廣東工業大學

本科毕业设计（论文）

基于贪心算法与图论算法相结合的最低成本产品召回系统的研究与设计

学　　院　机电工程学院

专　　业　机械设计制造及其自动化

（机械电子工程方向）

年级班别　2013 级（6）班

学　　号　3113000246

学生姓名　陈敏烽

指导教师　李志

2017 年 6 月

摘 要

现今，人们越来越注重产品质量，所以产品出现问题时，企业需要采取将产品召回企业本部的情况也不少见，这种情况称为产品召回。产品召回往往使得企业损失惨重，本文针对产品召回建立数学模型，为企业计算产品召回中的成本最优的运输方案。

本文在第二章通过建立数学模型、简化数学模型，将实际问题转换成数学问题进行解决，并在第三章分析比较不同的算法，利用计算机算法来将模型解决，然后运用数学公式方法来对算法进行证明，佐证了贪心算法适用于本文建立的数学模型；接着运用计算机高级编程语言对其实现，所以本文设计出能够快速解出产品召回中成本最优方案的系统。

最后得出了本文的结论，讨论了该系统的适用性，可以有效的帮助企业制定产品召回方案，大大地降低产品召回成本。

关键词：产品召回，贪心算法，图论算法，数学建模

Abstract

Nowadays，people pay more and more attention to product quality. So，it is common that companies take efforts to get all products sold back when the products have serious issues，which is called ‘prodect reacll’. Product recall often makes the enterprise suffered heavy losses. So，we talk about product recall in this article，and we establish a mathematical model to calculate the best way for the enterprise to transport the product in the product reall.

We establist a mathematical model, simplify the mathematical model and convert practical problem into math problems in Chapter two. We analyse and compare different algorithms to solve the model，and then we prove that we can use Greedy Algorithm to solve the problem. In the following，we calculate it with programming code. So，we decide a system which can calculate the best way in product reall quickly.

At last，we make a conclusion and talk about the applicability of the system. We can help enterprise decede the transport way in product recall，which can reduce the cost of product recall greatly.

Key words: product reall，Greedy Algorithm，Graph Theoretic Algorithm，Mathematical Model

目 录

1 绪论

1.1 研究背景

在现今的社会中，人们越来越对购买的产品重视，这不仅体现在售前，也体现在售后。也正因为这样，企业也越来越在产品质量上进行把关，力求做到零失误。尽管如此，企业还是有可能犯错的，产品的质量还是有可能出问题的。一旦出现这种情况，就意味着企业需要对产出的产品进行维护，这对企业来说会造成一定的损失。本文针对企业对产品维护的最严重后果——产品召回的过程所产生的损失进行优化，力求为企业做到最低成本的产品召回方案，让企业在产品召回过程中将损失减到最低。

1.1.1 产品召回概念

产品召回是指企业由于生产完成的产品出现致命性的问题，需要将已经生产完成、并且经过零售商销售出去的产品批量的回收至企业。

产品召回与普通的三包（退货、换货、保修）售后是不一样的，这种三包售后只是针对个人而言的售后服务，在法律上并不是必须的，在这个过程中如果企业没有履行好三包售后服务，企业只会在业界上失去名声，甚至如果只有极少数用户受到影响，那么企业可能并不会因为这部分用户造成过大损失，可能只会失去这部分用户而已[1]。所以，一定程度上说，三包售后是企业与消费者之间的一条维系双方长久利益的枢纽，而每一条枢纽之间是独立的，不存在太大的联系。

而产品召回则是面对批量的产品，对于那些还没销售出去，但在零售商销售的产品

也需要被回收，也就是所有有问题产品都需要被追溯回来。所以很多企业为了能够在这种情况准确的召回所有有缺陷的产品，会给产品加上追溯系统，追溯系统在产品召回中也是值得展开的一个话题，但本文不打算就这进行过多的讲述。所以说，产品召回是一个全方位的回收，涉及的数目是十分大的。

另外，我国在 2004 年也建立了第一个关于产品召回的规章制度，是关于汽车的产品召回——《缺陷汽车产品召回管理规定》[2]。可见我国对产品召回是多么的重视，意味着产品召回并不单只是企业与消费者之间的关系，更是政府对产品安全性的重视，需要企业对消费者负责任的表现。

1.1.2 产品召回对企业损失

对于企业来说，产品召回会造成很大的损失。不论是从成本上还是在声誉上都是极大的损失。

从声誉上说，企业的一次产品召回一定是大规模的，定必会登上各大媒体报纸，让整个地区的人都知道这件事。因为有时候产品召回单靠追溯系统是不行的，还需要通过媒体传播到用户中，让用户清楚产品回收的流程操作以便配合产品召回。俗话说“好事不出门，坏事传千里”，对比于光荣事迹，人们常常更容易记得一个企业的过失之处，并且这种在人们的聊家常中是很容易成为话题的。所以当企业迫不得已进行产品召回的时候，这个品牌在消费者中就会形成一种不好的印象，容易让消费者认为这个品牌的其他产品都会存在同样的问题或者其它隐患。这对企业重新树立品牌效应造成极大的影响，这将极大的影响到企业的消费额。曾在 2009 年丰田汽车就尝试过在中国由于汽车安全问题，被迫实施大规模召回汽车，经过这个事件之后，丰田汽车一度

