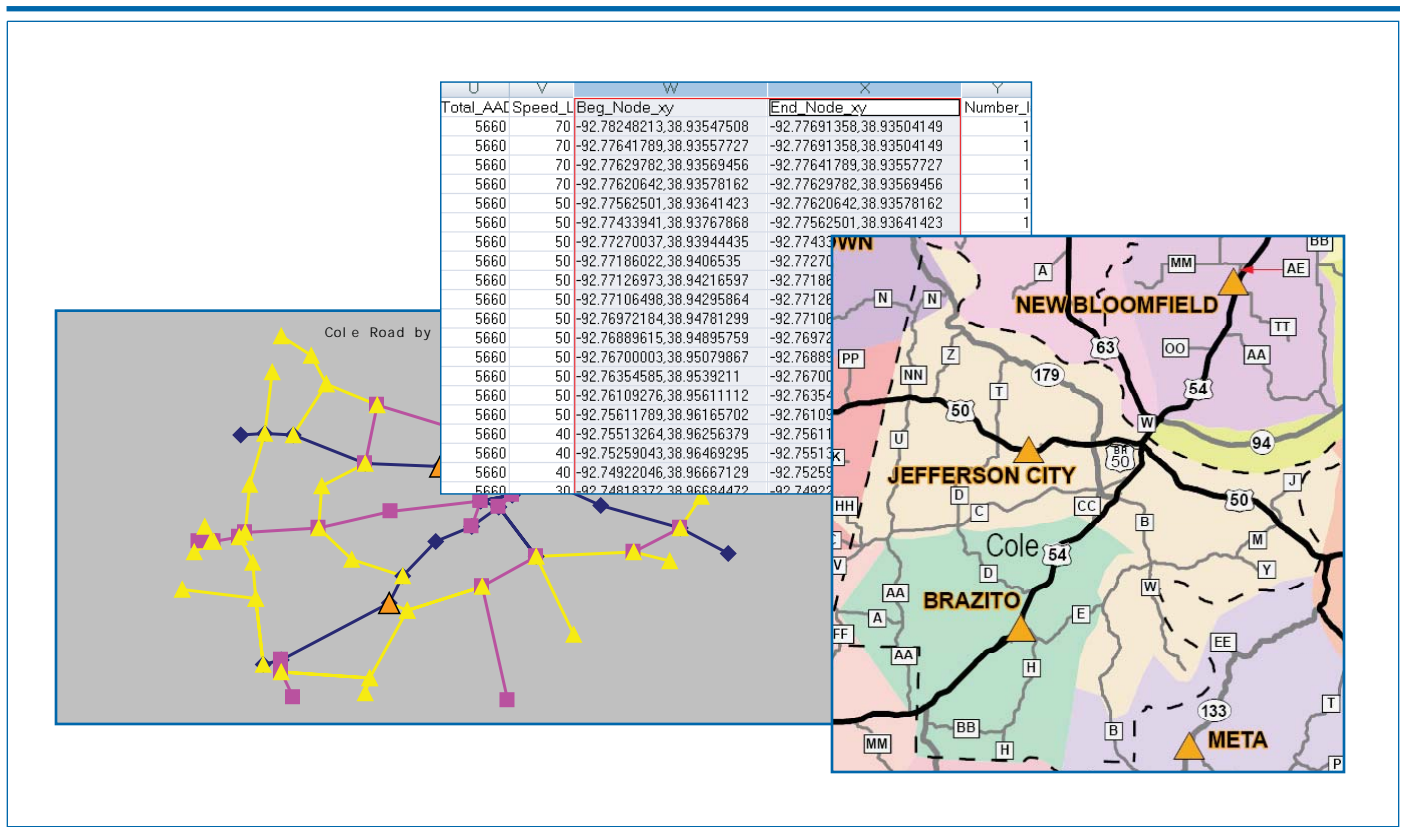


A Decision Support System for Optimal Depot and Fleet Management



**Final Report
August 2008**



IOWA STATE UNIVERSITY
Institute for Transportation

Sponsored by
University Transportation Centers Program,
U.S. Department of Transportation
(MTC Project 2007-04)

About the MTC

The mission of the University Transportation Centers (UTC) program is to advance U.S. technology and expertise in the many disciplines comprising transportation through the mechanisms of education, research, and technology transfer at university-based centers of excellence. The Midwest Transportation Consortium (MTC) is a Tier 1 University Transportation Center that includes Iowa State University, the University of Iowa, and the University of Northern Iowa. Iowa State University, through its Institute for Transportation (InTrans), is the MTC's lead institution.

Disclaimer Notice

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. The opinions, findings and conclusions expressed in this publication are those of the authors and not necessarily those of the sponsors.

The sponsors assume no liability for the contents or use of the information contained in this document. This report does not constitute a standard, specification, or regulation.

The sponsors do not endorse products or manufacturers. Trademarks or manufacturers' names appear in this report only because they are considered essential to the objective of the document.

Non-discrimination Statement

Iowa State University does not discriminate on the basis of race, color, age, religion, national origin, sexual orientation, gender identity, sex, marital status, disability, or status as a U.S. veteran. Inquiries can be directed to the Director of Equal Opportunity and Diversity, (515) 294-7612.

Technical Report Documentation Page

| | | | | | |
|--|--|--|--|--|------------------------|
| 1. Report No. MTC Project 2007-04 | | 2. Government Accession No. | | 3. Recipient's Catalog No. | |
| 4. Title and Subtitle A Decision Support System for Optimal Depot and Fleet Management | | | | 5. Report Date August 2008 | |
| | | | | 6. Performing Organization Code | |
| 7. Author(s) Wooseung Jang, James Noble, Cerry Klein, Charles Nemmers, Zhongwei Yu, and Ya Zhang | | | | 8. Performing Organization Report No. | |
| 9. Performing Organization Name and Address Midwest Transportation Consortium University of Missouri – Columbia Iowa State University College of Engineering 2711 South Loop Drive, Suite 4700 Columbia, MO 65211 Ames, IA 50010-8664 | | | | 10. Work Unit No. (TRAIS) | |
| | | | | 11. Contract or Grant No. | |
| 12. Sponsoring Organization Name and Address Midwest Transportation Consortium Missouri Department of Transportation Iowa State University 105 W. Capitol Avenue 2711 South Loop Drive, Suite 4700 Jefferson City, MO 65102 Ames, IA 50010-8664 | | | | 13. Type of Report and Period Covered Final Report | |
| | | | | 14. Sponsoring Agency Code | |
| 15. Supplementary Notes Visit www.intrans.iastate.edu/mtc for color PDF files of this and other research reports. | | | | | |
| 16. Abstract The objective of this project is to develop an optimization-based decision support system to address depot and fleet management issues associated with typical DOT maintenance system operations. In particular, this research proposes a systematic, heuristic-based optimization approach to integrate winter road maintenance planning decisions, which are typically the most intensive operation in the DOT maintenance system. Winter road maintenance operations require many complex strategic and operational planning decisions. The main problems include locating depots, designing sectors and routes, and configuring and scheduling vehicles. The complexity involved in each of these decisions has resulted mainly in research that approaches each of the problems separately, which can lead to isolated and suboptimal solutions. In addition to being integrated, a successful approach to these problems must consider the necessary practical aspects of each problem and include a straightforward, easily-executable solution methodology; otherwise the approach is unlikely to be implemented. | | | | | |
| 17. Key Words greedy-type heuristic—one arc-position exchange—one arc-position movement—winter road maintenance | | | | 18. Distribution Statement No restrictions. | |
| 19. Security Classification (of this report) Unclassified. | | 20. Security Classification (of this page) Unclassified. | | 21. No. of Pages 122 | 22. Price NA |

A DECISION SUPPORT SYSTEM FOR OPTIMAL DEPOT AND FLEET MANAGEMENT

**Final Report
August 2008**

Principal Investigator

Wooseung Jang
Associate Professor of Industrial and Manufacturing Systems Engineering
University of Missouri – Columbia

Co-Principal Investigators

James Noble
Associate Professor of Industrial Engineering
University of Missouri – Columbia

Cerry Klein
Chair and Lapierre Professor of Industrial and Manufacturing Systems Engineering
University of Missouri – Columbia

Charles Nemmers
Program Director of Transportation Infrastructure Center and Research Instructor
University of Missouri – Columbia

Research Assistants

Zhongwei Yu and Ya Zhang

Preparation of this report was financed in part
through funds provided by the U.S. Department of Transportation
through the Midwest Transportation Consortium (MTC Project 2007-04).
and the Missouri Department of Transportation

A report from
The Midwest Transportation Consortium

Iowa State University
2711 South Loop Drive, Suite 4700
Ames, IA 50010-8664
Phone: 515-294-8103
Fax: 515-294-0467
www.intrans.iastate.edu/mtc

TABLE OF CONTENTS

| | |
|--|------|
| ACKNOWLEDGMENTS | XI |
| EXECUTIVE SUMMARY | XIII |
| 1. INTRODUCTION | 1 |
| 2. PROBLEM FORMULATION AND SOLUTION METHODOLOGY | 3 |
| 2.1 Network Initialization | 3 |
| 2.2 Depot and Sector Selection | 3 |
| 2.3 Initial Route Construction | 5 |
| 2.4 Solution Improvement | 5 |
| 2.5 Fleet Configuration and Scheduling | 7 |
| 3. ALGORITHM IMPLEMENTATION | 9 |
| 3.1 Service Hierarchy | 9 |
| 3.2 Node Map | 10 |
| 3.3 Simplified Road Map | 11 |
| 3.4 Road Class Modification | 12 |
| 3.5 Data Filtering | 13 |
| 3.6 Program Running | 14 |
| 3.7 Route Result Modification and Route Map Generation | 14 |
| 3.8 Truck Assignment | 16 |
| 4. ANALYSIS AND RESULTS | 17 |
| 4.1 Cole County | 17 |
| 4.2 Callaway County | 23 |
| 4.3 Osage, Gasconade, and Maries Counties | 30 |
| 4.4 Cooper and Moniteau Counties | 39 |
| 4.5 Benton and Pettis Counties | 46 |
| 4.6 Morgan, Miller and Camden Counties | 55 |
| 4.7 Boone County | 64 |
| 4.8 Summary | 70 |
| 5. CONCLUSIONS | 72 |
| REFERENCES | 73 |
| APPENDIX A. COMPUTER PROGRAMS | A-1 |

LIST OF FIGURES

| | |
|---|----|
| Figure 3.1. Classification of roadways | 9 |
| Figure 3.2. Longitudes and altitudes of nodes | 10 |
| Figure 3.3. A sample node map | 11 |
| Figure 3.4. A simplified road map | 11 |
| Figure 3.5. Class A2 roads | 12 |
| Figure 3.6. Class A3 roads | 12 |
| Figure 3.7. Combined A2 and A3 roads | 13 |
| Figure 3.8. Road information data (EXCEL file) | 13 |
| Figure 3.9. Road information data (TXT file) | 14 |
| Figure 3.10. Route result details | 15 |
| Figure 3.11. Routes before modification | 15 |
| Figure 3.12. Routes after modification | 16 |
| Figure 4.1.1. Cole original map | 17 |
| Figure 4.1.2. Simplified map | 18 |
| Figure 4.1.3. Class A1 roads | 18 |
| Figure 4.1.4. Class A1 routes | 19 |
| Figure 4.1.5. Class A2 roads | 19 |
| Figure 4.1.6. Class A3 roads | 20 |
| Figure 4.1.7. Classes A2 and A3 roads | 20 |
| Figure 4.1.8. Classes A2 and A3 routes | 21 |
| Figure 4.1.9. Final routes | 21 |
| Figure 4.2.1. Callaway original map | 23 |
| Figure 4.2.2. Simplified map | 24 |
| Figure 4.2.3. Class A1 roads | 24 |
| Figure 4.2.4. Class A1 roads | 25 |
| Figure 4.2.5. Class A2 roads | 25 |
| Figure 4.2.6. Class A2 routes | 26 |
| Figure 4.2.7. Class A3 roads | 26 |
| Figure 4.2.8. Class A3 routes | 27 |
| Figure 4.2.9. Final routes | 27 |
| Figure 4.3.1. Osage, Gasconade, and Maries original map | 30 |
| Figure 4.3.2. Osage, Gasconade, and Maries road map | 31 |
| Figure 4.3.3. Class A1 roads | 31 |
| Figure 4.3.4. Class A1 routes | 32 |
| Figure 4.3.5. Class A2 roads | 32 |
| Figure 4.3.6. Class A2 route | 33 |
| Figure 4.3.7. Class A3 roads | 33 |
| Figure 4.3.8. Class A3 routes | 34 |
| Figure 4.3.9. Final routes | 35 |
| Figure 4.4.1. Cooper and Moniteau original map | 39 |
| Figure 4.4.2. Simplified map | 40 |
| Figure 4.4.3. Class A1 roads | 40 |
| Figure 4.4.4. Class A2 roads | 41 |
| Figure 4.4.5. Class A1 and A2 roads | 41 |
| Figure 4.4.6. Class A1 and A2 routes | 42 |

| | |
|--|----|
| Figure. 4.4.7 Class A3 roads..... | 42 |
| Figure 4.4.8. Class A3 routes..... | 43 |
| Figure 4.4.9. Service regions by depots..... | 43 |
| Figure 4.4.10. Final routes..... | 44 |
| Figure 4.5.1. Benton and Pettis original map..... | 46 |
| Figure 4.5.2. Benton and Pettis road map..... | 47 |
| Figure 4.5.3. Class A1 roads..... | 48 |
| Figure 4.5.4. Class A1 routes..... | 48 |
| Figure 4.5.5. Class A2 roads..... | 49 |
| Figure 4.5.6. Class A2 routes..... | 49 |
| Figure 4.5.7. Class A3 roads..... | 50 |
| Figure 4.5.8. Class A3 routes..... | 50 |
| Figure 4.5.9. Service regions by depots..... | 51 |
| Figure 4.5.10. Final routes..... | 51 |
| Figure 4.6.1. Morgan, Miller, and Camden original map..... | 55 |
| Figure 4.6.2. Simplified map..... | 56 |
| Figure 4.6.3. Class A1 roads..... | 57 |
| Figure 4.6.4. Class A1 routes..... | 57 |
| Figure 4.6.5. Class A2 roads..... | 58 |
| Figure 4.6.6. Class A2 routes..... | 58 |
| Figure 4.6.7. Class A3 roads..... | 59 |
| Figure 4.6.8. Class A3 routes..... | 59 |
| Figure 4.6.9. Service regions by depots..... | 60 |
| Figure 4.6.10. Final routes..... | 60 |
| Figure 4.7.1. Boone original map..... | 64 |
| Figure 4.7.2. Class A1 routes..... | 65 |
| Figure 4.7.3. Class A1' routes..... | 65 |
| Figure 4.7.4. Class A2 routes..... | 66 |
| Figure 4.7.5. Class A3 routes..... | 66 |
| Figure 4.7.6. Service regions by depots..... | 67 |
| Figure 4.7.7. Final routes..... | 67 |

LIST OF TABLES

| | |
|---|------|
| Table 1. Number of trucks | xiii |
| Table 2. Service hierarchy parameters..... | 9 |
| Table 3. A sample summary of service and deadheading time and distance..... | 16 |
| Table 4. A sample operation plan | 16 |
| Table 5. Route description | 22 |
| Table 6. Summary of routes..... | 22 |
| Table 7. Route operation plan..... | 22 |
| Table 8. Route description | 28 |
| Table 9. Summary of routes..... | 29 |
| Table 10. Route operation plan..... | 29 |
| Table 11. Route description | 35 |
| Table 12. Summary of routes..... | 37 |
| Table 13. Route operation plan..... | 38 |
| Table 14. Route description | 44 |
| Table 15. Summary of routes..... | 45 |
| Table 16. Route operation plan..... | 45 |
| Table 17. Route description | 52 |
| Table 18. Summary of routes..... | 53 |
| Table 19. Route operation plan..... | 54 |
| Table 20. Route description | 61 |
| Table 21. Summary of routes..... | 62 |
| Table 22. Route operation plan..... | 63 |
| Table 23. Route description | 68 |
| Table 24. Summary of routes..... | 69 |
| Table 25. Route operation plan..... | 70 |
| Table 26. Summary of results (*estimated number)..... | 71 |

ACKNOWLEDGMENTS

The research team would like to thank the Missouri Department of Transportation for their assistance in data collection and providing feedback throughout the project. We would especially like to recognize Tim Chojnacki and Jay Whaley's contributions to this project in their roles as technical advisors.

EXECUTIVE SUMMARY

The objective of this project is to develop an optimization-based decision support system to address depot and fleet management issues associated with typical DOT maintenance system operations. In particular, this research proposes a systematic, heuristic-based optimization approach to integrate winter road maintenance planning decisions, which are typically the most intensive operation in the DOT maintenance system. Winter road maintenance operations require many complex strategic and operational planning decisions. The main problems include locating depots, designing sectors and routes, and configuring and scheduling vehicles. The complexity involved in each of these decisions has resulted mainly in research that approaches each of the problems separately, which can lead to isolated and suboptimal solutions. In addition to being integrated, a successful approach to these problems must consider the necessary practical aspects of each problem and include a straightforward, easily-executable solution methodology; otherwise the approach is unlikely to be implemented.

The application of the integrated solution methodology developed in this research to the transportation network of District 5 in Missouri resulted in a very promising solution. As seen in the table below, the number of recommended trucks decreased from 169 to 136. This shows that our approach can maintain the same level of service with 19.5% less trucks.

Table 1. Number of trucks

| County | Service mileage | Number of trucks | |
|--------------------------|-----------------|------------------|-------------|
| | | Current | Recommended |
| Cole | 1551 | 17 | 11 |
| Callaway | 2588 | 18 | 17 |
| Osage, Gasconade, Maries | 2281 | 30 | 21 |
| Cooper, Moniteau | 2445 | 24 | 18 |
| Benton, Pettis | 3140 | 26 | 24 |
| Morgan, Miller, Camden | 3497 | 31 | 27 |
| Boone | 3988 | 23 | 18 |
| Total | 19490 | 169 | 136 |

The results from a real-world test problem support the assertion of the importance of a more integrated approach to the winter road maintenance problems. Our approach allows us to provide the same high standard of service with much fewer resources. The solution methodology is designed to aid winter road maintenance planners in making decisions regarding the interrelated problems studied in this research. These solutions are based on the effect that these decisions have on the agency's ability to achieve a desired level of service. The ability to solve the winter road maintenance planning problems in a more integrated manner should provide planners with the ability to make more informed, successful decisions. While this research and its application focuses on the winter maintenance operations, its usage can be applied to any depot and fleet management systems. The potential cost savings to the DOTs would accrue in many areas.

1. INTRODUCTION

Winter road maintenance operations require many complex planning decisions; the main strategic and operational problems include defining a service level policy, locating depots, designing sectors, routing service vehicles, configuring the vehicle fleet, and scheduling vehicles. All the activities are interrelated, in that each decision affects some or all of the other decisions and all together the decisions impact the agency's ability to provide the desired level of service.

Public agencies have been performing winter road maintenance planning and operational activities since as early as 1881 (Campbell and Langevin 2000). However, surveys by Gupta (1998) and Campbell and Langevin (2000) showed that most agencies do not utilize any formal methods for locating depots or designing routes. Both works suggest that most agencies still rely in large part on assessments dictated by field experiences. Winter road maintenance planning decisions include many complex characteristics, and it is very difficult to make these decisions solely on the basis of experience. Therefore, an opportunity exists for improving the planning process through the implementation of an optimization-based methodology.

This research attempts to solve winter road maintenance problems in an integrated manner. The need for such research has been recognized for some time, but it has not been undertaken because of the complexity of each of the problems involved. Reinert, Miller, and Dickerson (1985) point out that the four problems of depot location, sector design, vehicle routing, and vehicle scheduling would "ideally be solved simultaneously," but they dismissed the idea saying that it is "not practically accomplished." Traditionally, research approaches have followed this same belief, addressing the main winter road maintenance problems separately or, in a few cases, sequentially.

The problem of locating depots entails determining the number of depots to open and where to open them. This problem is often solved by choosing a preset number of depots to open from a set of candidate sites based on some predefined strategic and operational objectives. Designing sectors involves the assignment of arcs requiring service to the depots responsible for servicing them. The sector design problem is often solved in conjunction with the depot location problem, both in practice and in existing research methods.

The routing problem usually involves the determination of a set of routes, which are serviced by vehicles starting and ending at their respective depots, to optimize some performance criteria. Routes are typically constrained either by maximum duration or distance. If it is possible for a vehicle to service multiple routes prior to servicing any of the routes again, then the vehicle scheduling problem must be solved as well. Determining the fleet configuration depends on whether the fleet is assumed to be homogeneous. If the fleet is assumed to be homogeneous, the problem is reduced to a fleet sizing problem. Otherwise, the problem is more complex and consists of determining the numbers of each of the types of vehicles based at each depot.

Our approach solves the integration of the above-mentioned problems by iterating the individual problems with different parameters. The initial phase solves the problems of depot location, sector design, and initial route construction. Then, in the solution improvement phase, we attempt to improve the solution to the route and sector design problems based on the determined

depot locations, and we also determine the solutions to the vehicle scheduling and fleet configuration problems.

Our analysis is applied to the state highway road network in District 5, Missouri, which is serviced by the Missouri Department of Transportation (MoDOT). Note that we utilize the knowledge of experienced planners in MoDOT to determine the objectives, constraints, and other factors. This increases the chances that the solutions will be accepted and implemented. Our analysis provides solutions under different scenarios to aid planners in the decision-making process, rather than dictating a final solution. The detailed results are described in Section 4. In addition, mathematical models and algorithms are given in Sections 2 and 3. The computer programs developed to implement our results are given in Appendix A.

2. PROBLEM FORMULATION AND SOLUTION METHODOLOGY

The goal of our approach to the winter maintenance problem is to provide frequent service satisfying the target cycle times with a smaller number of trucks, if possible. Hence, the objective function in our modeling is to minimize the number of working trucks with the cycle time and truck capacity constraints. As a subsequent objective, we minimize the largest cycle time of each class, when the same number of trucks operates. We achieve these objectives through the nested iterations of depot and sector selection, initial route construction, route improvement, and fleet configuration and scheduling procedures until a better solution can be found. Each procedure is described in the following sections, and the implementation of the complete algorithm is given in the next chapter.

In this section, the route is defined as a trip from a beginning depot that then returns to the same depot with a refill of the truck at the end. The route time includes both service (i.e., spreading and plowing) and deadheading times, but not the time required to refill the truck. The cycle time of a route is defined as the time that elapses until the truck returns to the beginning of the same route after completing one cycle of assigned work. If a truck exclusively serves one route, which happens generally for A1 routes in our application, the cycle time would be the route time plus refill time. However, if a truck serves multiple routes, the service sequence and frequency affect the cycle time.

2.1 Network Initialization

The first step is to create one tour throughout the entire transportation network. This achieves two purposes: (1) verification of a strongly connected network, and (2) calculation of the shortest paths between every node and their corresponding distances. While there are many algorithms for solving this Chinese Postman Tour (CPT), we use the algorithm for the closed CPT presented by Thimbleby (2003). The algorithm determines an Eulerian tour, if possible; otherwise a tour with some required deadheading is determined. The shortest path from each node to every other node is also determined using the Floyd-Warshall algorithm, and the corresponding distances are stored in a matrix where they can be accessed easily for use in the algorithm (Skiena 1998).

2.2 Depot and Sector Selection

The depot location problem involves the selection of a predetermined number of depot sites to be opened out of a larger set of candidate depot sites. This is not an issue if existing or proposed depots are used. It is clear that the service time of any route is smaller, or equivalently the service frequency is higher, if the route is located near a depot because a service truck should start and end at a depot. Hence, a depot should be located at a place where many roadways are quickly accessible.

An integer programming model such as (1) can be used for this purpose. However, if depots and sectors are determined repeatedly as sub-routines of an integrated problem, the usage of such a model for a large problem becomes inefficient. In this research, we use the integer programming model (1) for the initial selection of depots and sectors, and employ a greedy-type heuristic to locate depots during iterative procedures.

Suppose that there are n potential depot locations and m arcs to service. In addition, assume that d_{ij} represents the distance between node i and arc j and that f_j represents the weight for arc j . Because some arcs, based on their class should be served more often, we use weights to appropriately represent their service frequencies. The weights used in our application correspond to the target service frequencies of six, two, and one during a 12-hour continuous storm for routes of classes A1, A2, and A3, respectively. The depot selection problem uses an objective function that minimizes the weighted distance so that depots would be kept near the higher priority routes. The problem can be modeled as follows:

$$\begin{aligned}
\min \quad & \sum_{i=1}^n \sum_{j=1}^m f_j d_{ij} x_{ij} \\
s.t. \quad & \sum_{i=1}^n y_i = K \\
& \sum_{i=1}^n x_{ij} = 1 \quad j = 1, \dots, m \\
& x_{ij} \leq y_i \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& x_{ij} = 0, 1 \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& y_i = 0, 1 \quad i = 1, \dots, n
\end{aligned} \tag{1}$$

The first constraint designates K depots out of n potential depot locations, where the selected depot has the value of y_i equal to one. The second constraint assigns every arc j to one node, which should be a selected depot location shown in the third constraint. The solution of this problem identifies the locations of K depots as well as their sectors expressed by assigned arcs, which have the value of x_{ij} equal to one for depot i .

The greedy heuristic we developed determines which depot to remove or add next, based on which has the most positive impact in terms of the total increased or decreased weighted distance. In other words, when a depot is removed from the list of potential depots, we measure the increase of the weighted distance by re-assigning arcs to the next nearest depot. The sum of these increases for all the arcs assigned to a depot represents the potential impact of the selected depot when it is removed. This value is mainly determined by the number of originally assigned arcs as well as the distance to the next nearest depot. A depot that has many arcs assigned to itself and does not have another depot nearby has a larger negative impact, and, hence, should not be removed from the list of depots. On the other hand, a depot with a smaller number of arcs assigned to itself or that has other depots nearby has the smaller impact. The overall weighted distance will not be increased much even though this depot is removed and its arcs are assigned to other depots. Similarly, a depot can be added to existing depots.

These procedures are repeated when a set of depots is selected from the original depot list or when different combinations of depots are selected for further testing. After the procedure, an initial sector is determined automatically by simply constructing a sector with a depot and arcs assigned to it.

2.3 Initial Route Construction

Since the service for the roadways must be provided according to priority rather than geography, route construction is performed separately for each class of roadways in each sector. This is done by creating four subnetworks per sector, one for each class, from the existing transportation network. Note that it may be necessary to add additional arcs from the other classes to each subnetwork to make them strongly connected. The original arcs are treated as service arcs for the purpose of routing, while any arcs added for connectivity are considered deadhead arcs for that subnetwork.

The initial routes are constructed based on the route-first, cluster-second method suggested by Marks and Stricker (1971) which includes additional constraints. In this approach, the CPT for a given subnetwork is determined first, and then the resulting tour is partitioned into routes based on operational constraints such as vehicle capacity and service frequency.

At the first construction of initial routes, the CPT is simply divided into feasible routes. This generates an upper limit on the number of feasible routes needed to serve the network. In subsequent iterations, some initial routes are combined to create one route, which reduces the number of total routes. This may cause initial routes to be infeasible, but they often become feasible after the solution improvement phase. The iterative process, which gradually reduces the number of total routes, stops when the final routes after solution improvement are still infeasible.

2.4 Solution Improvement

The solution improvement procedure attempts to improve the quality of initial routes and sector assignment. The procedure uses two heuristics: (1) one arc-position movement, and (2) one arc-position exchange. The arc movement heuristic removes a set of arcs in the same location, and then inserts them into other positions. The arc-exchange heuristic attempts moves among multiple routes, where one arc position is removed from each route, and then inserted into the best position within the other respective route. The arc movement is more flexible, in that it can move one arc at a time while the arc exchange algorithm is forced to move at least two arcs. As the distance and duration of a route approaches its maximum limits, it becomes less likely that there is enough capacity to add an additional arc. Exchanging arcs frees up enough capacity to make a move, which was not feasible in the movement algorithm.

These heuristics are similar to one arc movement and exchange heuristics in a single-depot problem by Qiao (1999), but in our multi-depot application, a set of arcs located in the same position sharing the same two end nodes might need to be moved simultaneously because of the sector requirement. For example, consider a segment of four-lane roadway connecting nodes A and B. This segment is considered four arcs with two heading west and the other two heading east in our network, and they should belong to the same sector because these four arcs are placed in the same location. Thus, our algorithm must move four arcs simultaneously if they are moved to another sector.

The heuristics only allow moves that improve the worst route time without further violating operational constraints. That is, the moves must decrease the longest route time of the routes

involved. In addition, both the truck capacity constraint and the cycle time constraint should either be met or improved. Recall that the capacity constraint is expressed as the maximum distance over which trucks can spread material, and the time constraint is 120, 360, and 720 minutes for class A1, A2, and A3 routes, respectively.

During our iterative process, the arc movement heuristic is first applied recursively, until no further improvement is found. Then, the arc-exchange heuristic is applied recursively, until no further improvement exists. The loop of these two algorithms is repeated, first for intra-sector improvement and next for inter-sector improvement, until no more improvement can be found. Moves within the same sector improve the route design, while moves between different sectors improve both route and sector designs.

As an illustrative example, the subroutine of the one arc-position movement between two sectors is given below. The movement within a sector is a special case with the parameter $f_a = 1$ in the subroutine. The arc exchange subroutine can be constructed similarly. These subroutines are applied repeatedly in the heuristic to find a better solution. In the subroutine, a service arc represents an arc that is actually served through spreading and plowing operations while a deadhead arc is used to connect service arcs. Assume that $T(R)$ and $D(R)$ compute the route time and the spreading and plowing distance of route R , respectively.

Step 1.0: Let S_1 and S_2 be the collection of routes of the same class in two different sectors and let t and d be the maximum route time and service distance of a feasible route in the given class, respectively.

Step 1.1: Select a service arc a_1 in a route in S_1 , and let $\{a_1, a_2, \dots, a_{f_a}\}$ be the collection of service arcs that share the same end nodes with a_1 , where f_a represents the number of service arcs (i.e., lanes of a roadway).

Step 1.2: Suppose that $a_i, 1 \leq i \leq f_a$, is a service arc of route R_i . Note that routes R_i and R_j can be the same for $i \neq j$. Replace a_i with deadhead arc(s) and call the new route \tilde{R}_i .

Step 2.1: Select a route in S_2 . Call it Q_i and find the best location to insert a_i . Note that deadhead arcs may need to be added to make Q_i connected. Call the resulting route \tilde{Q}_i .

Step 2.2: If none of the following four statements is true, go to Step 2.3. Otherwise, select a new route in S_2 and go to Step 2.1. If all the routes in S_2 were tested, go to Step 3.1.

- (1) $T(Q_i) \leq t$ and $T(\tilde{Q}_i) > t$,
- (2) $T(Q_i) > t$ and $T(\tilde{Q}_i) > T(Q_i)$,
- (3) $D(Q_i) \leq d$ and $D(\tilde{Q}_i) > d$, and
- (4) $D(Q_i) > d$ and $D(\tilde{Q}_i) > D(Q_i)$.

Step 2.3: If $i = f_a$ continue. Otherwise, Set $i = i + 1$ and go to Step 1.2.

Step 2.4: If $\max_{1 \leq i \leq f_a} \{T(\tilde{R}_i), T(\tilde{Q}_i)\} < \max_{1 \leq i \leq f_a} \{T(R_i), T(Q_i)\}$, accept \tilde{R}_i and \tilde{Q}_i , $1 \leq i \leq f_a$, as new routes in S_1 and S_2 , respectively.

Step 3.1: Select a new service arc in S_1 and go to Step 1.1. If the iterations of this step do not find routes with smaller route times, stop.

2.5 Fleet Configuration and Scheduling

In the last step, the fleet configuration and vehicle scheduling problems are solved based on the solution we have so far. These are solved separately for each depot, since a vehicle can only service routes that are assigned to its home depot. The fleet configuration and scheduling procedures attempt to serve routes with the minimum number of vehicles.

The fleet configuration problem determines how many of each type of vehicle is needed at each of the depots based on the routes associated with the depot. As previously stated, MoDOT's fleet consists of heavy-duty single-axle trucks and extra heavy-duty tandem axle trucks, which have different material capacities and hence different service distances. MoDOT requires the usage of the tandem axle trucks on the roadways in class A1, so that, when preferred, more than the standard amount of material is discharged, based on the intensities and characteristics of storms. MoDOT also prefers to use the single axle trucks for all the other roadways, but tandem axle trucks can be used if the substitution can decrease the total number of trucks operating. The fleet configuration decision is solved iteratively in our algorithm to test whether the use of larger capacity trucks can reduce the total number of trucks required.

The scheduling of trucks is determined by solving a mathematical programming model. Suppose that there are m feasible routes and n available trucks. In addition, s_j and t represent service time for route j and truck refill time, respectively. Because some routes, based on their classes, should be served more often, we use weights f_j to appropriately represent their service frequency. The weights correspond to the target service frequency of six, two, and one during a 12-hour continuous storm for routes of class A1, A2, and A3, respectively. Although service routes are created exclusively for each class, a vehicle can serve multiple routes in different classes with different frequencies. The problem then can be written as follows:

$$\begin{aligned}
& \min \sum_{i=1}^n y_i \\
& \text{s.t.} \quad \sum_{i=1}^n x_{ij} = 1 \quad j = 1, \dots, m \\
& \quad \sum_{j=1}^m f_j (s_j + t) x_{ij} \leq 12 \quad i = 1, \dots, n \\
& \quad x_{ij} \leq y_i \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& \quad x_{ij} = 0, 1 \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& \quad y_i = 0, 1 \quad i = 1, \dots, n
\end{aligned} \tag{2}$$

The objective function is to minimize the number of trucks used, where $y_i = 1$ means that the i -th truck is used. The first constraint assigns a route to one truck, where $x_{ij} = 1$ means that truck i serves route j . The second constraint ensures that each truck serves multiple routes according to its required service frequency within a 12-hour storm period. That is, for example, class A1 routes are guaranteed to be served six times during a 12-hour period. This practically satisfies cycle time constraints in our application because the lengths of feasible routes are limited by the truck capacity restriction. The third constraint makes sure that a route is assigned to only one operating truck.

3.2 Node Map

According to the classified data file, use the longitudes and altitudes of the node points to graph all the nodes .

| U | V | W | X | Y |
|-----------|---------|--------------------------|--------------------------|-----------|
| Total_AAC | Speed_L | Beg_Node_xy | End_Node_xy | Number_of |
| 5660 | 70 | -92.78248213,38.93547508 | -92.77691358,38.93504149 | 1 |
| 5660 | 70 | -92.77641789,38.93557727 | -92.77691358,38.93504149 | 1 |
| 5660 | 70 | -92.77629782,38.93569456 | -92.77641789,38.93557727 | 1 |
| 5660 | 70 | -92.77620642,38.93578162 | -92.77629782,38.93569456 | 1 |
| 5660 | 50 | -92.77562501,38.93641423 | -92.77620642,38.93578162 | 1 |
| 5660 | 50 | -92.77433941,38.93767868 | -92.77562501,38.93641423 | 1 |
| 5660 | 50 | -92.77270037,38.93944435 | -92.77433941,38.93767868 | 1 |
| 5660 | 50 | -92.77186022,38.9406535 | -92.77270037,38.93944435 | 1 |
| 5660 | 50 | -92.77126973,38.94216597 | -92.77186022,38.9406535 | 1 |
| 5660 | 50 | -92.77106498,38.94295864 | -92.77126973,38.94216597 | 1 |
| 5660 | 50 | -92.76972184,38.94781299 | -92.77106498,38.94295864 | 1 |
| 5660 | 50 | -92.76889615,38.94895759 | -92.76972184,38.94781299 | 1 |
| 5660 | 50 | -92.76700003,38.95079867 | -92.76889615,38.94895759 | 1 |
| 5660 | 50 | -92.76354585,38.9539211 | -92.76700003,38.95079867 | 1 |
| 5660 | 50 | -92.76109276,38.95611112 | -92.76354585,38.9539211 | 1 |
| 5660 | 50 | -92.75611789,38.96165702 | -92.76109276,38.95611112 | 1 |
| 5660 | 40 | -92.75513264,38.96256379 | -92.75611789,38.96165702 | 1 |
| 5660 | 40 | -92.75259043,38.96469295 | -92.75513264,38.96256379 | 1 |
| 5660 | 40 | -92.74922046,38.96667129 | -92.75259043,38.96469295 | 1 |
| 5660 | 30 | -92.74818372,38.96684472 | -92.74922046,38.96667129 | 1 |
| 5660 | 30 | -92.74762589,38.96695102 | -92.74818372,38.96684472 | 1 |
| 5660 | 30 | -92.74702306,38.96704922 | -92.74762589,38.96695102 | 1 |
| 5660 | 30 | -92.74464721,38.96747016 | -92.74702306,38.96704922 | 1 |

Figure 3.2. Longitudes and altitudes of nodes

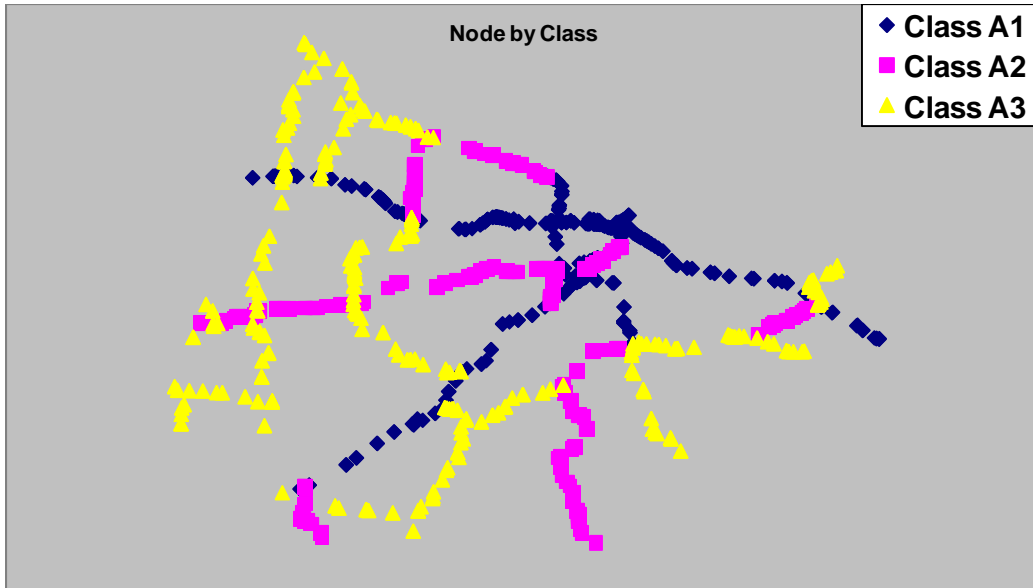


Figure 3.3. A sample node map

3.3 Simplified Road Map

Select the road junction points in the node map and number them. Then, graph the simplified road map by using straight lines to represent roads between two junction points.

