Track Inspection Planning and Risk Measurement Analysis

Prepared by
Dr. Dincer Konur
Dr. Suzanna Long
Dr. Ruwen Qin
Dr. Curt Elmore
Hadi Farhangi
Missouri University of Science and Technology, Engineering Management and Systems Engineering Department

Final Report Prepared for Missouri Department of Transportation
2015 January Project TR201409 Report cmr15-005

TECHNICAL REPORT DOCUMENTATION PAGE.

1. Report No.: cmr 15-005	2. Government Accession No.:	3. Recipient's Catalog No.:
4. Title and Subtitle: Track Inspection Planning and Risk Measurement Analysis		5. Report Date: 11/2014
		6. Performing Organization Code:
7. Author(s): Dincer Konur, Hadi Farhangi, Suzanna Long, Ruwen Qin, Curt Elmore		8. Performing Organization Report No.:
9. Performing Organization Name and Address: Missouri University of Science and Technology 1870 Miner Circle Rolla, MO 65409		10. Work Unit No.:
		11. Contract or Grant No.: TR201409
12. Sponsoring Agency Name and Address: Missouri Department of Transportation Research, Development and Technology PO BOX 270, Jefferson City, MO 65102		13. Type of Report and Period Covered: Final Report, 08/01/2013-11/14/2014
		14. Sponsoring Agency Code: MoDOT
15. Supplementary Notes: The investigation was conducted in cooperation with the U. S. Department of Transportation, Federal Highway Administration.		

16. Abstract: This project models track inspection operations on a railroad network and discusses how the inspection results can be used to measure the risk of failure on the tracks. In particular, the inspection times of the tracks, inspection frequency of the tracks, and times between consecutive inspections on the same tracks should be considered for scheduling inspections on the railroad tracks. Furthermore, an inspection plan should schedule inspections considering the characteristics of different tracks. Therefore, it is important to schedule track inspections such that the potential defects are captured as much as possible within minimum times to increase safety. The project formulates a mathematical optimization problem for the track inspection planning considering the practical settings of track inspection operations such as inspection times, inspection frequencies required, time between consecutive inspections, and importance of distinct tracks. The two objectives simultaneously captured in this model are minimization of total inspection times and maximization of the weighted inspections. An efficient solution method is proposed for solving this model. The solution method is compared to a scheduling procedure, which can be used in absence of the findings in this project, on a set of railroad track networks of different sizes. Based on the comparison, the solution method proposed proves to find improved inspection schedules regardless of the railroad network size. A review of the techniques on how to use the inspection results to measure risk of failure is provided.

17. Key Words: Track Inspection, Scheduling, Track Risk	18. Distribution Statement: No restrictions. This document is available to the public through National Technical Information Center, Springfield, Virginia 22161.

19. Security Classification (of this report): Unclassified.	20. Security Classification (of this page): Unclassified.	21. No of Pages:119	22. Price:

Form DOT F 1700.7 (06/98).

Executive Summary

One of the most important railroad safety operations that State Departments of Transportation and/or railroad companies need to plan is the inspection of railroad tracks. This project uses mathematical modeling and optimization approaches to analyze the track inspection operations on a railroad network and discusses possible procedures that can be used to interpret the inspection results.

In particular, a mathematical programming model is formulated to determine the best inspection planning. The model accounts for the following practical settings of track inspection planning operations.

- Inspection times of the tracks: Different tracks might require different inspection times due to their lengths and/or speed limits on them.
- Inspection frequencies of the tracks: Different tracks might require varying inspection frequencies due to their classifications, the traffic on them, and/or their importance regarding safety.
- Times between consecutive inspections on the same track: The time between consecutive inspections of the same track should be long enough allowing traffic on it. That is, a track should not be inspected right after its previous inspection as this would not allow sufficient time to detect new defects and/or accurately measure safety risks.
- Inspection importance of the tracks: Depending on their characteristics, different tracks can have different inspection importance. For instance, it might be more important to inspect different railroad classes or it might be considered more important to inspect the tracks with passenger and/or hazardous material traffic than to inspect tracks with low and/or non-hazardous freight traffic.

The project formulates a mathematical optimization problem for the track inspection planning considering the above practical settings of track inspection operations. It is important to schedule track inspections such that the potential defects are captured as much as possible within minimum times to increase safety to the maximum. Therefore, the two objectives simultaneously captured in this model are minimization of total inspection times and maximization of the total importance of the inspections.

A solution method which determines the inspection schedules is proposed for this model. Specifically, an inspection schedule facilitates decision making by telling (i) which tracks should be inspected, (ii) when these tracks should be inspected, and (iii) what should be the sequence of track inspections. A genetic algorithm is constructed to determine inspection schedules with low total inspection times as well as high total inspection importance. This method provides a set of alternative inspection schedules that are not only effective in terms of total inspection times but also total safety importance of the inspections.

In absence of this scheduling method, a simple dynamic procedure can be used for track inspection planning; however, it would not account for the total time and total importance of the inspections. Upon comparing the solution method proposed in this project to this simple procedure on a set of railroad track networks of different sizes, the solution method proposed proves to find improved inspection schedules regardless of the railroad network size. That is, the solution method will improve railroad safety by two means: (i) either reduce the total time

required for inspections or (ii) increase the inspections within a given time frame. Therefore, MoDOT can benefit from this method either by reduced resource usage to maintain railroad safety or increased safety with same amount of resource usage.

A review of the techniques on how to use the inspection results to measure risk of failure is provided. Particularly, reliability, defect development, and crack growth approaches are discussed for measuring the risk of failure for the tracks. Based on these approaches, a method for associating a risk value for a track segment after inspection, which uses the observations of the inspection, is recommended. This method calculates a risk value for an inspected track considering the crack size detected in case the crack is detected. In case the crack is not detected, this method calculates a risk value using the lifetime characteristics of the track, the accuracy of the inspection method, and the process of a crack from its initialization to failure development.

CONTENTS:

LIST OF PARAMETERS, VARIABLES, AND FUNCTIONS:

Parameters:

n	Number of tracks in the railroad network
k_{max}	Maximum number of inspections that can be completed
i, j	Indexes used for a track, $i, j = 1, 2, 3, \dots, n$
k, p	Indexes used for an inspection, $k, p = 1, 2, 3, \dots, k_{max}$
I	Set of tracks, $i \in I, I = \{1, 2, 3, \dots, n\}$
K	Set of inspections, $k \in K, K = \{1, \dots, k_{max}\}$
T	The length of the inspection period
M	A large number, $M \geq T$
T_{ik}	The start time of the k^{th} inspection on track $i \in I$
L_i	The number of required inspections for track $i \in I$
L	$1 \times n$ vector of L_i values, $L = [L_1, L_2, L_3, \dots, L_n]$
t_i	Inspection time of track $i \in I$
t	$1 \times n$ vector of t_i values, $t = [t_1, t_2, t_3, \dots, t_n]$
τ_i	The minimum time required between two consecutive inspections of track $i \in I$
τ	$1 \times n$ vector of τ_i values, $\tau = [\tau_1, \tau_2, \tau_3, \dots, \tau_n]$
$\hat{t}_{ij}$	The travel time from track $i \in I$ to track $j \in I$
$\hat{t}$	$n \times n$ matrix of $\hat{t}_{ij}$ values
w_i	The inspection importance of
w	$1 \times n$ vector of w_i values, $w = [w_1, w_2, w_3, \dots, w_n]$

Variables:

y_{ik}	1 if track $i \in I$ is inspected at inspection $k \in K$, 0 otherwise
Y	$n \times k_{max}$ matrix of y_{ik} values
z_{ijk}	1 if track $j \in I$ is inspected right after track $i \in I$ at the $(k+1)^{th}$ inspection, 0 otherwise
Z	$n \times n \times (k_{max} - 1)$ array of z_{ijk} values
T_{ik}	The start time of the k^{th} inspection on track $i \in I$

Model Functions:

$TT(Y, Z)$	Total time of inspections, $TT(Y, Z) = \sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik} + \sum_{k=1}^{k_{max}-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijk}$
$TW(Y)$	Total weight of inspections, $TW(Y) = \sum_{k=1}^{k_{max}} \sum_{i \in I} w_i y_{ik}$

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION AND LITERATURE REVIEW

One of the major problems that railroads have faced since the earliest days is the service failures on railroad tracks. The North American railroads have been inspecting their most costly infrastructure asset, the rail, since the late 1920's; and, with increased traffic and more frequent failures on railroads, rail inspection is more important today than it has ever been (C. NDT, 2013). To keep railroads safe and prevent any high maintenance costs caused by failures on the railroads, scheduled inspections must be performed on rail tracks, soil, and bridges. To plan such a scheduled inspection process, a problem formulation and optimization tools are needed. This section provides a review of inspection problems and optimization tools that are used to solve the inspection scheduling problems. In particular, the following three aspects of railroad inspection have been reviewed: types of inspection, inspection regulations, and track inspection planning.

1.1.Types of Inspection

There are several types of track inspections such as soil inspection, railroad bridge inspection, and railroad inspection, and each track inspection type has their subcategories. Figure 1 summarizes the different types of inspections and their subcategories, which are briefly discussed next.

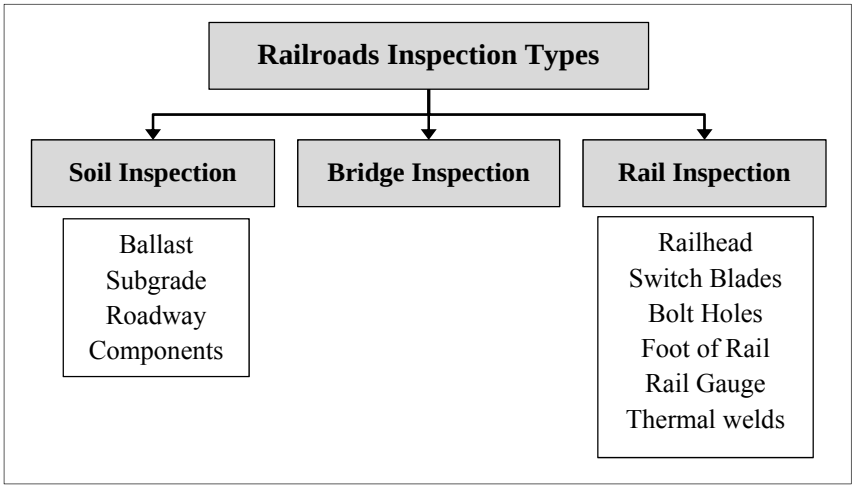


Figure 1. Different types of Inspection in a railroad

1.1.1. Soil Inspection

Soil inspection investigates the Ballast, Subgrade, and Roadway (BSR) component, which includes all earthen materials on the track structure, tracks, and embankments (Uzarski et al., 1993). Table 1 gives the definitions of BSR components. It focuses on the thickness of the ballast, subsoil material and geotechnical properties of subgrade (Hugenschmid, 1999). In addition, the inspection is executed mainly by digging trenches at evenly spaced intervals and in locations of special interest (Hugenschmid, 1999).

Table 1. Components of Soil Inspection and Their Definitions

Components	Definition
Ballast	"Ballast serves to secure the track structure in place, provide for track structure drainage, and transmit loads from the ties to the subgrade" (Uzarski et al. 1993).
Subgrade	"The subgrade is either the natural earth or placed fill upon which the track structure rests" (Uzarski et al. 1993).
Embankment	A road made of earth to build the tracks on it.

A major potential BSR defect is the vegetation growth, which includes grass, weeds, bushes, trees, and other natural cover (Uzarski et al., 1993). Therefore, soil inspection also investigates the plants on the railroads. For instance, Erikson et al. (2004) tried to improve the reliability of soil inspection by adding the investigation of plant growth in the inspection process.

1.1.2. Railroad Bridge Inspection

Many of the railroad bridges were not designed for the weights they are now bearing and as a consequence, structural failure rates are increasing (Lee et al., 1999). To prevent this failure, the bridges are inspected frequently. Bridge inspection is a regularly scheduled bridge safety inspection that is conducted on all railroad bridges (C. B. Transportation, 2012).

1.1.3. Rail Inspection

Rail inspection investigates the Rail Heads, Switch Blades, Bolt Holes, Foot of the Rail, Rail Gauge, Thermite Welds, and etc. (H. NDT, 2014). Some technologies such as laser induced ultra sound can improve the rail inspection (Cerniglia et al., 2006). Rail inspection and more particularly rail inspection planning is the focus of this project; hence, the review focuses on rail inspection. In what follows, the studies on rail inspection are reviewed. Later, rail inspection planning will be discussed in detail.

Railheads: Railhead is the highest part of a rail profile that the wheels of train roll on. Defects in a railhead and web can be detected with rail inspection cars and rail inspection tools equipped with special transducers such as wheel probes (Hayashi et al., 2007). Using ultrasonic surface wave, the critical surface cracks in the railhead can be detected (Hesse, 2007). There are many technologies available for this kind of rail inspection. Some of them are discussed by Uzarski et al. (1994).

Bottom Edges: Bottom edges of rails are easily damaged regions due to contact with soil and fastening. Bottom edges are blind zones for the conventional ultrasonic inspection techniques in which signals are input from a railhead (Hayashi et al., 2007). Kenderian et al. (2006) proposed an ultrasonic technology to inspect bottom of rails as well.

Track gauge: Track gauge is the spacing of the rails on a railway track. Babenko (2006) studied rail gauge and reliable identification and localization of structural defects on railroad tracks.

Crane inspection: Crane inspection should be done based on international standards (Gantrex, 2000). Gantrex (2000) argued that crane rails are inspected and replaced based on the subjective judgment and experience of the inspector, which can lead to either premature replacement or major problems caused by waiting too long to replace. Gantrex (2000) introduced some international standards to use for the inspection process.

1.2.Inspection Regulation

According to the US Department of Transportation (DOT) and Federal Railroad Administration (FRA), there are some roles and regulations regarding the rail inspection. The following discussion summarizes two sets of regulations announced by US DOT and FRA.

An inspector has the following responsibilities (O. D. Transportation, 2012):

- To conduct independent inspections of railroad facilities and equipment on tracks, highway/rail grade crossings, cabooses, bridges, cars, crew quarters, signals, switches, and safety devices,
- To investigate railroad accidents of derailments, hazardous material spills, grade crossing and/or railroad employee fatality,
- To collect data and exhibits to prepare reports of accident findings and safety violations and attend hearings, court proceedings, meetings and conferences pertaining to violations of Public Utilities Commission and /or state codes, rules and regulations, and operate a computer to enter and retrieve data.

According to Federal Railroad Administration (FRA, 2013), a railroad inspector must:

- Plan and implement periodic inspections to provide optimum coverage of the railroad track networks,
- Coordinate operation of track geometry tests and use the information to detect, locate, and evaluate deviations in cross-level, gage and profile of the track,
- Perform on-ground inspections, where defects are indicated, to determine the seriousness of the problems and the best means of correction.

1.3.Track Inspection Planning

Each year, railroad companies spend billions of dollars on track inspection to recover tracks from defects and damages and eliminate potential safety hazards (Peng, 2011). Currently, track maintenance planning is mostly manual and relies on the judgment of inspectors. There is a considerable potential to improve the process by using operations research techniques to develop solutions to the problems on track maintenance (Peng, 2011).

1.3.1. Optimization Problems in Track Inspection Planning

The literature review on track inspection planning shows that there are several aspects of optimization problems that are being considered. The optimization problems extend from (1) strategic to management, (2) maintenance to failing rates, and (3) scheduling to monitoring.
Uzarski et al. (1993) proposed a technology management approach to maintain the railroads, while Peterson (2012) and Andersson (2002) concentrate on the strategic view of

maintenance/inspection to develop a strategic plan for inspection. Kim and Farangopol (2011) investigated an optimization problem to minimize the expected total cost and the expected failure cost. Their formulation of the optimization model for inspection scheduling is extended to capture monitoring and scheduling as well.

The failing rate problem is another important issue in maintenance/inspection that is investigated by Prescott and Andrews (2013) and Kahima (2004). Prescott and Andrews (2013) consider deterioration rate of the track as the mainstream problem to do the maintenance and Kahima (2004) focuses on defect management. Kashima (2003) also analyzes the inspection scheduling to minimize the time considering interventions due to maintenance operations. Esveld (1990) describes an approach to analyze the deterioration of track components and suggests a method for determining maintenance and renewal intervals.

The main concern in the inspection/maintenance planning is the scheduling/monitoring issue. Acharya et al. (1991) discussed a rail replacement scheduling problem. Dell'Orce et al. (2001) studied the optimization of intervention on railway network at the right moment and monitoring. Budai-Balke (2009) tried to find the optimal time intervals for carrying out routine maintenance operations to minimize the track possession costs and maintenance costs. Podofillinia et al. (2005) tried to work on a problem to reduce the operation and maintenance expenditures while still assuring high safety standards. Finally, Higgins (1998) determined the best allocation of maintenance activities and crews so as to minimize the disruption to and from scheduled trains and to reduce completion time.

Another issue regarding the scheduling is the position of maintenance planning. Hall (2000) focused on utilization of maintenance facility positions where multiple cars are assigned to the same track. He tried to decrease the possibility of blocking the rail when different teams are on the move to do the maintenance.

1.3.2. Methods and Results in Track Inspection Planning

There is a range of optimization methods that are used in the track inspection planning such as operations research tools, multi-objective optimization approaches, probabilistic approach, fuzzy logic and evolutionary programming, Markov models, event tree analysis, and heuristic search algorithms. The following provides a short description of those methods in track inspection planning.

Andersson (2002) used an operations research method to find a solution to the strategic planning problem. He found some alternatives for maintenance procedure. He suggested two steps for maintenance planning: first step is to find the best action, and the second is to take action at the right time by the scheduling techniques (Andersson 2002). Similarly, Gordond et al. (2007) considered the planning inspections as a two-step problem; (1) adequate planning support to prevent inefficient or overlooked inspections and undetected defects, and (2) lack of a planning formalism for specifying inspection goals and developing and selecting inspection plans. They used a stochastic search algorithm for the inspection planning and they identified the best requirements to increase the efficiency of planning. They solved the second problem by using emerging sensors as a mainstream for quality control and noted that inspection planning can benefit from it. In addition, Podofillinia et al. (2005) adapted a multi-objective

optimization approach in an effort to optimize inspection and maintenance procedures with respect to both economical and safety-related aspects.

Dell'Orco et al. (2001) used fuzzy logic to model an optimization problem and solved it via an evolutionary programming algorithm. They handled all activities related to the rail tracks maintenance in order to respect a balance between safety and economic aspects. Budai-Balke (2009) also used evolutionary programming methods. Shiau et al. (2007) used genetic algorithm to concurrently determine both process planning and inspection planning operations in one problem. They preferred a genetic algorithm because of the size of the problem and the nonlinearity of the objective functions. Higgins (1998) used the tabu search heuristic for which the neighborhood is defined by swapping the order of jobs, maintenance crews, or both.

Kim and Frangopol (2011) used a probabilistic approach to seek the optimum cost-based inspection. Their solution provides the inspection times and quality of inspections. The optimum monitoring starting times and monitoring durations are also obtained by their optimization model. Kashima (2004) introduced an Event Tree (ET) analysis method and life-cycle cost (LCC) model to optimize inspection/repair intervention in rail defect management. ET analysis includes all events and actions with respect to inspection/repair intervention and LCC model takes time value of money into account.

1.4.Track Inspection Planning in the Project Content

In this project, a track inspection scheduling problem with practical restrictions and realistic objectives is formulated and solved. Specifically, a bi-objective non-linear-integer optimization problem, where the total time to complete the predetermined number of inspections on a given set of railroad tracks is minimized while the total importance of inspections is maximized, is modeled. This model explicitly considers the travel time from one track to another and the time required between two consecutive inspections of the same track. A genetic algorithm to approximate a set of Pareto efficient schedules for the resulting model is proposed and it is compared to a naïve greedy approach that can be used for inspection scheduling. The results of the comparison indicate that the proposed solution method finds improved schedules not only in terms of total time but also better total importance of the inspections. Next section details the formulation of the mathematical programming model.

2. TRACK INSPECTION PLANNING MODEL

The model focuses on determining a schedule for an inspection vehicle on a railroad network of tracks. In particular, suppose that the inspection vehicle should inspect the tracks on the railroad network for the next inspection period. Each track requires different number of inspections within the inspection period. Furthermore, inspection of different tracks takes different times due to the length of the tracks and/or the speed limit of the inspection vehicle on the track. After inspecting a track, the inspection vehicle moves to the next track to be inspected. Note that the inspection vehicle can travel on the regular road network. That is, the inspection vehicle is not restricted to travel on railroad tracks only. Particularly, the inspection vehicle can travel to another part of the railroad network on the road. Therefore, the inspection planning problem accounts for the travel time from one track to another.

Now, consider a railroad network with n tracks and let the tracks be indexed by i such that $i \in I = \{1, \dots, n\}$. As noted previously, different tracks might require different inspection frequencies due to their characteristics. Therefore, let L_i define the number of inspections required for track $i \in I$. Furthermore, let T denote the length of the inspection period. Due to different lengths and various speed limits, inspection times of the tracks vary. To capture distinct inspection times of the tracks, let t_i be the inspection time of track $i \in I$. Also, to capture the travel time from one track to another, let $\hat{t}_{ij}$ denote the inspection vehicle's travel time between track $i \in I$ and track $j \in I$. Finally, there should be sufficient time between consecutive inspections of the same track to allow traffic on the track; otherwise, the inspection vehicle would complete all of the required inspections on a track consecutively as this would minimize the travel times among the tracks of the given railroad network. Therefore, let τ_i denote the minimum time required between two consecutive inspections of track $i \in I$.

A track inspection plan specifies the order of inspections on the tracks. For instance, an inspection schedule on a railroad network with 4 tracks, namely, track 1, track 2, track 3, and track 4, which require 1, 2, 2, and 1 inspection(s), respectively, for the next inspection period states the order of inspections for the inspection vehicle as follows:

> Inspect track 1 → Travel to track 3 → Inspect track 3 → Travel to track 2 →
> Inspect track 2 → Travel to track 4 → Inspect track 4 → Travel to track 3 →
> Inspect track 3 → Travel to track 2 → Inspect track 2.

In the above schedule, the numbers of inspections on each of the tracks 1, 2, 3, and 4 are equal to the number of inspections required for each track. However, considering the length of the inspection period, if there is still time left, the inspection vehicle can travel to a track and inspect it if the time passed since the last inspection of the track is greater than the minimum time required between two consecutive inspections of that track. This would increase the probability of detecting a defect on this track within the given inspection time period. Therefore, the mathematical model should allow inspection of a track more but not less than the number of required inspections on that track within the inspection period.

In particular, given the length of the inspection period, one can calculate the maximum number of inspections that can be completed within the inspection period. The maximum number of inspections that can be calculated is equal to the length of the inspection period divided by the inspection time of the track which has the minimum inspection time among all of the tracks. The mathematical model, therefore, sets a limit on the number of inspections that can be completed. Specifically, let inspections be indexed by k such that $k \in K = \{1, \dots, k_{max}\}$. Here, k_{max} defines the maximum number of inspections possible and inspection k is the k^{th} inspection that the inspection vehicle completes. Then, once it is known which track is inspected in which inspection, the inspection schedule is known. For instance, the above inspection schedule of the network with four tracks can alternatively be defined as follows (considering there will be a maximum of 7 inspections within the inspection period):

Inspection 1	Inspection 2	Inspection 3	Inspection 4	Inspection 5	Inspection 6	Inspection 7
Track 1	Track 3	Track 2	Track 4	Track 3	Track 2	-

Therefore, the decision variables of the track inspection planning problem can be introduced as whether a track is inspected at an inspection or not. In particular, let

$$y_{ik} = \begin{cases} 1 \; if \; track \; i \; is \; inspected \; in \; the \; k^{th} \; inspection, \\ 0 \; otherwise, \end{cases}$$

and let Y be a $(n \times k_{\max})$ matrix of y_{ik} values. Considering the definition of y_{ik}, the total number of times track $i \in I$ is inspected is equal to $\sum_{k=1}^{k_{max}} y_{ik}$. Recall that each track $i \in I$ should be inspected at least L_i times; therefore, the inspection plan should guarantee that $\sum_{k=1}^{k_{max}} y_{ik} \geq L_i$ for each track $i \in I$.

Furthermore, considering the definitions of y_{ik} and t_i, one can note that the inspection vehicle is inspecting tracks for $\sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik}$ time units. However, it should be pointed out that $\sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik}$ does not include the travel times of the inspection vehicle between two tracks. In order to transit between two tracks and complete the inspections in an order between the tracks, the order variable z_{ijk} is defined, where

$$z_{ijk} = \begin{cases} 1 \; if \; track \; j \; is \; inspected \; right \; after \; track \; i \; in \; the \; (k+1)^{th} \; inspection, \\ 0 \; otherwise, \end{cases}$$

and let Z be the $(n \times n \times k_{\max})$ array of z_{ijk} values. For the schedule given in the above example for the railroad network with four tracks, $z_{131} = 1$, $z_{322} = 1$, $z_{243} = 1$, $z_{434} = 1$, and $z_{325} = 1$. Then, considering the definitions of z_{ijk} and $\hat{t}_{ij}$ values, the total time the inspection vehicle spends on travelling among the tracks amounts to $\sum_{k=1}^{k_{max}-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijk}$. Here, the travel time to the first inspected track is disregarded.

Recall that the time between two consecutive inspections of track $i \in I$ should be greater than or equal to the minimum time required between two consecutive inspections of track $i \in I$, i.e., τ_i. In particular, suppose that track $i \in I$ is to be inspected at the k^{th} inspection and let T_{ik} be the start time of the k^{th} inspection on track $i \in I$. Then, one can show that

$$T_{ik} = \sum_{p=1}^{k-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=1}^{k-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp},$$

where p defines the inspections completed prior to the k^{th} inspection. That is, T_{ik} is the summation of the inspection times of all previous inspections and the travel times among the previously inspected tracks plus the travel time to track $i \in I$ from the last inspected track. Furthermore, suppose that track $i \in I$ is to be inspected at the $(k+n)^{\text{th}}$ inspection as well, where $1 \leq n \leq k_{max} - k$ and let $T_{i(k+n)}$ be the start time of the $(k+n)^{\text{th}}$ inspection on track $i \in I$. Similarly, one can show that

$$T_{i(k+n)} = \sum_{p=1}^{k+n-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=1}^{k+n-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp}.$$

The time between the k^{th} and $(k+n)^{th}$ inspections must be greater than or equal to the minimum time required between two consecutive inspections of track $i \in I$, if track $i \in I$ is to be inspected at the k^{th} and $(k+n)^{th}$ inspections. That is, if $y_{ik} = y_{i(k+n)} = 1$, one should have $T_{i(k+n)} - T_{ik} \geq \tau_i$. In the mathematical model, this restriction is formulated as a constraint.

At this point, the total time for inspecting tracks, including track inspection times and the travel times among the inspected tracks, can be calculated. Specifically, total inspection time as a result of inspection decisions Y and travelling decisions Z, denoted as $TT(Y,Z)$, amounts to

$$TT(Y,Z) = \sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik} + \sum_{k=1}^{k_{max}-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijk},$$

where the first component defines the total time of inspecting the tracks and the second component defines the total travel times among the tracks on the given railroad network of tracks. Note that the total inspection time should be within the given length of the inspection period; therefore, $TT(Y,Z) \leq T$.

If the only purpose of track inspection planning was to minimize the total time to complete the required inspections within the inspection period, one would simply minimize $TT(Y,Z)$ subject to the total time constraint, the minimum required inspection constraints for the tracks, and the constraints for the minimum required times between two consecutive inspections of the same track. However, an inspection schedule should also maximize the possible safety achievements from the inspections within the inspection period. To this end, after the required inspections are completed, the inspection vehicle can be scheduled to inspect more tracks if there is time remaining. For instance, the tracks with higher importance, such as the tracks with passenger traffic, hazardous materials, or higher traffic volumes can have higher safety priorities. Therefore, if time allows, such tracks should be inspected more.

To capture the importance of inspections on different tracks, a weight is defined for each track. In particular, let w_i denote the weight of the inspection importance of track $i \in I$. The higher the w_i value is, it is more important to inspect track $i \in I$. Therefore, an inspection plan should not only minimize $TT(Y,Z)$ but also maximize the total weighted importance of the inspections. Considering the definitions of y_{ik} and w_i, total weighted importance as a result of inspection decisions Y, denoted as $TW(Y)$, amounts to

$$TW(Y) = \sum_{k=1}^{k_{max}} \sum_{i \in I} w_i y_{ik}.$$

The track inspection planning problem, ***TIPP***, has two objectives: minimization of total inspection time and maximization of total weighted important. ***TIPP*** is stated as follows:

$$\textbf{TIPP}: \quad Maximize \quad TW(Y) = \sum_{k=1}^{k_{max}} \sum_{i \in I} w_i y_{ik}$$

$$Minimize \quad TT(Y,Z) = \sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik} + \sum_{k=1}^{k_{max}-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijk}$$

$$such\ that \quad \sum_{k=1}^{k_{max}} y_{ik} \geq L_i, \qquad \forall i \in I \tag{1}$$

$$\sum_{k=1}^{k_{max}} \sum_{i \in I} t_i y_{ik} + \sum_{k=1}^{k_{max}-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijk} \leq T \tag{2}$$

$$\sum_{i \in I} y_{ik} \leq 1 \ \forall k \in K \tag{3}$$

$$z_{ijk} \leq y_{ik}, \qquad \forall i,j \in I; \forall k \in [1, \dots, k_{max}-1] \tag{4}$$

$$z_{ijk} \leq y_{j(k+1)}, \qquad \forall i,j \in I; \forall k \in [1, \dots, k_{max}-1] \tag{5}$$

$$z_{ijk} \geq y_{ik} + y_{j(k+1)} - 1, \qquad \forall i,j \in I; \forall k \in [1, \dots, k_{max}-1] \tag{6}$$

$$\sum_{p=k}^{k+n-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=k}^{k+n-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp} \geq \tau_i + M\left(y_{ik} + y_{i(k+n)} - 2\right), \forall i \in I;\ \forall k \in [1, \dots, k_{max}-1];\ \forall n \in [1, \dots, k_{max}-k] \tag{7}$$

$$y_{ik} \in \{0,1\}, \forall i \in I, \forall j \in I, \tag{8}$$

$$z_{ijk} \in \{0,1\}, \forall i \in I, \forall j \in I, \forall k \in K \tag{9}$$

Constraints given in Equation (1) ensure that the inspection plan is designed such that each track on the railroad network is inspected by at least the minimum number of required inspections. The constraint given in Equation (2) guarantees that the total inspection time is less than or equal to the length of the inspection period. The constraints given in Equation (3) enforce that at each inspection, at most one inspection is completed. Note that the mathematical model considers the maximum number of inspections and some of the

inspections will not be executed due to the length of the inspection period. Constraints given in Equation (4) restrict inspection of a track unless the inspection vehicle travels to that track from the previously inspected track. Similarly, constraints given in Equation (5) restrict the inspection of a track unless the inspection vehicle moves to the next track that will be inspected. Constraints given in Equation (6) define the travelling schedule of the inspection vehicle. Particularly, if inspection vehicle is travelling from track i to track j after inspection k, z_{ijk} has to be 1 by definition. That is, if $y_{ik} = 1$ and $y_{j(k+1)} = 1$, Equation (6) forces $z_{ijk} = 1$. On the other hand, if $y_{ik} + y_{j(k+1)} \leq 1$, z_{ijk} can be 0 or 1. At this case, Equations (5) and (4) will define z_{ijk} accordingly. Constraints given in Equation (7) prevent the time between two consecutive inspections of the same track to be less than the specified time between the consecutive inspections of that track. In particular, recall that the time of start for the k^{th} inspection on track i is $T_{ik} = \sum_{p=1}^{k-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=1}^{k-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp}$, and similarly, the start time of the $(k+n)^{th}$ inspection on the same track is $T_{i(k+n)} = \sum_{p=1}^{k+n-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=1}^{k+n-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp}$. The time between the k^{th} inspection and $(k+n)^{th}$ inspection must be greater than the minimum time required between two consecutive inspections of that particular track, i.e., τ_i. That is, if $y_{ik} = y_{i(k+n)} = 1$, one should have $T_{i(k+n)} - T_{ik} \geq \tau_i$. On the other hand, if $y_{ik} + y_{i(k+n)} \leq 1$, there should not be a restriction on $T_{i(k+n)} - T_{ik}$. Therefore, Equation (7) forces

$$\sum_{p=k}^{k+n-1} \sum_{i \in I} t_i y_{ip} + \sum_{p=k}^{k+n-1} \sum_{i \in I} \sum_{j \in I} \hat{t}_{ij} z_{ijp} \geq \tau_i + M\left(y_{ik} + y_{i(k+n)} - 2\right), \forall n \in [1, \dots, k_{max} - k]$$

where M is a large number (one can use the length of the inspection period, T, for M). Note that, if $y_{ik} = y_{i(k+n)} = 1$, it means that $y_{ik} + y_{i(k+n)} = 2$; thus, Equation (7) implies that $T_{i(k+n)} - T_{ik} \geq \tau_i$. On the other hand, if $y_{ik} + y_{i(k+n)} \leq 1$, Equation (7) implies that $T_{i(k+n)} - T_{ik} \geq -M$, which is true by definition as the $(k+n)^{th}$ inspection cannot start before the k^{th} inspection. Finally, constraints defined in Equation (8) are the binary restrictions of the inspection decisions and the constraints defined in Equation (9) are the binary restrictions of the travelling decisions.

In the next section, a solution method is developed to solve ***TIPP***. Note that T, w_i, t_i, τ_i, L_i, and $\hat{t}_{ij}$ values of the railroad network are input for ***TIPP***. The output of ***TIPP*** will be Y *and* Z variables, which define an inspection schedule for the inspection vehicle.

3. SOLUTION ANALYSIS

TIPP is a binary bi-objective optimization problem. Two common methods for solving multi-objective optimization problems are to reduce the multi-objective model into a single-objective model and generating a set of Pareto efficient solutions. One can reduce the bi-objective model into a single objective model by assigning weights to the objective functions and adding them to have the single objective or formulating a single-objective model that minimizes the maximum deviation from the optimum solutions of the individual objectives. In

this project, we will use the later method by adapting an evolutionary heuristic algorithm to approximate a set of Pareto efficient solutions.

Pareto-Front is the set of Pareto efficient solutions for multi-objective optimization problems. A solution is defined to be Pareto efficient if there is no other solution that is better in terms of all of the objectives. In particular, the following formal definition is used:

> A track inspection schedule (Y^*, Z^*) is Pareto efficient if there exists no other track inspection schedule (Y, Z) such that $TT(Y, Z) \leq TT(Y^*, Z^*)$ and $TW(Y, Z) \geq TW(Y^*, Z^*)$.

Approximating a Pareto-Front for multi-objective optimization models gives a set of solutions to the decision maker, among which the decision maker can select a solution. However, approximating a Pareto-Front for multi-objective binary-programming models is challenging. It should be noted that even with a single objective, ***TIPP*** is a complex optimization problem. Therefore, heuristic algorithms are commonly used for such optimization problems. Specifically, due to the binary definitions of the decision variables of ***TIPP***, evolutionary methods such as genetic algorithms can be efficiently used to approximate a Pareto-Front for ***TIPP***.

In what follows, a genetic algorithm is developed for ***TIPP***. Prior to discussing the details of this genetic algorithm, it should be noted that a simple dynamic heuristic method can be used to solve ***TIPP*** in absence of the modeling approach given in this project. This dynamic approach, which is referred to as greedy heuristic, iteratively schedules inspections for the inspection vehicle. Therefore, the details of the greedy heuristic are explained next before developing the genetic algorithm.

3.1.Greedy Heuristic Approach

Greedy heuristic approach is a very simple and naïve method that can create feasible track inspection schedules. Specifically, given an inspected track, the greedy heuristic determines the next track to be inspected. This process is repeated until all of the tracks inspected at least the minimum required number of times or the inspection period ends. Considering the two objectives of the track inspection planning problem, two greedy heuristics can be defined: time minimizing and weight maximizing. In both of the approaches, the following parameters are used in the algorithmic descriptions of the greedy heuristic approaches:

- CT : current time, i.e., the time spent until the inspection completion at the current track
- r^j : the remaining number of inspections for track j
- pst^j: possible start time for track j as the next track to be inspected
- pft^j: possible finish time for track j's inspection as the next track to be inspected
- lit^j: completion time of track j's latest inspection

Now, suppose that the last track inspected is track i; then, the following equations hold true:

$CT = lit^i$,
$pst^j = CT + \hat{t}_{ij}$,
$pft^j = CT + \hat{t}_{ij} + t_j$.

Track j can be the next track to be inspected if there has passed sufficient time from its latest inspection and the possible finish time is within the inspection period. That is, if $pst^j \geq lit^j + \tau_j$ and $pft^j \leq T$, track j can be inspected next since the time between two consecutive inspections of track j will be greater than the minimum required time between two consecutive inspections of track j as well as its inspection can be completed within the inspection period. Depending on the next track selection criteria, the greedy heuristic approaches will iteratively select next tracks for inspection and update the current time (CT), possible start times of the tracks (pst^j), possible finish times of the tracks (pft^j), the latest completion times of the tracks (lit^j), and the remaining inspections of the tracks (r^j). Note that if $r^j = 0$ for all tracks, then the minimum numbers of required inspections on all tracks are completed. Therefore, in the iterative selection processes for the next track, the greedy heuristic approaches first considers the tracks such that $pst^j \geq lit^j + \tau_j$, $pft^j \leq T$, and $r^j \geq 1$ as the next track to be inspected. If there is no track with remaining inspections that can be the next track to be inspected due to their minimum time requirement between two consecutive inspections or their late completion times, the next track can be selected from the tracks with no remaining inspections.

3.1.1. Time Minimizing Greedy Heuristic

In the time minimizing greedy heuristic, starting from a randomly selected track, the next track to be inspected will be the one with the least travel time from the current track plus inspection time within the set of tracks that can be inspected next considering the time constraint between two consecutive inspections of the same track and its remaining inspections. If there is no feasible track available to be the next track with remaining inspections, the next track selected is the one with the least travel time plus inspection time among the feasible tracks with no remaining inspections. The details of the time minimizing greedy heuristic are as follows:

- Time Minimizing Greedy Scheduler:
    1. Let T, w_i, t_i, τ_i, L_i, and $\hat{t}_{ij}$ values be given.
    2. Set $CT = 0$ and $k = 1$.
    3. For each track j such that $j \in I$
        - Let $pst^j = 0$, $pft^j = t_j$, $lit^j = -T$, $r^j = L_j$
    4. Let track i such that $i \in I$ be randomly selected.
    5. Set $y_{ik} = 1$ and $y_{jk} = 0$ for j such that $j \neq i$.
        - Let $CT = t_i$, $lit^i = CT$, $pst^i = CT$, $pft^i = CT + t_i$, and $r^i = L_i - 1$
        - Define greedy index for track i as $g_i = T$
    6. For each track j such that $j \neq i$,
        - Calculate $pst^j = CT + \hat{t}_{ij}$, $pft^j = CT + \hat{t}_{ij} + t_j$
        - Define greedy index for track j as $g_j = \hat{t}_{ij} + t_j$

7. Determine the set of feasible tracks with remaining inspections, $FTwRI$, such that $FTwRI = \{j: pst^j \geq lit^j + \tau_j, pft^j \leq T, r^j \geq 1\}$.
    - If $FTwRI = \emptyset$,
        - If $max_j\{r^j\} = 0$, go to Step 8
        - Else, determine the set of feasible tracks, FT, such that $FT = \{j: pst^j \geq lit^j + \tau_j, pft^j \leq T\}$.
            - If $FT = \emptyset$, go to Step 8.
            - Else, let $i = argmin\{g_j: j \in FT\}$ and set $k = k + 1$ and go to Step 5.
    - Else, let $i = argmin\{g_j: j \in FTwRI\}$ and set $k = k + 1$.
        - Go to Step 5.
8. Return y_{ik} values.

Note that the time minimizing greedy heuristic terminates in case of the following situations: when there is no more required inspection left, the algorithm terminates as additional inspection would increase the total time or when there is no track that can be inspected as the next track, the algorithm terminates as additional inspection would not be completed within the inspection period. In the former case, a feasible schedule is determined. On the other hand, in the latter case, a feasible schedule is not determined due to the starting track. Therefore, in termination, if $\max\{j: r^j\} \geq 1$, one can set $TT(Y,Z) = \infty$ and $TW(Y,Z) = -\infty$. Otherwise, if $max_j\{r^j\} = 0$, a feasible schedule is determined. In such a case, once y_{ik} values are returned, z_{ijk} values can be easily determined. In particular, if $y_{ik} = 1$ and $y_{j(k+1)} = 1$, $z_{ijk} = 1$. Therefore, $TT(Y,Z)$ and $TW(Y,Z)$ values can be calculated. Starting the time minimizing greedy heuristic scheduler with each track being the first track to be inspected, at most n number of inspection schedules can be determined.

3.1.2. Weight Maximizing Greedy Heuristic

In the weight maximizing greedy heuristic, the only difference from the time maximizing greedy heuristic is that instead of selecting the next track to be inspected as the track with the minimum travel time from the current track plus the inspection time, the next track to be inspected is selected as the one with the maximum weight from the set of feasible tracks with remaining inspections. If the set of feasible tracks with remaining inspections is empty, unlike the time minimizing greedy heuristic, the weight maximizing greedy heuristic continues to schedule track inspections. The main termination criteria is exceeding the total time of the inspection period. The details of the weight maximizing greedy heuristic are as follows:

- Weight Maximizing Greedy Scheduler:
    1. Let T, w_i, t_i, τ_i, L_i, and $\hat{t}_{ij}$ values be given.
    2. Set $CT = 0$ and $k = 1$.
    3. For each track j such that $j \in I$
        - Let $pst^j = 0, pft^j = t_j, lit^j = -T, r^j = L_j$
    4. Let track i such that $i \in I$ be randomly selected.
    5. Set $y_{ik} = 1$ and $y_{jk} = 0$ for j such that $j \neq i$.
        - Let $CT = t_i, lit^i = CT, pst^i = CT, pft^i = CT + t_i$, and $r^i = L_i - 1$

- Define greedy index for track i as $g_i = -w_i$

6. For each track j such that $j \neq i$,
   - Calculate $pst^j = CT + \hat{t}_{ij}$, $pft^j = CT + \hat{t}_{ij} + t_j$
   - Define greedy index for track j as $g_j = w_j$
7. Determine the set of feasible tracks with remaining inspections, $FTwRI$, such that $FTwRI = \{j: pst^j \geq lit^j + \tau_j, pft^j \leq T, r^j \geq 1\}$.
   - If $FTwRI = \emptyset$,
     - If $min_j\{pft^j\} \geq T$, go to Step 8
     - Else, determine the set of feasible tracks, FT, such that $FT = \{j: pst^j \geq lit^j + \tau_j, pft^j \leq T\}$.
       - If $FT = \emptyset$, go to Step 8.
       - Else, let $i = argmin\{g_j: j \in FT\}$ and set $k = k + 1$ and go to Step 5.
   - Else, let $i = argmin\{g_j: j \in FTwRI\}$ and set $k = k + 1$.
     - Go to Step 5.
8. Return y_{ik} values.

Note that the weight maximizing greedy heuristic terminates in case of the following situations: when there is no time left to complete another inspection or when there is no track that can be inspected as the next track. Therefore, at termination, a feasible schedule is determined if $max_j\{r^j\} = 0$. If $\max\{j: r^j\} \geq 1$, one can set $TT(Y,Z) = \infty$ and $TW(Y,Z) = -\infty$. Otherwise, once y_{ik} values are returned, z_{ijk} values can be easily determined and $TT(Y,Z)$ and $TW(Y,Z)$ values can be calculated. Starting the weight maximizing greedy heuristic scheduler with each track being the first track to be inspected, at most n number of inspection schedules can be determined.

Applying these two greedy approaches with each track as the starting track, one can generate at most $2n$ track inspection schedules. Using the Pareto front generation method that will be detailed for the genetic algorithm, one can select the Pareto efficient solutions from this set of at most $2n$ schedules as the output of the greedy heuristic approach. Next, the details of the genetic algorithm are explained.

3.2.Genetic Algorithm Approach

There are four main steps of the genetic algorithm: chromosome representation and initialization, fitness evaluation, mutation, and termination. The details of these steps are as follows.

3.2.1. Chromosome Representation and Initialization

The decision variables Y and Z in ***TIPP*** can be shown in the sequence of a chromosome's genes. This representation will help satisfy the ordering constraints defined in Equations (4), (5), and (6). Furthermore, by considering only Y variables, one can easily calculate the values of the Z variables in the model as discussed for the greedy heuristic. However, the chromosome will become a matrix of zeroes and ones (with one 1 in each column); hence, it

will be a very large matrix. To tackle this problem, an integer representation for chromosomes is chosen. In this representation, every gene is an integer number that represents a track number and the whole chromosome shows the sequence of tracks that need to be inspected. Integer representation not only satisfies the ordering constraints, but also it satisfies the constraints given in Equation (3), which allow at most one inspection at a time for each inspection.

For example, suppose that the set of tracks is $I = \{1,2,3,4\}$ and the minimum inspections required for each track is $L = \{2,3,1,1\}$. This means that the track 1 needs to be inspected at least twice, track 2 needs to be inspected at least three times and tracks 3 and 4 need at least one inspection. Some chromosomes that can convey a solution to this problem are illustrated in Figure 2 (note that, for illustration purposes, these chromosomes ignore the time constraint between two consecutive inspections of the same tracks).

Chromosome 1:	2	1	2	3	2	1	4			
Chromosome 2:	1	2	3	1	2	4	1	3	2	4

Figure 2. Examples of Integer representation of a chromosome

In Chromosome 1, track 1 is inspected twice, track 2 is inspected 3 times, track 3 is inspected once, and track 4 is inspected once. Specifically, first track 2 is inspected, then track 1, then track 2 again, then track 3, then track 2 again, then track 1 again, and then track 4 are inspected. Therefore, the order of inspections defined in the chromosome. Note that, in Chromosome 1, the number of times each track is inspected is equal to the minimum number of inspections required for each track. In Chromosome 2, on the other hand, track 1 is inspected 3 times, track 2 is inspected 3 times, track 3 is inspected twice, and track 4 is inspected twice, which is possible if the inspection period is long enough.

To initiate the genetic algorithm, a set of feasible chromosomes should be generated as the initial population. Note that the representation of the chromosome is already satisfying constraints stated in Equations (3), (4), (5), (6), (8), and (9) in ***TIPP***. Therefore, in creating a feasible chromosome, one should focus on satisfying the constraints stated in Equation (1), i.e., the minimum inspection requirements for the tracks, the constraint defined in Equation (2), i.e., the length of the inspection period, and the constraints defined in Equation (7), i.e., the minimum time between two consecutive inspections of the same tracks. In creating chromosomes, we initially ignore the length of the inspection period. This constraint will be accounted for in the fitness evaluation step. Initial focus is on accounting for the constraints given in Equations (1) and (7). To do so, we use a dynamic approach for generating a feasible chromosome. Particularly, we add genes to a chromosome one by one until a feasible chromosome is generated. Note that each gene of a chromosome represents an inspection, and the value of the gene is the track inspected at that inspection. The details of generating a feasible chromosome are as follows.

First, we randomly select the track to be inspected at the first inspection. Here, we define and update the following statistics as we generate the next tracks to be inspected.

- *The spent time (ST):* The spent time calculates the time of inspections plus the travel times in the whole process. The spent time is updated at the end of each gene addition. This is defined as CT in the greedy heuristic algorithms.
- *The earliest start times (EST):* After each gene generation (that is, after each inspection), an earliest start time is defined for each track. The earliest start time for a track represents the time when an inspection can start on that track after the current inspection. Particularly, the earliest start time of track i as the next inspection is equal to the time the inspection at the last track is completed, i.e., the current spent time, plus the time to travel from the last inspected track to track i. This is defined similar to possible start time defined for the greedy heuristic algorithms. In particular, if track j is the last inspected track, earliest start time for track i is equal to

$$est^i = ST + \hat{t}_{ji}.$$

- *The latest start time (LST):* Each time a gene is added to the chromosome, we update the latest start time of the track added. The latest start time for track i represents the time when the last inspection of track i was completed. For instance, if the next gene to be added to the chromosome is track i, we will check whether the time between the earliest start of track i and the latest start time of track i is greater than or equal to the time needed between two consecutive inspections of track i, i.e., τ_i. If so, track i can be inspected as the next track, otherwise, it cannot be inspected at the next inspection. If inspected, the latest start time of track i is updated. This is defined similar to latest inspection time defined for the greedy heuristic algorithms.
- *Left inspections (gtl):* After each time a track is inspected, we calculate the remaining inspections left for each track. Left inspections will be used in choosing the next track to be inspected, i.e., the next gene to be added to the chromosome. This is defined similar to the remaining inspections defined for the greedy heuristic algorithms.

The following routines are used to generate feasible chromosomes. Particularly, Routine 1 creates a part of chromosome with m genes using gtl (in Routine 1, gtd is initially defined identical to gtl to randomize the order of tracks to be inspected in the case the current order of genes in gtl cannot generate m genes). Then, Routine 2 is using Routine 1 to generate a feasible chromosome. Initially, we generate n chromosomes as the first population.

Routine 1: Create Parts of A Chromosome:
Step 1: Set $g = |chromosome|$
Step 2: Constitute gtl and let $gtd = gtl$
Step 3: Set $counter = 0$
Step 4: While $counter \leq m$
Step 5: $y = gtl(1)$
Step 6: $EST = ST + \hat{t}(chromosome(g), y)$
Step 7: if $EST - LST(y) \geq \tau(y)$
Step 8: $chrom = chrom + \{y\}$
Step 9: $LST(y) = EST$
Step 10: $ST = EST + t(y)$
Step 11: Randomize elements of gtl

Step 12: $g = g + 1$
Step 13: $counter = counter + 1$
Step 14: else $gtl = gtl - \{y\}$
Step 15: if $gtl = \emptyset$
Step 16: $gtl = gtd$
Step 17: Return $chromosome$, LST and ST

Routine 2: Create A Feasible Chromosome:
Step 1: Set $inspections = \{0, \dots, 0\}$, $LST = -\tau$
Step 2: Set $x = random\ integer$ and $chromosome = x$
Step 3: Set $ST = t(x)$ and $inspections(x) = inspections(x) + 1$
Step 4: While $sum(L > inspections) > 0$
Step 5: Execute Routine 1
Step 6: Return $chromosome$, LST, ST and $inspections$

3.2.2. Fitness Evaluation

Suppose TW and TT are the total weighted inspections and the total time regarding the chromosome, respectively, and O is a feasible chromosome (solution) for $\boldsymbol{TIPP}$. Now, suppose a set of solutions R as a population of chromosomes is given such that O^r denotes the r^{th} solution in R. In fitness evaluation of population R, the purpose is to select the best chromosomes in the set. Since the problem has two objectives, we focus on generating the Pareto efficient solutions out of a given population. A solution is Pareto efficient if it is not dominated by another solution. Considering the minimization of TT and maximization of TW and two solutions O^s and O^r, unless $(TT^s, TW^s) = (TT^r, TW^r)$, O^s Pareto dominates O^r if $TT^s \leq TT^r$ and $TW^s \geq TW^r$. Considering this, the following routine can be used to generate all Pareto efficient solutions denoted by $PE(R)$ from a given set of solutions R.

Routine 3: Determining $PE(R)$:
Step 1: Set $s = 1$
Step 2: While $s \leq |R| - 1$
Step 3: Set $r = s + 1$
Step 4: While $r \leq |R|$
Step 5: Unless $(TT^s, TW^s) = (TT^r, TW^r)$
Step 6: if $TT^s \leq TT^r$ and $TW^s \geq TW^r$
Step 7: Set $R = R - \{O^r\}$ and $r = r - 1$
Step 8: if $TT^s \geq TT^r$ and $TW^s \leq TW^r$
Step 9: Set $R = R - \{O^s\}$ and $s = s - 1$ and $r = |R| + 1$
Step 10: Set $r = r + 1$
Step 11: Set $s = s + 1$
Step 12: Return $PE(R) = R$

Then, given a population R, $PE(R)$ is the set of parent chromosomes for the next population. Before continuing with the mutation step, we eliminate the chromosomes with TT values that are greater than the inspection period. That is, if a chromosome O^r such that $O^r \in PE(R)$ has $TT^r > T$, this chromosome is removed from $PE(R)$ and we have $PE(R) = PE(R) - \{O^r\}$.

3.2.3. Mutation

One important step of the genetic algorithm is mutation. After fitness evaluation of a population, the chromosomes in the next population are created by mutating the parent chromosomes of the current population. That is, Pareto efficient solutions in the current population will go through mutation operations to create children. Next population will include both the set of parents of the chromosomes and the children created with mutation. Having the parent chromosomes within the next population guarantees that the Pareto-Front is not worsening over the populations. Furthermore, we add randomly generated feasible chromosomes into the population. This will avoid getting stuck in local Pareto-Fronts.

The mutation operation works as follows. Given a parent chromosome, we first segment it into several parts, let's say p parts. Then, a mutation operation is applied to each part of the parent chromosome. Particularly, we randomly select a gene from the part of the parent chromosome and assume that the part of the parent chromosome from the beginning to this gene is fixed. Then, we create a feasible chromosome starting from the selected gene using Routine 1. Routine 4 describes the mutation operation for a given parent chromosome.

Routine 4: Mutation
Step 1: For $i = 1{:}\,p$
Step 2: Set $r{:=}\, random$
Step 3: Set $children(i){:=}\, parent(i, 1{:}\,r)$
Step 4: While $\min(L - inspections) > 0$
Step 5: Routine 1
Step 6: Return $children = \{children(i){:}\ \forall i\}$

Therefore, each parent chromosome generates p new chromosomes. The newly generated chromosomes plus the parent chromosomes constitute the next population.

3.2.4. Termination

If the set of $PE(R)$ does not change for a given number of populations, for example for k times, the algorithm stops and the Pareto-Front of the last population will be returned as the set of alternative solutions to the decision maker. A flow chart for the genetic algorithm is presented in Figure 3.