陷入信用危机，并且在之后几个月的销售额同比下降 15%以上[3]。由此可见，就是像丰田这样的世界 500 强企业在面对产品召回事件之后，也会瞬间陷入困境；一旦小企业面临产品召回，后果是不堪设想的。

从成本上说，企业需要将已经销售到各个地方的产品统一召回到生产地进行研究和改良，这在运输成本上也是一笔巨大的费用。因为产品召回一般是需要消费者将产品送到附近的专营店，再统一将所有产品从各个专营店运送到生产总部，这在运输上将会耗费巨大的人力成本和金钱成本。

1.2 研究内容

本文将通过利用计算机算法、程序实现的方式来研究产品召回的最优成本方案，本文希望通过研究，得出一个具有通用性的，能够对大部分产品召回问题进行最优成本方案分析，通过输入对应产品召回系统的参数，可以输出该产品召回中最优成本方案的系统。

1.3 研究的意义

本文通过最优成本产品召回研究，能够在为企业进行产品召回的时候大大降低成本，节省运输成本，能够在承受产品召回这个巨大的损失的时候做到止损。

由于产品召回的数量一般都是很大的，企业在面对产品回收的时候一般处于急于应对的状况，而去一般企业也没有产品召回的经历，所以在回收的过程中可能由于运输顺序的未被合理安排而导致在损失的基础上雪上加霜。本文研究方案的意义则是为企业在产品召回的时候合理选择成本最有方案，有利于企业避免无谓的损失。

2 模型建立

本文将利用数学建模的方法研究出产品召回成本最优方案。数学建模一般将实际问题简化，将可变参数合理的简化，同时不失严谨性，将复杂的生活问题变成易于处理的数学问题，利用数学知识进行解决[4]。而简化的这一过程并不会使得实际问题失去可研究性和失去实际应用的参照性，简化得到的数学模型的解决依然能作为实际的问题的参照。

2.1 模型描述

产品召回的实际问题为：企业需要将所有有缺陷的产品都从每一个指定的地点运送到企业生产总部；这一指定地点在实际中可能是分销商的地点，也可能是汽车召回中的 4S 店，在实际问题中是合理的。

在产品召回的过程中，企业一般对分散在各个指定地点（分销商）的产品被回收到生产总部的速度要求不大，并不会需要急于将产品回收到总部，因为这样对回收的成本要求太高。所以一般产品召回的操作都是企业给所有消费者传达消息，让消费者自行将产品送到指定的地点，而一般来说这个流程都是在一定的时间内，而超过额定时间之后的产品都会进行个别处理，这些个别处理不在本研究的范围之内。所以企业会在这个时间过后，统一将分散在不同地点的产品运输到生产总部。

这个问题中，已知的是指定地点的个数，指定地点距离生产总部的距离，各个指定地点之间的距离，在每个指定地点中的产品个数。

2.2 简化模型

在研究产品召回的最优成本方案中，本文以电脑为例，那么指定的地点就是遍布在每个城市的电脑店。消费者在规定的时间将自己购买的电脑送到距离自己最近的电脑店，然后电脑公司在规定的时间之后按照运输成本最低的方案将电脑统一运输到总部。

毫无疑问地，将不同地点的产品都运输到总部的方法可以是分别从产品所在地运送到目的地。但一般来说，如果不同的地点的距离相距不是太远，这显然不是最优的方案，是可以先将电脑店 A 的电脑运送到电脑店 B 处，然后统一运送到目的地。但是这样就需要衡量了，需要确定电脑店之间的电脑怎么互相运送，这正是本文研究的方向。

不失一般性，我们假设运输成本与距离成正比；同时，运输成本还与选择的交通工具有关。在企业对产品召回的速度要求不高的时候，我们可以假设所有电脑店的电脑都是用货车运输到目的地。而对于货车来说，也有不同的载荷的区分，一般来说，货车越大，运输成本越高。而货车的大小又与产品的数量有关，这之间的关系基本可以成正相关，所以也可以理解为货车的运输费与产品的数量成正比。由此可见，产品召回的运输费和距离、产品数量有必然联系。

所以本模型假设运输成本与距离成正比，比例系数为 x；由于一般来说货车与货车之间的运输费之间的差距不会太大，所以我们可以将货车的选择看成运输成本中的一个额外运输费，这个额外运输费与产品数量成正比，比例系数为 y。所以我们可以将运输费量化成与距离、数量相关的二元一次函数。假如距离为 s，产品数量为 a，那么运输费 w = s*x + a*y。这个 x 和 y 需要根据实际情况来确定它的值，本文假定在研究的区