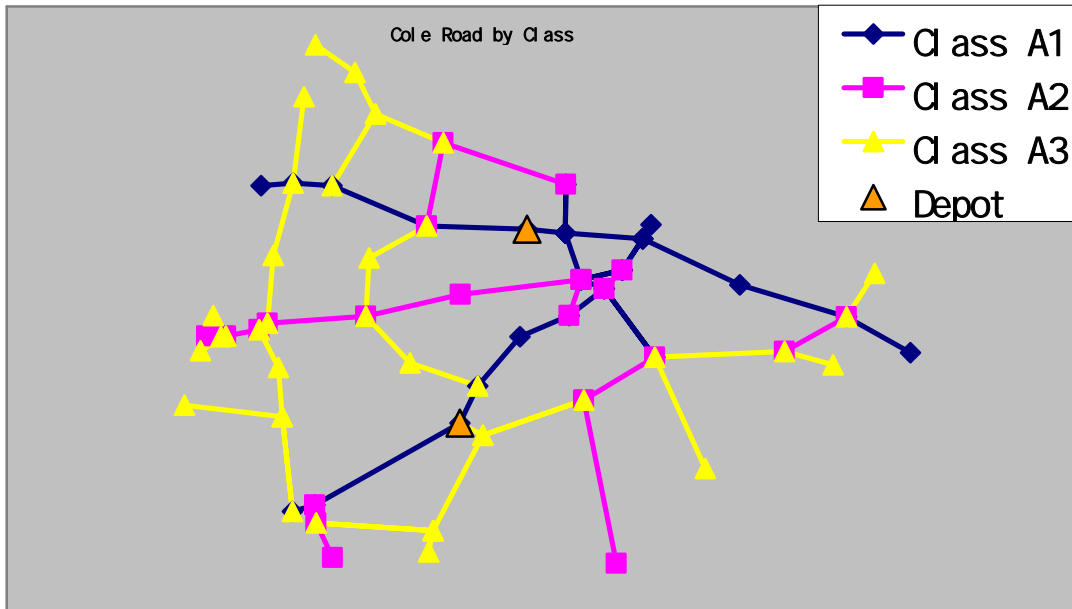


Figure 3.4. A simplified road map

3.4 Road Class Modification

Change the classes of roads or combine different classes if necessary. For example, in the given scenario, Class A2 roads are mostly scattered around. There would be a lot of deadhead distance and waste of time if trucks are only assigned to this class. Therefore, Class A2 and Class A3 are combined and analyzed together.

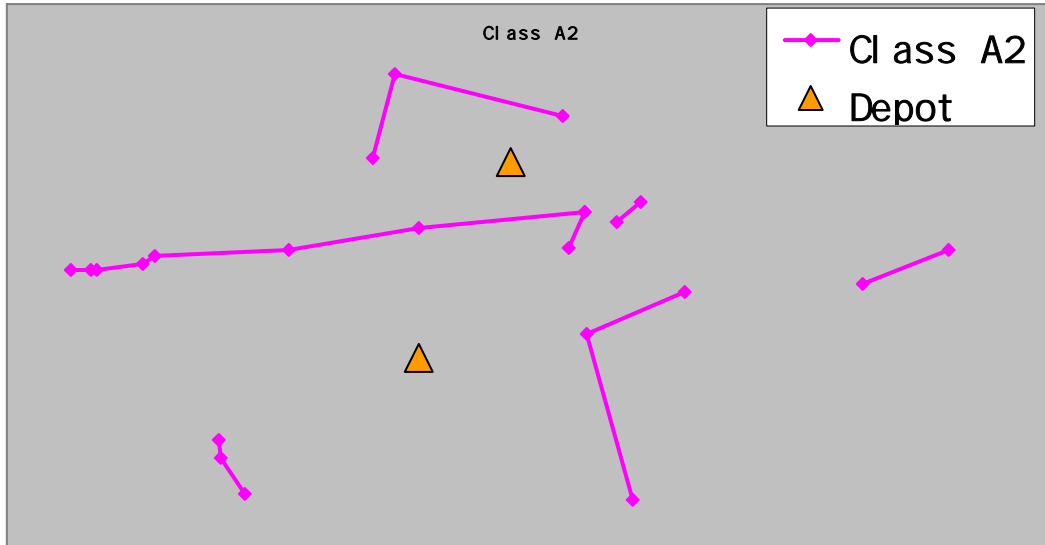


Figure 3.5. Class A2 roads

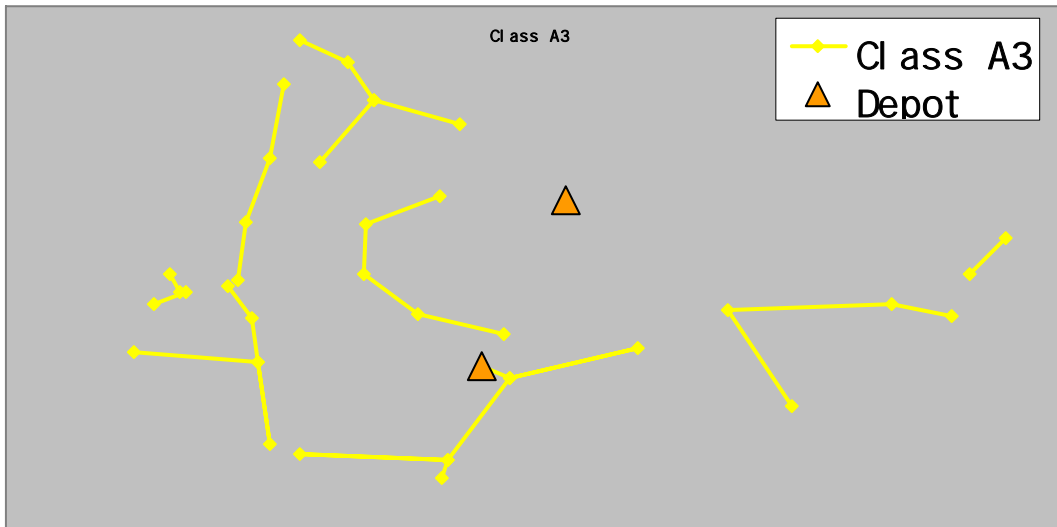


Figure 3.6. Class A3 roads

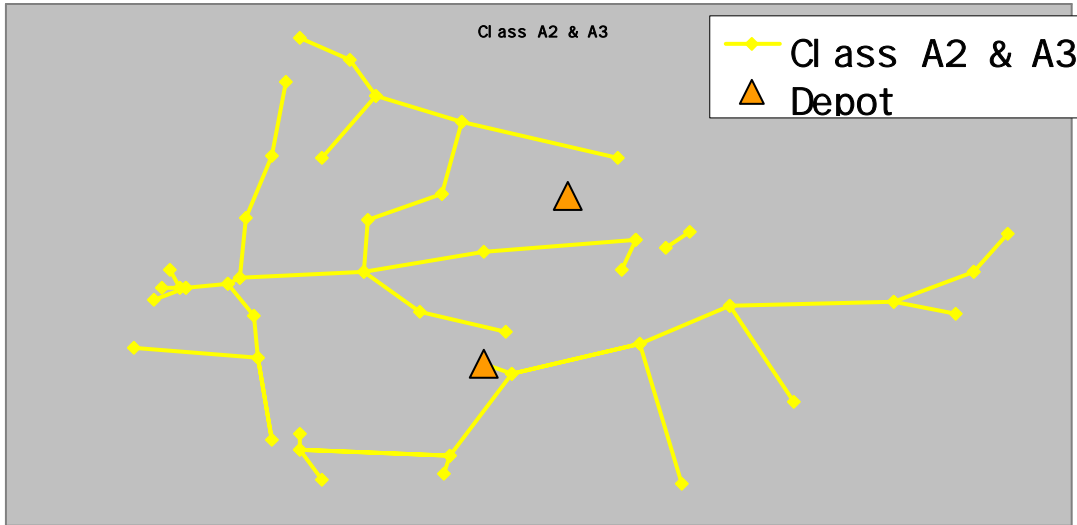


Figure 3.7. Combined A2 and A3 roads

3.5 Data Filtering

Filter and modify the original node information data (EXCEL file) into a usable format (TXT file) in the developed computer programs.

| | A | B | C | D | E | F | G | H |
|----|--------|----------|------|----|----------------------|--------------------------|---------------------|-------|
| 1 | # ID | Alpha ID | From | To | Centerli ne miles | Service Time (min) | Road / Direction | Class |
| 2 | 1, 2 | 50W01 | 1 | 2 | 1.161 | 1.7415 | 50W | A1 |
| 3 | 2, 3 | 50W02 | 2 | 3 | 1.42 | 2.13 | 50W | A1 |
| 4 | 3, 4 | 50W03 | 3 | 4 | 4.101 | 6.1515 | 50W | A1 |
| 5 | 4, 51 | 50W04 | 4 | 51 | 2.86 | 4.29 | 50W | A1 |
| 6 | 4, 51 | 50W05 | 4 | 51 | 2.86 | 4.29 | 50W | A1 |
| 7 | 51, 5 | 50W06 | 51 | 5 | 2.46 | 3.69 | 50W | A1 |
| 8 | 51, 5 | 50W07 | 51 | 5 | 2.46 | 3.69 | 50W | A1 |
| 9 | 5, 10 | 179S01 | 5 | 10 | 3.014 | 4.521 | 179S | A1 |
| 10 | 5, 12 | 179S02 | 5 | 12 | 2.338 | 3.507 | 179S | A1 |
| 11 | 12, 13 | CW01 | 12 | 13 | 1.626 | 2.439 | CW | A1 |
| 12 | 12, 14 | 179S03 | 12 | 14 | 1.064 | 1.596 | 179S | A1 |
| 13 | 14, 15 | BS01 | 14 | 15 | 4.678 | 7.017 | BS | A1 |
| 14 | 14, 16 | 54W01 | 14 | 16 | 1.899 | 2.8485 | 54W | A1 |
| 15 | 14, 16 | 54W02 | 14 | 16 | 1.899 | 2.8485 | 54W | A1 |
| 16 | 16, 17 | 54W03 | 16 | 17 | 2.098 | 3.147 | 54W | A1 |

Figure 3.8. Road information data (EXCEL file)

| Start Node | End Node | Class | Start ID | End ID | Distance | Weight | Class | Depot |
|------------|----------|-------|----------|--------|----------|--------|-------|-------|
| "1, 2" | 50w01 | 1 | 2 | 1.161 | 1.7415 | 50w | A1 | |
| "2, 3" | 50w02 | 2 | 3 | 1.42 | 2.13 | 50w | A1 | |
| "3, 4" | 50w03 | 3 | 4 | 4.101 | 6.1515 | 50w | A1 | |
| "4, 51" | 50w04 | 4 | 51 | 2.86 | 4.29 | 50w | A1 | |
| "4, 51" | 50w05 | 4 | 51 | 2.86 | 4.29 | 50w | A1 | |
| "51, 5" | 50w06 | 51 | 5 | 2.46 | 3.69 | 50w | A1 | |
| "51, 5" | 50w07 | 51 | 5 | 2.46 | 3.69 | 50w | A1 | |
| "5, 10" | 179s01 | 5 | 10 | 3.014 | 4.521 | 179s | A1 | |
| "5, 12" | 179s02 | 5 | 12 | 2.338 | 3.507 | 179s | A1 | |
| "12, 13" | Cw01 | 12 | 13 | 1.626 | 2.439 | Cw | A1 | |
| "12, 14" | 179s03 | 12 | 14 | 1.064 | 1.596 | 179s | A1 | |
| "14, 15" | BS01 | 14 | 15 | 4.678 | 7.017 | BS | A1 | |
| "14, 16" | 54w01 | 14 | 16 | 1.899 | 2.8485 | 54w | A1 | |
| "14, 16" | 54w02 | 14 | 16 | 1.899 | 2.8485 | 54w | A1 | |
| "16, 17" | 54w03 | 16 | 17 | 2.098 | 3.147 | 54w | A1 | |
| "16, 17" | 54w04 | 16 | 17 | 2.098 | 3.147 | 54w | A1 | |
| "17, 18" | 54w05 | 17 | 18 | 3.157 | 4.7355 | 54w | A1 | |
| "17, 18" | 54w06 | 17 | 18 | 3.157 | 4.7355 | 54w | A1 | |

Figure 3.9. Road information data (TXT file)

3.6 Program Running

Run the following developed programs sequentially. See Appendix A for programs.

1. Run CPP.java, which generates LeastNtime.txt and LeastDistance.txt files;
2. Run AssignDpt.java, which assigns roads in the same class to the nearest depot;
3. Run CPP1.java, which generates the Chinese Postman Tour for each depot and class;
4. Run TourDiv2.java, which divides the big Chinese Postman Tour into smaller units based on the service time and vehicle capacity limit;
5. Run CPP.java again to the smaller tours; and
6. Run AddDH.java, which adds deadheading roads into the smaller tours to make them complete.

3.7 Route Result Modification and Route Map Generation

The desired routes are generated after running all the programs sequentially. However, it is often necessary to modify computer-generated routes so that they are better fit to human operation even though quantitative measures deteriorate. For example, Figure 3.10 and 3.11 show computer-generated results and a map. However, the route7_A1_83 (green route) has two separated sub routes. Even though this is globally optimal, the local operation does not like separate routes. So, one of them is combined into another route, route9_A1_83, shown in Figure 3.12.

Cole_DH_A1 - Notepad

File Edit Format View Help

| Route | Arc | From | To | Miles | Time | Class |
|-------|-------|------|----|-------|--------|-------|
| 1 | 50E10 | 51 | 4 | 2.86 | 4.29 | A1 |
| 1 | 50W04 | 4 | 51 | 2.86 | 4.29 | A1 |
| 1 | 50W07 | 51 | 5 | 2.46 | 3.69 | A1 |
| 1 | 54w14 | 5 | 6 | 2.883 | 4.3245 | A1 |
| 1 | 50E01 | 6 | 7 | 4.512 | 6.768 | A1 |
| 1 | 50E04 | 7 | 8 | 4.328 | 6.492 | A1 |
| 1 | 50E06 | 8 | 9 | 2.973 | 4.4595 | A1 |
| 1 | 50W13 | 9 | 8 | 2.973 | 4.4595 | A1 |
| 1 | 50E05 | 8 | 9 | 2.973 | 4.4595 | A1 |
| 1 | 50W12 | 9 | 8 | 2.973 | 4.4595 | A1 |
| 1 | 50W11 | 8 | 7 | 4.328 | 6.492 | A1 |
| 1 | 50E03 | 7 | 8 | 4.328 | 6.492 | A1 |
| 1 | 50W10 | 8 | 7 | 4.328 | 6.492 | A1 |
| 1 | 50W08 | 7 | 6 | 4.512 | 6.768 | A1 |
| 1 | 54w18 | 6 | 11 | 0.609 | 0.9135 | A1 |
| 1 | 54E18 | 11 | 6 | 0.609 | 0.9135 | A1 |
| 1 | 54w17 | 6 | 11 | 0.609 | 0.9135 | A1 |
| 1 | 54E17 | 11 | 6 | 0.609 | 0.9135 | A1 |
| 1 | 54w16 | 6 | 13 | 2.109 | 3.1635 | A1 |
| 1 | 54E16 | 13 | 6 | 2.109 | 3.1635 | A1 |
| 1 | 54w15 | 6 | 13 | 2.109 | 3.1635 | A1 |
| 1 | 54E15 | 13 | 6 | 2.109 | 3.1635 | A1 |
| 1 | 54E14 | 6 | 5 | 2.883 | 4.3245 | A1 |
| 1 | 54w13 | 5 | 6 | 2.883 | 4.3245 | A1 |
| 1 | 54E13 | 6 | 5 | 2.883 | 4.3245 | A1 |
| 1 | 50E12 | 5 | 51 | 2.46 | 2.46 | 50E-D |
| 2 | 50W06 | 51 | 5 | 2.46 | 2.46 | 50W-D |
| 2 | 54w13 | 5 | 6 | 2.883 | 2.883 | 50W-D |
| 2 | 50E02 | 6 | 7 | 4.512 | 6.768 | A1 |
| 2 | 50W09 | 7 | 6 | 4.512 | 6.768 | A1 |

Route Number

Service Road

Deadheading

Figure 3.10. Route result details

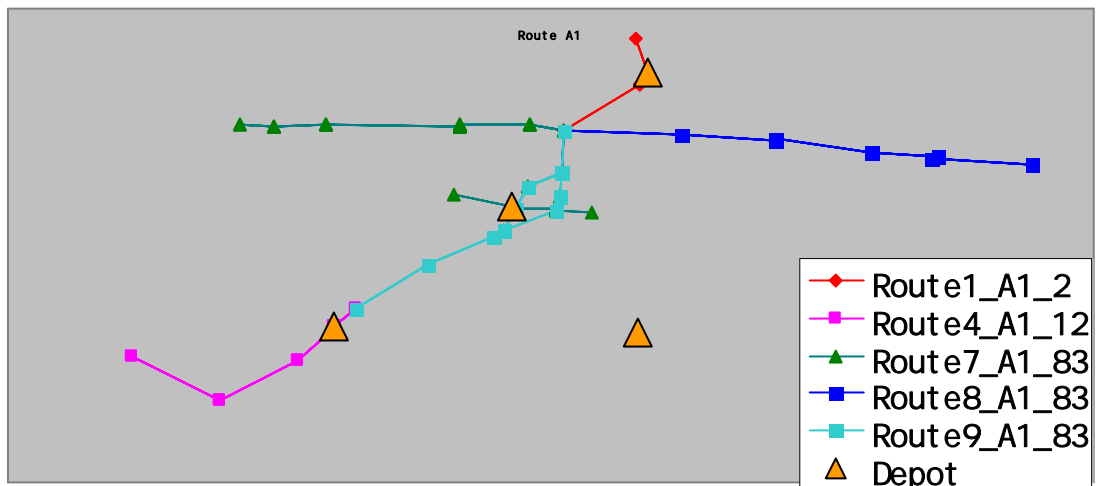


Figure 3.11. Routes before modification

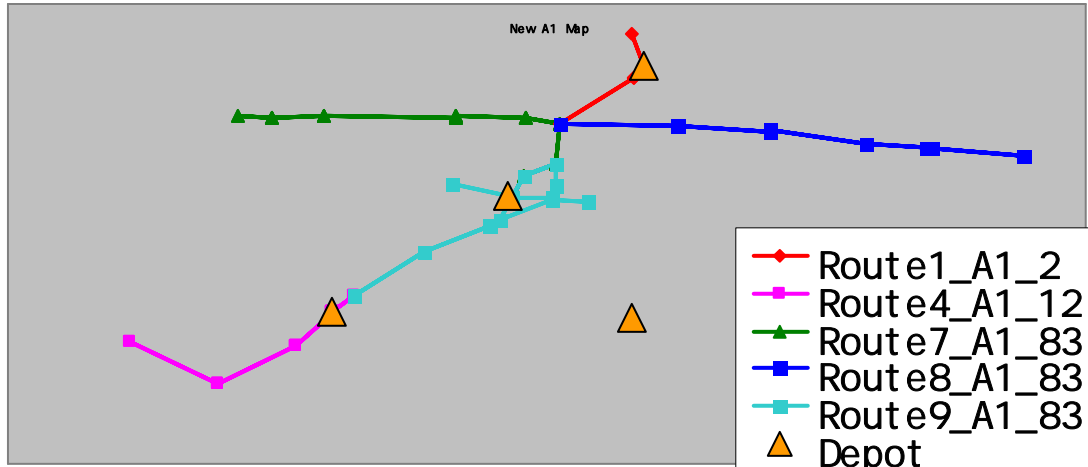


Figure 3.12. Routes after modification

3.8 Truck Assignment

Compute the total service and deadheading time and distance for each route. Then, assign routes to trucks to make the operation plan. According to the duration time limit, routes can be combined to minimize the number of trucks.

Table 3. A sample summary of service and deadheading time and distance

| Route | Class | Depot | Total time | Service time | DH time | Total distance | Service distance | DH distance |
|-------|-------|----------------|------------|--------------|---------|----------------|------------------|-------------|
| 1 | A1 | Brazito | 1.49 | 1.49 | 0.00 | 59.64 | 59.64 | 0.00 |
| 2 | A1 | Jefferson City | 1.64 | 1.48 | 0.16 | 67.39 | 57.82 | 9.57 |
| 3 | A1 | Jefferson City | 1.55 | 1.47 | 0.08 | 63.70 | 58.78 | 4.92 |
| 4 | A1 | Jefferson City | 1.66 | 1.11 | 0.54 | 72.31 | 39.79 | 32.53 |
| 5 | A3 | Brazito | 2.60 | 2.40 | 0.20 | 84.09 | 72.06 | 12.03 |
| 6 | A3 | Brazito | 2.56 | 2.52 | 0.04 | 77.83 | 75.56 | 2.27 |
| 7 | A3 | Jefferson City | 2.30 | 2.02 | 0.28 | 77.33 | 60.57 | 16.76 |
| 8 | A3 | Jefferson City | 1.89 | 1.57 | 0.32 | 66.36 | 47.23 | 19.12 |

Table 4. A sample operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|----------------|----------|-------|---------|----------|-------------|------------------|
| Brazito | 1 | A1 | 1.49 | 1 | | 1.49 |
| | 5 | A3 | 2.60 | 2 | 5-6 | 5.16 |
| | 6 | A3 | 2.56 | | | 5.16 |
| Jefferson City | 2 | A1 | 1.64 | 3 | | 1.64 |
| | 3 | A1 | 1.55 | 4 | | 1.55 |
| | 4 | A1 | 1.66 | 5 | | 1.66 |
| | 7 | A3 | 2.30 | 6 | 8-7 | 4.19 |
| | 8 | A3 | 1.89 | | | 4.19 |

4. ANALYSIS AND RESULTS

The solution procedure described in the previous chapter was applied to the transportation network of District 5 in Missouri. The results for individual counties are described in sections one through seven, and the summary of those is given in Section eight.

4.1 Cole County

Figure 4.1.1 shows the existing two depots in Jefferson City and Brazito and service area in Cole County.

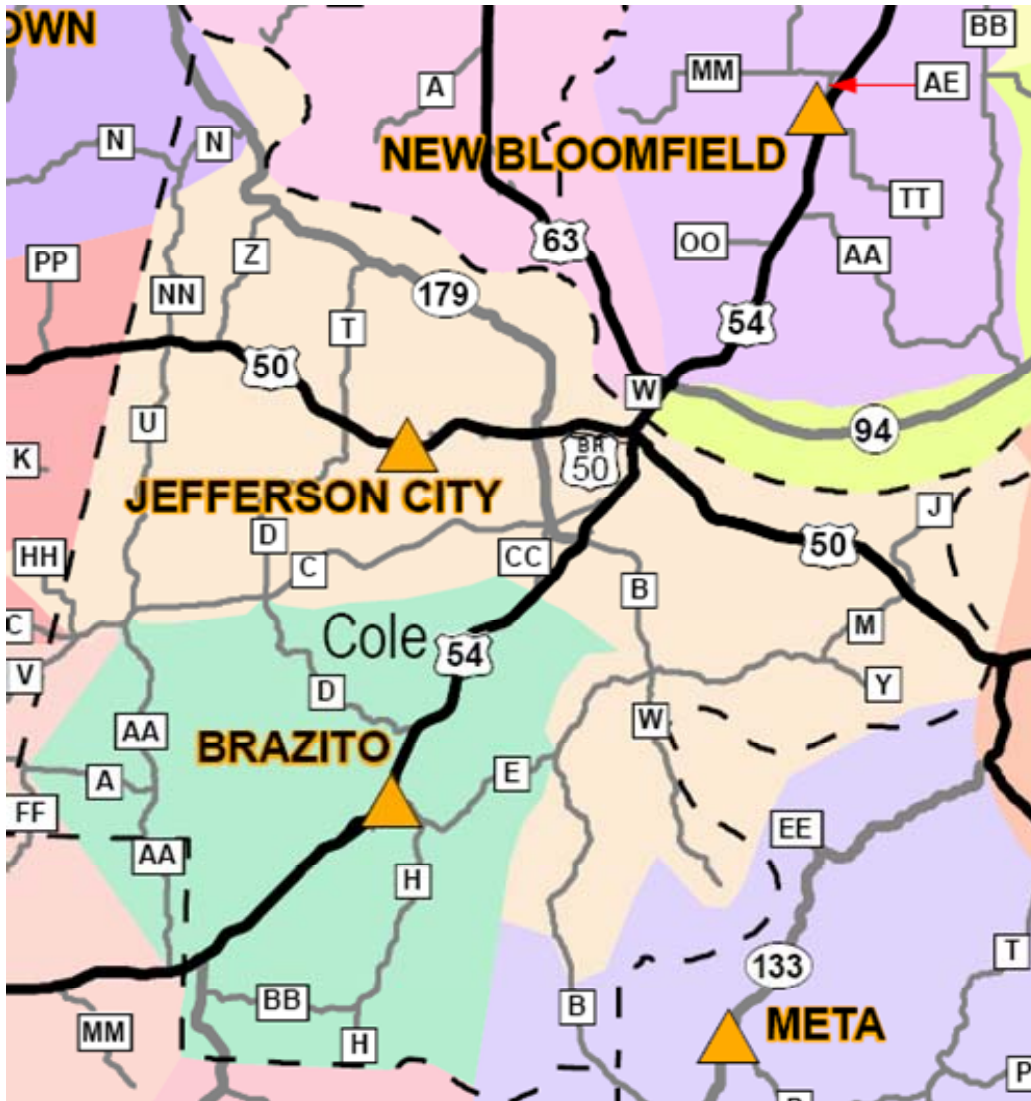


Figure 4.1.1. Cole original map

The complex original map can be simplified by nodes and lines, which represent road junction points and roads between junctions. Figure 4.1.2 shows the simplified map of Cole County with roads distinguished by their classes.

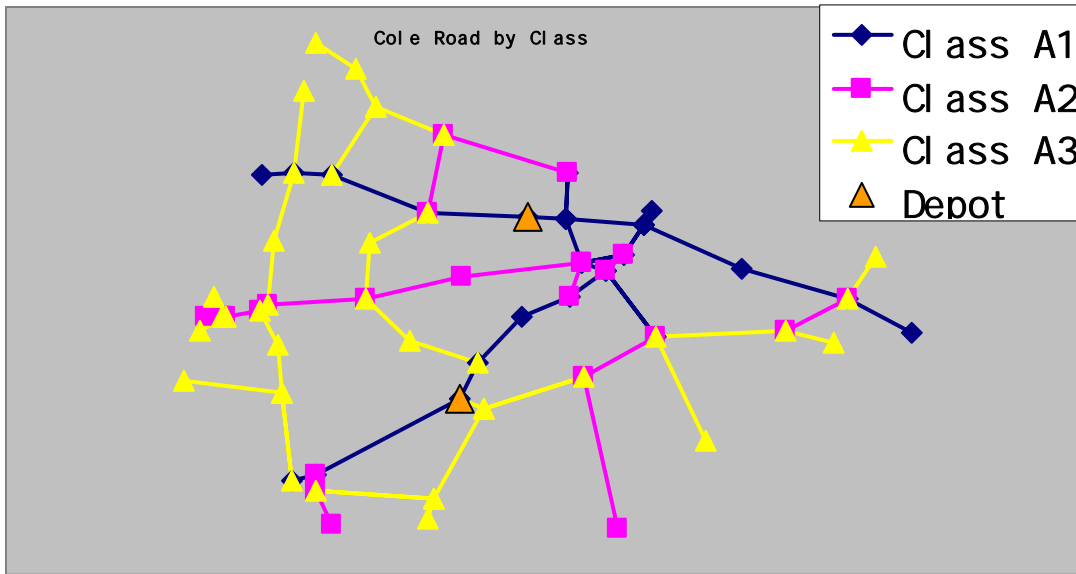


Figure 4.1.2. Simplified map

Figures 4.1.3–4.1.8 show the road maps for each class and best routes to serve. Because Class A2 roads are mostly scattered around, roads in Classes A2 and A3 are analyzed together. The routes assigned to depots in Brazito and Jefferson City are denoted by numbers 19 and 51, respectively.

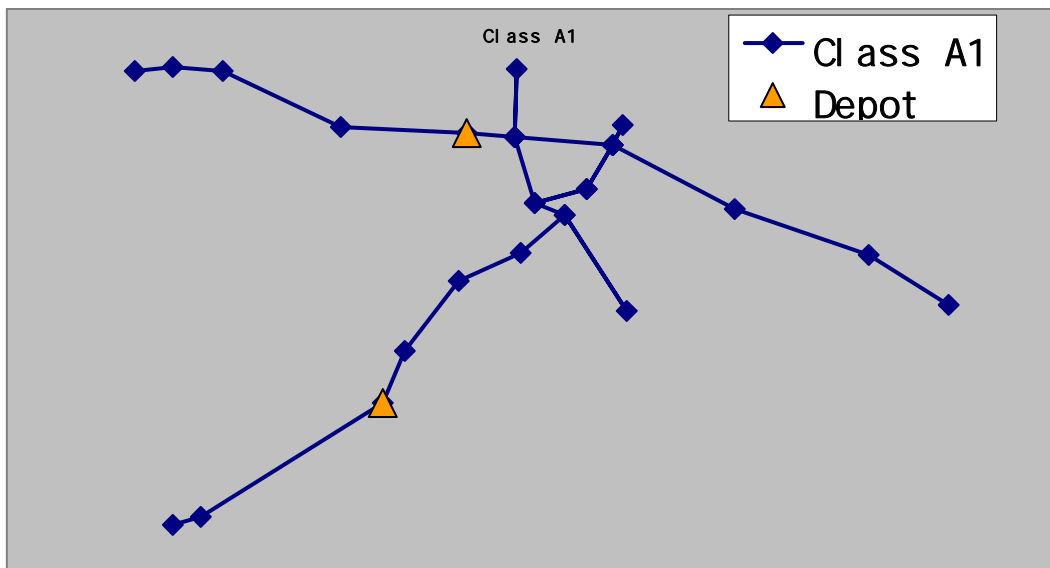


Figure 4.1.3. Class A1 roads

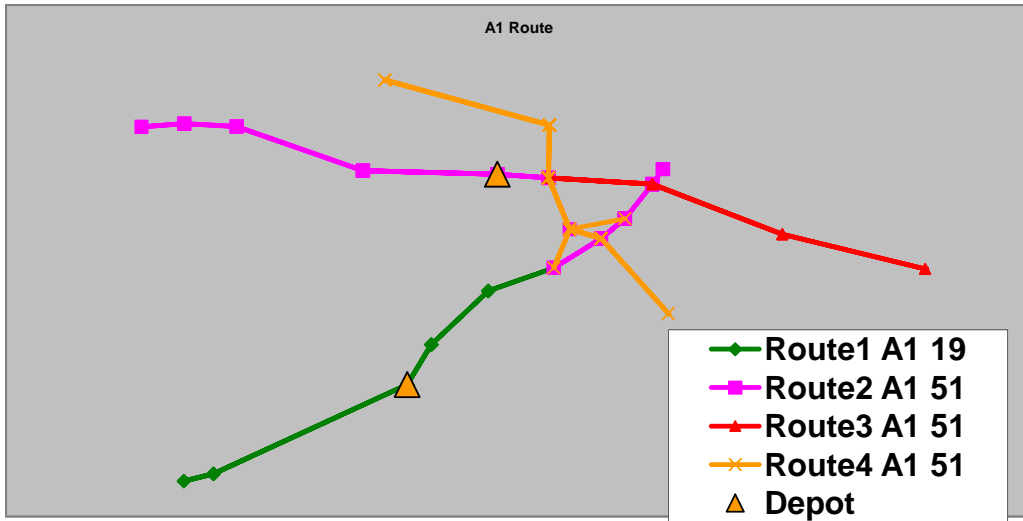


Figure 4.1.4. Class A1 routes

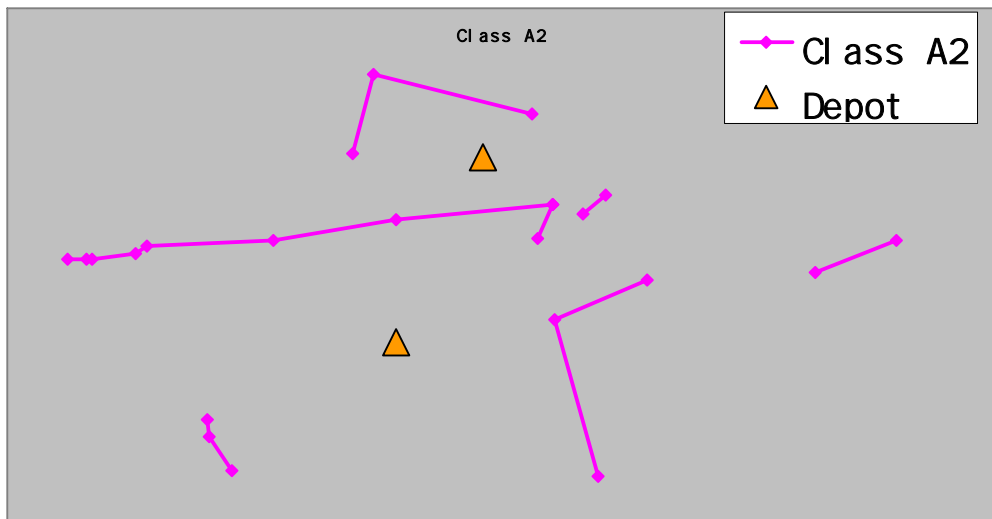


Figure 4.1.5. Class A2 roads

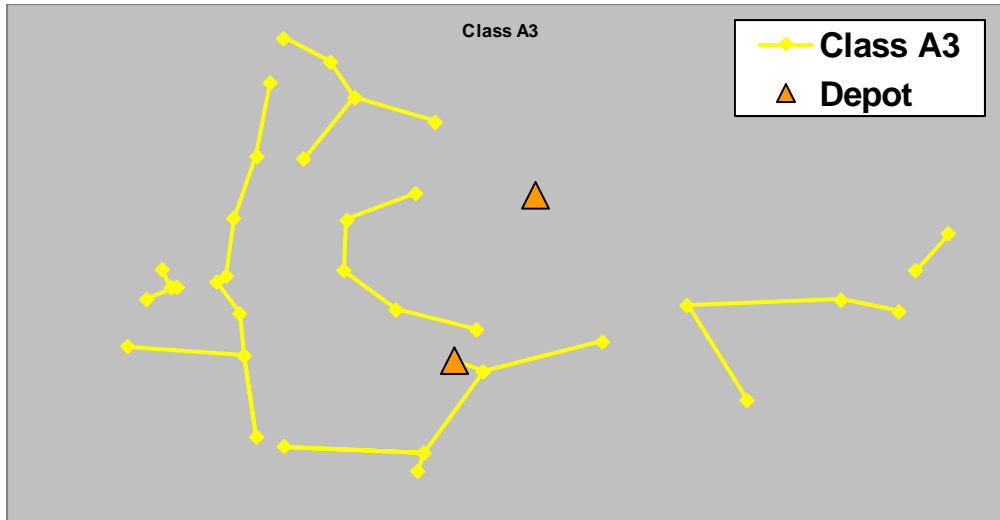


Figure 4.1.6. Class A3 roads

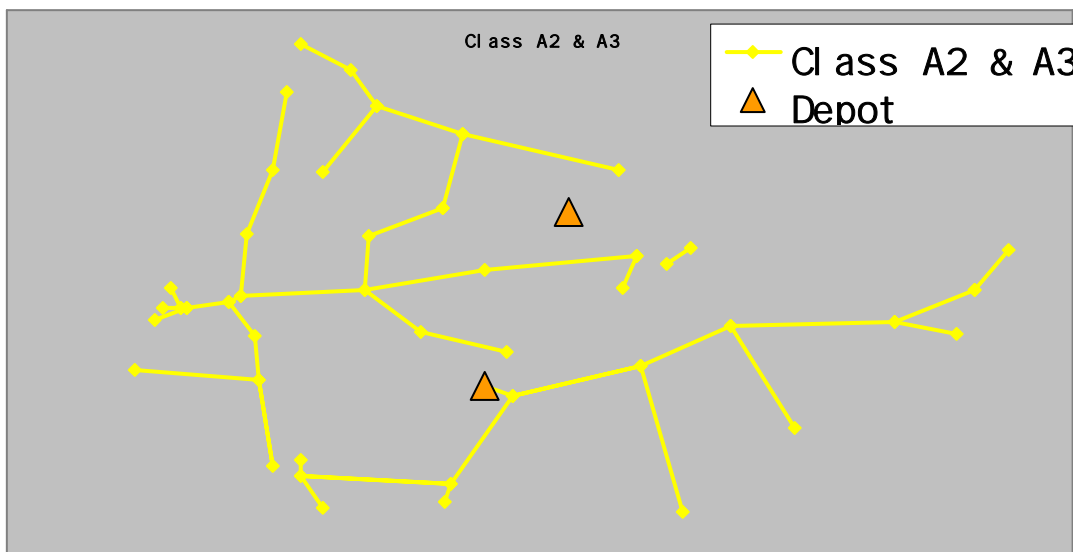


Figure 4.1.7. Classes A2 and A3 roads

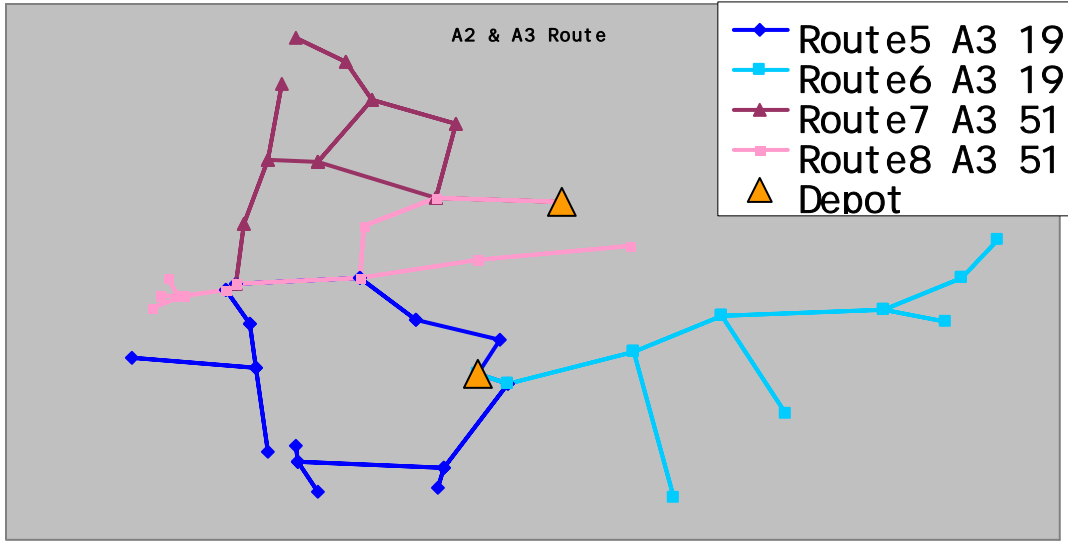


Figure 4.1.8. Classes A2 and A3 routes

Figure 4.1.9 graphically shows final routes recommended for the county, and Table 5 describes these routes.

