1

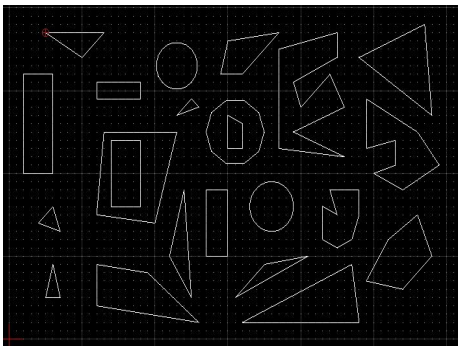

图 5.16　DXF 样例 2

Figure 5.16 DXF sample 2

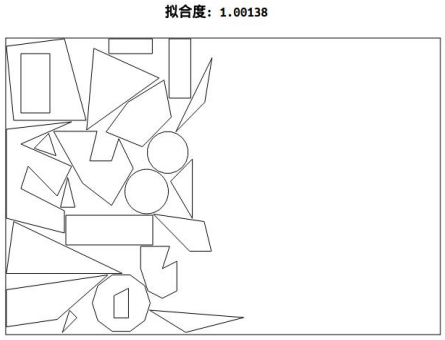

图 5.17　样例 2 的排样

Figure 5.17 nest for sample 2

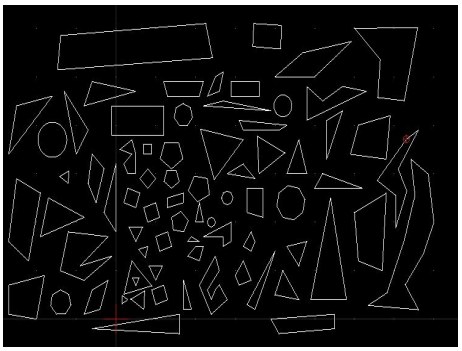

图 5.18 DXF 样例 3

Figure 5.18 DXF sample 3

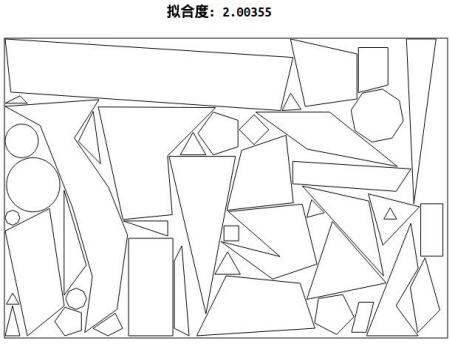

图 5.19 样例 3 的一部分排样

Figure 5.19 part 1 of nest of sample 3

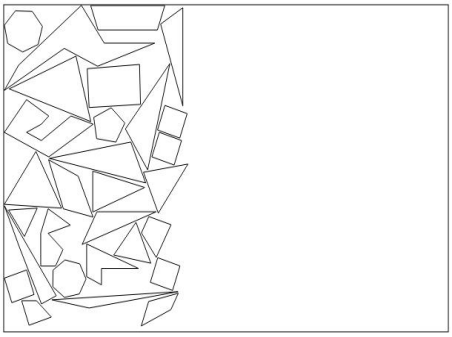

图 5.20 样例 3 的剩余部分排样

Figure 5.20 rest part of nest of sample

5.4.2 刀具路径切割算法的案例

图 5.21 排样 1

Figure 5.21 nest sample case 1

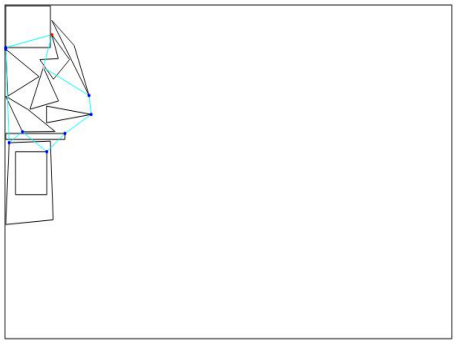

图 5.22 排样 1 的切割路径

Figure 5.22 cutting path of nest sample 1

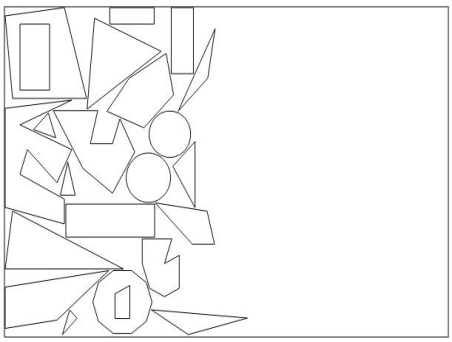

图 5.23　排样 2

Figure 5.23 nest sample case 2

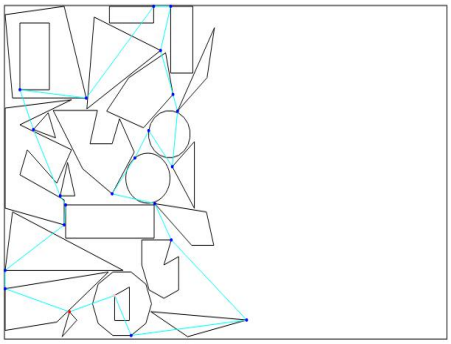

图 5.24　排样 2 的切割路径

Figure 5.24 cutting path of sample case 2

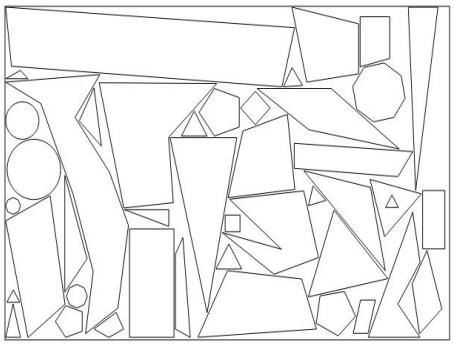