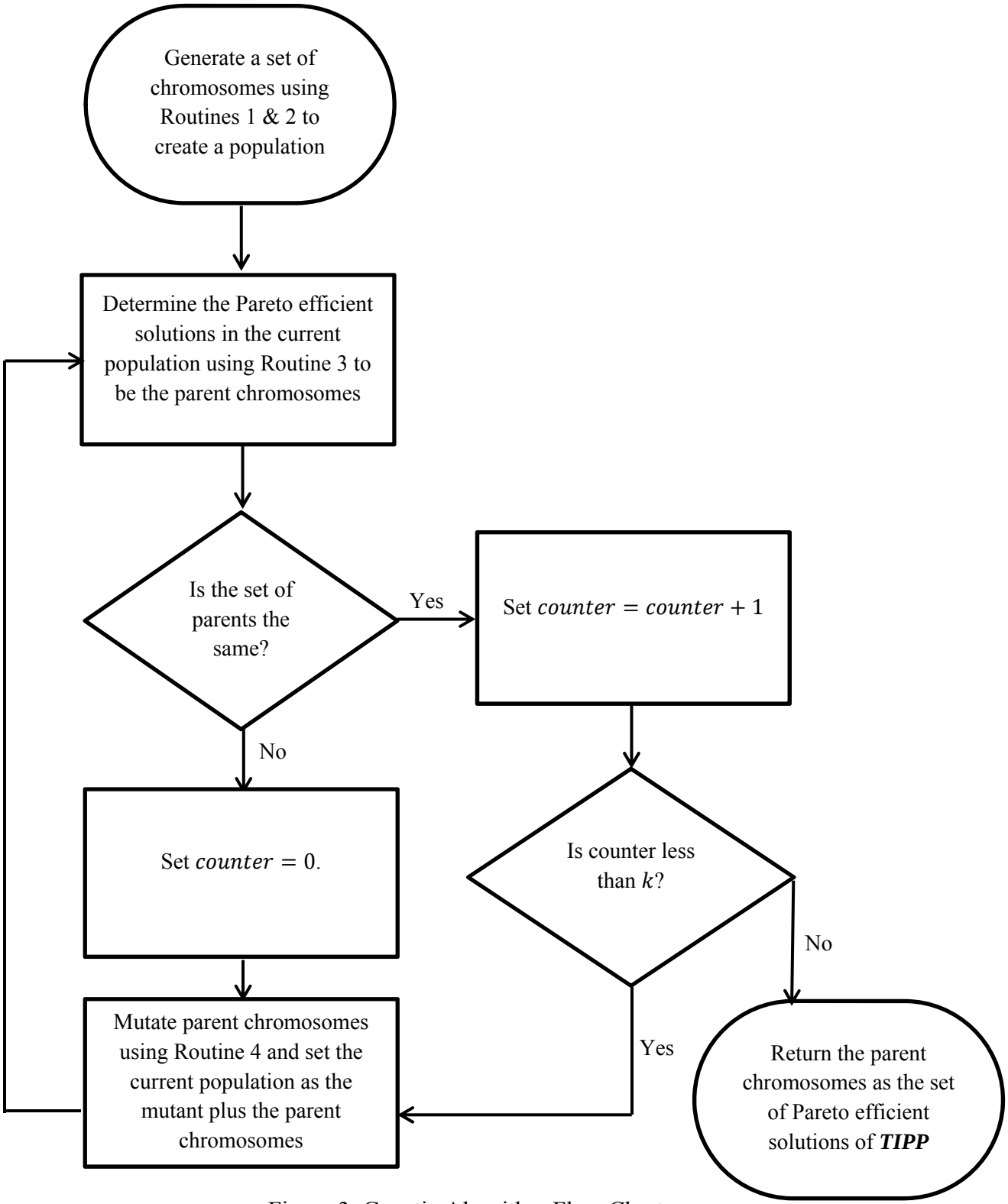


Figure 3. Genetic Algorithm Flow Chart

4. NUMERICAL STUDIES

In this section, the genetic algorithm is used to solve track inspection planning problem on different railroad networks. The convergence properties of the genetic algorithm are analyzed. Furthermore, the genetic algorithm is compared to the greedy heuristic method. Recall that there are two greedy heuristic schedulers, time minimizing and weight maximizing. One can use these schedulers to generate a set of track inspection schedules (note that at most n schedules will be constructed with each greedy approach). Then, Routine 3 can be used to determine the Pareto efficient inspection schedules out of these schedules.

In this section, we randomly generate 10 problem instances for different sizes of railroad networks. Specifically, we consider railroad networks with number of tracks equal to $n = \{100, 150, 200, 250, 300, 350, 400, 450, 500\}$. The problem parameters are randomly generated from uniform distributions denoted as $U[a, b]$ where a is the lower bound of the distribution and b is the upper bound of the distribution.

- Minimum number of required inspections on the tracks are defined to be between 1 and 5, that is, it is assumed that $L_i \sim U[1,5]$.
- Inspection time of a track is defined to be between 2 and 4 hours, that is, it is assumed that $t_i \sim U[2,4]$.
- Minimum time required between two consecutive inspections on a track is defined to be between 25 and 250 hours (i.e., 1 to over 10 days), that is, it is assumed that $\tau_i \sim U[25,250]$.
- Travel time between two tracks is defined to be between 0.5 and 6 hours (considering that a vehicle can travel from one point to the furthest point within Missouri in 6 hours), that is, it is assumed that $\hat{t}_{ij} \sim U[0.5,6]$.
- The importance weight of a track is defined to be between 0 and 1, that is, it is assumed that $w_i \sim U[0,1]$.
- The length of the inspection period will heavily depend on the number of tracks to be inspected on the railroad network. Therefore, in defining the length of the inspection period, we accept the maximum total inspection time resulted by the greedy heuristic as the inspection period length.

Two sets of numerical analysis are considered: convergence of the genetic algorithm and comparison of the genetic algorithm to the greedy heuristic algorithm. All of the algorithms (and the routines needed in them) are coded in Matlab 2014a. Furthermore, the problem instances are solved using 4GB RAM, 2.53 GHz i5 core CPU, and Windows 7 with 64 bit operating system.

For each problem instance solved for each specific network size, the following statistics about the greedy heuristic algorithm are documented.

- The population size ($|P^{GR}|$): The population size of the greedy heuristic algorithm is the total number of inspection schedules constructed using the greedy schedulers. That is, it is the sum of the number of inspection schedules generated with time minimizing greedy scheduler by starting it with each track as the first track to be inspected and the number of inspection schedules generated with weight maximizing greedy scheduler by starting it with each track as the first track to be inspected.

- The Pareto-Front size ($|PF^{GR}|$): The Pareto-Front size of the greedy heuristic algorithm is the total number of Pareto efficient schedules among the schedules generated by the greedy schedulers. Here, Routine 3 is applied on the set of schedules generated by the time minimizing and weight maximizing schedulers to determine the set of Pareto efficient solutions output by the greedy heuristic algorithm.
- The number of iterations (I^{GR}): The number of iterations defines the number of populations evaluated, i.e., the number of times Routine 3 is executed. In greedy heuristic algorithm, this is equal to 1 for any problem instance.
- The computational time (CPU^{GR}): The computational time (in seconds) of the greedy heuristic algorithm is the total time used to determine the final set of Pareto efficient solutions.

For each problem instance solved for each specific network size, the following statistics about the genetic algorithm are documented.

- The average population size ($|P^{GA}|$): The average population size of the genetic algorithm is the average of the population sizes over all iterations of the genetic algorithm. Note that the genetic algorithm might have different number of schedules in each population.
- The Pareto-Front size ($|PF^{GA}|$): The Pareto-Front size of the genetic algorithm is the total number of Pareto efficient schedules returned by the genetic algorithm at termination.
- The number of iterations (I^{GA}): The number of iterations defines the number of populations evaluated, i.e., the number of times Routine 3 is executed. In genetic algorithm, this is equal to the number of populations evaluated until termination.
- The computational time (CPU^{GA}): The computational time (in seconds) of the genetic algorithm is the total time used to determine the final set of Pareto efficient solutions.

4.1. Convergence of the Genetic Algorithm

To analyze the convergence of the genetic algorithm, we first demonstrate how the numbers of Pareto efficient inspection schedules in each population change over the iterations. Figure 4 in Appendix A illustrates the numbers of Pareto efficient solutions in each population, i.e., the size of the parent chromosomes, evaluated by the genetic algorithm for each of the 90 problem instances solved. As can be seen in Figure 4, the numbers of Pareto efficient schedules in the populations increase over the iterations. That is, the genetic algorithm generates more Pareto efficient track inspection schedules as the iteration number increases.

Furthermore, we document the intermediate Pareto-Fronts when the 25%, 50%, 75% and 100% iterations are completed. Figure 5 in Appendix B illustrates the Pareto-Fronts at intermediate iterations of the genetic algorithm for each of the 90 problem instances solved. As can be seen in Figure 5, the Pareto efficient solutions within the populations are improving over time. This tells that the genetic algorithm finds better track inspection schedules, i.e., schedules with lower total times and higher total weights, as more iterations are executed.

Finally, we document how the average total weight to total time ratio is changing over iterations of the genetic algorithm. In particular, the ratio of total weight to total time of a schedule Y, Z is defined as follows:

$$H(Y,Z) = \frac{TW(Y)}{TT(Y,Z)}.$$

Note that if a schedule has high $H(Y,Z)$ ratio, it indicates that this schedule is an efficient track inspection schedule as it is capable of inspecting important tracks in less times. The $H(Y,Z)$ can be used as a measure of performance of the inspection schedules. Figure 6 given in Appendix C documents how the averages of the $H(Y,Z)$ ratios of the Pareto efficient solutions in each iteration of the genetic algorithm change over iterations. As can be seen in Figure 6, the average $H(Y,Z)$ ratios are increasing over the iterations. This suggests that the genetic algorithm determines more efficient inspection schedules over iterations.

Recall that both the genetic algorithm and the greedy heuristic algorithm suggest a set of inspection schedules to the decision maker. Then, the decision maker should select one schedule from the set of alternatives proposed. The $H(Y,Z)$ ratio can be used to make a final selection decision by the decision maker. Specifically, the decision maker can select a schedule with high $H(Y,Z)$ ratio to guarantee that inspection is executed efficiently considering the importance of the tracks within reasonable total inspection time.

4.2.Comparison of Genetic Algorithm and Greedy Heuristic Algorithm

Genetic algorithm and the greedy heuristic algorithm are compared in terms of two aspects: quantitative aspects and qualitative aspects.

4.2.1. Quantitative Comparison

In quantitative comparison, the iteration numbers, population sizes, number of Pareto efficient solutions returned (i.e., the size of the Pareto-Fronts), and the computational times (in seconds) are compared. Specifically, we also document whether the genetic algorithm or the greedy heuristic algorithm returns more Pareto efficient solutions.

Tables 2-10 given in the Appendix D summarize the quantitative comparison of the problem instances solved with each network size $n = \{100, 150, 200, 250, 300, 350, 400, 450, 500\}$.

Table 11 illustrates the quantitative comparison of the genetic algorithm and greedy heuristic algorithm on average, i.e., over all 10 problem instances solved within each network size.

Table 11. Average Values for Quantitative Comparison for Each Network Size

	$Greedy$ Heuristic Algorithm				Genetic Algorithm					
n	$\|P^{GR}\|$	I^{GR}	$\|PF^{GR}\|$	CPU^{GR}	$\|P^{GA}\|$	I^{GA}	$\|PF^{GA}\|$	CPU^{GA}	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
100	200	1	30.2	8.6	3602.8	286	209.1	13,094	0%	100%
150	300	1	31.2	10.5	2723.8	167.4	164.8	8,109	0%	100%
200	400	1	31	15.6	2619.3	169.4	155.7	10,649	0%	100%
250	500	1	32.2	20.2	2713.3	147.8	149.9	13,068	0%	100%
300	600	1	36.4	28.1	2891.9	173.1	160.2	19,858	0%	100%
350	700	1	37.3	40.7	2704.5	159.9	141.4	22,966	0%	100%
400	800	1	41.9	57.2	2477.5	126.7	130.4	19,742	0%	100%
450	900	1	41	74	2653.8	141	137.2	28,730	0%	100%
500	1000	1	38.9	91.6	2649.3	147.1	130.1	33,180	0%	100%
Avg.	600	1	35.6	38.5	2781.8	168	153.2	18,822	0%	100%

The following observations are due to Table 11.

- For each network size n, the genetic algorithm evaluates more inspection schedules on average for each population. Specifically, the greedy heuristic algorithm evaluates one population and the population size was 2n for any problem instance solved; therefore, on average the greedy heuristic algorithm has 600 inspection schedules. On the other hand, the genetic algorithm evaluates over 2700 inspection schedules on average per population.
- As noted and can be seen in Table 11, the greedy heuristic algorithm terminates after the first iteration. On the other hand, the genetic algorithm terminates after 168 iterations on average. That is, 168 populations are evaluated on average.
- As expected and can be seen in Table 11, the genetic algorithm requires more computational time until termination. Specifically, while the greedy heuristic algorithm takes 38 seconds on average, the genetic algorithm takes over 18,800 seconds on average.
- The genetic algorithm returns more Pareto efficient inspection schedules than the greedy heuristic algorithm. In particular, the greedy heuristic algorithm returns 35 inspection schedules on average while the genetic algorithm returns over 150 inspection schedules on average. That is, the genetic algorithm is able find more Pareto efficient track inspection schedules on average.
- Finally, Table 11 summarizes the percentage of the problem instances solved such that the genetic algorithm finds more Pareto efficient solutions than the greedy heuristic algorithm. While having more Pareto efficient solutions on average does not mean that the genetic algorithm finds more Pareto efficient solutions for each problem instance solved, it can be seen in Table 11 that for 100% of the problem instances, the genetic algorithm finds more Pareto efficient solutions than the greedy heuristic algorithm.

Based on the quantitative comparison, it can be concluded that the genetic algorithm finds more Pareto efficient solutions in all of the problem instances at an expense of increased

computational time. However, the computational times of the genetic algorithm are still reasonable for planning problems.

4.2.2. Qualitative Comparison

In qualitative comparison, the average $H(Y,Z)$ values are compared along with how the Pareto-Front of the genetic algorithm performs compared to the Pareto-Front of the greedy heuristic algorithm. That is, in addition to comparing the average $H(Y,Z)$ values over the Pareto efficient solutions returned, the sets of Pareto efficient solutions are compared. Let H^{GA} and H^{GR} denote the average of the $H(Y,Z)$ values of the Pareto efficient inspection schedules over the Pareto efficient inspection schedules determine by the genetic algorithm and greedy heuristic algorithm, respectively.

Let the Pareto-Front of genetic algorithm be denoted by PF^{GA} and Pareto-Front of greedy heuristic algorithm be denoted by PF^{GR}. The comparison is made based on the Pareto dominance between two final Pareto-Fronts. In other words, the following rule is applied:

Unless $PF^{GA} \equiv PF^{GR}$, PF^{GA} dominates PF^{GR} if $PF(PF^{GA} \cup PF^{GR}) \equiv PF^{GA}$.

Note that $PF(PF^{GA} \cup PF^{GR})$ defines the set of Pareto efficient inspection schedules within the set of inspection schedules returned by the genetic algorithm and the greedy heuristic algorithm. If $PF(PF^{GA} \cup PF^{GR}) \equiv PF^{GA}$ when $PF^{GA} \neq PF^{GR}$, it implies that the inspection schedules returned by the genetic algorithm are Pareto superior compared to the inspection schedules returned by the greedy heuristic algorithm; thus, the Pareto-Front of the genetic algorithm dominates the Pareto-Front of the greedy heuristic algorithm. Dominance relation between two Pareto Front is shown by the sign $\gg$ in which, if the final Pareto-Front of the genetic algorithm dominates the Pareto Front of the greedy heuristic algorithm, we will write $\mathrm{PF}^{\mathrm{GA}} \gg \mathrm{PF}^{\mathrm{GR}}$ and if the final Pareto-Front of the greedy heuristic algorithm dominates the Pareto Front of the genetic algorithm, we will write $\mathrm{PF}^{\mathrm{GR}} \gg \mathrm{PF}^{\mathrm{GA}}$. We note that it is possible not to have neither $\mathrm{PF}^{\mathrm{GA}} \gg \mathrm{PF}^{\mathrm{GR}}$ nor $\mathrm{PF}^{\mathrm{GR}} \gg \mathrm{PF}^{\mathrm{GA}}$. In such a case, neither $\mathrm{PF}^{\mathrm{GA}}$ nor $\mathrm{PF}^{\mathrm{GR}}$ dominates the other one, which is shown as $PF^{GR} \sim PF^{GA}$.

For notational simplicity, we let $PF(PF^{GA} \cup PF^{GR}) = PF^{U}$. In addition to the above statistics, we also document the percentage of the Pareto efficient inspection schedules in PF^{U} that are coming from PF^{GA} and the percentage of Pareto efficient inspection schedules in PF^{U} that are coming from PF^{GR}. Tables 12-20 given in the Appendix E summarize the qualitative comparison of the problem instances solved with each network size $n = \{100, 150, 200, 250, 300, 350, 400, 450, 500\}$. Table 21 illustrates the qualitative comparison of the genetic algorithm and greedy heuristic algorithm on average, i.e., over all 10 problem instances solved within each network size.

Table 21. Average Values for Qualitative Comparison for Each Network Size

n	H^{GR}	H^{GA}	$H^{GR} \geq H^{GA}$	$H^{GR} < H^{GA}$	$\lvert PF^U \rvert$	GR % in $\lvert PF^U \rvert$	GA % in $\lvert PF^U \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
100	0.1065	0.1149	0%	100%	209.1	0%	100%	0%	100%	0%
150	0.1086	0.1163	0%	100%	164.8	0%	100%	0%	100%	0%
200	0.1164	0.1207	0%	100%	155.7	0%	100%	0%	100%	0%
250	0.1184	0.1220	0%	100%	149.9	0%	100%	0%	100%	0%
300	0.1195	0.1235	0%	100%	160.2	0%	100%	0%	100%	0%
350	0.1217	0.1244	0%	100%	141.4	0.23%	99.94%	0%	90%	10%
400	0.1236	0.1246	0%	100%	130.3	0.28%	99.85%	0%	90%	10%
450	0.1243	0.1254	0%	100%	137.3	0.19%	100%	0%	90%	10%
500	0.1240	0.1261	0%	100%	130.2	0.54%	99.91%	0%	90%	10%
Avg.	0.1180	0.1220	0%	100%	153.2	0.14%	99.97%	0%	96%	4%

The following observations are due to Table 21.

- For each network size n, the Pareto efficient inspection schedules returned by the genetic algorithm have higher $H(Y,Z)$ values than the Pareto efficient inspection schedules returned by the greedy heuristic algorithm on average. This suggests that the inspection schedules returned by the genetic algorithm are more effective and more preferable. This can also be observed in Figure 6 given in Appendix C.
- In all of the problem instances solved for each network size, the Pareto efficient inspection schedules returned by the genetic algorithm had higher average $H(Y,Z)$ values.
- In 96% of the problem instances solved, the set of Pareto efficient inspection schedules returned by the genetic algorithm dominates the set of Pareto efficient inspection schedules returned by the greedy heuristic algorithm. In particular, while the set of Pareto efficient inspection schedules returned by the greedy heuristic algorithm never dominated the set of Pareto efficient inspection schedules returned by the genetic algorithm, they were indifferent only in 4% of the problem instances.
- The union set of Pareto efficient inspection schedules consists of over 99% of the Pareto efficient solutions returned by the genetic algorithm on average. This suggests that the inspection schedules returned by the genetic algorithm have better total time and better total weight performances. The Figure 7 given in Appendix F demonstrates the comparison of the Pareto efficient inspection schedules returned by the genetic algorithm and the greedy heuristic algorithm. As can be seen in Figure 7, in all of the problem instances solved, the genetic algorithm was able to find solutions with lower total times and higher total weights for track inspection planning problem.

Based on the qualitative comparison, it can be concluded that the genetic algorithm finds better inspection schedulers, i.e., inspection schedules with lower total time and higher total weights. Furthermore, the total weight to total time ratio is higher in the inspection schedules returned by the genetic algorithm. This suggests that the genetic algorithm is a better method for solving the track inspection planning problem.

5. IMPLEMENTATION DETAILS

Inspection of tracks is the defect detection process of the railroads (Office of Railroad Safety, 2011). According to the Office of Safety Assurance and Compliance (2002), inspections must be performed based on a pre-planned schedule. These inspections are categorized into (1) visual inspection performed by inspectors, (2) automated track inspection program (ATIP) which is effective after 2012. Assuming ATIP is a non-destructive process, the primary technologies to do so are ultrasonic and inductive methods. According to the Office of Railroad Safety (2011), ultrasonic method is based on sound waves and induction method is based on magnetic field. Ultrasonic technology is the most frequently used method, and induction is currently used as a complementary system to the ultrasonic method.

This work is mostly applicable to ATIP; therefore, we review the related methods to associate risk of failure using the track inspection results. In Appendix G, we provide a user's guide along with the description of the Matlab 2014a function files that include the codes for the solution methods discussed for the track inspection planning problem. Figures 8 and 9 given in Appendix G illustrate the input and output Excel files for the algorithms.

All tracks must be inspected according to Table 22. However, tangent tracks 600 feet or less constructed of concrete crossties, which is a rectangular support for the rails in railroad tracks, including but not limited to isolated track segments, experimental or test track segments, highway-rail crossings, and wayside detectors, are excluded.

Table 22. Minimum Inspection Requirements per Year Based on Class and Type of Tracks*

Class of Track	Freight Trains Operating Over Track With An Annual Tonnage of:			Passenger Trains Operating Over Track With An Annual Tonnage of:	
	Less than 40 mgt	40 to 60 mgt	Greater than 60 mgt	Less than 20 mgt	Greater than or equal to 20 mgt
5 or above	2	3	4	3	3
4	2	3	4	2	3
3	1	2	2	2	2
2	0	0	0	1	1
1	0	0	0	0	0
Excepted track	0	0	0	N/A	N/A

* If a track is subject to both freight and passenger traffic, the higher frequency is applied in case of two different frequency values.

We note that the Department of Transportation Records (2014) imply slightly tighter bounds on the inspection intervals for track classes 3, 4 and 5. As long as they provide the frequency

and the inspection intervals, one can define the parameters needed to implement the Track Inspection Planning Problem stated in this project.
The class of a track is determined based on several factors. One of the most important factors to classify the tracks is the operational speed. Table 23 shows the operational speed for different classes of tracks. Table 24 shows the track classification based on the track geometry. One can find the classes of tracks by referring to Tables 23 and 24.

Table 23. FRA Track Classes based on Operating Speed

FRA track class	Maximum Allowable Operating Speed (miles/hour) for Freight Train is:	Maximum Allowable Operating Speed (miles/hour) for Passenger Train is:
Excepted Track	10	N/A
Class 1	10	15
Class 2	25	30
Class 3	40	60
Class 4	60	80
Class 5	80	90

Table 24. FRA Track Classes based on Track Geometry

FRA track class	Minimum Number and Type of Crossties Each 39 feet segment of track should have:	
	Tangent track and curved track less than or equal to 2 degrees	Turnouts and curved track greater than 2 degrees
Class 1	5	6
Class 2	8	9
Class 3	8	10
Class 4,5	12	14

The frequency and the inspection intervals given in Table 22 can be translated to the minimum number of inspections required for each track $i \in I$, that is, L_i values, and the required time between two consecutive inspections for each track $i \in I$, that is, τ_i values, respectively. The longest frequency noted in Table 22 is one inspection per year, that is, the minimum number of inspections required is one and the time between two consecutive inspections is one year (or 365 days) for class 4 and 5 tracks with concrete crossties when the inspection period is one year. The most frequent inspection is twice per week with one day between consecutive inspections for other types of class 4 and 5 tracks. For this situation, the minimum number of inspections required is 2 per week or 104 per year and time between two consecutive inspections is one day. Using the class of track segments and ATIP requirements, one can define L_i and τ_i values for each track segment to be inspected.

Each track segment requires a specific time for inspection denoted by $t = \{t_i\}_{i=1}^{n}$; vector of the track inspection times. This time defines the time it takes for the track inspection vehicle to inspect the specific track segment: it might be defined depending on the length of the track and the speed of the inspection vehicle on tracks; hence, one can define this parameter easily. There is also a travel time between two track segments denoted by $\hat{t}_{ij}$ for track i and track j. To be more precise, the inspection vehicle might go from one track segment in a region to another track segment, and the time for reaching from one segment to another is the travel time between the segments. It might be defined considering the distance between track segments and the average travel speed of the inspection vehicle on road. Note that one can assume $\hat{t}_{ij} = \hat{t}_{ji}$; however, this is not required as a condition in the algorithms proposed.

Each track segment has an individual importance for being inspected. One might define the importance of inspection of a track segment depending on the likelihood of defects/failure on that segment. In this sense, the concept in the reliability analysis discussed in the next section may enable finding the importance weight of the tracks by using intensity function or hazard function for each track.

6. RISK MEASUREMENT ANALYSIS

In this section, we propose use of different approaches for associating risk with the tracks after their inspection. Specifically, our focus is on three approaches: reliability, defect development, and crack growth. Based on our discussion, crack growth method is the most suitable for this project as it is able to utilize the inspection results, which we consider as the crack size detected in the track segments, the time until next inspection, and the accuracy of the inspection vehicle. A set of rules are suggested to define risk based on the crack growth approach.

6.1.Reliability Approach

Failure can be quantified using the concepts in reliability analysis (Merrick et al., 2005; Shyr et al., 1996) or probabilistic assessment (Rocha et al., 2014). In the reliability analysis, one deals with the failure using concepts such as “Intensity function”, “Survival Function”, and “Hazard Function” (Merrick et al., 2005; Shyr et al., 1996).

The tracks with no defects are survived tracks, modelled by the survival function, while defective tracks are perished ones, modelled by the hazard function (Shyr et al., 1996). Therefore, hazard rate is an indicator of defect rate of a track which is not detected with a defect. By this definition, the hazard rate can be used to determine the priority of tracks to be inspected (importance weight of the track inspections).

In one approach to calculate the hazard rate (Shyr et al., 1996), one uses the probability of the defect regime being of type j, and the conditional hazard rate given a type j defect regime. Then hazard rate can be calculated as

$$h_X(x) = \sum_{j=1}^{J} h_{X|j}(x) P(j|x),$$

where $P(j|x)$ is the probability of rail being in type j defect regime at usage level x; and $h_{X|j}(x)$ is the conditional hazard rate given the rail is in type j defect regime. Shyr et al., (1996) introduced a method to calculate $P(j|x)$ and $h_{X|j}(x)$. In order to use this method, one needs to know the different type of defects presented in (Office of Railroad Safety, 2011) along their hazard functions.

Unlike Shyr et al., (1996) that focused on hazard rate to indicate defect rate of an un-defected track, Merrick et al. (2005) investigated the number of defects in a given track. According to Merrick et al., (2005), the number of defects occurring in rail tracks is a function of the cumulative traffic usage, which follows a non-homogenous Poisson point process (NHPP). For this purpose, they introduced a parametric intensity function; NHPP. In this function, the number of defects occurring in track (N) is a function of cumulative traffic usage (t), by parameters L, α, γ which is denoted by:

$$(N(t)|\alpha,\gamma,L) \sim NHPP(L\alpha\gamma t^{\gamma-1})$$

where, L denotes length of segment of network, α is a covariate that varies from one rail to another and is conditionally independent (Merrick et al., 2004). Large number of covariate parameters (for each track, there exists a different value of α) in this function may cause difficulty to model the defect rate in the tracks.

Different than reliability analysis, one can use simulation and probability models to analyze the failure in tracks. Similar work has been done on failure of railroad bridges, in which, the safety of bridges for high speed trains are assessed (Rocha et al., 2014). Their work can also be applied into inspection and maintenance process by defining an appropriate procedure. As they have suggested, the first stage is defining the basic random variables, as well as their distribution. These variables should include all the parameters which their change affects the dynamic behavior of the inspection-Maintenance process. The second stage is to generate the values for each variable of simulation. At the third stage, one analyzes the outcome using an analytic approach.

Reliability based approaches depend on very detailed probability calculations and they generally focus on failure rate calculations of tracks with no observed defects instead of associating a risk of failure using the data observed after inspection and the time until next inspection. Specifically, as noted by Kashima (2004), the observations after track inspections are based on the crack size on the tracks. Therefore, we next focus on approaches that utilize defect information gathered after inspections.

6.2.Defect Development Approach

Failure in a track segment generally follows a process from an undetected crack into defects, from a defect to a failure, and from a failure to breakage (Kashima, 2004; Peterz, 2004). When a crack in a rail segment starts to grow it may be undetected at first but, after some time, it will eventually be detectable (Kashima, 2004; Peterz, 2004). Peterz (2004) defined the interval between the time of the detection of a crack and the time of failure as a P-F interval. A P-F interval is presented in Figure 10 below, where Tinit is the initialization of a crack, Tdet is the time when a crack is detected, and Tcrit is the time the crack reaches the critical size that causes failure. It is discussed that the P-F interval has an exponential distribution and to prevent the breakage, they suggested an inspection procedure during the P-F interval.

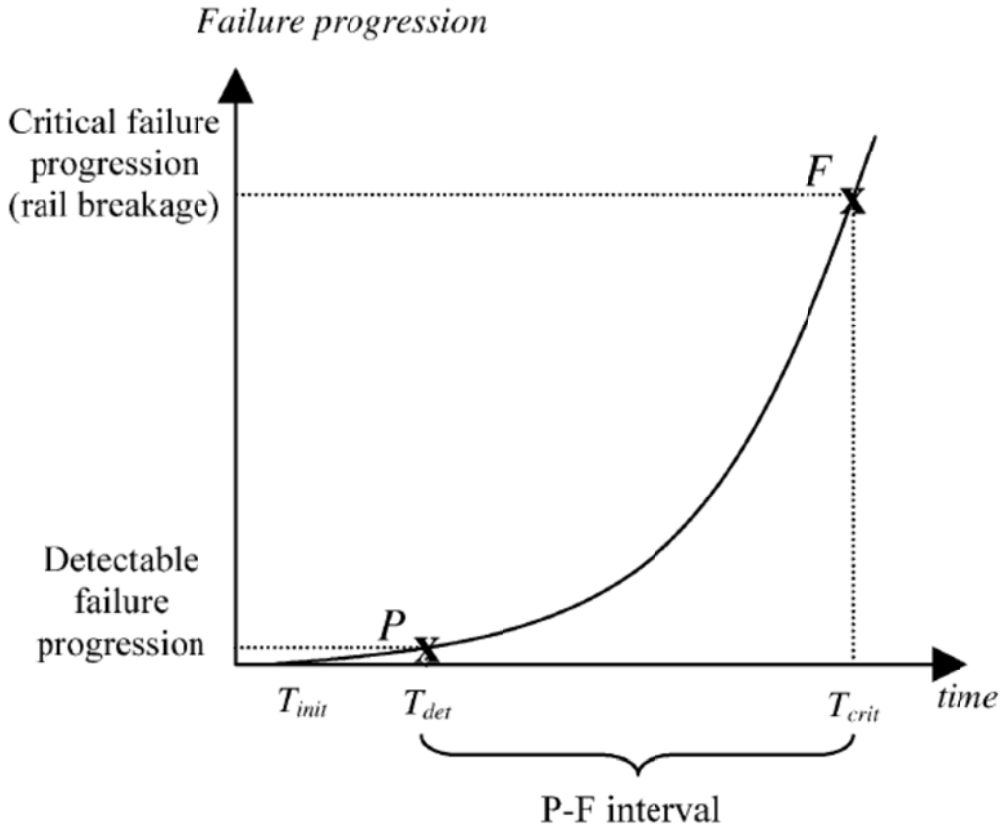


Figure 10. P-F Interval Illustration (Podofillini et al., 2006)

Inspection process by definition is trying to detect the defects in track to prevent the failure in the railroad network. However, inspection is not perfect (Kashima, 2004; Peterz, 2004), i.e., it cannot always detect the defects. More importantly, even if the inspection is based on advanced technologies, such as ultrasonic inspection, there is a chance of not detecting the defect. Therefore, the time between consecutive inspections of the same track can set as the time period, in which an undetected crack might develop into a failure (Peterz, 2004). The risk of failure until the next inspection is, therefore, needed for appropriate policy implementation for the track maintenance planning.

6.2.1. Risk Definition

According to Kashima (2004), risk in an interval between consecutive inspections of a segment of a track can be calculated by:

$$Risk = \sum_{r=1}^{R} S_r \times N_r,$$

where S_r is the severity of failure type r and N_r is the total expected number of failures of type r in the interval between two consecutive inspections such that $r = 1, \dots, R$ represents the different types of failure. In (Anderson et al., 2003) one can find the different types of failures in tracks that can lead to a breakage. Based on (Zhao et al., 2007), the severity is defined as the percentage of failures that leads to breakage such as derailment. Severity is constant for a period of time when the track structure and the track operating speed do not change (Zhao et al., 2007). Different studies have used different values for severity (Zhao et al., 2007; Podofillinia at al., 2005; Anderson at al., 2003]; however, values less than one can be used for severity. For more information regarding the total number of accidents and breakage (derailments), one may review department of transportation manual on railroad safety (2002). Next, we discuss how to calculate the expected number of failures on a track segment between two consecutive inspections on that track segment.

6.2.2. Calculating the Expected Number of Failures between Inspections

Based on Zhao et al. (2007[a,b]), consider the inspection intervals $\langle t_1, t_2, \dots, t_{j-1}, t_j, \dots, t_m \rangle$ of a track segment and failure types $r = 1, \dots, R$ for the same segment. We are trying to find the risk on the same segment in the inspection interval $[t_{j-1}, t_j]$. Based on our discussion on the development of failure, we can consider two cases:

- In the first case, defects occur in interval $[t_{j-1}, t_j]$ and lead to a failure in the same interval
- In the second case, defects occurred before the interval $[t_{j-1}, t_j]$ and leads to a failure in that interval.

Then, the expected number of failures due to defect type r (EF_r) in interval $[t_{j-1}, t_j]$, considering both cases, can be calculated based on the following formula (Zhao et al., 2007):

$$EF_r(t_{j-1}, t_j) = \sum_{k=1}^{j} \left\{ (1-\beta)^{j-k} \int_{t_{k-1}}^{t_k} \gamma(\tau) \left[G_r(t_j - \tau) - G_r(t_{j-1} - \tau) \right] d\tau \right\}.$$

In this formula, $\gamma(\tau)$ is the rate of type r defect occurrence at time τ. This rate is also called the intensity function (Merrick et al., 2005). β is the detection rate. In addition, $G(x)$ denotes the cumulative distribution function of delay time being less than x, i.e., the time until defect initialization. It should be noted that

$$\int_{t_{j-1}}^{t} \gamma(\tau) G_r(t-\tau) d\tau = \int_{t_{j-1}}^{t} \gamma(\tau) P\{Y_r \le t-\tau\} d\tau,$$

where Y_r is delay time for type r defect.

Based on the above equations, if all possible failure types are considered, the risk of a track between two consecutive inspections can be calculated as follows:

$$\begin{aligned} Risk(t_{j-1}, t_j) &= \sum_{r=1}^{R} S_r \times EF_r(t_{j-1}, t_j) \\ &= \sum_{r=1}^{R} \sum_{k=1}^{j} S_r \times \left\{ (1-\beta)^{j-k} \int_{t_{k-1}}^{t_k} \gamma(\tau) \left[G_r(t_j - \tau) - G_r(t_{j-1} - \tau) \right] d\tau \right\}. \end{aligned}$$

Note that one needs the detection rate for calculating the risk. Next, we discuss how detection rate of inspection can be determined.

6.2.3. Detection Rate

Detection rate of a defect is denoted by β. This parameter can be calculated based on practical observations or using regression formulae. Introduced by (Zhao et al., 2007), the following formula can help calculate the detection rate of defects:

$$\beta = 1 - \exp\left[-5 \frac{A(t) - A_1}{A^* - A_1}\right],$$

Where

A^*: 75 % of the rail head area (HA)
A_1: 5 % of the rail head area (HA)
t: time
$A(t) = A_1 + A^* \frac{t}{L_{PF}}$,

such that $A(t)$ defines the size of the crack in terms of percentage of the rail head area and L_{PF} is the mean of the P-F interval.

However, Zhao et al. (2007) used an average detection rate, which is 0.7 for ultrasonic inspection. In practice, we have to either use this value or values similar to it suggested by (Zarembski et al., 2005), or calculate the average, which should be calculated based on:

$$\beta_{ave<t_1;t_2>} = \frac{\int_{t_1}^{t_2} d\beta}{t_2 - t_1}.$$

Other than the calculation methods, the data on the inspection vehicle (inspection method) can also be used to estimate the defect detection probabilities of an inspection method. Figure 11 illustrates how the detection probability of different inspection methods change depending on the crack size. It is not surprising that as the crack size increases, the probability of detection increases as well.

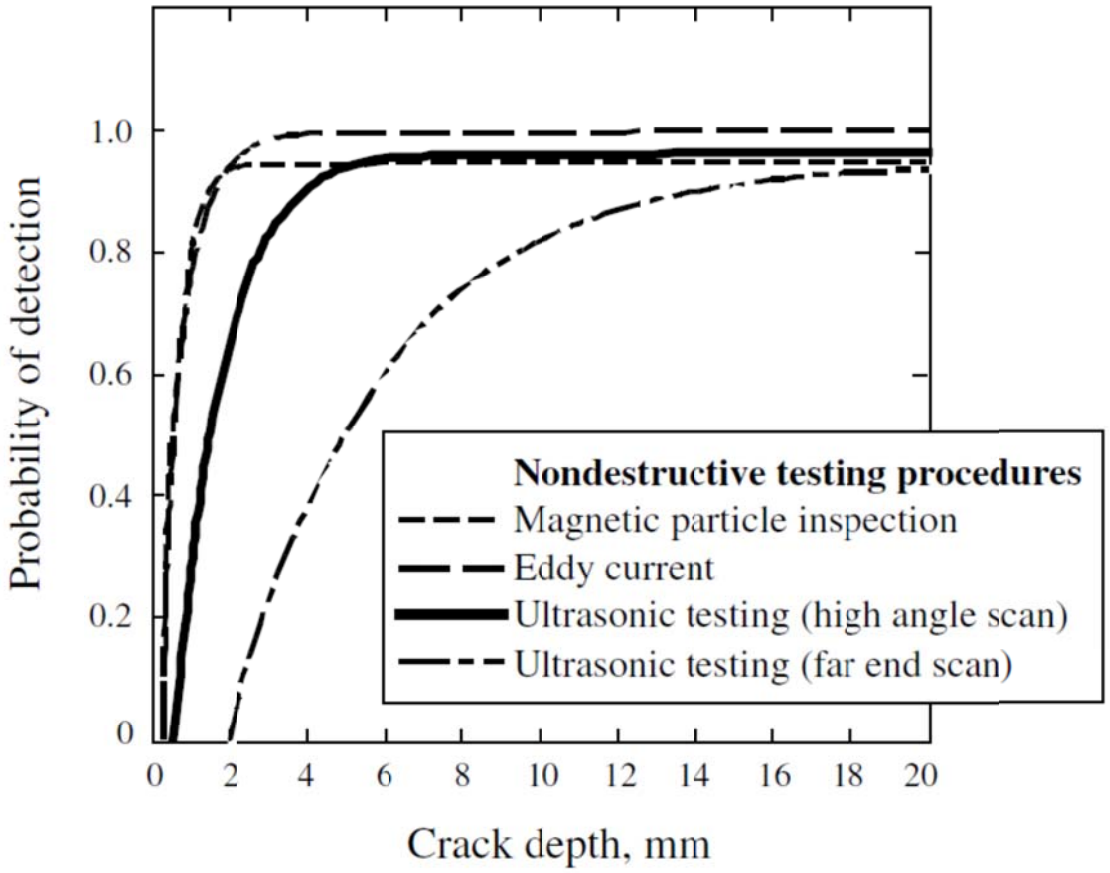


Figure 11. Detection Probabilities for Different Inspection Methods (Zerbski et al., 2005)

Furthermore, there are studies that suggest use of specific probability distributions for probability of detection (POD). For instance, Zheng and Ellingwood (1998) suggest use of exponential distribution while Zhao and Haldar (1994) recommend the use of a lognormal distribution. Kashima (2004) utilizes a lognormal distribution for POD depending on the crack size as shown in Figure 12.

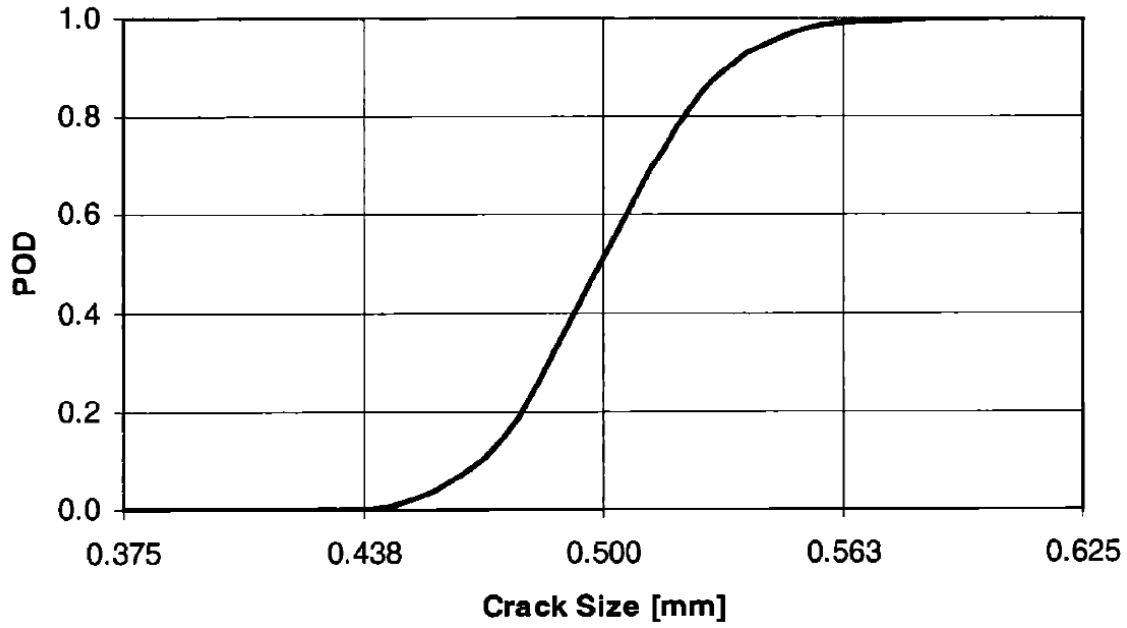


Figure 12. Lognormal Detection Probabilities of Cracks (Kashima, 2004)

6.3.Crack Growth Approach

As is discussed previously, track inspection is not capable of detecting all of the cracks on track segments. This imperfection should be taken into account while associating a risk value for a track segment after each inspection until its next inspection. In doing so, one can relate how a crack develops into a failure and the estimated lifetime of a track segment. In the following discussion, we develop a risk measurement method after each inspection considering the following two cases:

- A crack is detected during the inspection,
- A crack is not detected during the inspection.

Prior to the analysis of each case, we next discuss the crack growth and track lifetime models as preliminary analysis for risk measurement.

6.3.1. Crack Growth and Track Lifetime Models

Crack growth process is usually studied based on Paris' Law. This law is stated as follows:

$$\frac{da}{dN} = C(\Delta K)^m,$$

where a is the size of crack, N is the number of stress cycles, C and m are material parameters. According to Kashima (2004), the crack growth in tracks follows the Paris's Law. For steel components, C and m are equal to 3 and 10^{-11}, respectively (these values will vary depending

on the material Zhao et al., 2007). In addition, ΔK is the stress intensity factor range and can be calculated using the following formula:

$$\Delta K = K_{max} - K_{min} = YS\sqrt{\pi a},$$

In the above equation, Y is the geometric correction factor and S is the tensile stress range, which is around 45 percent of the ultimate tensile stress for a normal quality railroad track and the ultimate tensile stress is $32\ kg/mm^2$. In Peters (2004), we can find different formulae to calculate the geometric correction factor Y for small length of cracks. For example,

$$Y = \sqrt{\sec\left(\frac{\pi a}{w}\right)}\ , 0 < a \leq 0.5,$$

or

$$Y = \sqrt{\frac{w}{\pi a}\tan\left(\frac{\pi a}{w}\right)}\ , 0 < a \leq 0.5,$$

where, w is the width of plate in millimeters (mm). Using these equations, one then can find a more general form of Paris' Law for the track cracks as follows:

$$N(a_0, a_c) = \begin{cases} \int_{a_0}^{a_c} \frac{1}{CS^m}\left(\frac{\pi a}{\cos\left(\left(\frac{\pi a}{w}\right)\right)}\right)^{-\frac{m}{2}} da\,, & 0 < a \leq 0.5 \\ \int_{a_0}^{a_c} \frac{1}{CS^m}(\pi a)^{-\frac{m}{2}} da\,, & a > 0.5 \end{cases}$$

$N(a_0, a_c)$ defines the number of stress cycles for the crack to grow from length a_0 to a_c (length of a in mm). Furthermore, using the train traffic data on the track segment, the number of stress cycles can be used to estimate the time for a crack to grow from length a_0 to a_c (where a stress cycle implies a load on the track).

Using the crack growth rate, the lifetime of a track segment after crack initialization can be determined. In particular, let us define the following parameters:

- α: the crack growth rate (calculated based on Paris's law),
- AC : critical crack size (if a crack reaches size AC, the track segment is assumed failed)

Then, the following graph, given in Figure 13, depicts the lifetime of a track segment. Therefore, the time to failure after crack initialization will amount to $\frac{AC}{\alpha}$. For instance, if one knows that the total time of a track from the start of its use until failure is LT, the crack initialization time is $LT - \frac{AC}{\alpha}$. Furthermore, given the size of a crack of size A, one can determine the time until failure as $\frac{AC-A}{\alpha}$.

In most materials and systems, the lifetime of a system is generally modeled using a probability distribution. Let LT denote the lifetime of a track. Then, based on the material of the track, one can define the probability of failure at time T as $Pr(LT = T)$. At this point, one needs to determine the probability density function for the lifetime of a track. It is common that exponential distributions are used to model system/material lifetimes. If we assume that the lifetime of a track segment is exponentially distributed, $Pr(LT = T)$ decreases exponentially as T increases. Figure 14 shows the probability density function for a track's lifetime in case of exponential distribution is used for modeling the lifetime of the track.

Next, we explain how to measure risk using the crack growth, track lifetime, and probability of detection for each of the cases described above.

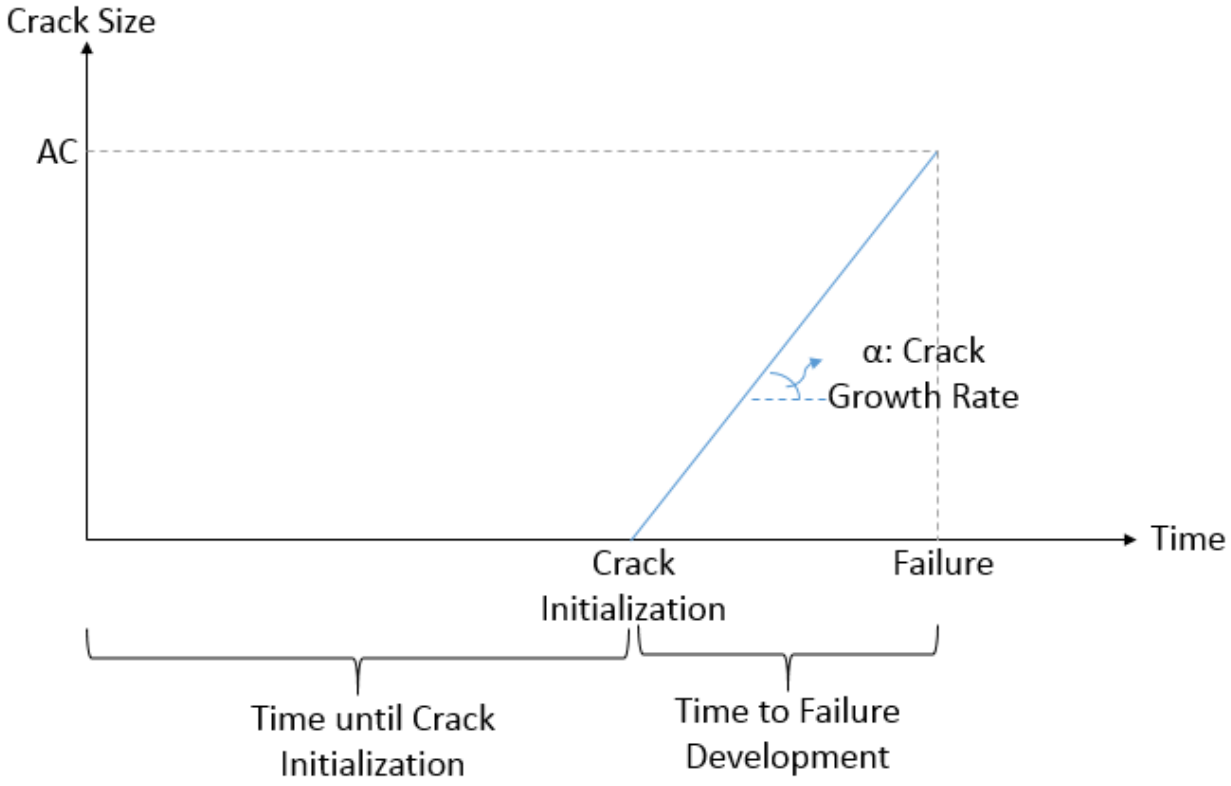


Figure 13. Crack Growth over Time

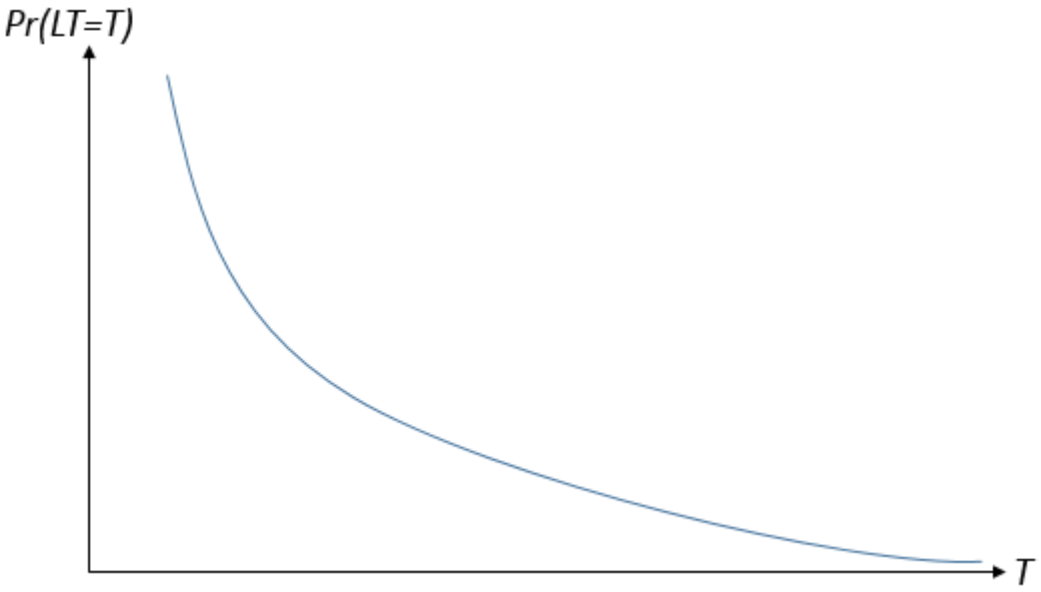


Figure 14. Exponentially Distributed Track Lifetime

6.3.2. Risk Measurement when Crack Is Detected

Right after inspection of a track, it is possible that the inspection detected a crack. In this case, the crack size is measured. Let us assume that the crack size is A. Depending on the crack size, the Track Safety Standards Compliance Manual (FRA, 2002) details the actions that should be taken. In Appendix H, these actions are summarized for different defect types and their sizes (see Table 25).

We note that it is possible that, depending on the size of crack, the actions noted in the Track Safety Standards Compliance Manual (FRA, 2002) might suggest delaying maintenance or waiting until next inspection. At this point, given the crack size observed, one may determine the time it will take this crack to reach the critical crack size AC. As noted above, given crack size A, the time it will take this crack to reach the critical size is $\frac{AC-A}{\alpha}$. Furthermore, given an inspection schedule, we know the time until the next inspection of the track, which is inspected and detected with a crack. Therefore, we can recommend the following actions:

Given a track is inspected and a crack size of A is detected:

1) Determine the time until the next inspection, denoted by T1
2) Determine the time until the critical crack size, denoted by T2 where $T2 = \frac{AC-A}{\alpha}$
    a. If T1<T2, follow the action given in the Track Safety Standards Compliance Manual (FRA, 2002).
    b. If T1>T2, schedule inspection on the track earlier than T2.

6.3.3. Risk Measurement when Crack Is Not Detected

Recall that it is possible for the inspection vehicle to not detect a crack even if there is a crack on a track. In this case, one should consider the probability of failure until the next inspection on that track. Therefore, we next develop a risk measure considering the probability of failure until the next inspection. In doing so, we use the track lifetime probabilities, probabilities of not detecting a crack, and the crack growth rates.

In particular, let us define the probability of no detection given that there is a crack size of A. As suggested by Kashima (2004), a lognormal distribution can be used to determine the probability density function of no detection depending on the crack size (see, e.g., Figure 12). Let this probability be defined as $\Pr(No\ detection \mid Crack\ size\ A)$. From the Bayesian conditional probability equation, one can note that

$$\Pr(No\ detection \mid Crack\ size\ A) = \frac{\Pr(No\ detection\ and\ Crack\ Size\ A)}{\Pr(Crack\ Size\ A)}.$$

Here, we need to determine the probability of no-detection in case there is actually a crack of size A, i.e., $\Pr(No\ detection\ and\ Crack\ Size\ A)$. It follows from the above equation that

$$\Pr(No\ detection\ and\ Crack\ Size\ A) = \Pr(No\ detection \mid Crack\ size\ A) \times \Pr(Crack\ Size\ A).$$

From the probability distribution of no detection given the crack size, one already knows $\Pr(No\ detection \mid Crack\ size\ A)$. Therefore, to determine the probability of not detecting a crack when there is a crack of size A, $\Pr(Crack\ Size\ A)$ should be determined. In doing so, let us define the following parameters:

- AT: age of the track when inspected last
- $f(t)$: probability density function of the track's life time

Now, if there is a crack size of A indeed, the lifetime of the track will be its current age plus the time until the current crack develops to a failure. We already know that if the crack size is A, after $\frac{AC-A}{\alpha}$ time units, the track will fail. Therefore, we can say that the track's lifetime will be around $AT + \frac{AC-A}{\alpha}$. From the probability density function of the track's life time, we know that

$$\Pr\left(AT + \frac{AC-A}{\alpha} - e \le LF \le AT + \frac{AC-A}{\alpha} + e\right) = \int_{AT+\frac{AC-A}{\alpha}-e}^{AT+\frac{AC-A}{\alpha}+e} \mathrm{f(t)}\, dt,$$

where e is a very small number. This probability also defines the probability of having a crack of size A at the current inspection (which is not detected). Therefore,

$$\Pr(Crack\ Size\ A) = \int_{AT+\frac{AC-A}{\alpha}-e}^{AT+\frac{AC-A}{\alpha}+e} \mathrm{f(t)}\, dt.$$

It then follows that

$$\Pr(No\ detection\ and\ Crack\ Size\ A) = \Pr(No\ detection \mid Crack\ size\ A) \times \int_{AT+\frac{AC-A}{\alpha}-e}^{AT+\frac{AC-A}{\alpha}+e} \mathrm{f(t)}\, dt.$$

Using $\Pr(No\ detection\ and\ Crack\ Size\ A)$, we can determine the probability of remaining life being less than the time until the next inspection as follows.