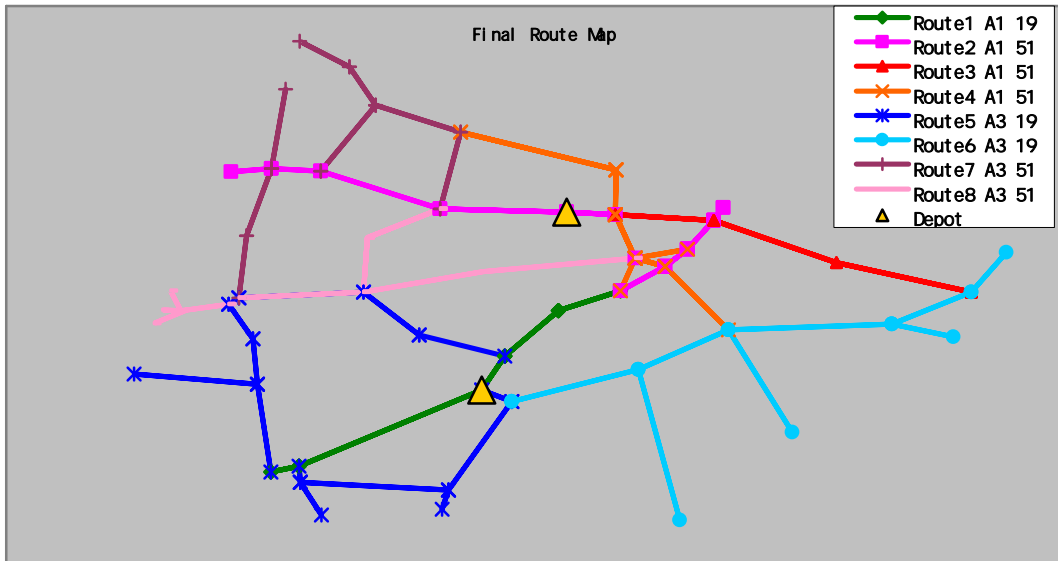


Figure 4.1.9. Final routes

Table 5. Route description

| Route | Class | Depot | Description |
|-------|-------|----------------|---|
| 1 | A1 | Brazito | 54, from int. with AA to int. with CC |
| 2 | A1 | Jefferson City | 50, from int. with 179 towards west; 54, from int. with CC towards north |
| 3 | A1 | Jefferson City | 50, from int. with 179 towards east |
| 4 | A1 | Jefferson City | 179, from int. with T to int. with 54; B, from int. with 54 to int. with W; C, from int. with 179 to int. with 54; CC |
| 5 | A3 | Brazito | D, from int. with 54 to int. with C; AA; A; H; BB; 17, from int. with 54 towards south |
| 6 | A3 | Brazito | J; M; Y; W; B, from int. with W towards south; E |
| 7 | A3 | Jefferson City | 179, from int. with T towards north; T; Z; NN; U; |
| 8 | A3 | Jefferson City | D, from int. with 50 to int. with C; C, from int. with 179 towards west; HH; V |

The summary of these routes is given in Table 6. The table shows necessary time and distance to serve each route. As shown in Table 7, these routes can be run by six snow plow trucks.

Table 6. Summary of routes

| Route | Class | Depot | Total time | Service time | DH time | Total distance | Service distance | DH Distance |
|-------|-------|----------------|------------|--------------|---------|----------------|------------------|-------------|
| 1 | A1 | Brazito | 1.49 | 1.49 | 0.00 | 59.64 | 59.64 | 0.00 |
| 2 | A1 | Jefferson City | 1.64 | 1.48 | 0.16 | 67.39 | 57.82 | 9.57 |
| 3 | A1 | Jefferson City | 1.55 | 1.47 | 0.08 | 63.70 | 58.78 | 4.92 |
| 4 | A1 | Jefferson City | 1.66 | 1.11 | 0.54 | 72.31 | 39.79 | 32.53 |
| 5 | A3 | Brazito | 2.60 | 2.40 | 0.20 | 84.09 | 72.06 | 12.03 |
| 6 | A3 | Brazito | 2.56 | 2.52 | 0.04 | 77.83 | 75.56 | 2.27 |
| 7 | A3 | Jefferson City | 2.30 | 2.02 | 0.28 | 77.33 | 60.57 | 16.76 |
| 8 | A3 | Jefferson City | 1.89 | 1.57 | 0.32 | 66.36 | 47.23 | 19.12 |

Table 7. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|----------------|----------|-------|---------|----------|-------------|------------------|
| Brazito | 1 | A1 | 1.49 | 1 | | 1.49 |
| | 5 | A3 | 2.60 | 2 | 5-6 | 5.16 |
| | 6 | A3 | 2.56 | | | 5.16 |
| Jefferson City | 2 | A1 | 1.64 | 3 | | 1.64 |
| | 3 | A1 | 1.55 | 4 | | 1.55 |
| | 4 | A1 | 1.66 | 5 | | 1.66 |
| | 7 | A3 | 2.30 | 6 | 8-7 | 4.19 |
| | 8 | A3 | 1.89 | | | 4.19 |

4.2 Callaway County

Figure 4.2.1 shows the existing four depots in Auxvasse, Fulton, Loomfield, and Mokane, and service area in Callaway County.

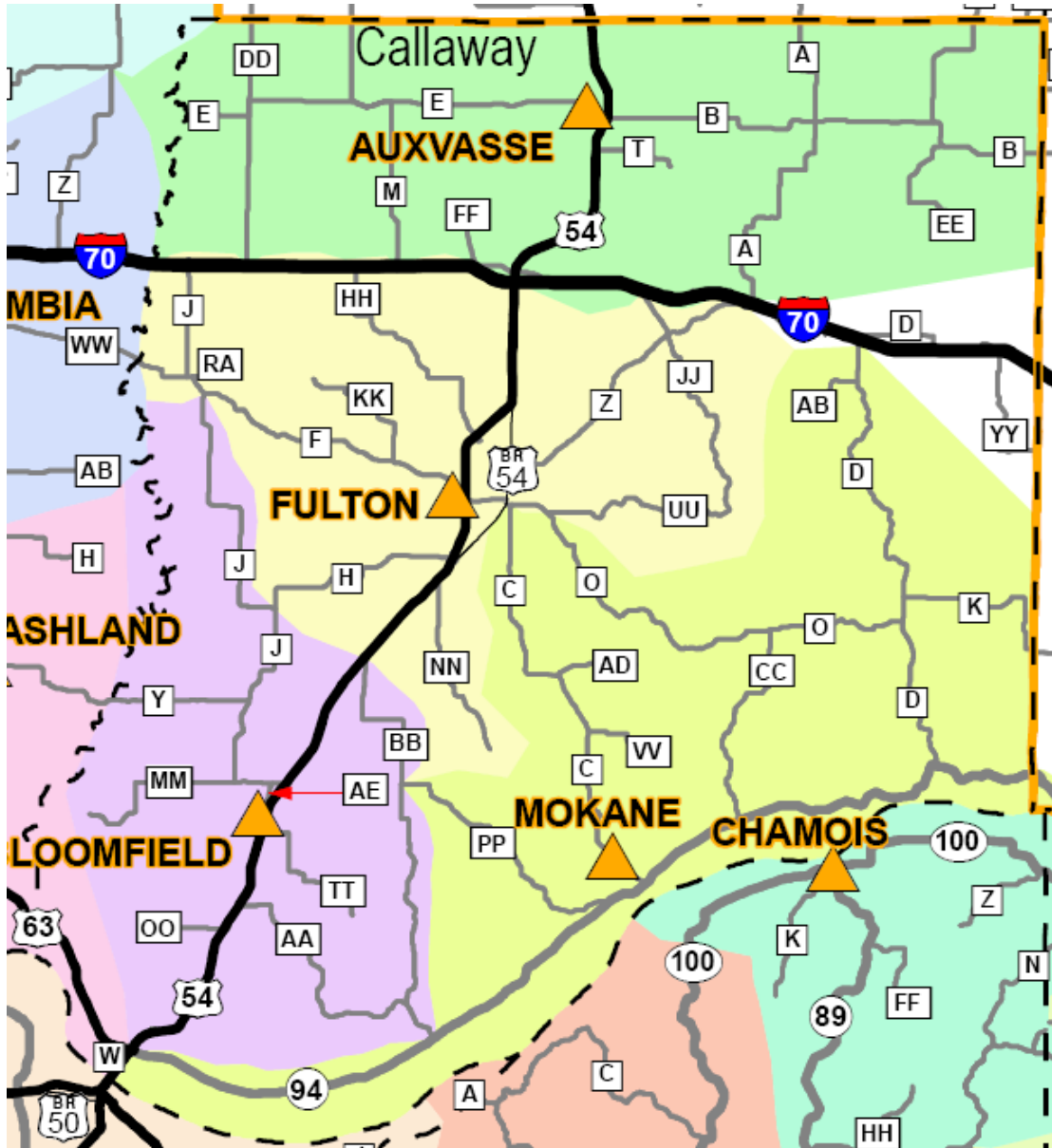


Figure 4.2.1. Callaway original map

Figure 4.2.2 shows the simplified map of Callaway County with roads distinguished by their classes.

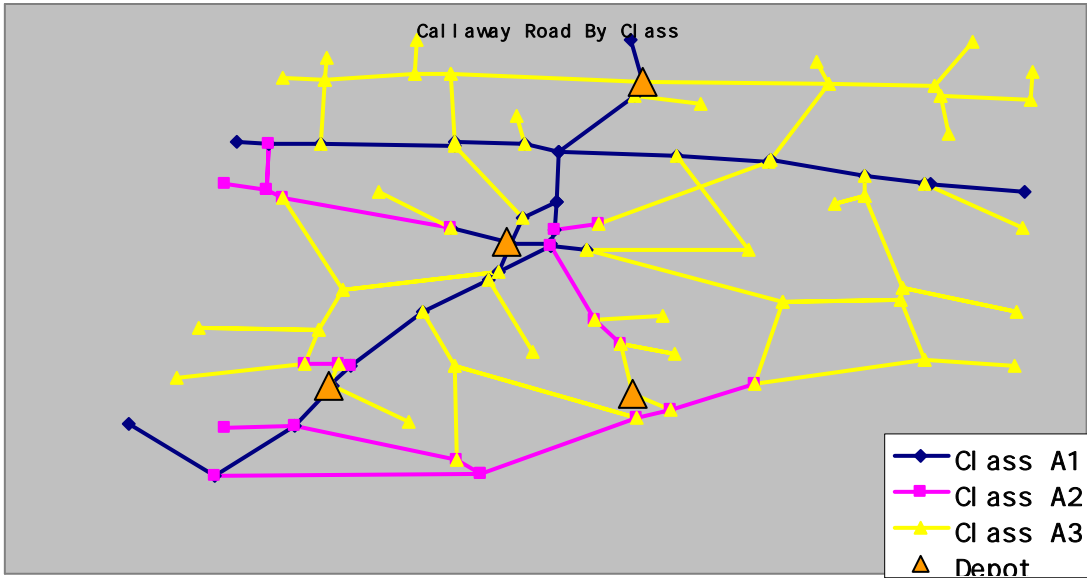


Figure 4.2.2. Simplified map

Figures 4.2.3–4.2.8 show the road maps for each class and best routes to serve. The routes assigned to depots in Auxvasse, Fulton, Loomfield, and Mokane are denoted by numbers 2, 83, 12 and 84, respectively.

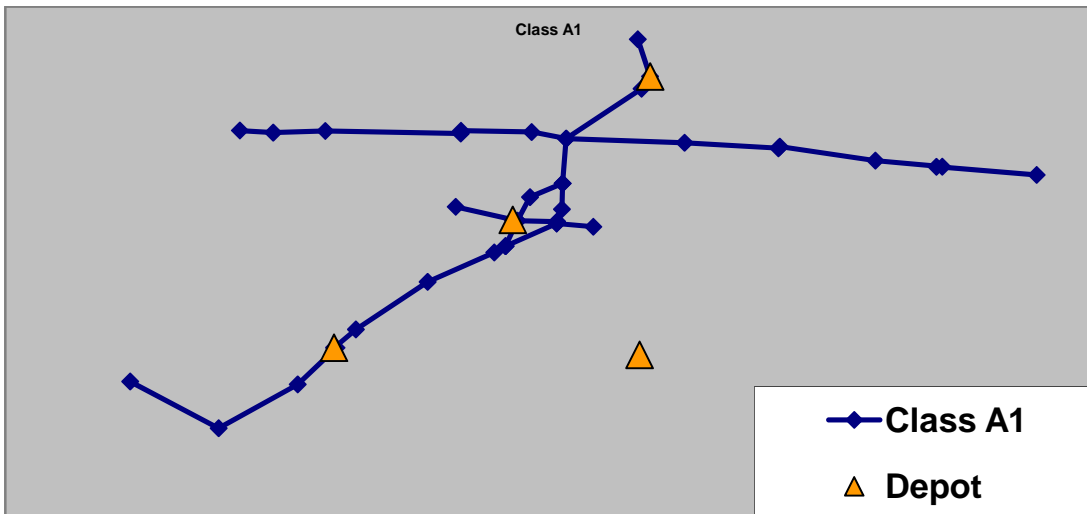


Figure 4.2.3. Class A1 roads

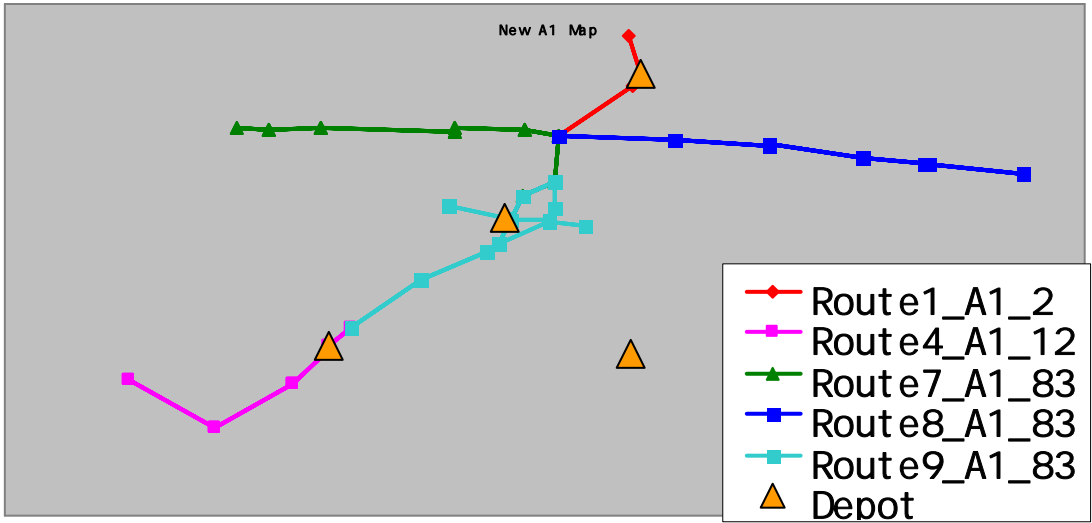


Figure 4.2.4. Class A1 roads

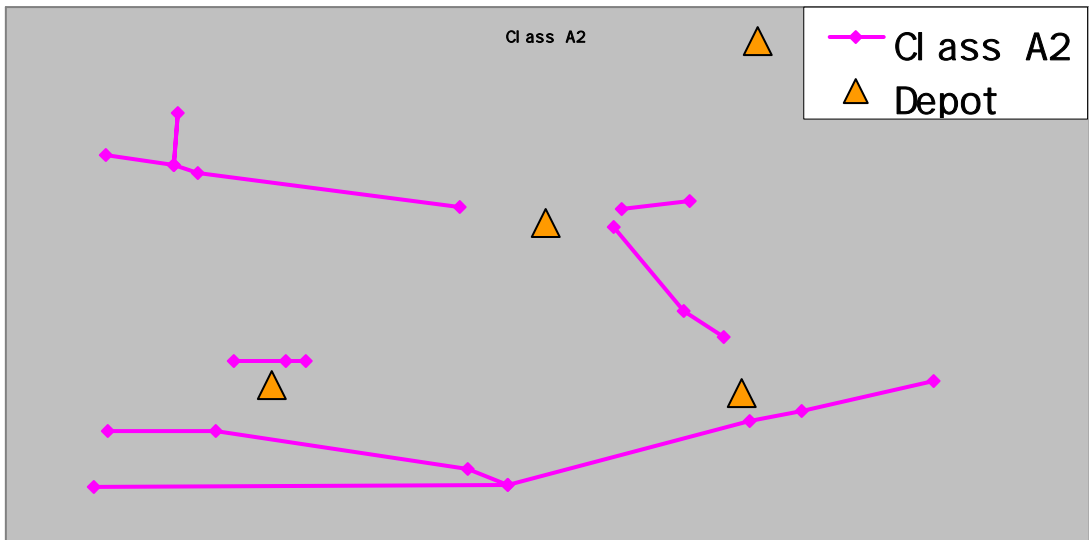


Figure 4.2.5. Class A2 roads

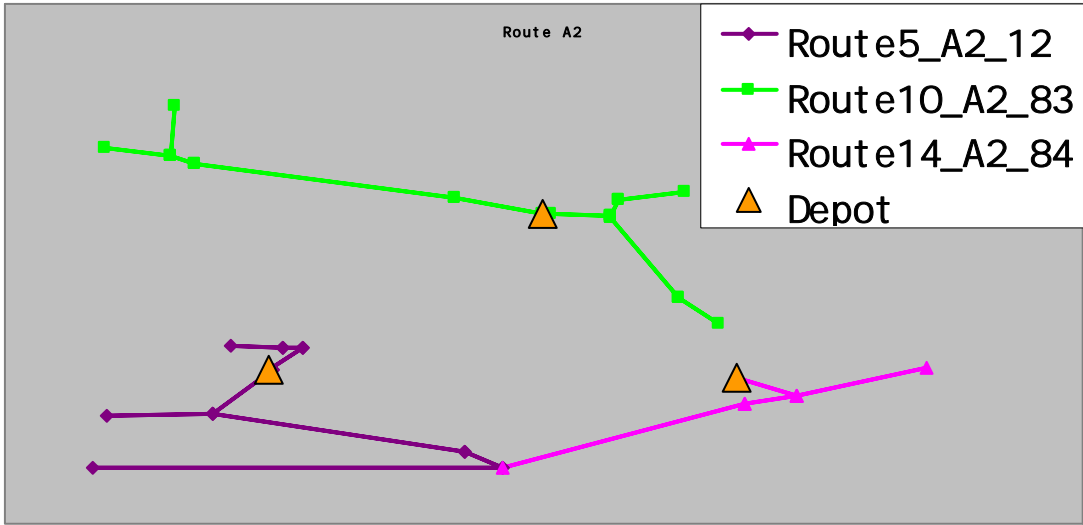


Figure 4.2.6. Class A2 routes

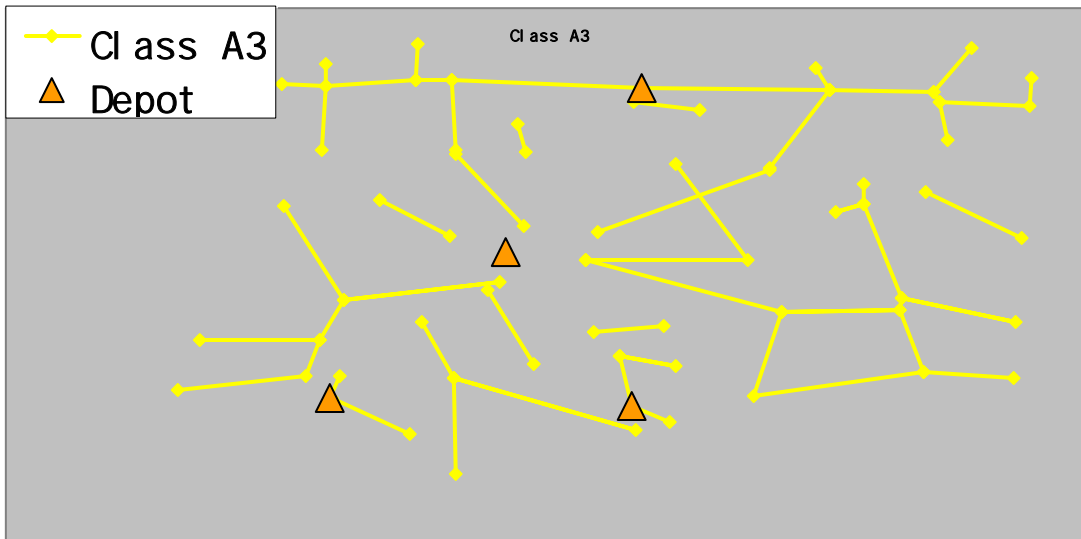


Figure 4.2.7. Class A3 roads

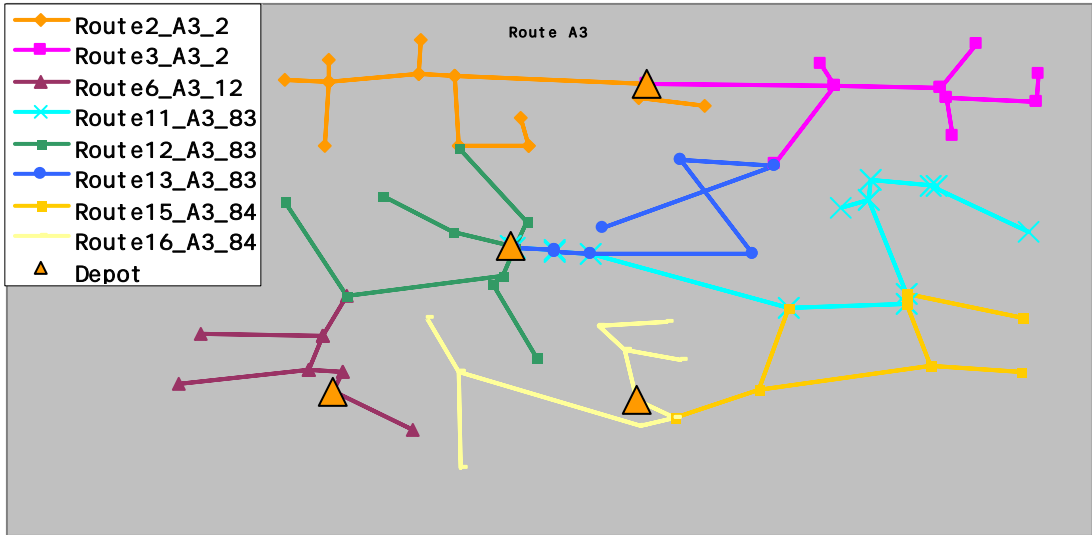


Figure 4.2.8. Class A3 routes

Figure 4.2.9 graphically shows final routes recommended for the county, and Table 8 describes these routes.

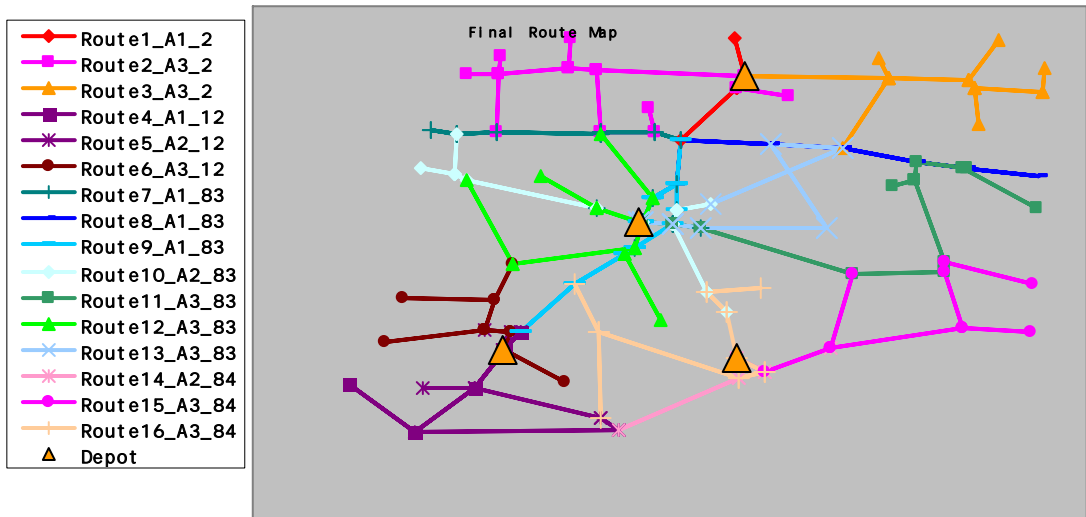


Figure 4.2.9. Final routes

Table 8. Route description

| Route | Class | Depot | Description |
|--------------|--------------|--------------|---|
| 1 | A1 | Auxvasse | 54, from int. with 70 towards north |
| 2 | A3 | Auxvasse | E; DD; M; E; FF; T |
| 3 | A3 | Auxvasse | B; A; N; EE; ZZ |
| 4 | A1 | Loomfield | 54, from int. with 63 to int. with J; 63, from int. with 54 towards west |
| 5 | A2 | Loomfield | J, form int. with MM to int. with 54; OO; AA; 94, from int. with 54 to int. with AA |
| 6 | A3 | Loomfield | J, from int. with H to int. with MM; Y; MM; TT |
| 7 | A1 | Fulton | 70, from int. with 54 towards west; 54, from int with 70 to int with BR54; |
| 8 | A1 | Fulton | 70, from int. with 54 towards east |
| 9 | A1 | Fulton | 54, from int. with BR54 to int. with J; F, from int. with KK to int. with O; O; from int. with C to int. with UU |
| 10 | A2 | Fulton | WW; RA; J, from int. with 70 to int. with RA; F, from int. with RA to int. with KK; Z, from int. with 54; C, from int. with O to int. with VV |
| 11 | A3 | Fulton | O, from int. with UU to int. with D; D, from int. with O to int. with 70; AB; YY |
| 12 | A3 | Fulton | HH; KK; J, from int. with F to int. with H; H; NN |
| 13 | A3 | Fulton | UU; JJ; Z |
| 14 | A2 | Mokane | 94, from int. with AA to int. with CC |
| 15 | A3 | Mokane | 94, int. with CC towards east; CC; D, from int. with O to int. with 94; K |
| 16 | A3 | Mokane | BB; PP; C, int. with VV to int. with 94; AD; VV |

The summary of these routes is given in Table 9. The table shows necessary time and distance to serve each route. As shown in Table 10, these routes can be run by ten snow plow trucks.

Table 9. Summary of routes

| Route | Class | Depot | Total time | Service time | DH time | Total distance | Service distance | DH distance |
|-------|-------|-----------|------------|--------------|---------|----------------|------------------|-------------|
| 1 | A1 | Auxvasse | 0.96 | 0.96 | 0.00 | 38.40 | 38.40 | 0.00 |
| 2 | A3 | Auxvasse | 2.44 | 2.32 | 0.13 | 77.09 | 69.57 | 7.52 |
| 3 | A3 | Auxvasse | 2.43 | 2.43 | 0.00 | 73.00 | 73.00 | 0.00 |
| 4 | A1 | Loomfield | 1.58 | 1.58 | 0.00 | 63.29 | 63.29 | 0.00 |
| 5 | A2 | Loomfield | 1.81 | 1.64 | 0.18 | 59.77 | 49.07 | 10.69 |
| 6 | A3 | Loomfield | 1.57 | 1.52 | 0.05 | 48.41 | 45.67 | 2.73 |
| 7 | A1 | Fulton | 1.64 | 1.51 | 0.13 | 68.16 | 60.40 | 7.76 |
| 8 | A1 | Fulton | 1.93 | 1.68 | 0.25 | 82.13 | 67.10 | 15.04 |
| 9 | A1 | Fulton | 1.99 | 1.99 | 0.00 | 79.51 | 79.51 | 0.00 |
| 10 | A2 | Fulton | 1.65 | 1.47 | 0.17 | 54.64 | 44.24 | 10.40 |
| 11 | A3 | Fulton | 2.61 | 2.45 | 0.16 | 83.28 | 73.43 | 9.85 |
| 12 | A3 | Fulton | 2.66 | 2.40 | 0.26 | 87.59 | 71.96 | 15.63 |
| 13 | A3 | Fulton | 1.70 | 1.50 | 0.20 | 57.08 | 44.88 | 12.20 |
| 14 | A2 | Mokane | 0.85 | 0.79 | 0.06 | 27.37 | 23.81 | 3.56 |
| 15 | A3 | Mokane | 2.15 | 2.00 | 0.15 | 68.96 | 59.99 | 8.97 |
| 16 | A3 | Mokane | 2.21 | 2.07 | 0.14 | 70.55 | 62.16 | 8.40 |

Table 10. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|-----------|----------|-------|---------|----------|-------------|------------------|
| Auxvasse | 1 | A1 | 0.96 | 1 | | 0.96 |
| | 2 | A3 | 2.44 | 2 | 2-3 | 4.87 |
| | 3 | A3 | 2.43 | | | 4.87 |
| Loomfield | 4 | A1 | 1.58 | 3 | | 1.58 |
| | 5 | A2 | 1.81 | 4 | 5-6-5 | 3.38 |
| | 6 | A3 | 1.57 | | | 5.19 |
| Fulton | 7 | A1 | 1.64 | 5 | | 1.64 |
| | 8 | A1 | 1.93 | 6 | | 1.93 |
| | 9 | A1 | 1.99 | 7 | | 1.99 |
| | 10 | A2 | 1.65 | 8 | | 1.65 |
| | 11 | A3 | 2.61 | | | 6.97 |
| | 12 | A3 | 2.66 | 9 | 11-12-13 | 6.97 |
| | 13 | A3 | 1.70 | | | 6.97 |
| Mokane | 14 | A2 | 0.85 | | | 3.00 |
| | 15 | A3 | 2.15 | 10 | 14-15-14-16 | 6.06 |
| | 16 | A3 | 2.21 | | | 6.06 |

4.3 Osage, Gasconade, and Maries Counties

Figure 4.3.1 shows the existing seven depots in Linn, Drake, Vienna, Belle, Owensville, Chamois, and Meta, and service areas in Osage, Gasconade, and Maries Counties.

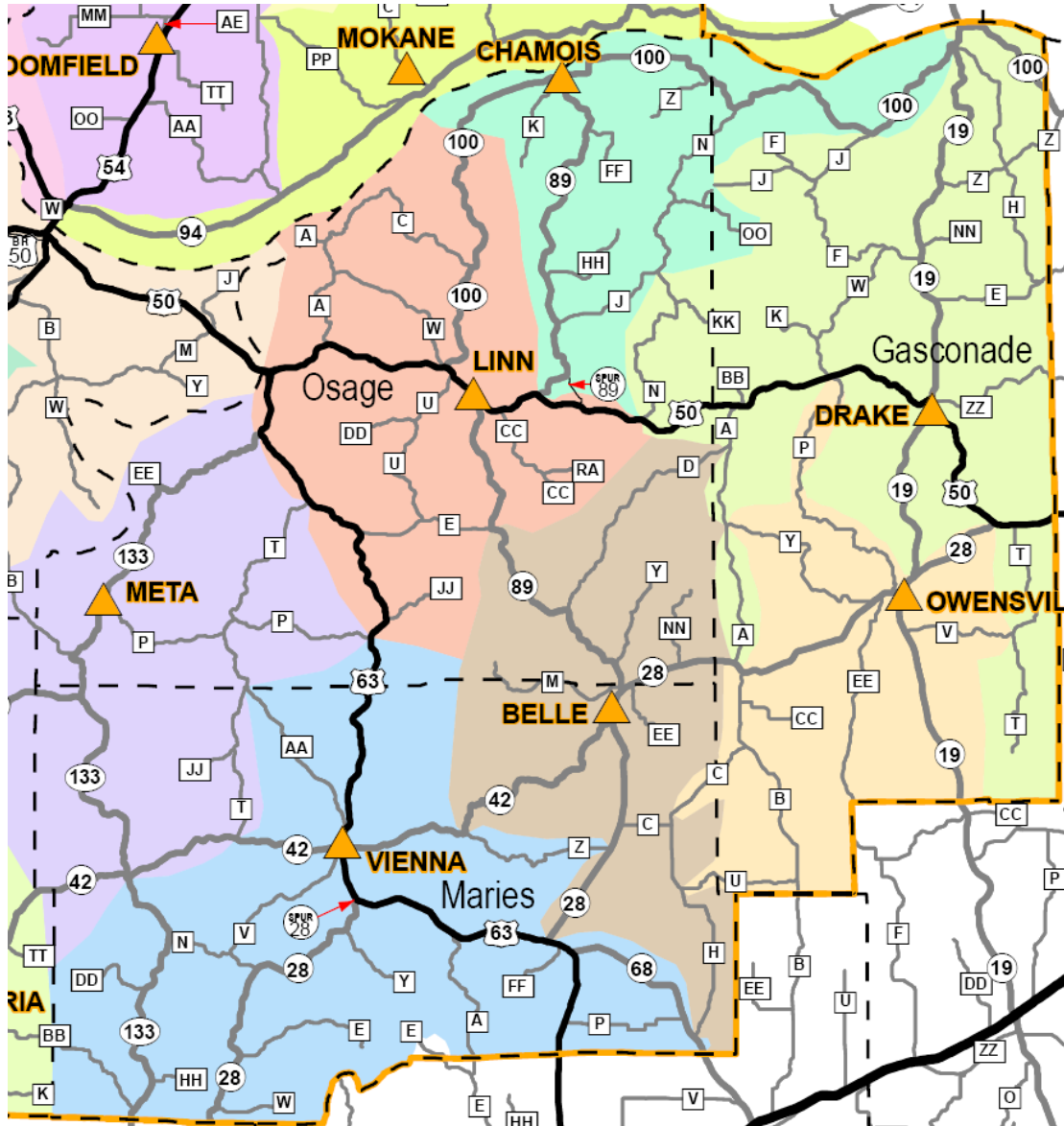


Figure 4.3.1. Osage, Gasconade, and Maries original map

Figure 4.3.2 shows the map of Osage, Gasconade, and Maries Counties with roads distinguished by their classes.

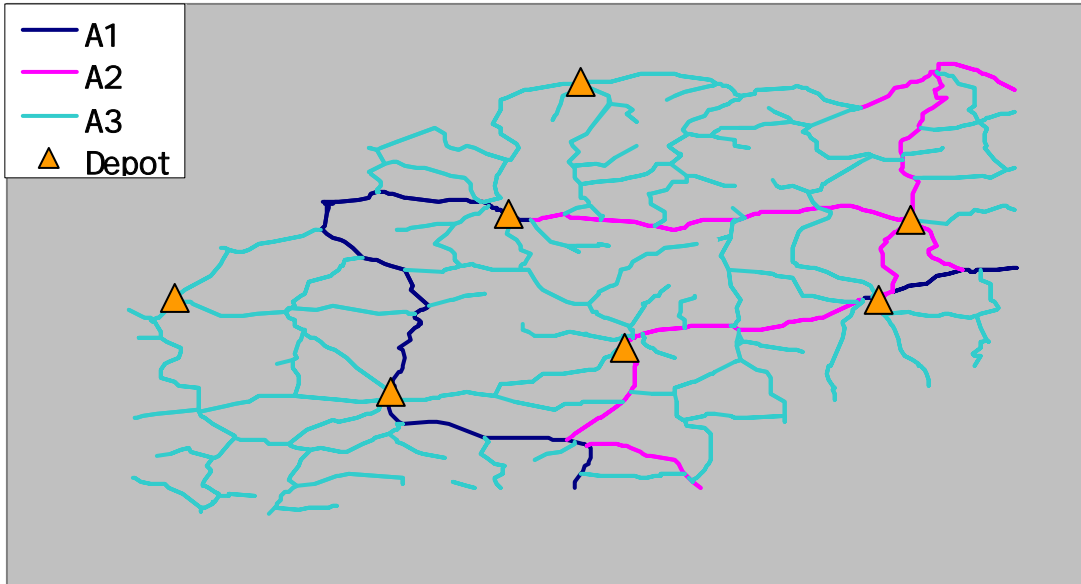


Figure 4.3.2. Osage, Gasconade, and Maries road map

Figures 4.3.3–4.3.8 show the road maps for each class and best routes to serve. The routes assigned to depots in Linn, Drake, Vienna, Belle, Owensville, Chamois, and Meta are denoted by numbers 7, 14, 24, 37, 46, 58 and 72, respectively.