域内，不同地区的 x 和 y 总是保持一致的，因为在我国之内，每公里的运输费用总是相差不太大的。

通过分析之后，我们将模型简化成以上这个二元一次函数之后，我们就可以更加容易的分析和处理这个问题了。

上面的二元一次函数为两点之间的运费公式，而模型需要解决的是 n 个分销商将产品运输到生产总部的运费总和的最小值以及对应的方案。所以本模型的目的是根据任意两点之间的运输成本为 w = s*x + a*y，找出将所有产品运输到目的地，而运费最低的方案，并计算出最低运费。其中分销商之间产品可以相互运输，意味着可以通过将分销商 A 的产品运输到分销商 B，然后再统一将两个（甚至三个）分销商的产品运输到目的地的方式来降低运输成本。

3 模型解决

本模型的解决是基于计算机编程，利用算法对模型进行求解，即编写一套程序，使得输入产品召回的基本参数，令程序输出产品召回最优成本的完整方案以及对应的运输成本。本文求解模型的算法融合了贪心算法和图论算法，利用图论算法的思想将连通图的边和点转换成二维数组，再利用贪心算法对模型求解。

3.1 解决模型思路

本文通过将模型简化，已经将问题变为图上 n 个分销商将 a 个产品运输到某个指定的地方的模型，而方案选择主要依赖于 n 个点之间的相互运输，并最终都运输到目的地的选择。而 n 个点之间运输的选择有（n!）种选择，如果需要遍历这（n!）种选择，该系统的性能将会非常低，当 n 很大的时候，计算机甚至可能无法运算而导致宕机，所以本文通过计算机算法来对系统进行优化。

对于任意两点 i 和 j，决定是否采取将一方的产品运输到另一方，再统一运输到目的地的是该方案是否比分别运输到目的地更加节省成本，下文称“将 i 的产品运输到 j，然后统一运输到目的地”为“ij 合并运输”、“将 i 和 j 的产品分别运输到目的地”为“ij 分别运输”。

根据这个描述，我们可以列出如下式子，我们定义 a_i 为地点 i 的产品数量，s_i 为地点 i 与目的地之间的距离，s_{ij} 为地点 i 和 j 之间的距离，所以 $s_{ij} = s_{ji}$。那么对于两个地点而言，将 i 的产品运输到 j，然后统一运输到目的地的总运输成本为

$$s_{ij}x + a_i y + (a_i + a_j)y + s_j x \tag{3.1}$$

而将 i 和 j 的产品分别单独运送到目的地的成本总和为

$$s_i x + a_i \mathrm{y} + s_j x + a_j \mathrm{y} \tag{3.2}$$

所以，当

$$s_{ij} x + a_i \mathrm{y} + \left(a_i + a_j\right) y + s_j x < s_i x + a_i \mathrm{y} + s_j x + a_j \mathrm{y} \tag{3.3}$$

即

$$a_i \mathrm{y} < \left(s_i - s_{ij}\right) x \tag{3.4}$$

时，将 i 的产品运输到 j，再统一运送到目的地的方案是更加有节省成本的。节省的总成本为

$$r = \left(s_i - s_{ij}\right) x - a_i \mathrm{y} \tag{3.5}$$

由此，当$r > 0$时，可以选择两地点之间互相运输。

对于$n = 2$的情况，显然只需要一次判断即可，但在实际中，n 往往远大于 2，所以本文需要给出 n 为任何可能值的通用解法。对于$n > 2$的情况，当我们选择将一个点的产品运输到另一个点（我们称之为接受者）之后，接收者的产品数量会发生变化，那么整个局面就和之前不一样了，相当于是变成了另一个产品召回系统；如果改变系统的这一个步骤不是促使系统朝最优方案的必要步骤，那么无论变幻之后系统怎么计算，也无法保证计算出整个系统本来应有的最优方案了。

本文考虑到贪心算法的思想，对于有 n 个点的系统，通过 n 次循环，每次循环对系统中剩余的点两两计算“ij 合并运输”比“ij 分别运输”的节省成本，取出在剩余系统中节省成本最高的两点，将该两点按成本最优方式合并，使得系统中减少一个点；通过这样 n 次循环，每次循环减少一个点的方式，可以在有限且快速的时间内计算出整个系统的运输成本最优方案。假如某次循环中，没有两点“合并运输”比“分别运

输”的方案成本更低，那么说明目前的系统上所有的点都分别运输到目的地的成本是最优的。相关证明见第四章。

3.2 图论算法简介

3.2.1 图论算法介绍

在计算机算法中，图论算法扮演者一个很重要的角色。图论算法是基于图，包括有向图、无向图等。图是由若干个点和若干条边组成，点可以分布在任何地方，边则是连结点和点之间的线[5]；而边既可以是有方向的，也可以是无方向的。