Recall that the time until the next inspection is T1. Furthermore, we know that if there is a crack of size A, the remaining life of the track is $T2 = \frac{AC-A}{\alpha}$. In this case, one can note that if T2>T1 with high probability, i.e., it is highly unlikely that the track will fail before its next inspection, no action needed to be taken. On the other hand, if T2<T1 with high probability, i.e., it is highly likely that the track will fail before its next inspection, we recommend taking action for further inspection. To decide on that we need to determine the probability of T2>T1 in case of no detection of a crack. One can note that T2>T1 implies that $\frac{AC-A}{\alpha} > T1$. This further indicates that $A < AC - \alpha T1$. Therefore, one can discuss that

$$\begin{aligned}\Pr(No\ detection\ and\ Remaining\ life\ is\ less\ than\ T1) &= \int_0^{AC-\alpha T1} \Pr(No\ detection\ and\ Crack\ Size\ A)\, dA \\ &= \int_0^{AC-\alpha T1} \int_{AT+\frac{AC-A}{\alpha}-e}^{AT+\frac{AC-A}{\alpha}+e} \Pr(No\ detection \mid Crack\ size\ A)\mathrm{f(t)}\, dt\, dA.\end{aligned}$$

Therefore, we can recommend the following actions:

Given a track is inspected and no crack is detected:

1) Determine the time until the next inspection, denoted by T1
2) Determine the probability of remaining life of the track being less than T1, denoted by Risk
    a. If Risk<E, do not take action
    b. If Risk>E, schedule inspection using the remedial actions given the Track Safety Standards Compliance Manual (FRA, 2002).

Here, E defines the threshold risk value to be specified by the decision maker.

7. CONCLUSIONS

This project developed mathematical modeling and optimization approaches to analyze the track inspection operations on a railroad network and discussed possible procedures that can be used to interpret the inspection results.

In particular, the mathematical programming model formulated, which is referred to as Track Inspection Planning Problem (***TIPP***), determines the best inspection schedule for an inspection vehicle. The model accounted for the following practical settings of track inspection planning operations: inspection times of the tracks, inspection frequencies of the tracks, times between consecutive inspections on the same track, and the inspection importance of the tracks.

Furthermore, as it is crucial to schedule track inspections such that the potential defects are captured as much as possible within minimum times to increase safety to the maximum, two objectives are simultaneously considered in this model. Specifically, minimization of total inspection times and maximization of the total importance of the inspections are simultaneously considered.

The resulting model was a bi-objective non-linear integer-programming problem, which is one of the most difficult optimization problems. A genetic algorithm is developed to determine the track inspection schedules that result in low total inspection times as well as high total inspection importance. The genetic algorithm provides a set of alternative track inspection schedules for the inspection vehicle. These track inspection schedules are not only effective in terms of total inspection times but also total safety importance of the inspections.

In absence of the methods developed in this project, a simple scheduling procedure can be used for determining the track inspection schedules. However, this method would not directly account for the total time and total importance of the inspections. We modified this method considering two approaches: time minimizing and weight maximizing approaches. Using these two approaches, a greedy heuristic algorithm was constructed.

Upon comparing the solution method proposed in this project to the simple greedy heuristic method on a set of railroad track networks of different sizes, the genetic algorithm method proposed proves to find improved inspection schedules regardless of the railroad network size. Therefore, the genetic algorithm can be used to determine effective track inspection planning schedules for the inspection vehicle. Finally, implementation details, a review of the techniques on how to use the inspection results to measure risk of failure, and a method to measure risk on the tracks are provided.

8. REFERENCES

Acharya, D., Mishalani, R., Martland, D. C., & Eshelby, E. J. (1991). Repoman: an Overall Computer-Aided Decision Support System For Planning Rail Replacements. *International Heavy Haul Railway Conference* (pp. 113-123). Vancouver, Canada : Transportation Research Board.

Amaya, A., Langevin, A., & Trépanier, M. (2007). The capacitated arc routing problem with refill points. *Operations Research Letters*, 45-53.

Anderson, R., Dick, C. T., & Barkan, C. P. (2003). Railroad Derailment Factors Affecting Hazardous Materials Transportation Risk. *Transportation Research Board of the National Academies*, *1825*, pp. 64–74. Washington, D.C.

Andersson, M. (2002). *Strategic Planning of Track Maintenance.* Stockholm: Department of Infrastructures, Kungl Tekniska Hogskolan.

Babenko, P. (2006). *Visual Inspection of Railroad Tracks.* Orlando.

Birmingham, U. o. (2008). *Sustainable Development, Global Change and Ecosystems; Rail Inspection Technologies.*

Budai-Balke, G. (2009). *Operations Research Models for Scheduling Railway Infrastructure Maintenance.* Rotterdam: Erasmus University Rotterdam.

Cerniglia, D., Garcia, G., Kalay, S., & Prior, F. (2006). *Application of Laser Induced Ultrasound for Rail Inspection.* Railway Research Center.

Dell'Orco, M., Ottomanelli, M., Pace, P., & Pascoschi, G. (2001). Intelligent Decision Support Tools for Optimal Planning of Rail Track Maintenance. *Euro Working Group Transportation*, 218-223.

Eriksen, A., Gascoyne, J., & Al-Nuaimy, W. (2004). Improved Productivity & Reliability of Ballast Inspection using Road-Rail Multi-Channel GPR. *Preceedings of Railway Engineering.* London, UK: IEEE.

Esveld, C. (1990). Computer-aided maintenance and renewal of track. *Railroad Conference* (pp. 165-170). Chicago, IL: IEEE.

Gantrex. (2000). *Crane Rail Inspection.* Gantrex.

Gordon, C., Akinci, B., & Garrett, J. H. (2007). Formalism for Construction Inspection Planning: Requirements and Process Concept. *Journal of Computing in Civil Engineering*, 29-38.

Hall, R. W. (2000). Scheduling and facility design for transit railcar maintenance. *Transportation Research Part A: Policy and Practice*, 67–84.

Hayashi, T., Miyazaki, Y., Murase, M., & Abe, T. (2007). Guided Wave Inspection for Bottom Edge of Rails. *AIP Conf. Proc.* Portland, Oregon: American Institute of Physics.

Hesse, D. (2007). *Rail Inspection Using Ultrasonic Surface Wave.* London.

Higgins, H. (1998). Scheduling of railway track-maintenance activities and crews. *Journal of the Operational Society*, 1026-1033.

Hugenschmid, J. (1999). Railway track inspection using GPR. *Journal of Applied Geophysics*, 147-155.

Kashima, T. (2004). *Reliability-Based Optimization of Rail Inspection.* Cambridge, Massachusetts: Massachusetts Institute of Technology.

Kenderian, S., Djordjevic, B. B., Cerniglia, D., & Garcia, G. (2006). Dynamic railroad inspection using the laser-air hybrid ultrasonic technique. *Insight - Non-Destructive Testing and Condition Monitoring*, 336-341.

Kim, S., & Frangopol, D. M. (2011). Cost-Based Optimum Scheduling of Inspection and Monitoring for Fatigue-Sensitive Structures under Uncertainty. *Journal of Structural Engineering*, 1319-1331.

Lannez, S., Artigues, C., Damay, J., & Gendreau, M. (2010). A railroad maintenance problem solved with a cut and column generation matheuristic. *"Triennial Symposium on Transportation Analysis (TRISTAN VII)*.

Lee, W., Lee, J., Henderson, C., Taylor, H. F., James, R., Lee, C. E., et al. (1999, March 1). Railroad bridge instrumentation with fiber-optic sensors. *Applied Optics, 38*(7), 1110-1114.

Merrick, J. R., Soyer, R., & Mazzuchi, T. A. (2005). Are Maintenance Practices for Railroad Tracks Effective? *Journal of the American Statistical Association, 100*(465), 17-25.

NDT, C. (2013, 10 13). *Rail Inspection*. Retrieved from NDT Resource Center: http://www.ndt-ed.org/AboutNDT/SelectedApplications/RailInspection/RailInspection.htm

NDT, H. (n.d.). *A document outlining the emergence of the eddy current NDT inspection method as an important part of rail maintenance and safety.*

Office of Railroad Safety, F. R. (2011, August). Track Inspector Rail Defect Reference Manual. United States. Retrieved October 20, 2014, from http://www.fra.dot.gov/elib/document/2130/

Office of Safety Assurance and Compliance, F. R. (2002, January 1). Track Safety Standards Compliance Manual. Retrieved October 24, 2014, from https://www.hsdl.org/?view&did=15770

Peng, F., & Ouyang, Y. (2012). Track maintenance production team scheduling in railroad networks. *Transportation Research Part B*, 1474–1488.

Peterson, B. O. (2012). Leveraging Technology to Facilitate Predictive Maintenance Planning. *AREMA.* Chicago, IL: AREMA (American Railway Engineering and Maintenance-of-way Association).

Peterz, N. (2004). *Fracture Mechanics.* New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers.

Podofillinia, L., Zioa, E., & Vatn, J. (2005). Risk-informed optimisation of railway tracks inspection and maintenance procedures. *Reliability Engineering and System Safety*, 20-35.

Prescott, D., & Andrews, J. (2013). Modelling Maintenance in Railway Infrastructure Management. *Reliability and Maintainability Symposium.* Orlando, FL: IEEE.

Pugno, N., Ciavarella, M., Cornetti, P., & Carpinter, A. (2006). A generalized Paris' law for fatigue crack growth. *Journal of the Mechanics and Physics of Solids, 54*, 1333-1349.

Rocha, J. M., Henriques, A. A., & Calçada, R. (2014). Probabilistic safety assessment of a short span high-speed railway bridge. *Engineering Structures, 71*, 99-111.

Shiau, Y.-R., Lin, M.-H., & Chuang, W.-C. (2007, May 23). Concurrent process/inspection planning for a customized manufacturing system based on genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 746-755.

Shyr, F., & Ben-Akiva, M. (1996). Modeling Rail Fatigue Behavior with Multiple Hazard. *Journal of Infrastructural Engineering*,(2), 73-82.

Transportation, C. B. (2012). *Connecticut Railroad Bridge Management Program.*

Transportation, D. o. (2014, January 24). Track Safety Standards, Improving Rail Integrity; Final Rule. Retrieved October 24, 2014, from http://www.fra.dot.gov/eLib/Document/3546

Transportation, O. D. (2012, 02 26). Track Inspector Rail Defect Reference Manual. Ohio, United States.

Uzarski, D. R., Brown, D. G., Harris, R. W., & Plotkin, D. E. (1993). *Maintenance Management of U.S. Army Railroad Networks-the RAILER System: Detailed Track Inspection Manual.* Champaign, IL: US Army Corps of Engineers; Construction Engineering Research Laboratories.

Zarembski, M. A., & Palese, J. (2005). Characterization of Broken Rail Risk for Freight and Passenger Railway Operations., (pp. 25-28). Chicago, IL.

Zhao, J., Chan, A. H., & Burrow, M. P. (2007). Probabilistic Model for Predicting Rail Breaks and Controlling Risk of Derailment. *Transportation Research Record, 1995*, 76-83.

Zhao, J., Chan, A. H., Roberts, C., & Madelin, K. B. (2007). Reliability evaluation and optimization of imperfect inspections for a component with multi-defects. *Reliability Engineering and System Safety, 92*, 65-73.

Zhao, Z., Haldar, A., & Breen, F. J. (1994). Fatigue-Reliability Updating through Inspections of Steel Bridges. *Journal of Structural Engineering, 120*(5), 1624–1642.

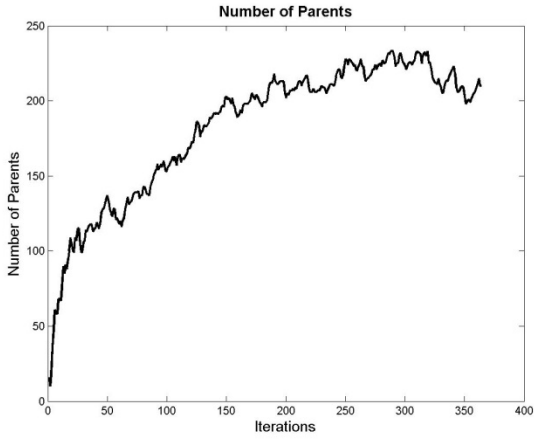
Zheng, R., & Ellingwood, B. (1998). Role of non-destructive evaluation in time-dependent reliability analysis. *Structural Safety, 20*(4), 325-339.

9. APPENDIX

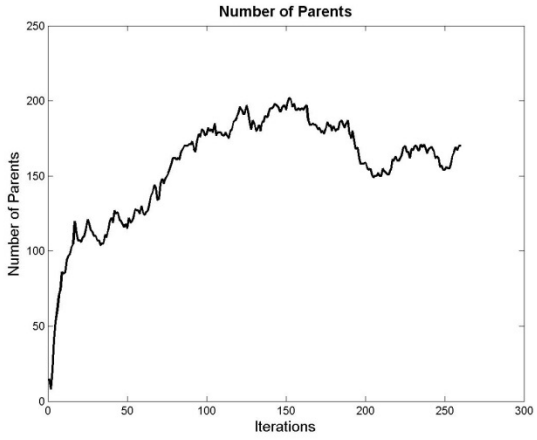
Appendix A: Parent Size of the Iterations of the Genetic Algorithm

Figure 4: Number of Parent Chromosomes vs. Iterations

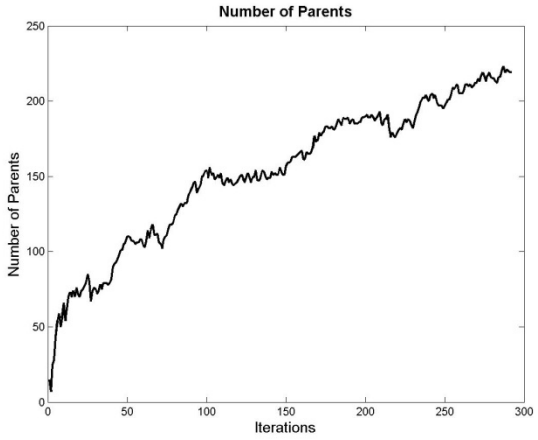
Network Size = 100, Problem Instance = 1



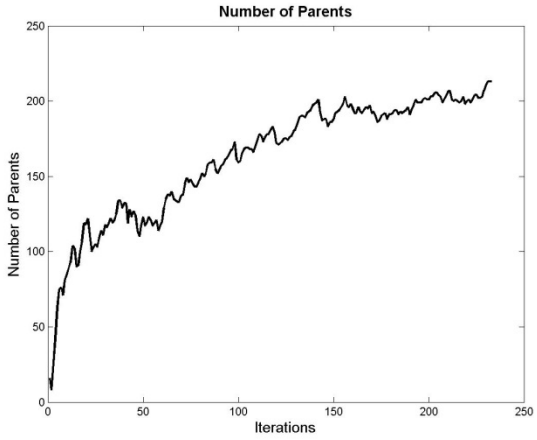
Network Size = 100, Problem Instance = 2



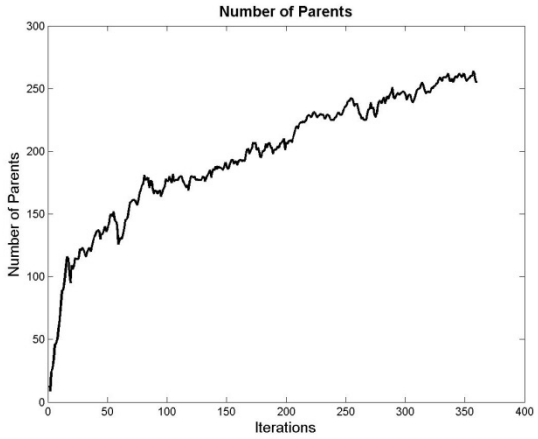
Network Size = 100, Problem Instance = 3



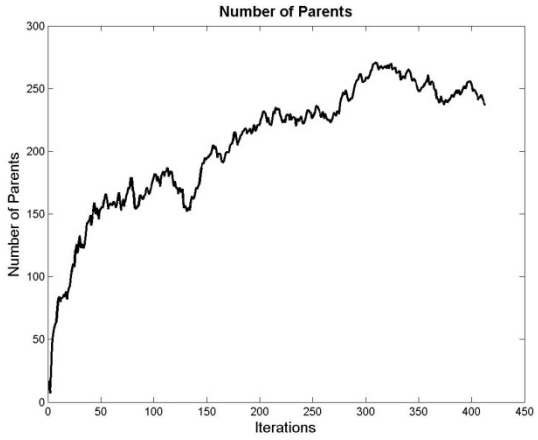
Network Size = 100, Problem Instance = 4



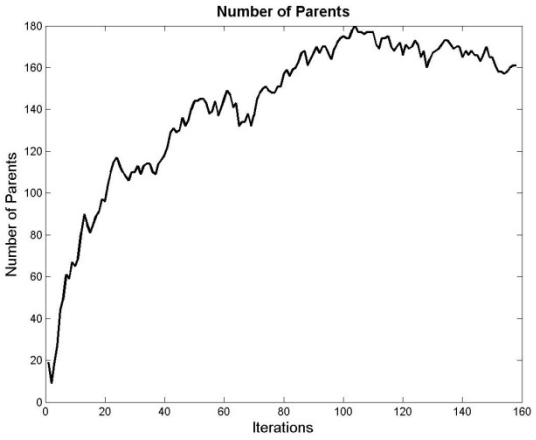
Network Size = 100, Problem Instance = 5



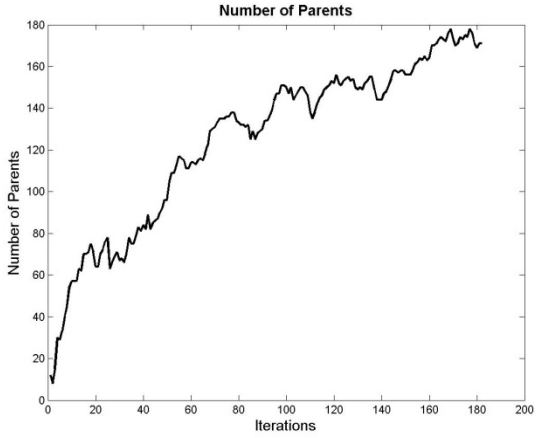
Network Size = 100, Problem Instance = 6



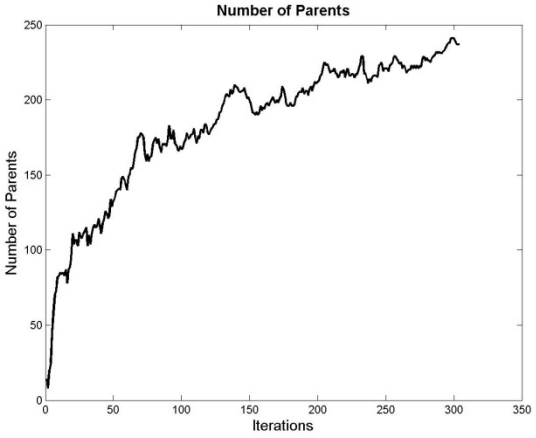
Network Size = 100, Problem Instance = 7



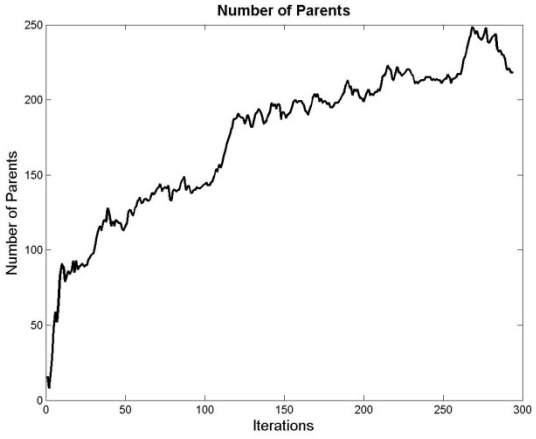
Network Size = 100, Problem Instance = 8



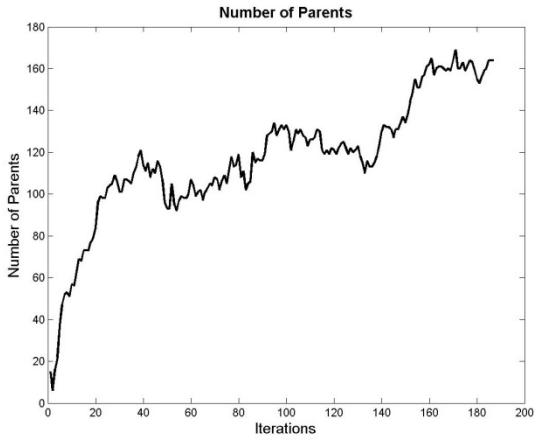
Network Size = 100, Problem Instance = 9



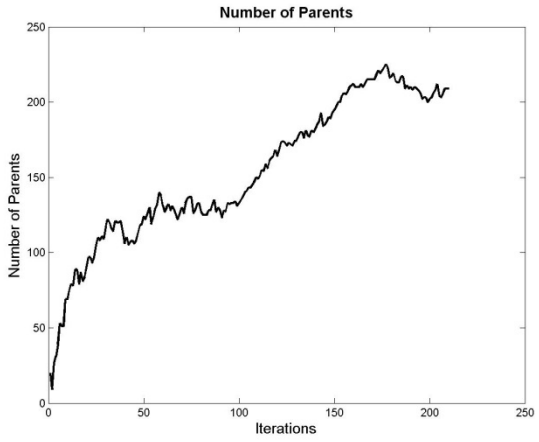
Network Size = 100, Problem Instance = 10



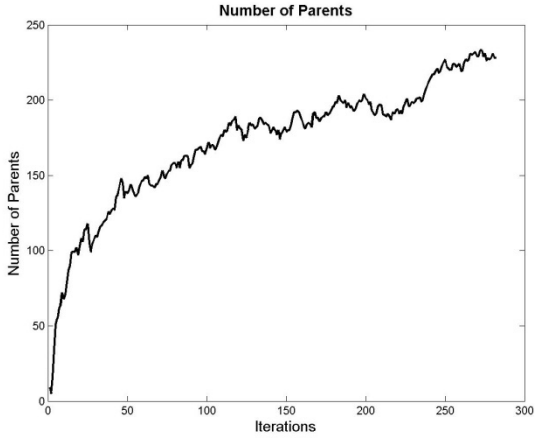
Network Size = 150, Problem Instance = 1



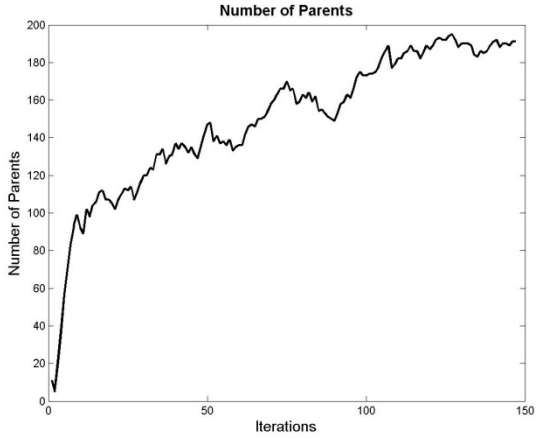
Network Size = 150, Problem Instance = 2



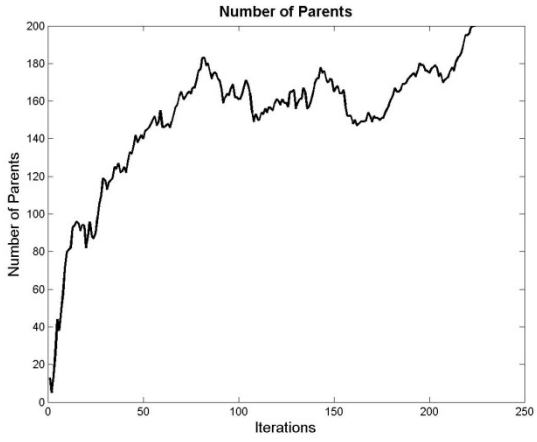
Network Size = 150, Problem Instance = 3



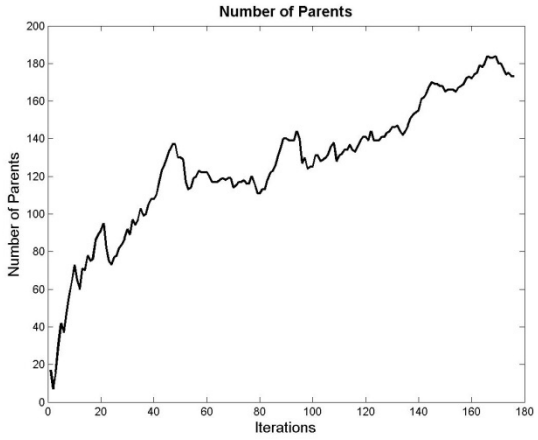
Network Size = 150, Problem Instance = 4



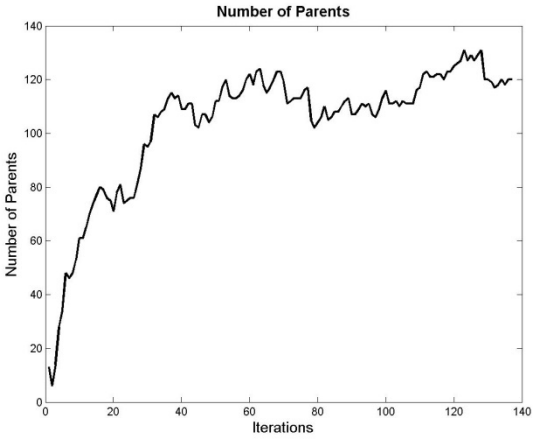
Network Size = 150, Problem Instance = 5



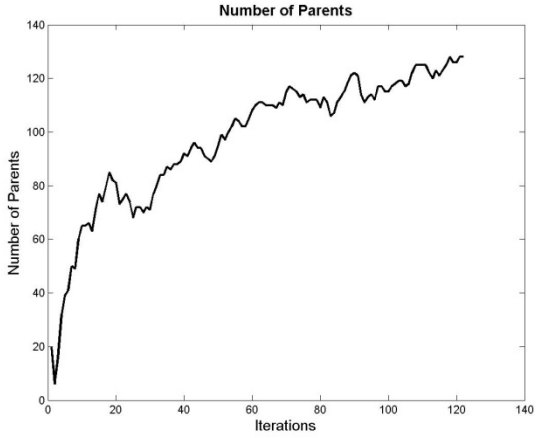
Network Size = 150, Problem Instance = 6



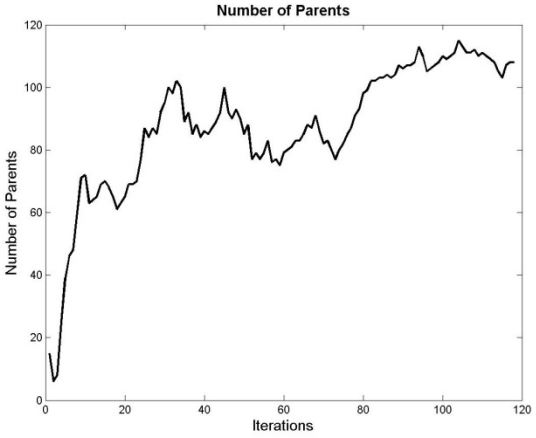
Network Size = 150, Problem Instance = 7



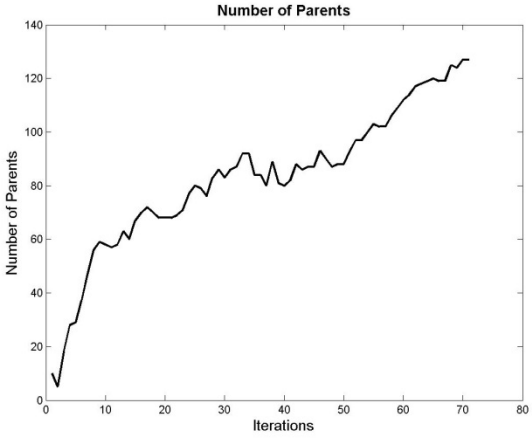
Network Size = 150, Problem Instance = 8



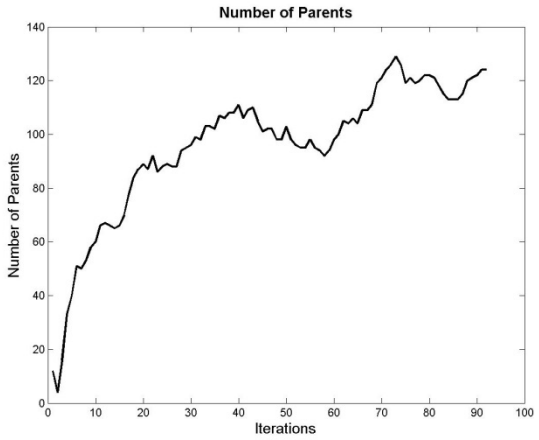
Network Size = 150, Problem Instance = 9



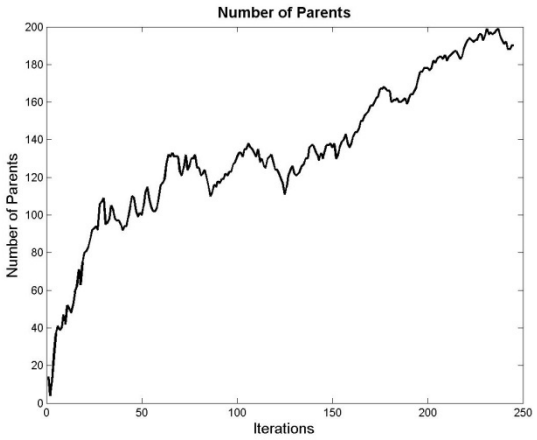
Network Size = 150, Problem Instance = 10



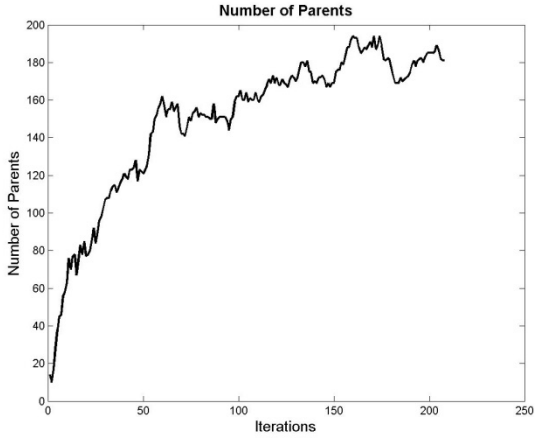
Network Size = 200, Problem Instance = 1



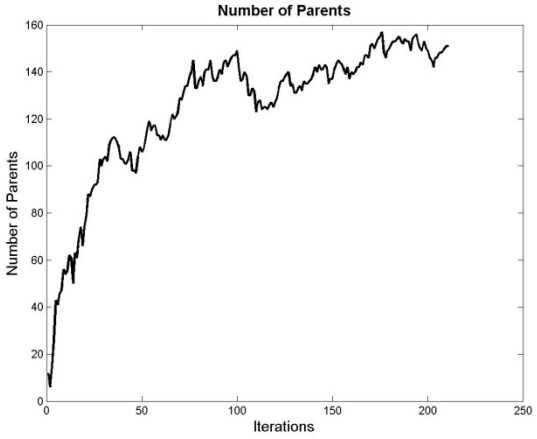
Network Size = 200, Problem Instance = 2



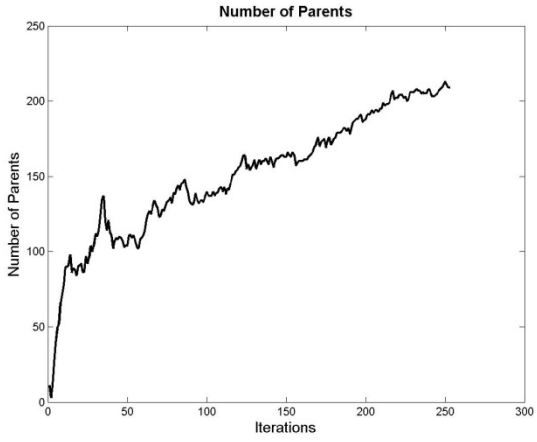
Network Size = 200, Problem Instance = 3



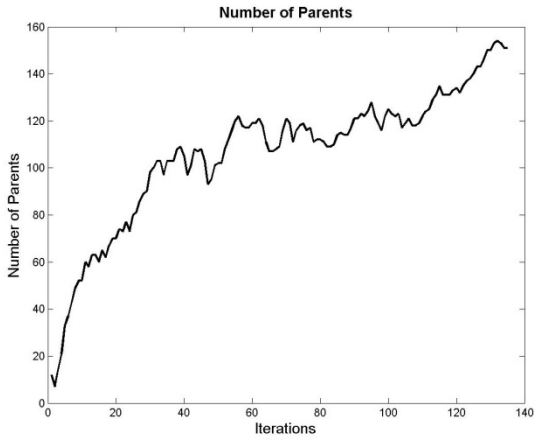
Network Size = 200, Problem Instance = 4



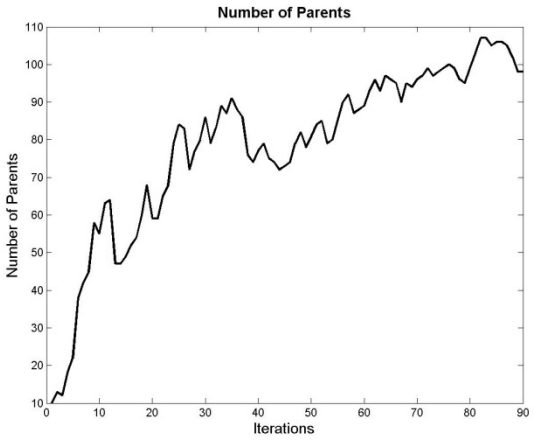
Network Size = 200, Problem Instance = 5



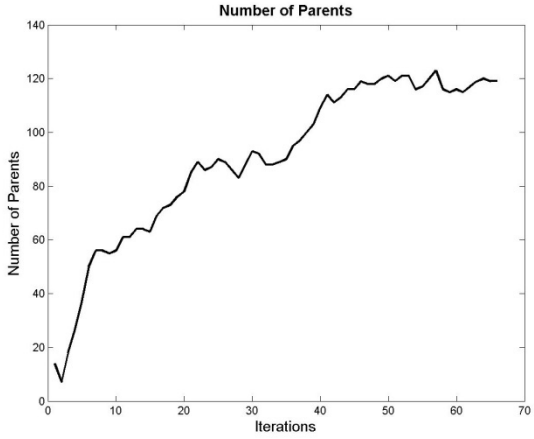
Network Size = 200, Problem Instance = 6



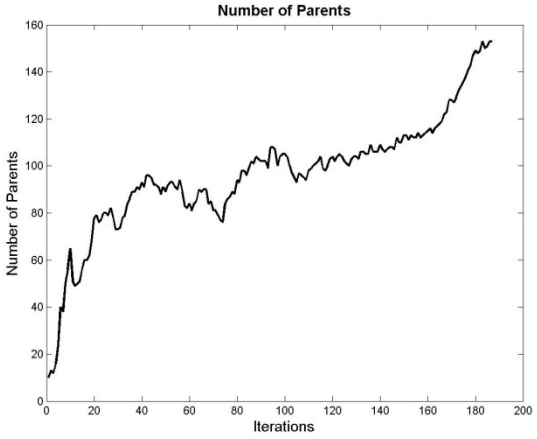
Network Size = 200, Problem Instance = 7



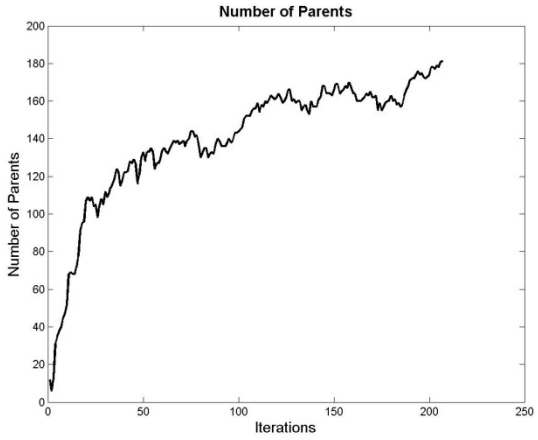
Network Size = 200, Problem Instance = 8



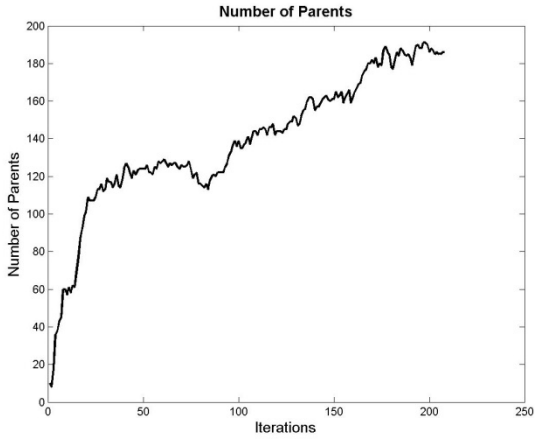
Network Size = 200, Problem Instance = 9



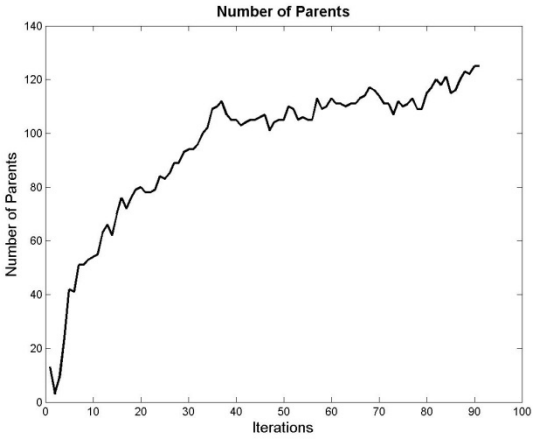
Network Size = 200, Problem Instance = 10



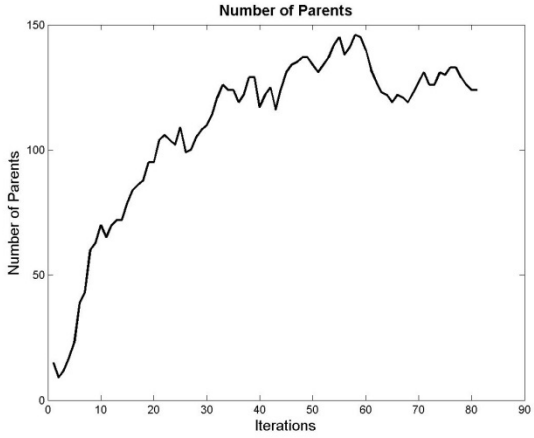
Network Size = 250, Problem Instance = 1



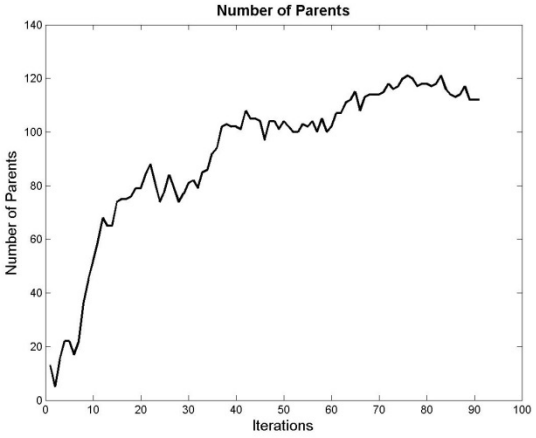
Network Size = 250, Problem Instance = 2



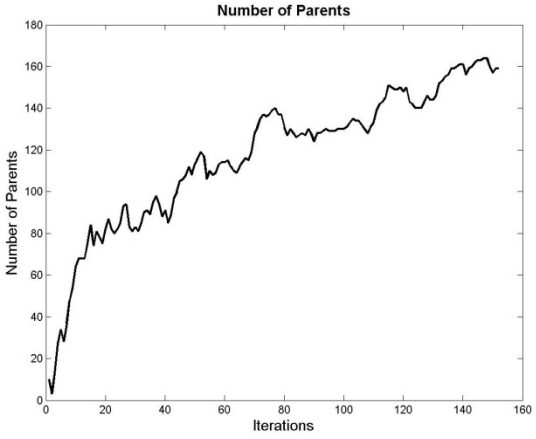
Network Size = 250, Problem Instance = 3



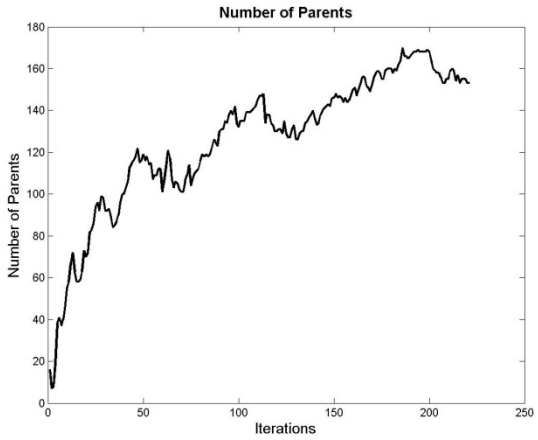
Network Size = 250, Problem Instance = 4



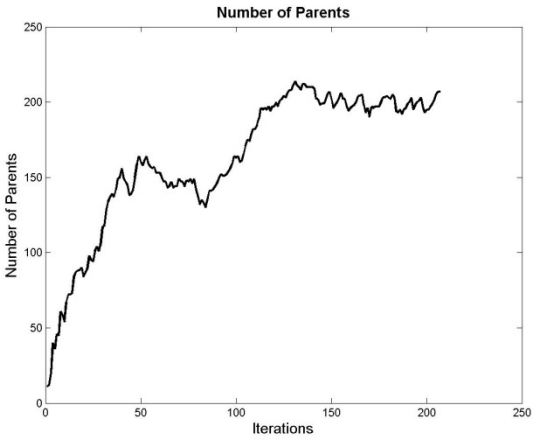
Network Size = 250, Problem Instance = 5



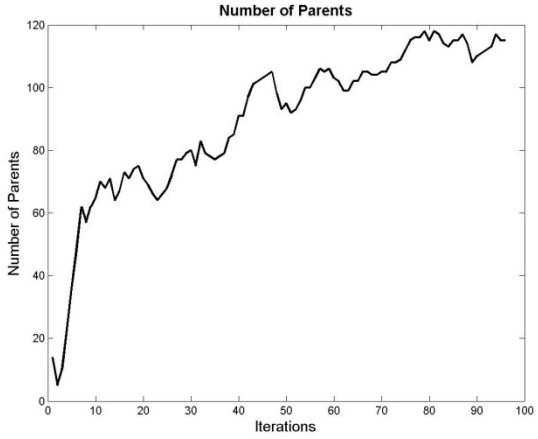
Network Size = 250, Problem Instance = 6



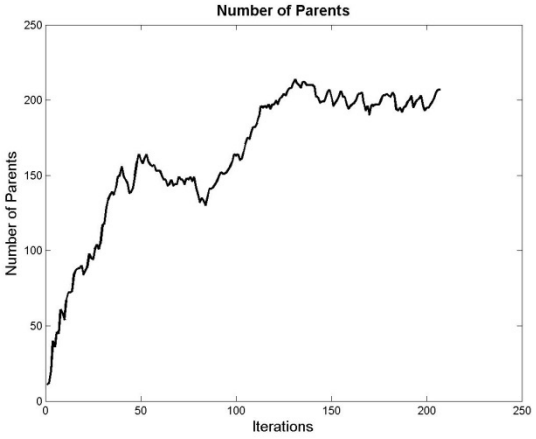
Network Size = 250, Problem Instance = 7



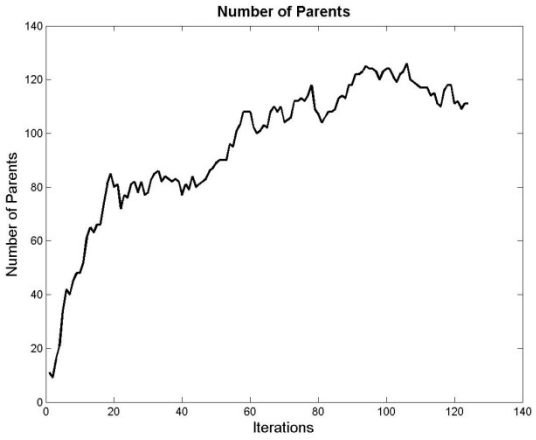
Network Size = 250, Problem Instance = 8



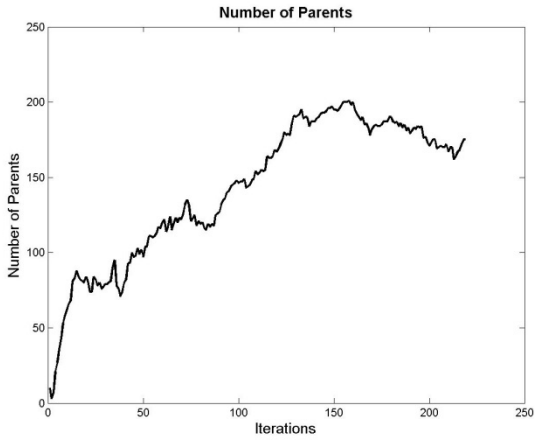
Network Size = 250, Problem Instance = 9



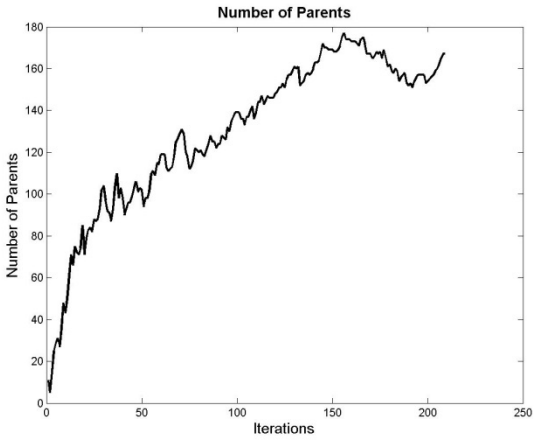
Network Size = 250, Problem Instance = 10



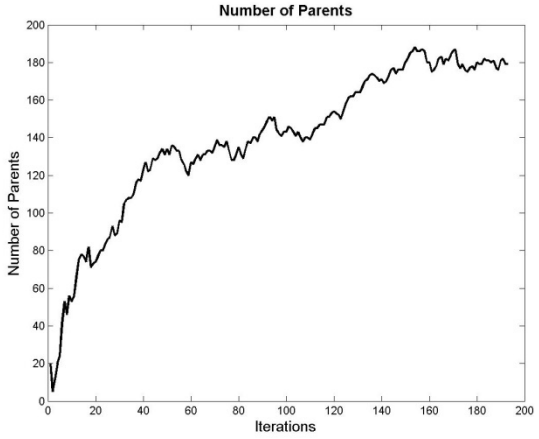
Network Size = 300, Problem Instance = 1



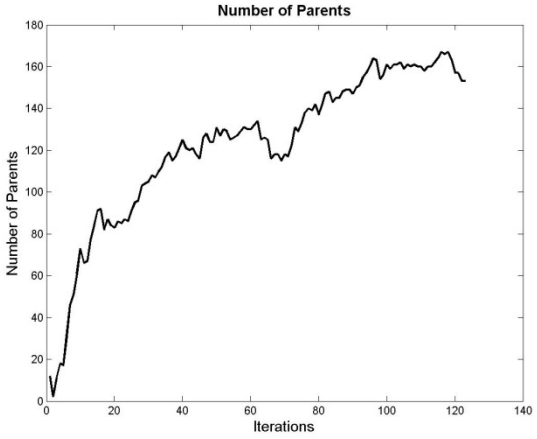
Network Size = 300, Problem Instance = 2



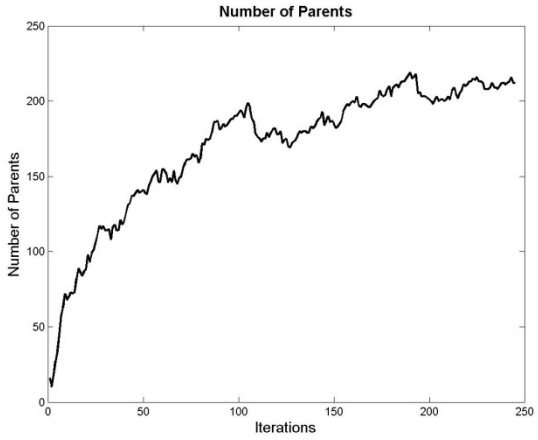
Network Size = 300, Problem Instance = 3



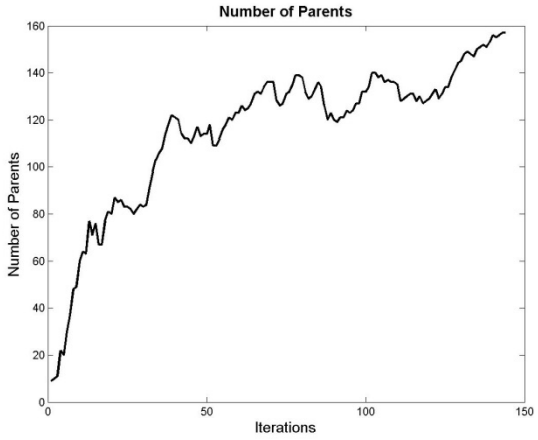
Network Size = 300, Problem Instance = 4



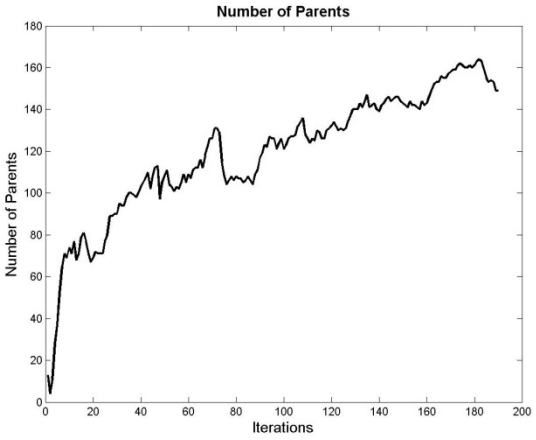
Network Size = 300, Problem Instance = 5



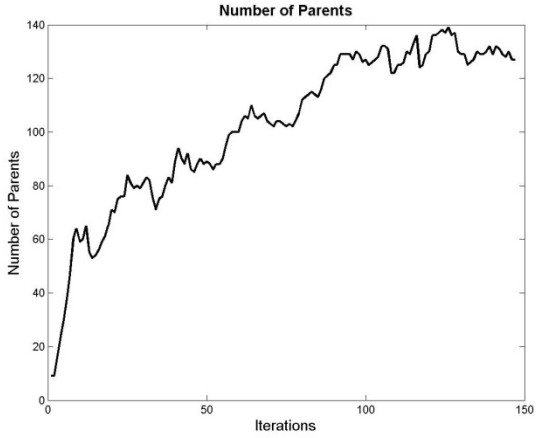
Network Size = 300, Problem Instance = 6



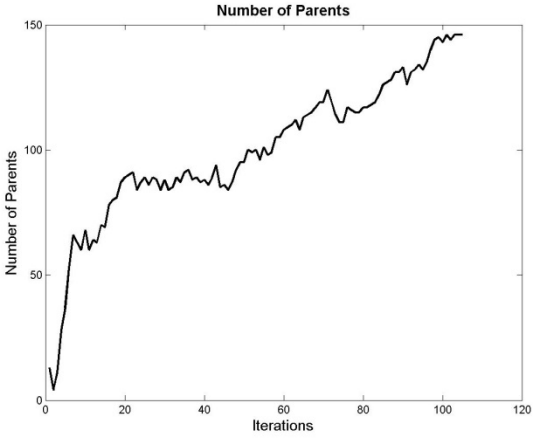
Network Size = 300, Problem Instance = 7



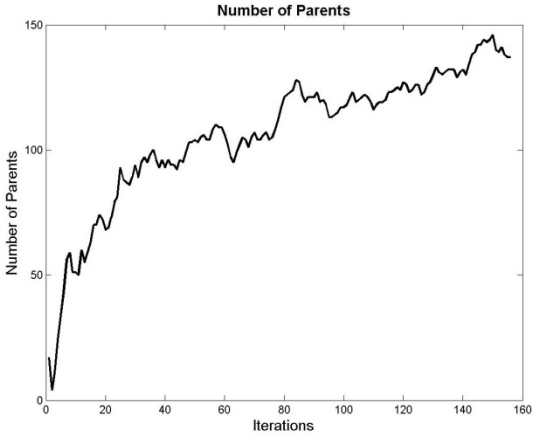
Network Size = 300, Problem Instance = 8



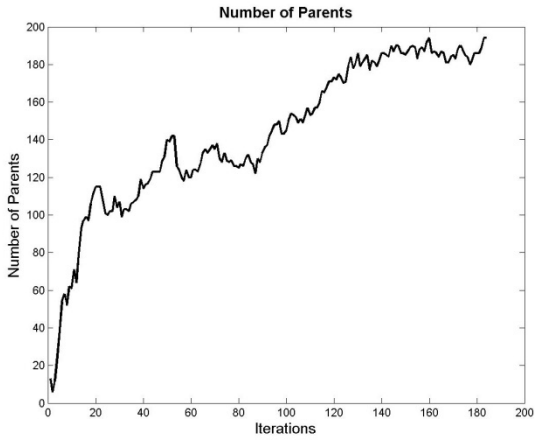
Network Size = 300, Problem Instance = 9



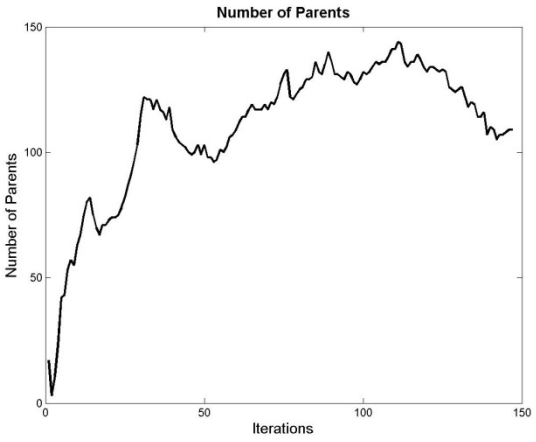
Network Size = 300, Problem Instance = 10



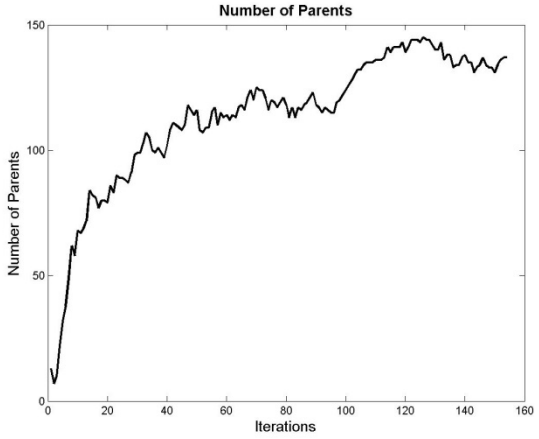
Network Size = 350, Problem Instance = 1