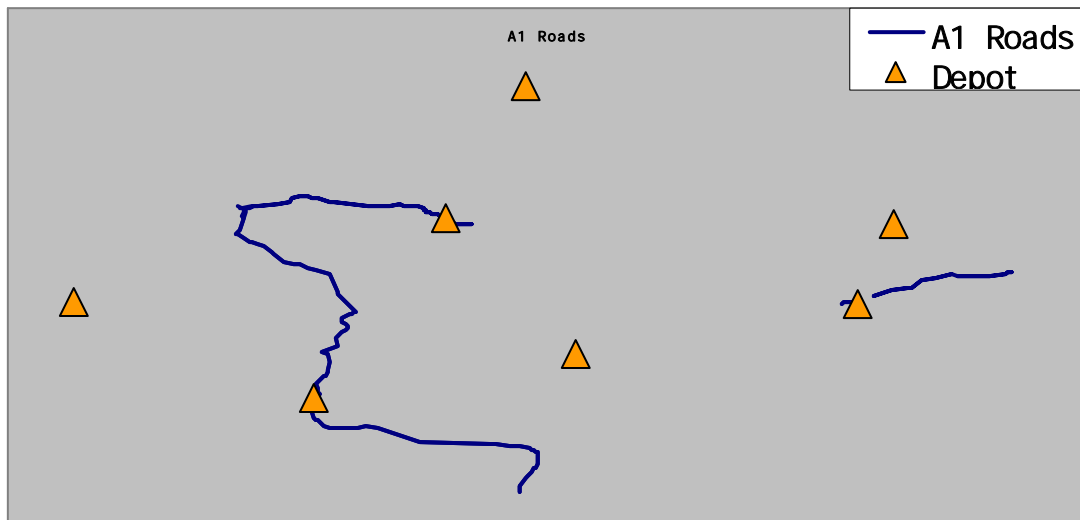


Figure 4.3.3. Class A1 roads

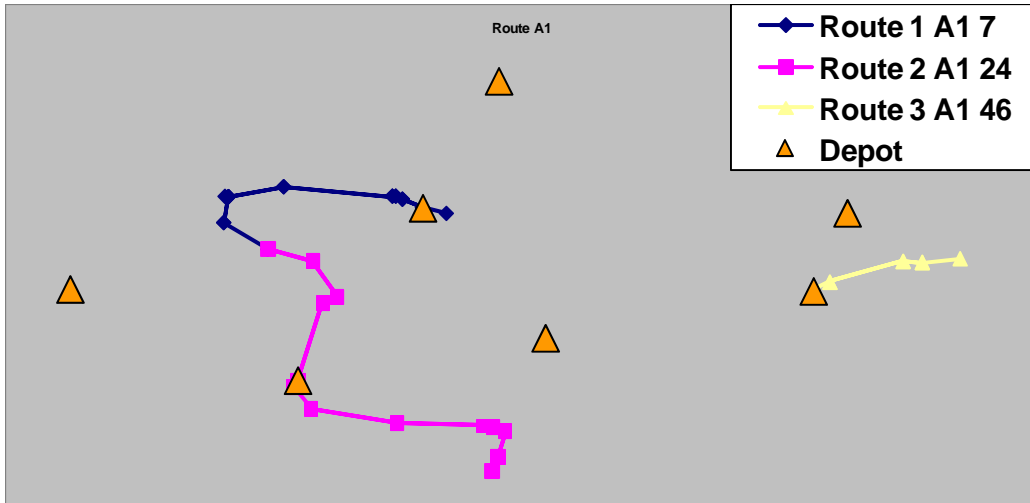


Figure 4.3.4. Class A1 routes

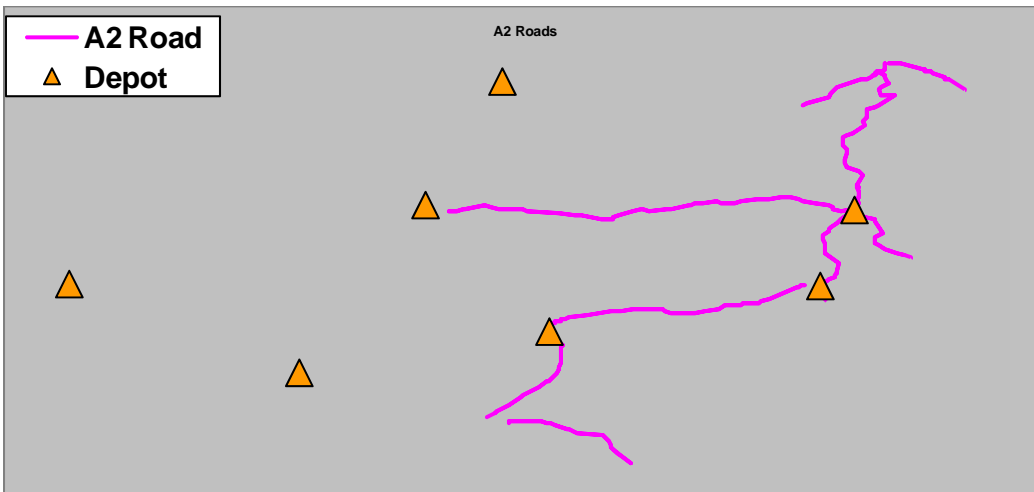


Figure 4.3.5. Class A2 roads

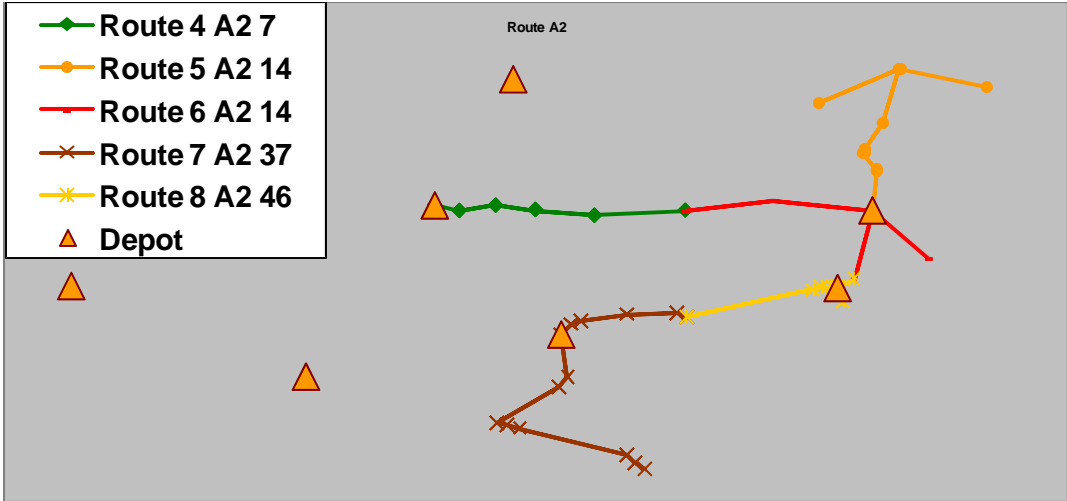


Figure 4.3.6. Class A2 route

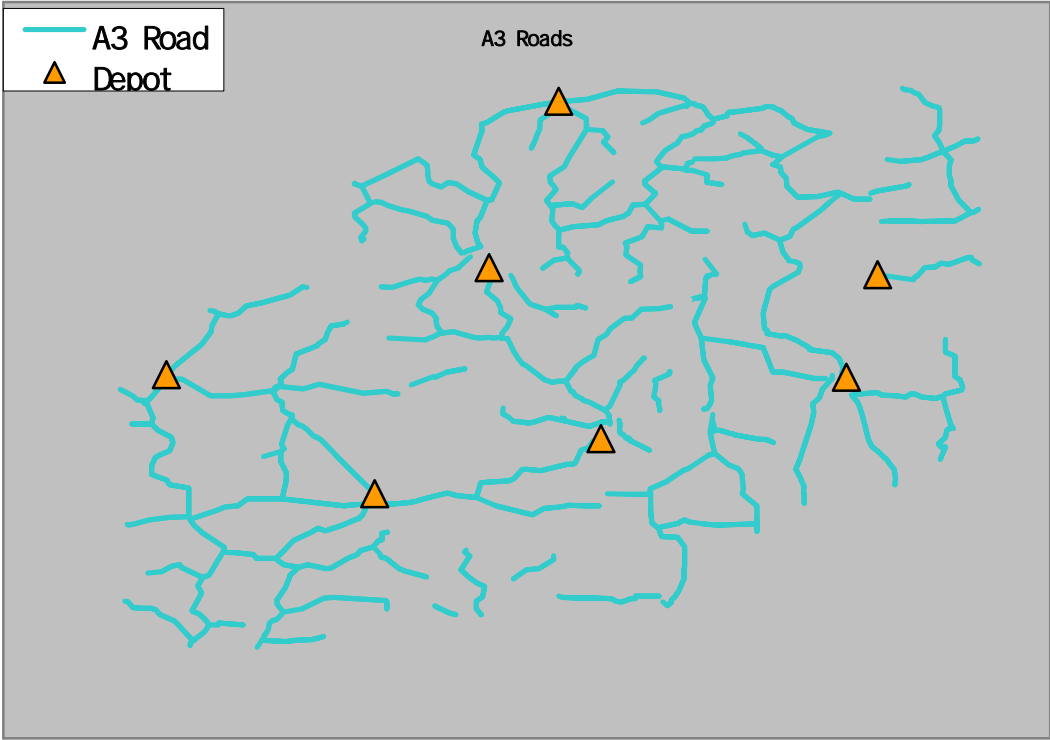


Figure 4.3.7. Class A3 roads

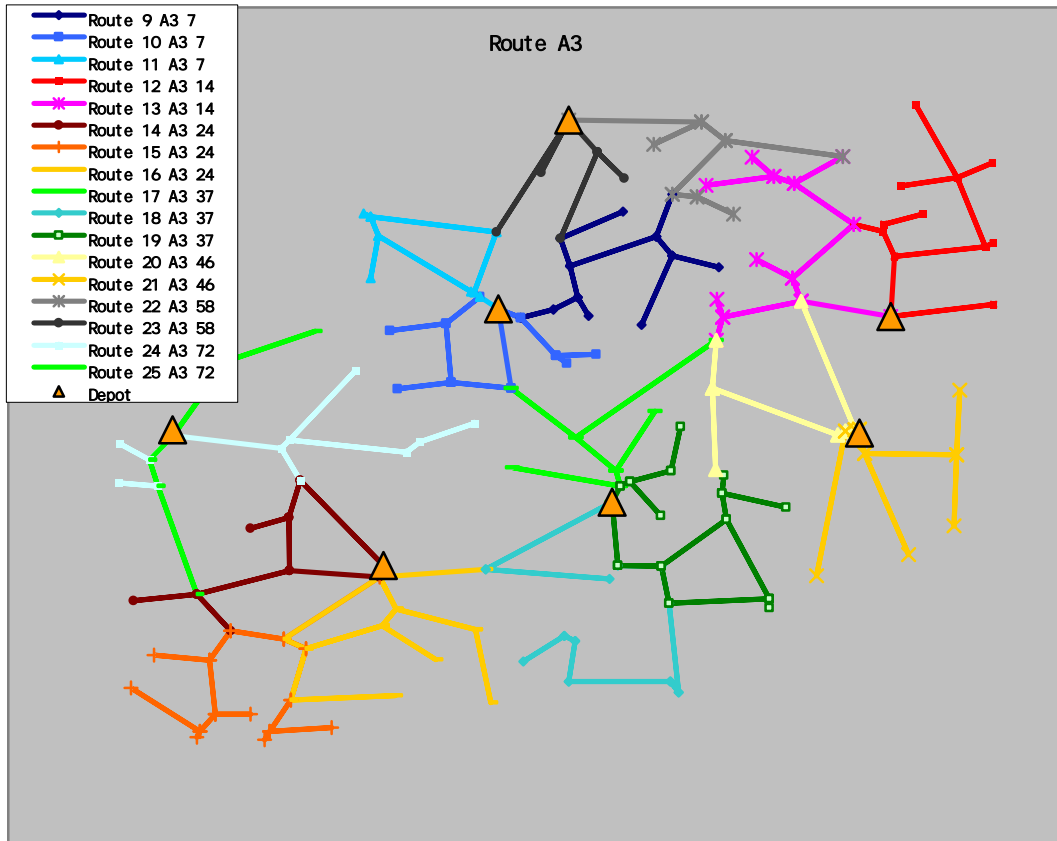


Figure 4.3.8. Class A3 routes

Figure 4.3.9 graphically shows final routes recommended for the three counties, and Table 11 describes these routes.

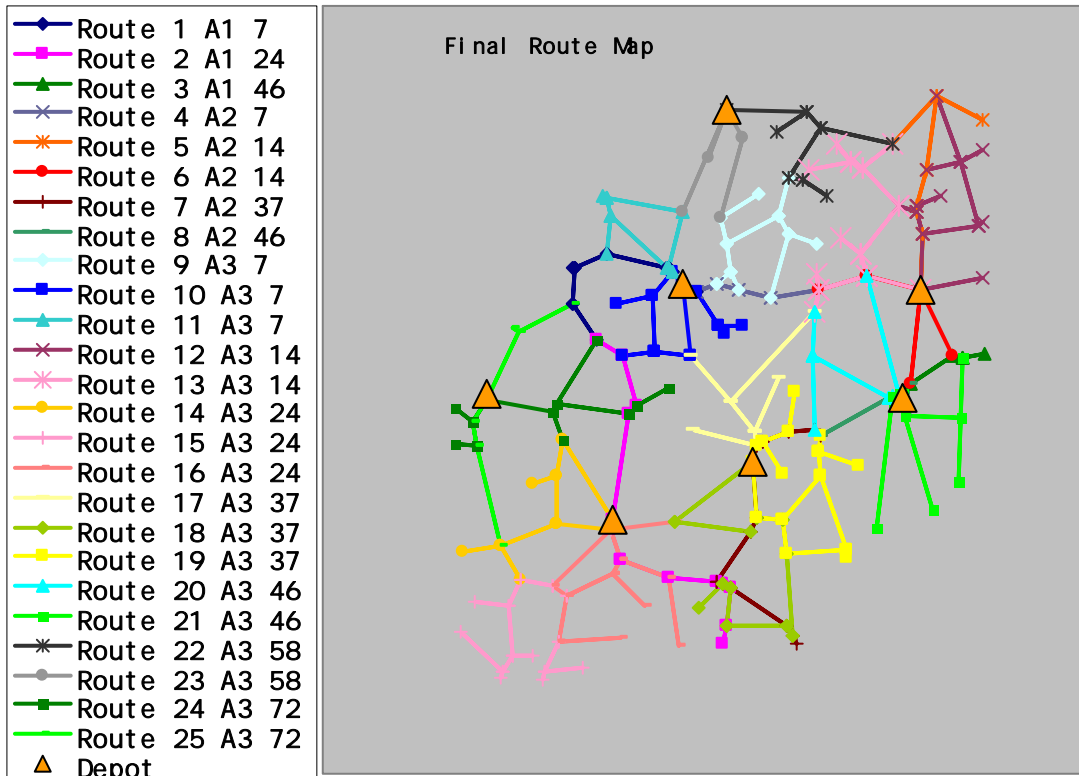


Figure 4.3.9. Final routes

Table 11. Route description

| Route | Class | Depot | Description |
|-------|-------|------------|--|
| 1 | A1 | Linn | 50, from int. with 63 to int. with CC in Osage; 63, from int. with 50 to int. with T in Osage |
| 2 | A1 | Vienna | 63, from int. with T in Osage towards south in Maries |
| 3 | A1 | Owensville | 28, from int. with 19 to int. with 50; 50, from int. with 28 towards east in Gasconade |
| 4 | A2 | Linn | 50, from int. with CC in Osage to int. with A in Gasconade |
| 5 | A2 | Drake | 19, from int. with 50 to int. with 100; 100, from int. with J in Gasconade towards east |
| 6 | A2 | Drake | 50, from int. with A in Gasconade to int. with 28; 19, from int. with 50 to int. with 28 |
| 7 | A2 | Belle | 28, from int. with B in Gasconade to int. with 63; 68, from int. with 63 towards south |
| 8 | A2 | Owensville | 28, from int. with 19 to int. with B in Gasconade; 19, from int with 28, to int. with V in Gasconade |
| 9 | A3 | Linn | 89, from int. with 50 to int. with HH in Osage; SPUR89; HH; J; KK; N from int. with V to int. with 50 in Osage |

Table 11. Route description (continued)

| Route | Class | Depot | Description |
|--------------|--------------|--------------|--|
| 10 | A3 | Linn | CC; RA; U; E; DD in Osage; 89, from int. with 50 to int. with E in Osage |
| 11 | A3 | Linn | A; C; W in Osage; 100, from int. with C to int. with 50 |
| 12 | A3 | Drake | ZZ; E; H; Z; NN in Gasconade; F, from int. with W in Gasconade to int. with 19 |
| 13 | A3 | Drake | J; K; W; F from int. with W towards north in Gasconade; BB; A, from int. with 50 to int. with D in Gasconade |
| 14 | A3 | Vienna | AA; JJ in Maries; T from int. with AA in Maries to int. with 42; 42, from int. with 50 towards west; 133, from int. with 42 to int. with N in Maries |
| 15 | A3 | Vienna | V; DD; HH; BB; N in Maries; 133, from int. with N in Maries towards south; 28, from int. with N towards south; W |
| 16 | A3 | Vienna | E; Y; A in Maries; 28, from int. with N in Maries to int. with 63; 42, from int. with Z in Maries to int. with 63 |
| 17 | A3 | Belle | D; Y in Osage; M in Maries; 89, from int. with E in Osage to int. with 28 |
| 18 | A3 | Belle | 42, from int. with 28 to int. with Z in Maries; Z in Maries; C, from int. with 28 to int. with H in Maries; P; H from int. with U in Maries to int. with 68; FF in Maries; |
| 19 | A3 | Belle | EE; NN in Maries; C, from int with H in Maries to int. with B in Gasconade; H from int. with C to int. with U in Maries; U; B; CC in Gasconade |
| 20 | A3 | Owensville | P; Y; A, from int. with D in Gasconade to int. with 28 |
| 21 | A3 | Owensville | T; V; EE in Gasconade; 19, from int. with V in Gasconade towards south |
| 22 | A3 | Chamois | Z; V; OO in Osage; N from int. with 100 to int. with V in Osage; 100 from int. with J in Gasconade to int. with 89 |
| 23 | A3 | Chamois | K; FF in Osage; 89, from int. with 100 to int. with HH in Osage; 100, from int. with 89 to int. with C in Osage |
| 24 | A3 | Meta | B in Osage; 52 in Maries; T from int. with 63 to int. with AA in Maries; P |
| 25 | A3 | Meta | EE in Osage; 133, from int. with 63 to int. with 42 |

The summary of these routes is given in Table 12. The table shows necessary time and distance to serve each route. As shown in Table 13, these routes can be run by six snow plow trucks.

Table 12. Summary of routes

| Route | Class | Depot | Total time | Service time | DH time | Total distance | Service distance | DH distance |
|--------------|--------------|--------------|-------------------|---------------------|----------------|-----------------------|-------------------------|--------------------|
| 1 | A1 | Linn | 0.95 | 0.95 | 0.00 | 38.07 | 38.07 | 0.00 |
| 2 | A1 | Vienna | 1.80 | 1.80 | 0.00 | 72.19 | 72.19 | 0.00 |
| 3 | A1 | Owensville | 0.40 | 0.36 | 0.04 | 16.60 | 14.32 | 2.27 |
| 4 | A2 | Linn | 0.82 | 0.78 | 0.05 | 26.17 | 23.32 | 2.85 |
| 5 | A2 | Drake | 2.16 | 2.00 | 0.15 | 69.29 | 60.06 | 9.23 |
| 6 | A2 | Drake | 1.70 | 1.70 | 0.00 | 51.12 | 51.12 | 0.00 |
| 7 | A2 | Belle | 1.49 | 1.49 | 0.00 | 36.31 | 36.31 | 0.00 |
| 8 | A2 | Owensville | 0.75 | 0.75 | 0.00 | 22.45 | 22.45 | 0.00 |
| 9 | A3 | Linn | 2.42 | 2.30 | 0.11 | 75.80 | 69.11 | 6.69 |
| 10 | A3 | Linn | 2.15 | 2.11 | 0.05 | 66.05 | 63.20 | 2.85 |
| 11 | A3 | Linn | 2.05 | 1.99 | 0.06 | 63.28 | 59.59 | 3.69 |
| 12 | A3 | Drake | 2.56 | 2.32 | 0.24 | 84.22 | 69.60 | 14.62 |
| 13 | A3 | Drake | 2.45 | 2.11 | 0.33 | 83.44 | 63.40 | 20.04 |
| 14 | A3 | Vienna | 2.36 | 2.36 | 0.00 | 70.84 | 70.84 | 0.00 |
| 15 | A3 | Vienna | 2.95 | 2.95 | 0.00 | 88.36 | 88.36 | 0.00 |
| 16 | A3 | Vienna | 2.70 | 2.00 | 0.69 | 101.79 | 60.10 | 41.69 |
| 17 | A3 | Belle | 2.35 | 2.31 | 0.05 | 71.96 | 69.21 | 2.75 |
| 18 | A3 | Belle | 2.85 | 2.40 | 0.46 | 99.23 | 71.91 | 27.33 |
| 19 | A3 | Belle | 2.84 | 2.50 | 0.34 | 95.30 | 74.92 | 20.38 |
| 20 | A3 | Owensville | 2.35 | 2.31 | 0.04 | 71.88 | 69.41 | 2.47 |
| 21 | A3 | Owensville | 2.58 | 2.50 | 0.09 | 80.07 | 74.89 | 5.18 |
| 22 | A3 | Chamois | 2.05 | 2.05 | 0.00 | 61.55 | 61.55 | 0.00 |
| 23 | A3 | Chamois | 1.82 | 1.82 | 0.00 | 54.57 | 54.57 | 0.00 |
| 24 | A3 | Meta | 2.32 | 2.13 | 0.19 | 75.46 | 63.89 | 11.56 |
| 25 | A3 | Meta | 2.10 | 2.10 | 0.00 | 63.06 | 63.06 | 0.00 |

Table 13. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|------------|----------|-------|---------|----------|-------------|------------------|
| Linn | 1 | A1 | 0.95 | 1 | 1-4 | 1.77 |
| | 4 | A2 | 0.82 | | | 1.77 |
| | 9 | A3 | 2.42 | 2 | 9-10-11 | 6.62 |
| | 10 | A3 | 2.15 | | | 6.62 |
| | 11 | A3 | 2.05 | | | 6.62 |
| Drake | 5 | A2 | 2.16 | 3 | 5-6 | 3.86 |
| | 6 | A2 | 1.70 | | | 3.86 |
| | 12 | A3 | 2.56 | 4 | 12-13 | 5.01 |
| | 13 | A3 | 2.45 | | | 5.01 |
| Vienna | 2 | A1 | 1.80 | 5 | | 1.80 |
| | 14 | A3 | 2.36 | 6 | 14-15-16 | 8.01 |
| | 15 | A3 | 2.95 | | | 8.01 |
| | 16 | A3 | 2.70 | | | 8.01 |
| | | | | | | |
| Belle | 7 | A2 | 1.49 | 7 | | 1.49 |
| | 17 | A3 | 2.35 | 8 | 17-18-19 | 8.03 |
| | 18 | A3 | 2.85 | | | 8.03 |
| | 19 | A3 | 2.83 | | | 8.03 |
| | | | | | | |
| Owensville | 3 | A1 | 0.40 | 9 | 3-8 | 1.15 |
| | 8 | A2 | 0.75 | | | 1.15 |
| | 20 | A3 | 2.35 | 10 | 20-21 | 4.93 |
| | 21 | A3 | 2.58 | | | 4.93 |
| Chamois | 22 | A3 | 2.05 | 11 | 22-23 | 3.87 |
| | 23 | A3 | 1.82 | | | 3.87 |
| Meta | 24 | A3 | 2.32 | 12 | 24-25 | 4.42 |
| | 25 | A3 | 2.10 | | | 4.42 |

4.4 Cooper and Moniteau Counties

Figure 4.4.1 shows the existing five depots in Blackwater, Boonville, Jamestown, California, and Tipton, and service areas in Cooper and Moniteau counties.

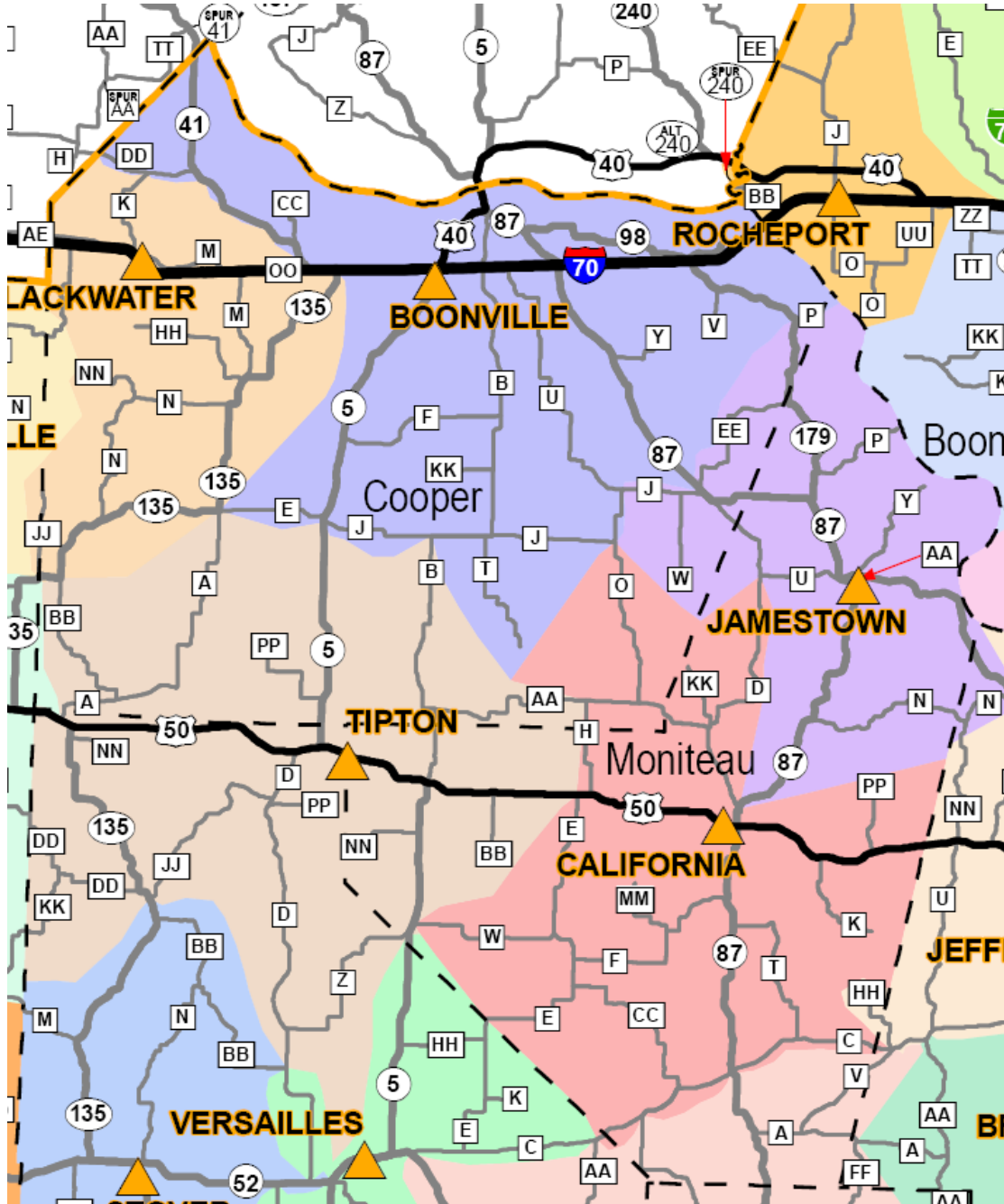


Figure 4.4.1. Cooper and Moniteau original map

Figure 4.4.2 shows the simplified map of Cooper and Moniteau Counties with roads distinguished by their classes.

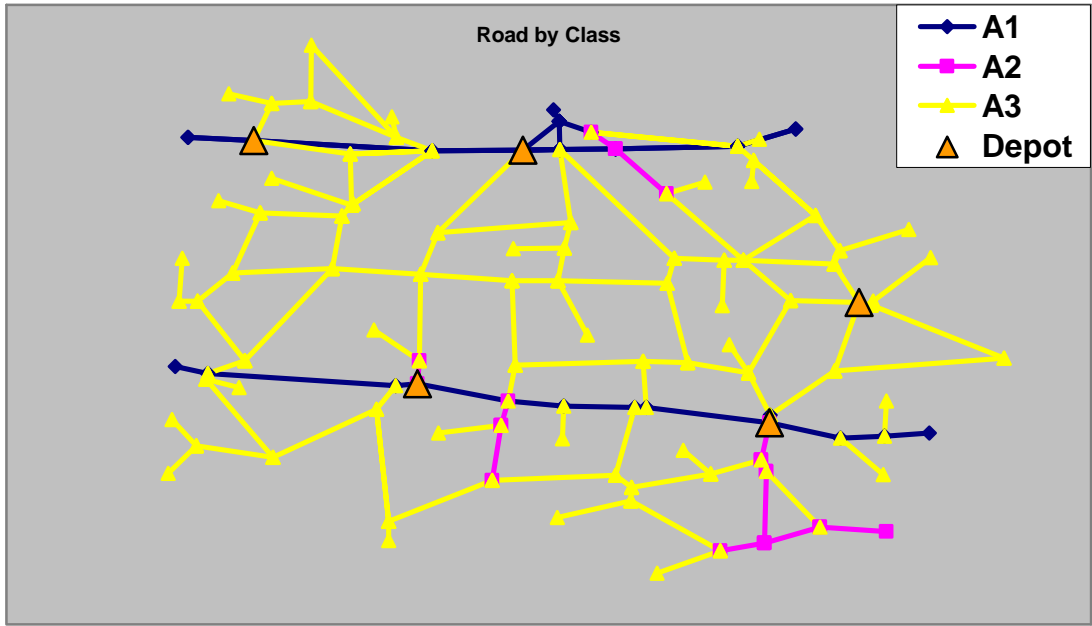


Figure 4.4.2. Simplified map

Figures 4.4.3–4.4.8 show the road maps for each class and best routes to serve. Because Class A2 roads are mostly scattered around, roads in Classes A1 and A2 are analyzed together. The routes assigned to depots in Blackwater, Boonville, Jamestown, California, and Tipton are denoted by numbers 92, 61, 11, 21 and 73, respectively.

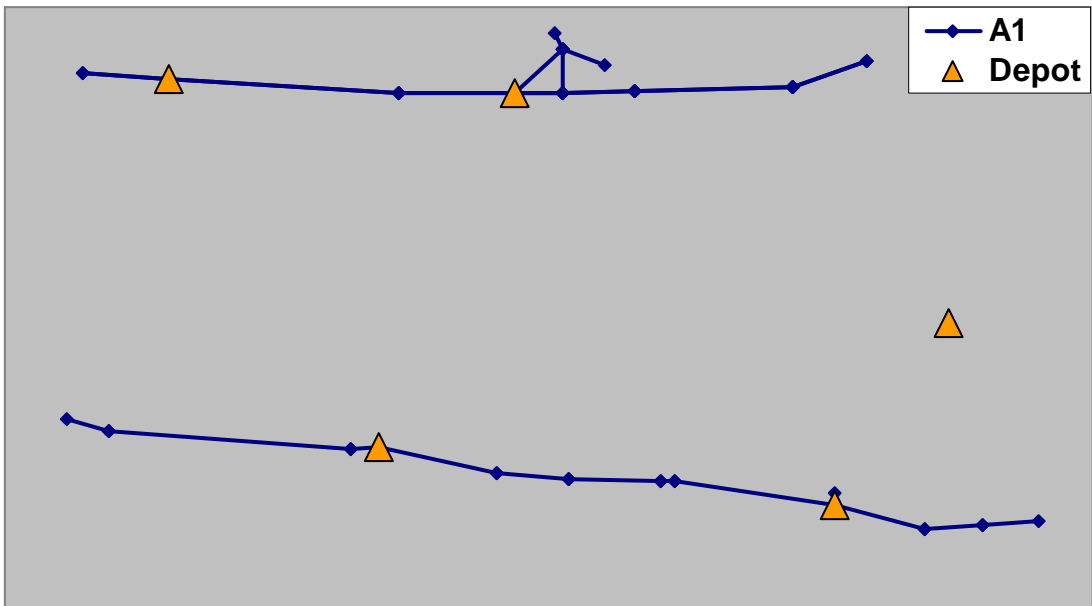


Figure 4.4.3. Class A1 roads

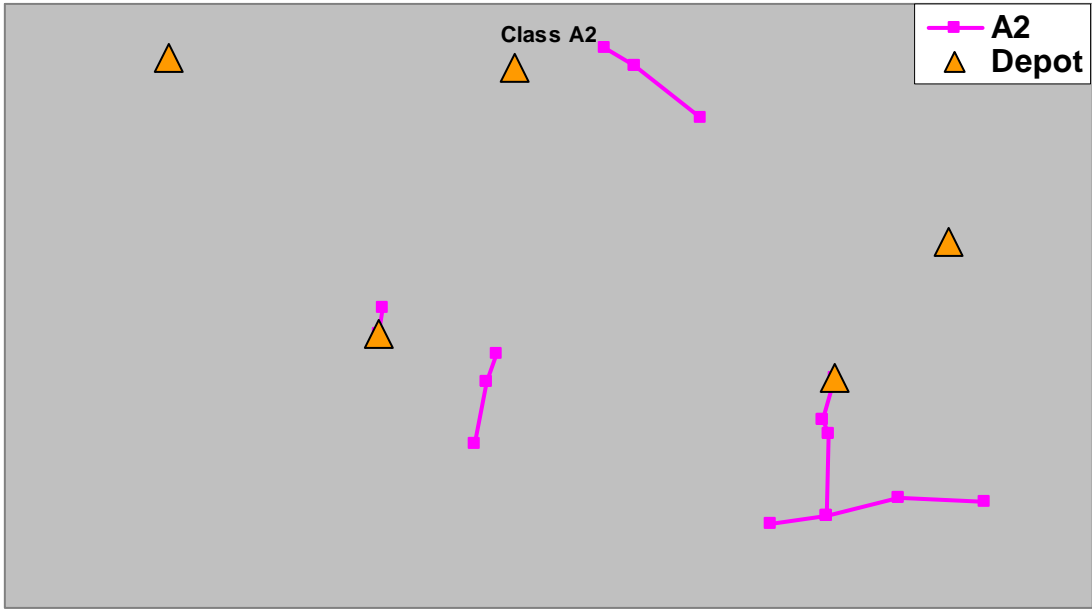


Figure 4.4.4. Class A2 roads

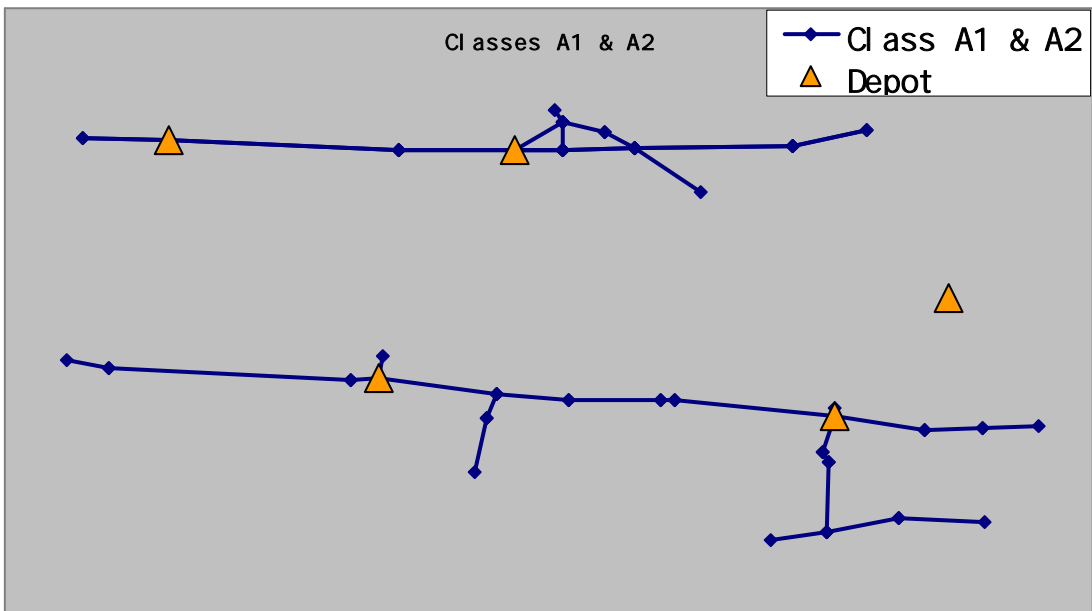


Figure 4.4.5. Class A1 and A2 roads

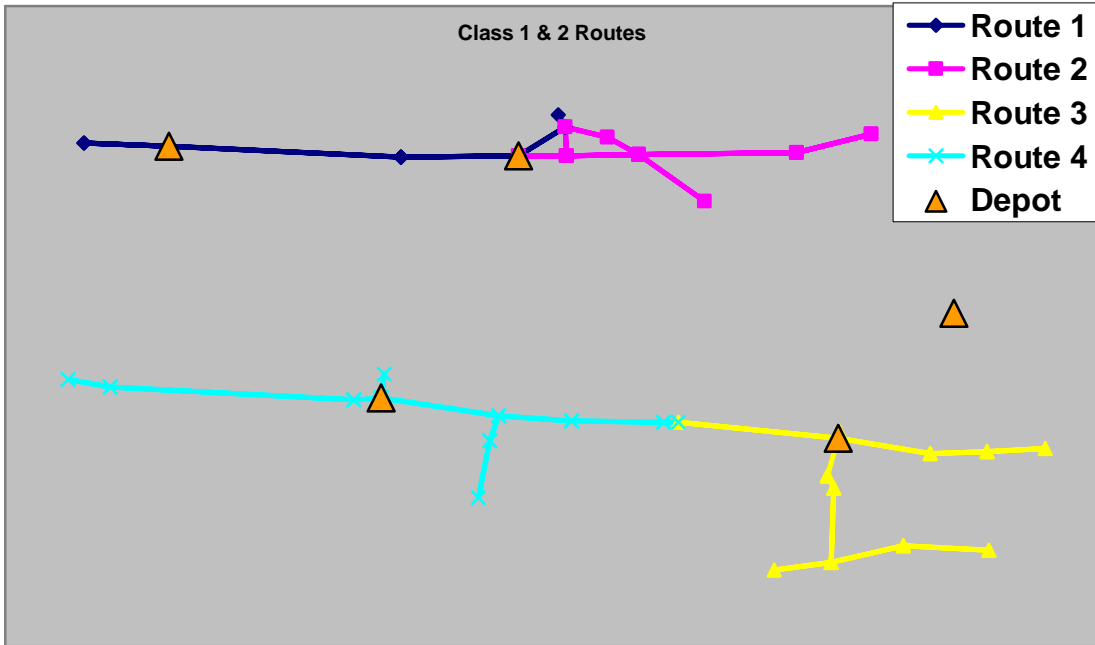


Figure 4.4.6. Class A1 and A2 routes

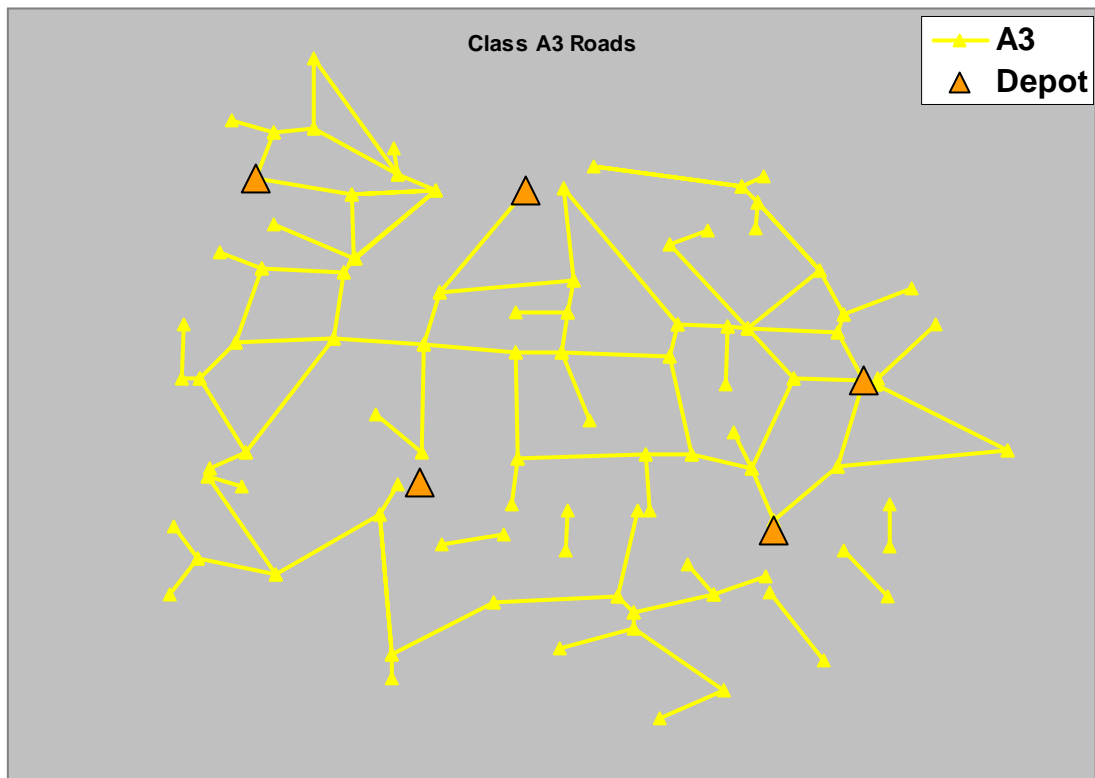


Figure. 4.4.7 Class A3 roads

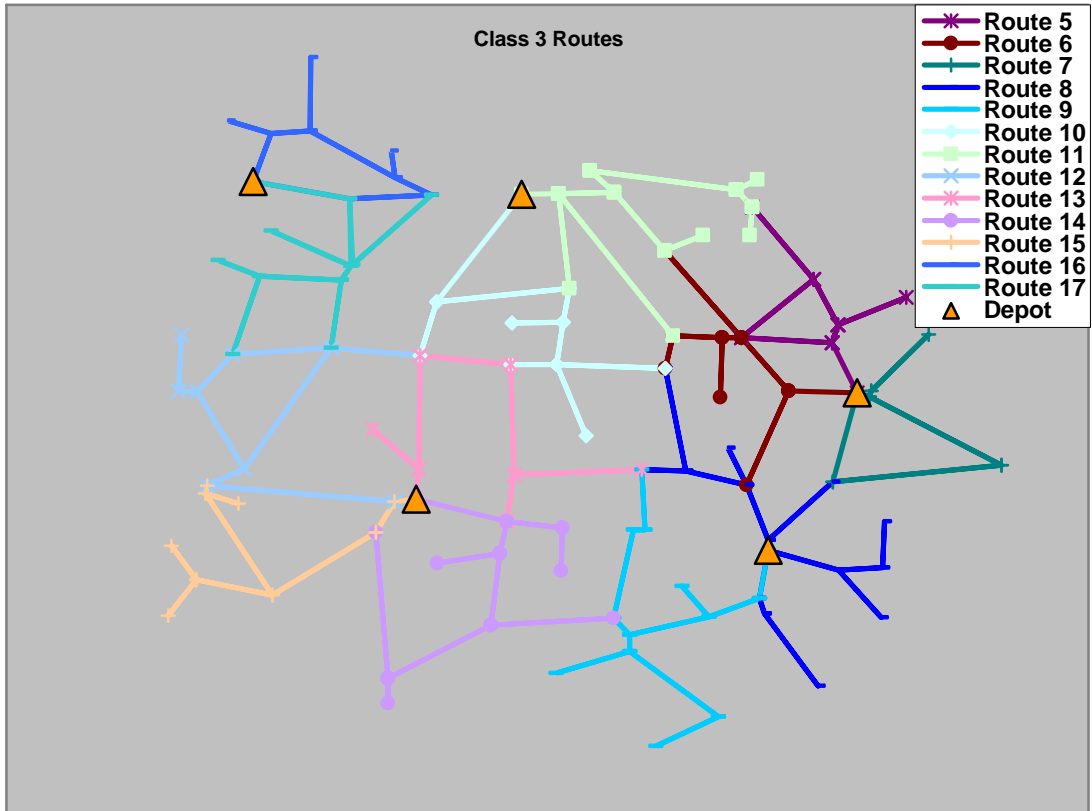


Figure 4.4.8. Class A3 routes

Figure 4.4.9 graphically shows service regions for the five depots, and Figure 4.4.10 graphically shows final routes recommended for the two counties. Table 14 describes all the routes.