图 5.25　排样 3

Figure 5.25 nest sample case3

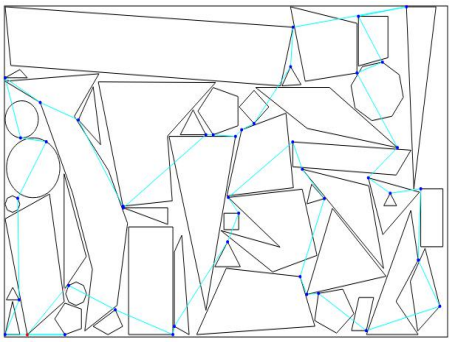

图 5.26 排样 3 的切割路径

Figure 5.26 cutting path of nest sample 3

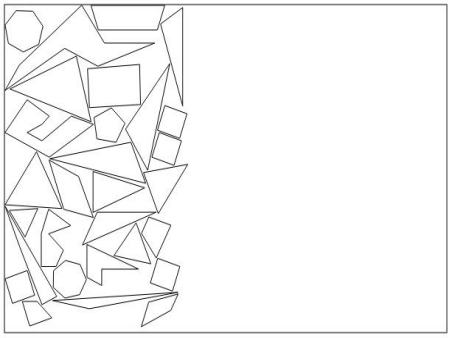

图 5.27 排样 4

Figure 5.27 nest sample case 4

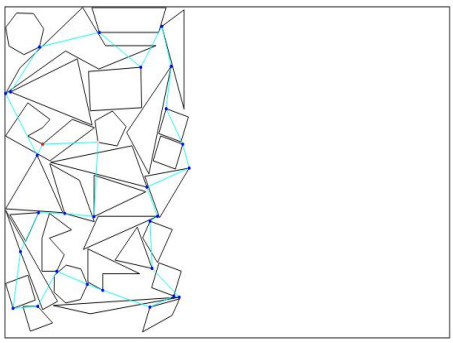

图 5.28 排样 4 的切割路径

Figure 5.28 cutting path of nest sample 4

5.5 本章小结

本章从 test.dxf 出发，提取文件中各个多边形的数据，并加以优化。然后借助排样算法，得到多边形的排样结果。最后，根据排样结果并利用切割路径算法，生成最终的较优的最短路径。

第 6 章　结论与展望

6.1　工作总结

板材排样问题和路径切割问题是制造业经常面临的问题。排样问题是将小部件(零件)布局到板材上，该问题广泛出现在机械、钢材、玻璃等行业中。在钣金排样领域，板材排样和切割路径算法密切相关。一般而言，首先需要完成部件的排样处理，然后进行切割操作。总结本论文的研究工作，主要的结论如下：

(1)首先是数据的提取。DXF 文件格式是一种通用的数据交换格式，且其格式相对复杂。数据读取的思想采用了表驱动模型，采用了现成的库，将 DXF 文件中存储的多边形数据提取出来，并做加工处理，为后面的排样算法和切割路径算法提供数据支持。

(2)排样部分包括了两个子算法。一个是 NFP 算法的实现过程，另一个是遗传算法的实现。NFP 算法的目标是将不同的部件不重叠放置，而遗传算法是优化全局的排样。通过结合这两个过程，实现了一个改进的排样算法。并通过算法分析和测试，验证了算法的可行性和有效性。