一般来说，图论算法的图都是二维的图，而在绘制图的过程中，边不一定是直线或者线段，可以是曲线，因为边的长短是没有意义的，我们会给边赋予一个权值，来代表这条边的值的大小。所以我们也可以认为那些没有边的两点，其实是有一条权值为正无穷的边。其中这一点很重要，因为本文在表达连通图的时候，是将连通图转换成二维数组，这就要求二维数组的每一个元素都有值，所以本文对于没有边的两点定义为权值为无穷大的边，那么在算法实现中，这种无穷大的值可以定义为机器语言的最大值，这一定义有助于算法实现和利用编程进行求解。

3.2.2 图论算法应用

在我们的日常生活中，也不乏图论算法的例子，最为明显的则是地铁图了。现在有许多地铁 app、地铁小程序都有提供两个地铁站之间的最短搭乘距离，而计算出这些最短距离的方法一般都是运用图论算法中的最短路径问题算法。最短路径问题是图论算法中最典型的问题之一，这当中有非常出名的 Dijkstra、Floyd 等算法。对于 G=(V，

E)的连通图，Dijkstra 的算法复杂度为 O(V^2+E)，Floyd 的时间复杂度则是 O(V^3)，因为 Floyd 的数据是利用矩阵来记录[6]。关于图论算法，在计算机编程中，这是一门很深的学问，本文在这只做简单的介绍。

3.3 贪心算法简介

3.3.1 贪心算法介绍

贪心算法是计算机算法的基础算法之一，它没有涉及图这种高级数据结构，这是一种纯算法，并非那种需要同时对数据结构有所了解才能掌握的算法，所以它在我们日常生活中也经常用到。贪心算法顾名思义就是利用贪心的思想，就是每一次都选择最优的，得到最后的方案也是最优的[7]。

从贪心算法的含义中可以知道，贪心算法立足的是局部最优，通过每一次局部最优得到的方案则是最终的方案。每一次都取最优这是很好理解的，但是这样的方案究竟是不是最优的方案呢，这是需要证明的，所以使用贪心算法是有一定的适用场景的。

3.3.2 贪心算法与动态规划算法对比

那么与贪心算法的局部最优对应的则是动态规划算法的全局最优，动态规划算法的每一次抉择并不是当下最优的，但它的每一次抉择是为了让剩余的抉择的最优和本次抉择的加权选择为最优，所以动态规划和贪心算法在策略上有本质的区别[8]。正因为动态规划需要保证全局最优，动态规划往往是需要列出决策之间的递推式关系，利用递推的关系来解决问题，所以一旦无法列出递推式，就无法使用动态规划算法[9]，本文亦如是。

一般来说，动态规划算法的适用范围会更加广，因为贪心算法有一定的限定条件，但是一旦遇到的问题适合用贪心算法，利用贪心算法解决问题在性能上会更加优化，这也是本文是用贪心算法的原因之一。

3.4 算法复杂度与算法实现

对于上述算法，对系统 n 个点进行 n 次循环，每一次都对剩余的点两两计算，两两计算的算法复杂度为$O(n^2)$，那么总的算法复杂度则是$O(n^3)$。这个算法复杂度与图论算法中的最短路径算法的矩阵表示形式的算法复杂度是一样的，而设计图论的算法的复杂度利用矩阵表示的算法一般不低于这个复杂度，说明该算法复杂度是可以接受的。

考虑到算法涉及数组比较多，所以本文利用对数据实现较为方便的 python 语言对算法进行实现[10]。具体代码见附录 A。

4 算法可行性证明

对于本文提出的算法，在解决问题上不难理解，但是需要论述本文的算法能正确解出产品召回成本最优方案上，还需要一定的数学证明，本章将会对该算法可行性进行证明。

4.1 合并运输必然是该点的全部产品

这一描述说明的是，不存在“将某点的产品部分运输到另一点”比“将某点的全部产品运输到另一点”的成本更优的方案；即若选择 i 的产品运输到 j，必然是将 i 的全部产品运输到 j。

（1） “将 i 的产品部分运输到 j，然后将剩余产品单独运输到目的地”比“将 i 的产品全部运输到 j，然后一起运输到目的地”的成本更高。

方案一：将 i 的 m 个产品运输到 j，然后剩余$a_i - \mathrm{m}$个产品留在 i 处，然后分别将 i 和 j 的产品分别运输到目的地（其他点，本处以目的地为例，其他点可同理，方案二亦如是），那么该流程总成本为

$$w_1 = (s_{ij}x + my) + \left[s_j x + \left(a_j + m\right)y\right] + [s_i x + (a_i - m)y] \tag{4.1}$$

化简得

$$w_1 = (s_{ij}x + my) + \left(s_j x + a_j y\right) + (s_i x + a_i y) \tag{4.2}$$

方案二：将 i 的产品全部运输到 j，然后将 j 的产品全部运输到目的地（其他点），那么该流程总成本为

$$w_2 = (s_{ij}x + a_i y) + \left[s_j x + \left(a_j + a_i\right)y\right] \tag{4.3}$$