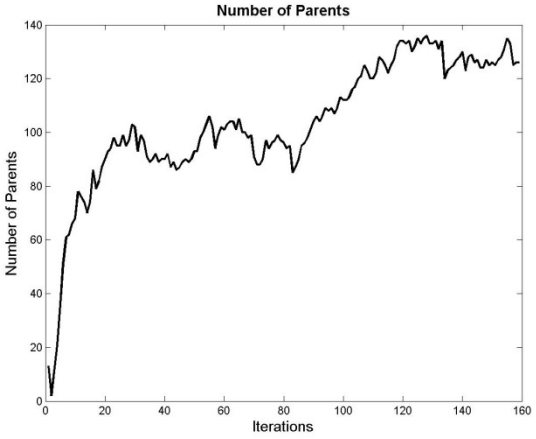
Network Size = 350, Problem Instance = 2



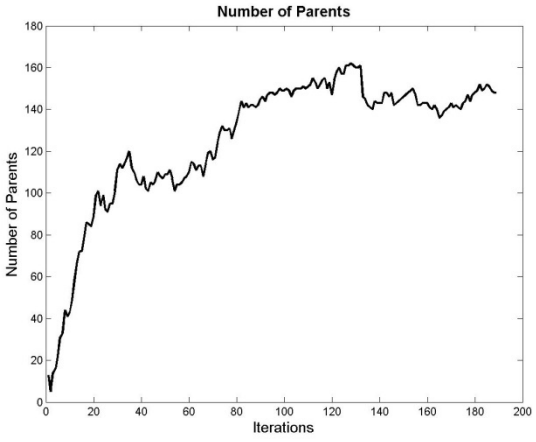
Network Size = 350, Problem Instance = 3



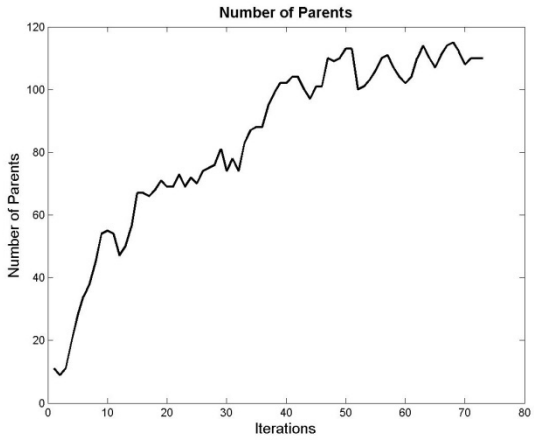
Network Size = 350, Problem Instance = 4



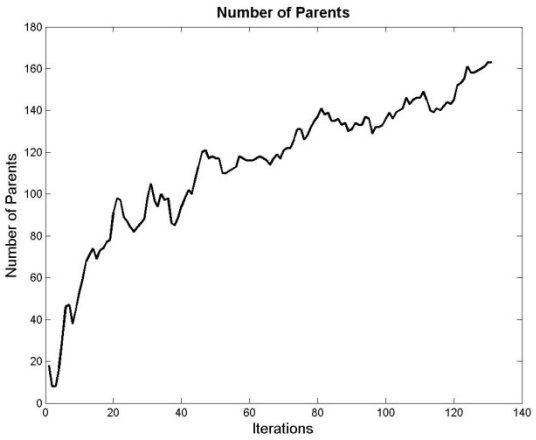
Network Size = 350, Problem Instance = 5



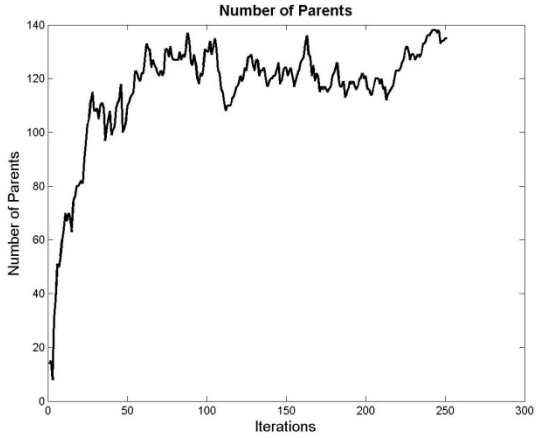
Network Size = 350, Problem Instance = 6



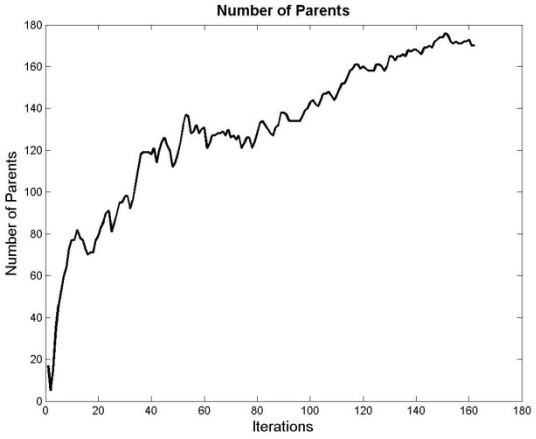
Network Size = 350, Problem Instance = 7



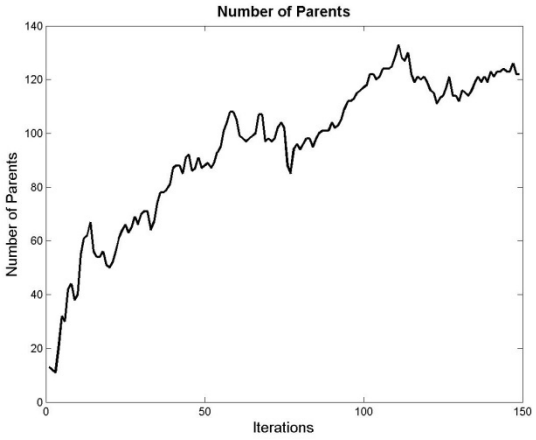
Network Size = 350, Problem Instance = 8



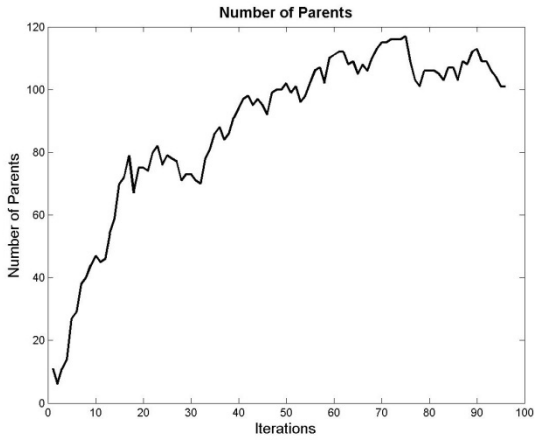
Network Size = 350, Problem Instance = 9



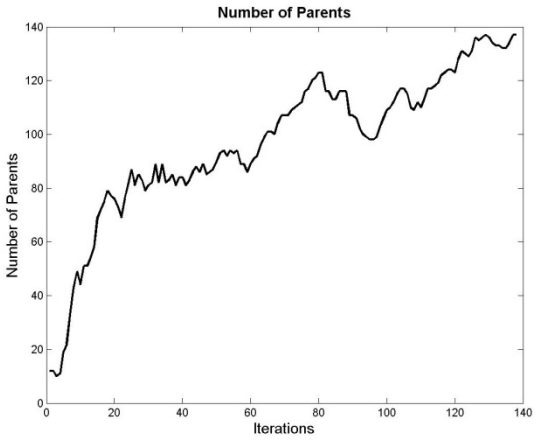
Network Size = 350, Problem Instance = 10



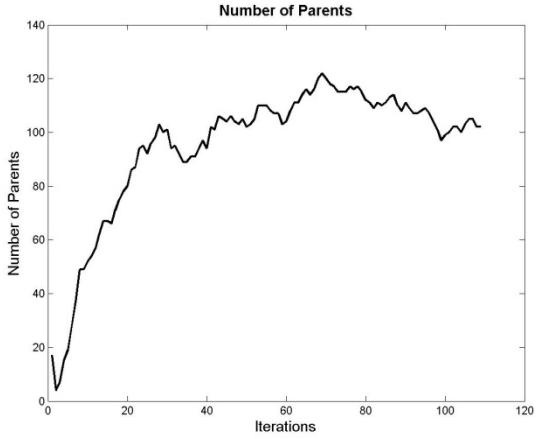
Network Size = 400, Problem Instance = 1



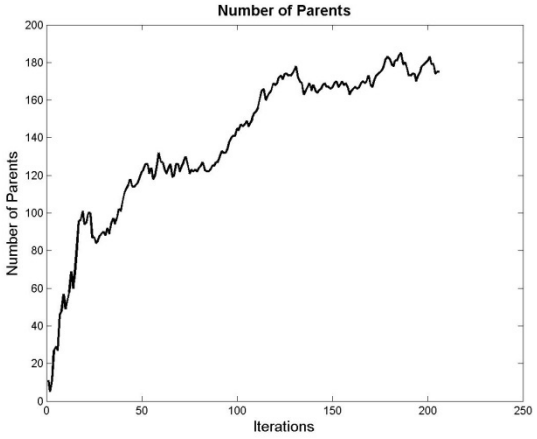
Network Size = 400, Problem Instance = 2



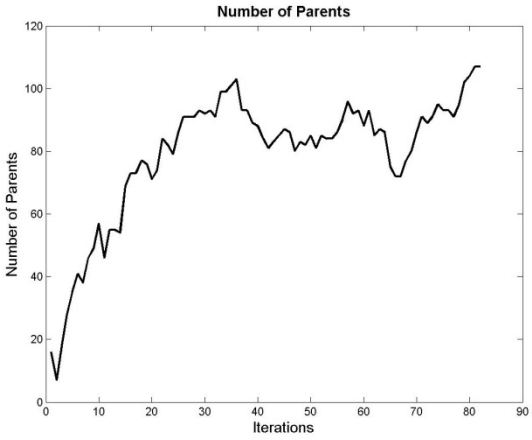
Network Size = 400, Problem Instance = 3



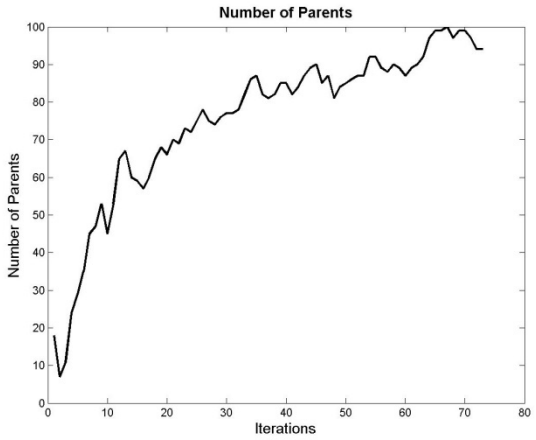
Network Size = 400, Problem Instance = 4



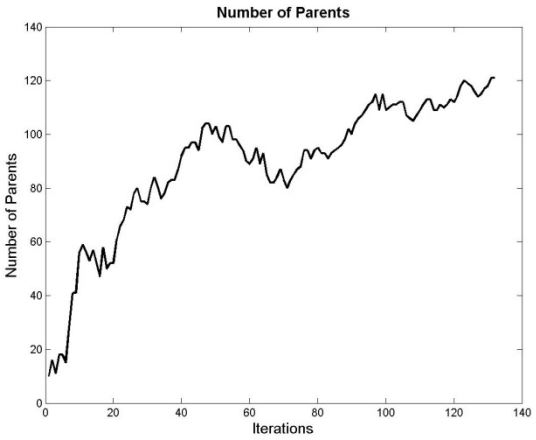
Network Size = 400, Problem Instance = 5



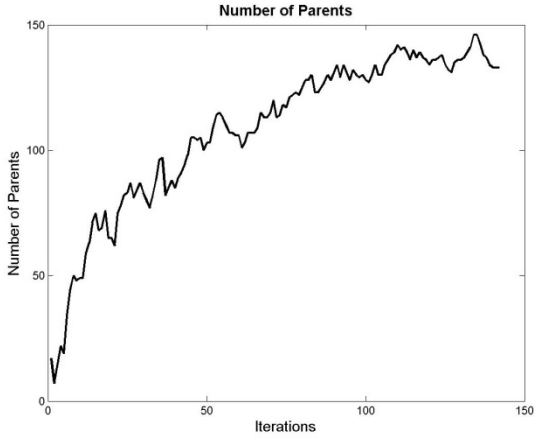
Network Size = 400, Problem Instance = 6



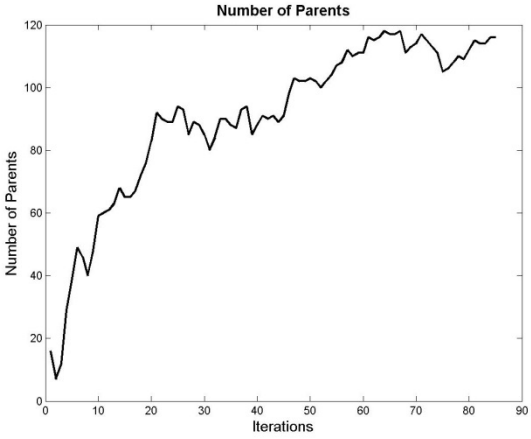
Network Size = 400, Problem Instance = 7



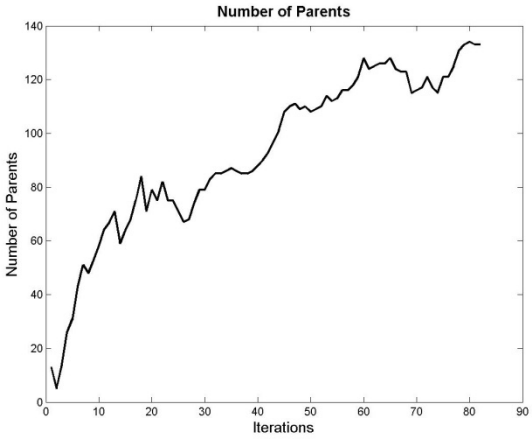
Network Size = 400, Problem Instance = 8



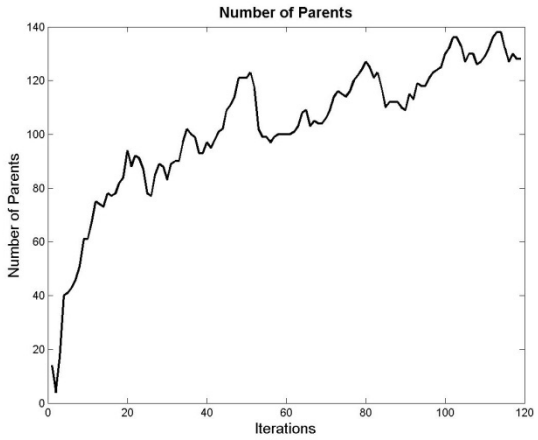
Network Size = 400, Problem Instance = 9



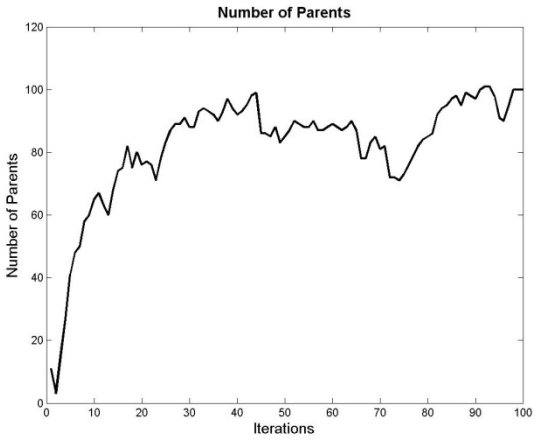
Network Size = 400, Problem Instance = 10



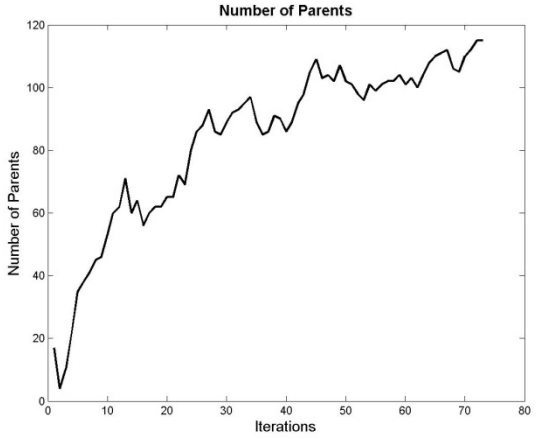
Network Size = 450, Problem Instance = 1



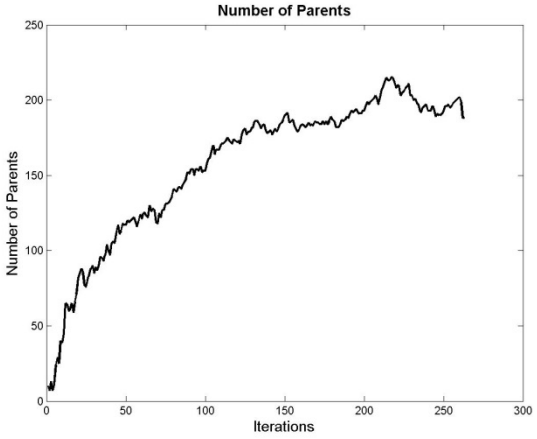
Network Size = 450, Problem Instance = 2



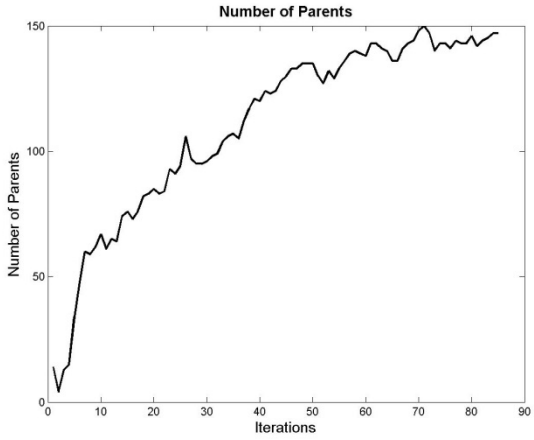
Network Size = 450, Problem Instance = 3



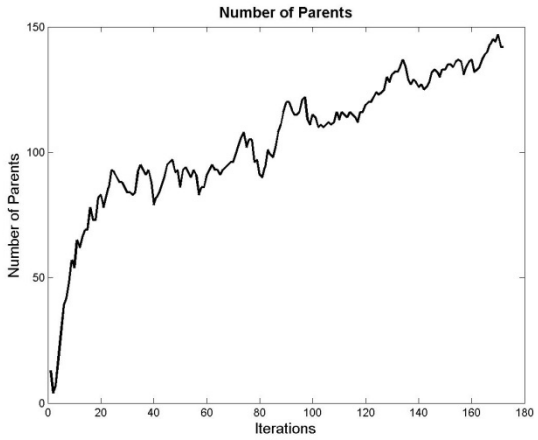
Network Size = 450, Problem Instance = 4



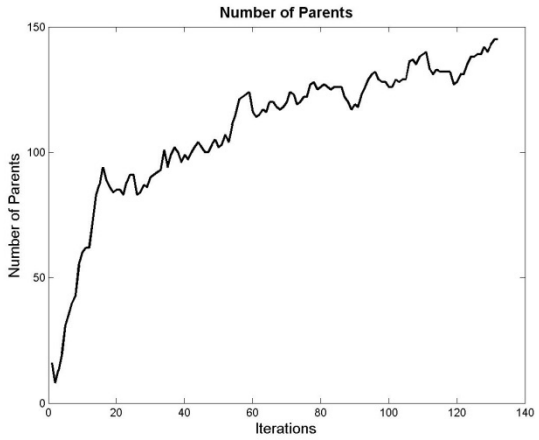
Network Size = 450, Problem Instance = 5



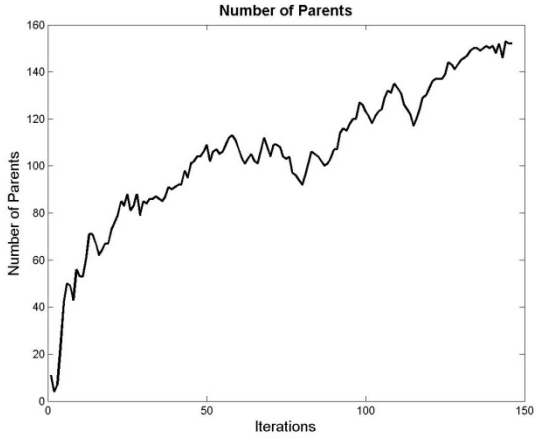
Network Size = 450, Problem Instance = 6



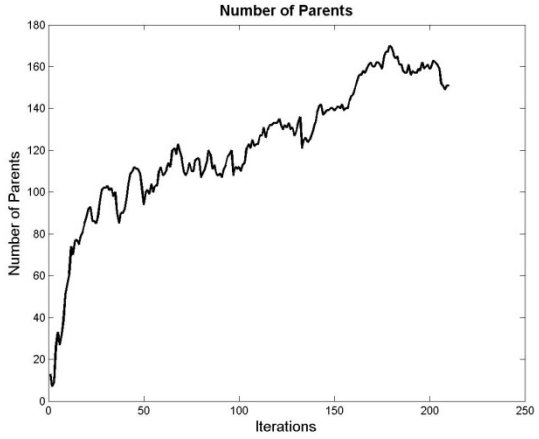
Network Size = 450, Problem Instance = 7



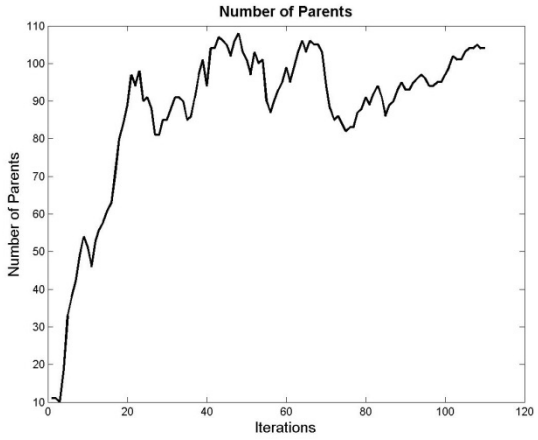
Network Size = 450, Problem Instance = 8



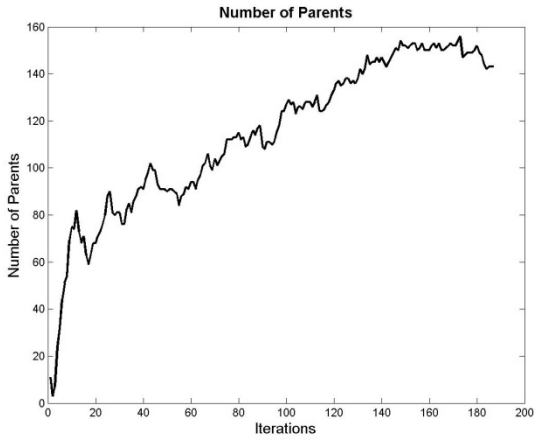
Network Size = 450, Problem Instance = 9



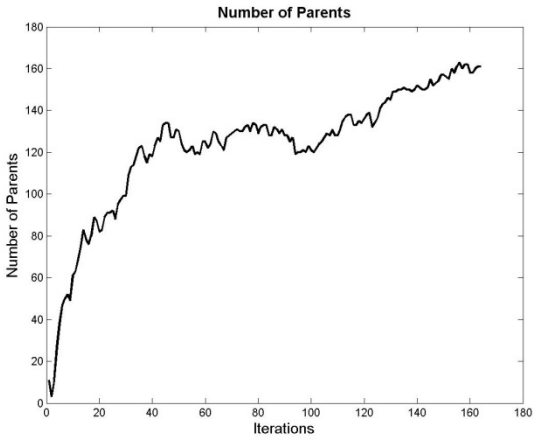
Network Size = 450, Problem Instance = 10



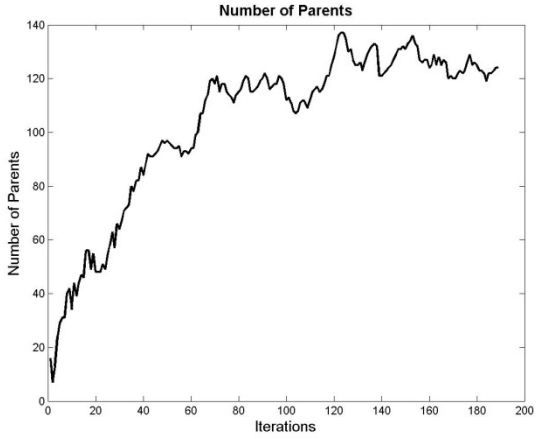
Network Size = 500, Problem Instance = 1



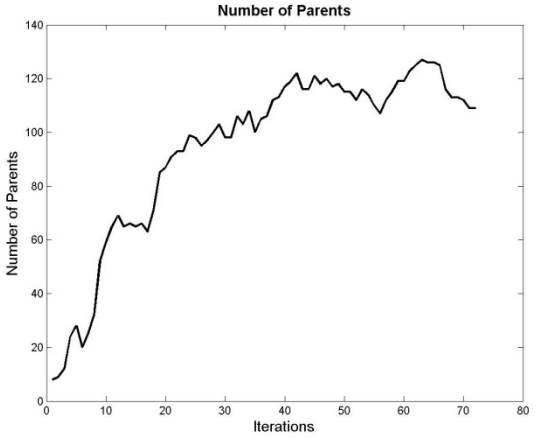
Network Size = 500, Problem Instance = 2



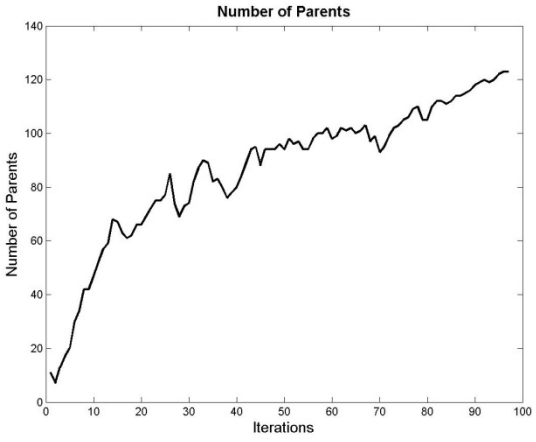
Network Size = 500, Problem Instance = 3



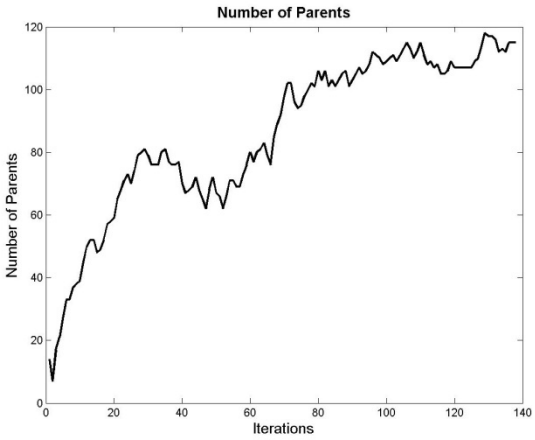
Network Size = 500, Problem Instance = 4



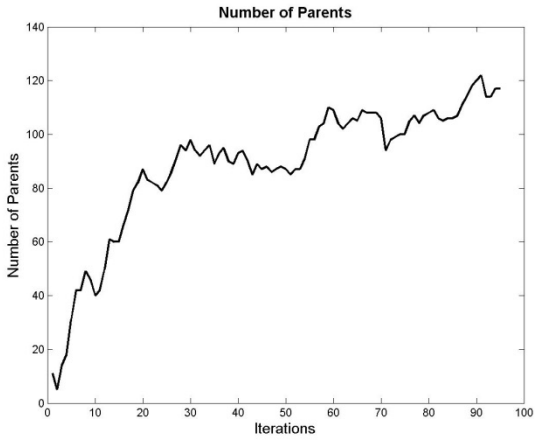
Network Size = 500, Problem Instance = 5



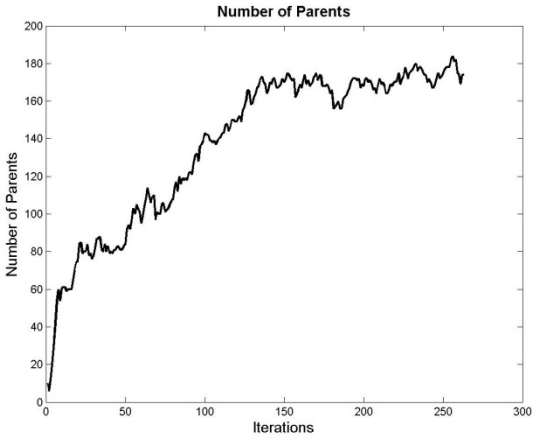
Network Size = 500, Problem Instance = 6



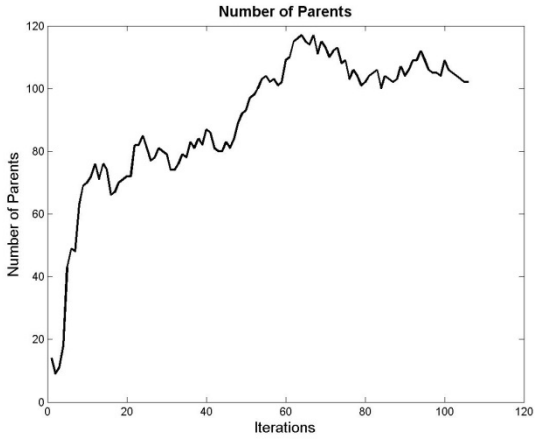
Network Size = 500, Problem Instance = 7



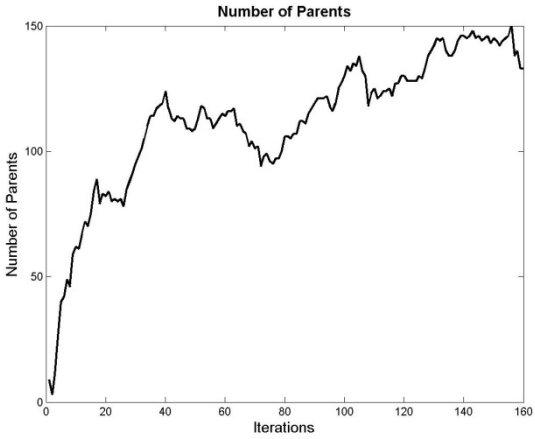
Network Size = 500, Problem Instance = 8



Network Size = 500, Problem Instance = 9



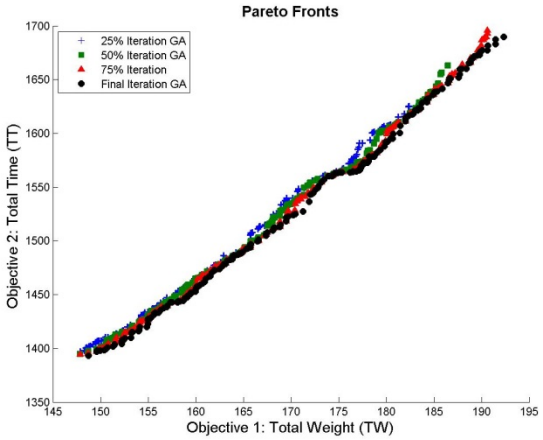
Network Size = 500, Problem Instance = 10



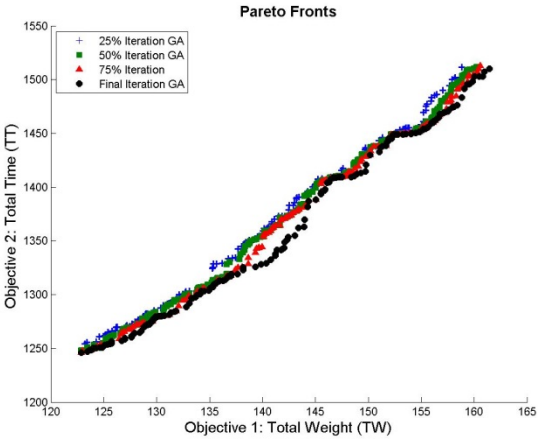
Appendix B: Improvements of the Pareto Fronts of the Genetic Algorithm

Figure 5: Parent Chromosomes over Iterations

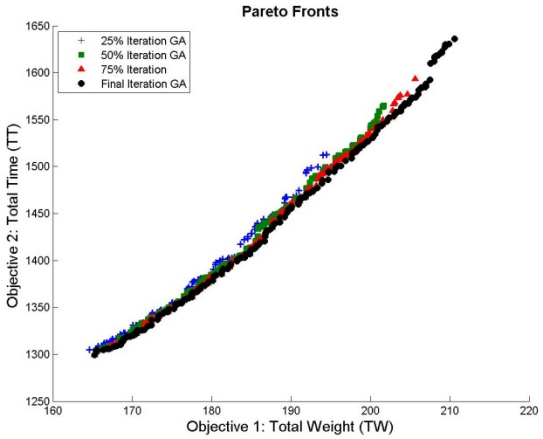
Network Size = 100, Problem Instance = 1



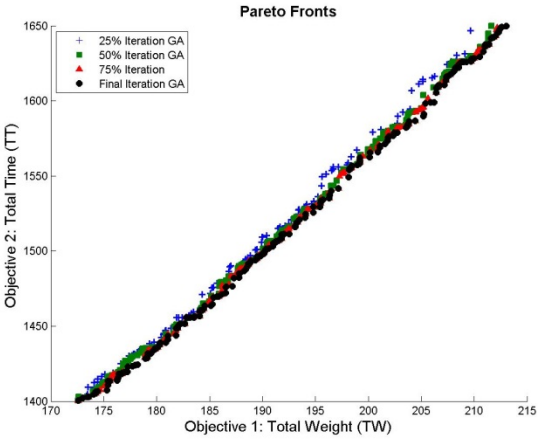
Network Size = 100, Problem Instance = 2



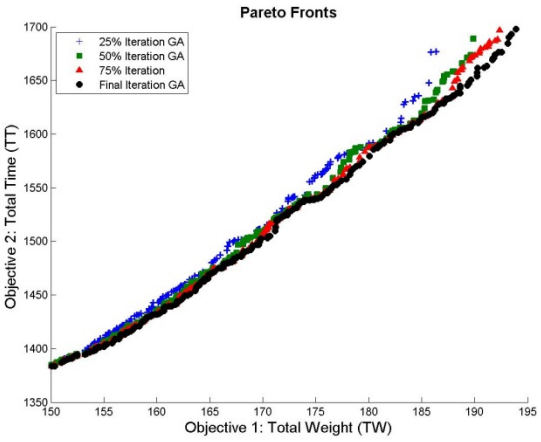
Network Size = 100, Problem Instance = 3



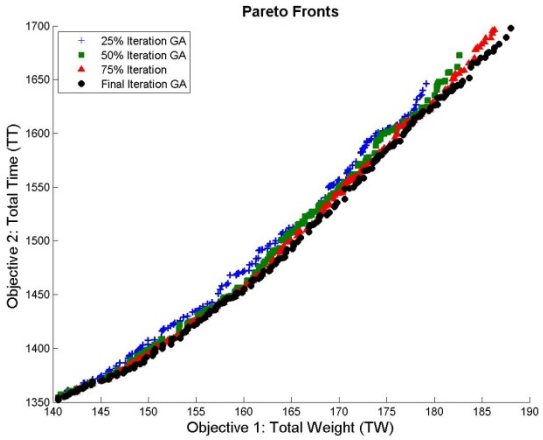
Network Size = 100, Problem Instance = 4



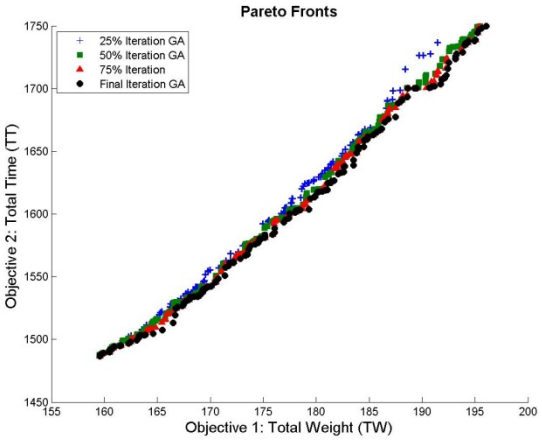
Network Size = 100, Problem Instance = 5



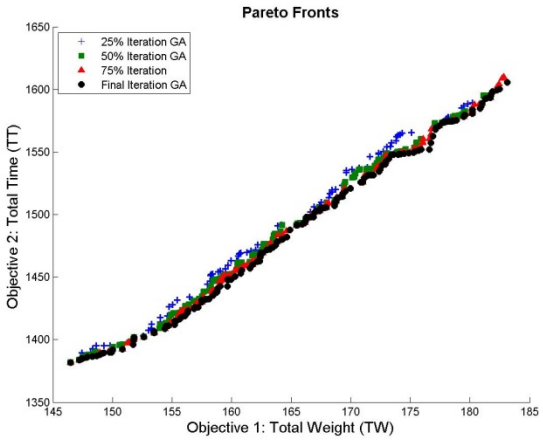
Network Size = 100, Problem Instance = 6



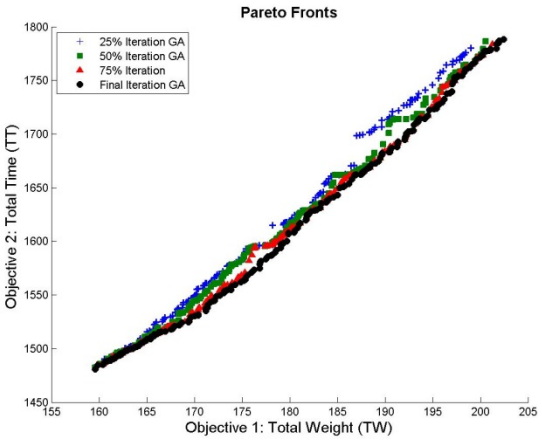
Network Size = 100, Problem Instance = 7



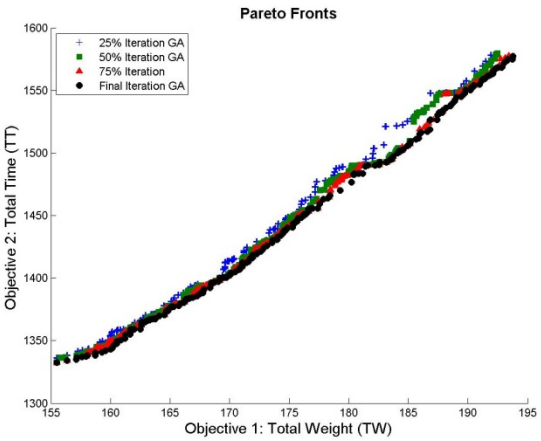
Network Size = 100, Problem Instance = 8



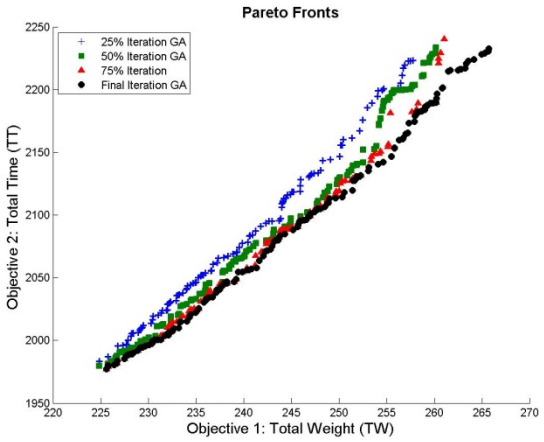
Network Size = 100, Problem Instance = 9



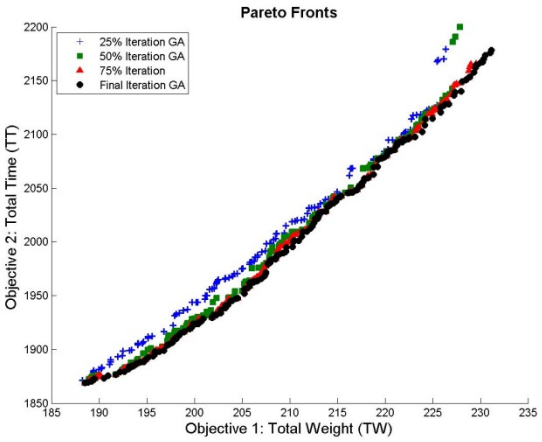
Network Size = 100, Problem Instance = 10



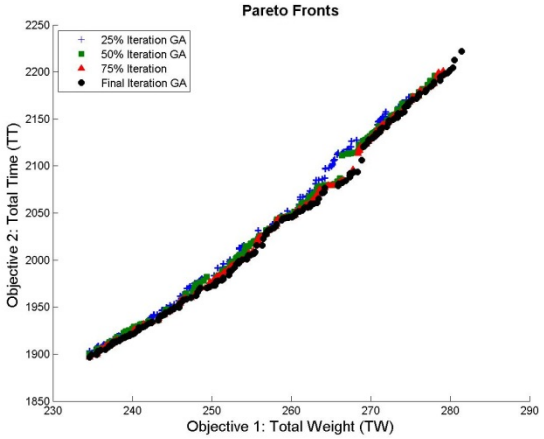
Network Size = 150, Problem Instance = 1



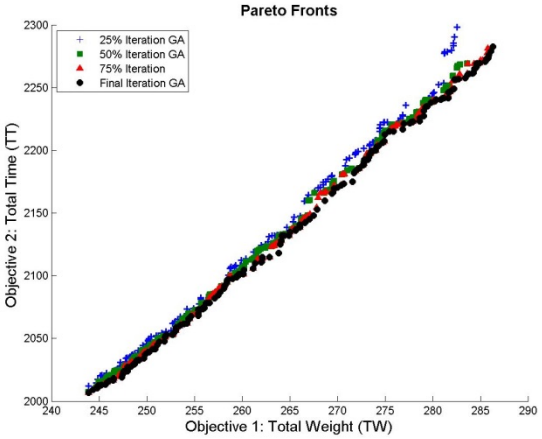
Network Size = 150, Problem Instance = 2



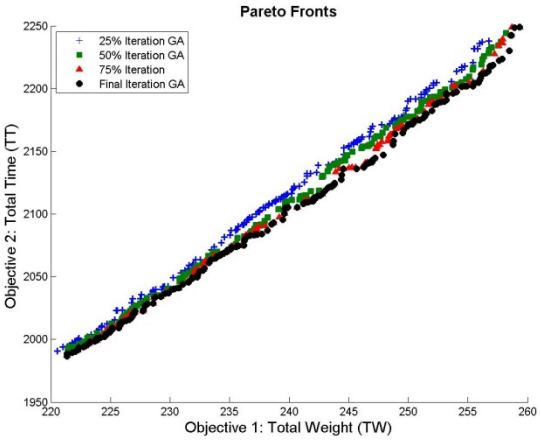
Network Size = 150, Problem Instance = 3



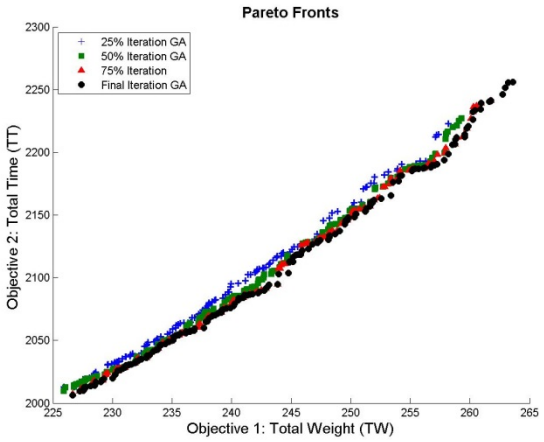
Network Size = 150, Problem Instance = 4



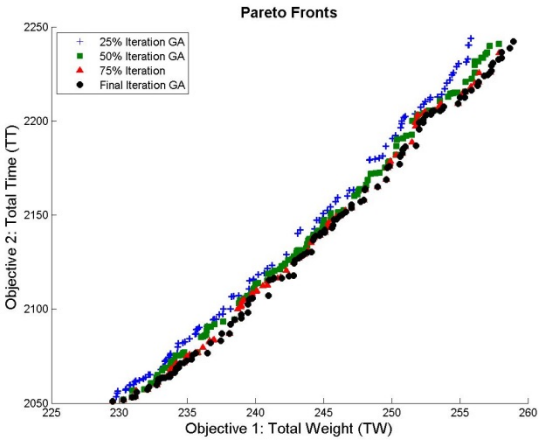
Network Size = 150, Problem Instance = 5



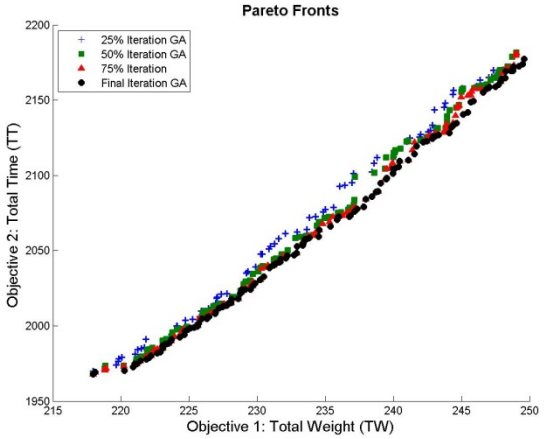
Network Size = 150, Problem Instance = 6



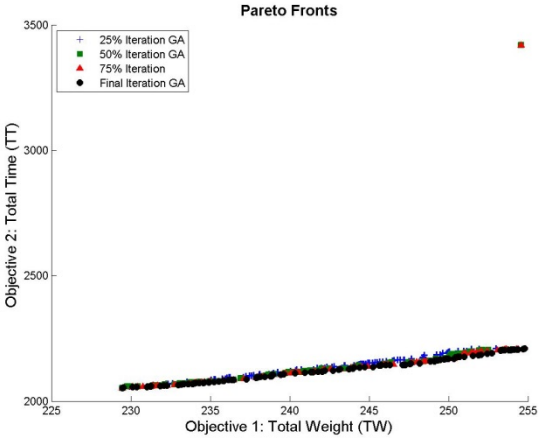
Network Size = 150, Problem Instance = 7



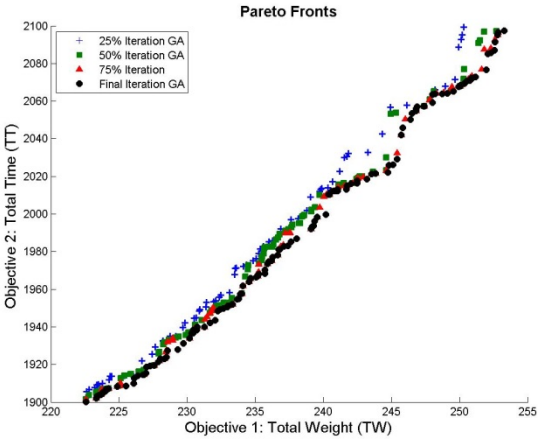
Network Size = 150, Problem Instance = 8



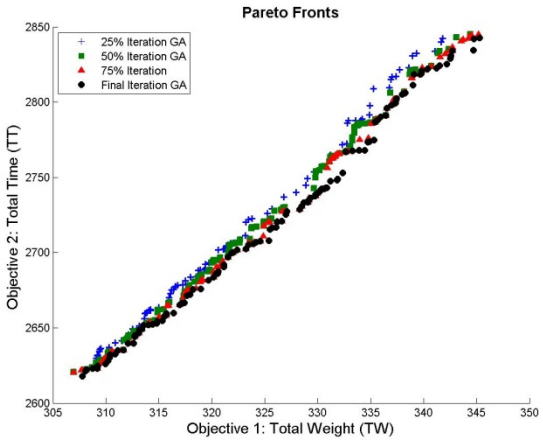
Network Size = 150, Problem Instance = 9



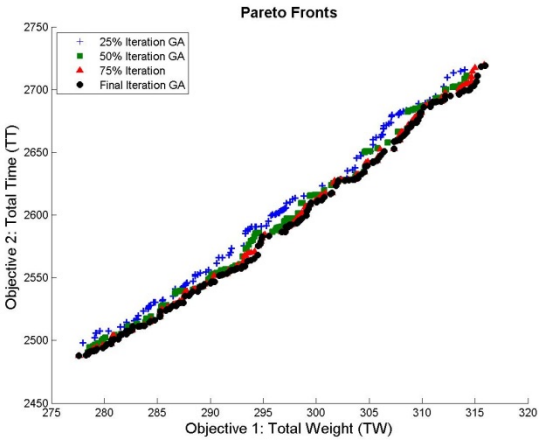
Network Size = 150, Problem Instance = 10



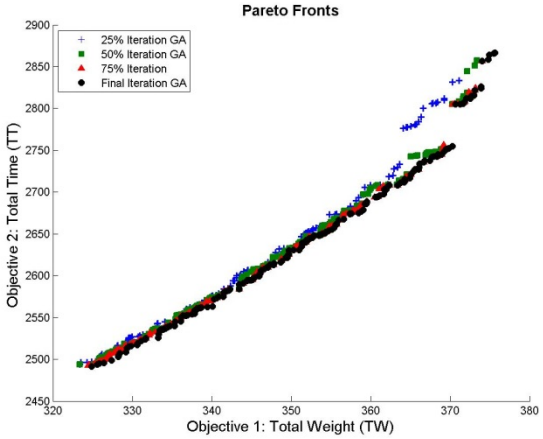
Network Size = 200, Problem Instance = 1



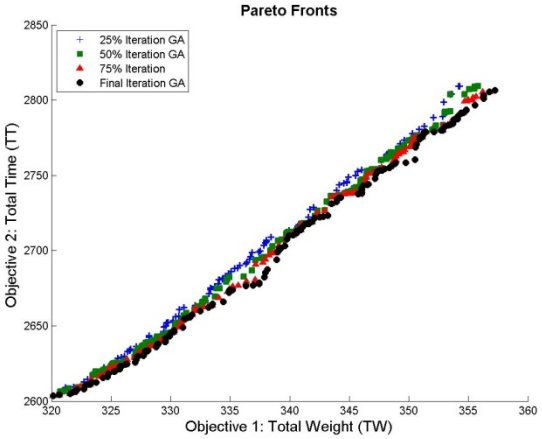
Network Size = 200, Problem Instance = 2



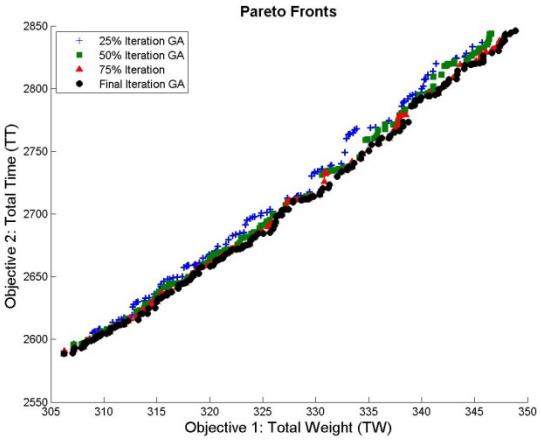
Network Size = 200, Problem Instance = 3



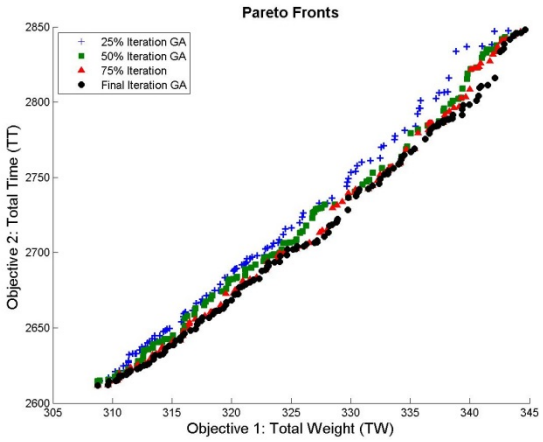
Network Size = 200, Problem Instance = 4



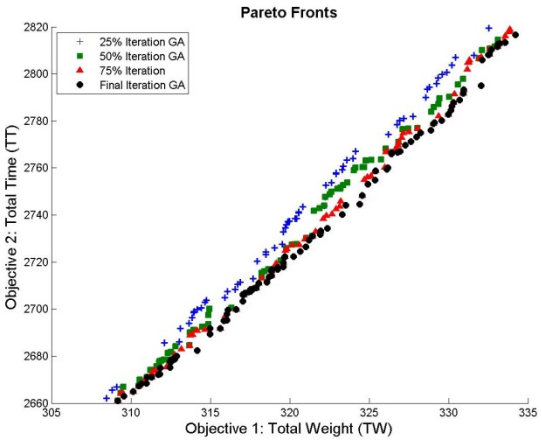
Network Size = 200, Problem Instance = 5



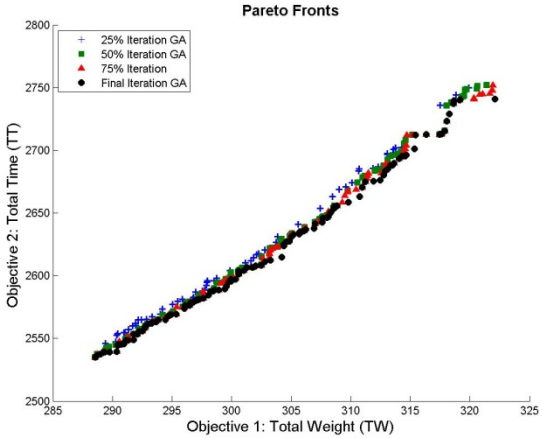
Network Size = 200, Problem Instance = 6



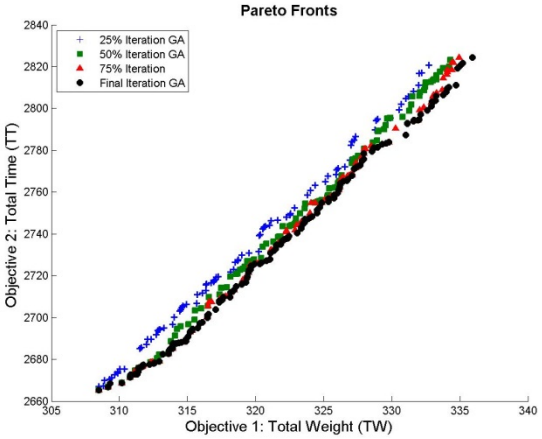
Network Size = 200, Problem Instance = 7



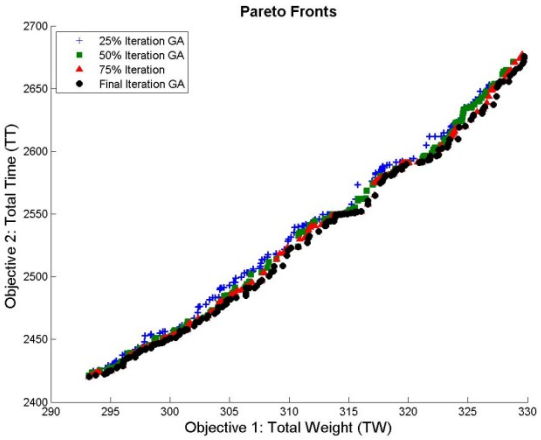
Network Size = 200, Problem Instance = 8