(3)对于切割路径问题而言，先将其转化为 TSP 问题。接着利用蚁群算法对其求解。在改进的蚁群算法的中，提出了“第一次选择”的思想，并通过对算法的分析和测试，证明了算法的可行性和有效性。

6.2　工作展望

本论文从 DXF 文件读取出发，实现了排样算法和切割路径算法。可是，对于实现的排样算法和切割路径算法，都存在一定的局限性，主要有以下几个方面的问题：

(1)对于排样算法，只考虑了闭合的多边形这种情况。尽管算法可以处理存在孔洞的多边形，但是还是无法解决能否在孔洞中继续放置部件的问题。第二个问题是，因为只针对多边形这种形状，并没有考虑与曲线相关的形状，例如 B 样条曲线，Bezier 曲线，圆形，椭圆形等等。第三是只针对闭合的形状，而不闭合的也没有进行考虑。因此，这些地方是在后续工作中需要特别改进。最后，由于

使用了启发式的遗传算法，测试结果时好时差，那么说明参数的选择以及算法还是需要改进。

(2)对于路径切割算法，一个是问题转化部分。将路径切割算法转化为 TSP 问题采用了一定的随机因子，而并没有提前根据多边形的特性有针对性的选择“代表”顶点。也就是说，选择算法还有待改进的地方。接着是蚁群算法，同样这是一个启发式算法，具有很强的随机性，故生成的解具有很不确定性。并且有一个很大的缺点，就是对于有孔洞的部件而言，该算法并没有先寻路到孔洞，在寻路到外轮廓。其二，还可以看到当寻路到某一个带孔洞的部件时，它先经过外轮廓，然后再经过其他的部件，最后最寻路到孔洞．而在实际的情况中，一般而言，对于有孔洞的部件而言，先是寻路(切割)完孔洞，然后寻路(切割)外轮廓。

(3)接下来的工作可以对切割路径生成 NC 代码。因为 NC 代码是可以在数控机器上切实可执行的，这样就可以判断切割路径算法在实践中是否确实高效。同时，经过以上的工作，可以开发出一个完整的套料软件用于实践。

参考文献

曹德列.不规则图形数控切割关键技术研究[D].武汉:华中科技大学工业工程,2012.

高伟增,张宝剑,陈付贵,等.基于遗传算法的切割路径优化[J].西南交通大学学报,2005,40(4):5.

龚清洪.基于文件的图元优化排序[J].计算机应用,2006,26(1):169~171.

韩志仁,林德强.基于 BL 碰撞算法求解排样过程中干涉的问题[J].沈阳航空航天大学学报,2011,28(02):1-4.

扈少华,张淋江,潘立武,管卫利.矩形件套裁排样的一种优化解法[J].机械设计与制造,2018(06):219-221+225.

罗强,李世红,袁跃兰,饶运清,刘泉辉.基于复合评价因子的改进遗传算法求解矩形件排样问题[J].锻压技术,2018,43(02):172-181.

沈志荣.数控切割排序和套料的算法研究与软件实现[D].华侨大学,2014.

张青,刘芳.矩形件排样最优化问题求解[J].现代电子技术,2017,40(22):72-74.

张伟,安鲁陵,张臣,等.基于蚁群算法的矩形件切割路径优化[J].机械科学与技术,2011,30(3)：390~393.

周家智,尹令,张素敏.基于遗传模拟退火算法的布局优化研究[J].图学学报,2018,39(03):567-572.

Agarwal PK, Flato EHD. Polygon decomposition for efficient construction of Minkowski sums[J]. Computational Geometry Theory and Applications 21, 2002, pages 39-61.

Arango Serna MD, Serna Uran CA. A Memetic Algorithm for the Traveling Salesman Problem[J]. IEEE Latin America Transactions,2015,13(8):2674-2679.

Art RC. An approach to the two dimensional irregular cutting stock problem[J]. IBM Cambridge Scientific Centre,1996,Report 36-Y08.

Bennell JA, Dowsland KA, Dowsland WB. The irregular cutting-stock problem - a new procedure for deriving the no-fit polygon[J]. Computers and Operations Research 28, 2001,271-287.

Blazewicz J, Hawryluk P, Walkowiak R. Using Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem[J]. Tabu Search (eds. Glover, F., Laguna, M., Taillard, E., Werra, D.), 1993, Vol. 41 of Annals of Operations Research, Baltzer, J. C. AG.

Brooks RL , Smith CAB, Stone AH, et al. The dissection of rectangles into squares[J]. 1940.