由于当我们选择将 i 点的产品运输到 j 点的条件为“合并运输”比“分别运输”成本低，即

$$w_2 < (s_j x + a_j y) + (s_i x + a_i y) \tag{4.4}$$

由式(4.2)可知

$$w_2 < (s_j x + a_j y) + (s_i x + a_i y) < w_1 \tag{4.5}$$

所以方案二的成本比方案一的成本更优。同理的，假如将 i 的产品部分运输到 j，然后将 i 的产品和 j 的产品分别运输到另一个点 k，那么必然“将 i 全部运输到 j 然后一起运输到 k”或“将 i 和 j 分别单独运输到 k”的成本更低。所以当将某点的产品合并运输，必然是该点的全部产品。

（2） “将 i 的产品部分运输到 j，然后剩余产品运输到 k，最后 j、k 分别运输到目的地”比“将 i 的产品全部运输到 j，然后 j、k 分别运输到目的地”成本更高。

方案一：将 i 的 m 个产品运输到 j，然后将剩余$a_i - \mathrm{m}$个产品运输到 k 处，然后将 j、k 的产品分别运输到目的地（其他点，本处以目的地为例，其他点可同理，方案二亦如是），那么该流程总成本为

$$w_{11} = (s_{ij} x + m y) + [s_{ik} x + (a_i - m) y] \tag{4.6}$$

$$w_{12} = [s_j x + (a_j + m) y] + [s_k x + (a_k + a_i - m) y] \tag{4.7}$$

$$w_1 = w_{11} + w_{12} \tag{4.8}$$

化简得

$$w_1 = (s_{ij} x + a_i y) + [s_j x + (a_i + a_j) y] + (s_k x + a_k y) + s_{ik} x \tag{4.9}$$

方案二：将 i 的产品全部运输到 j，然后 j、k 分别运输到目的地（其他点），那么该流程总成本为

$$w_2 = \left(s_{ij}x + a_i y\right) + \left[s_j x + \left(a_i + a_j\right)y\right] + \left(s_k x + a_k y\right) \tag{4.10}$$

显然

$$w_2 = w_1 - s_{ik}x < w_1 \tag{4.11}$$

所以方案二的成本比方案一的成本更优。同理的，假如将 i 的产品部分运输到 j，剩余产品运输到 k，然后将 j 和 k 分别运输到另一点 g，那么必然将 i 全部运输到 j，然后 j、k 分别运输到 g 的成本更低。由此可见，当将某点的产品合并运输，必然是该点的全部产品。

依此类推的，当将某点的产品分成不止两份的运输到不同的点，可以将 4.1.1 证明中的的 m 变成一系列的数，容易证明得到该方案的成本依然比将该点的产品统一运输到另一点的成本高。同样的，当将某点的产品分成两份，按照不同的路径运输，并最终都运输到目的地，可以将 4.1.2 的 k 变成一系列的点，也不难证明处该方案的成本依然不是最优的。

所以，一旦选择将某点的产品运输到另一点，必然是将该点的产品全部运输到对应点。这一证明很重要，能为算法的循环中每一次减少一个点提供数学证明支撑。

4.2 循环减少点的方式选择该循环中节省最多的点

在本文的算法中，利用贪心算法，循环 n 次，每一次选择“合并运输”中成本节省最多的方式来减少一个点，使系统变成对比上一次循环减少一个点的系统，这一操作破坏了该系统，而本节则证明该操作即使破坏原有系统，但是能保证该操作为最优方案的必要操作。

假设在一次循环中，计算出节省成本最多的方案为将 i 的产品合并到 j 处（定义将

i 运输到 j 为 i →j，下文同)，那么对于系统来说，被破坏的点为 i 变为没了，j 的产品变多了。那么对于可能影响的操作进行证明：

①i→j，对于本来可以和 j 进行合并运输的点不会产生影响。我们知道，对于 kj 两点是否进行“合并运输”，取决于

$$r = (s_k - s_{jk})x - a_k y > 0 \tag{4.12}$$

是否成立，当式(4.12)满足的时候，我们会考虑将 k 和 j“合并运输”；由式(4.12)可知，r 只与s_k，s_{jk}，x，y，a_k有关，所以对于本来可以和 j“合并运输”的，在 i→j 后，相关参数的值也没有变动的，所以这些点依然可以考虑与 j“合并运输”。

②i→j 之后，j 的数量增大，对于本来存在 j→k（k 为不同于 i、j 的点）的情况可能不再符合。假设最先选择 j→k，那么我们将无法选择 i→j；此时我们只能选择 i→end（end 指最终的目的地，下同)；所以我们来将此方案与本文中最优方案中的运输选择进行运费比较。即比较 j→k，i→end，k→end 和 i→j，j→end，k→end。前者总成本为

$$r_1 = a_j y + s_{jk} x + a_i y + s_i x + (a_j + a_k) y + s_k x \tag{4.13}$$

后者总成本为

$$r_2 = a_i y + s_{ij} x + (a_i + a_j) y + s_j x + a_k y + s_k x \tag{4.14}$$

将两成本相减得

$$r_1 - r_2 = (s_i x - a_i y - s_{ij} x) - (s_j x - a_j y - s_{jk} x) \tag{4.15}$$