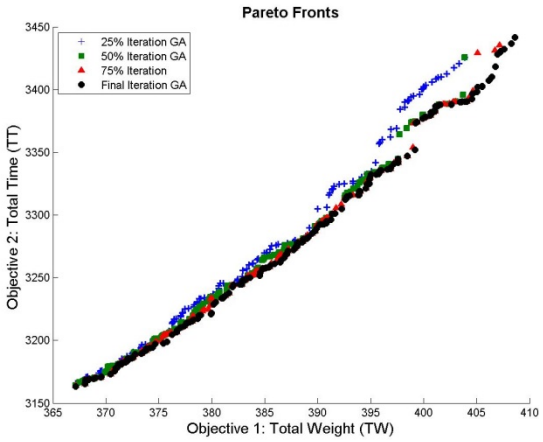
Network Size = 200, Problem Instance = 9



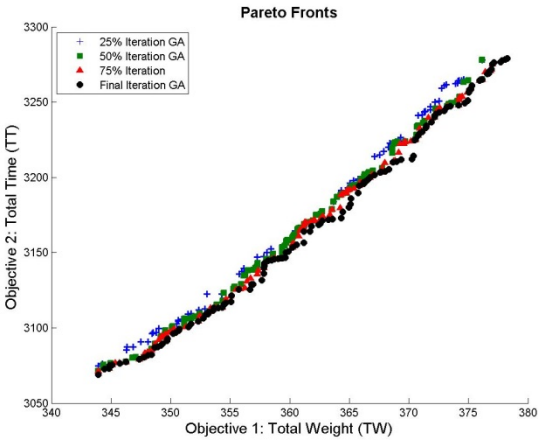
Network Size = 200, Problem Instance = 10



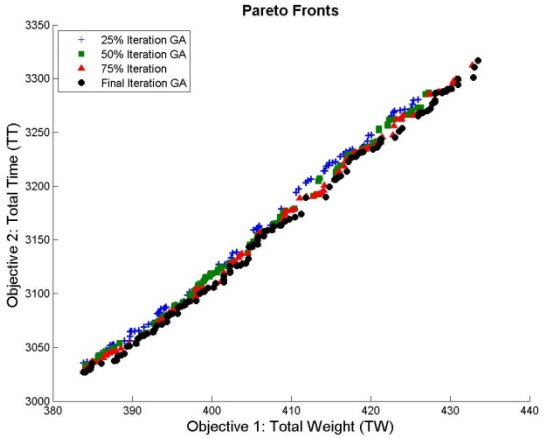
Network Size = 250, Problem Instance = 1



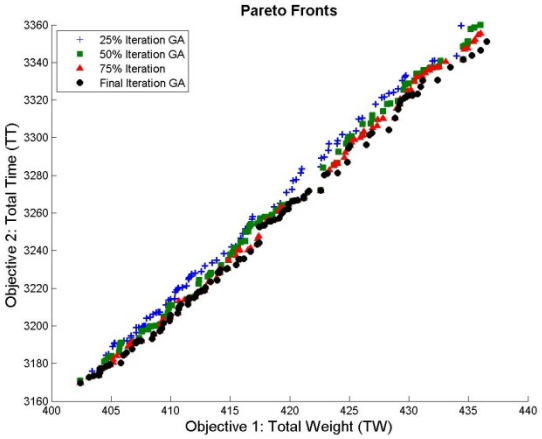
Network Size = 250, Problem Instance = 2



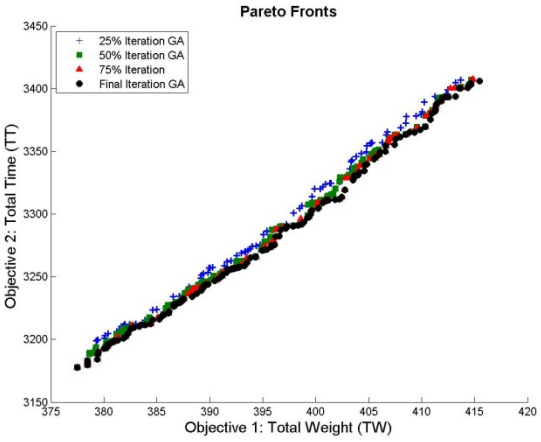
Network Size = 250, Problem Instance = 3



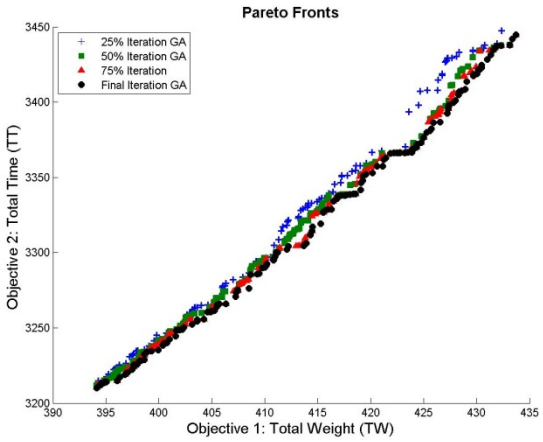
Network Size = 250, Problem Instance = 4



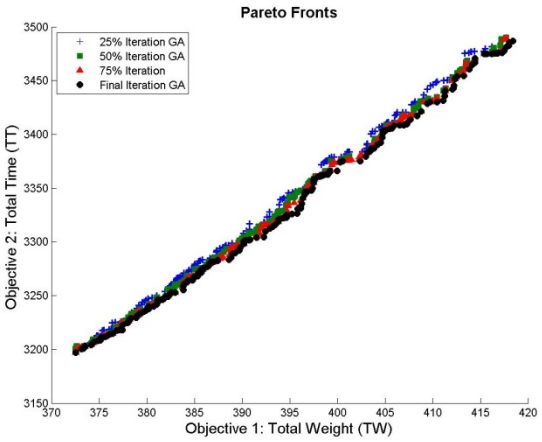
Network Size = 250, Problem Instance = 5



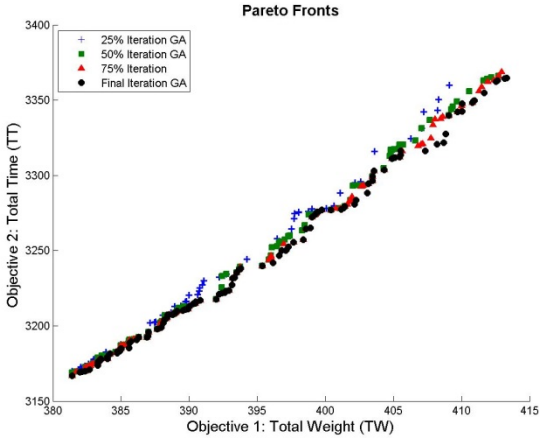
Network Size = 250, Problem Instance = 6



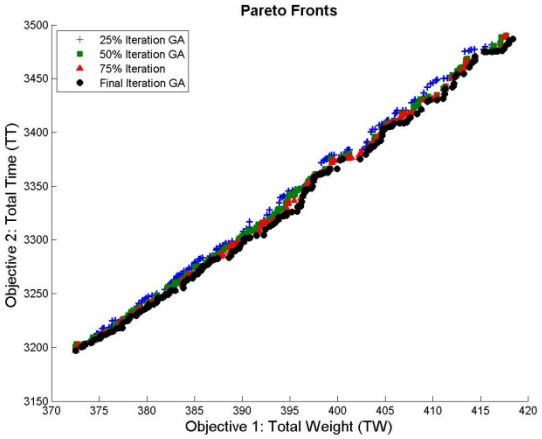
Network Size = 250, Problem Instance = 7



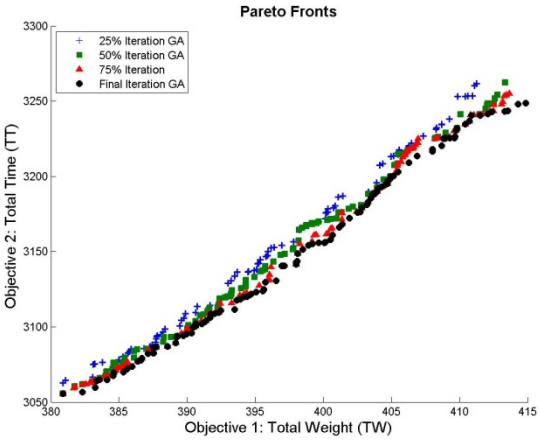
Network Size = 250, Problem Instance = 8



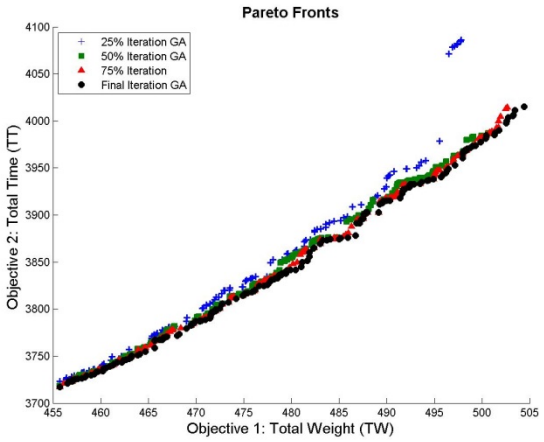
Network Size = 250, Problem Instance = 9



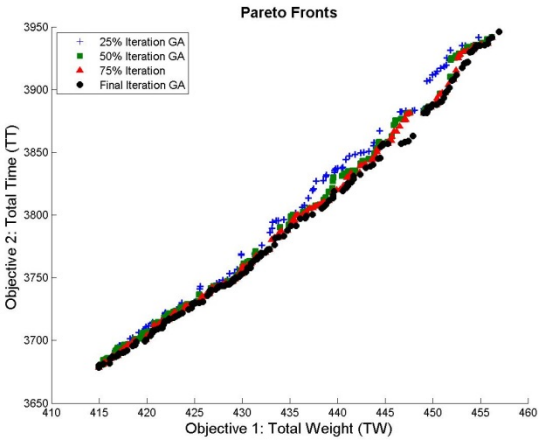
Network Size = 250, Problem Instance = 10



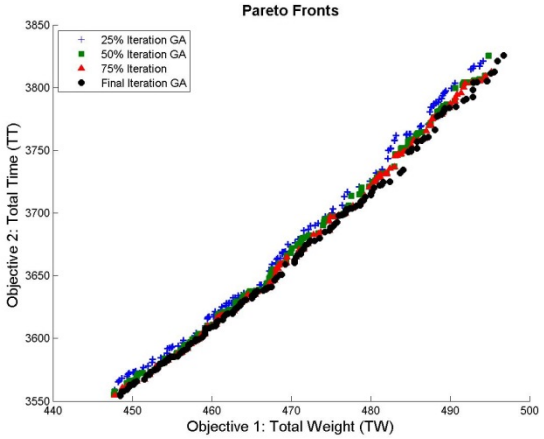
Network Size = 300, Problem Instance = 1



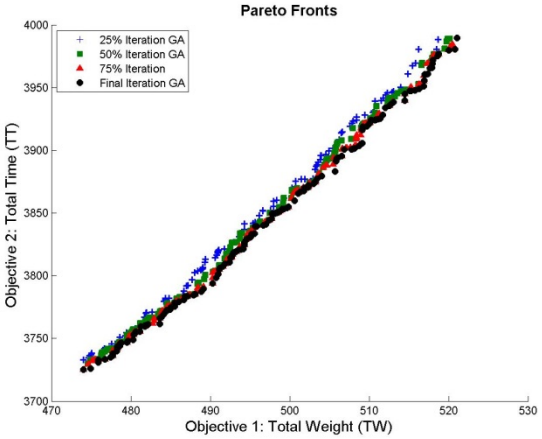
Network Size = 300, Problem Instance = 2



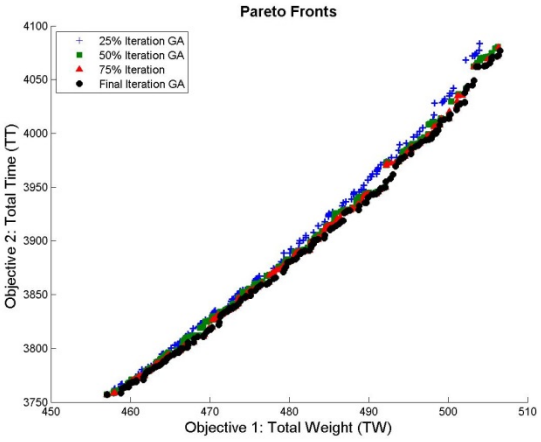
Network Size = 300, Problem Instance = 3



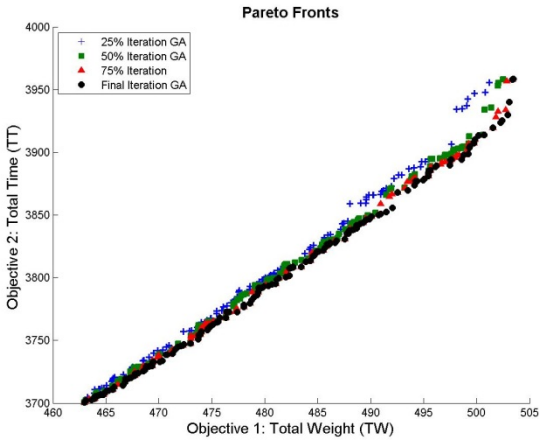
Network Size = 300, Problem Instance = 4



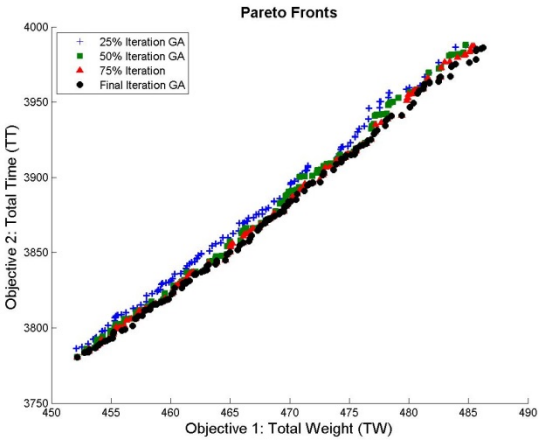
Network Size = 300, Problem Instance = 5



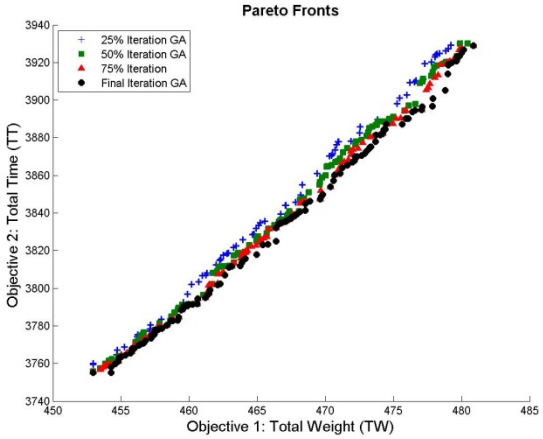
Network Size = 300, Problem Instance = 6



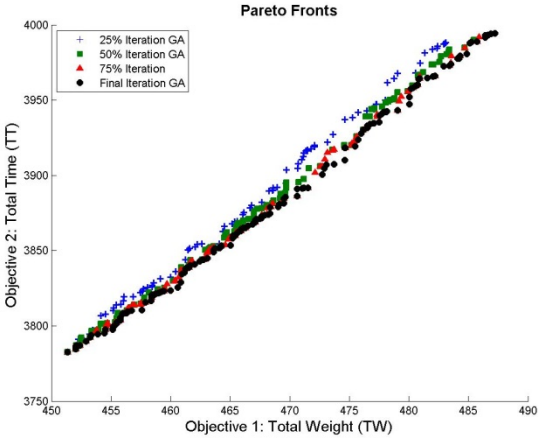
Network Size = 300, Problem Instance = 7



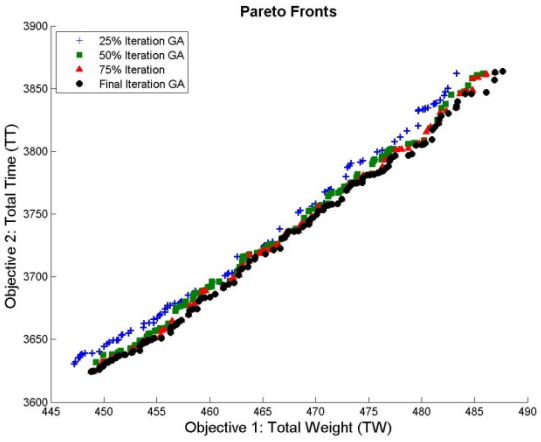
Network Size = 300, Problem Instance = 8



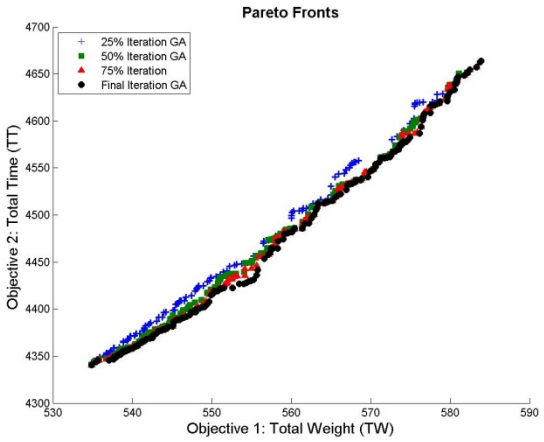
Network Size = 300, Problem Instance = 9



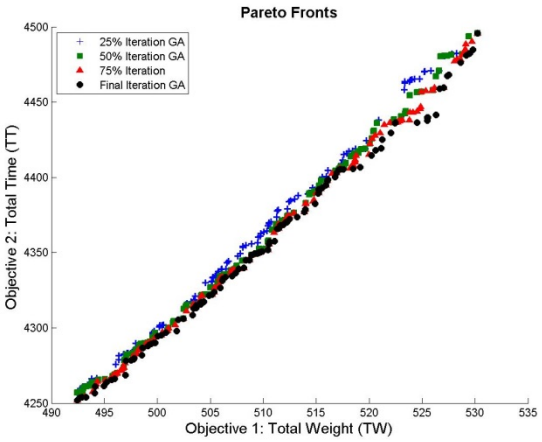
Network Size = 300, Problem Instance = 10



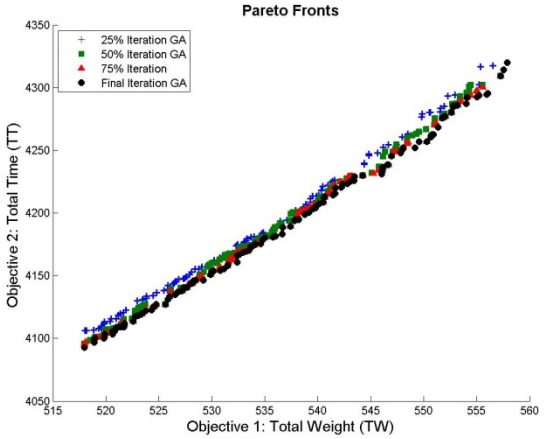
Network Size = 350, Problem Instance = 1



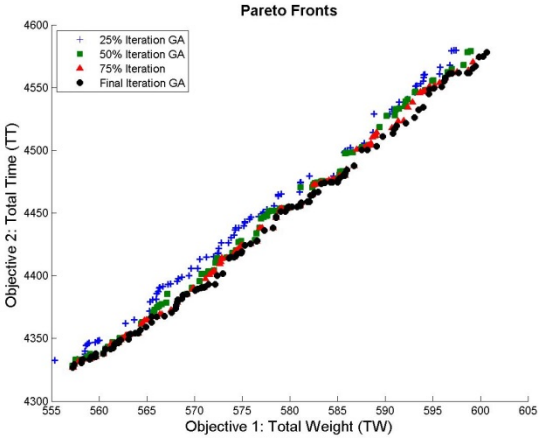
Network Size = 350, Problem Instance = 2



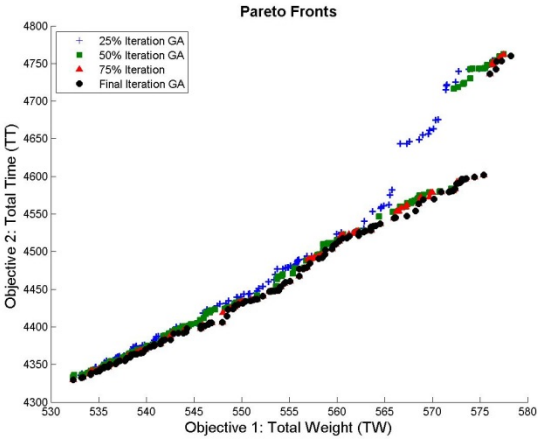
Network Size = 350, Problem Instance = 3



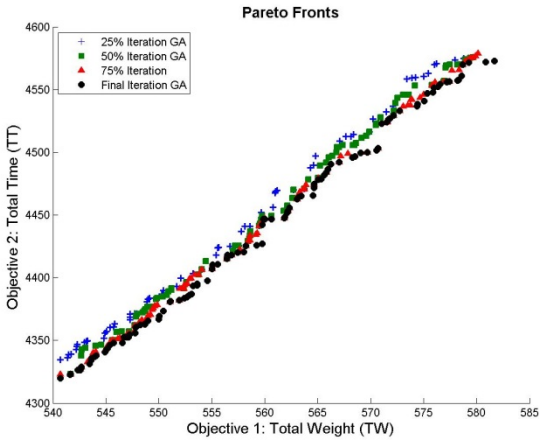
Network Size = 350, Problem Instance = 4



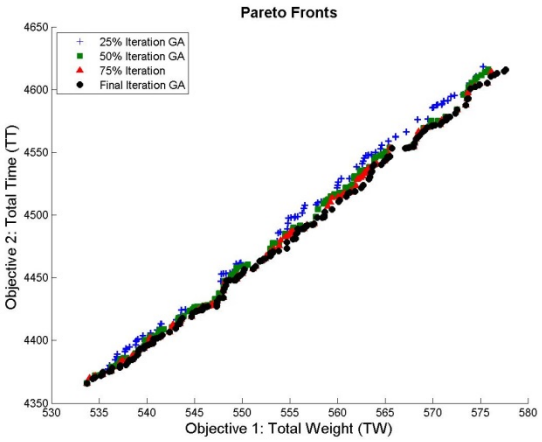
Network Size = 350, Problem Instance = 5



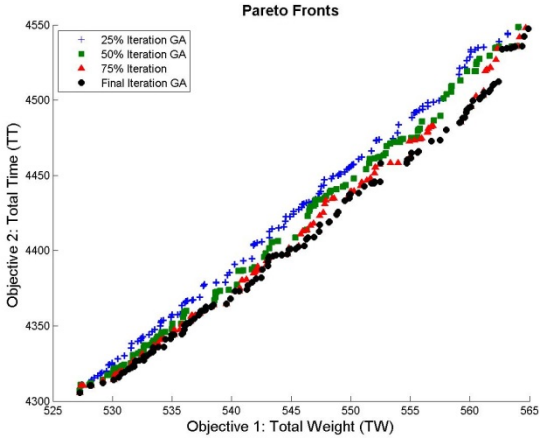
Network Size = 350, Problem Instance = 6



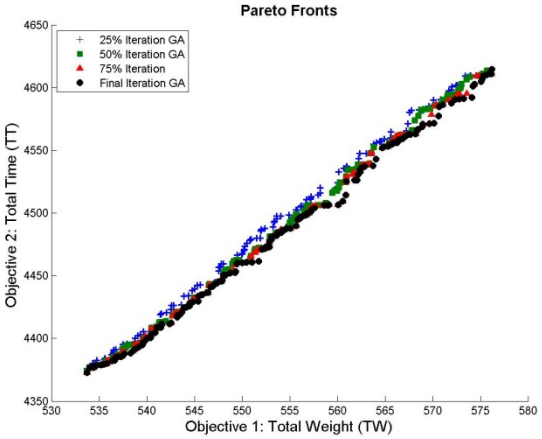
Network Size = 350, Problem Instance = 7



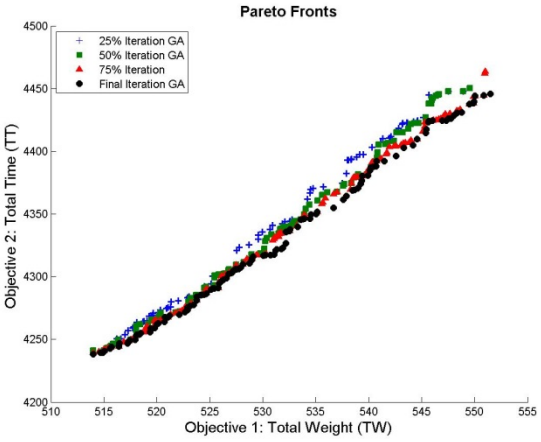
Network Size = 350, Problem Instance = 8



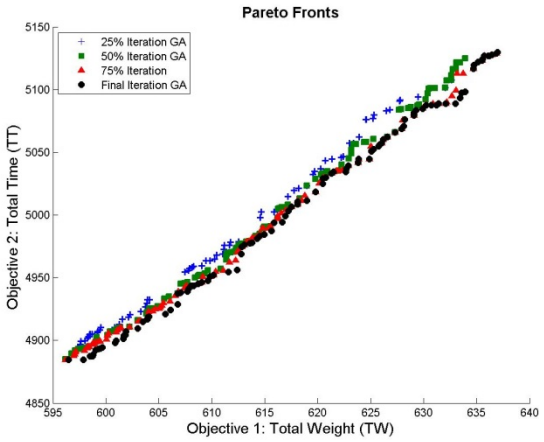
Network Size = 350, Problem Instance = 9



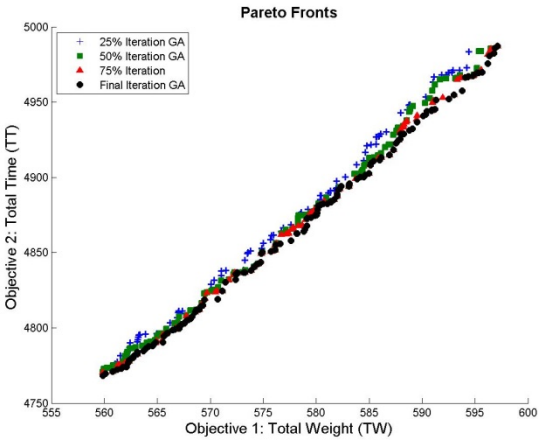
Network Size = 350, Problem Instance = 10



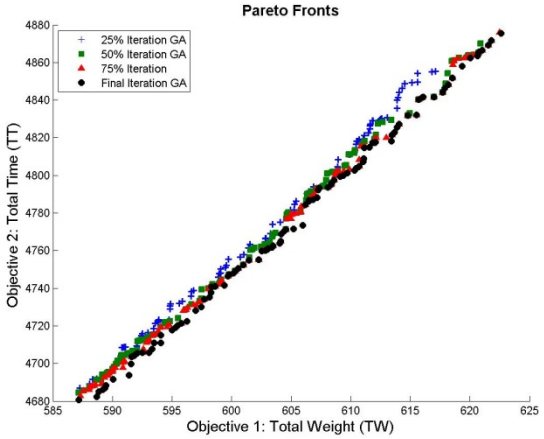
Network Size = 400, Problem Instance = 1



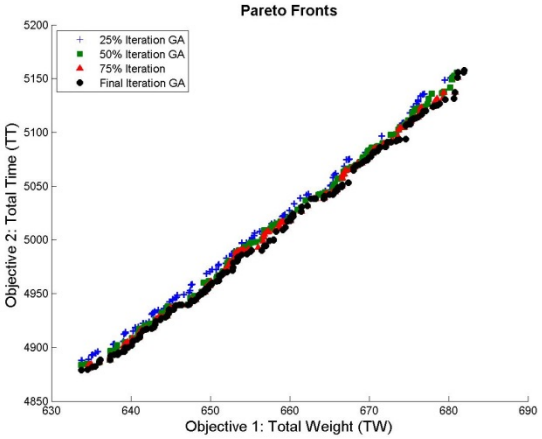
Network Size = 400, Problem Instance = 2



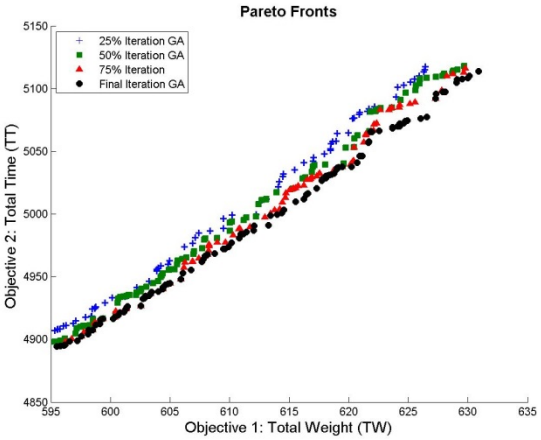
Network Size = 400, Problem Instance = 3



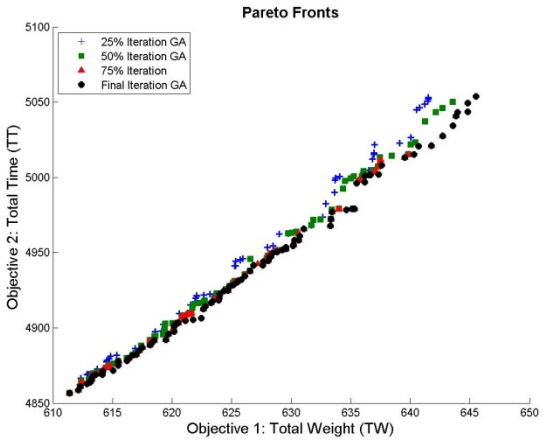
Network Size = 400, Problem Instance = 4



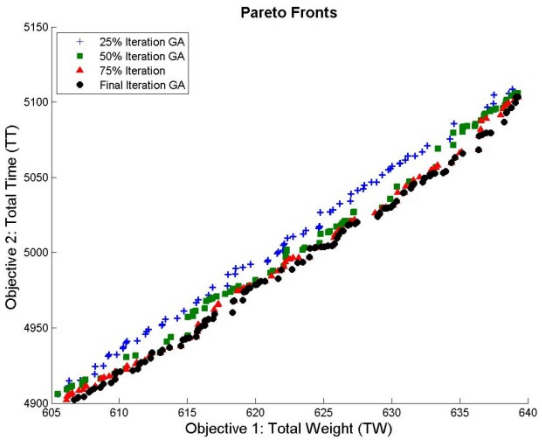
Network Size = 400, Problem Instance = 5



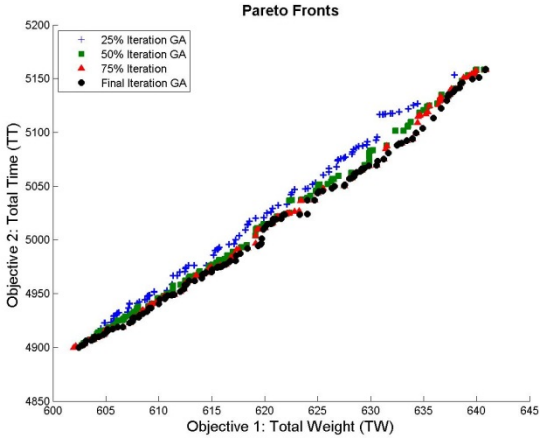
Network Size = 400, Problem Instance = 6



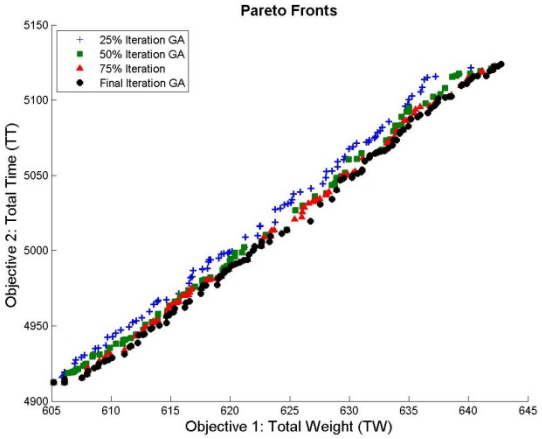
Network Size = 400, Problem Instance = 7



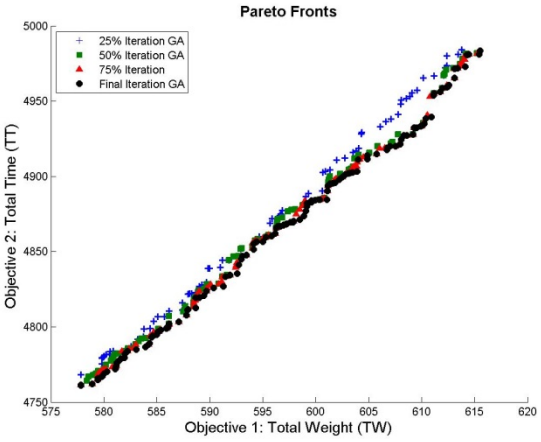
Network Size = 400, Problem Instance = 8



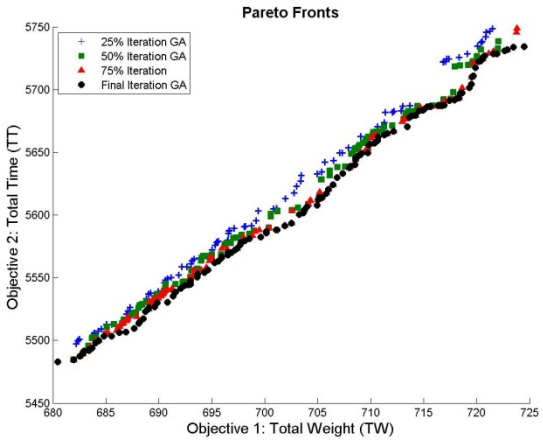
Network Size = 400, Problem Instance = 9



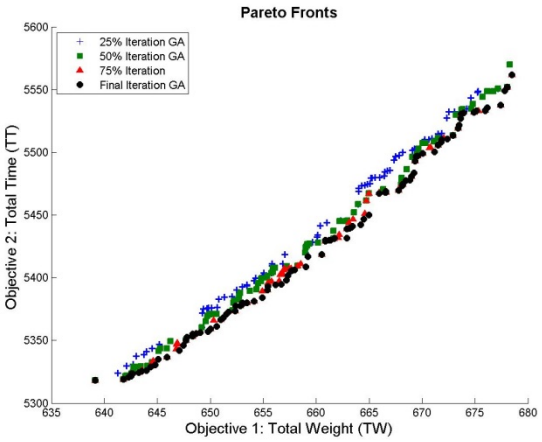
Network Size = 400, Problem Instance = 10



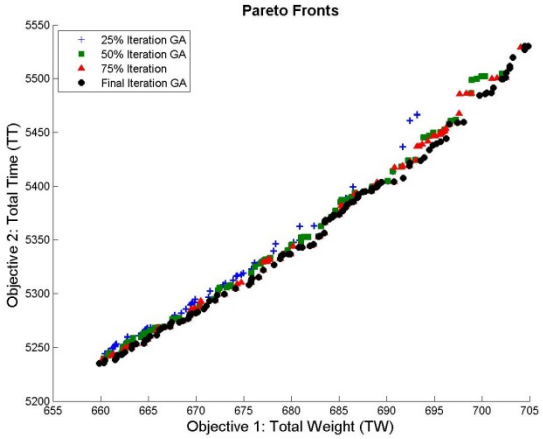
Network Size = 450, Problem Instance = 1



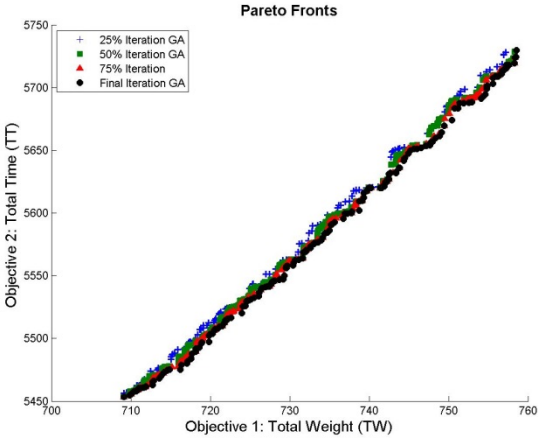
Network Size = 450, Problem Instance = 2



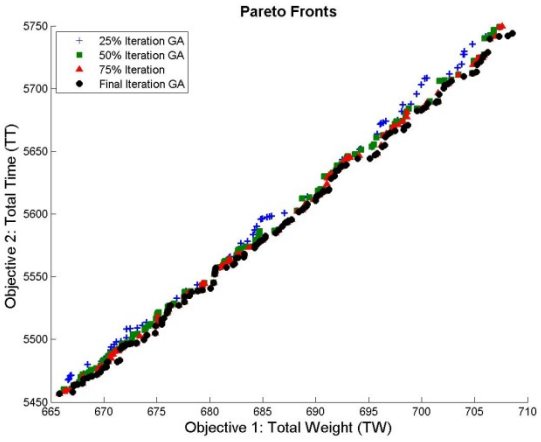
Network Size = 450, Problem Instance = 3



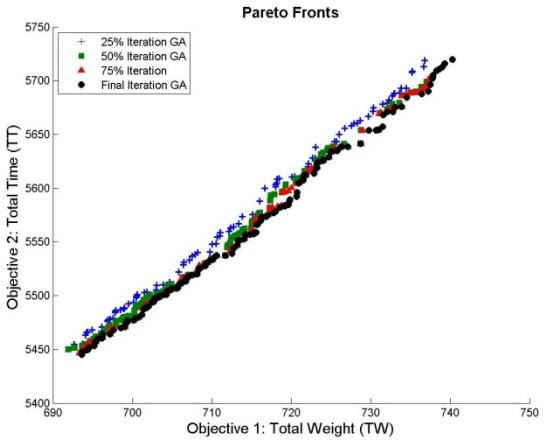
Network Size = 450, Problem Instance = 4



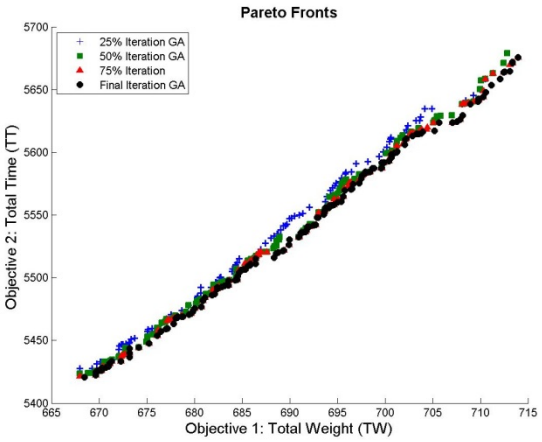
Network Size = 450, Problem Instance = 5



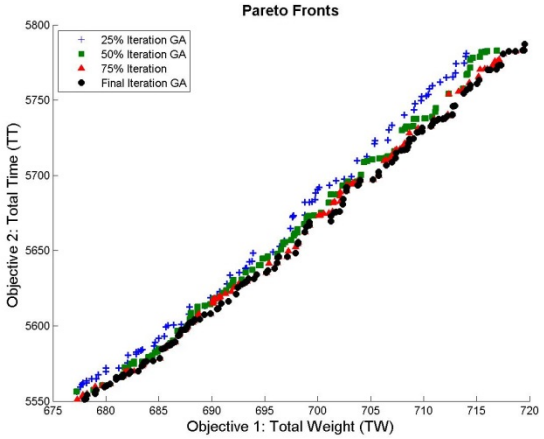
Network Size = 450, Problem Instance = 6



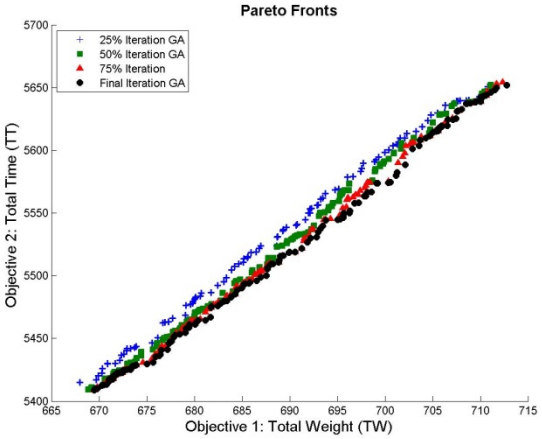
Network Size = 450, Problem Instance = 7



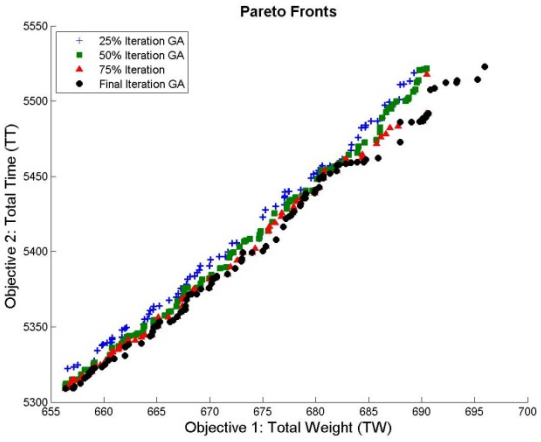
Network Size = 450, Problem Instance = 8



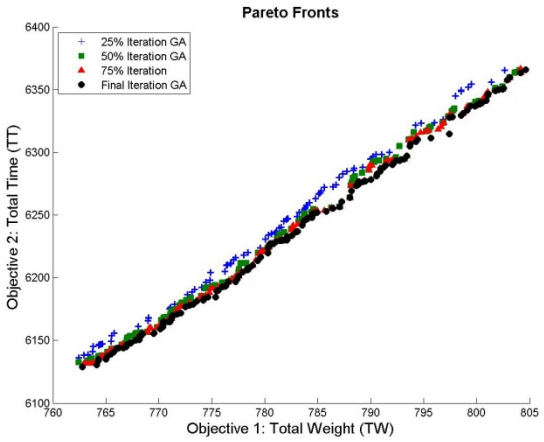
Network Size = 450, Problem Instance = 9



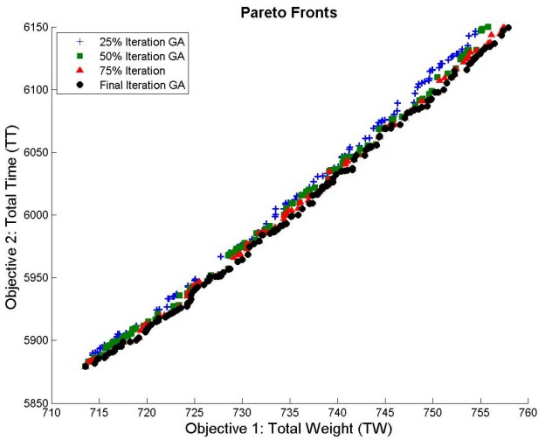
Network Size = 450, Problem Instance = 10



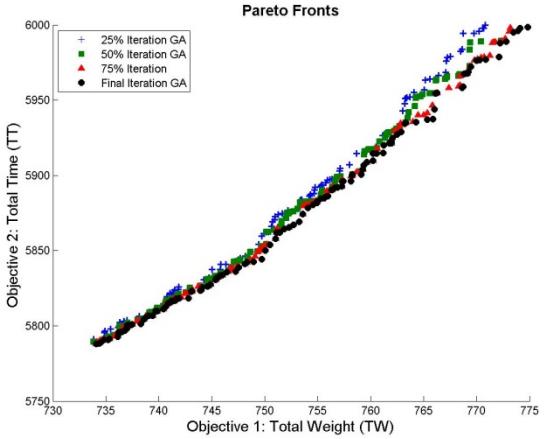
Network Size = 500, Problem Instance = 1



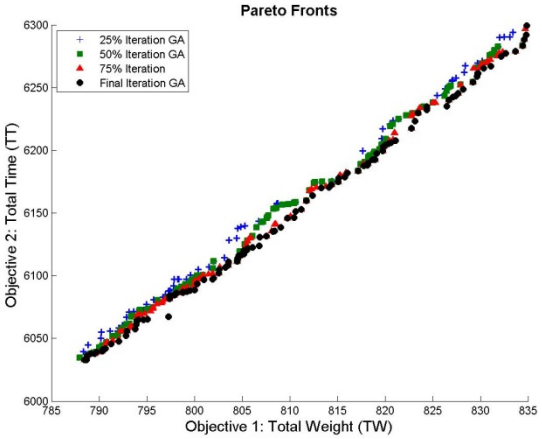
Network Size = 500, Problem Instance = 2



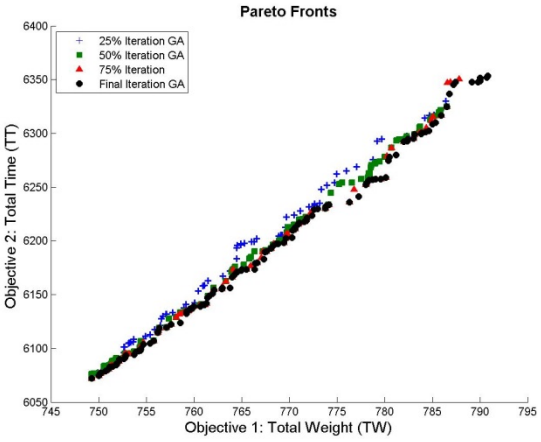
Network Size = 500, Problem Instance = 3



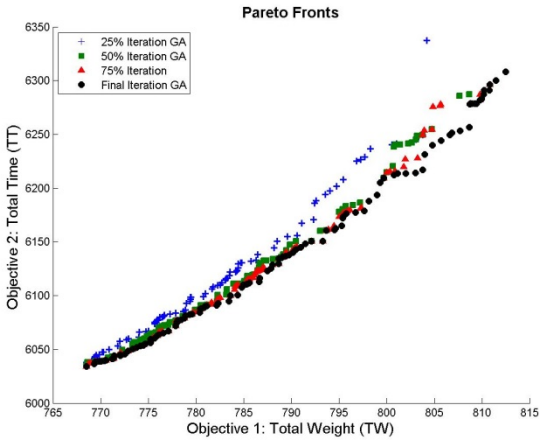
Network Size = 500, Problem Instance = 4



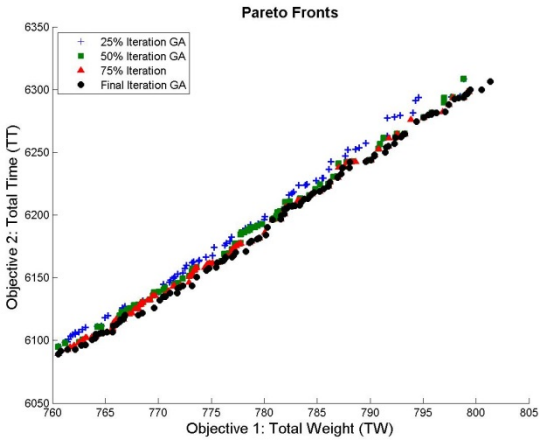
Network Size = 500, Problem Instance = 5



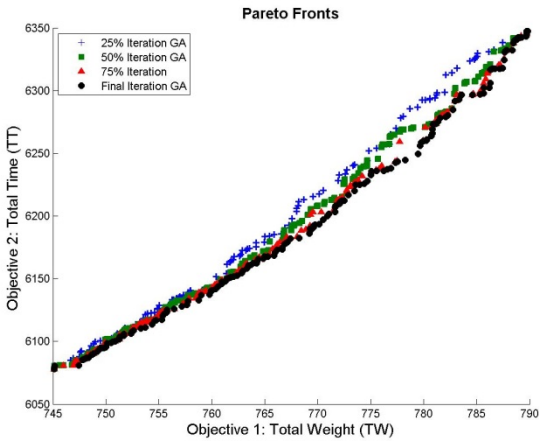
Network Size = 500, Problem Instance = 6



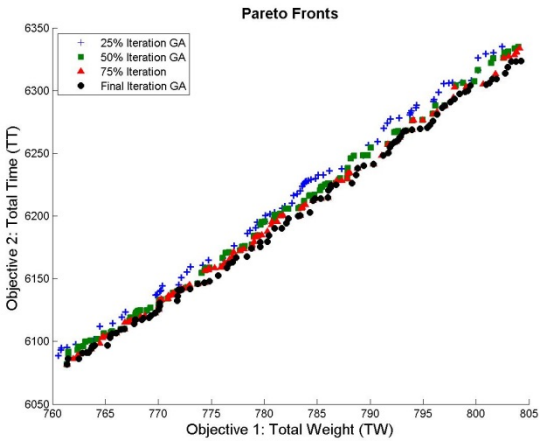
Network Size = 500, Problem Instance = 7



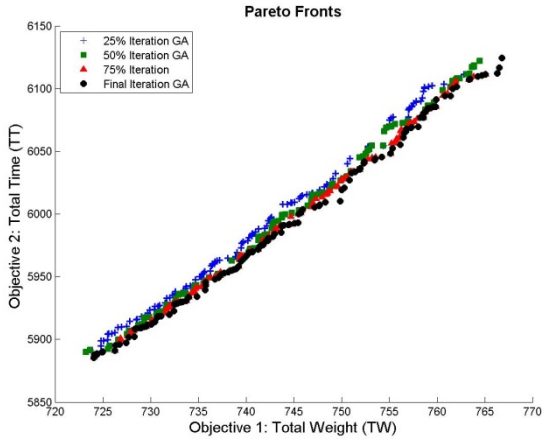
Network Size = 500, Problem Instance = 8



Network Size = 500, Problem Instance = 9



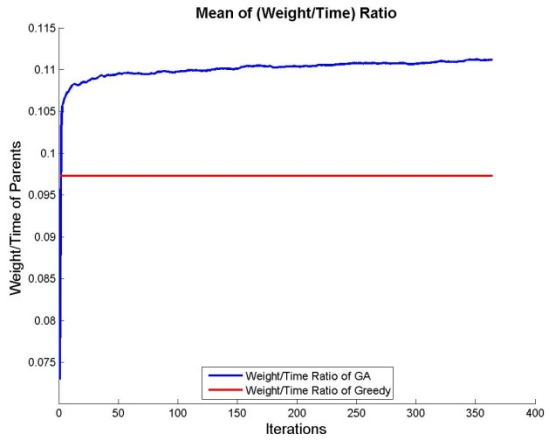
Network Size = 500, Problem Instance = 10



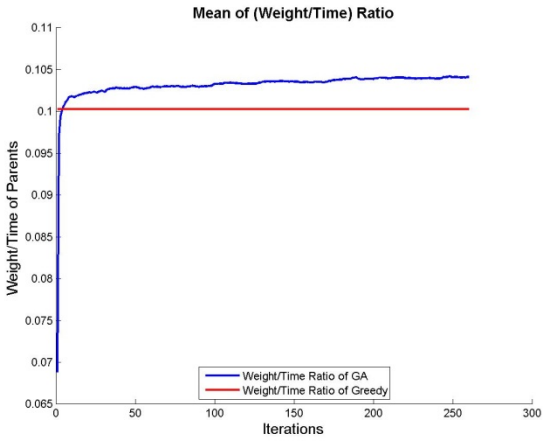
Appendix C: Improvement of the Weight to Time Ratio of The Genetic Algorithm

Figure 6: Average TW/TT Ratios vs. Iterations

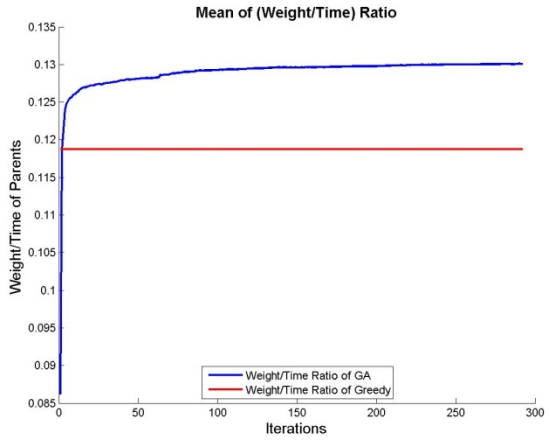
Network Size = 100, Problem Instance = 1



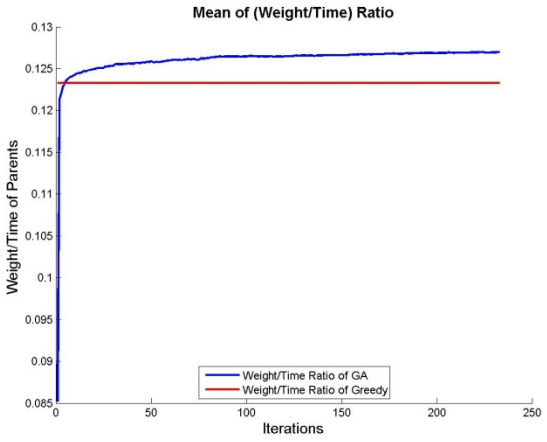
Network Size = 100, Problem Instance = 2



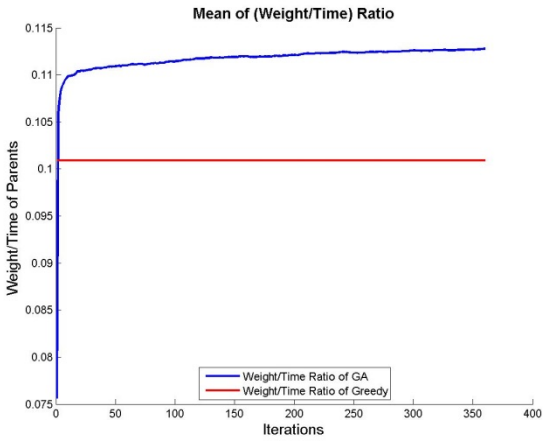
Network Size = 100, Problem Instance = 3



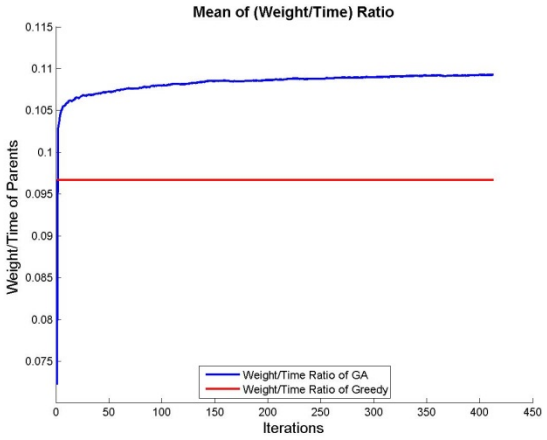
Network Size = 100, Problem Instance = 4



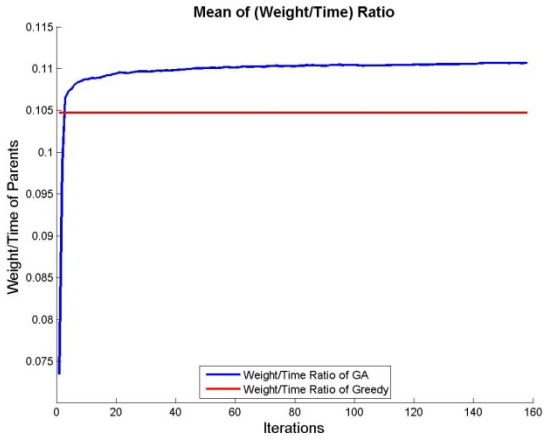
Network Size = 100, Problem Instance = 5