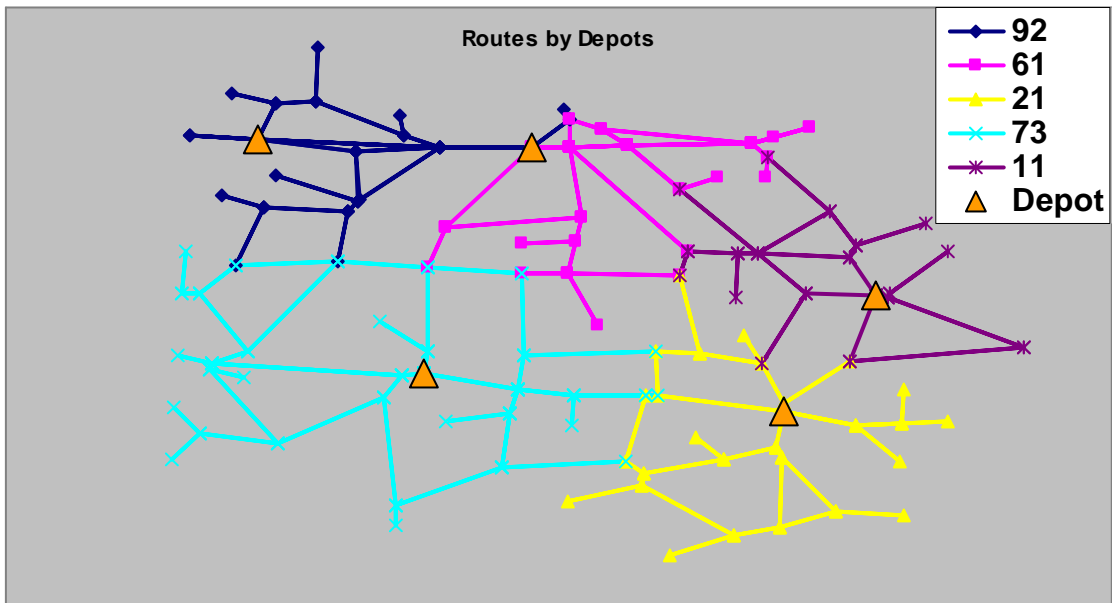


Figure 4.4.9. Service regions by depots

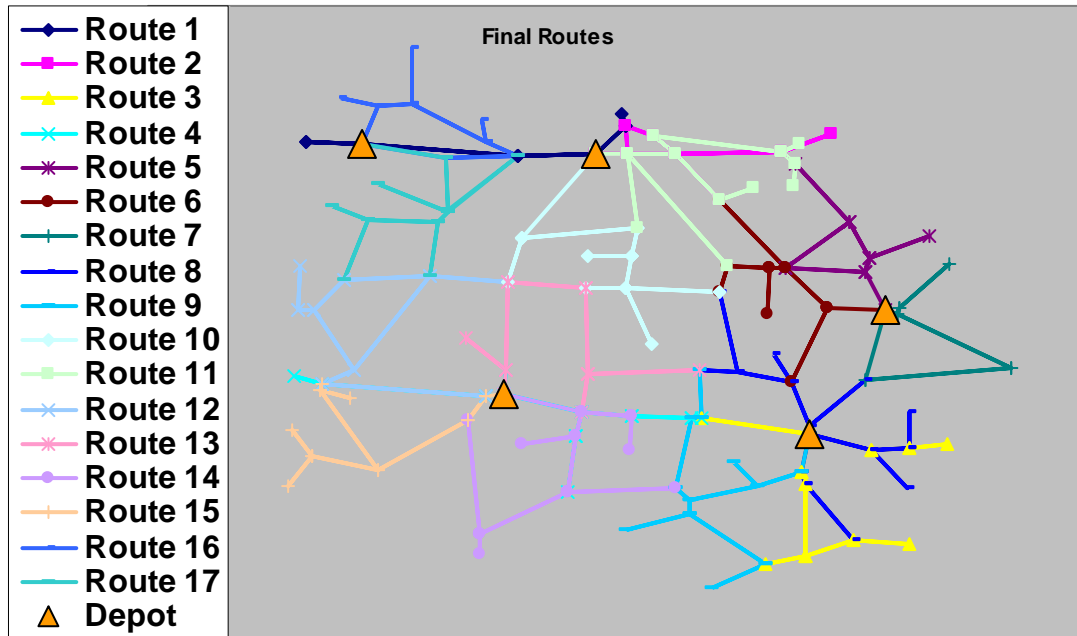


Figure 4.4.10. Final routes

Table 14. Route description

| Route # | Depot | Class | Description |
|---------|------------|-------|--|
| 1 | Blackwater | A1&A2 | 70 from west end to int. with 5, and 40 |
| 2 | Boonville | A1&A2 | 70 from int. with 5 to east end, B from int. with 70 to north end, 87 from int. with 40 to int. with Y |
| 3 | California | A1&A2 | 50 from east end to int. with H, 87 from int. with KK to int. with C, and C |
| 4 | Tipton | A1&A2 | 50 from int. with H to west end, 5 from int. with 50 to int. with Z |
| 5 | Jamestown | A3 | 87 and 179 from Jamestown depot to int. with EE, P, and EE |
| 6 | Jamestown | A3 | U, D, J W, and 87 from int. with J to int. with Y(Boonville) |
| 7 | Jamestown | A3 | 179 from Jamestown depot to int. with N, Y(Jamestown), AA, N, 87 from int. with Jamestown depot to int. with N |
| 8 | California | A3 | PP, K, 87 from int. with 50 to int. with O, KK, T, H from int. with AA to int. with O |
| 9 | California | A3 | F, MM, E, C, CC, H from int. with 50 to int. with AA |
| 10 | Boonville | A3 | 5 from int. with 70 to int. with J, F, KK, T, J from int. with T to int. with B, B from int. with F to int. with T |
| 11 | Boonville | A3 | B from int. with 50 to int. with F, U, Y, 98, V, 179 from int. with 98 to int. with V |
| 12 | Tipton | A3 | 50 - A - BB - A - 135 - JJ - 135 - E - A - 135 - 50 |
| 13 | Tipton | A3 | 5(N) - PP - 5 - J - B - AA - B - J - 5 |
| 14 | Tipton | A3 | BB, NN, W, Z, D |
| 15 | Tipton | A3 | D, JJ, DD, KK, NN |
| 16 | Blackwater | A3 | M from int. with Blackwater depot to int. with OO, OO, 41, CC, K, DD |
| 17 | Blackwater | A3 | M from int. with OO to int. with 135, 135 from int. with 70 to int. with A, N, NN, HH |

The summary of these routes is given in Table 15. The table shows necessary time and distance to serve each route. As shown in Table 16, these routes can be run by nine snow plow trucks.

Table 15. Summary of routes

| Route # | Depot | Class | Total DH time | Total service time | Total cycle time | Total DH distance | Total service distance | Total cycle distance |
|---------|------------|-------|---------------|--------------------|------------------|-------------------|------------------------|----------------------|
| 1 | Blackwater | A1&A2 | 0.00 | 1.75 | 1.75 | 0.00 | 70.08 | 70.08 |
| 2 | Boonville | A1&A2 | 0.00 | 1.87 | 1.87 | 0.00 | 70.78 | 70.78 |
| 3 | California | A1&A2 | 0.00 | 1.91 | 1.91 | 0.00 | 64.39 | 64.39 |
| 4 | Tipton | A1&A2 | 0.00 | 1.65 | 1.65 | 0.00 | 60.62 | 60.62 |
| 5 | Jamestown | A3 | 0.00 | 1.94 | 1.94 | 0.00 | 58.19 | 58.19 |
| 6 | Jamestown | A3 | 0.00 | 2.01 | 2.01 | 0.00 | 60.42 | 60.42 |
| 7 | Jamestown | A3 | 0.00 | 2.10 | 2.10 | 0.00 | 62.98 | 62.98 |
| 8 | California | A3 | 0.46 | 2.43 | 2.89 | 20.86 | 72.85 | 93.71 |
| 9 | California | A3 | 0.18 | 2.53 | 2.70 | 7.24 | 75.83 | 83.07 |
| 10 | Boonville | A3 | 0.00 | 2.46 | 2.46 | 0.00 | 73.83 | 73.83 |
| 11 | Boonville | A3 | 0.47 | 2.17 | 2.64 | 20.58 | 65.00 | 85.58 |
| 12 | Tipton | A3 | 0.39 | 2.26 | 2.65 | 19.49 | 67.70 | 87.20 |
| 13 | Tipton | A3 | 0.09 | 2.17 | 2.27 | 3.62 | 65.23 | 68.86 |
| 14 | Tipton | A3 | 0.60 | 1.99 | 2.60 | 26.94 | 59.83 | 86.77 |
| 15 | Tipton | A3 | 0.04 | 1.95 | 1.99 | 1.97 | 58.42 | 60.39 |
| 16 | Blackwater | A3 | 0.00 | 2.01 | 2.01 | 0.00 | 60.35 | 60.35 |
| 17 | Blackwater | A3 | 0.24 | 2.27 | 2.51 | 9.57 | 68.03 | 77.60 |

Table 16. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|------------|----------|---------|---------|----------|-------------|------------------|
| Blackwater | 1 | A1 & A2 | 1.75 | 1 | | 1.75 |
| | 16 | A3 | 2.01 | 2 | 16-17 | 4.52 |
| | 17 | A3 | 2.51 | | | 4.52 |
| Boonville | 2 | A1 & A2 | 1.87 | 3 | | 1.75 |
| | 10 | A3 | 2.46 | 4 | 10-11 | 5.10 |
| | 11 | A3 | 2.64 | | | 5.10 |
| Jamestown | 5 | A3 | 1.94 | | | 6.05 |
| | 6 | A3 | 2.01 | 5 | 5-6-7 | 6.05 |
| | 7 | A3 | 2.10 | | | 6.05 |
| California | 3 | A1 & A2 | 1.91 | 6 | | 1.91 |
| | 8 | A3 | 2.89 | 7 | 8-9 | 5.59 |
| | 9 | A3 | 2.70 | | | 5.59 |
| Tipton | 4 | A1 & A2 | 1.65 | 8 | | 1.65 |
| | 12 | A3 | 2.65 | | | 9.51 |
| | 13 | A3 | 2.27 | 9 | 12-13-14-15 | 9.51 |
| | 14 | A3 | 2.60 | | | 9.51 |
| | 15 | A3 | 1.99 | | | 9.51 |

4.5 Benton and Pettis Counties

Figure 4.5.1 shows the existing six depots in Hughesville, Sedalia, Lamonte, Cole Camp, Lincoln, and Warsaw in Benton and Pettis counties.

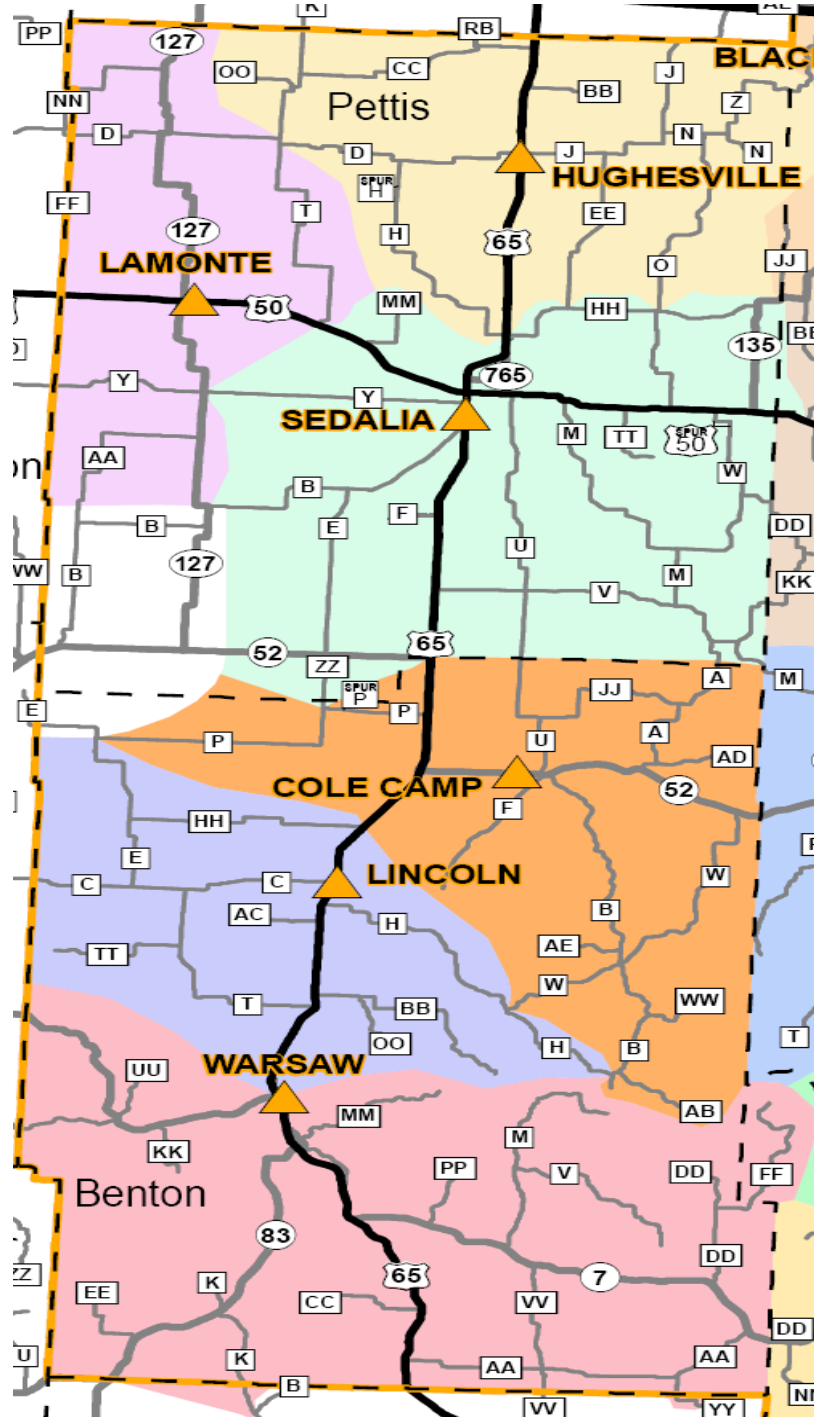


Figure 4.5.1. Benton and Pettis original map

Figure 4.5.2 shows the map of Benton and Pettis Counties with roads distinguished by their classes.

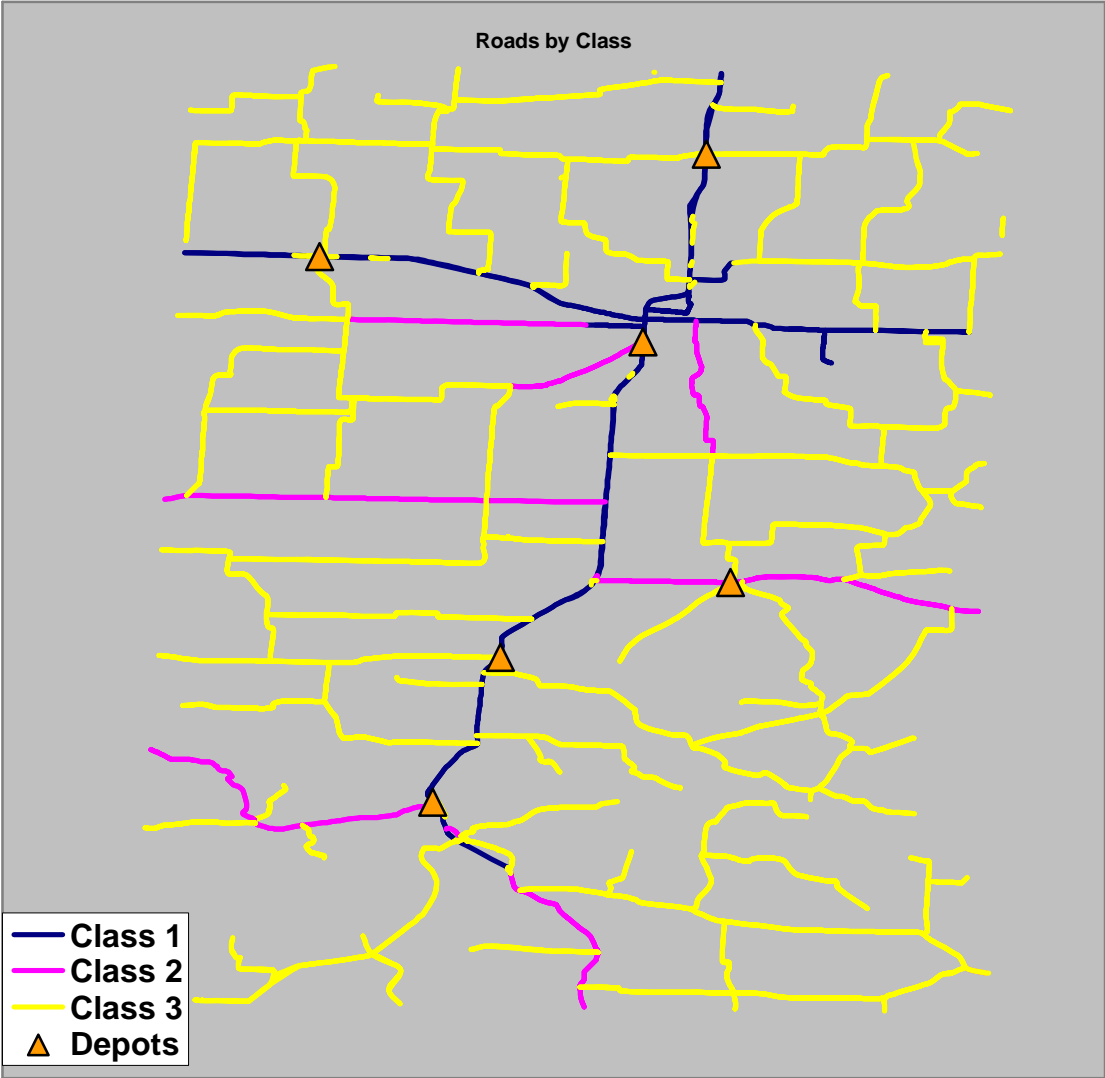


Figure 4.5.2. Benton and Pettis road map

Figures 4.5.3–4.5.8 show the road maps for each class and best routes to serve. The routes assigned to depots are in Hughesville, Sedalia, Lamonte, Cole Camp, Lincoln, and Warsaw.

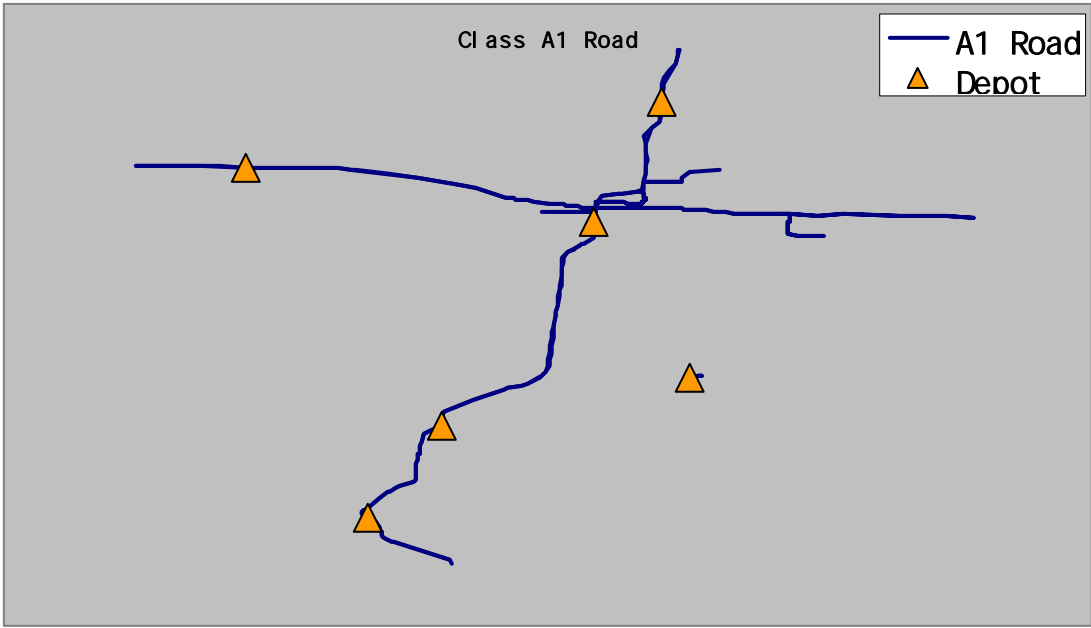


Figure 4.5.3. Class A1 roads

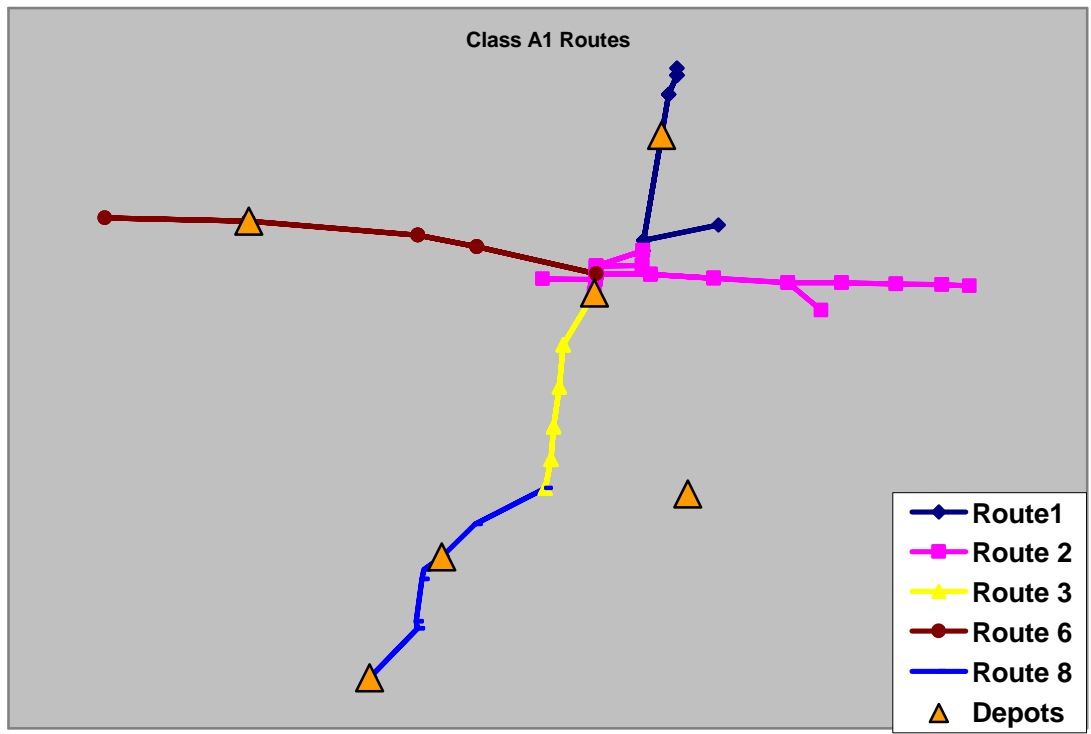


Figure 4.5.4. Class A1 routes

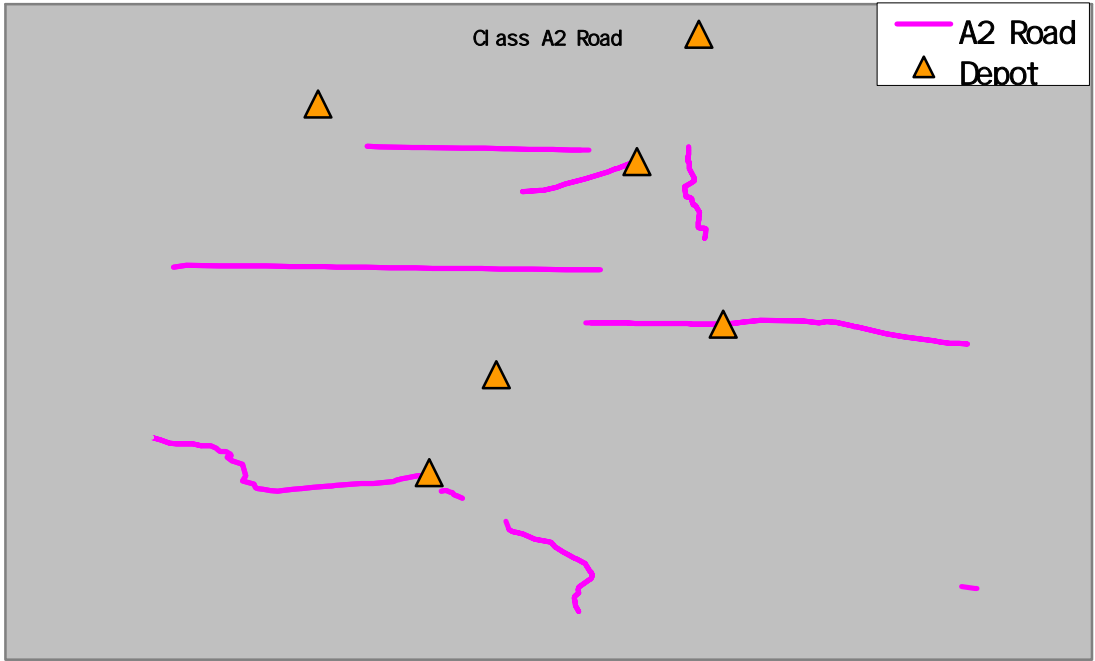


Figure 4.5.5. Class A2 roads



Figure 4.5.6. Class A2 routes

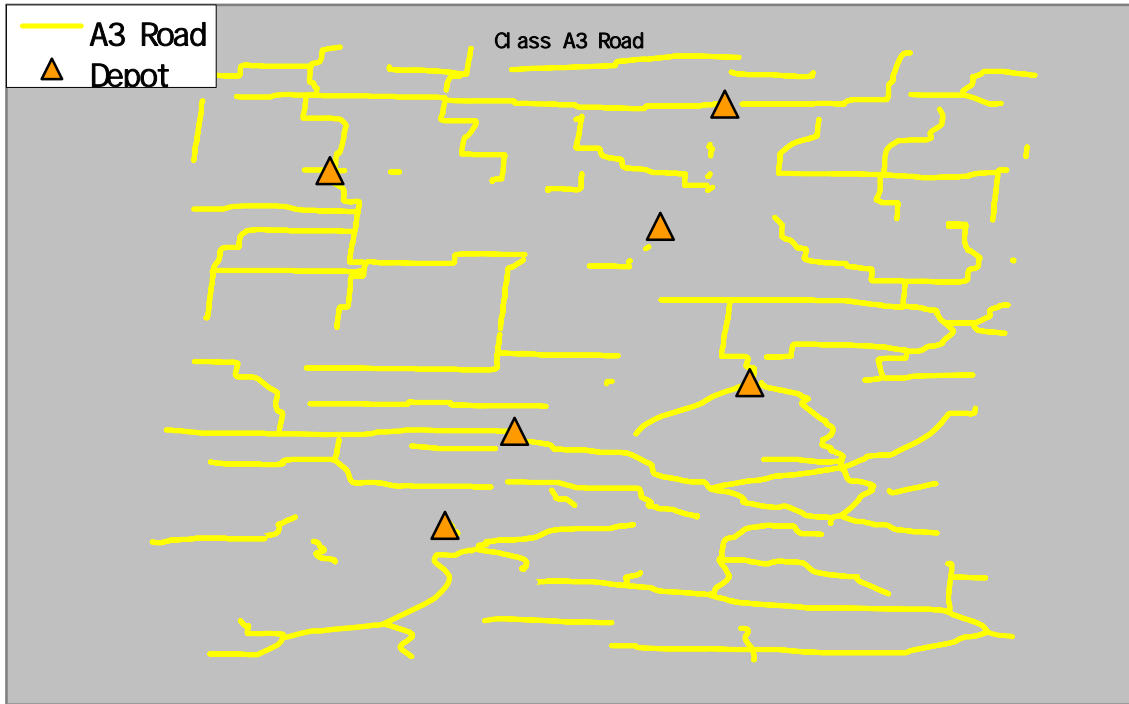


Figure 4.5.7. Class A3 roads

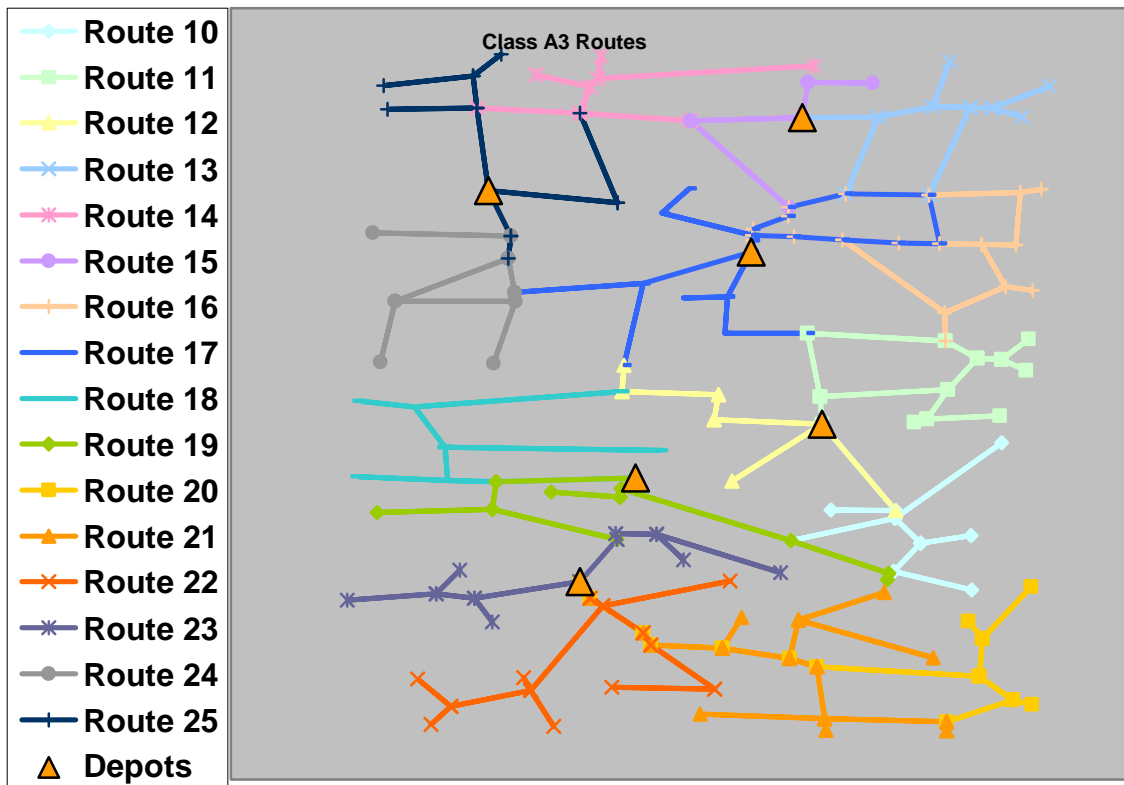


Figure 4.5.8. Class A3 routes

Figure 4.5.9 graphically shows the service regions for the six depots, and Figure 4.5.10 graphically shows final routes recommended for the two counties. Table 17 describes all the routes.