Burke EK, Hellier RS, R.Kendall G,et al. Complete and robust no-fit polygon generation for the irregular stock cutting problem[J]. European Journal of Operational Research, Elsevier, 2007,vol.179(1), pages 27-49, May.

Chentsov AG, Chentsov AA. Dynamic programming in the routing problem with constraints and costs depending on a list of tasks[J]. Doklady Mathematics, 2013, 88(3):637-640.

Coffman EG, Garey MR, Johnson DS, et al. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms[J]. SIAM J. Comput., 1980, Vol 9, pp 808-826.

Dantzig GB. Maximization of linear function of variables subject to linear inequalities[J]. Activity Analysis of Production & Allocation, 1951：339-347.

Dorigo M, Maniezzo M, Colorni A. The ant system: Optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man, and Cybernetics－Part B, 1996, 26(1):9-41.

Dowsland KA, Dowsland WB. Packing problems[J]. European Journal of Operations Research 56, 1992, pages 2-14.

Eisemann K. The Trim Problem[J]. Management Science, 1957, Vol 3, pp 279-284.

Fischetti M, Salazar González JJ, Toth P. The symmetric generalized traveling salesman polytope[J]. Networks, 1995, 26(2):113-123.

Fischetti M, Salazar-Gonzalez JJ, Toth P. A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem[J]. Operations Research 45 (3), 1997, pp. 378-394.

Gilmore PC, Gomory RE. A Linear Programming Approach to the Cutting-Stock Problem[J]. Operations Research, 1961, vol 9, pp 849-859.

Gilmore PC, Gomory RE. Multistage Cutting Stock Problems of Two and More Dimensions[J]. Operations Research, 1965, vol 13, pp 94-120

Jain P, Fenyes P, Richter R. Optimal Blank Nesting Using Simulated Annealing[J]. Journal of Mechanical Design (Transactions of the ASME), March 1992, Vol 114, pp 160-165

Kantorovich LV, Zalgaller VA. Optimal Calculation for Subdivision in the Material Industry. Leningrad. Lenizdat. 1951, p198.

Korotaeva LN, Trukhin MP, Chentsov AG. On the routing of connections[J]. Institute of Mathematics & Mechanics, 1997, (12):175–192.

Kroger B. Guillotineable bin packing: A genetic approach[J]. European Journal of Operational Research, 1995, 84(3): 645~661.

O'Rourke J. Computational Geometry in C, second ed. Cambridge University Press, 1998,ISBN 0-521-64976-5.

Paull AE. Linear Programming: A Key to Optimum Newsprint Production. Pulp Paper Magazine. Canada. 1956, Vol 57.

Petunin AA, Chentsov AG, Chentsov PA. About a Routing Problem of the Tool Motion on Sheet Cutting[J]. N.n.krasovskii Institute of Mathematics & Mechanics, 2015.

Silberholz J, Golden B. The Generalized Traveling Salesman Problem: A New Genetic Algorithm Approach[J]. Operations Research/computer Science Interfaces, 2007, 37:165-181.

Snyder LV, Daskin MS. A random-key genetic algorithm for the generalized traveling salesman problem[J]. European Journal of Operational Research, 2006, 174(1):38-53.

Vajda S. Trim Loss Reduction. Readings in Linear Programming[J]. New York, Wiley, 1958, pp78-84.

致 谢

时光荏苒，转眼间三年的紧张而又充实的硕士生涯即将画上句号。在三年的学习期间，我得到了很多老师、同学和朋友的关怀和帮忙。在学位论文即将完成之际，我要向所有期间给予我支持、帮忙和鼓励的人表示我最诚挚的谢意。

首先，我要感谢我的导师王品老师对我的教导。从论文的选题、构思、撰写到最终的定稿，您都给了我悉心的指导和热情的帮忙，让我的毕业论文能够顺利的完成。王老师对工作的认真负责、对学术的钻研精神和严谨的学风，都是值得我终生学习的。

其次，感谢国科大和沈计所的全体领导和老师，由于他们的悉心教导，我学到了专业的计算机相关的专业知识，掌握了扎实的技能。在此，我特别感谢师兄邹善席，他在我迷茫的时候，给予了极大的鼓励和支持。同时，也感谢论文指导组老师对论文的修改提出了宝贵意见，使我的论文更加完善。

最后，感谢我的家人在此期间给予我的包容、关爱和鼓励，以及所有陪我一路走来的同学和朋友，正是由于他们的支持和照顾，我才能安心学习，并顺利完成我的学业。

毕业在即，在今后的工作和生活中，我会铭记师长们的教诲，继续不懈努力和追求，来报答所有支持和帮忙过我的人！

2019 年 6 月