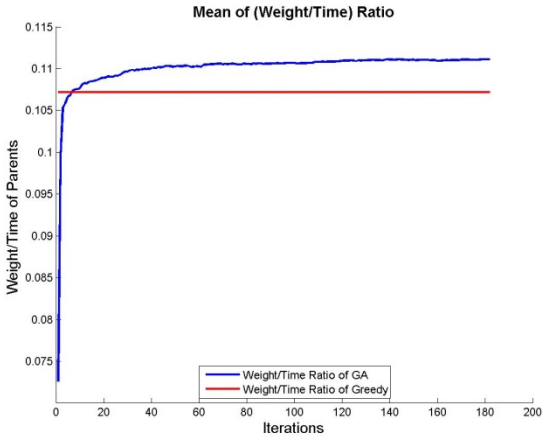
Network Size = 100, Problem Instance = 6



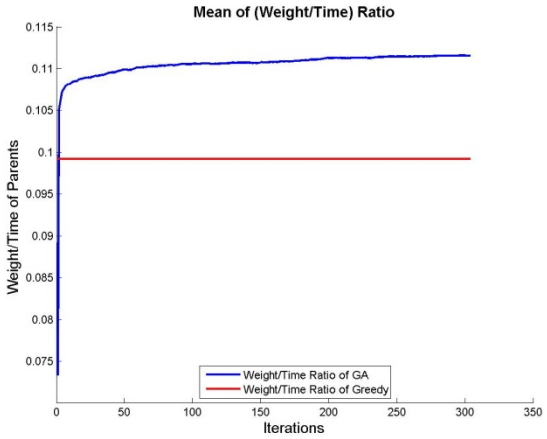
Network Size = 100, Problem Instance = 7



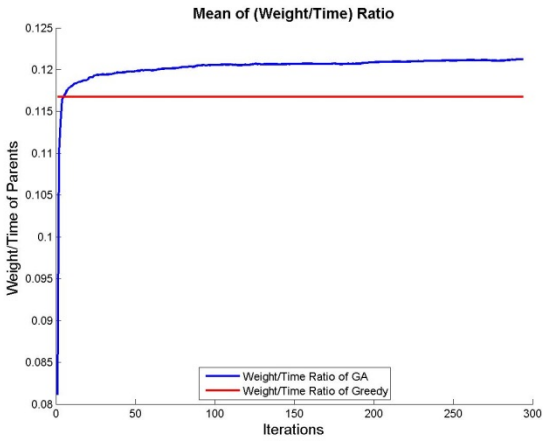
Network Size = 100, Problem Instance = 8



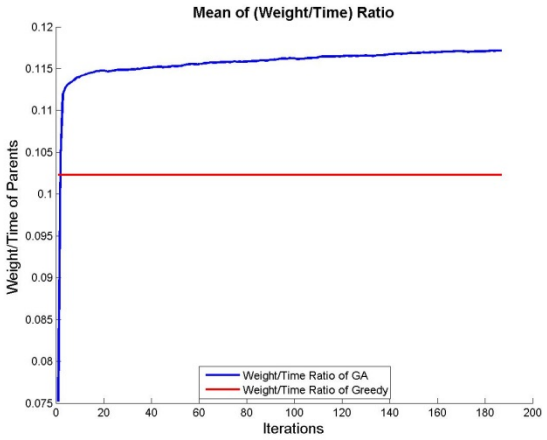
Network Size = 100, Problem Instance = 9



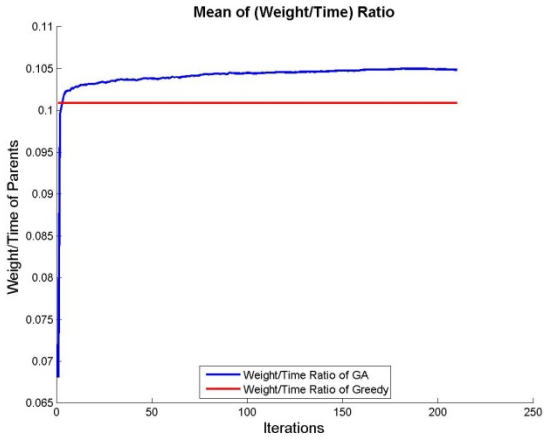
Network Size = 100, Problem Instance = 10



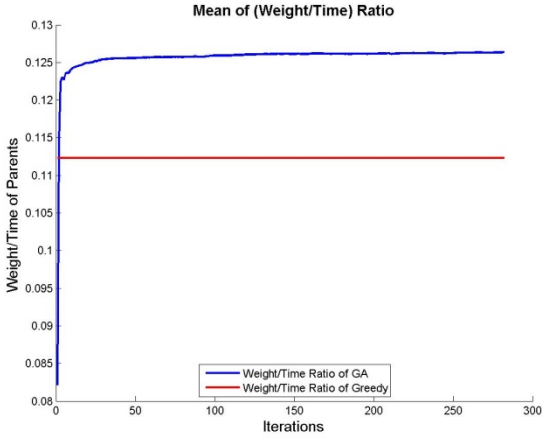
Network Size = 150, Problem Instance = 1



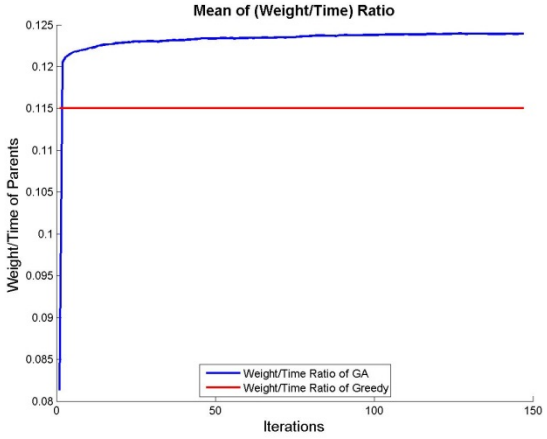
Network Size = 150, Problem Instance = 2



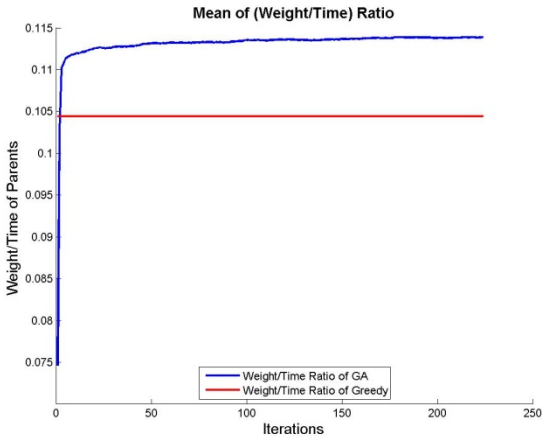
Network Size = 150, Problem Instance = 3



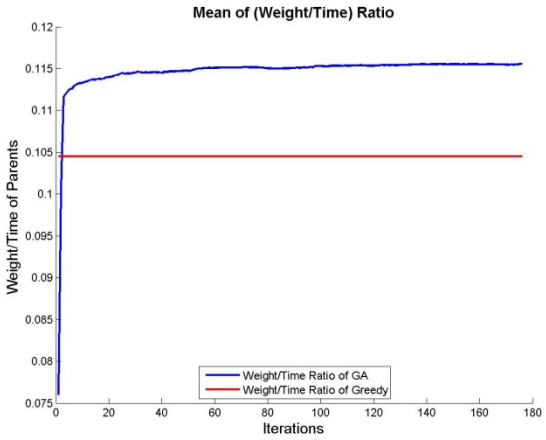
Network Size = 150, Problem Instance = 4



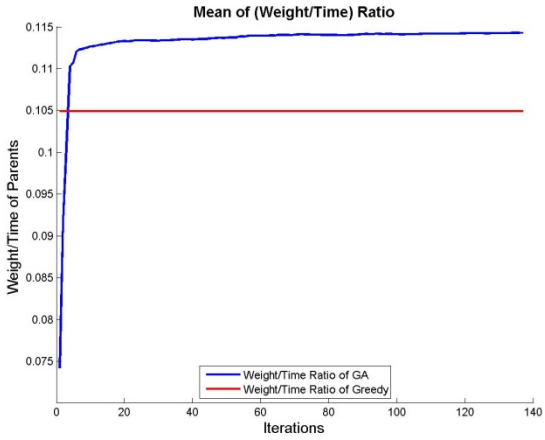
Network Size = 150, Problem Instance = 5



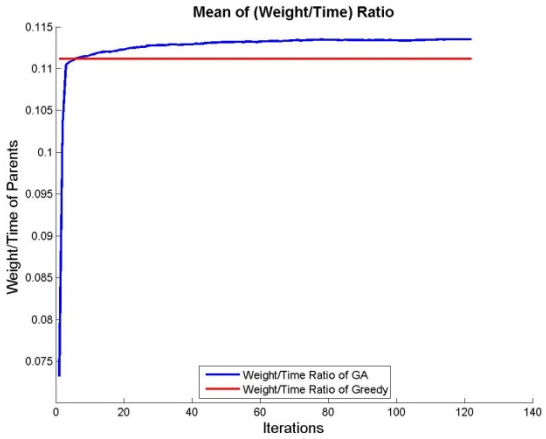
Network Size = 150, Problem Instance = 6



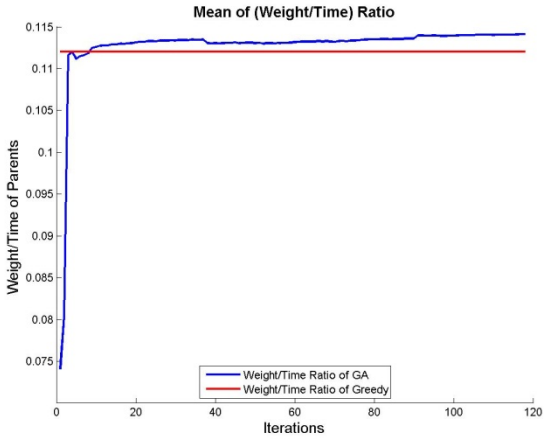
Network Size = 150, Problem Instance = 7



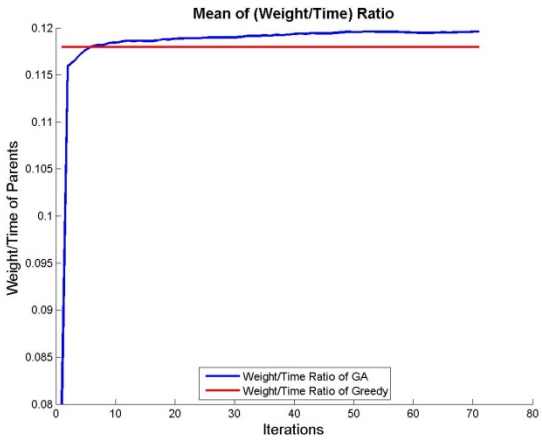
Network Size = 150, Problem Instance = 8



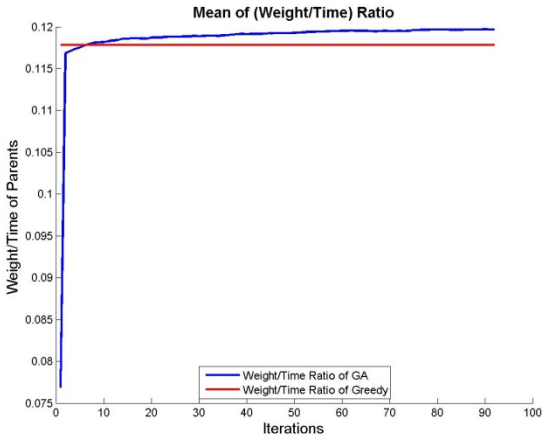
Network Size = 150, Problem Instance = 9



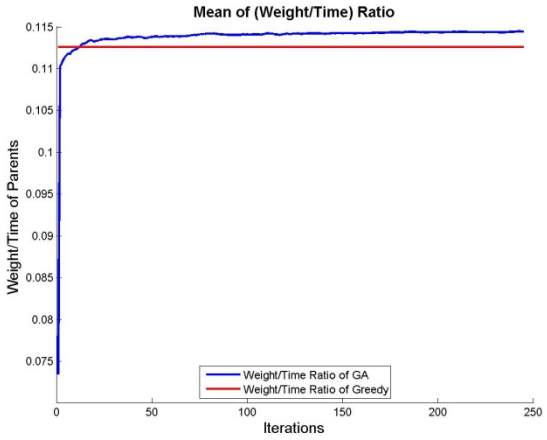
Network Size = 150, Problem Instance = 10



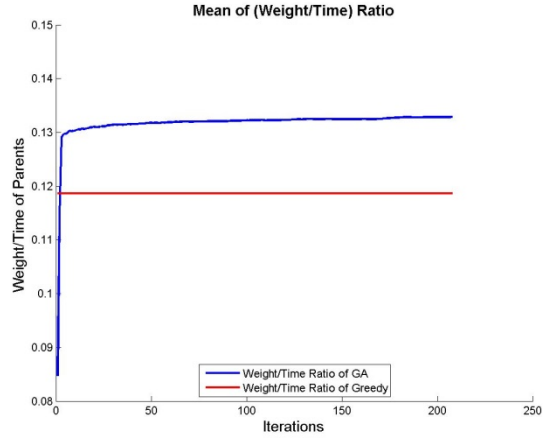
Network Size = 200, Problem Instance = 1



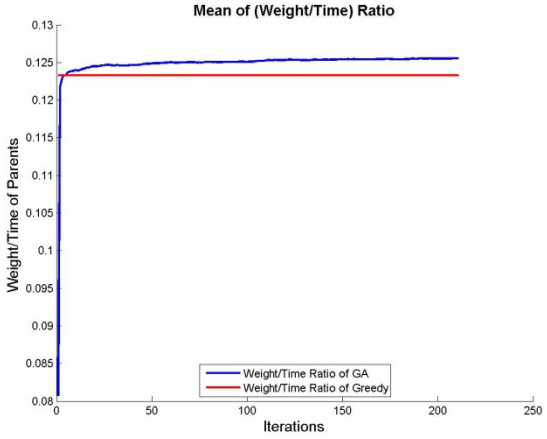
Network Size = 200, Problem Instance = 2



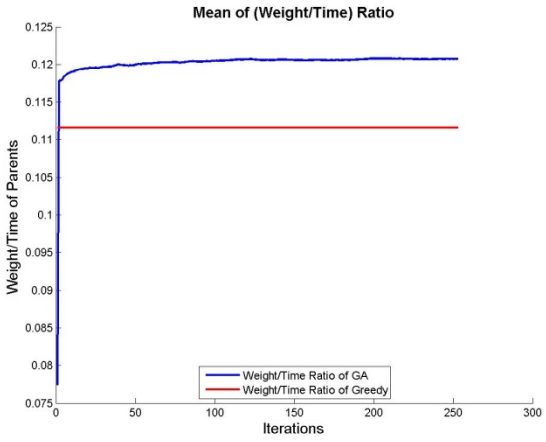
Network Size = 200, Problem Instance = 3



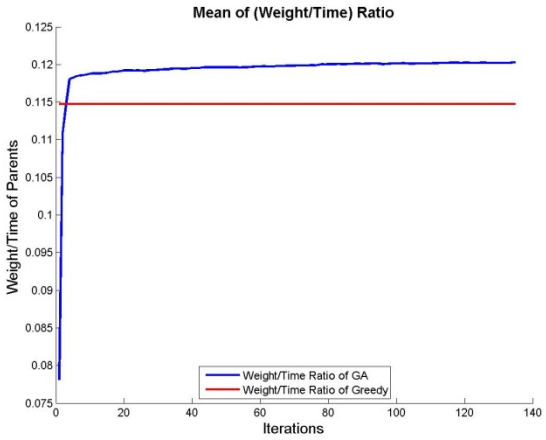
Network Size = 200, Problem Instance = 4



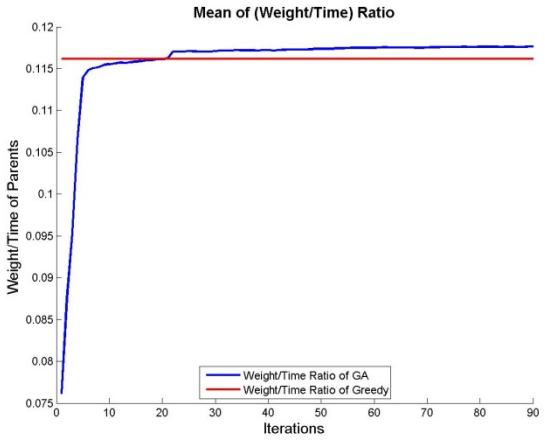
Network Size = 200, Problem Instance = 5



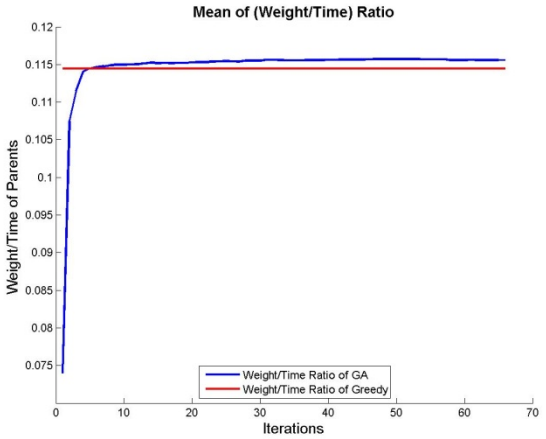
Network Size = 200, Problem Instance = 6



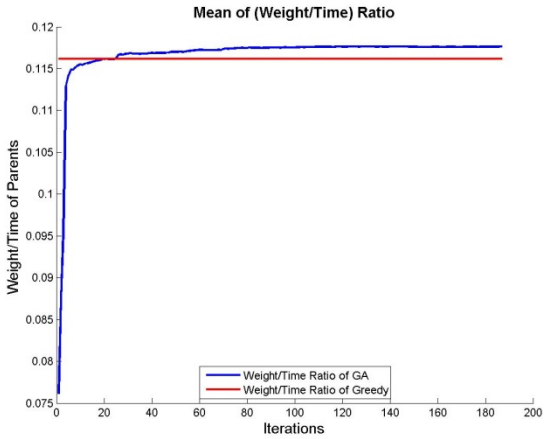
Network Size = 200, Problem Instance = 7



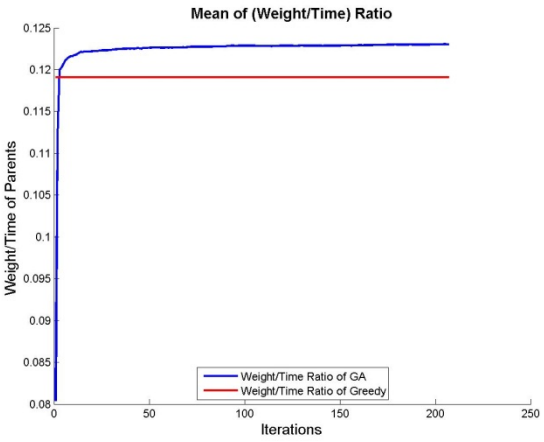
Network Size = 200, Problem Instance = 8



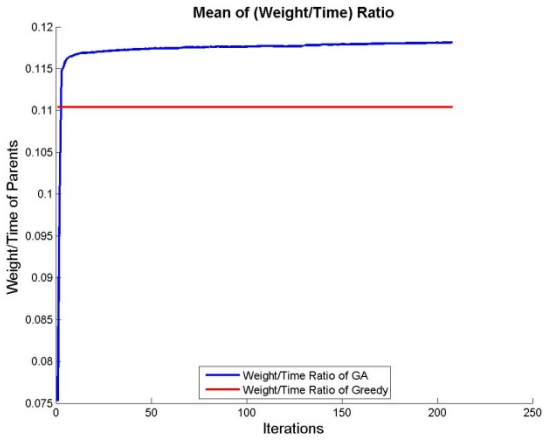
Network Size = 200, Problem Instance = 9



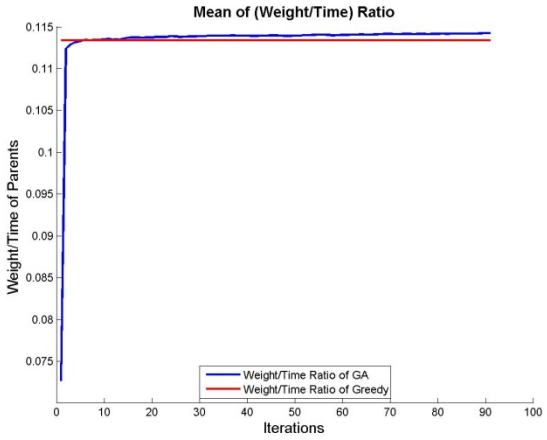
Network Size = 200, Problem Instance = 10



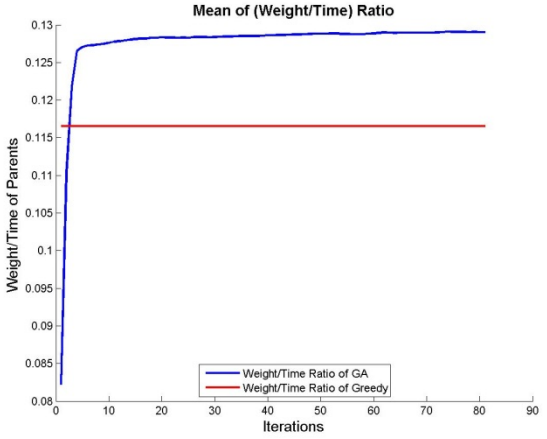
Network Size = 250, Problem Instance = 1



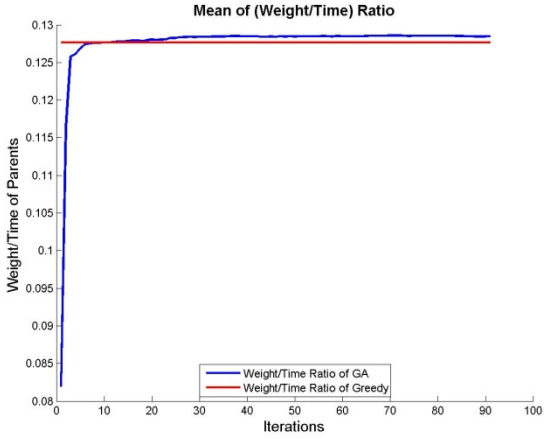
Network Size = 250, Problem Instance = 2



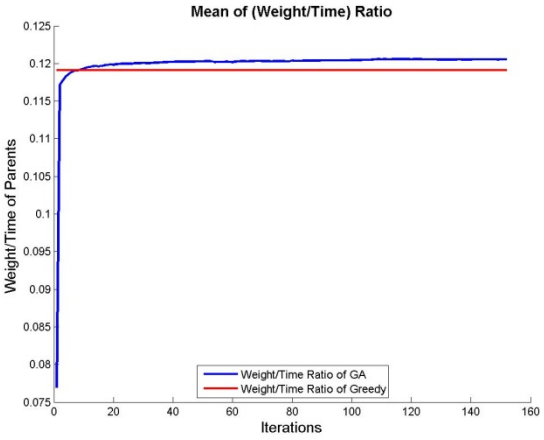
Network Size = 250, Problem Instance = 3



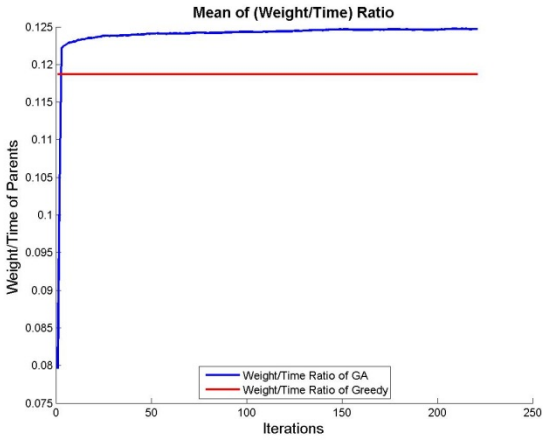
Network Size = 250, Problem Instance = 4



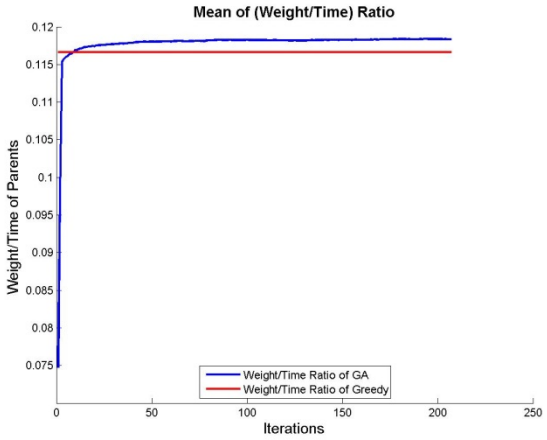
Network Size = 250, Problem Instance = 5



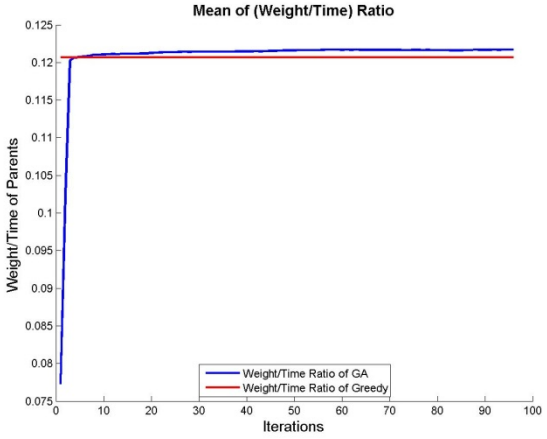
Network Size = 250, Problem Instance = 6



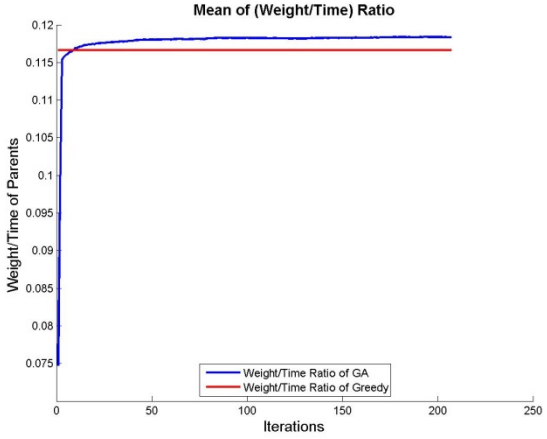
Network Size = 250, Problem Instance = 7



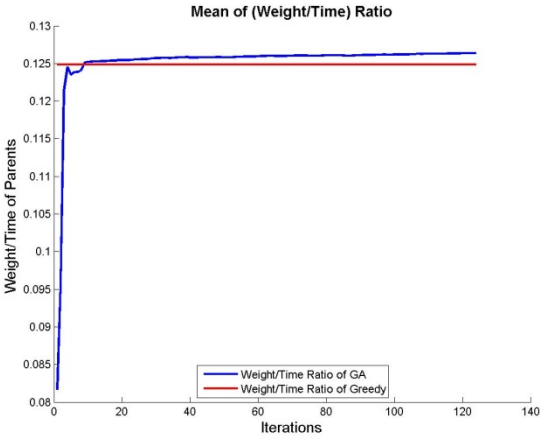
Network Size = 250, Problem Instance = 8



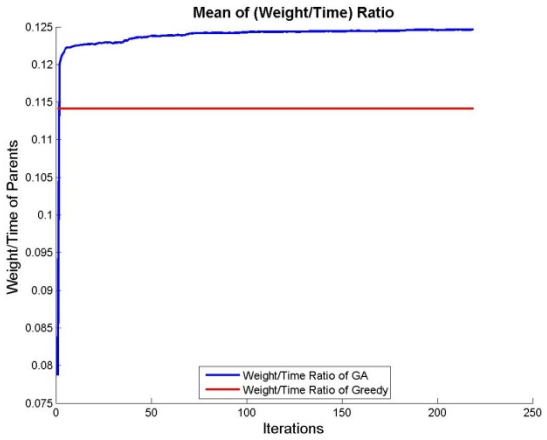
Network Size = 250, Problem Instance = 9



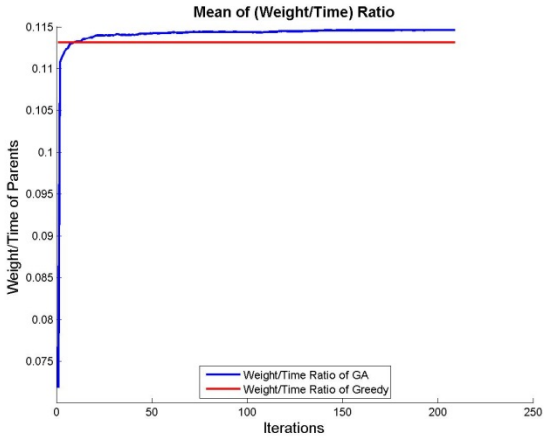
Network Size = 250, Problem Instance = 10



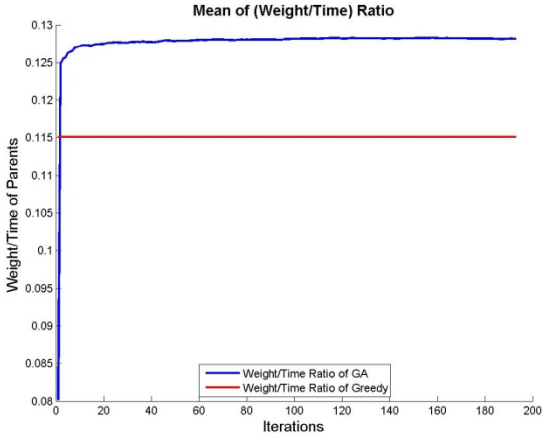
Network Size = 300, Problem Instance = 1



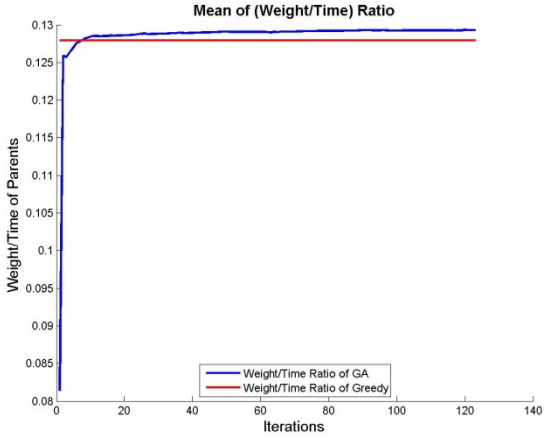
Network Size = 300, Problem Instance = 2



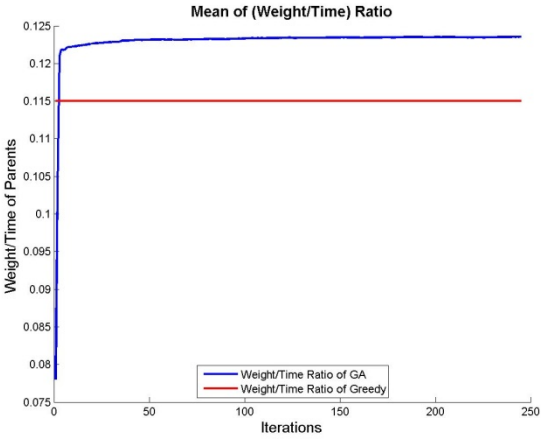
Network Size = 300, Problem Instance = 3



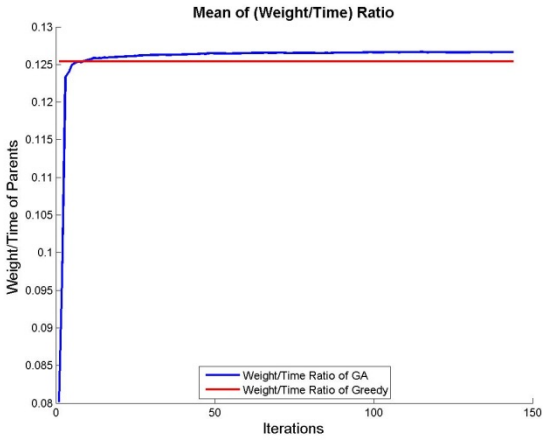
Network Size = 300, Problem Instance = 4



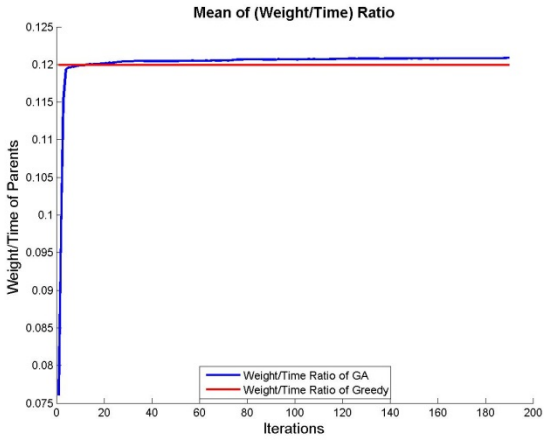
Network Size = 300, Problem Instance = 5



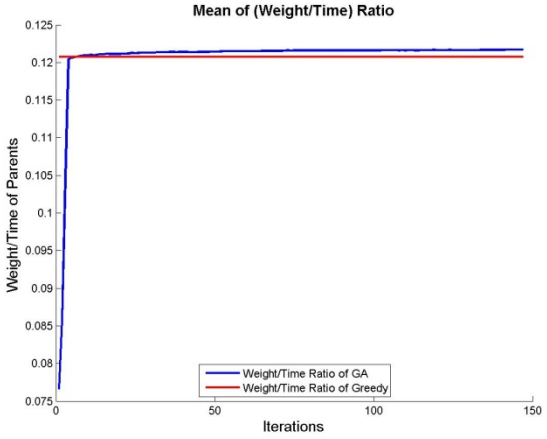
Network Size = 300, Problem Instance = 6



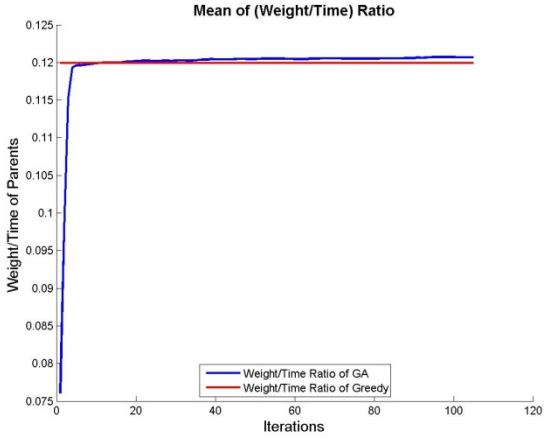
Network Size = 300, Problem Instance = 7



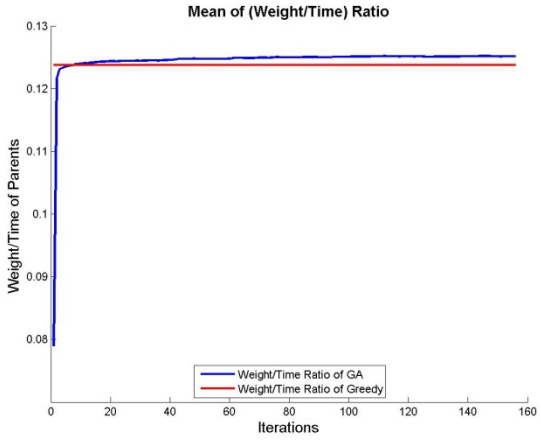
Network Size = 300, Problem Instance = 8



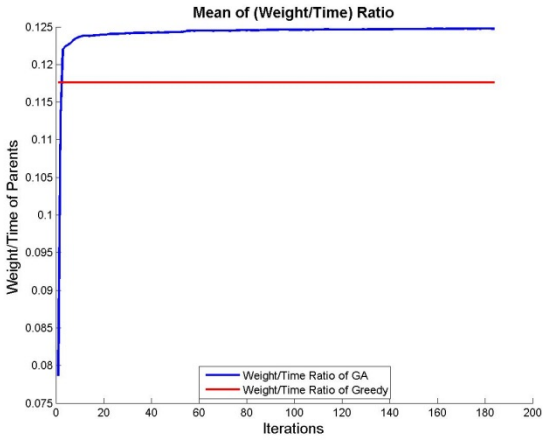
Network Size = 300, Problem Instance = 9



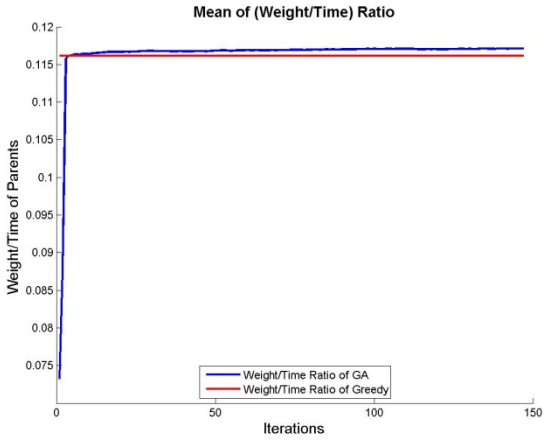
Network Size = 300, Problem Instance = 10



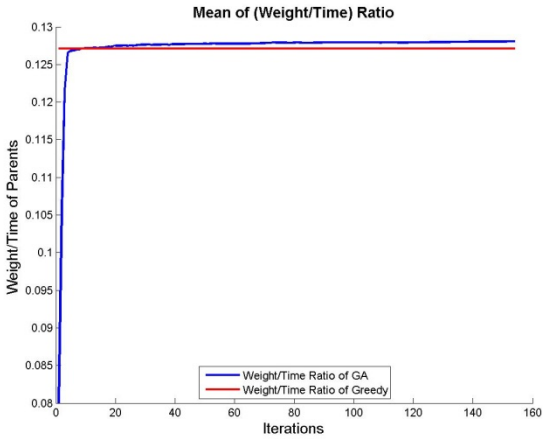
Network Size = 350, Problem Instance = 1



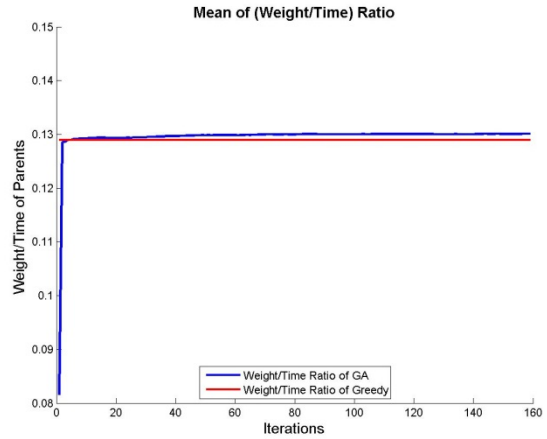
Network Size = 350, Problem Instance = 2



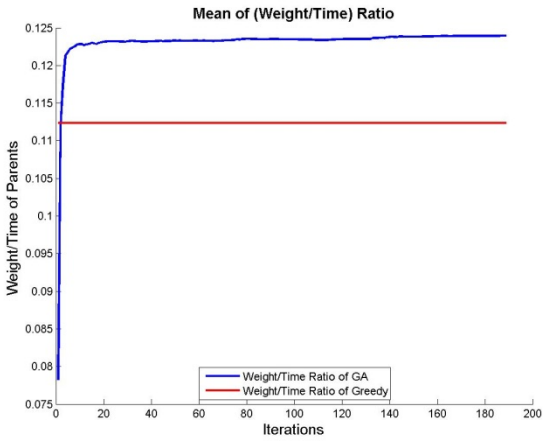
Network Size = 350, Problem Instance = 3



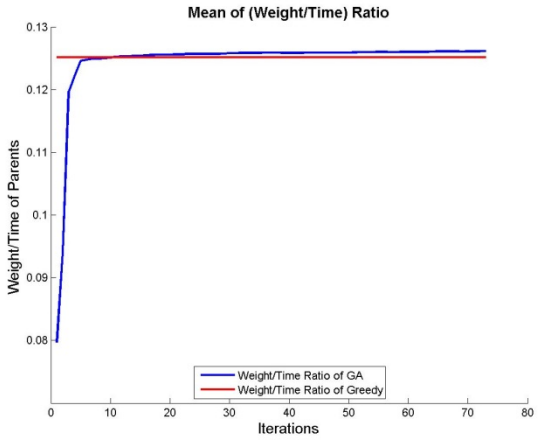
Network Size = 350, Problem Instance = 4



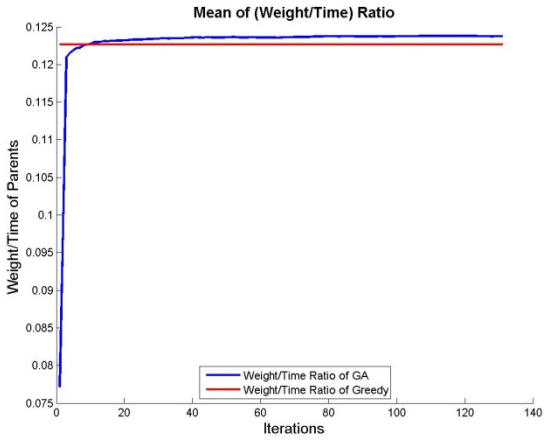
Network Size = 350, Problem Instance = 5



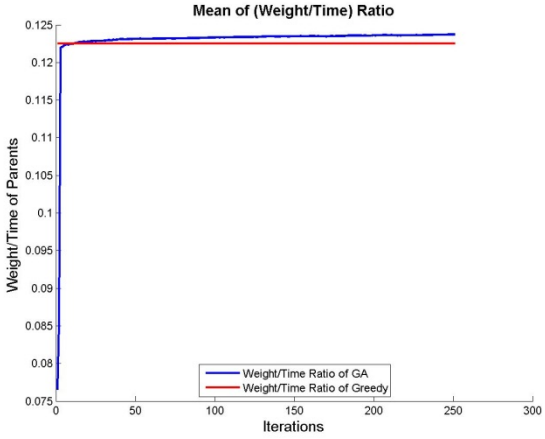
Network Size = 350, Problem Instance = 6



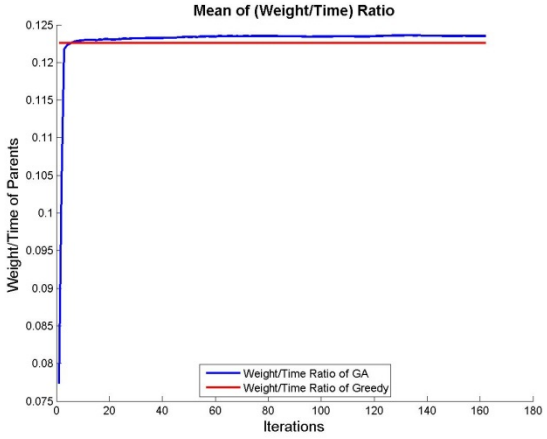
Network Size = 350, Problem Instance = 7



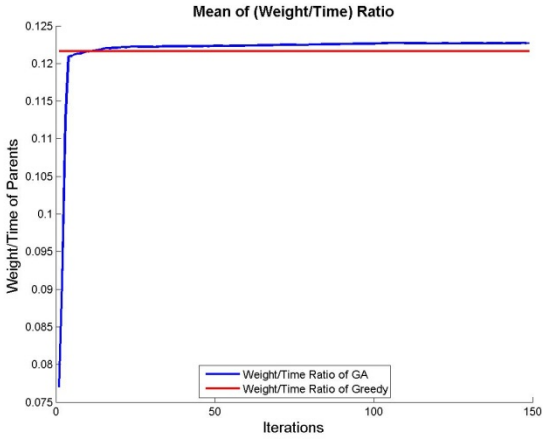
Network Size = 350, Problem Instance = 8



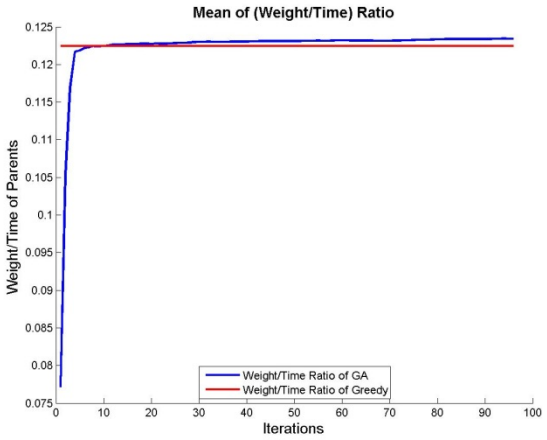
Network Size = 350, Problem Instance = 9



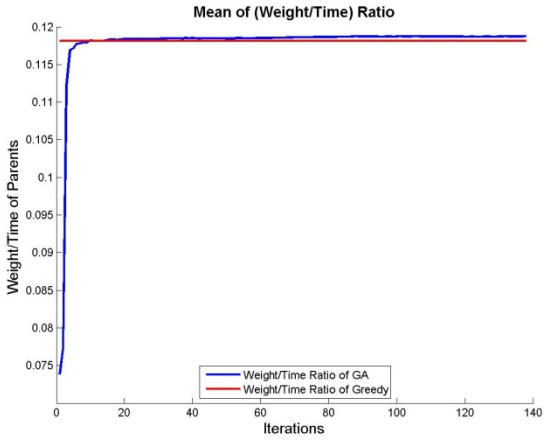
Network Size = 350, Problem Instance = 10



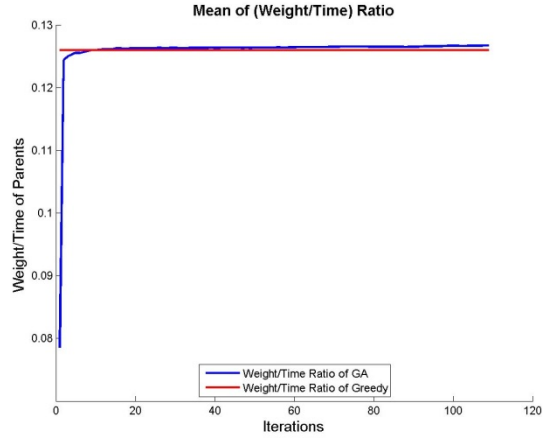
Network Size = 400, Problem Instance = 1



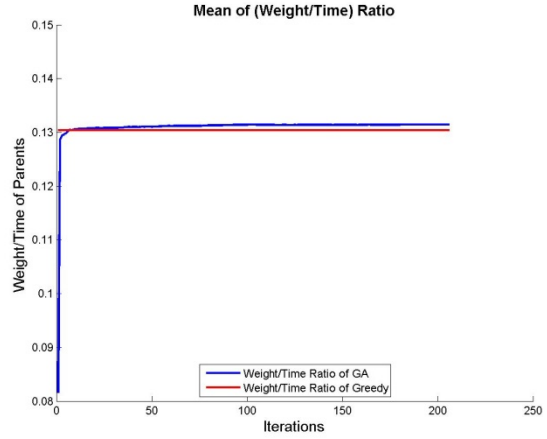
Network Size = 400, Problem Instance = 2



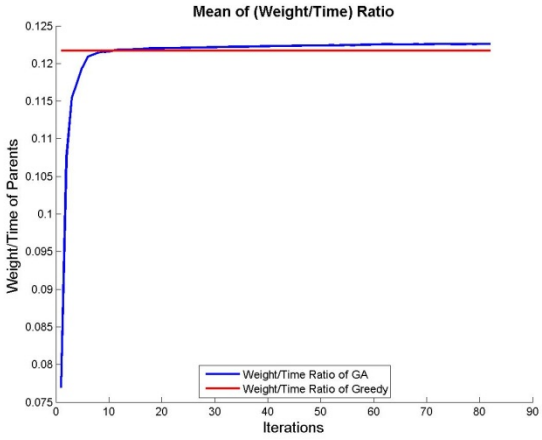
Network Size = 400, Problem Instance = 3



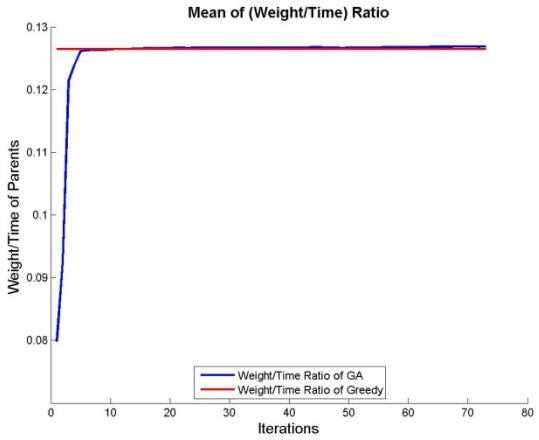
Network Size = 400, Problem Instance = 4



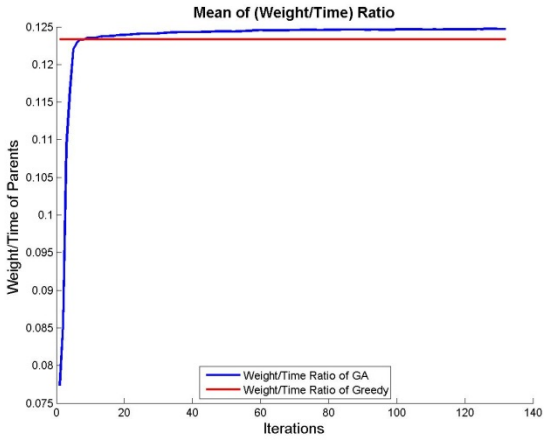
Network Size = 400, Problem Instance = 5



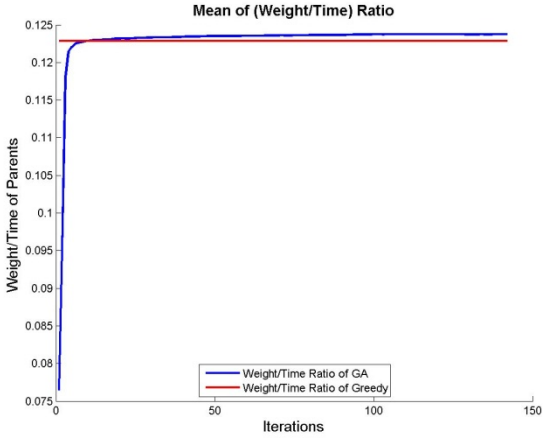
Network Size = 400, Problem Instance = 6



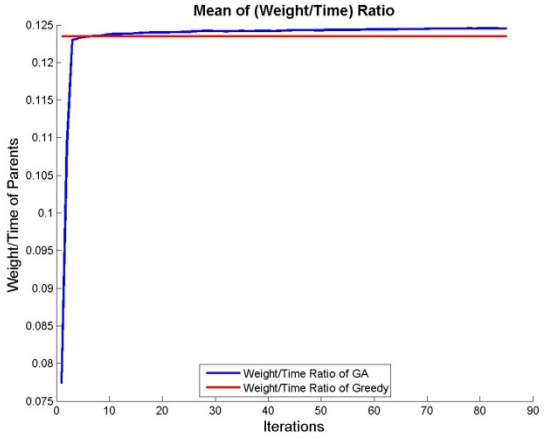
Network Size = 400, Problem Instance = 7



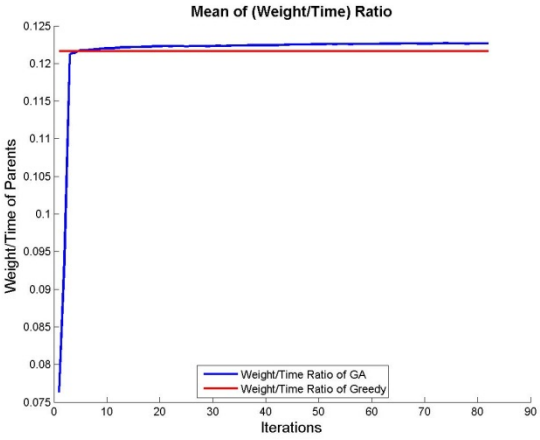
Network Size = 400, Problem Instance = 8



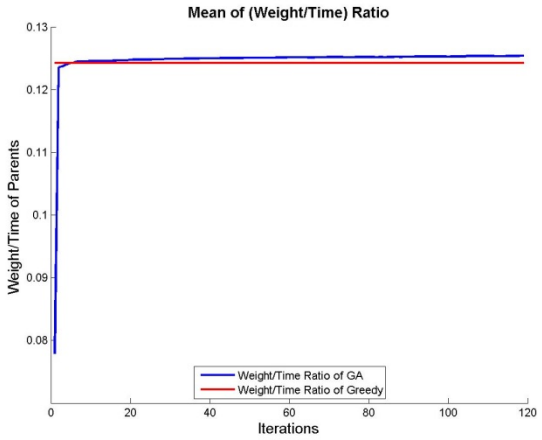
Network Size = 400, Problem Instance = 9



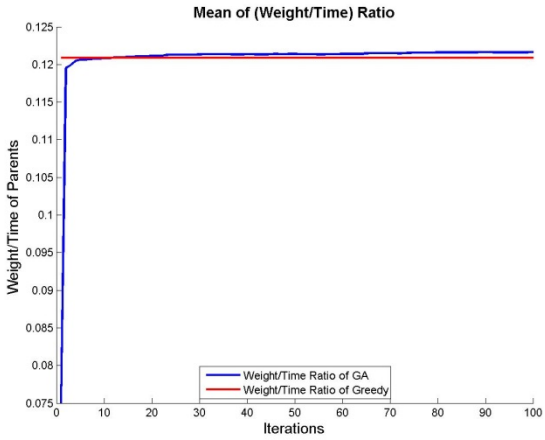
Network Size = 400, Problem Instance = 10



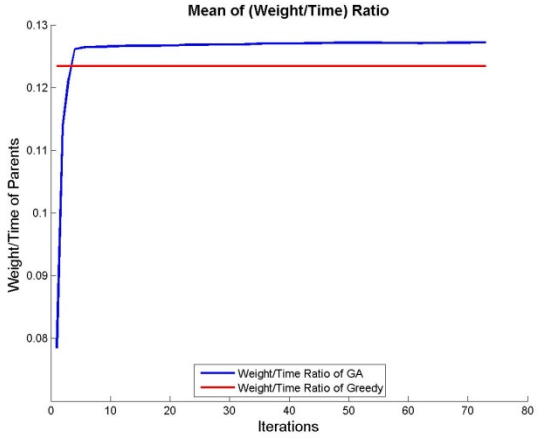
Network Size = 450, Problem Instance = 1



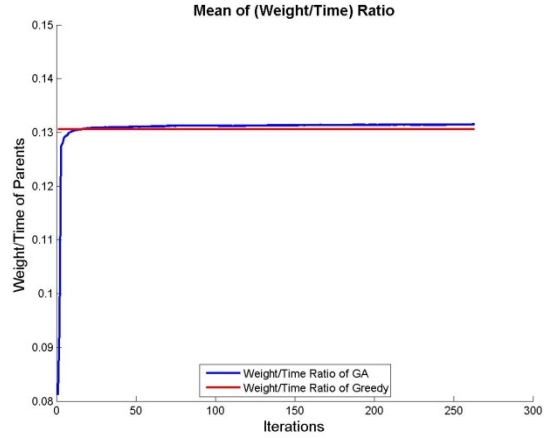
Network Size = 450, Problem Instance = 2