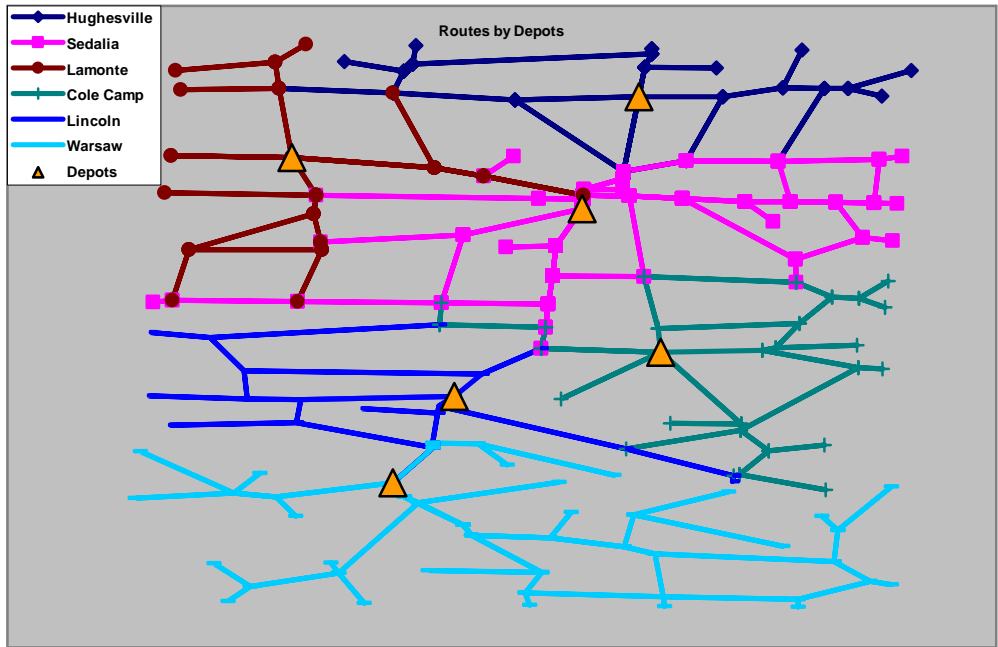


Figure 4.5.9. Service regions by depots

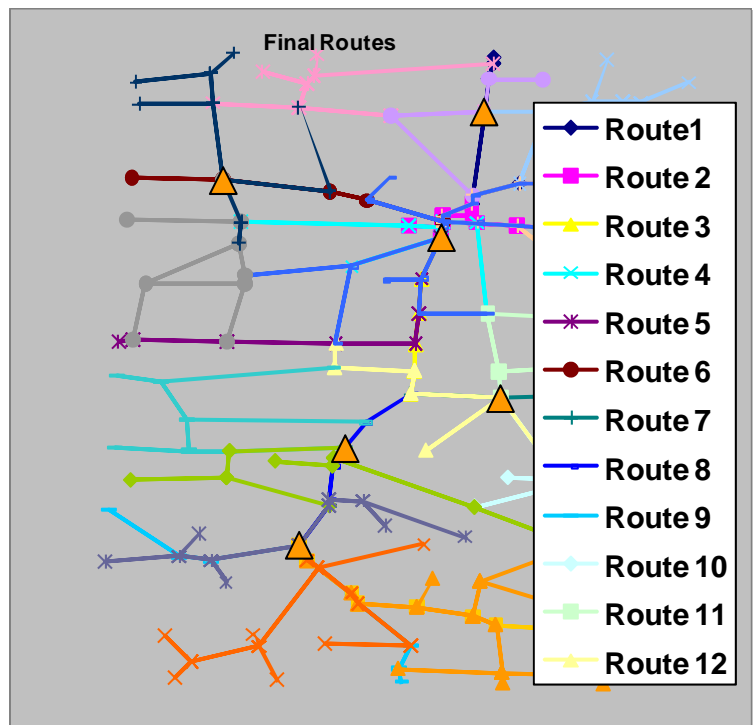


Figure 4.5.10. Final routes

Table 17. Route description

| Route # | Depot | Class | Description |
|----------------|--------------|--------------|--|
| 1 | Hughesville | Class 1 | 65 from north to int. with 765, HH from int. with 65 to int. with EE |
| 2 | Sedalia | Class 1 | 765, 50 (east to 65) |
| 3 | Sedalia | Class 1 | 65 from depot Sedalia to int. with 52 |
| 4 | Sedalia | Class 2 | Y from int. with 65 to int. with 127, B from int. with 65 to int. with, U from int. with 50 to int. with V |
| 5 | Sedalia | Class 2 | 52 (west to 65) |
| 6 | Lamonte | Class 1 | 50 (west to 65) |
| 7 | Cole Camp | Class 2 | 52 (east to 65) |
| 8 | Lincoln | Class 1 | 65 from int. with 52 to depot Warsaw |
| 9 | Warsaw | Class 1 & 2 | 65 from depot Warsaw to south, 7 (west to 65) |
| 10 | Cole Camp | Class 3 | B, W, WW, AB, AE |
| 11 | Cole Camp | Class 3 | U, JJ, A, M, KK, AD, V |
| 12 | Cole Camp | Class 3 | F, P, ZZ, B |
| 13 | Hughesville | Class 3 | J, N, Z, O, EE |
| 14 | Hughesville | Class 3 | D, K, CC, OO |
| 15 | Hughesville | Class 3 | D, H, BB |
| 16 | Sedalia | Class 3 | M, W, DD, 135, HH, O |
| 17 | Sedalia | Class 3 | O, MM, V, F, E, B |
| 18 | Lincoln | Class 3 | C, E, HH, P |
| 19 | Lincoln | Class 3 | H, B, AC, C, T, TT |
| 20 | Warsaw | Class 3 | 7 (east to 65), AA, DD, FF |
| 21 | Warsaw | Class 3 | VV, AA, YY, M, V, PP |
| 22 | Warsaw | Class 3 | 83, MM, CC, K, EE |
| 23 | Warsaw | Class 3 | KK, UU, Z, BB, OO |
| 24 | Lamonte | Class 3 | 127, B, AA, Y |
| 25 | Lamonte | Class 3 | 127, NN, D, T |

The summary of these routes is given in Table 18. The table shows necessary time and distance to serve each route. As shown in Table 19, these routes can be run by twelve snow plow trucks.

Table 18. Summary of routes

| Route # | Depot | Class | Total DH time | Total service time | Total cycle time | Total DH distance | Total service distance | Total cycle distance |
|----------------|--------------|--------------|----------------------|---------------------------|-------------------------|--------------------------|-------------------------------|-----------------------------|
| 1 | Hughesville | Class 1 | 0.00 | 1.56 | 1.56 | 0.00 | 62.44 | 62.44 |
| 2 | Sedalia | Class 1 | 0.00 | 1.44 | 1.44 | 0.00 | 57.65 | 57.65 |
| 3 | Sedalia | Class 1 | 0.00 | 1.52 | 1.52 | 0.00 | 60.72 | 60.72 |
| 4 | Sedalia | Class 2 | 0.18 | 1.44 | 1.62 | 8.85 | 43.20 | 52.05 |
| 5 | Sedalia | Class 2 | 0.42 | 0.87 | 1.29 | 20.87 | 26.06 | 46.93 |
| 6 | Lamonte | Class 1 | 0.00 | 1.47 | 1.47 | 0.00 | 58.94 | 58.94 |
| 7 | Cole Camp | Class 2 | 0.00 | 0.84 | 0.84 | 0.00 | 25.06 | 25.06 |
| 8 | Lincoln | Class 1 | 0.00 | 0.80 | 0.80 | 0.00 | 32.01 | 32.01 |
| 9 | Warsaw | Class 1 & 2 | 0.00 | 1.74 | 1.74 | 0.00 | 55.07 | 55.07 |
| 10 | Cole Camp | Class 3 | 0.30 | 2.36 | 2.66 | 11.92 | 70.82 | 82.75 |
| 11 | Cole Camp | Class 3 | 0.07 | 2.44 | 2.51 | 2.64 | 73.19 | 75.83 |
| 12 | Cole Camp | Class 3 | 0.54 | 1.12 | 1.66 | 22.67 | 33.54 | 56.21 |
| 13 | Hughesville | Class 3 | 0.00 | 2.30 | 2.30 | 0.00 | 69.03 | 69.03 |
| 14 | Hughesville | Class 3 | 0.15 | 1.69 | 1.84 | 8.86 | 50.65 | 59.51 |
| 15 | Hughesville | Class 3 | 0.10 | 1.19 | 1.29 | 6.30 | 35.69 | 41.99 |
| 16 | Sedalia | Class 3 | 0.51 | 2.43 | 2.93 | 24.62 | 72.83 | 97.44 |
| 17 | Sedalia | Class 3 | 1.11 | 1.56 | 2.67 | 52.14 | 46.73 | 98.87 |
| 18 | Lincoln | Class 3 | 0.13 | 2.34 | 2.47 | 5.24 | 70.17 | 75.41 |
| 19 | Lincoln | Class 3 | 0.21 | 2.28 | 2.48 | 8.95 | 68.31 | 77.26 |
| 20 | Warsaw | Class 3 | 0.27 | 2.17 | 2.43 | 13.34 | 65.00 | 78.33 |
| 21 | Warsaw | Class 3 | 0.60 | 2.45 | 3.05 | 26.86 | 73.37 | 100.22 |
| 22 | Warsaw | Class 3 | 0.30 | 2.49 | 2.79 | 15.12 | 74.71 | 89.83 |
| 23 | Warsaw | Class 3 | 0.49 | 1.36 | 1.84 | 21.36 | 40.72 | 62.07 |
| 24 | Lamonte | Class 3 | 0.17 | 2.46 | 2.63 | 6.76 | 73.75 | 80.52 |
| 25 | Lamonte | Class 3 | 0.36 | 2.37 | 2.74 | 16.53 | 71.24 | 87.77 |

Table 19. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|-------------|----------|---------|---------|----------|----------------|------------------|
| Hughesville | 1 | A1 | 1.56 | 1 | | 1.56 |
| | 13 | A3 | 2.30 | | | 5.43 |
| | 14 | A3 | 1.84 | 2 | 13-14-15 | 5.43 |
| | 15 | A3 | 1.29 | | | 5.43 |
| Sedalia | 2 | A1 | 1.44 | 3 | | 1.44 |
| | 3 | A1 | 1.52 | 4 | | 1.52 |
| | 4 | A2 | 1.62 | 5 | 4-5 | 2.91 |
| | 5 | A2 | 1.29 | | | 2.91 |
| | 16 | A3 | 2.93 | 6 | 16-17 | 5.60 |
| | 17 | A3 | 2.67 | | | 5.60 |
| Lamonte | 6 | A1 | 1.47 | 7 | | 1.47 |
| | 24 | A3 | 2.63 | 8 | 24-25 | 5.37 |
| | 25 | A3 | 2.74 | | | 5.37 |
| Cole Camp | 7 | A2 | 0.84 | | | 3.50 |
| | 10 | A3 | 2.66 | 9 | 7-10-7-11-7-12 | 9.35 |
| | 11 | A3 | 2.51 | | | 9.35 |
| | 12 | A3 | 1.66 | | | 9.35 |
| Lincoln | 8 | A1 | 0.80 | 10 | | 0.80 |
| | 18 | A3 | 2.47 | 11 | 18-19 | 4.95 |
| | 19 | A3 | 2.48 | | | 4.95 |
| Warsaw | 9 | A1 & A2 | 1.74 | 12 | | 1.74 |
| | 20 | A3 | 2.43 | | | 10.11 |
| | 21 | A3 | 3.05 | 13 | 20-21-22-23 | 10.11 |
| | 22 | A3 | 2.79 | | | 10.11 |
| | 23 | A3 | 1.84 | | | 10.11 |

4.6 Morgan, Miller and Camden Counties

Figure 4.6.1 shows the existing eight depots in Stover, Versailles, Eldon, Tuscumbia, Osage Beach, Iberia, Montreal, and Camdenton, and service areas in Morgan, Miller, and Camden counties.

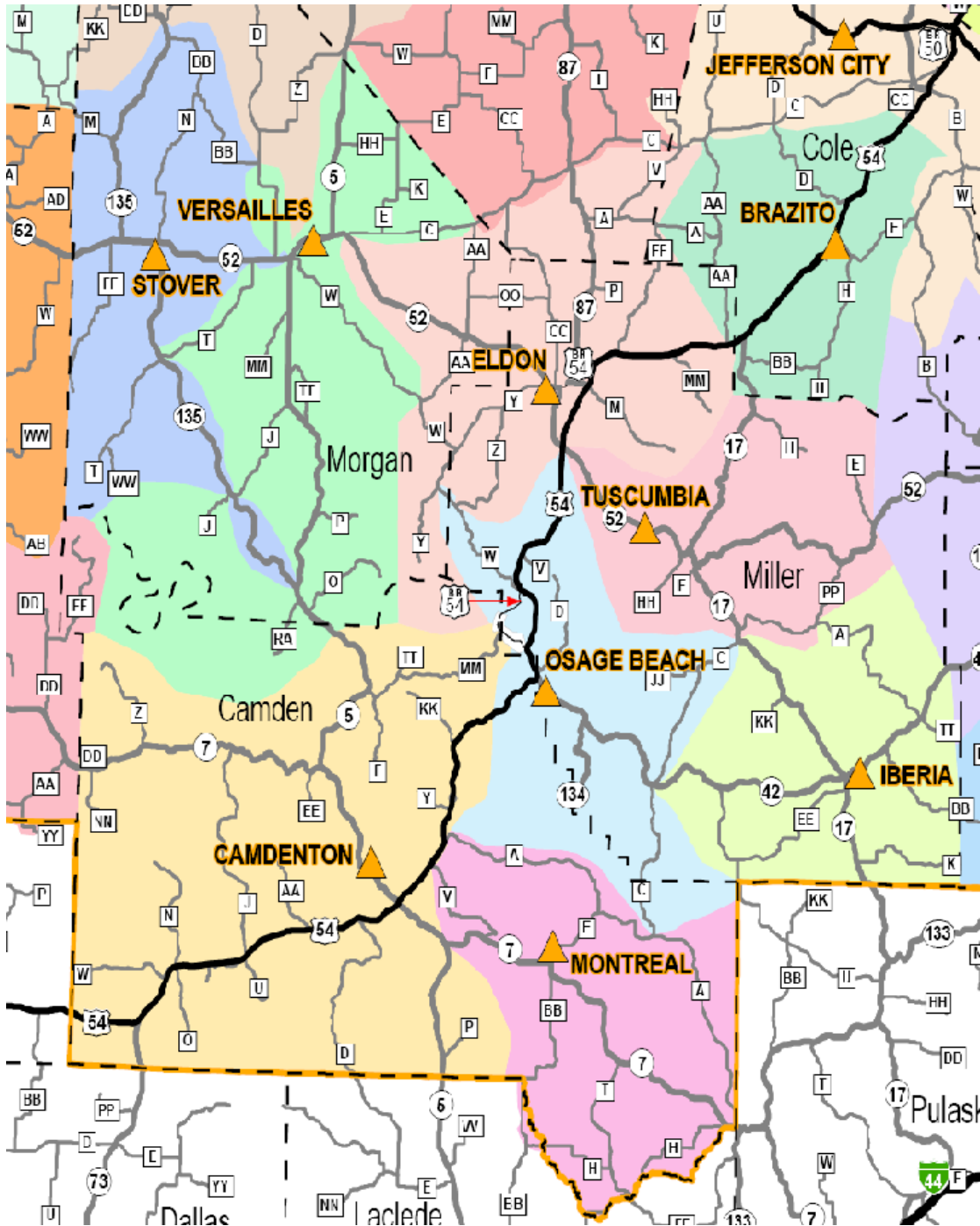


Figure 4.6.1. Morgan, Miller, and Camden original map

Figure 4.6.2 shows the simplified map of Morgan, Miller, and Camden Counties with roads distinguished by their classes.

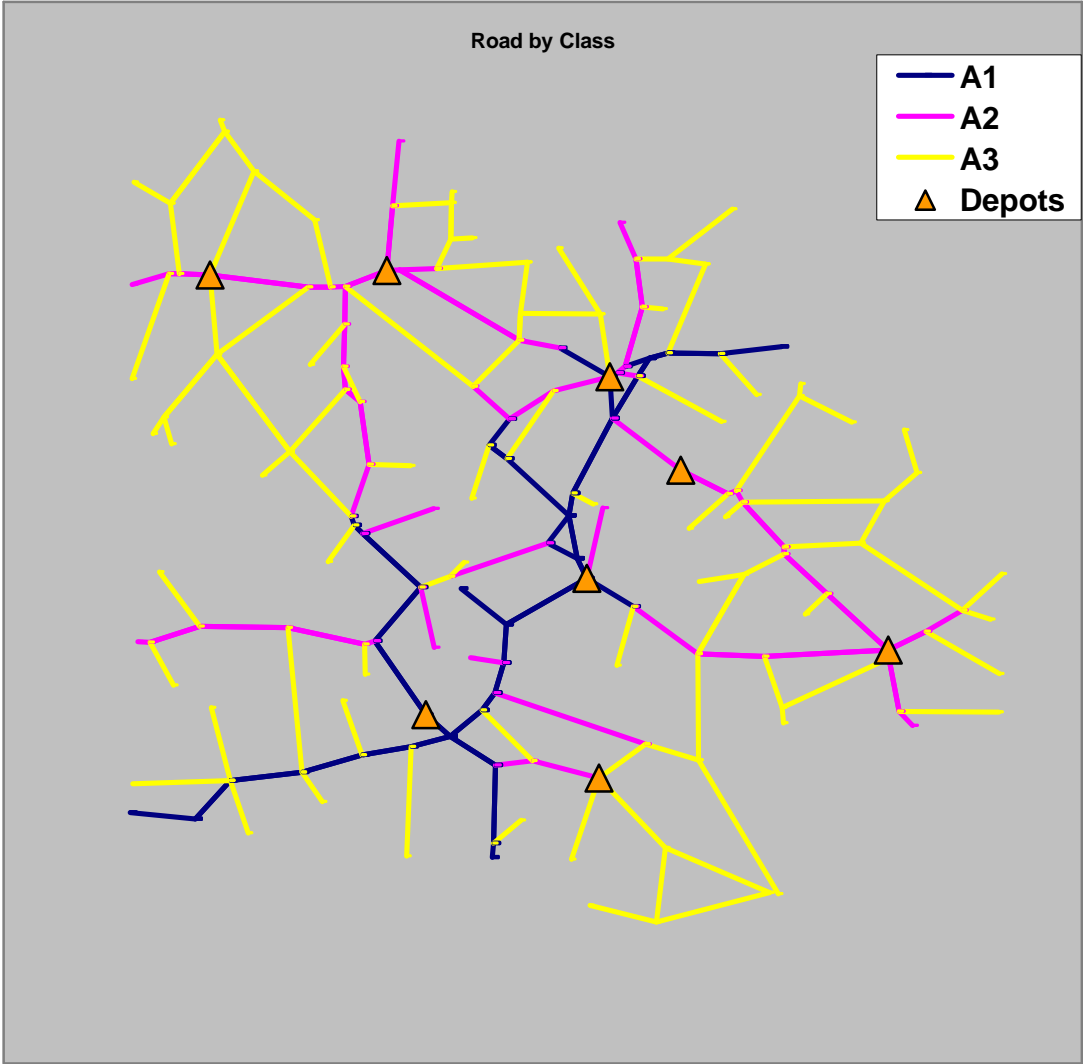


Figure 4.6.2. Simplified map

Figures 4.6.3–4.6.8 show the road maps for each class and best routes to serve. The routes assigned to depots in Stover, Versailles, Eldon, Tuscumbia, Osage Beach, Iberia, Montreal, and Camdenton are denoted by numbers 143, 111, 59, 42, 65, 12, 62 and 103.

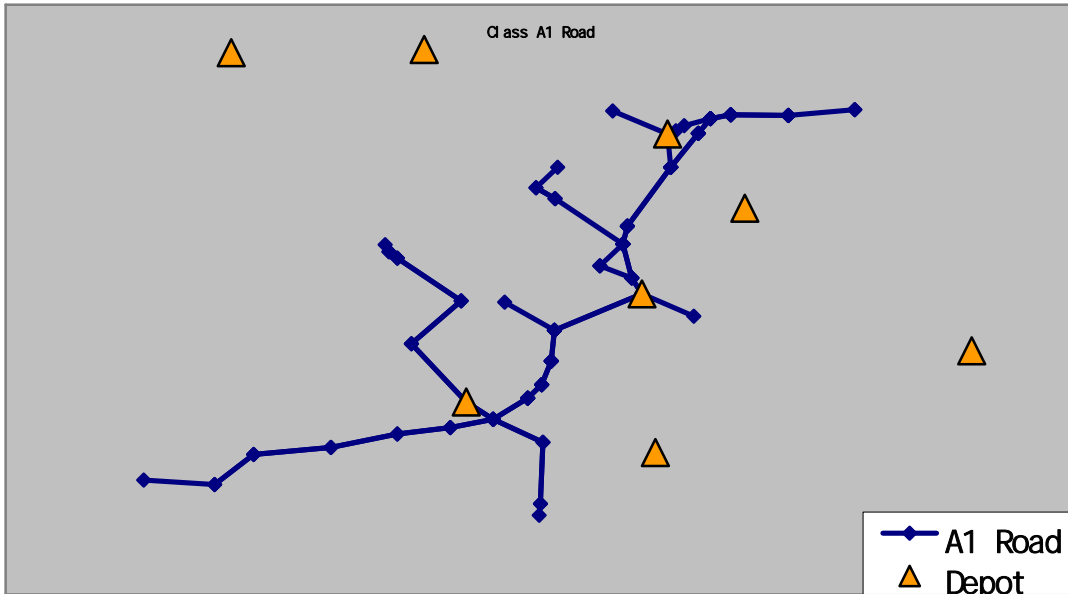


Figure 4.6.3. Class A1 roads

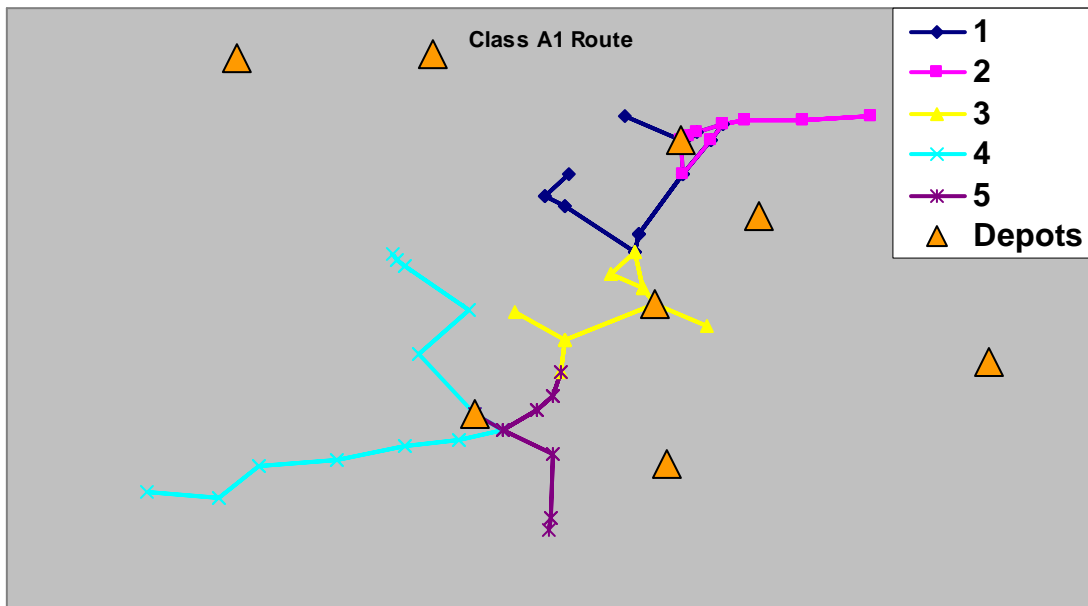


Figure 4.6.4. Class A1 routes

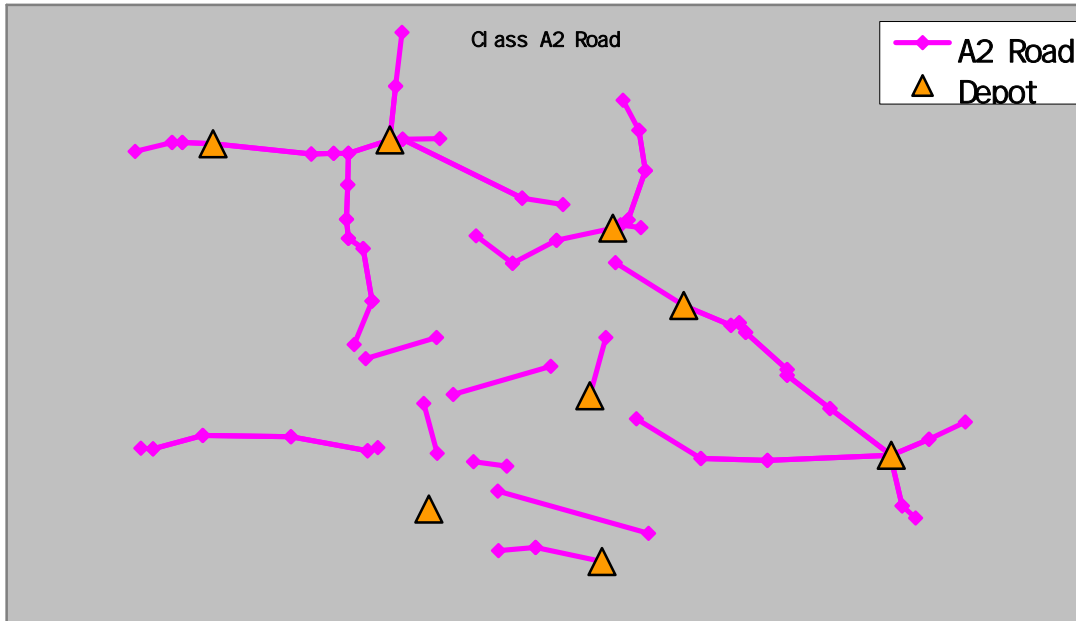


Figure 4.6.5. Class A2 roads

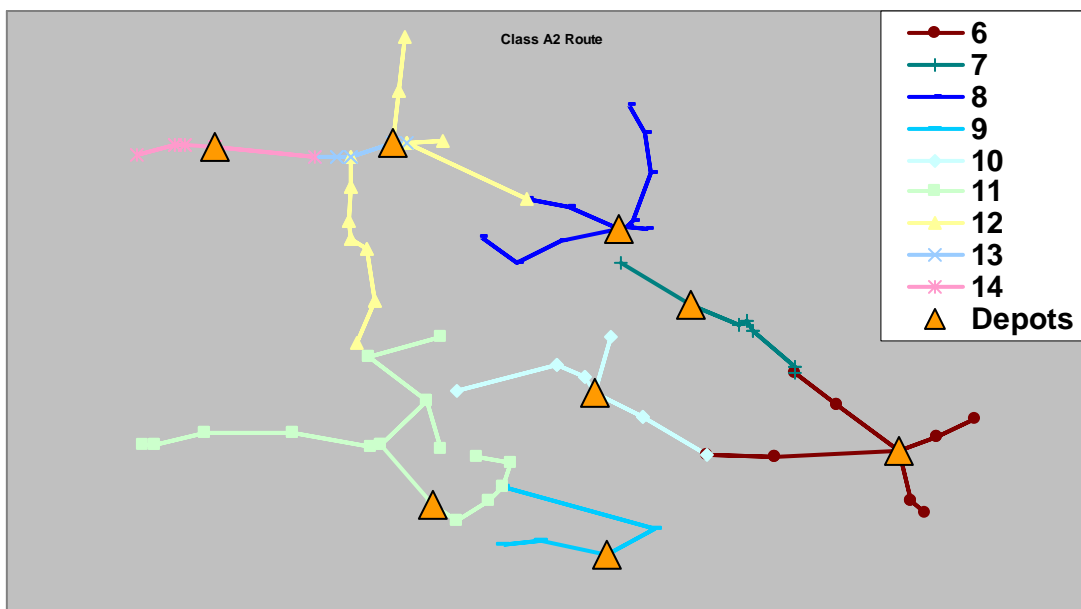


Figure 4.6.6. Class A2 routes

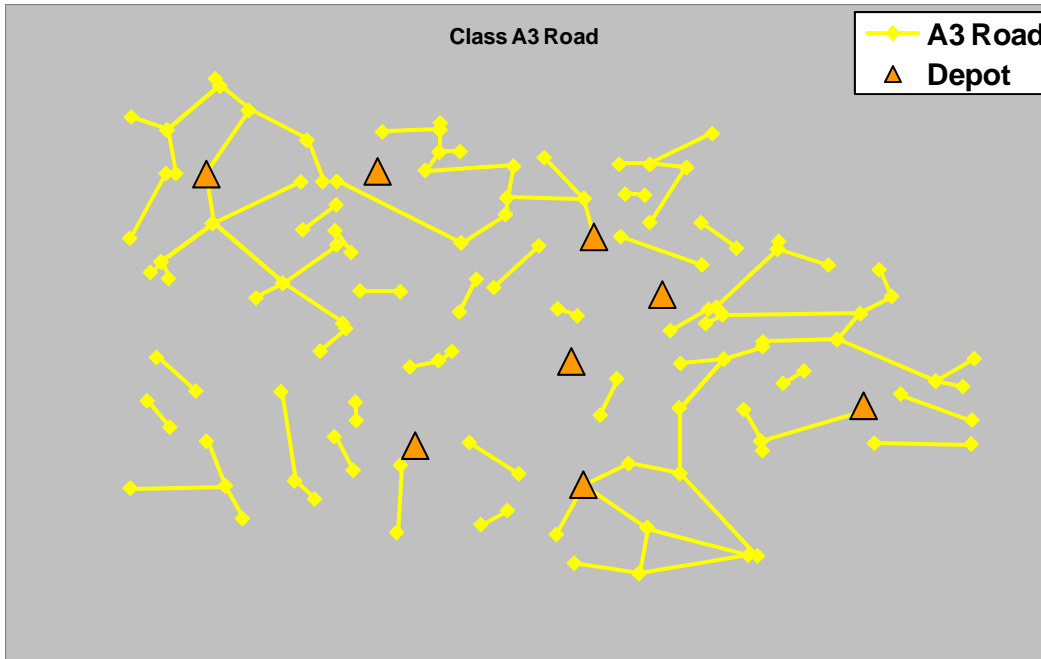


Figure 4.6.7. Class A3 roads

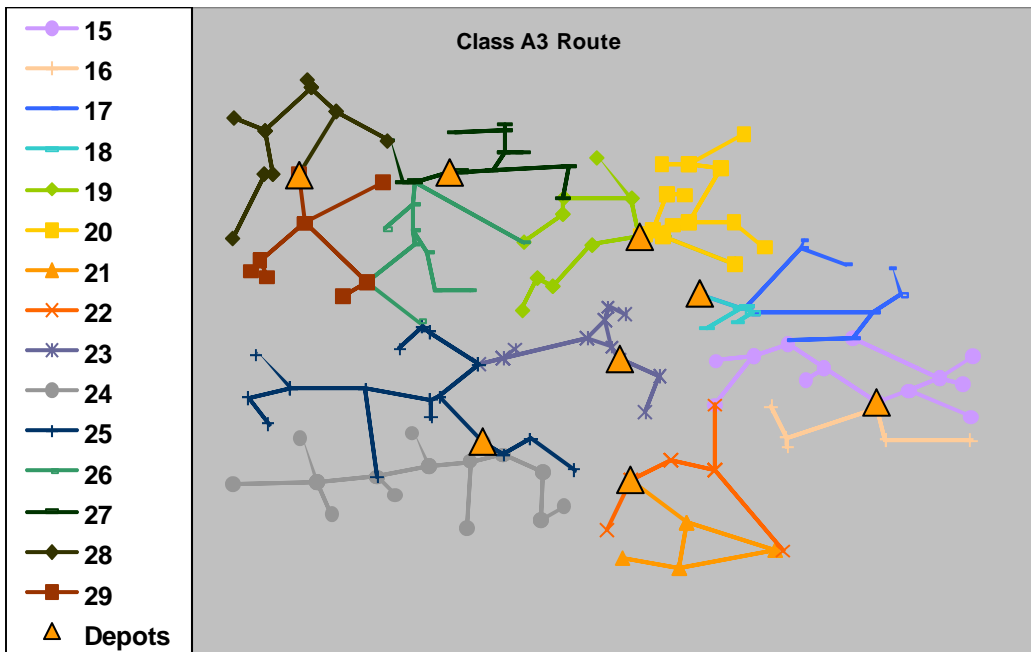


Figure 4.6.8. Class A3 routes

Figure 4.6.9 graphically shows the service regions for the six depots, and Figure 4.6.10 graphically shows final routes recommended for the two counties. Table 20 describes all the routes.

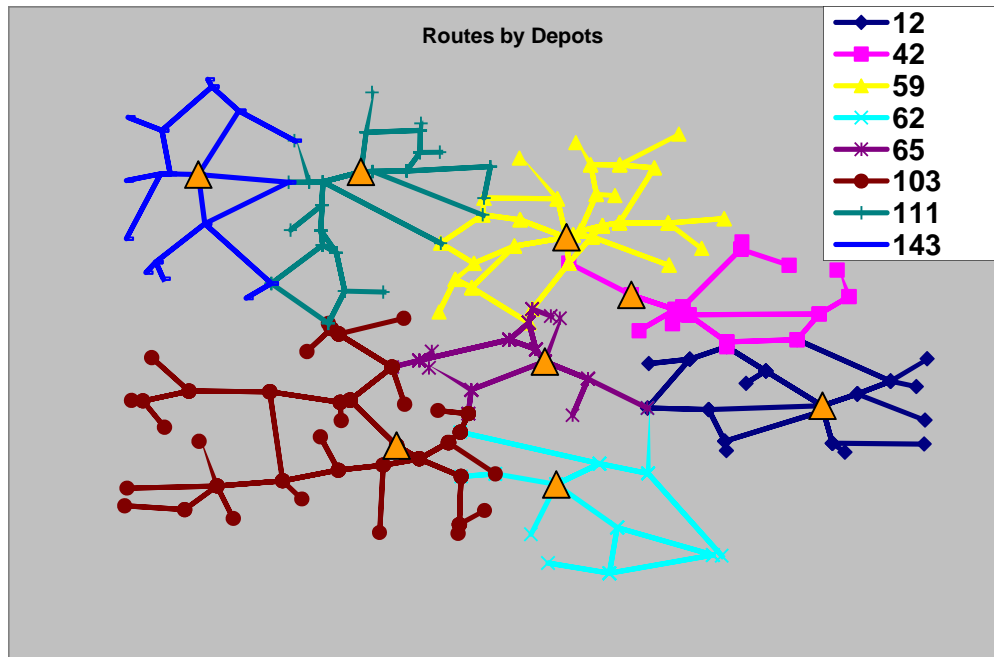


Figure 4.6.9. Service regions by depots

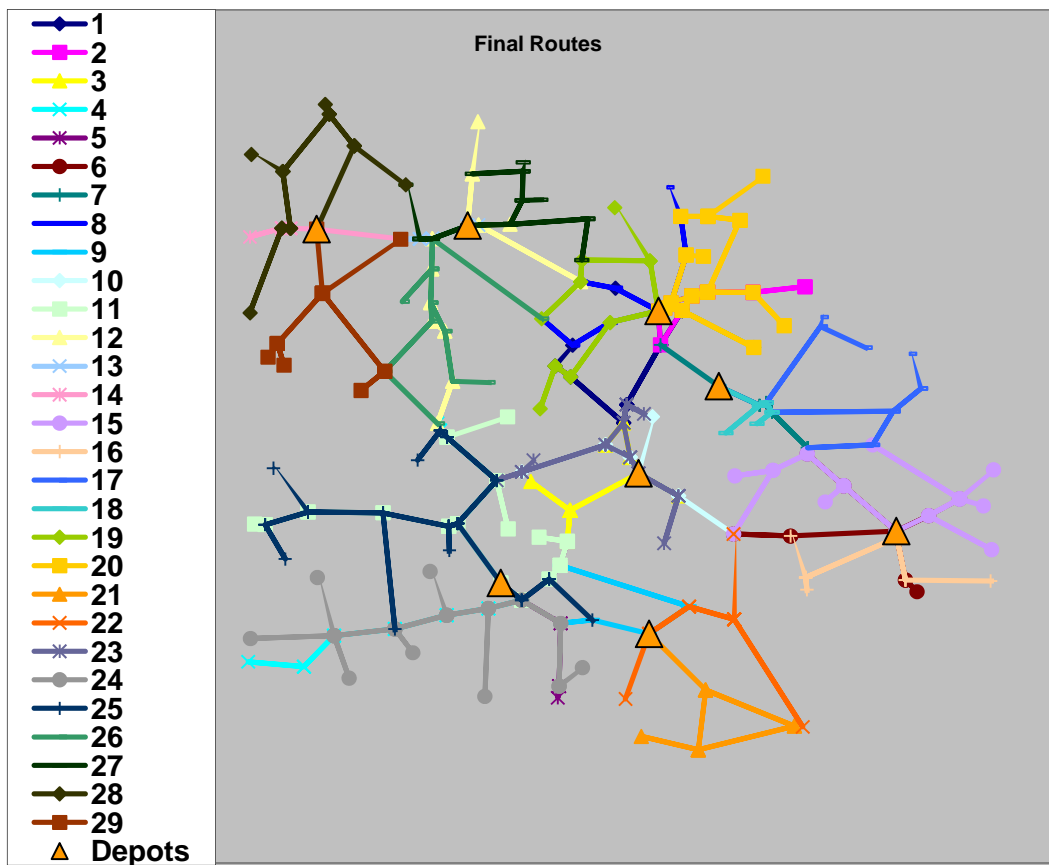


Figure 4.6.10. Final routes

Table 20. Route description

| Route # | Depot | Class | Description |
|----------------|--------------|--------------|---|
| 1 | Eldon | A1 | BU54, 52 from int. with 54 to Miller west limit, 54 from int. with 52 to int. with W, W |
| 2 | Eldon | A1 | 54 from int. with 52 to east |
| 3 | Osage Beach | A1 | BU 54, 54 from int. with W to int. with KK, KK |
| 4 | Camdenton | A1 | 54 from int. with 5 to west, 5 from int. with 135 to depot Camdenton |
| 5 | Camdenton | A1 | 5 from depot Camdenton to south, 54 from int. with KK to int. with 5 |
| 6 | Iberia | A2 | 42 from int. with C to east, 17 from int. with C to south |
| 7 | Tuscumbia | A2 | 52 from int. with 54 to int. with 17, 17 from int. with 52 to int. with C |
| 8 | Eldon | A2 | 87, Y, W |
| 9 | Montreal | A2 | A, 7 from depot Montreal to int. with 5 |
| 10 | Osage Beach | A2 | 42, D, MM |
| 11 | Camdenton | A2 | Y, F, O, 7 from west to int. with 5 |
| 12 | Versailles | A2 | 5 from north to int. with 135 |
| 13 | Versailles | A2 | 52 from depot Versailles to int. with T |
| 14 | Stover | A2 | 52 from int. with T to west |
| 15 | Iberia | A3 | KK, C, JJ, BB, TT, A |
| 16 | Iberia | A3 | K, EE, U |
| 17 | Tuscumbia | A3 | 17, H, 52, E, PP, A |
| 18 | Tuscumbia | A3 | F, HH |
| 19 | Eldon | A3 | CC, OO, AA, Y, Z, M |
| 20 | Eldon | A3 | M, MM, FF, A, V, P |
| 21 | Montreal | A3 | 7, H, T |
| 22 | Montreal | A3 | E, A, C, BB |
| 23 | Osage Beach | A3 | 134, V, TT |
| 24 | Camdenton | A3 | D, AA, O, N, W, U, P |
| 25 | Camdenton | A3 | V, RA, EE, J, Z, NN |
| 26 | Versailles | A3 | W, TT, P, J, 135, MM |
| 27 | Versailles | A3 | D, C, AA, E, K, HH |
| 28 | Stover | A3 | N, BB, 135, M, FF |
| 29 | Stover | A3 | 135, T, J, WW |

The summary of these routes is given in Table 21. The table shows necessary time and distance to serve each route. As shown in Table 22, these routes can be run by 17 snow plow trucks.