即两成本之差为 i→j 的节省成本与 j→k 的节省成本之差。由题设可知，i→j 的节省成本大于 j→k 的节省成本。所以

$$r_1 - r_2 = (s_i x - a_i y - s_{ij} x) - (s_j x - a_j y - s_{jk} x) > 0 \tag{4.15}$$

所以此方案成本大于算法求得的最优方案成本。

由①和②可知，当每次循环中选取“合并运输”节省成本最多的方案，使得系统的各个点之间的关系改变，但是这个操作并不影响最优方案，并且要最优方案的必要条件。所以我们是可以利用贪心算法来对模型进行解决实现。贪心算法同时保证了算法的效率，能保证算法不会无止境的运行下去。

5 模型应用分析

5.1 模型适用场景

从模型的建立开始，本模型追求的是成本的最优，力求将系统中各个点按尽可能的成本运输到目的地，模型对各个点运输的顺序并不关心，只关心最终的结果而不考虑中间过程。所以在一定程度上，本模型是在牺牲效率而追求成本最优。

一般来说，在计算机算法中，我们对算法进行不断的优化，优化到最后是利用空间来换取时间，意味着我们追求效率的时候是需要牺牲空间的，例如我们常用哈希表来优化查找性能。本模型解决问题也是依照相同的规律，在模型追求金钱成本的同时，需要牺牲运输时间成本。所以本模型适用于只要求运输的成本而没有具体时效要求的场景上，抑或者在运输问题上有时间限制，但是给定的时间限制足够大，对各点乱序运输不产生影响的问题。而本文解决的产品召回问题就是最典型的不追求时效的问题，对于企业而言，产品召回后对产品进行一系列的维修、丢弃等操作，并没有额定完成时间，所以只要求尽可能的降低成本。

总的来说，本模型适用于像产品召回这样不要求时效的汇总问题，能有效的节省运输成本。对于追求时效的物流问题[11]，本模型可能不太适用；但本模型仍然可以在这类物流运输问题的某一环节中应用。例如在物流运输中，需要将地区 A 的物品运输到地区 B，但地区 A 中存在多个投递点，而运输中仍然需要将多个投递点的物品统一运输到地区 A 的总分拣点。在这一步骤中，只要保证时间是足够的，仍然可以使用本模型来优化这一步骤的运输成本。而一般来说，本模型提及到的“合并运输”并不十分降低效率，因为“合并运输”一般针对距离比较近的两点，所以一般来说完成整个系

统的最低成本方案运输的时间是满足物流运输中的这一步骤的效率要求的。

5.2 模型优劣分析

5.2.1 模型优点

本模型在运输问题上着手解决成本问题，能够有效的降低产品召回等问题的运输成本，能够利用可靠的安排运输顺序的方法免除资源的浪费，为企业避免不必要的损失。同时能够为物流运输等问题节省局部运输的成本。

另外，本模型的可操作性强，能够将理论的问题映射到实际问题中去。本模型易于理解，并且在算法上实现简单，只要有一定计算机基础的人，在理解模型后，不难将此模型进行实现。

5.2.2 模型缺点

模型只能解决满足特定条件的运输问题，但实际中，完全满足该特定条件的问题并不十分常见，所以模型的适用性比较低。即使模型可以适用于物流运输问题中的局部运输，可以降低局部运输成本，但对于目前单个运输成本越来越低的情况下，利用该模型可降低的成本并不多。模型只在运输中各个点分布比较密集，而且距离总目的地并不十分近的情况下可以降低大量运输成本，在各个点与目的地相距比较近的情况下可降低的运输总成本并不显著。

5.3 案例分析

本文在这里列出一个产品召回的问题，并通过本文设计的系统计算出结果。假设在

系统中有 4 个分销商，现需要将 4 个分销商的产品都召回到企业总部。为了简化问题，此处提及费用、距离等均省略实际单位。4 个分销商的产品数量分别为 20、70、90、100；4 个分销商与企业总部的距离分别为 90、80、70、50；运输成本与距离的比例系数 x 设为 7，运输成本与常品数量的比例系数 y 设为 1；4 个分销商两两之间的距离为

$$s = \begin{matrix} 0 & 11 & 23 & 100 \\ 11 & 0 & 11 & 100 \\ 23 & 11 & 0 & 100 \\ 100 & 100 & 100 & 0 \end{matrix}$$