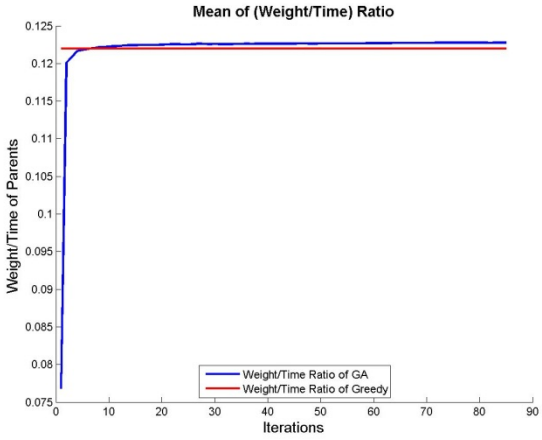
Network Size = 450, Problem Instance = 3



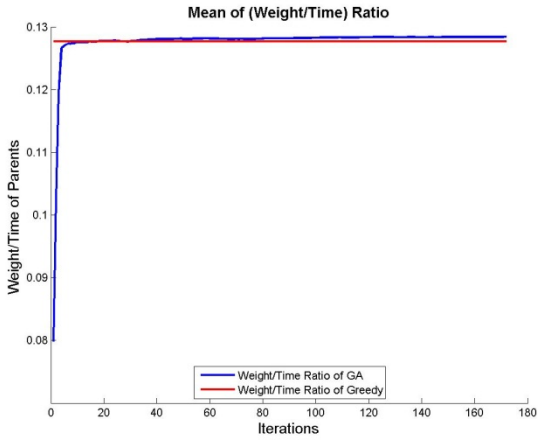
Network Size = 450, Problem Instance = 4



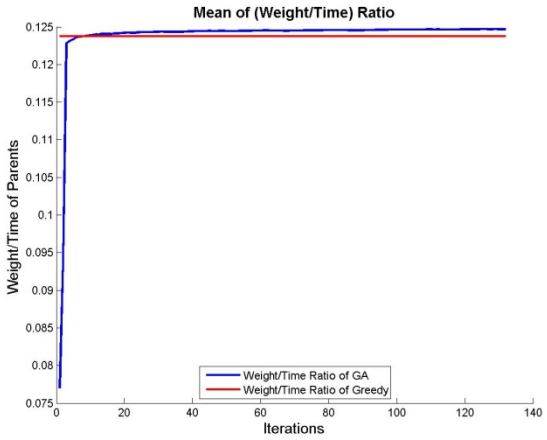
Network Size = 450, Problem Instance = 5



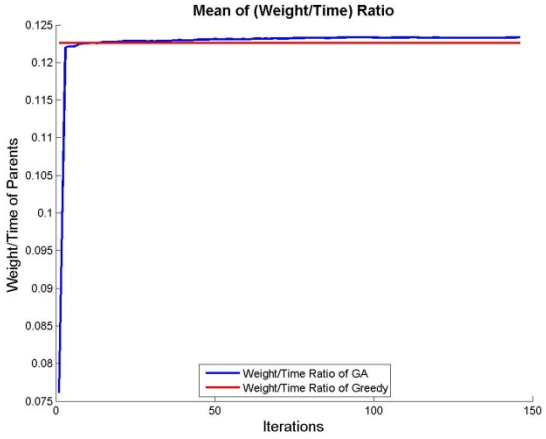
Network Size = 450, Problem Instance = 6



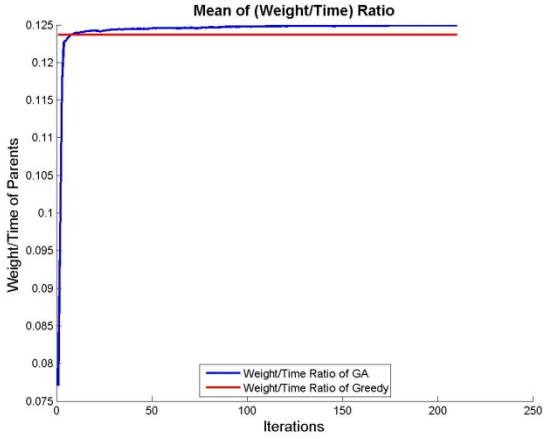
Network Size = 450, Problem Instance = 7



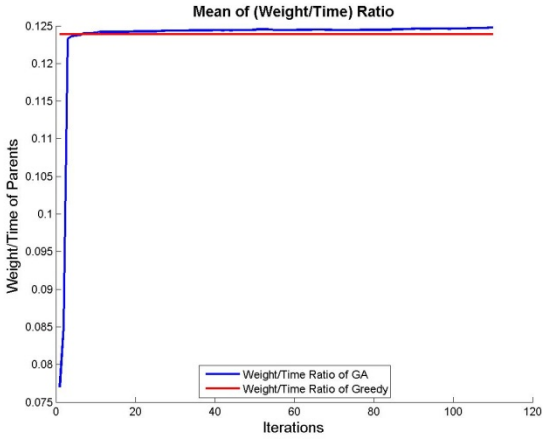
Network Size = 450, Problem Instance = 8



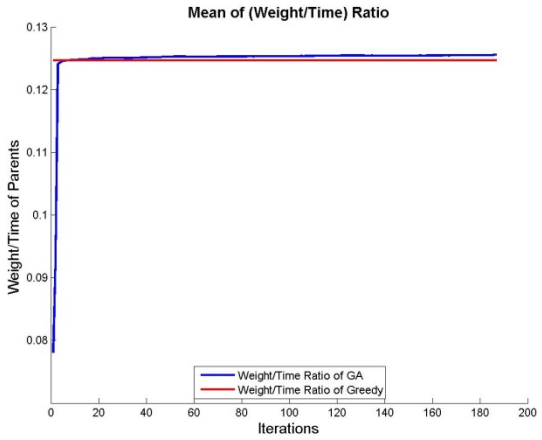
Network Size = 450, Problem Instance = 9



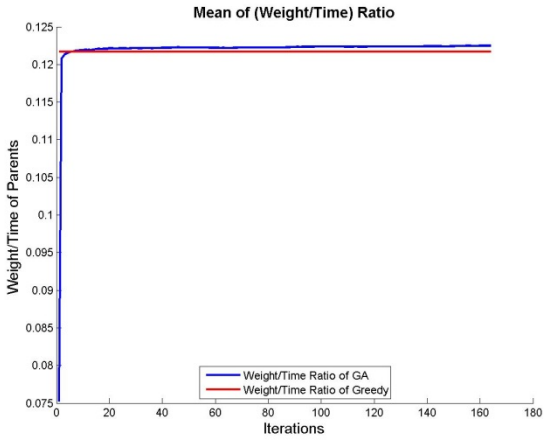
Network Size = 450, Problem Instance = 10



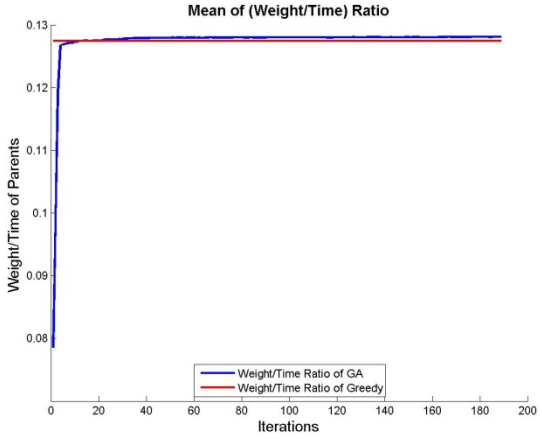
Network Size = 500, Problem Instance = 1



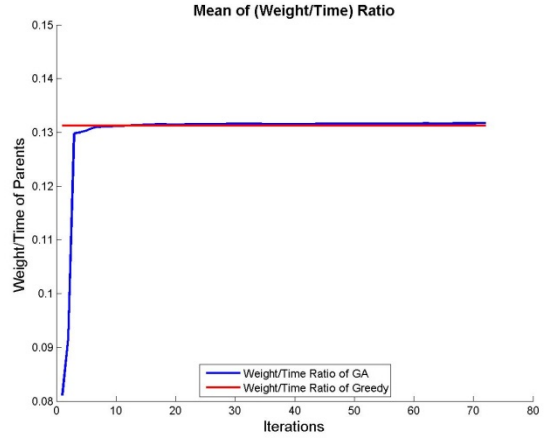
Network Size = 500, Problem Instance = 2



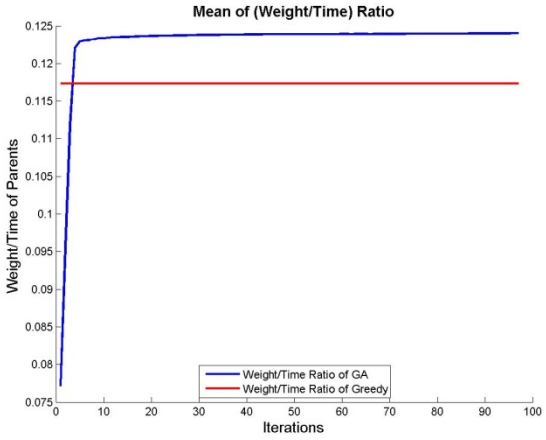
Network Size = 500, Problem Instance = 3



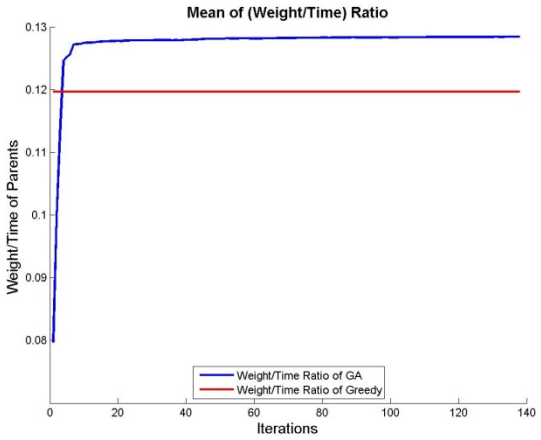
Network Size = 500, Problem Instance = 4



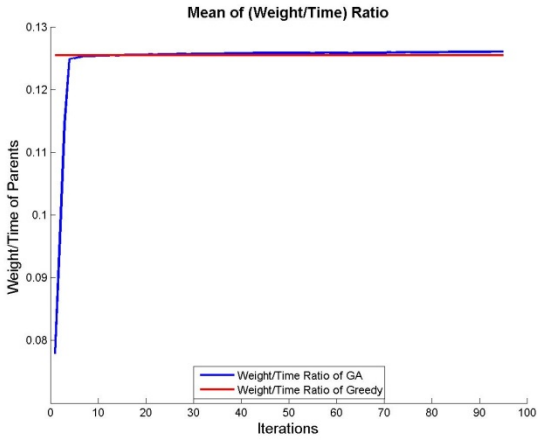
Network Size = 500, Problem Instance = 5



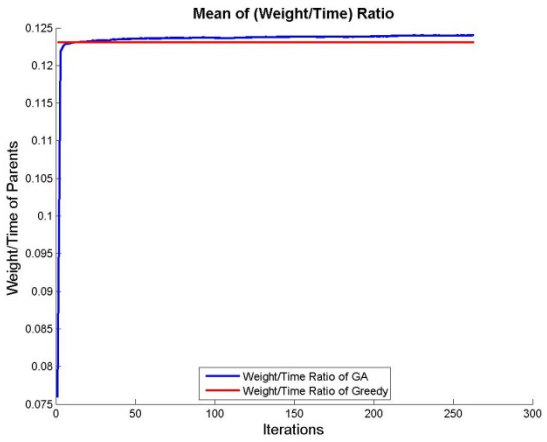
Network Size = 500, Problem Instance = 6



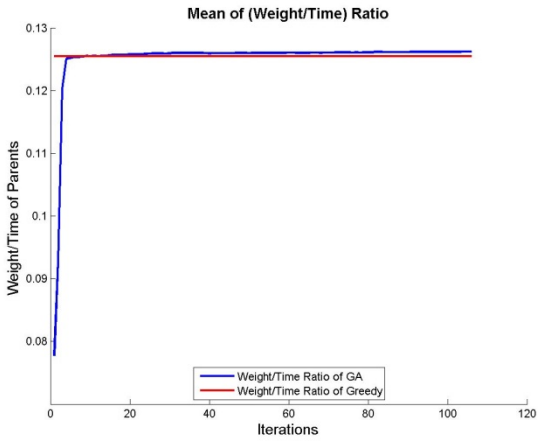
Network Size = 500, Problem Instance = 7



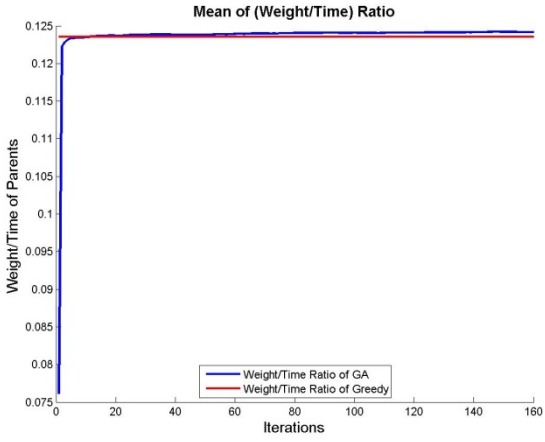
Network Size = 500, Problem Instance = 8



Network Size = 500, Problem Instance = 9



Network Size = 500, Problem Instance = 10



Appendix D: Quantitative Comparison of the Genetic and Greedy Algorithms

Table 2. Quantitative Comparison between Greedy Algorithm and GA; n=100 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	200	1	30	2.9	3,909	364	210	18,890	0	1
2	200	1	24	7.2	3,370	260	170	9,940	0	1
3	200	1	30	7.1	3,285	292	219	11,403	0	1
4	200	1	35	7.7	3,483	233	213	9,349	0	1
5	200	1	31	8.4	4,192	360	255	18,890	0	1
6	200	1	30	6.9	4,389	413	237	24,093	0	1
7	200	1	32	8.4	3,037	158	161	5,534	0	1
8	200	1	19	9.5	2,692	182	171	5,530	0	1
9	200	1	38	19.6	3,931	304	237	15,354	0	1
10	200	1	33	8.2	3,738	294	218	12,693	0	1
Average	200	1	30.2	8.6	3,602	286	209.1	13,095	0	1

Table 3. Quantitative Comparison between Greedy Algorithm and GA; n=150 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	300	1	31	5.4	2,588	187	164	7,969	0	1
2	300	1	27	10.5	3,340	210	209	12,240	0	1
3	300	1	31	10.3	3,747	282	228	18,442	0	1
4	300	1	38	9.4	3,244	147	191	8,071	0	1
5	300	1	40	12.3	3,272	224	200	12,043	0	1
6	300	1	29	10	2,772	176	173	7,854	0	1
7	300	1	30	10.7	2,301	137	120	4,981	0	1
8	300	1	29	13.4	2,172	122	128	4,133	0	1
9	300	1	30	12.5	1,957	118	108	3,498	0	1
10	300	1	27	10.6	1,843	71	127	1,870	0	1
Average	300	1	31.2	10.5	2,724	167	164	8,110	0	1

Table 4. Quantitative Comparison between Greedy Algorithm and GA; n=200 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	400	1	30	14.8	2,177	92	124	4,338	0	1
2	400	1	31	14.5	2,986	245	190	16,517	0	1
3	400	1	38	15.8	3,305	208	181	15,919	0	1
4	400	1	27	16	2,797	211	151	13,493	0	1
5	400	1	28	15	3,339	253	209	19,681	0	1
6	400	1	30	13	2,401	135	151	7,239	0	1
7	400	1	30	16	1,821	90	98	3,533	0	1
8	400	1	27	17	2,053	66	119	2,943	0	1
9	400	1	30	16.6	2,205	187	153	8,825	0	1
10	400	1	39	16.5	3,108	207	181	14,008	0	1
Average	400	1	31	15.6	2619.3	169.4	155.7	10,649	0	1

Table 5. Quantitative Comparison between Greedy Algorithm and GA; n=250 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	500	1	41	18.4	3,133	208	186	20,127	0	1
2	500	1	29	19.7	2,210	91	125	5,862	0	1
3	500	1	37	20.3	2,492	81	124	5,769	0	1
4	500	1	28	19.6	2,140	91	112	5,670	0	1
5	500	1	33	20.4	2,681	152	159	12,132	0	1
6	500	1	29	19.3	2,870	221	153	19,441	0	1
7	500	1	31	21	3,642	207	207	24,078	0	1
8	500	1	36	20.9	2,101	96	115	5,807	0	1
9	500	1	31	21	3,642	207	207	24,151	0	1
10	500	1	27	19	2,218	124	111	7,653	0	1
Average	500	1	32.2	20.2	2713.3	147.8	149.9	13,068	0	1

Table 6. Quantitative Comparison between Greedy Algorithm and GA; n=300 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	600	1	35	25.9	3,269	219	175	27,743	0	1
2	600	1	30	27.6	3,005	209	167	24,115	0	1
3	600	1	35	27.4	3,194	193	179	23,154	0	1
4	600	1	51	26.9	2,838	123	153	13,527	0	1
5	600	1	36	27.7	3,849	245	212	38,534	0	1
6	600	1	41	29.3	2,689	144	157	14,389	0	1
7	600	1	38	30	2,777	190	149	20,144	0	1
8	600	1	28	28.9	2,426	147	127	13,361	0	1
9	600	1	38	29.9	2,359	105	146	9,427	0	1
10	600	1	32	27.2	2,509	156	137	14,195	0	1
Average	600	1	36.4	28.1	2891.9	173.1	160.2	19,858	0	1

Table 7. Quantitative Comparison between Greedy Algorithm and GA; n=350 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	700	1	37	40	3,324	184	194	50,753	0	1
2	700	1	35	39	2,643	147	109	18,073	0	1
3	700	1	37	39.4	2,706	154	137	19,108	0	1
4	700	1	44	40.9	2,503	159	126	18,800	0	1
5	700	1	37	38.9	2,967	189	148	27,391	0	1
6	700	1	34	45.3	2,070	73	110	6,998	0	1
7	700	1	38	43.9	2,708	131	163	16,733	0	1
8	700	1	34	40.6	2,767	251	135	32,654	0	1
9	700	1	43	39.6	3,041	162	170	23,348	0	1
10	700	1	34	38.9	2,313	149	122	15,812	0	1
Average	700	1	37.3	40.7	2704.5	159.9	141.4	22,966	0	1

Table 8. Quantitative Comparison between Greedy Algorithm and GA; n=400 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	800	1	41	86.6	2,219	96	101	18,539	0	1
2	800	1	36	53.8	2,413	138	137	18,685	0	1
3	800	1	44	52	2,369	109	102	14,363	0	1
4	800	1	49	52.6	3,275	206	175	41,069	0	1
5	800	1	39	52	2,032	82	107	9,361	0	1
6	800	1	31	59.6	3,009	195	179	34,922	0	1
7	800	1	39	53.9	2,246	132	121	16,919	0	1
8	800	1	48	54.2	2,644	142	133	21,878	0	1
9	800	1	45	54	2,264	85	116	11,001	0	1
10	800	1	47	53.2	2,304	82	133	10,689	0	1
Average	800	1	41.9	57.2	2477.5	126.7	130.4	19,742	0	1

Table 9. Quantitative Comparison between Greedy Algorithm and GA; n=450 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	900	1	39	113.2	2,575	119	128	33,491	0	1
2	900	1	52	67.2	2,147	100	100	14,417	0	1
3	900	1	32	67.3	2,166	73	115	10,767	0	1
4	900	1	51	71.5	3,726	263	188	71,134	0	1
5	900	1	43	69.3	2,726	85	147	15,971	0	1
6	900	1	48	70.5	2,617	172	142	30,709	0	1
7	900	1	36	69.4	2,709	132	145	24,571	0	1
8	900	1	33	74.4	2,638	146	152	26,954	0	1
9	900	1	43	68.5	2,979	210	151	42,685	0	1
10	900	1	33	68.7	2,253	110	104	16,602	0	1
Average	900	1	41	74	2653.8	141	137.2	28,730	0	1

Table 10. Quantitative Comparison between Greedy Algorithm and GA; n=500 Tracks

	$Greedy$				GA					
Instances	$\|Pop\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|Pop^{ave}\|$	$\|Iter\|$	$\|PF\|$	CPU	$\|PF^{GR}\| \geq \|PF^{GA}\|$	$\|PF^{GR}\| < \|PF^{GA}\|$
1	1000	1	41	92.3	2,878	187	143	43,576	0	1
2	1000	1	48	89.9	3,053	164	161	40,740	0	1
3	1000	1	32	87.7	2,664	189	124	40,264	0	1
4	1000	1	50	91.9	2,437	72	109	14,489	0	1
5	1000	1	31	90	2,285	97	123	18,163	0	1
6	1000	1	37	91	2,283	138	115	25,692	0	1
7	1000	1	37	91.4	2,325	95	117	17,879	0	1
8	1000	1	44	95.7	3,368	263	174	74,069	0	1
9	1000	1	33	94.7	2,365	106	102	20,261	0	1
10	1000	1	36	90.8	2,832	160	133	36,669	0	1
Average	1000	1	38.9	91.6	2649.3	147.1	130.1	33,180	0	1

Appendix E: Qualitative Comparison of the Genetic and Greedy Algorithms

Table 12. Qualitative Comparison between Greedy Algorithm and GA; n=100 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.0973	0.1112	0	1	210	0	1	0	1	0
2	0.1002	0.1041	0	1	170	0	1	0	1	0
3	0.1187	0.1301	0	1	219	0	1	0	1	0
4	0.1233	0.1270	0	1	213	0	1	0	1	0
5	0.1009	0.1128	0	1	255	0	1	0	1	0
6	0.0967	0.1092	0	1	237	0	1	0	1	0
7	0.1047	0.1107	0	1	161	0	1	0	1	0
8	0.1072	0.1111	0	1	171	0	1	0	1	0
9	0.0992	0.1116	0	1	237	0	1	0	1	0
10	0.1168	0.1212	0	1	218	0	1	0	1	0
Average	0.1065	0.1149	0	1	209	0	1	0	1	0

Table 13. Qualitative Comparison between Greedy Algorithm and GA; n=150 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1023	0.1172	0	1	164	0	1	0	1	0
2	0.1009	0.1049	0	1	209	0	1	0	1	0
3	0.1123	0.1263	0	1	228	0	1	0	1	0
4	0.1151	0.1239	0	1	191	0	1	0	1	0
5	0.1044	0.1139	0	1	200	0	1	0	1	0
6	0.1045	0.1155	0	1	173	0	1	0	1	0
7	0.1049	0.1143	0	1	120	0	1	0	1	0
8	0.1112	0.1135	0	1	128	0	1	0	1	0
9	0.1120	0.1141	0	1	108	0	1	0	1	0
10	0.1180	0.1196	0	1	127	0	1	0	1	0
Average	0.1086	0.1163	0	1	164	0	1	0	1	0

Table 14. Qualitative Comparison between Greedy Algorithm and GA; n=200 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1178	0.1197	0	1	124	0	1	0	1	0
2	0.1125	0.1144	0	1	190	0	1	0	1	0
3	0.1186	0.1328	0	1	181	0	1	0	1	0
4	0.1233	0.1255	0	1	151	0	1	0	1	0
5	0.1116	0.1207	0	1	209	0	1	0	1	0
6	0.1148	0.1202	0	1	151	0	1	0	1	0
7	0.1162	0.1176	0	1	98	0	1	0	1	0
8	0.1144	0.1156	0	1	119	0	1	0	1	0
9	0.1162	0.1176	0	1	153	0	1	0	1	0
10	0.1191	0.1230	0	1	181	0	1	0	1	0
Average	0.1164	0.1207	0	1	155	0	1	0	1	0

Table 15. Qualitative Comparison between Greedy Algorithm and GA; n=250 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1104	0.1181	0	1	186	0	1	0	1	0
2	0.1134	0.1142	0	1	125	0	1	0	1	0
3	0.1165	0.1290	0	1	124	0	1	0	1	0
4	0.1276	0.1285	0	1	112	0	1	0	1	0
5	0.1191	0.1206	0	1	159	0	1	0	1	0
6	0.1187	0.1247	0	1	153	0	1	0	1	0
7	0.1166	0.1183	0	1	207	0	1	0	1	0
8	0.1206	0.1217	0	1	115	0	1	0	1	0
9	0.1166	0.1183	0	1	207	0	1	0	1	0
10	0.1248	0.1264	0	1	111	0	1	0	1	0
Average	0.1184	0.1220	0	1	149	0	1	0	1	0

Table 16. Qualitative Comparison between Greedy Algorithm and GA; n=300 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1141	0.1246	0	1	175	0	1	0	1	0
2	0.1131	0.1146	0	1	167	0	1	0	1	0
3	0.1151	0.1281	0	1	179	0	1	0	1	0
4	0.1280	0.1293	0	1	153	0	1	0	1	0
5	0.1150	0.1235	0	1	212	0	1	0	1	0
6	0.1254	0.1266	0	1	157	0	1	0	1	0
7	0.1199	0.1209	0	1	149	0	1	0	1	0
8	0.1208	0.1217	0	1	127	0	1	0	1	0
9	0.1199	0.1207	0	1	146	0	1	0	1	0
10	0.1237	0.1252	0	1	137	0	1	0	1	0
Average	0.1195	0.1235	0	1	160	0	1	0	1	0

Table 17. Qualitative Comparison between Greedy Algorithm and GA; n=350 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1176	0.1248	0	1	194	0	1	0	1	0
2	0.1162	0.1171	0	1	109	0	1	0	1	0
3	0.1271	0.1281	0	1	137	0	1	0	1	0
4	0.1290	0.1301	0	1	126	0	1	0	1	0
5	0.1123	0.1240	0	1	148	0	1	0	1	0
6	0.1252	0.1261	0	1	110	0	1	0	1	0
7	0.1227	0.1237	0	1	163	0	1	0	1	0
8	0.1225	0.1237	0	1	135	0	1	0	1	0
9	0.1226	0.1235	0	1	170	0.02	99.41	0	0	1
10	0.1216	0.1227	0	1	122	0	1	0	1	0
Average	0.1217	0.1244	0	1	141.4	0	1	0	1	0

Table 18. Qualitative Comparison between Greedy Algorithm and GA; n=400 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1225	0.1234	0	1	101	0	1	0	1	0
2	0.1181	0.1187	0	1	136	0.028	98.54	0	0	1
3	0.1260	0.1267	0	1	102	0	1	0	1	0
4	0.1304	0.1314	0	1	175	0	1	0	1	0
5	0.1217	0.1226	0	1	107	0	1	0	1	0
6	0.1217	0.1276	0	1	179	0	1	0	1	0
7	0.1234	0.1247	0	1	121	0	1	0	1	0
8	0.1229	0.1238	0	1	133	0	1	0	1	0
9	0.1235	0.1245	0	1	116	0	1	0	1	0
10	0.1216	0.1227	0	1	133	0	1	0	1	0
Average	0.1232	0.1246	0	1	130.3	0	1	0	1	0

Table 19. Qualitative Comparison between Greedy Algorithm and GA; n=450 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1243	0.1254	0	1	128	0	1	0	1	0
2	0.1209	0.1216	0	1	101	0.019	1	0	0	1
3	0.1234	0.1272	0	1	115	0	1	0	1	0
4	0.1306	0.1315	0	1	188	0	1	0	1	0
5	0.1220	0.1228	0	1	147	0	1	0	1	0
6	0.1277	0.1284	0	1	142	0	1	0	1	0
7	0.1237	0.1247	0	1	145	0	1	0	1	0
8	0.1226	0.1233	0	1	152	0	1	0	1	0
9	0.1237	0.1249	0	1	151	0	1	0	1	0
10	0.1239	0.1247	0	1	104	0	1	0	1	0
Average	0.1243	0.1254	0	1	137.3	0	1	0	1	0

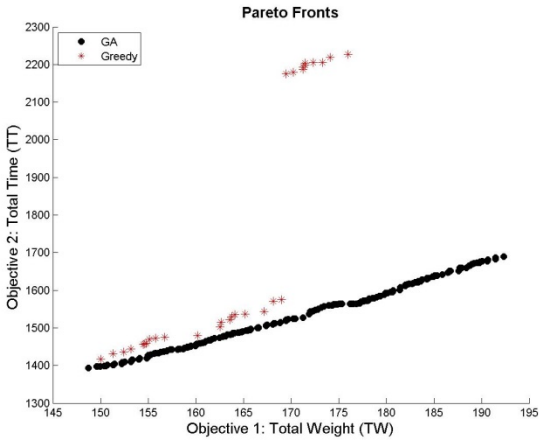
Table 20. Qualitative Comparison between Greedy Algorithm and GA; n=500 Tracks

Instances	$\frac{TW^{GR}}{TT}$	$\frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} \geq \frac{TW^{GA}}{TT}$	$\frac{TW^{GR}}{TT} < \frac{TW^{GA}}{TT}$	$\lvert PF^{U} \rvert$	GR % in $\lvert PF^{U} \rvert$	GA % in $\lvert PF^{U} \rvert$	$PF^{GR} \gg PF^{GA}$	$PF^{GR} \ll PF^{GA}$	$PF^{GR} \sim PF^{GA}$
1	0.1246	0.1255	0	1	143	0	1	0	1	0
2	0.1217	0.1225	0	1	161	0	1	0	1	0
3	0.1274	0.1281	0	1	124	0	1	0	1	0
4	0.1312	0.1317	0	1	109	0	1	0	1	0
5	0.1173	0.1240	0	1	123	0	1	0	1	0
6	0.1197	0.1284	0	1	115	0	1	0	1	0
7	0.1254	0.1260	0	1	118	0.054	99.15	0	0	1
8	0.1230	0.1240	0	1	174	0	1	0	1	0
9	0.1254	0.1262	0	1	102	0	1	0	1	0
10	0.1235	0.1242	0	1	133	0	1	0	1	0
Average	0.1239	0.1261	0	1	130	0	1	0	1	0

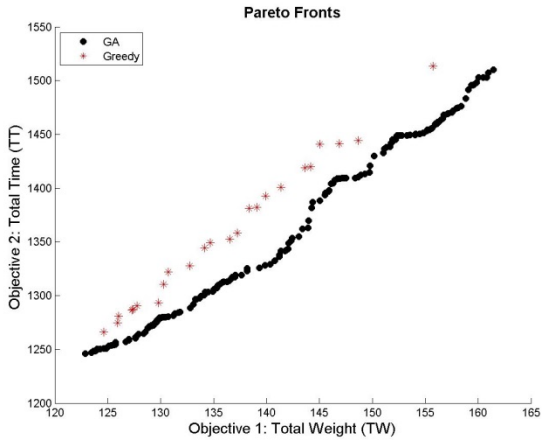
Appendix F: Comparison of the Pareto Fronts of the Genetic and Greedy Algorithms

Figure 7: Pareto Fronts of the Genetic and Greedy Algorithms

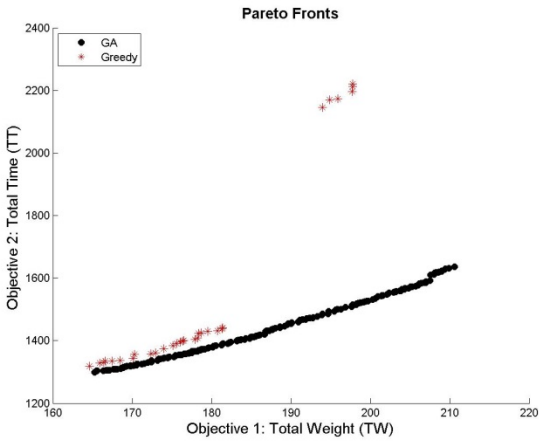
Network Size = 100, Problem Instance = 1



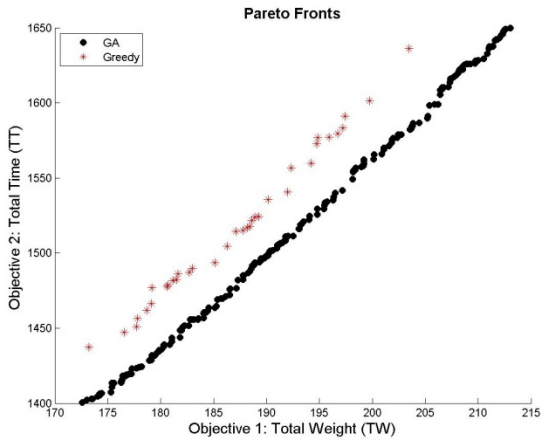
Network Size = 100, Problem Instance = 2



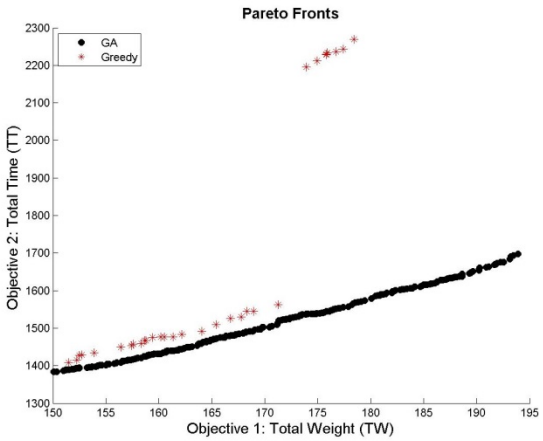
Network Size = 100, Problem Instance = 3



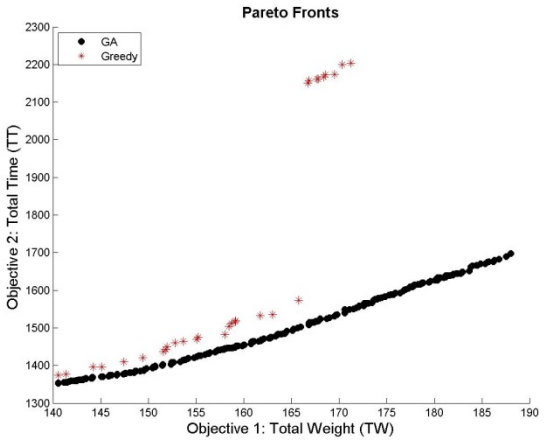
Network Size = 100, Problem Instance = 4



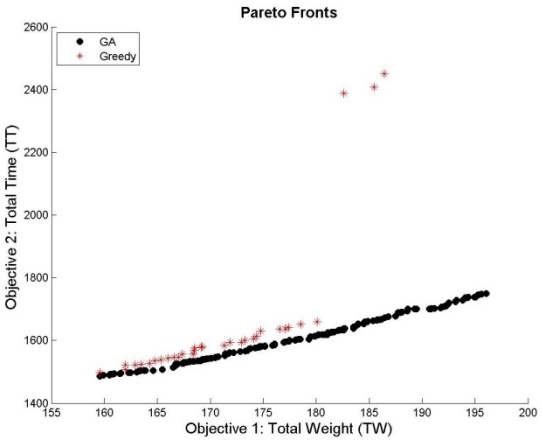
Network Size = 100, Problem Instance = 5



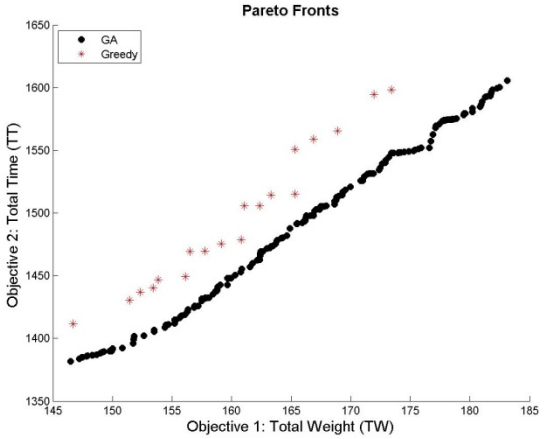
Network Size = 100, Problem Instance = 6



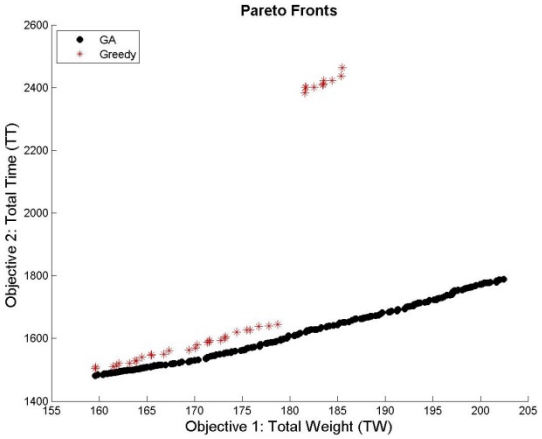
Network Size = 100, Problem Instance = 7



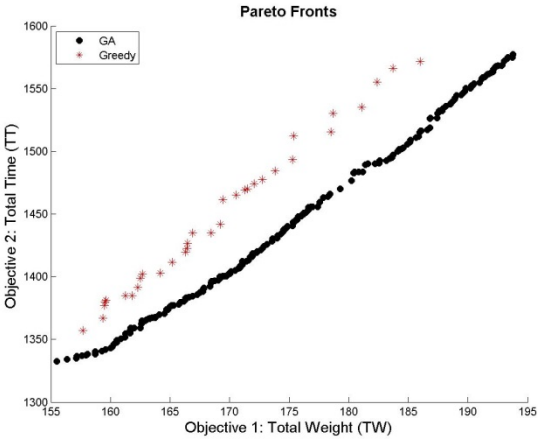
Network Size = 100, Problem Instance = 8



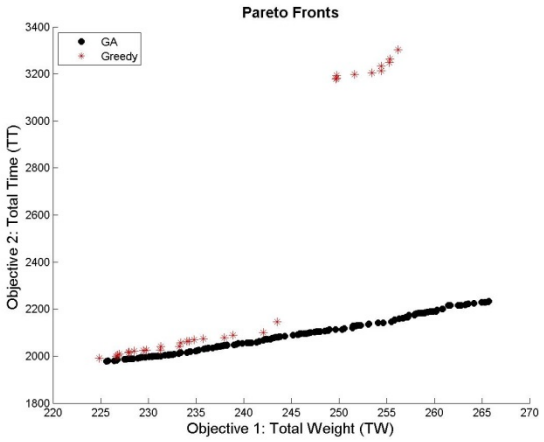
Network Size = 100, Problem Instance = 9



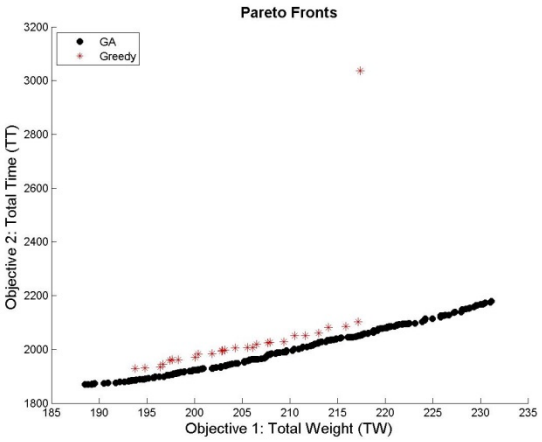
Network Size = 100, Problem Instance = 10



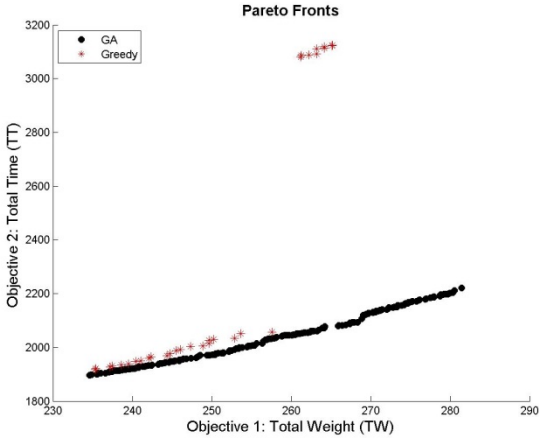
Network Size = 150, Problem Instance = 1



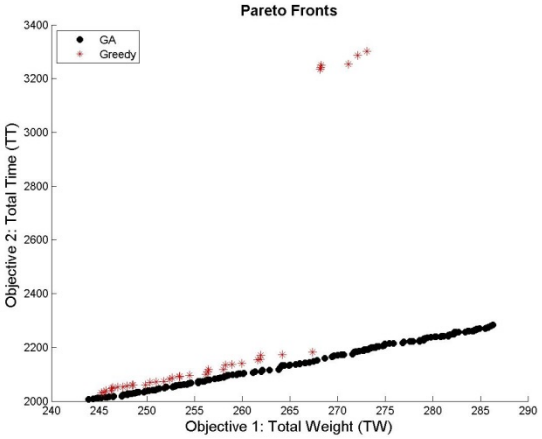
Network Size = 150, Problem Instance = 2



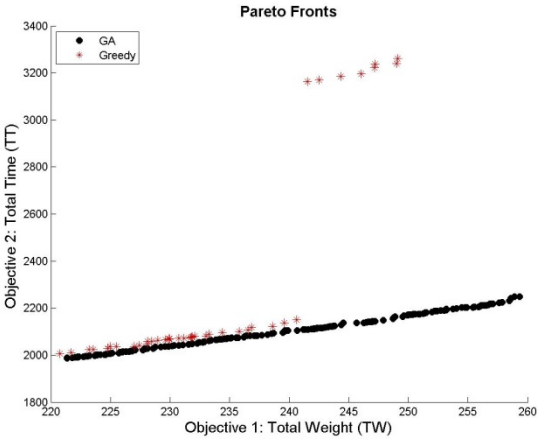
Network Size = 150, Problem Instance = 3



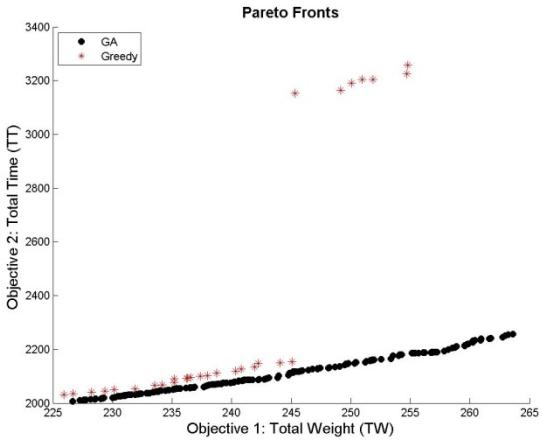
Network Size = 150, Problem Instance = 4



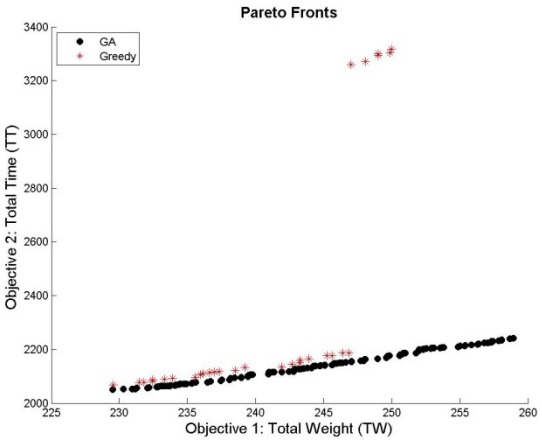
Network Size = 150, Problem Instance = 5



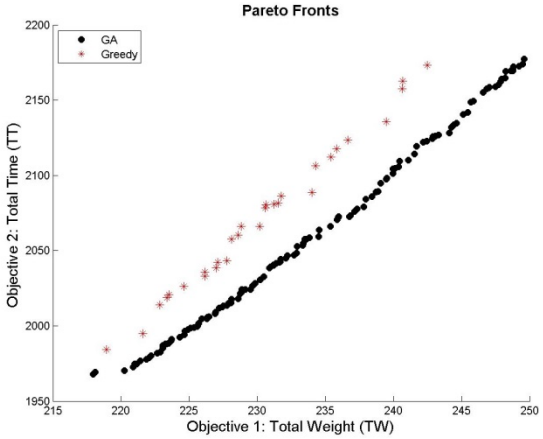
Network Size = 150, Problem Instance = 6



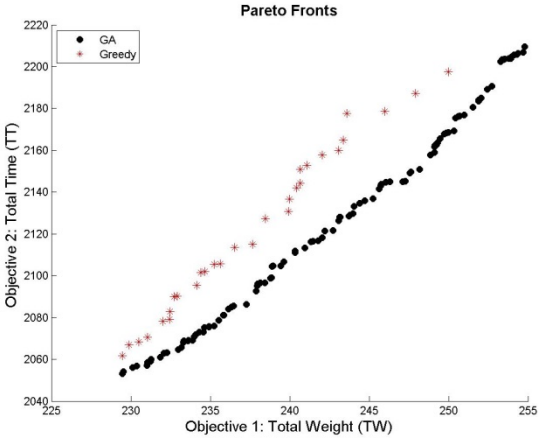
Network Size = 150, Problem Instance = 7



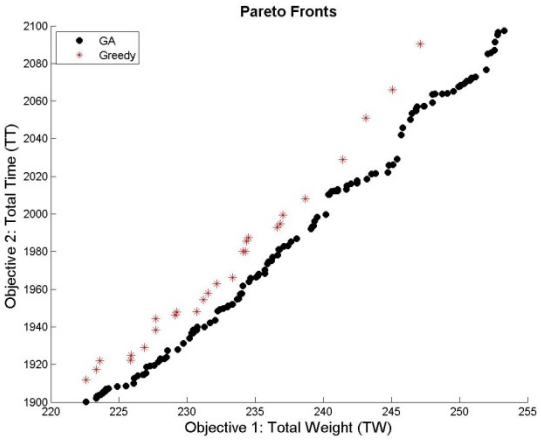
Network Size = 150, Problem Instance = 8



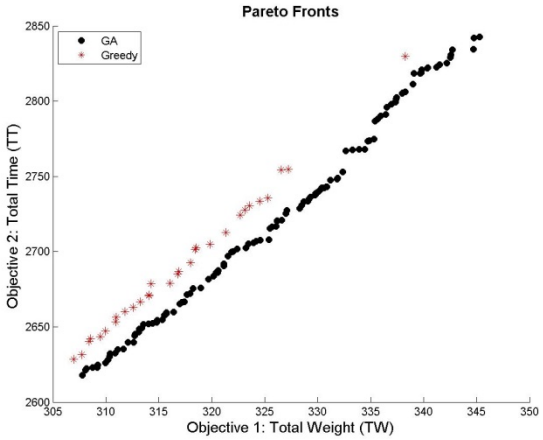
Network Size = 150, Problem Instance = 9



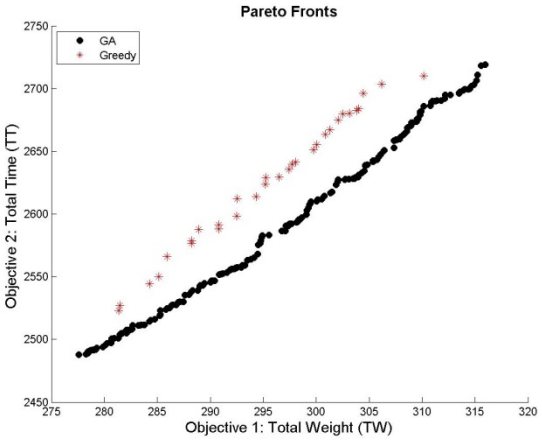
Network Size = 150, Problem Instance = 10



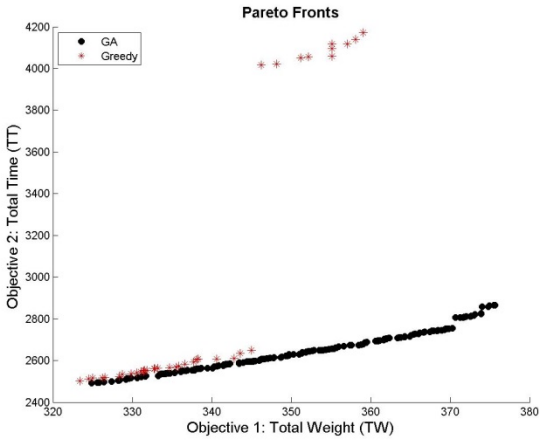
Network Size = 200, Problem Instance = 1



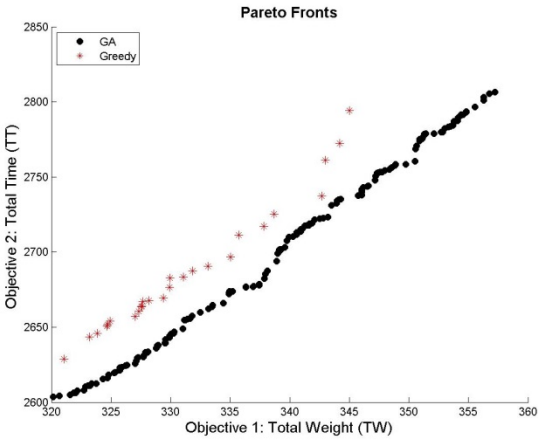
Network Size = 200, Problem Instance = 2



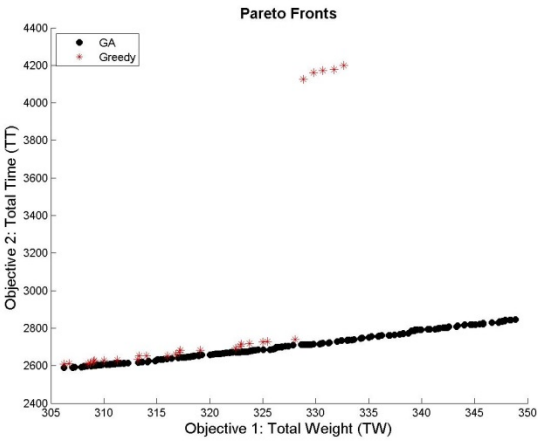
Network Size = 200, Problem Instance = 3



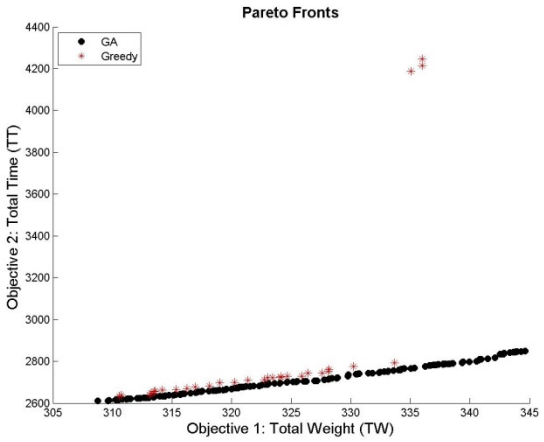
Network Size = 200, Problem Instance = 4



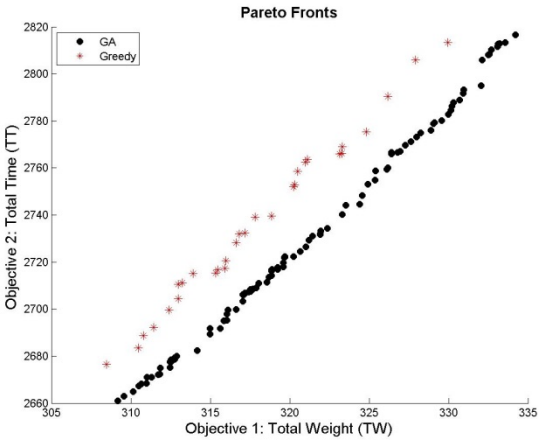
Network Size = 200, Problem Instance = 5



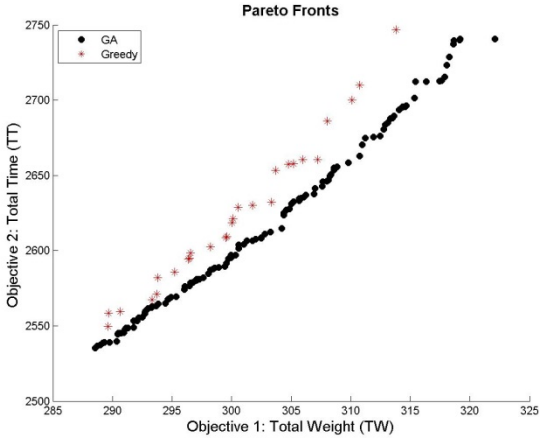
Network Size = 200, Problem Instance = 6



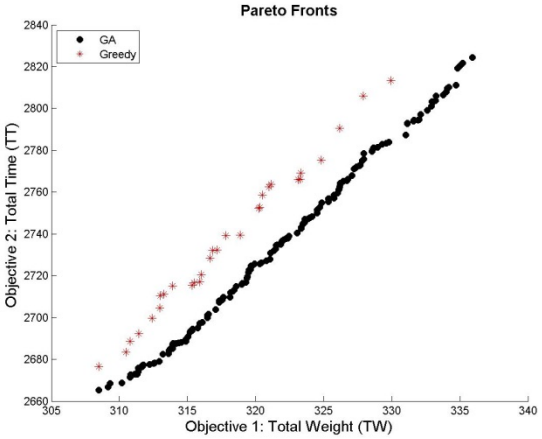
Network Size = 200, Problem Instance = 7



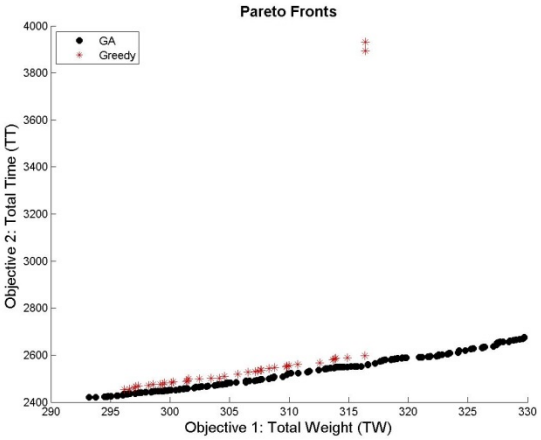
Network Size = 200, Problem Instance = 8



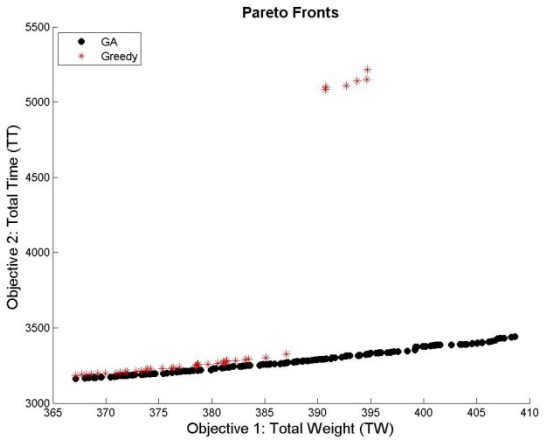
Network Size = 200, Problem Instance = 9



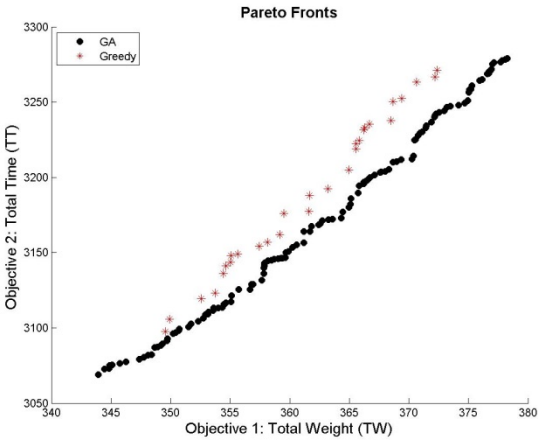
Network Size = 200, Problem Instance = 10