Table 21. Summary of routes

| Route # | Depot | Class | Total DH time | Total service time | Total cycle time | Total DH distance | Total service distance | Total cycle distance |
|----------------|--------------|--------------|----------------------|---------------------------|-------------------------|--------------------------|-------------------------------|-----------------------------|
| 1 | Eldon | A1 | 0.03 | 1.89 | 1.92 | 1.14 | 75.63 | 76.76 |
| 2 | Eldon | A1 | 0.03 | 0.98 | 1.01 | 1.14 | 39.32 | 40.46 |
| 3 | Osage Beach | A1 | 0.00 | 1.91 | 1.91 | 0.00 | 76.21 | 76.21 |
| 4 | Camdenton | A1 | 0.04 | 1.95 | 1.99 | 2.06 | 77.82 | 79.87 |
| 5 | Camdenton | A1 | 0.04 | 1.13 | 1.17 | 2.06 | 45.13 | 47.19 |
| 6 | Iberia | A2 | 0.00 | 2.10 | 2.10 | 0.00 | 62.96 | 62.96 |
| 7 | Tuscumbia | A2 | 0.00 | 1.00 | 1.00 | 0.00 | 29.91 | 29.91 |
| 8 | Eldon | A2 | 0.16 | 1.63 | 1.80 | 8.25 | 48.91 | 57.16 |
| 9 | Montreal | A2 | 0.20 | 1.18 | 1.37 | 7.88 | 35.33 | 43.20 |
| 10 | Osage Beach | A2 | 0.28 | 1.28 | 1.56 | 13.81 | 38.50 | 52.31 |
| 11 | Camdenton | A2 | 0.98 | 1.86 | 2.85 | 49.05 | 55.93 | 104.97 |
| 12 | Versailles | A2 | 0.09 | 2.49 | 2.58 | 3.44 | 74.73 | 78.17 |
| 13 | Versailles | A2 | 0.09 | 0.25 | 0.34 | 3.44 | 7.54 | 10.97 |
| 14 | Stover | A2 | 0.00 | 0.68 | 0.68 | 0.00 | 20.38 | 20.38 |
| 15 | Iberia | A3 | 0.71 | 2.33 | 3.04 | 28.46 | 69.86 | 98.32 |
| 16 | Iberia | A3 | 0.21 | 1.30 | 1.52 | 8.56 | 39.12 | 47.68 |
| 17 | Tuscumbia | A3 | 0.25 | 2.50 | 2.75 | 9.81 | 74.99 | 84.81 |
| 18 | Tuscumbia | A3 | 0.25 | 0.31 | 0.56 | 9.81 | 9.33 | 19.14 |
| 19 | Eldon | A3 | 0.43 | 2.31 | 2.74 | 18.12 | 69.28 | 87.41 |
| 20 | Eldon | A3 | 0.65 | 1.60 | 2.24 | 28.39 | 47.86 | 76.25 |
| 21 | Montreal | A3 | 0.00 | 2.23 | 2.23 | 0.00 | 67.05 | 67.05 |
| 22 | Montreal | A3 | 0.00 | 2.23 | 2.23 | 0.00 | 66.78 | 66.78 |
| 23 | Osage Beach | A3 | 0.80 | 0.64 | 1.43 | 36.02 | 19.09 | 55.11 |
| 24 | Camdenton | A3 | 0.95 | 2.46 | 3.42 | 47.72 | 73.86 | 121.57 |
| 25 | Camdenton | A3 | 1.59 | 2.17 | 3.76 | 72.20 | 65.08 | 137.28 |
| 26 | Versailles | A3 | 0.71 | 2.19 | 2.90 | 28.44 | 65.82 | 94.26 |
| 27 | Versailles | A3 | 0.32 | 1.61 | 1.92 | 12.61 | 48.22 | 60.83 |
| 28 | Stover | A3 | 0.03 | 2.70 | 2.72 | 1.13 | 80.86 | 81.99 |
| 29 | Stover | A3 | 0.00 | 2.20 | 2.20 | 0.00 | 66.10 | 66.10 |

Table 22. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|-------------|----------|-------|---------|----------|-------------------|------------------|
| Eldon | 1 | A1 | 1.92 | 1 | | 1.92 |
| | 2 | A1 | 1.01 | 2 | | 1.01 |
| | 8 | A2 | 1.80 | | | 4.54 |
| | 19 | A3 | 2.74 | 3 | 8-19-8-20 | 8.58 |
| | 20 | A3 | 2.24 | | | 8.58 |
| Osage Beach | 3 | A1 | 1.91 | 4 | | 1.91 |
| | 10 | A2 | 1.56 | | | 2.99 |
| | 23 | A3 | 1.43 | 5 | 10-23 | 4.98 |
| Camdenton | 4 | A1 | 1.99 | 6 | | 1.99 |
| | 5 | A1 | 1.17 | 7 | | 1.17 |
| | 11 | A2 | 2.85 | 8 | | 2.85 |
| | 24 | A3 | 3.42 | | | 7.18 |
| | 25 | A3 | 3.76 | 9 | 24-25 | 7.18 |
| Iberia | 6 | A2 | 2.10 | | | 5.14 |
| | 15 | A3 | 3.04 | 10 | 6-15-6-16 | 8.76 |
| | 16 | A3 | 1.52 | | | 8.76 |
| Tuscumbia | 7 | A2 | 1.00 | | | 3.75 |
| | 17 | A3 | 2.75 | 11 | 7-17-7-18 | 5.31 |
| | 18 | A3 | 0.56 | | | 5.31 |
| Montreal | 9 | A2 | 1.37 | | | 3.60 |
| | 21 | A3 | 2.23 | 12 | 9-21-9-22 | 7.20 |
| | 22 | A3 | 2.23 | | | 7.20 |
| Versailles | 12 | A2 | 2.58 | | | 5.82 |
| | 13 | A2 | 0.34 | | | 5.82 |
| | 26 | A3 | 2.90 | 13 | 12-13-26-12-13-27 | 10.66 |
| | 27 | A3 | 1.92 | | | 10.66 |
| Stover | 14 | A2 | 0.68 | | | 2.40 |
| | 28 | A3 | 2.72 | 14 | 14-28-14-29 | 6.28 |
| | 29 | A3 | 2.20 | | | 6.28 |

4.7 Boone County

Figure 4.7.1 shows the existing five depots in Harrisburg, Hallsville, Columbia, Rocheport, and Ashland, and service areas in Boone County.

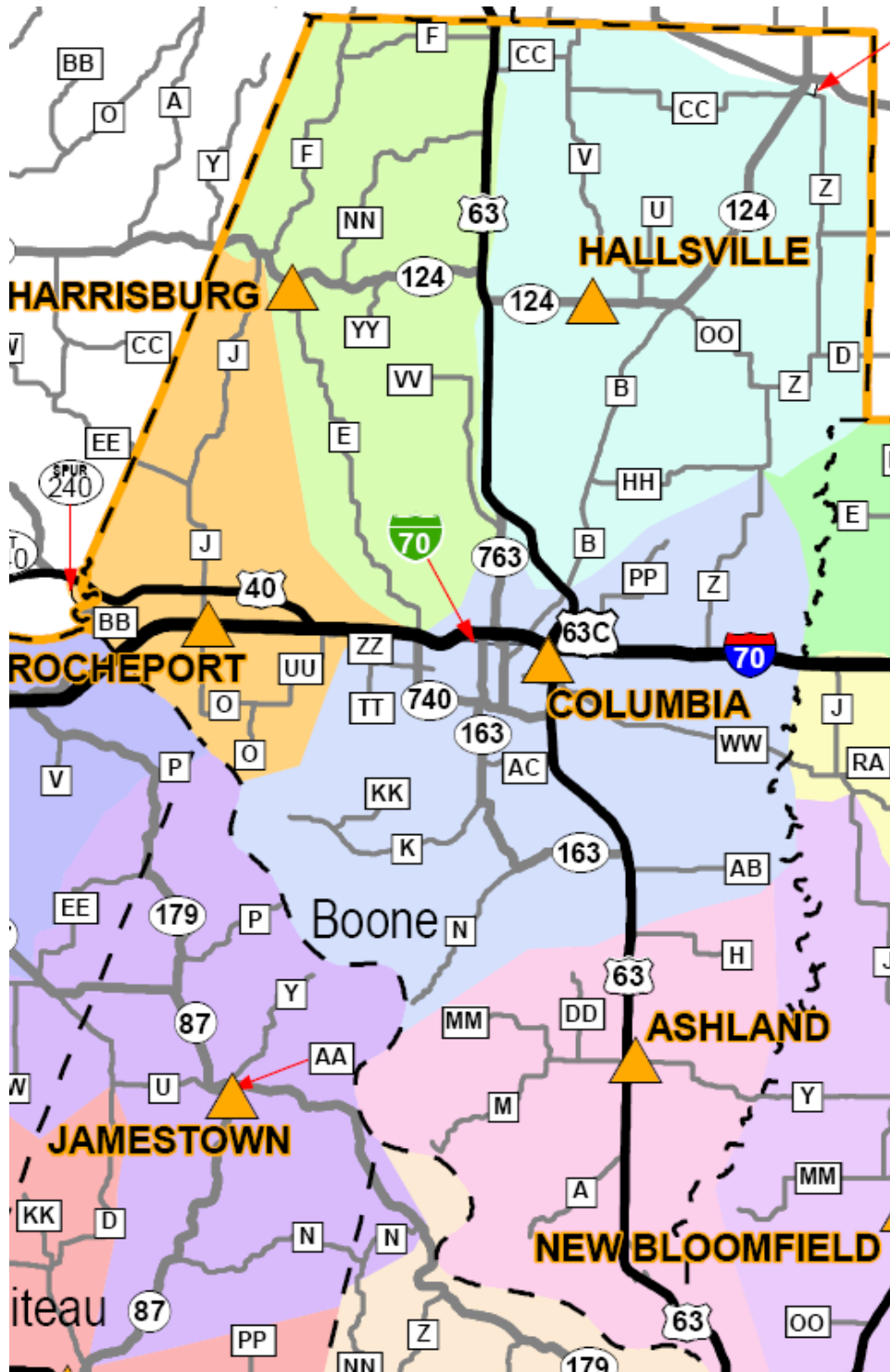


Figure 4.7.1. Boone original map

Figures 4.7.2–4.7.5 show the route maps for each class and best routes to serve. The routes assigned to depots in Harrisburg, Hallsville, Columbia, Rocheport, and Ashland are denoted by numbers 64, 138, 9, 3 and 19, respectively. In Boone County, we divide A1 class into two sub classes: A1, I-70 and Highway 63; and A1', Class A1 non-highway roads.

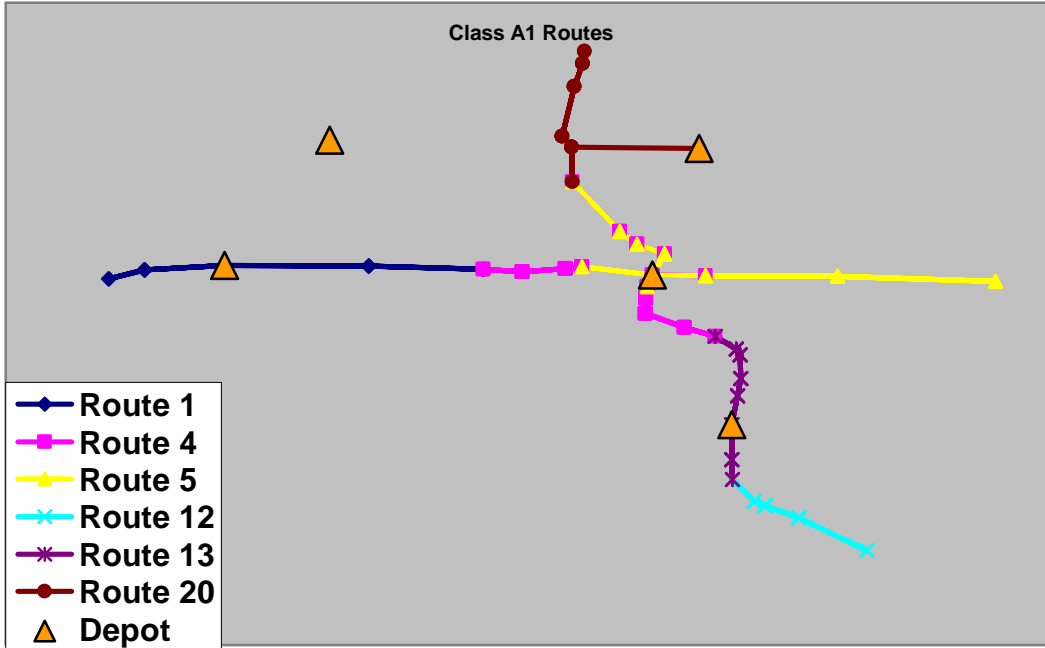


Figure 4.7.2. Class A1 routes

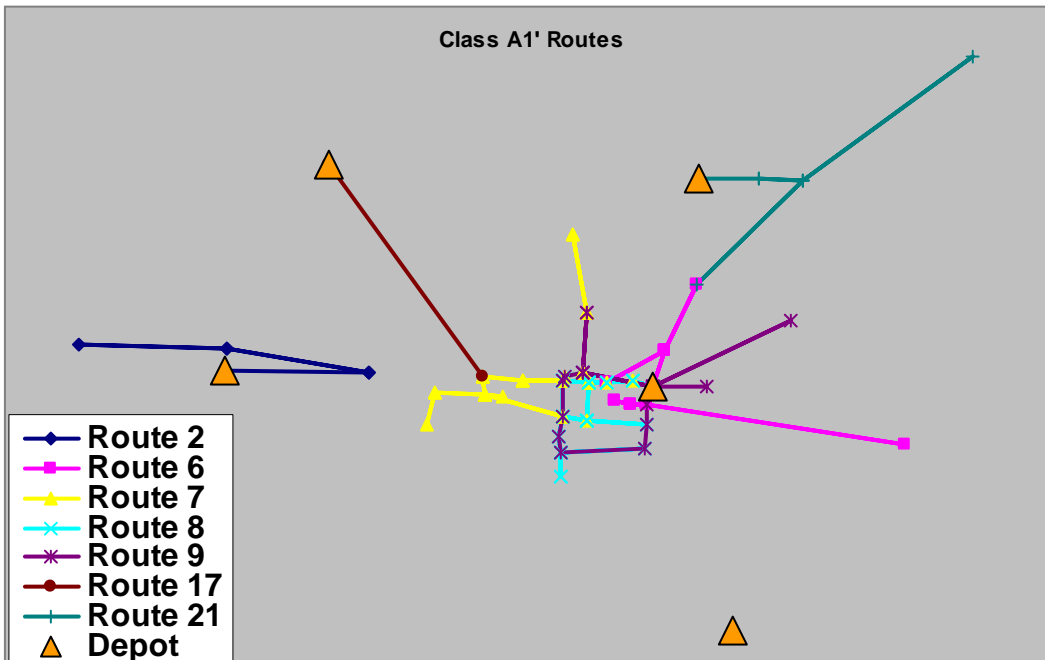


Figure 4.7.3. Class A1' routes

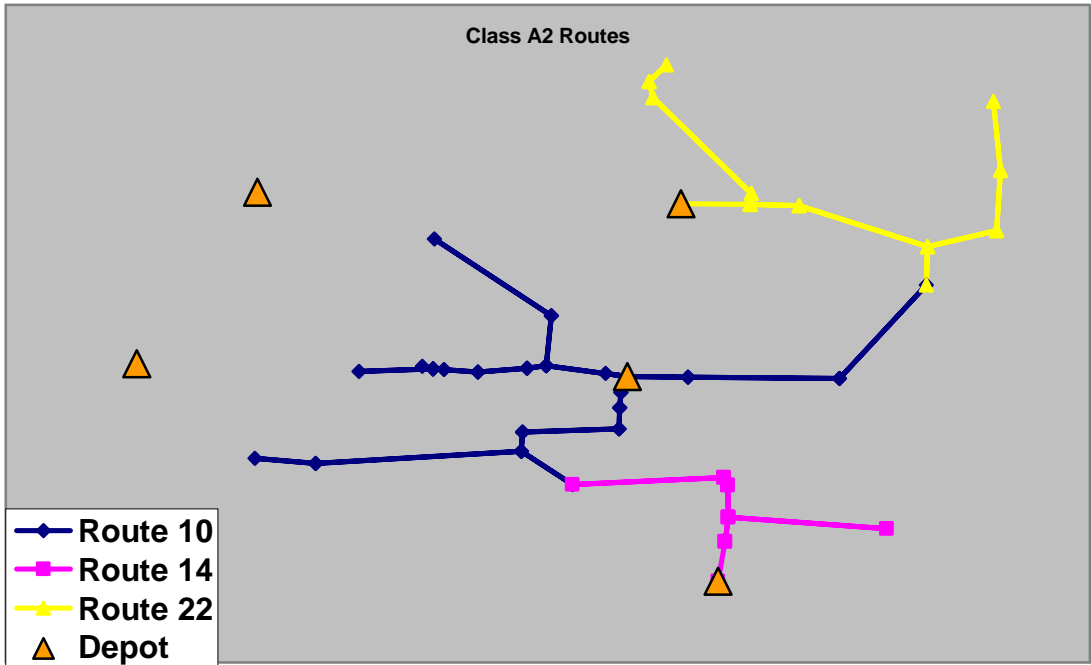


Figure 4.7.4. Class A2 routes

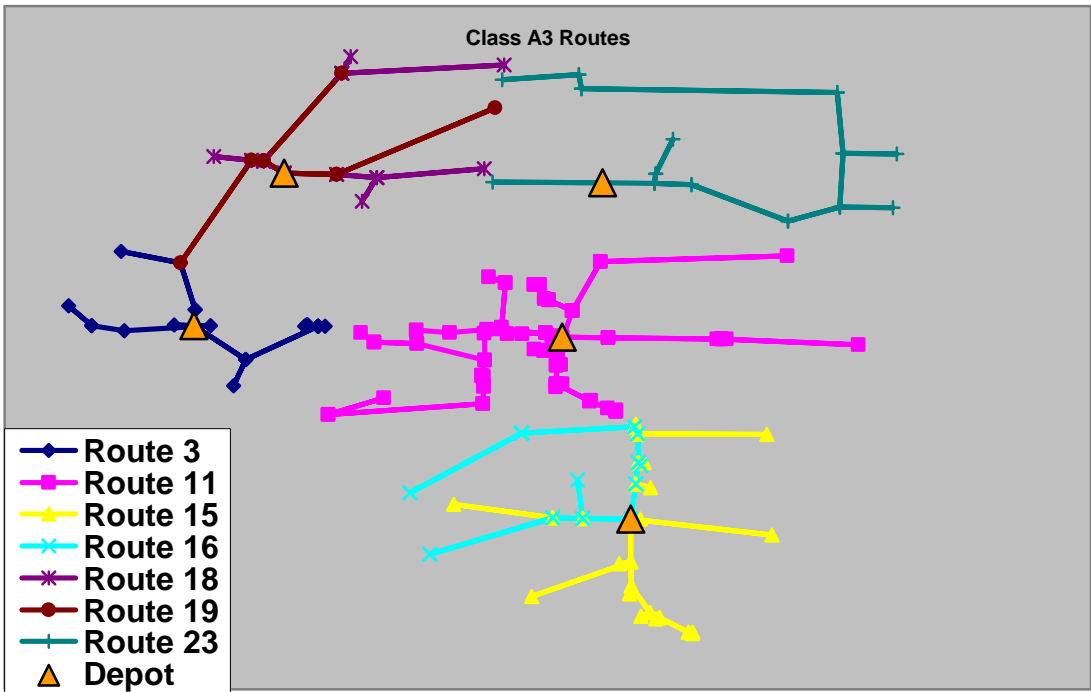


Figure 4.7.5. Class A3 routes

Figure 4.7.6 graphically shows service regions for the five depots, and Figure 4.7.7 graphically shows final routes recommended for the Boone County. Table 21 describes all the routes.

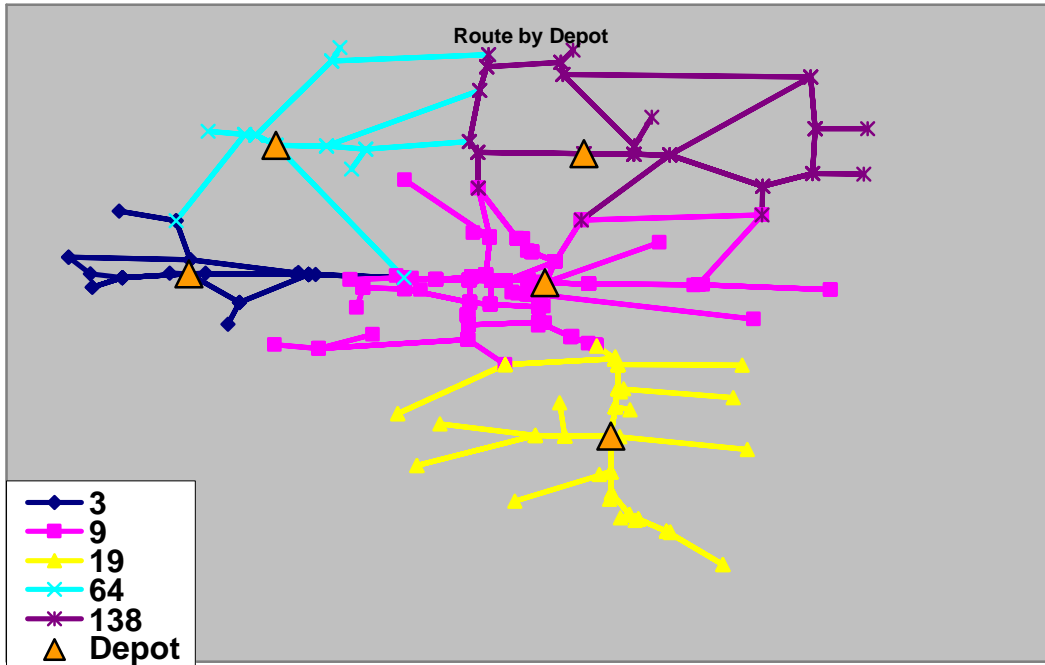


Figure 4.7.6. Service regions by depots

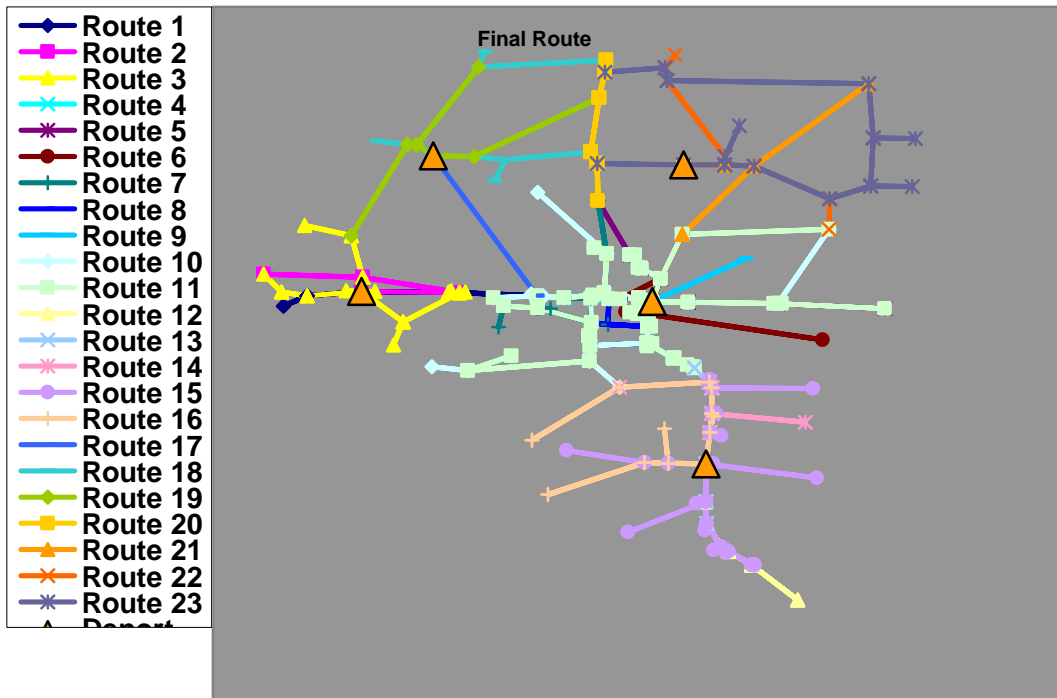


Figure 4.7.7. Final routes

Table 23. Route description

| Route # | Depot | Class | Route description |
|---------|------------|-------|---|
| 1 | Rocheport | A1 | 70, from int. with 179 to int. with E |
| 2 | Rocheport | A1' | Hwy 40, from int. with 70 to int. with BB |
| 3 | Rocheport | A3 | J (from int. with 70 to int. with EE), EE, O, UU, BB |
| 4 | Columbia | A1 | 63 from near int. with 163 to near int. with 763; 70 from int. with E to ? |
| 5 | Columbia | A1 | 63 from int. with WW to ?, 70 from int. with 763 to int. with J |
| 6 | Columbia | A1' | WW, B (from int. with 70 to int. with HH), 63 from int. with WW to int. with B |
| 7 | Columbia | A1' | 763 (south of 70), 740 (west of 163), TT, LP70? |
| 8 | Columbia | A1' | 163 (from 70 to int. with K), 740 (east of 163), AC |
| 9 | Columbia | A1' | 763 (north of 70), 163, PP |
| 10 | Columbia | A2 | VV, Z, K, 163 (from int. with K to int. with N), A3 roads along 70 (int. with E & int. with PP) |
| 11 | Columbia | A3 | KK, ZZ, HH, A4 roads along WW & 163 & 70 (from int. with PP to int. with J) & 63 (from near int. with 163 to near int. with 763) |
| 12 | Ashland | A1 | 63 from south of int. with A to int. with 54 |
| 13 | Ashland | A1 | 63 from south of int. with A to near int. with 763 |
| 14 | Ashland | A2 | 163 (from int. with 63 to int. with N), H |
| 15 | Ashland | A3 | Y, A, AB, M, MM, A4 roads along 63 from int. with 54 to int. with 163 |
| 16 | Ashland | A3 | M ?, DD, N |
| 17 | Harrisburg | A1' | E |
| 18 | Harrisburg | A3 | F(from int. with 63 to int. with T), T, 124 (from int. with F to int. with NN), YY |
| 19 | Harrisburg | A3 | NN, F(from int. with 124 to int. with T), J |
| 20 | Hallsville | A1 | 63 from near int. with 763 to int. with F |
| 21 | Hallsville | A1' | B (from int. with HH to int. with OO), 124 (from int. with OO to int. with Z), |
| 22 | Hallsville | A2 | V, Z |
| 23 | Hallsville | A3 | 124 (from int. with 63 to int. with OO), OO, CC, FF, D, U |

The summary of these routes is given in Table 24. The table shows necessary time and distance to serve each route. As shown in Table 25, these routes can be run by nine snow plow trucks.

Table 24. Summary of routes

| Route # | Depot | Class | Total DH time | Total service time | Total cycle time | Total DH distance | Total service distance | Total cycle distance |
|----------------|--------------|--------------|--------------------------|-----------------------------------|---------------------------------|------------------------------|---------------------------------------|-------------------------------------|
| 1 | Rocheport | A1 | 0.00 | 1.29 | 1.29 | 0.00 | 51.46 | 51.46 |
| 2 | Rocheport | A1' | 0.15 | 0.50 | 0.66 | 7.69 | 15.15 | 22.84 |
| 3 | Rocheport | A3 | 0.08 | 1.68 | 1.76 | 4.17 | 50.40 | 54.58 |
| 4 | Columbia | A1 | 0.09 | 1.42 | 1.51 | 3.67 | 56.77 | 60.44 |
| 5 | Columbia | A1 | 0.18 | 1.28 | 1.46 | 8.08 | 51.35 | 59.42 |
| 6 | Columbia | A1' | 0.04 | 1.87 | 1.91 | 1.67 | 56.19 | 57.86 |
| 7 | Columbia | A1' | 0.19 | 1.67 | 1.86 | 7.98 | 50.14 | 58.12 |
| 8 | Columbia | A1' | 0.20 | 1.65 | 1.85 | 8.62 | 49.52 | 58.14 |
| 9 | Columbia | A1' | 0.30 | 1.09 | 1.40 | 13.78 | 32.85 | 46.62 |
| 10 | Columbia | A2 | 0.79 | 2.14 | 2.93 | 35.01 | 64.34 | 99.36 |
| 11 | Columbia | A3 | 1.82 | 2.15 | 3.97 | 79.30 | 64.47 | 143.77 |
| 12 | Ashland | A1 | 0.21 | 1.02 | 1.23 | 10.59 | 40.64 | 51.23 |
| 13 | Ashland | A1 | 0.00 | 1.32 | 1.32 | 0.00 | 52.77 | 52.77 |
| 14 | Ashland | A2 | 0.26 | 0.51 | 0.77 | 13.07 | 15.30 | 28.37 |
| 15 | Ashland | A3 | 0.64 | 1.59 | 2.24 | 32.18 | 47.78 | 79.96 |
| 16 | Ashland | A3 | 1.21 | 1.05 | 2.26 | 51.03 | 31.63 | 82.65 |
| 17 | Harrisburg | A1' | 0.00 | 0.90 | 0.90 | 0.00 | 27.09 | 27.09 |
| 18 | Harrisburg | A3 | 0.27 | 1.46 | 1.73 | 10.72 | 43.71 | 54.43 |
| 19 | Harrisburg | A3 | 0.34 | 1.38 | 1.72 | 13.63 | 41.43 | 55.05 |
| 20 | Hallsville | A1 | 0.16 | 1.34 | 1.50 | 1.66 | 53.43 | 55.09 |
| 21 | Hallsville | A1' | 0.14 | 0.98 | 1.11 | 5.45 | 29.28 | 34.73 |
| 22 | Hallsville | A2 | 0.49 | 1.53 | 2.02 | 19.71 | 45.88 | 65.59 |
| 23 | Hallsville | A3 | 0.54 | 1.93 | 2.47 | 21.47 | 57.93 | 79.40 |

Table 25. Route operation plan

| Depot ID | Route ID | Class | RT time | Truck ID | RT schedule | Cycle time (hrs) |
|------------|----------|-------|---------|----------|-------------|------------------|
| Rocheport | 1 | A1 | 1.29 | 1 | 1-2 | 1.95 |
| | 2 | A1' | 0.66 | | | 1.95 |
| | 3 | A3 | 1.76 | 2 | 1.76 | |
| Columbia | 4 | A1 | 1.51 | 3 | | 1.51 |
| | 5 | A1 | 1.46 | 4 | | 1.46 |
| | 6 | A1' | 1.91 | 5 | | 1.91 |
| | 7 | A1' | 1.86 | 6 | | 1.86 |
| | 8 | A1' | 1.85 | 7 | | 1.85 |
| | 9 | A1' | 1.40 | 8 | | 1.40 |
| | 10 | A2 | 2.93 | 9 | | 2.93 |
| | 11 | A3 | 3.97 | 10 | | 3.97 |
| Ashland | 12 | A1 | 1.23 | 11 | | 1.23 |
| | 13 | A1 | 1.32 | 12 | | 1.32 |
| | 14 | A2 | 0.77 | | | 3.03 |
| | 15 | A3 | 2.24 | 13 | 14-15-14-16 | 6.04 |
| | 16 | A3 | 2.26 | | | 6.04 |
| Harrisburg | 17 | A1' | 0.90 | 14 | | 0.90 |
| | 18 | A3 | 1.73 | 15 | 18-19 | 3.45 |
| | 19 | A3 | 1.72 | | | 3.45 |
| Hallsville | 20 | A1 | 1.50 | 16 | | 1.50 |
| | 21 | A1' | 1.11 | 17 | | 1.11 |
| | 22 | A2 | 2.02 | 18 | 22-23 | 4.49 |
| | 23 | A3 | 2.47 | | | 4.49 |

4.8 Summary

Table 26 presents the summary of results. The first and second columns show counties and locations of truck depots used in those counties. The next column represents actual mileage of roads to be served in each region assigned to the depot. The fourth column shows the actual service mileage in each region during 12 hours, considering the service frequency. The last two columns show the current and recommended numbers of trucks used in each region. Note that the recommended number is the optimal (minimal) number of trucks computed in previous sections plus one to three additional trucks, depending on the service mileage, to prepare for emergency and heavy storm scenarios.

As seen in the last row, there are a total of 7,969 miles of roads to serve and MoDOT actually serves 19,490 miles during 12 hours of regular storm, considering different service frequency based on ADT. MoDOT currently uses about 169 trucks and our solution can use only 136

trucks to provide the same level of service. That is, we can decrease the number of trucks by 19.5%.

Table 26. Summary of results (*estimated number)

| County | Depot | Road mileage | Service mileage for 12 hours | Number of trucks | |
|-----------|----------------|--------------|------------------------------|------------------|-------------|
| | | | | Current | Recommended |
| Cole | Brazito | 207 | 505 | 3 | 4 |
| | Jefferson City | 264 | 1046 | 14 | 7 |
| Callaway | Auxvasse | 181 | 373 | 5 | 3 |
| | Loomfield | 158 | 524 | 3 | 4 |
| | Fulton | 442 | 1521 | 7 | 8 |
| | Mokane | 146 | 170 | 3 | 2 |
| Osage | Linn | 253 | 467 | 5 | 4 |
| Gasconade | Drake | 244 | 355 | 6 | 3 |
| Maries | Vienna | 291 | 652 | 6 | 4 |
| | Belle | 252 | 289 | 3 | 3 |
| | Owensville | 181 | 275 | 3 | 3 |
| | Chamois | 116 | 116 | 4 | 2 |
| | Meta | 127 | 127 | 3 | 2 |
| Cooper | Blackwater | 198 | 549 | 4 | 4 |
| Moniteau | Boonville | 210 | 564 | 7 | 4 |
| | Jamestown | 182 | 182 | 3 | 2 |
| | California | 213 | 535 | 5 | 4 |
| | Tipton | 312 | 615 | 5* | 4 |
| Benton | Hughesville | 218 | 530 | 3* | 4 |
| Pettis | Sedalia | 307 | 968 | 9 | 7 |
| | Lamonte | 204 | 499 | 3 | 4 |
| | Cole Camp | 203 | 228 | 3 | 2 |
| | Lincoln | 170 | 331 | 3 | 3 |
| | Warsaw | 309 | 584 | 5 | 4 |
| Morgan | Eldon | 281 | 905 | 6 | 6 |
| Miller | Osage Beach | 134 | 553 | 4 | 4 |
| Camden | Camdenton | 318 | 989 | 6* | 7 |
| | Iberia | 172 | 235 | 3 | 2 |
| | Tuscumbia | 114 | 144 | 3 | 2 |
| | Montreal | 169 | 204 | 3 | 2 |
| | Versailles | 196 | 279 | 3* | 2 |
| | Stover | 167 | 188 | 3* | 2 |
| Boone | Rocheport | 117 | 450 | 3 | 2 |
| | Columbia | 426 | 1974 | 11* | 8 |
| | Ashland | 188 | 670 | 3 | 3 |
| | Harrisburg | 112 | 248 | 3* | 2 |
| | Hallsville | 187 | 646 | 3* | 3 |
| Total | | 7969 | 19490 | 169 | 136 |

5. CONCLUSIONS

The objective of this research was to develop a systematic, heuristic-based optimization approach to integrate the winter road maintenance planning decisions for depot location, sector design, vehicle route design, vehicle scheduling, and fleet configuration. The solution methodology that we developed achieves the objective of a more integrated approach to the problems considered. Additionally, the research achieved the goal of considering practical, real-world objectives, constraints, and problem characteristics. This was made possible by working with MoDOT to identify the necessary aspects of each of the planning problems studied. The inclusion of a heterogeneous fleet provided a better representation of MoDOT's current operations. The inclusion of the vehicle scheduling supported the idea that, when service frequency and vehicle capacity are considered, it is possible for a vehicle to service multiple routes. Last, the consideration of service frequency in lieu of minimizing deadheading time accurately reflects MoDOT's strategy.

The solution applied to the transportation network of District 5, Missouri, is very promising. The results indicate that our solution would allow MoDOT to maintain the same high level of service with significantly less resources. The promising results from the real world problem support the assertion of the importance of a more integrated approach to the winter road maintenance problems studied in this research. Our work should provide winter road maintenance planners with the ability to make more informed, economically beneficial, and successful decisions.

REFERENCES

- Campbell JF, Langevin A. Roadway snow and ice control. In: Dror M, editor. Arc routing: theory, solutions and applications. Boston, MA: Kluwer, 2000. p. 389–418.
- Gupta JD. Development of a model to assess costs of opening a new or closing an existing outpost or county garage. Report No FHWA/OH-99/003, University of Toledo, Ohio, 1998.
- Marks HD, Stricker R. Routing for public service vehicles. Journal of the Urban Planning and Development Division 1971;97:165–78.
- Qiao, H. Capacitated rural directed arc routing problem: algorithms and applications. Master's thesis, University of Maryland, College Park, 1999.
- Reinert KA, Miller TR, Dickerson HG. A location-assignment model for urban snow and ice control operations. Urban Analysis 1985;8 :175–91.
- Skiena S. The Algorithm Design Manual. Springer, 1998
- Thimbleby, H. The directed Chinese Postman Problem. Software Practice and Experience 2003;33:1081-1096.