即分销商 i 与分销商 j 的距离为 s[i][j]，分销商 1 与分销商 2 的距离为 s[1][2]。

根据上述的已知条件，我们可以将这些参数输入本文设计出的系统中（示例见附录 B），最后可以得出最低成本产品召回方案为“将 0 的产品全部运送到 1 分销商处；将 1 的产品全部运送到 2 分销商处；余下的分销商的商品分别独自运到生产总部。总费用为：1384”（这里分销商的起始坐标为 0）。

我们可以计算得将 4 个分销商分别运送到目的地的总费用为 2310，由此可见本文设计的最低成本产品召回系统输出的方案在这种情况下可以比分别运输到目的地的传统方案节省近一半的成本。

本案例在实际中也是很常见的，分销商 0、1、2 是距离比较接近的，而分销商 3 是分散的，所以最优方案是将分销商 0、1、2 按照一定顺序合并，而分销商 3 则单独运输到目的地。

总 结

本文研究了产品召回问题中，通过合理安排系统中各个点的运输顺序，并且通过两两之间互相运输来降低总成本的最优成本方案。本文在研究中利用到了计算机算法中的图论算法来将图论问题按照矩阵的表示形式来转换成数组的元素处理问题，并且利用到贪心算法来控制算法运行的效率，使算法在可接受的时间内完成，并且保证算法不会陷入死循环中。

通过这次研究，我对我本来应该没什么机会接触的产品召回有了一定的研究，也对我国关于产品召回的制度有了认识，让我这个工科生能够了解到工科以外的知识。同时，这次研究对于我的检索能力有了很大的提升，因为我需要不断的去检索前人关于这类研究的研究成果，让我去借鉴参考，让我知道我的研究应该怎么下手。

我作为一名机械的学生，一直对计算机编程、计算机算法感兴趣，这一次研究让我能够真正的进入到计算机编程的学习中，通过问题提出，我需要自学数学建模，通过数学建模来对问题进行简化建模，并且需要不断的变更模型来有效的解决问题。可以说我在解决这个问题上，也反复建了不下十次模型，并且对模型进行批判，最终才能建成论文中提及的模型。然后我就需要根据模型选择合适、高效的算法来对模型进行求解。在这过程中，我研究了《算法导论》这一本入门计算机算法的书，让我对计算机算法有了很深刻的理解，这也对我的逻辑思维有一个很大的锻炼。紧接着，我需要对我想到的算法进行数学证明，这当中是需要大量的数学运算的，即使看起来觉得很显然的东西，也是需要通过一笔一画来列公式进行证明的，不然很可能只是我自己的一厢情愿而已。所以我对自己的数学知识进行了一段时间的补充，这让我的数学计算

能力有了一定的提升。

同时，在这次研究中，我需要利用计算机编程语言来实现算法，来建立一套系统，所以我自学了 python 语言，了解到了 python 语言的各种好处，能够快速的实现我自己的算法，最终我能够编写出一套能够解决产品召回最低成本方案的系统，只要按照程序的指引，输入产品召回中各个点的参数，系统可以自动生成最低成本的运输方案。

在我这次研究中，我的研究仍然存在着一些不足，例如我编写的程序只是实现产品召回最低成本运输方案的一个算法实现程序，而还没有实现系统的界面表示，并且程序运行必须在有安装 python 环境的机器上运行，仍未做成在任何环境下都能运行的应用程序或者小程序，在这方面可以进行改善。

总的来说，我在这一次研究中，能够学到了很多计算机方面的知识，能够对我日后的发展道路有很大的帮助，也让我发掘到了自己的一些数学天赋，让我提高了自己的逻辑思维能力，这在我未来工作道路上会有不少的作用。

参 考 文 献

[1] 倪娜. 缺陷产品召回制度法律问题研究[D].复旦大学,2008.

[2] 谢志利. 我国缺陷产品召回制度与产品责任制度以及三包制度辨析[J]. 产品安全与召回,2012,(02):52-56.

[3] 本报记者 崔进贤. 丰田汽车召回事件影响初显[N]. 中国工业报,2010-02-26B01.

[4] 寇海燕. 例谈 “数学建模” 在解决实际问题中的应用[J]. 学周刊,2014,(11):212.

[5] 王丽. 图论在算法设计中的应用[D].西安电子科技大学,2010.

[6] 张建军,杜莉. 最短路径算法的分析与优化[J]. 北京工业职业技术学院学报,2009,03:26-31.

[7] Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein,殷建平,徐云,王刚,刘晓光,苏明,邹恒明,王宏志. 算法导论(原书第 3 版)[J]. 计算机教育,2013,10:51.

[8] 董军军. 动态规划算法和贪心算法的比较与分析[J]. 软件导刊,2008,(02):129-130.

[9] Zhong-Liang Zhang,Jie Chen. Implementable Strategy Research of Brake Energy Recovery Based on Dynamic Programming Algorithm for a Parallel Hydraulic Hybrid Bus[J]. International Journal of Automation & Computing,2014,03:249-255.

[10] 透明. 关于 Python[J]. 程序员,2002,(03):68-69.

[11] 张志哲,杨俊琴,杨东援. 时效物流运输网路规划研究[J]. 公路交通科技,2005,(12):152-156.

致 谢

在这篇论文顺利完成之际，我要对在我进行论文编写中，对我模型建立、算法设计、算法证明等上面给予我帮助的李志老师表示我衷心的感谢。在我的论文编写中，李志老师给我很多指导的意见，教会我如何快速的去检索资料，而免于我走很多冤枉路。在给我指导的过程中，很多时候李老师都是知道我遇到的问题的答案是什么，但是李老师并没有给我立刻指出答案，而是教我如何去找寻答案，给我方向，让我自己去寻找答案，所以我在寻找答案的过程中能够学习到很多知识，并且印象更加的深刻。

同时，我还要感谢在我论文中常常给予论文编写帮助的陈光锋师兄，他以自己的经验给我一些论文写作技巧，让我将论文看起来不那么凌乱，让我的论文更加像一篇毕业论文，他和李老师一起给我的论文一遍又一遍的指出问题，对我的论文严格把关，让我能够完成这一篇毕业论文。同时，我还要感谢在我学习期间给我关心和鼓励的老师和同学，因为有你们，我才能完成这一份论文，谢谢你们！

毕业论文并不代表学习的结束，这正是我新的学习生活的开始，我将一直保持对知识的饥渴，保持不断的学习，让我能够在工作中有一番作为。

感谢各位老师的批评指导。

附 录 A

```
####### 最低成本产品召回系统 python 代码 #######

#!/usr/bin/python

# -*- coding: utf-8 -*-

import sys


print('请输入分销商的数量：')

line = sys.stdin.readline().strip()

n = int(line)


print('请依次输入每个分销商中的产品数量：')

line = sys.stdin.readline().strip()

astr = line.split()

a = [0 for i in range(n)]

for i in range(n):

    a[i] = int(astr[i])


print('请依次输入每个分销商与生产总数的距离：')

line = sys.stdin.readline().strip()

sToDesStr = line.split()
```

```
# 每个分销商到目的地（生产总部）的距离
sToDes = [0 for i in range(n)]
for i in range(n):
    sToDes[i] = int(sToDesStr[i])

print('请输入运输成本与距离的比例系数 x：')
line = sys.stdin.readline().strip()
x = int(line)

print('请输入货车大小造成的额外运输费与产品数量的比例系数 y：')
line = sys.stdin.readline().strip()
y = int(line)

print('请输入%s 个分销商之间的距离，按%s*%s 的矩阵形式输入：' %(n, n, n))
# 分销商与分销商之间的距离
s = [[0 for i in range(n)] for j in range(n)]
count = 0
while True:
    line = sys.stdin.readline().strip()
    if line == '':
        break
```

```
    lines = line.split()
    for i in range(n):
        s[count][i] = int(lines[i])
    count += 1

# 分销商是否已经确定方案
judge = [0 for i in range(n)]

# 方案输出
res = ''

# 总费用
cost = 0

for k in range(n):
    f = 0
    to = -1
    tmp = 0
    for i in range(n):
        if judge[f] == 1:
            f += 1
```

```
            if judge[i] == 1:
                continue
            for j in range(n):
                if j == i or judge[j] == 1:
                    continue
                mid = sToDes[i]*x + a[i]*y + sToDes[j]*x + a[j]*y - (s[i][j]*x + a[i]*y +
(sToDes[j]*x + (a[i] + a[j])*y))
                if mid > tmp:
                    tmp = mid
                    f = i
                    to = j
        if to == -1:
            if res == '':
                res += '每个分销商的商品分别独自运到生产总部。'
            else:
                res += '余下的分销商的商品分别独自运到生产总部。'
            for m in range(n):
                if judge[m] == 0:
                    cost += sToDes[m]*x + a[m]*y
            break
        else:
```

```
            judge[f] = 1
            a[to] += a[f]
            res += '将 %s 的产品全部运送到 %s 分销商处；' %(f, to)
            cost += s[f][to]*x + a[f]*y

print('最优方案如下：')
print(res)
print('总费用为：%s' %(cost))
```

附 录 B

系统运行范例：

输入：

请输入分销商的数量：

4

请依次输入每个分销商中的产品数量：

20 70 90 100

请依次输入每个分销商与生产总数的距离：

90 80 70 50

请输入运输成本与距离的比例系数 x：

7

请输入货车大小造成的额外运输费与产品数量的比例系数 y：

1

请输入 4 个分销商之间的距离，按 4*4 的矩阵形式输入：

0 11 23 100

11 0 11 100

23 11 0 100

100 100 100 0

输出：

最优方案如下：

将 0 的产品全部运送到 1 分销商处；将 1 的产品全部运送到 2 分销商处；余下的分销商的商品分别独自运到生产总部。

总费用为：1384