APPENDIX A. COMPUTER PROGRAMS

CPP.JAVA

```
/**
 * Class for finding and printing the optimal Chinese Postman tour of multidigraphs.
 * For more details, read <a
href="http://www.ucl.ac.uk/harold/cpp">http://www.ucl.ac.uk/harold/cpp</a>.
 *
 * @author Harold Thimbleby, 2001, 2, 3
 */

// Chinese Postman Code
// Harold Thimbleby, 2001-3

// <tex file="class.tex">
import java.io.*;
import java.util.*;
import java.lang.*;

public class CPP
{
    int N; // number of vertices
    int delta[]; // deltas of vertices
    int neg[], pos[]; // unbalanced vertices
    int arcs[][]; // adjacency matrix, counts arcs between vertices
    Vector label[][]; // vectors of labels of arcs (for each vertex pair)
    int f[][]; // repeated arcs in CPT
    double c[][]; // costs of cheapest arcs or paths
    String cheapestLabel[][]; // labels of cheapest arcs
    boolean defined[][]; // whether path cost is defined between vertices
    int path[][]; // spanning tree of the graph
    double basicCost; // total cost of traversing each arc once

    void solve()
    {
        leastCostPaths(0);
        checkValid();
        findUnbalanced();
        findFeasible();
        while( improvements() );
    }

    // <literal>${\vdots}$\vbox{ }</literal><null>
    // Other declarations are described below
    // <literal>${\vdots}$\</literal>
    // </tex><tex file="constructor.tex">
    // allocate array memory, and instantiate graph object
    CPP(int vertices)
```

```

{
    if( (N = vertices) <= 0 ) throw new Error("Graph is empty");
    delta = new int[N];
    defined = new boolean[N][N];
    label = new Vector[N][N];
    c = new double[N][N];
    f = new int[N][N];
    arcs = new int[N][N];
    cheapestLabel = new String[N][N];
    path = new int[N][N];
    basicCost = 0;
}

// </tex><tex file="addedge.tex">
CPP addArc(String lab, int u, int v, double cost)
{
    if( !defined[u][v] ) label[u][v] = new Vector();
    label[u][v].addElement(lab);
    basicCost += cost;
    if( !defined[u][v] || c[u][v] > cost )
    {
        c[u][v] = cost;
        cheapestLabel[u][v] = lab;
        defined[u][v] = true;
        path[u][v] = v;
    }
    arcs[u][v]++;
    delta[u]++;
    delta[v]--;
    return this;
}
//</tex>

/** Floyd-Warshall algorithm
 * Assumes no negative self-cycles.
 * Finds least cost paths or terminates on finding any non-trivial negative cycle.
 */
// <tex file="floyd.tex">
void leastCostPaths(int id)
{
    try{
        for( int k = 0; k < N; k++ )
            for( int i = 0; i < N; i++ )
                if( defined[i][k] )
                    for( int j = 0; j < N; j++ )
                        if( defined[k][j]
                            && (!defined[i][j] || c[i][j] > c[i][k]+c[k][j]) )
                        {
                            path[i][j] = path[i][k];
                            c[i][j] = c[i][k]+c[k][j];
                            defined[i][j] = true;
                        }
                    }
                }
            }
    }
}

```

```

negative cycle
if( i == j && c[i][j] < 0 ) return; // stop on
}
if (id == 0) {
    FileWriter outFile = new FileWriter("leastCost.txt");

    for( int i = 0; i < N; i++ )
    {
        for( int j = 0; j < N; j++ )
        {
            Double temp = new Double(c[i][j]);
            String result = temp.toString();
            outFile.write(result,0,result.length());
            outFile.write("\t",0,1);
        }
        outFile.write("\n",0,1);
    }

    outFile.close();
}

catch(java.io.IOException e)
{
    System.out.println(e);
}
catch(NumberFormatException ee)
{
    System.out.println(ee);
}

}

// </tex><tex file="check.tex">
void checkValid()
{
    for( int i = 0; i < N; i++ )
    {
        for( int j = 0; j < N; j++ )
            if( !defined[i][j] ) throw new Error(i+" "+j+"Graph is not strongly
connected");
            if( c[i][i] < 0 ) throw new Error("Graph has a negative cycle");
    }
}

// </tex><tex file="cost.tex">
double cost()
{
    return basicCost+phi();
}

double phi()

```

```

    {
        double phi = 0;
        for( int i = 0; i < N; i++ )
            for( int j = 0; j < N; j++ )
                phi += c[i][j]*f[i][j];
        return phi;
    }
//</tex><tex file="degrees.tex">

void findUnbalanced()
{
    int nn = 0, np = 0; // number of vertices of negative/positive delta

    for( int i = 0; i < N; i++ )
        if( delta[i] < 0 ) nn++;
        else if( delta[i] > 0 ) np++;

    neg = new int[nn];
    pos = new int[np];
    nn = np = 0;
    for( int i = 0; i < N; i++ ) // initialise sets
        if( delta[i] < 0 ) neg[nn++] = i;
        else if( delta[i] > 0 ) pos[np++] = i;
}
//</tex><tex file="greedy.tex">

void findFeasible()
{
    // delete next 3 lines to be faster, but non-reentrant
    int delta[] = new int[N];
    for( int i = 0; i < N; i++ )
        delta[i] = this.delta[i];

    for( int u = 0; u < neg.length; u++ )
    {
        int i = neg[u];
        for( int v = 0; v < pos.length; v++ )
        {
            int j = pos[v];
            f[i][j] = -delta[i] < delta[j]? -delta[i]: delta[j];
            delta[i] += f[i][j];
            delta[j] -= f[i][j];
        }
    }
}

// </tex><tex file="iterate.tex">
boolean improvements()
{
    CPP residual = new CPP(N);
    for( int u = 0; u < neg.length; u++ )
    {
        int i = neg[u];
        for( int v = 0; v < pos.length; v++ )
        {
            int j = pos[v];

```

```

        residual.addArc(null, i, j, c[i][j]);
        if( f[i][j] != 0 ) residual.addArc(null, j, i, -c[i][j]);
    }
}
residual.leastCostPaths(1); // find a negative cycle
for( int i = 0; i < N; i++ )
    if( residual.c[i][i] < 0 ) // cancel the cycle (if any)
    {
        int k = 0, u, v;
        boolean kunset = true;
        u = i; do // find k to cancel
        {
            v = residual.path[u][i];
            if( residual.c[u][v] < 0 && (kunset || k > f[v][u]) )
            {
                k = f[v][u];
                kunset = false;
            }
        } while( (u = v) != i );
        u = i; do // cancel k along the cycle
        {
            v = residual.path[u][i];
            if( residual.c[u][v] < 0 ) f[v][u] -= k;
            else f[u][v] += k;
        } while( (u = v) != i );
        return true; // have another go
    }
return false; // no improvements found
}

// </tex><tex file="printCPT.tex">
static final int NONE = -1; // anything < 0

int findPath(int from, int f[][] ) // find a path between unbalanced vertices
{
    for( int i = 0; i < N; i++ )
        if( f[from][i] > 0 ) return i;
    return NONE;
}

void printCPT(int startVertex)
{
    try{
        int v = startVertex;

        // delete next 7 lines to be faster, but non-reentrant
        int arcs[][] = new int[N][N];
        int f[][] = new int[N][N];
        for( int i = 0; i < N; i++ )
            for( int j = 0; j < N; j++ )
            {
                arcs[i][j] = this.arcs[i][j];
                f[i][j] = this.f[i][j];
            }
    }
}

```

```

PrintWriter outFile = new PrintWriter("CPT.txt");

while( true )
{
    int u = v;
    if( (v = findPath(u, f)) != NONE )
    {
        f[u][v]--; // remove path
        for( int p; u != v; u = p ) // break down path into its arcs
        {
            p = path[u][v];
            outFile.println("Take arc "+cheapestLabel[u][p]
                +" from "+u+" to "+p);
            //System.out.println("Take arc "+cheapestLabel[u][p]
            //    +" from "+u+" to "+p);
        }
    }
    else
    {
        int bridgeVertex = path[u][startVertex];
        if( arcs[u][bridgeVertex] == 0 )
            break; // finished if bridge already used
        v = bridgeVertex;
        for( int i = 0; i < N; i++ ) // find an unused arc, using bridge last
            if( i != bridgeVertex && arcs[u][i] > 0 )
            {
                v = i;
                break;
            }
        arcs[u][v]--; // decrement count of parallel arcs

        outFile.println("Take arc "+label[u][v].elementAt(arcs[u][v])
            +" from "+u+" to "+v); // use each arc label in turn
        //System.out.println("Take arc "+label[u][v].elementAt(arcs[u][v])
        //    +" from "+u+" to "+v); // use each arc label in turn
    }
}
outFile.close();
}
catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}
}
// </tex>

static public void main(String args[])
{
    try{

        String[] list = new String[3];

```

```

Integer temp;
String clas, inputf;
int dpt;

for ( int index = 0;(index < args.length) && ( index < 6 ); index++)
{
    list[index] = args[index];
}

if (list[0].compareTo("-help")==0)
{
    System.out.print("Command Line: java CPP class depot inputf\n");
    System.out.print("class-the class of the roads\n");
    System.out.print("depot-assigned depot\n");
    System.out.print("inputf-input file name\n");
    return;
}
else
{
    clas = list[0];

    temp = new Integer(list[1]);
    dpt = temp.intValue();

    inputf = list[2];
}

```

```

String idno;
String alphaid, cls, direction;
int from;
int to;
double cost, stime;

int total = 0;

int maxnode=0;

Scanner inFile0 = new Scanner(new FileReader(inputf));

for(; inFile0.hasNextLine(); inFile0.nextLine())
{
    idno = inFile0.next();
    alphaid = inFile0.next();
    from = inFile0.nextInt();

    if(maxnode<from) maxnode=from;
}

```

```

inFile0.close();

int chk[];
chk = new int[maxnode+1];

for(int i=0;i<maxnode+1;i++) chk[i]=0;

CPP G = new CPP(maxnode+1); // create a graph of four vertices

Scanner inFile = new Scanner(new FileReader(inputf));

for(; inFile.hasNextLine(); inFile.nextLine())
{
idno = inFile.next();
alphaid = inFile.next();
from = inFile.nextInt();
to = inFile.nextInt();
cost = inFile.nextDouble();
stime = inFile.nextDouble();
direction = inFile.next();
cls = inFile.next();

chk[from]=1;
chk[to]=1;

if (cls.compareTo("A1")!=0 && cls.compareTo("A1")==0 ) stime=cost*1.2;
if (cls.compareTo("A2")!=0 && cls.compareTo("A2")==0 ) stime=cost*1.5;
if (cls.compareTo("A3")!=0 && cls.compareTo("A3")==0 ) stime=cost*1.5;
if (cls.compareTo("A4")!=0 && cls.compareTo("A4")==0 ) stime=cost*1.5;

G.addArc(alphaid, from, to, cost);

}
inFile.close();

G.addArc("Vitual", 0, dpt, 0);
G.addArc("Vitual", dpt, 0, 0);

for(int i=1;i<maxnode;i++)
{
    if(chk[i]==0)
    {
        G.addArc("Vitual", 0, i, 0);
        G.addArc("Vitual", i, 0, 0);
    }
}

```

```

/*      CPP G = new CPP(5); // create a graph of four vertices

      // add the arcs for the example graph
      G.addArc("a", 0, 1, 1).addArc("b", 1, 4, 1).addArc("c", 4, 3, 1)
        .addArc("d", 3, 0, 1).addArc("e", 0, 2, 1).addArc("f", 2, 4, 1)
        .addArc("g", 1, 2, 1).addArc("h", 2, 3, 1).addArc("i", 0, 4, 1)
        .addArc("j", 3, 1, 1);

*/

      System.out.println("//<tex file=\"output.tex\">");
//<tex file="test.tex">

      G.solve(); // find the CPT
      G.printCPT(0); // print it, starting from vertex 0
      System.out.println("Cost = "+(double)(int)(G.cost()*1000)/1000);
//</tex>

      System.out.println("//</tex>");
//      OpenCPP.test();
      }

      catch(java.io.FileNotFoundException e)
      {
      System.out.println(e);
      }
      }

// Print arcs and f
void debugarcf()
{
    for( int i = 0; i < N; i++ )
    {
        System.out.print("f["+i+"]= ");
        for( int j = 0; j < N; j++ )
            System.out.print(f[i][j]+" ");
        System.out.print(" arcs["+i+"]= ");
        for( int j = 0; j < N; j++ )
            System.out.print(arcs[i][j]+" ");
        System.out.println();
    }
}

// Print out most of the matrices: defined, path and f
void debug()
{
    for( int i = 0; i < N; i++ )
    {
        System.out.print(i+" ");
        for( int j = 0; j < N; j++ )
            System.out.print(j+": "+(defined[i][j]?"T":"F")+ " "+
                c[i][j]+" p="+path[i][j]+" f="+f[i][j]+" ");
        System.out.println();
    }
}

```

```

// Print out non zero f elements, and phi
void debugf()
{
    double sum = 0;
    for( int i = 0; i < N; i++ )
    {
        boolean any = false;
        for( int j = 0; j < N; j++ )
            if( f[i][j] != 0 )
                {
                    any = true;

System.out.print("f("+i+", "+j+": "+label[i][j]+")="+f[i][j]+"@"+c[i][j]+" ");
                    sum += f[i][j]*c[i][j];
                }
        if( any )
            System.out.println();
    }
    System.out.println("-->phi="+sum);
}

// Print out cost matrix.
void debugc()
{
    for( int i = 0; i < N; i++ )
    {
        boolean any = false;
        for( int j = 0; j < N; j++ )
            if( c[i][j] != 0 )
                {
                    any = true;

System.out.print("c("+i+", "+j+": "+label[i][j]+")="+c[i][j]+" ");
                }
        if( any )
            System.out.println();
    }
}
}

```

AssignDpt.JAVA

```

import java.io.*;
import java.util.*;

public class AssignDpt
{
    int ARCNO = 370;
    int D[], rD[];
}

```

```

int AssignDpt;
double AssignDist;

String arcsD[][];

AssignDpt(int N)
{
    D = new int[N];
    rD = new int[N-1];
    arcsD = new String[ARCNO][8];
}

AssignDpt InitD(int N) throws IOException
{
    int i,j;

    for(i=0;i<N;i++)
    {
        InputStreamReader reader = new InputStreamReader(System.in);
        BufferedReader input = new BufferedReader(reader);
        System.out.print("Enter Depot "+i+": ");

        String t1 = input.readLine();
        Integer t2 = new Integer(t1);
        D[i]=t2.intValue();

    }

    return this;
}

void DoAssign(int dptno, String Netfile)
{
    try{

        String idno;
        String alphaid, cls, direction, acls;
        int from;
        int to;
        double cost, stime, dist, wcost, wdist, totalwd;
        int dpt, Sfre=1;
        Double tempD;
        int i=0;

        System.out.println("Reading Network Files...");

        Scanner inFile = new Scanner(new FileReader(Netfile));

```

```

totalwd=0;

for(i=0; inFile.hasNextLine(); inFile.nextLine())
{
    //System.out.println("Arc"+i);
    idno = inFile.next();
    alphaid = inFile.next();
    from = inFile.nextInt();
    to = inFile.nextInt();
    cost = inFile.nextDouble();
    stime = inFile.nextDouble();
    direction = inFile.next();
    cls = inFile.next();

    if (cls.compareTo("A1")==0 || cls.compareTo("A2")==0 ) Sfre = 6;
    if (cls.compareTo("A3")==0 ) Sfre = 2;
    if (cls.compareTo("A4")==0 ) Sfre = 1;

    nearestD(from, to, dptno, dptno);

    arcsD[i][0] = idno;
    arcsD[i][1] = alphaid;
    arcsD[i][2] = (new Integer(from)).toString();
    arcsD[i][3] = (new Integer(to)).toString();
    arcsD[i][4] = (new Double(cost)).toString();
    arcsD[i][5] = (new Integer(Sfre)).toString();
    arcsD[i][6] = (new Integer(AssignDpt)).toString();
    arcsD[i][7] = (new Double(AssignDist)).toString();

    totalwd = totalwd + (Sfre*AssignDist/2);
    i++;
}

inFile.close();

}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}
}

void AfterAssign(int dptno, int rmdpt, String Outputfile)
{

```

```

try{

int from, to, i, j, k, Sfre=1, rdpt=D[0];
double dist, cost, wcost, wdist;
Integer temp, tdpt;
Double temp1;
double RecD[][];
RecD = new double[dptno][3];

for(i=0;i<dptno;i++)
    for(j=0;j<3;j++)
        RecD[i][j]=0;

PrintWriter outFile = new PrintWriter(Outputfile);

    outFile.println("# ID\tAlpha ID\tService Frequency\tArc Length\tNearest
Depot\tDistance to Depot(sum)\tWeighted Length\tWeighted Distance");

PrintWriter outFileD = new PrintWriter("D"+Outputfile);

    outFileD.println("Depot #\t# of Arcs Assigned\tWeighted Length\tWeighted
Distance");

if(rmdpt < dptno) rdpt = D[rmdpt];

for(k=rmdpt;k<dptno-1;k++)
    D[k]=D[k+1];

for(i=0; i<ARCNO; i++)
{
    tdpt = (new Double(arcsD[i][6])).intValue();
    if(rmdpt < dptno)
    if(tdpt == rdpt)
    {
        temp = new Integer(arcsD[i][2]);
        from = temp.intValue();

        temp = new Integer(arcsD[i][3]);
        to = temp.intValue();

        nearestD(from, to, dptno-1, dptno-1);

        arcsD[i][6] = (new Double(AssignDpt)).toString();
        arcsD[i][7] = (new Double(AssignDist)).toString();

    }

    temp = new Integer(arcsD[i][5]);

```

```

        Sfre = temp.intValue();

        temp1 = new Double(arcsD[i][4]);
        cost = temp1.doubleValue();

        temp1 = new Double(arcsD[i][7]);
        dist = temp1.doubleValue();

        wcost = cost*Sfre;
        wdist = (dist*Sfre)/2;

        outFile.println(arcsD[i][0] + "\t" + arcsD[i][1] + "\t" + Sfre + "\t" + cost +
"\t" + arcsD[i][6] + "\t" + dist
                                + "\t" + wcost + "\t" + wdist);

        tdpt = (new Double(arcsD[i][6])).intValue();
        for(j=0;j<dptno;j++)
        {
            if(tdpt==D[j])
            {
                RecD[j][0]=RecD[j][0]+1;
                RecD[j][1]=RecD[j][1]+wcost;
                RecD[j][2]=RecD[j][2]+wdist;
            }
        }
    }

    outFile.close();

    for(j=0;j<dptno-1;j++)
    {
        outFileD.println(D[j] + "\t" + RecD[j][0] + "\t" + RecD[j][1] + "\t" +
RecD[j][2]);
    }

    if(rmdpt==dptno)
        outFileD.println(D[j] + "\t" + RecD[j][0] + "\t" + RecD[j][1] + "\t" +
RecD[j][2]);

    outFileD.close();

}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}
}

```

```

public static void main(String[] args)
{
    try{

        String[] list = new String[2];

        String Netfile, Outputfile;
        int maxdptno, dptno, sdpt, i;
        Integer temp00;

        for ( int index = 0;(index < args.length) && ( index < 2 ); index++)
        {
            list[index] = args[index];
        }

        if (list[0].compareTo("-help")==0)
        {
            System.out.print("Command Line: java FindDpt Netfile NOofDepot\n");
            System.out.print("Netfile-input Network file name\n");
            System.out.print("NOofDepot-Number of Depots\n");
            return;
        }
        else
        {
            Netfile = list[0];

            temp00 = new Integer(list[1]);
            maxdptno = temp00.intValue();
        }

        dptno = maxdptno;

        AssignDpt AD = new AssignDpt(dptno);

        AD.InitD(dptno);

        AD.DoAssign(dptno, Netfile);
        AD.AfterAssign(dptno, dptno, "Arc00.txt");

    }

    catch(java.io.FileNotFoundException e)
    {
        System.out.println(e);
    }
    catch(java.io.IOException e)

```

```

        {
        System.out.println(e);
        }
    }

double LeastCost(int from, int to)
{
    double result=0;
    try{
    if (from == to) return 0;
    Scanner costFile = new Scanner(new FileReader("LeastDistance.txt"));
    int i,j;

    for(i=0; i<=from; costFile.nextLine(), i++)
    {
        for(j=0; j<=to; j++)
        {
            result = costFile.nextDouble();
        }
    }

    catch(java.io.FileNotFoundException e)
    {
    System.out.println(e);
    }

    return result;
}

void nearestD(int from, int to, int dptno, int rmdpt)
{
    int i,j, sd;
    double temp1, temp2;
    double result=0, sdist, dist;

    sd=0;
    sdist=9999;

    if(rmdpt>dptno-1)
    {
        for(i=0;i<dptno;i++)
        {
            dist=LeastCost(from, D[i]) + LeastCost(to, D[i]);

            if(sdist>dist)
            {

```

```

                sd=i;
                sdist=dist;
            }
        }

        AssignDpt=D[sd];
        AssignDist=sdist;
    }
    else
    {
        for(i=0;i<dptno-1;i++)
        {
            dist=LeastCost(from, rD[i]) + LeastCost(to, rD[i]);

            if(sdist>dist)
            {
                sd=i;
                sdist=dist;
            }
        }

        AssignDpt=rD[sd];
        AssignDist=sdist;
    }
}
}
}

```

TourDiv2.JAVA

```

import java.io.*;
import java.util.*;

public class TourDiv2
{
    public static void main(String[] args) throws IOException
    {

        try{

            String[] list = new String[7];

            Integer temp00;
            String clas, CPTfile, Netfile;
            int dptN, N, MAXMILE, MAXTIME;

```

```

for ( int index = 0;(index < args.length) && ( index < 7 ); index++)
{
    list[index] = args[index];
}

if (list[0].compareTo("-help")==0)
{
    System.out.print("Command Line: java TourDiv class N MAXMILE
MAXTIME DepotN CPTfile Netfile\n");
    System.out.print("class-the class of the roads\n");
    System.out.print("N-total arc number\n");
    System.out.print("MAXMILE-Maximum distance for this class\n");
    System.out.print("MAXTIME-Maximum time for this class\n");
    System.out.print("DepotN-How many depots assigned\n");
    System.out.print("CPTfile-input CPT file name\n");
    System.out.print("Netfile-input Network file name\n");
    return;
}
else
{
    clas = list[0];

    temp00 = new Integer(list[1]);
    N = temp00.intValue();

    temp00 = new Integer(list[2]);
    MAXMILE = temp00.intValue();

    temp00 = new Integer(list[3]);
    MAXTIME = temp00.intValue();

    temp00 = new Integer(list[4]);
    dptN = temp00.intValue();

    CPTfile = list[5];

    Netfile = list[6];
}

int D[];
D = new int[dptN];

int q;
for(q=0;q<dptN;q++)
{
    InputStreamReader reader = new InputStreamReader(System.in);
    BufferedReader input = new BufferedReader(reader);

```

```

        System.out.print("Enter Assigned Depot "+q+": ");

        String t1 = input.readLine();
        Integer t2 = new Integer(t1);
        D[q]=t2.intValue();

        System.out.println("Assigned Depot "+q+" is: "+D[q]);
    }

    double dbtime[], detime[];
    dbtime = new double[dptN]; //depot to beginning point time
    detime = new double[dptN]; //ending point to depot time

    int dpt=0;

    String idno;
    String alphaid;
    int from;
    int to;
    String cls, direction;
    double cost,stime;
    String net[[[]], cpt[[[]], tt[[[]];
    net = new String[N][4]; //0 - Alpha ID; 1 - Distance; 2 - Service Time; 3 - Class
    cpt = new String[N][6]; //0 - Alpha ID; 1 - From; 2 - To;
                                //3 - Distance; 4 - Service Time; 5 - Class
    tt = new String[1][6];
    double totalS=0, totalF=10000;

    int n,k,i;

    double totalD = 0;
    double totalT = 0;
    double BeginT = 0;
    double EndT = 0;

    Scanner netFile = new Scanner(new FileReader(Netfile));

    for(n=0; netFile.hasNextLine(); netFile.nextLine(),n++)
    {
        idno = netFile.next();
        alphaid = netFile.next();
        from = netFile.nextInt();
        to = netFile.nextInt();
        cost = netFile.nextDouble();
        stime = netFile.nextDouble();
        direction = netFile.next();
        cls = netFile.next();
    }

```

```

if (clas.compareTo("A1")!=0 && cls.compareTo("A1")==0 ) stime=cost*1.2;
if (clas.compareTo("A2")!=0 && cls.compareTo("A2")==0 ) stime=cost*1.5;
if (clas.compareTo("A3")!=0 && cls.compareTo("A3")==0 ) stime=cost*1.5;
if (clas.compareTo("A4")!=0 && cls.compareTo("A4")==0 ) stime=cost*1.5;

```

```

Double temp = new Double(cost);
String result = temp.toString();

```

```

Double temp0 = new Double(stime);
String result0 = temp0.toString();

```

```

net[n][0] = alphaid;
net[n][1] = result;
net[n][2] = result0;
net[n][3] = cls;
}
netFile.close();

```

```

Scanner cptFile = new Scanner(new FileReader(CPTfile));

```

```

for(n=0; cptFile.hasNextLine(); cptFile.nextLine(),n++)

```

```

{
String str1 = cptFile.next();
String str2 = cptFile.next();
String str3 = cptFile.next();
String str4 = cptFile.next();
String str5 = cptFile.next();
String str6 = cptFile.next();
String str7 = cptFile.next();

```

```

cpt[n][0] = str3;
cpt[n][1] = str5;
cpt[n][2] = str7;
}
cptFile.close();

```

```

for(n=0;n<N;n++)

```

```

{
for(k=0;k<N;k++)
{
//if(n==1 && k==1)System.out.println("haha"+cpt[n][0].length()+", "+

```

```

net[k][0].length());

```

```

if (cpt[n][0].compareTo(net[k][0])==0)

```

```

{
//System.out.println("haha"+cpt[n][0]+", "+ net[k][0]);
cpt[n][3]=net[k][1];
cpt[n][4]=net[k][2];

```

```

        cpt[n][5]=net[k][3];
    }
}

int m,s,c,finalc=0;

PrintWriter FFile = new PrintWriter("Final.txt");

for(c=0;c<N;c++)
{
    PrintWriter outFile = new PrintWriter(c+"DTour.txt");
    PrintWriter outFile3 = new PrintWriter(c+"TourSum.txt");

    outFile.println("Route\tArc\tFrom\tTo\tMiles\tTime\tClass");

outFile3.println("Route\tClass\tWeight\tTime\tDistance\tDepot\tDeadhead");

    totalS=0;
    totalD=0;
    totalT=0;

    for(n=0,i=1,k=0;n<N;n++)
    {
        if(k!=i)
        {
            if (cpt[n][5].compareTo(clas)!=0) continue;

System.out.println("=====
=====");

            System.out.println("Tour " + i + " begins: ");

            k=i;
            Integer from1 = new Integer(cpt[n][1]);

            for(q=0;q<dptN;q++)
            {
                dbtime[q] = LeastTime(D[q], from1.intValue());
            }

        }

        Integer from1 = new Integer(cpt[n][1]);
        Integer to1 = new Integer(cpt[n][2]);

```

```

Double tempD = new Double(cpt[n][3]);
Double tempT = new Double(cpt[n][4]);

if (cpt[n][5].compareTo(clas)==0)
{
    totalD = totalD + tempD.doubleValue();
}

totalT = totalT + tempT.doubleValue();

for(q=0;q<dptN;q++)
{
    detime[q] = LeastTime(to1.intValue(), D[q]);
}

double tempt = dbtime[0]+detime[0];
dpt = D[0];
for(q=1;q<dptN;q++)
{
    if (dbtime[q]+detime[q] < tempt)
    {
        dpt = D[q];
        tempt = dbtime[q]+detime[q];
    }
}

if (totalD>MAXMILE || (totalT + tempt)>MAXTIME)
{
    totalD = totalD - tempD.doubleValue();
    totalT = totalT - tempT.doubleValue();
    n--;

    for(q=0;q<dptN;q++)
    {
        detime[q] = LeastTime(from1.intValue(), D[q]);
    }

    double tempt1 = dbtime[0]+detime[0];
    dpt = D[0];
    for(q=1;q<dptN;q++)
    {
        if (dbtime[q]+detime[q] < tempt1)
        {
            dpt = D[q];
            tempt1 = dbtime[q]+detime[q];
        }
    }
}

```

```

    }

    totalS = totalS + tempt1;

    System.out.println("Tour " + i + " ends. Total Miles:
"+totalD);

    System.out.println("Tour " + i + " ends. Total Time:
"+totalT+ " Deadhead: "+ tempt1);

    System.out.println("=====
=====");

    double tottime = totalT + tempt1;
    outFile3.println("R"+ i + "\t"+clas + "\t\t" + tottime + "\t"
+ totalD + "\t" + dpt + "\t" + tempt1);

    i++;
    totalD=0;
    totalT=0;
}
else
{
    System.out.println("Take arc "+cpt[n][0]+" from
"+cpt[n][1]+" to "+cpt[n][2]+" - "+ cpt[n][3]+" miles"+" - "+ cpt[n][4]+" mins");

    outFile.println(i + "\t" + cpt[n][0] + "\t" + cpt[n][1] + "\t" +
cpt[n][2] + "\t" + cpt[n][3]
+ "\t" + cpt[n][4] + "\t" + cpt[n][5]);

    if(n==N-1)
    {
        totalS = totalS + tempt;

        System.out.println("Tour " + i + " ends. Total Miles:
"+totalD);

        System.out.println("Tour " + i + " ends. Total Time:
"+totalT+ " Deadhead: "+ tempt);

        System.out.println("=====
=====");

        double tottime = totalT + tempt;
        outFile3.println("R"+ i + "\t"+clas + "\t\t" + tottime + "\t"
+ totalD + "\t" + dpt + "\t" + tempt);
    }
}
}

```

```

    }

    FFile.println("Round "+c+" TotalScore: "+totalS);

    if (totalS < totalF)
    {
        finalc = c;
        totalF = totalS;
    }

    outFile.close();
    outFile3.close();

    for(m=0;m<6;m++)
        tt[0][m]=cpt[0][m];

    for(s=1;s<N;s++)
    {
        for(m=0;m<6;m++)
            cpt[s-1][m]=cpt[s][m];
    }

    for(m=0;m<6;m++)
        cpt[N-1][m]=tt[0][m];

}

FFile.println("Final result is: " + finalc + " Total Score is: "+totalF);
System.out.println("Final result is: " + finalc + " Total Score is: "+totalF);
FFile.close();

}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}

}

static double LeastTime(int from, int to)
{
    double result=0;
    try{
        if (from == to) return 0;
    }

```

```

Scanner costFile = new Scanner(new FileReader("leastNTime.txt"));
int i,j;

for(i=0; i<=from; costFile.nextLine(), i++)
{
    for(j=0; j<=to; j++)
    {
        result = costFile.nextDouble();
    }
}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}

return result;
}
}

```

AddDH.JAVA

```

import java.io.*;
import java.util.*;

public class AddDH
{
    int D[];

    String route[][][];
    String net[][];

    // allocate array memory
    AddDH(int N, int M)
    {
        D = new int[N];
        route = new String[N][M][7];
        net = new String[500][6];
    }

    AddDH addNet(int n, String str0, String str1, String str2, String str3, String
str5)
    {

```

```

        net[n][0] = str0;
        net[n][1] = str1;
        net[n][2] = str2;
        net[n][3] = str3;
        net[n][4] = str4;
        net[n][5] = str5;

        return this;
    }

```

AddDH addRoute(int n, String str0, String str1, String str2, String str3, String str4, String str5, String str6)

```

    {
        Integer temp = new Integer(str0);
        int rno = temp.intValue()-1;

        route[rno][n][0] = str0;    //Route No.
        route[rno][n][1] = str1;    //Alpha ID
        route[rno][n][2] = str2;    //From
        route[rno][n][3] = str3;    //To
        route[rno][n][4] = str4;    //Distance
        route[rno][n][5] = str5;    //Service Time
        route[rno][n][6] = str6;    //Class

        return this;
    }

```

AddDH InitDnR(int N, int M) throws IOException

```

    {
        int i,j;

        for(i=0;i<N;i++)
        {
            InputStreamReader reader = new InputStreamReader(System.in);
            BufferedReader input = new BufferedReader(reader);
            System.out.print("Enter Assigned Depot for Route "+i+": ");

            String t1 = input.readLine();
            Integer t2 = new Integer(t1);
            D[i]=t2.intValue();

            System.out.println("Assigned Depot for Route "+i+" is: "+D[i]);

        }

        for(i=0;i<N;i++)    //initialize the route array
        {
            for(j=0;j<M;j++)

```

```

        route[i][j][0]= new String("00");
    }

    for(i=0;i<500;i++)          //initialize the route array
    {
        net[i][0]= new String("00");
    }

    return this;
}

public static void main(String[] args)
{

    try{

        String[] list = new String[4];

        Integer temp00;
        String inputfile, Netfile;
        int N, M;

        for ( int index = 0;(index < args.length) && ( index < 4 ); index++)
        {
            list[index] = args[index];
        }

        if (list[0].compareTo("-help")==0)
        {
            System.out.print("Command Line: java AddDH N M inputfile Netfile\n");
            System.out.print("N-Maximum route number in this class\n");
            System.out.print("M-maximum arc number within a route\n");
            System.out.print("inputfile-input file name\n");
            System.out.print("Netfile-input Network file name\n");
            return;
        }
        else
        {
            temp00 = new Integer(list[0]);
            N = temp00.intValue();

            temp00 = new Integer(list[1]);
            M = temp00.intValue();

            inputfile = list[2];

            Netfile = list[3];

```

```

}

int i,j,k,h,n,m,r;

String idno;
String alphaid;
String from0;
String to0;
String cls;
String cost,stime;
double ttime;

AddDH AD = new AddDH(N, M);

AD.InitDnR(N, M);

Scanner netFile0 = new Scanner(new FileReader(Netfile));

for(n=0; netFile0.hasNextLine(); netFile0.nextLine(),n++)
{
    idno = netFile0.next();
    alphaid = netFile0.next();
    from0 = netFile0.next();
    to0 = netFile0.next();
    cost = netFile0.next();
    stime = netFile0.next();
    cls = netFile0.next();

    Double t1 = new Double(cost);
    double t2 = t1.doubleValue();

    ttime = t2;
    if (cls.compareTo("A1")==0 ) ttime=t2* 1.2;
    if (cls.compareTo("A2")==0 ) ttime=t2* 1.5;
    if (cls.compareTo("A3")==0 ) ttime=t2* 1.5;
    if (cls.compareTo("A4")==0 ) ttime=t2* 1.5;

    Double tstime = new Double(ttime);
    String rstime = tstime.toString();

    AD.addNet(n, alphaid, from0, to0, cost, rstime, cls);
}
netFile0.close();

Scanner tourFile = new Scanner(new FileReader(inputfile));

String prno = "1";
int pto=0;

```

```

for(n=0; tourFile.hasNextLine(); tourFile.nextLine(),n++)
{
    String str0 = tourFile.next();
    String str1 = tourFile.next();
    String str2 = tourFile.next();
    String str3 = tourFile.next();
    String str4 = tourFile.next();
    String str5 = tourFile.next();
    String str6 = tourFile.next();

    Integer temp = new Integer(str2);
    int from = temp.intValue();

    if(str0.compareTo(prno)!=0)
    {
        AD.AddDeadhead(N, str0, n, pto, 0, 0);
        n=0;
    }

    if(n==0 || pto!=from) n=AD.AddDeadhead(N, str0, n, pto, from, 0);

    Integer temp1 = new Integer(str3);
    int to = temp1.intValue();

    pto = to;

    prno = str0;

    AD.addRoute(n, str0, str1, str2, str3, str4, str5, str6);
}

AD.AddDeadhead(N, prno, n, pto, 0, 1);

tourFile.close();

AD.output("TourWithDH.txt", N);

}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}
catch(java.io.IOException e)

```

```

        {
        System.out.println(e);
        }
    }

int AddDeadhead(int N, String str0, int n, int p, int q, int ind)
{
    String result="", arcname;

    int strlen=0, pos=1, c=0;

    Integer temp1 = new Integer(str0);
    int rno = temp1.intValue()-1;

    if(n==0 && q!=D[rno])
    {
        result = LeastPath(D[rno], q);
        System.out.println("from "+D[rno]+" to "+q+" : "+ result); //insert DH
        strlen = result.length();
        for(int i=0;i<strlen;i++)
        {
            if(i!=0 && result.charAt(i)==' ')
            {
                arcname = result.substring(pos,i);

                System.out.println(arcname);
                pos = i+1;

                for(c=0; net[c][0].compareTo("00")!=0; c++)
                {
                    if(net[c][0].compareTo(arcname)==0)
                    {
                        System.out.println("Add Arc: " + arcname);
                        addRoute(n, str0, net[c][0], net[c][1],
net[c][2], net[c][3], net[c][4], net[c][5]+"-D");
                        n=n+1;
                        break;
                    }
                }
            }
        }

        return n;
    }

    if(q==0)

```

```

{
    if(ind==0 && p!=D[rno-1])
    {
        result = LeastPath(p, D[rno-1]);
        System.out.println("from "+p+" to "+D[rno-1]+ " : "+ result);
        strlen = result.length();
        for(int i=0;i<strlen;i++)
        {
            if(i!=0 && result.charAt(i)=='/')
            {
                arcname = result.substring(pos,i);

                //System.out.println(arcname);
                pos = i+1;

                for(c=0; net[c][0].compareTo("00")!=0; c++)
                {
                    if(net[c][0].compareTo(arcname)==0)
                    {
                        System.out.println("Add Arc: " +
arcname);

                        Integer tt = new Integer(rno);
                        addRoute(n, tt.toString(), net[c][0],
net[c][1], net[c][2], net[c][3], net[c][4], net[c][5]+"-D");
                        n=n+1;
                        break;
                    }
                }
            }
        }
    }
}

if(ind==1 && p!=D[rno])
{
    result = LeastPath(p, D[rno]);
    System.out.println("from "+p+" to "+D[rno]+ " : "+ result);
    strlen = result.length();
    for(int i=0;i<strlen;i++)
    {
        if(i!=0 && result.charAt(i)=='/')
        {
            arcname = result.substring(pos,i);

            //System.out.println(arcname);
            pos = i+1;

            for(c=0; net[c][0].compareTo("00")!=0; c++)

```

```

arcname);
net[c][2], net[c][3], net[c][4], net[c][5]+"-D");
        {
            if(net[c][0].compareTo(arcname)==0)
            {
                System.out.println("Add Arc: " +
                    addRoute(n, str0, net[c][0], net[c][1],
                        n=n+1;
                        break;
                    }
            }
        }
    }
}
return n;
}

if(n!=0 && q!=0 && p!=q)
{
    result = LeastPath(p, q);
    System.out.println("from "+p+" to "+q+" : "+ result);
    strlen = result.length();
    for(int i=0;i<strlen;i++)
    {
        if(i!=0 && result.charAt(i)=='')
        {
            arcname = result.substring(pos,i);

            //System.out.println(arcname);
            pos = i+1;

            for(c=0; net[c][0].compareTo("00")!=0; c++)
            {
                if(net[c][0].compareTo(arcname)==0)
                {
                    System.out.println("Add Arc: " + arcname);
                    addRoute(n, str0, net[c][0], net[c][1],
                        net[c][2], net[c][3], net[c][4], net[c][5]+"-D");
                    n=n+1;
                    break;
                }
            }
        }
    }
}
return n;
}
}

```

```

        return n;
    }

void output(String filename, int N)
{
    try{
        PrintWriter outFile = new PrintWriter(filename);

        outFile.println("Route\tArc\tFrom\tTo\tMiles\tTime\tClass");

        for(int i=0;i<N;i++)
        {
            for(int j=0;route[i][j][0].compareTo("00")!=0;j++)
            {
                outFile.println(route[i][j][0] + "\t" + route[i][j][1] + "\t" +
route[i][j][2] + "\t" + route[i][j][3]
                                + "\t" + route[i][j][4] + "\t" + route[i][j][5] + "\t" +
route[i][j][6]);
            }
        }

        outFile.close();

    }

    catch(java.io.FileNotFoundException e)
    {
        System.out.println(e);
    }
}

String LeastPath(int from, int to)
{
    String result=new String("");
    try{
        if (from == to) return "";

        Scanner costFile = new Scanner(new FileReader("leastPath.txt"));
        int i,j;

        for(i=0; i<=from; costFile.nextLine(), i++)
        {
            for(j=0; j<=to; j++)
            {
                result = costFile.next();
            }
        }
    }
}

```

```

        }
    }
}

catch(java.io.FileNotFoundException e)
{
    System.out.println(e);
}

return result;
}

void outputnet(String filename)
{
    try{
        PrintWriter outFile = new PrintWriter(filename);

        outFile.println("Route\tArc\tFrom\tTo\tMiles\tTime\tClass");

        for(int i=0;net[i][0].compareTo("00")!=0;i++)
        {
            net[i][3] +
                outFile.println("0\t"+net[i][0)+"\t" + net[i][1] + "\t" + net[i][2] + "\t" +
                "\t" + net[i][4] + "\t" + net[i][5]);
        }

        outFile.close();

    }

    catch(java.io.FileNotFoundException e)
    {
        System.out.println(e);
    }
}

}

```