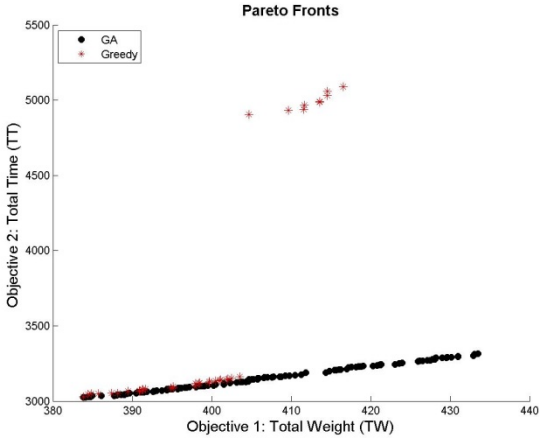
Network Size = 250, Problem Instance = 1



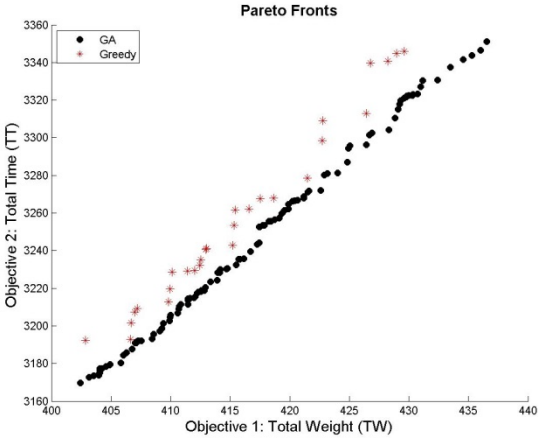
Network Size = 250, Problem Instance = 2



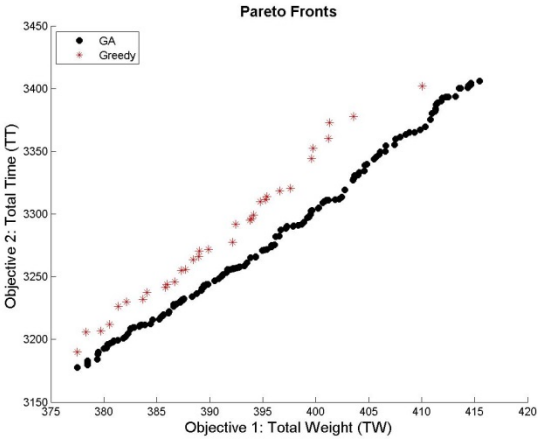
Network Size = 250, Problem Instance = 3



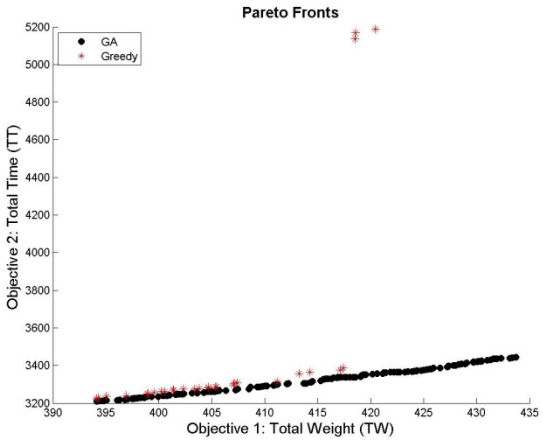
Network Size = 250, Problem Instance = 4



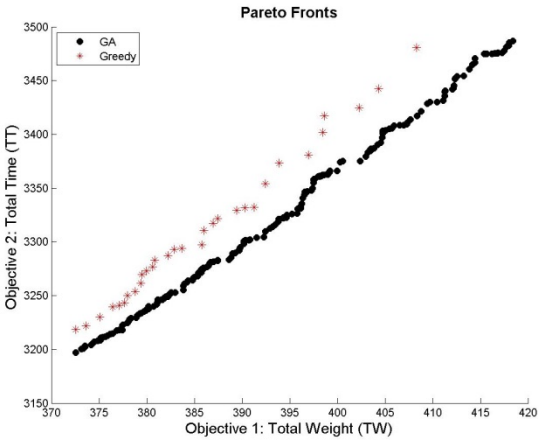
Network Size = 250, Problem Instance = 5



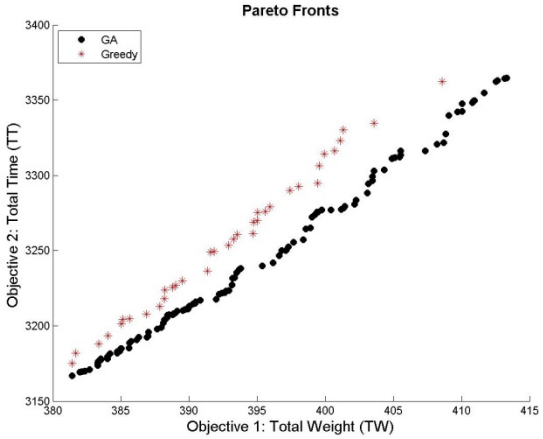
Network Size = 250, Problem Instance = 6



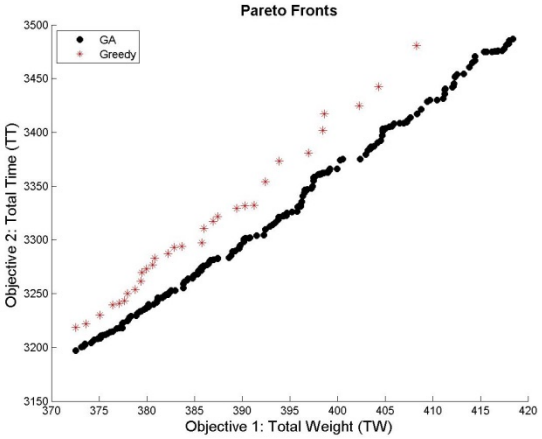
Network Size = 250, Problem Instance = 7



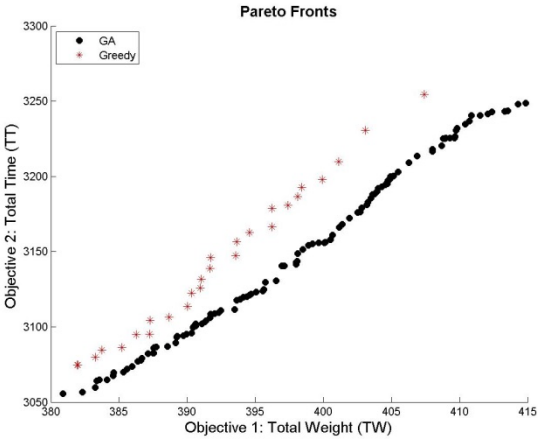
Network Size = 250, Problem Instance = 8



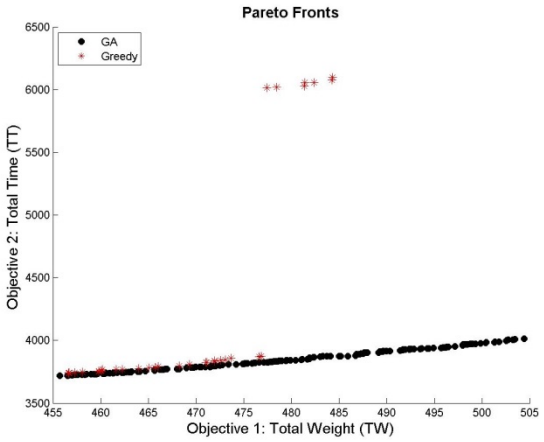
Network Size = 250, Problem Instance = 9



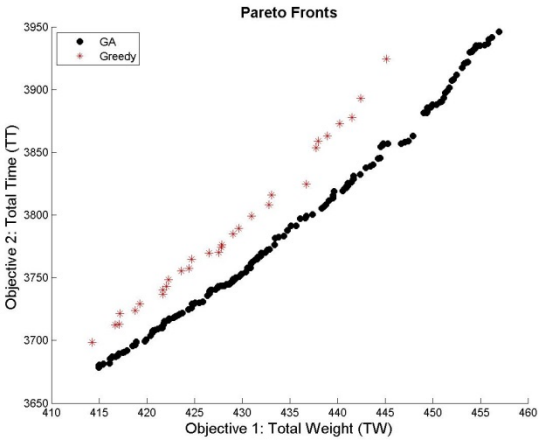
Network Size = 250, Problem Instance = 10



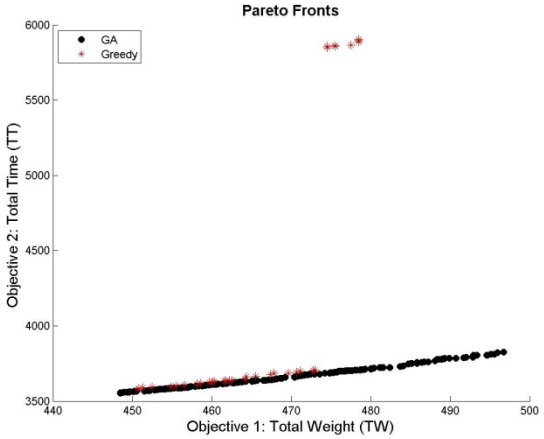
Network Size = 300, Problem Instance = 1



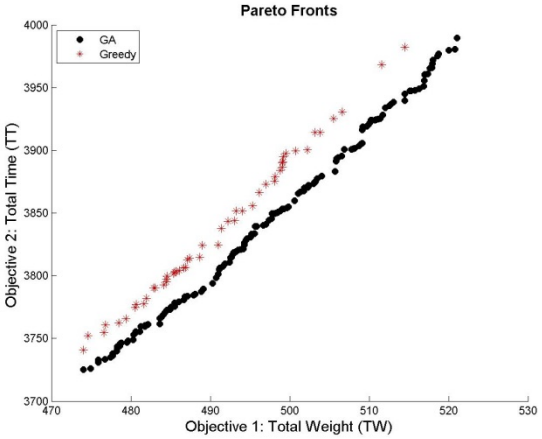
Network Size = 300, Problem Instance = 2



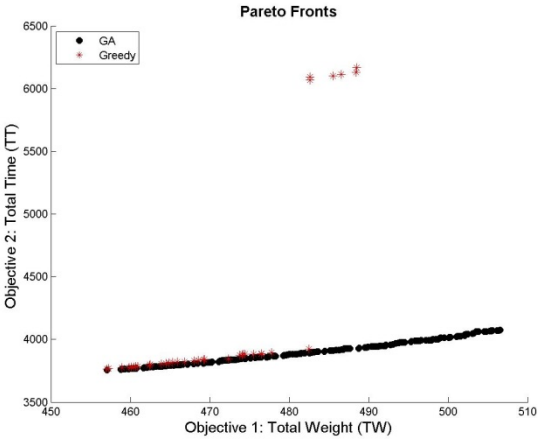
Network Size = 300, Problem Instance = 3



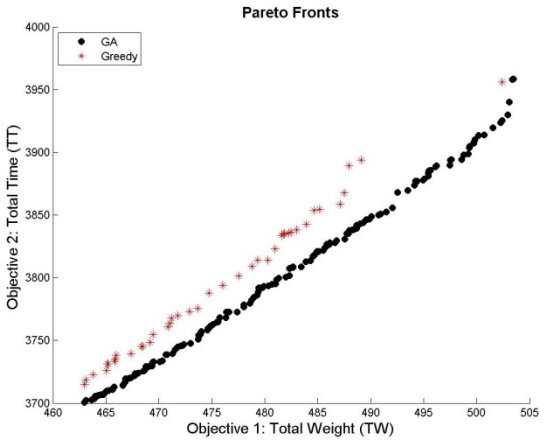
Network Size = 300, Problem Instance = 4



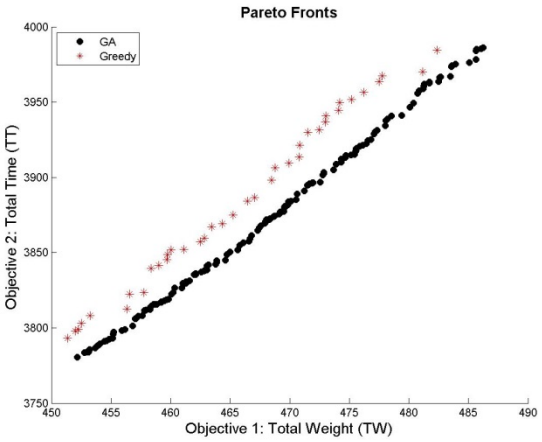
Network Size = 300, Problem Instance = 5



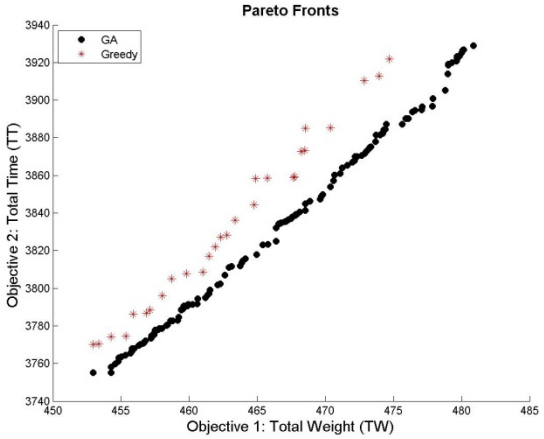
Network Size = 300, Problem Instance = 6



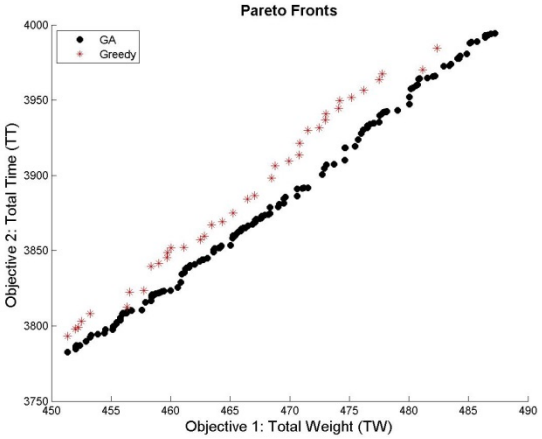
Network Size = 300, Problem Instance = 7



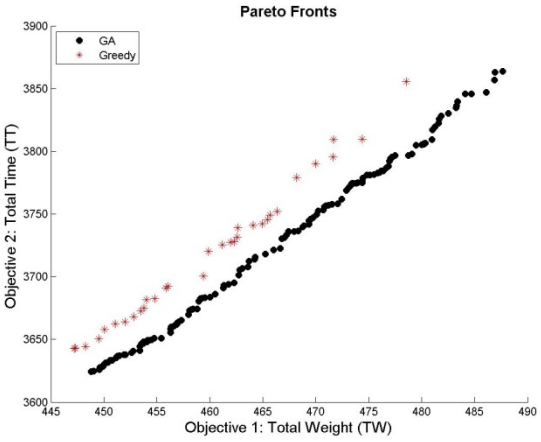
Network Size = 300, Problem Instance = 8



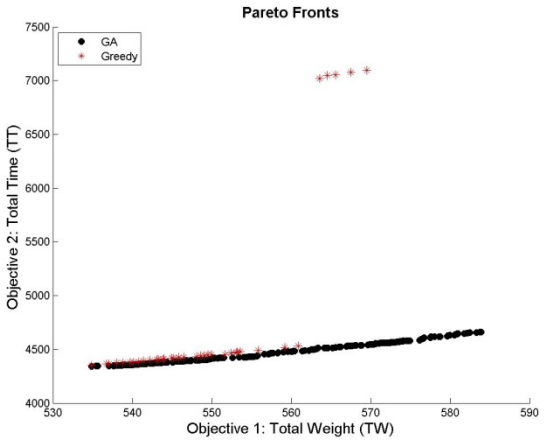
Network Size = 300, Problem Instance = 9



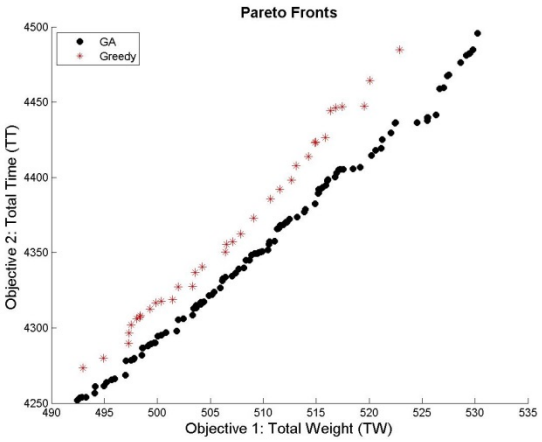
Network Size = 300, Problem Instance = 10



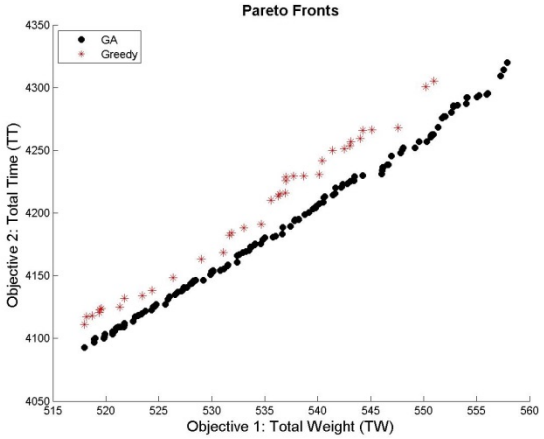
Network Size = 350, Problem Instance = 1



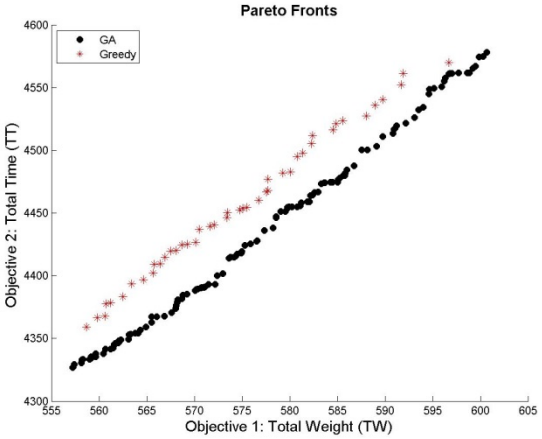
Network Size = 350, Problem Instance = 2



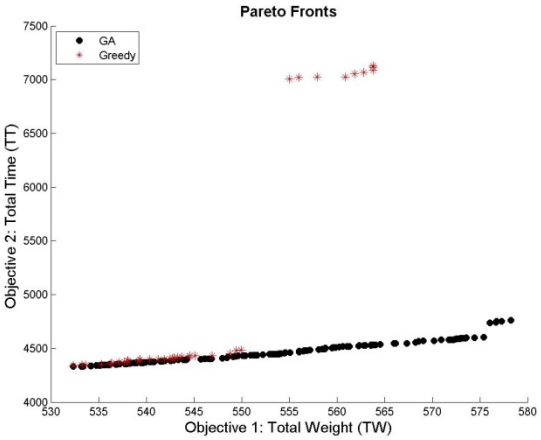
Network Size = 350, Problem Instance = 3



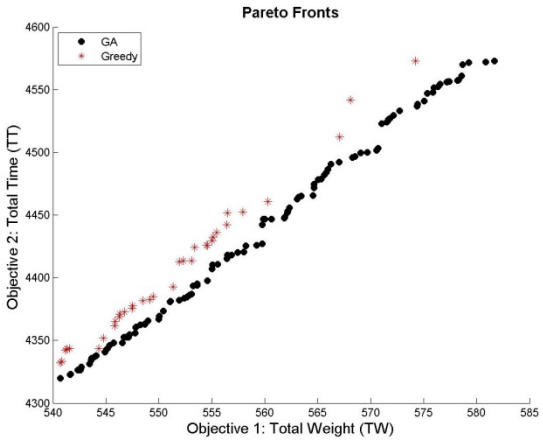
Network Size = 350, Problem Instance = 4



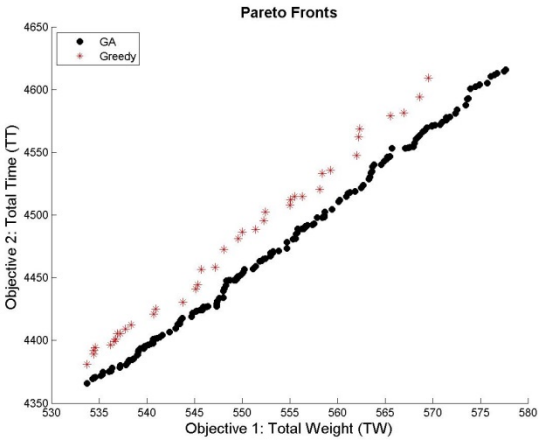
Network Size = 350, Problem Instance = 5



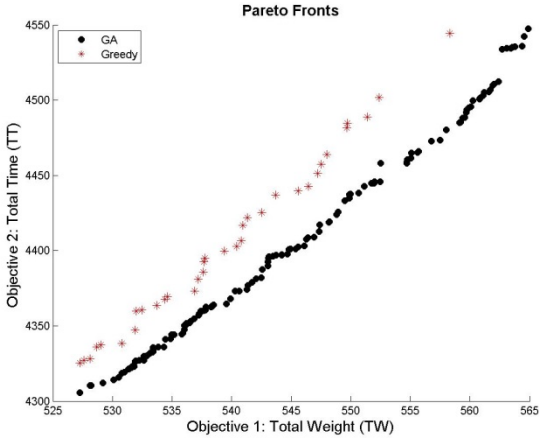
Network Size = 350, Problem Instance = 6



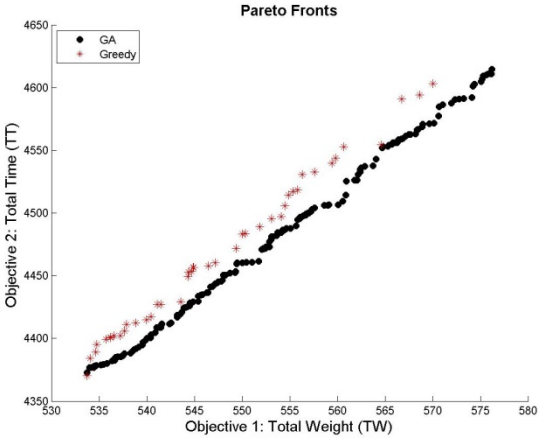
Network Size = 350, Problem Instance = 7



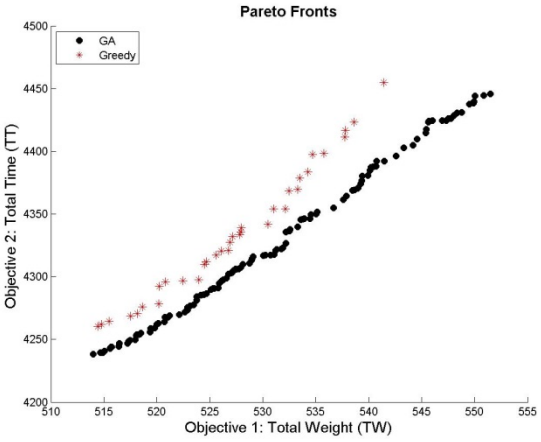
Network Size = 350, Problem Instance = 8



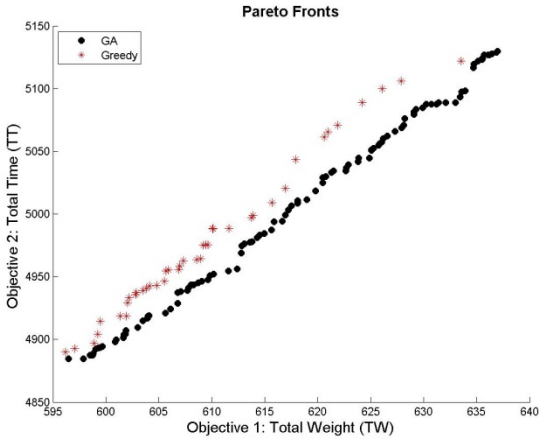
Network Size = 350, Problem Instance = 9



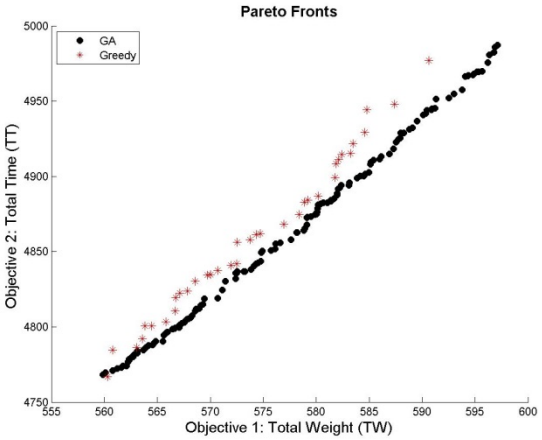
Network Size = 350, Problem Instance = 10



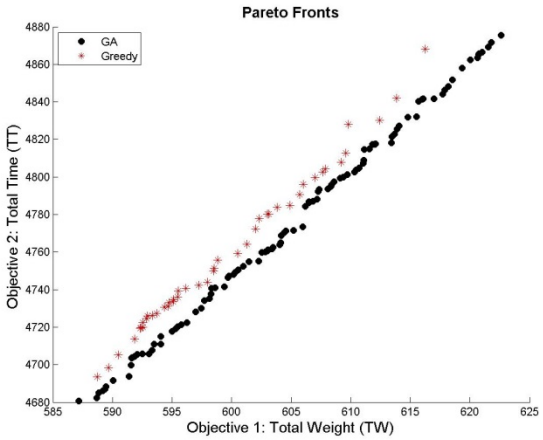
Network Size = 400, Problem Instance = 1



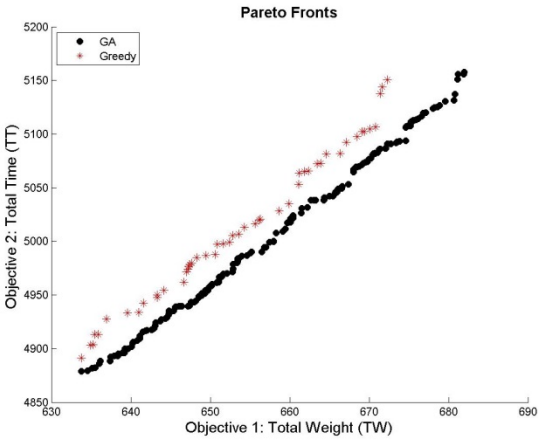
Network Size = 400, Problem Instance = 2



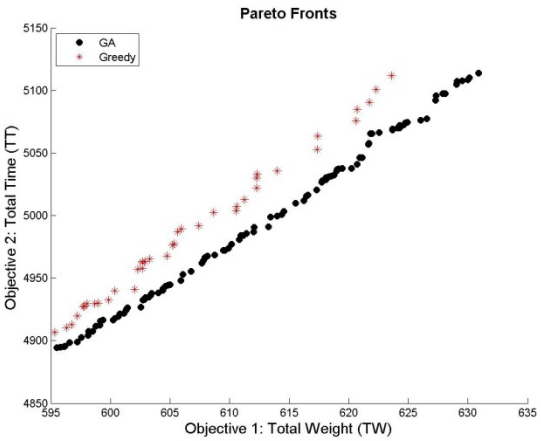
Network Size = 400, Problem Instance = 3



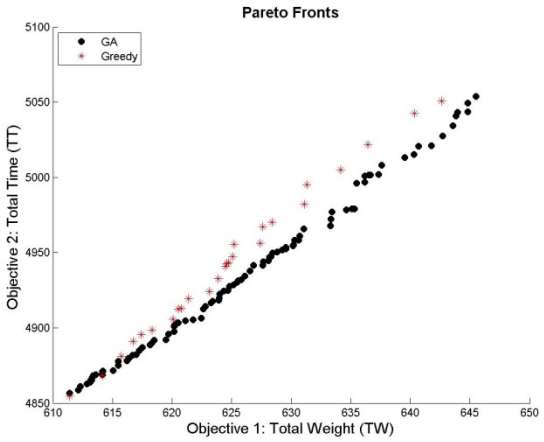
Network Size = 400, Problem Instance = 4



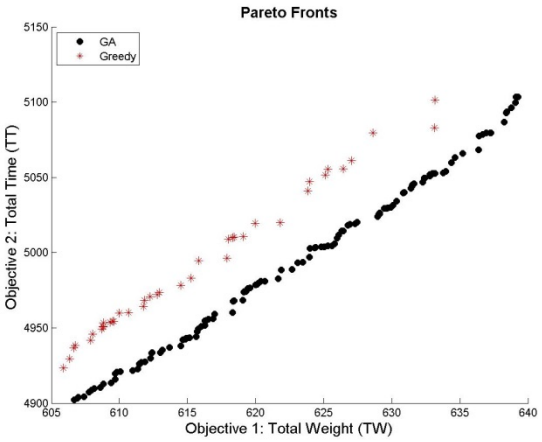
Network Size = 400, Problem Instance = 5



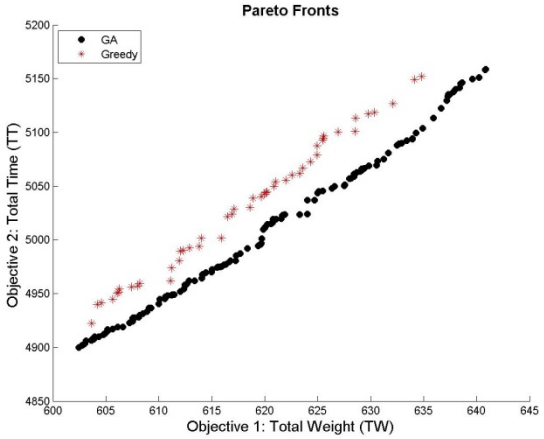
Network Size = 400, Problem Instance = 6



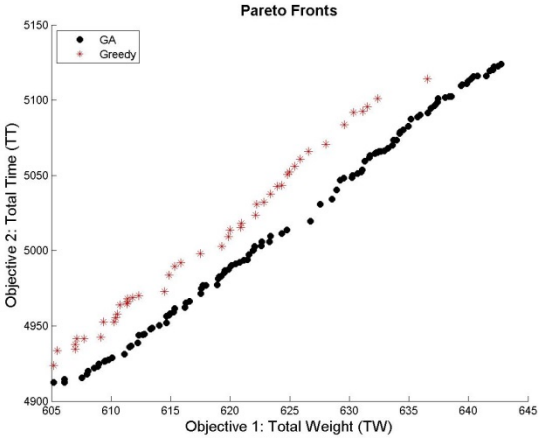
Network Size = 400, Problem Instance = 7



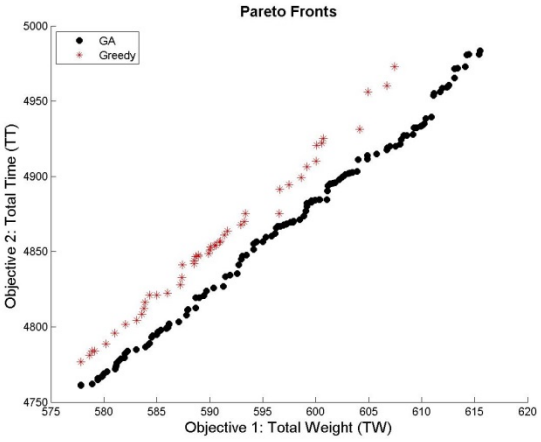
Network Size = 400, Problem Instance = 8



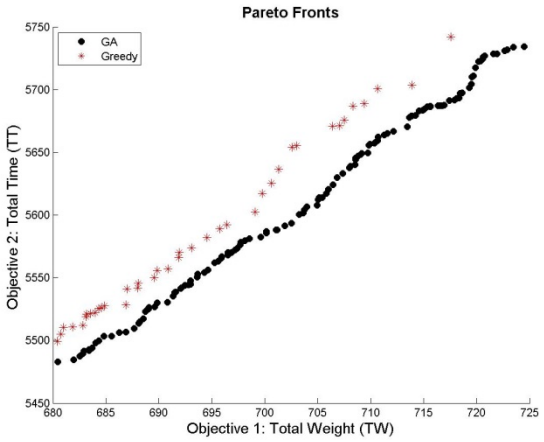
Network Size = 400, Problem Instance = 9



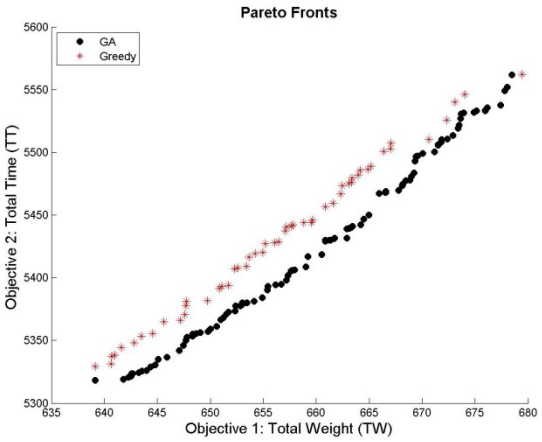
Network Size = 400, Problem Instance = 10



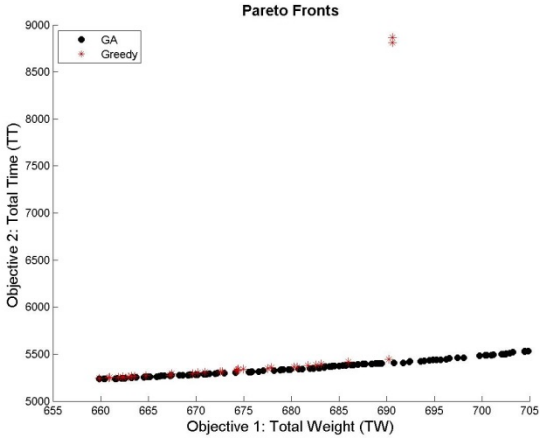
Network Size = 450, Problem Instance = 1



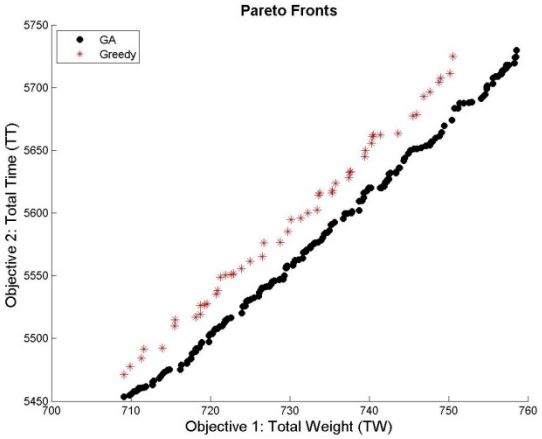
Network Size = 450, Problem Instance = 2



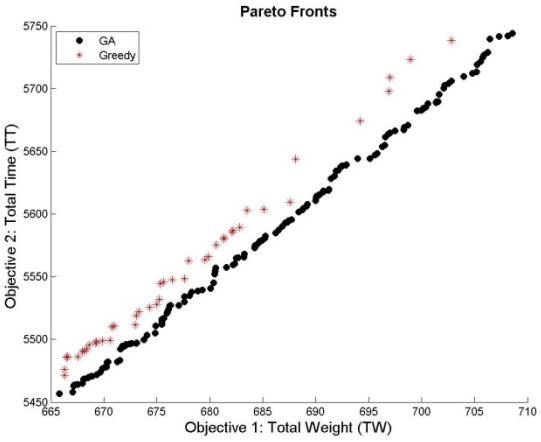
Network Size = 450, Problem Instance = 3



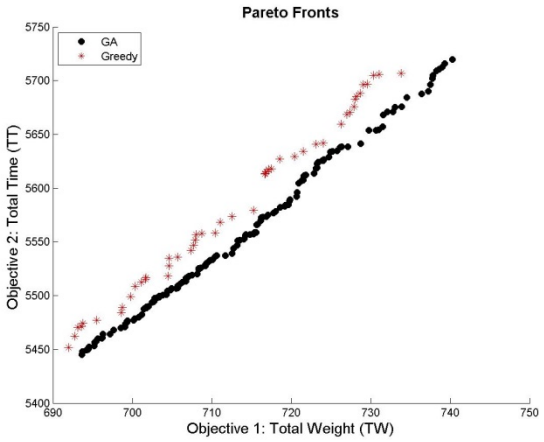
Network Size = 450, Problem Instance = 4



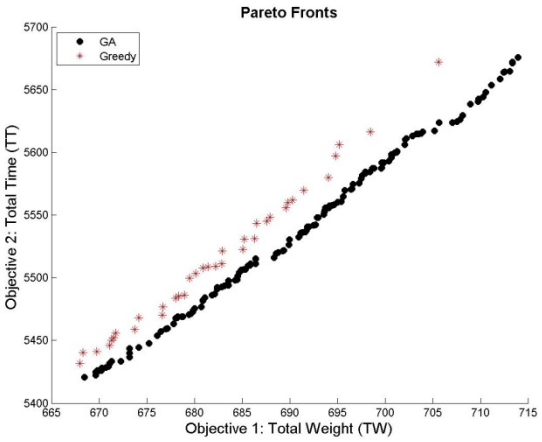
Network Size = 450, Problem Instance = 5



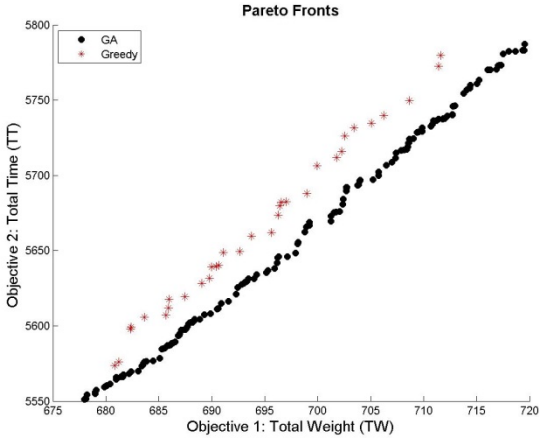
Network Size = 450, Problem Instance = 6



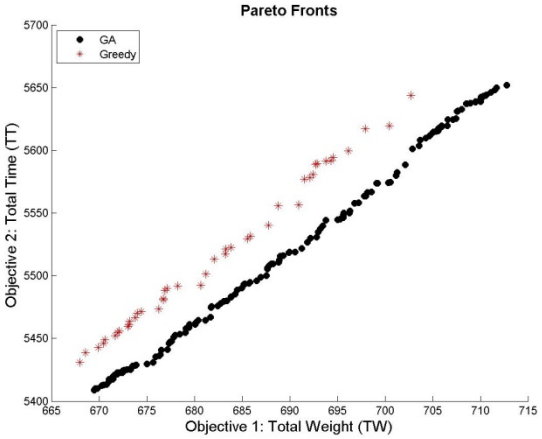
Network Size = 450, Problem Instance = 7



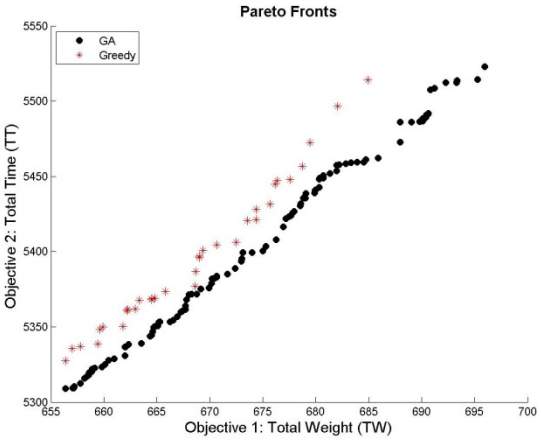
Network Size = 450, Problem Instance = 8



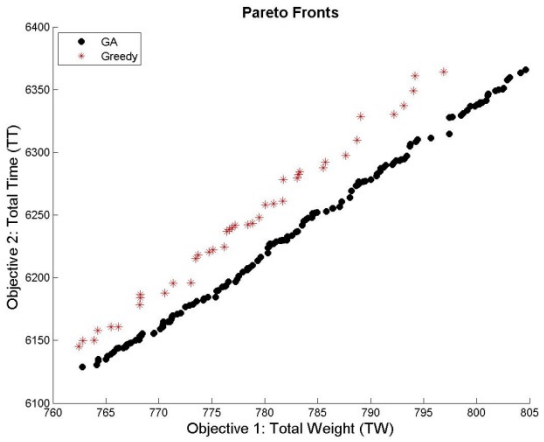
Network Size = 450, Problem Instance = 9



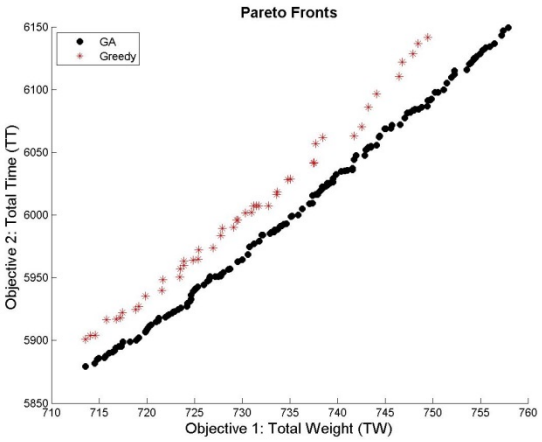
Network Size = 450, Problem Instance = 10



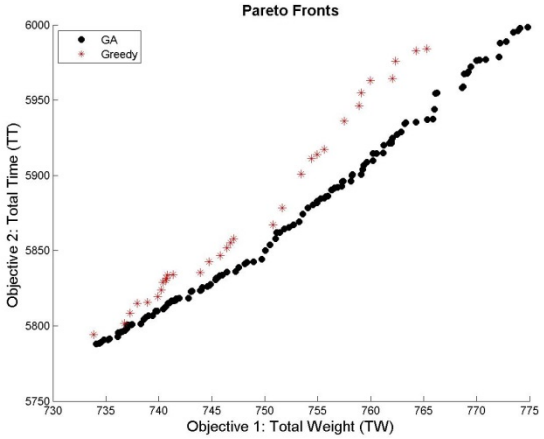
Network Size = 500, Problem Instance = 1



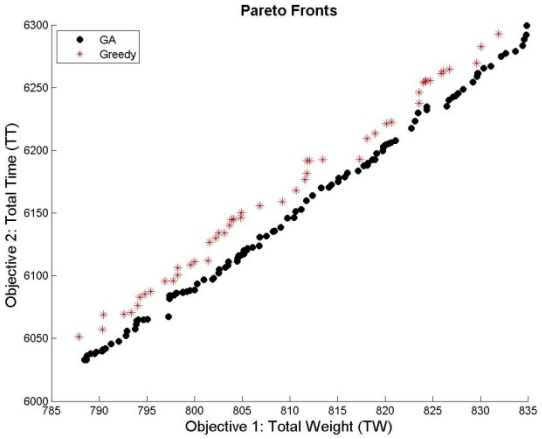
Network Size = 500, Problem Instance = 2



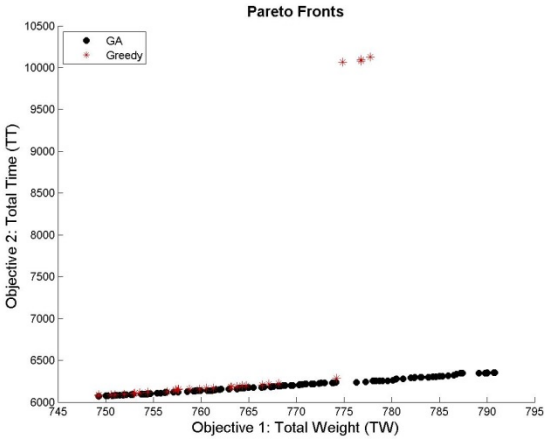
Network Size = 500, Problem Instance = 3



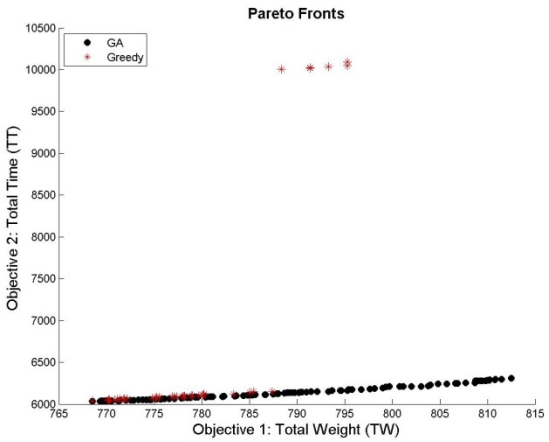
Network Size = 500, Problem Instance = 4



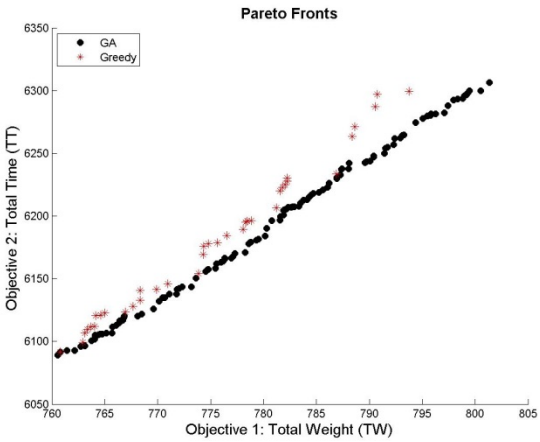
Network Size = 500, Problem Instance = 5



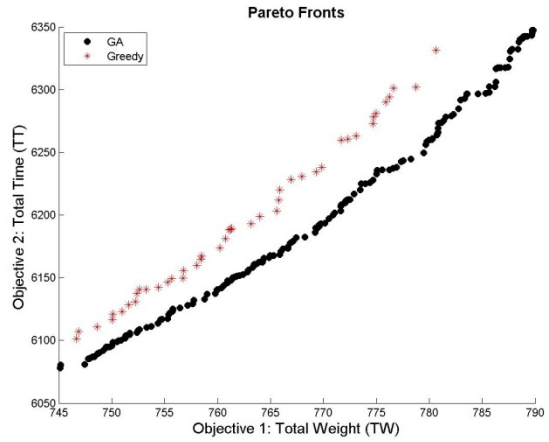
Network Size = 500, Problem Instance = 6



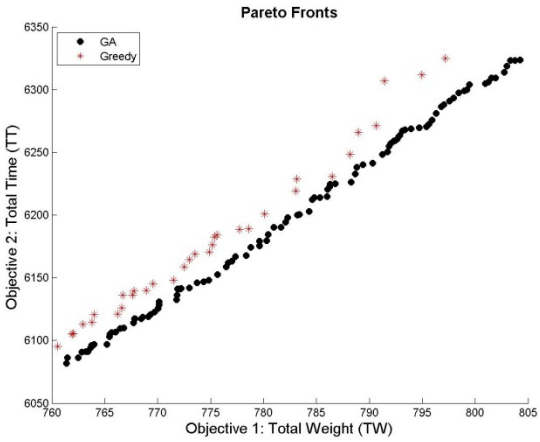
Network Size = 500, Problem Instance = 7



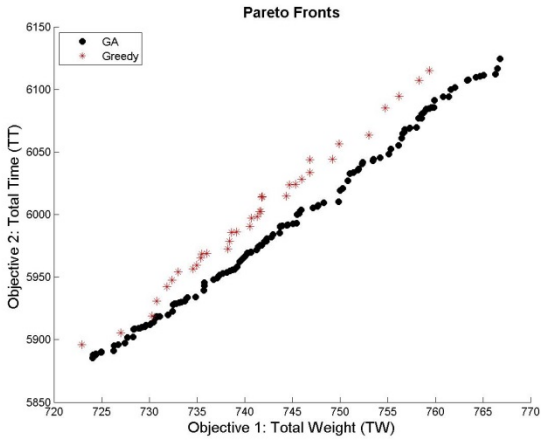
Network Size = 500, Problem Instance = 8



Network Size = 500, Problem Instance = 9



Network Size = 500, Problem Instance = 10



Appendix G: User's Guide for Track Inspection Planning Algorithms

In this manual, the details of the codes and how they can be used are explained for the genetic algorithm (GA) and the greedy heuristic algorithm (Greedy). Input-output for each algorithm is explained along with what each Matlab 2014a function does in execution of the algorithms.

G.1. Input-Output for the Algorithms

G.1.1. Input Parameters for the Algorithms

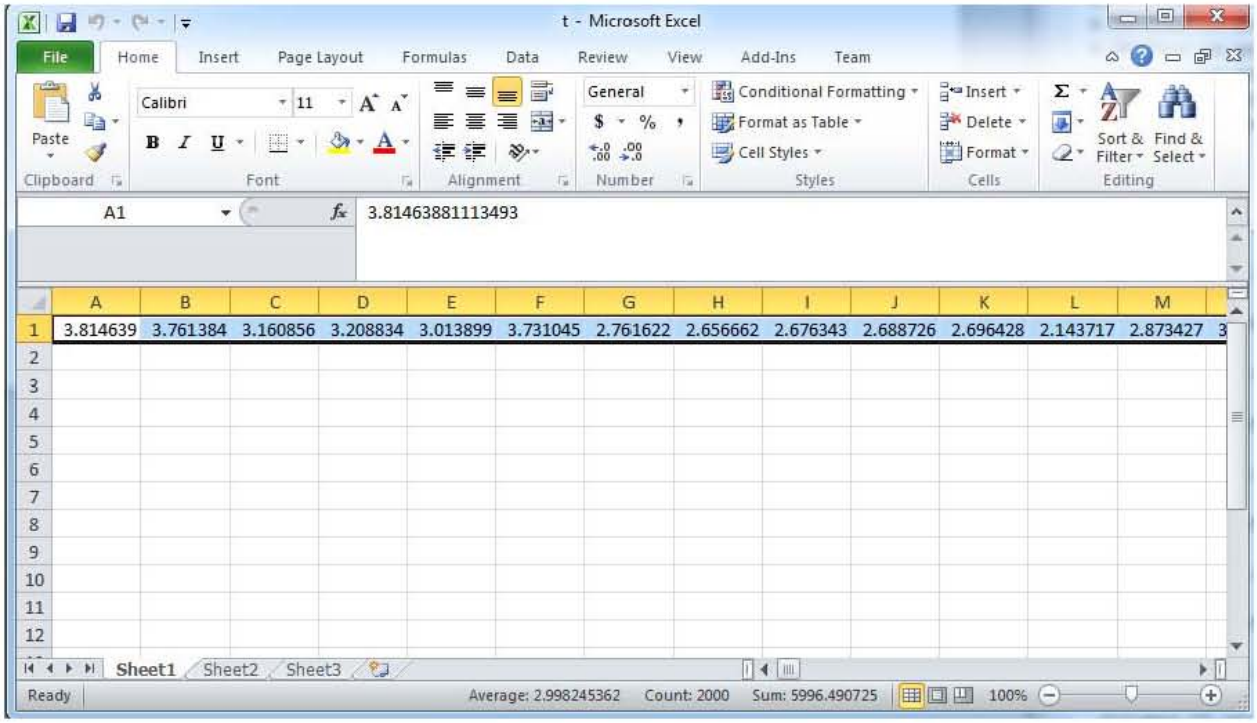
For both of the algorithms, the list of the input parameters is given in Table 25.

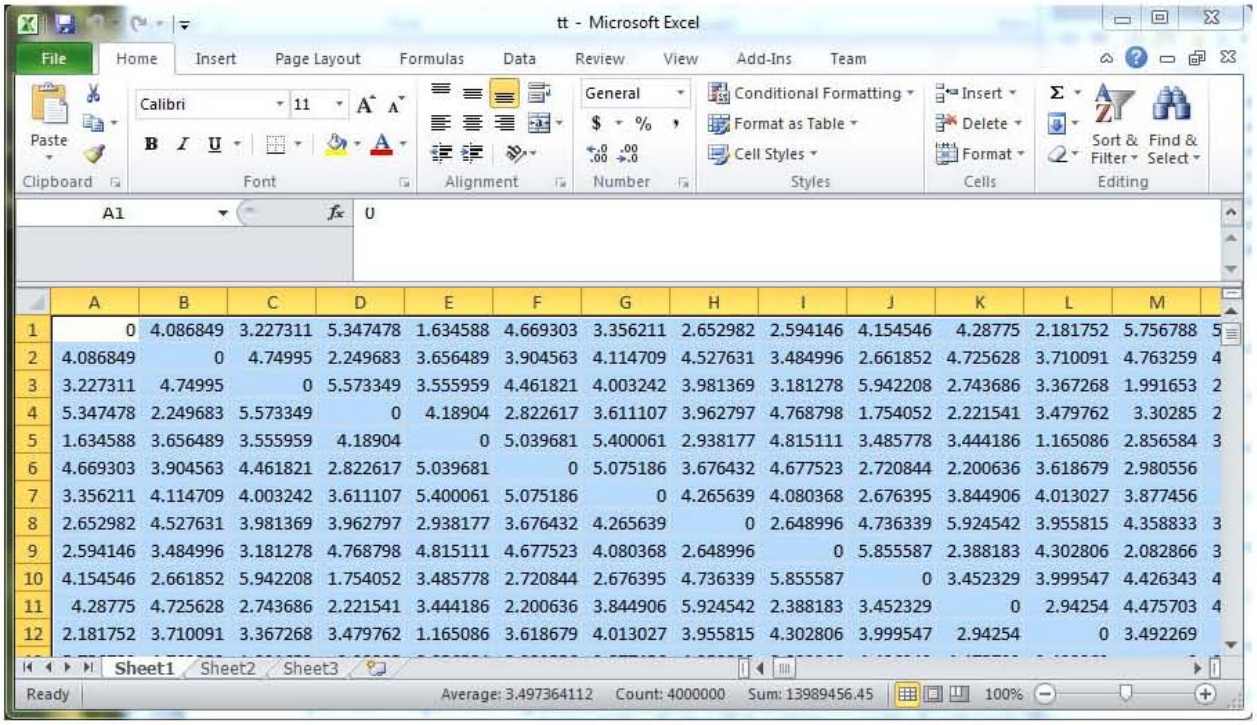
Table 25. List of Input Parameters for Genetic and Greedy Heuristic Algorithms

Name	Notation	Description
t	$t_{1\times n}$	A row vector of size n that contains inspection time of each track
tt	$\hat{t}_{n\times n}$	A matrix of size $n \times n$ that contains traveling time between tracks
tau	$\tau_{1\times n}$	A row vector of size n that shows minimum time between consecutive inspection of the same track
T	T	A scalar variable that represent time horizon
W	$W_{1\times n}$	A row vector of size n that contains a weight of each track (To improve safety)
L	$L_{1\times n}$	A row vector of size n that shows minimum required inspection for each track

Each of these parameters must be saved on the same directory as the directory of Matlab 2014a function files. All of them must be Excel files without any header or any symbol or alphabet. In addition, all vectors must be row-vectors in the Excel files. Please note that if any vector becomes a column vector or their sizes do not match, then there will be error in command window of Matlab 2014a. For example, Figure 8 shows screen shots of Excel files for t and tt.

Figure 8: Excel Screenshots for Excel Input Files

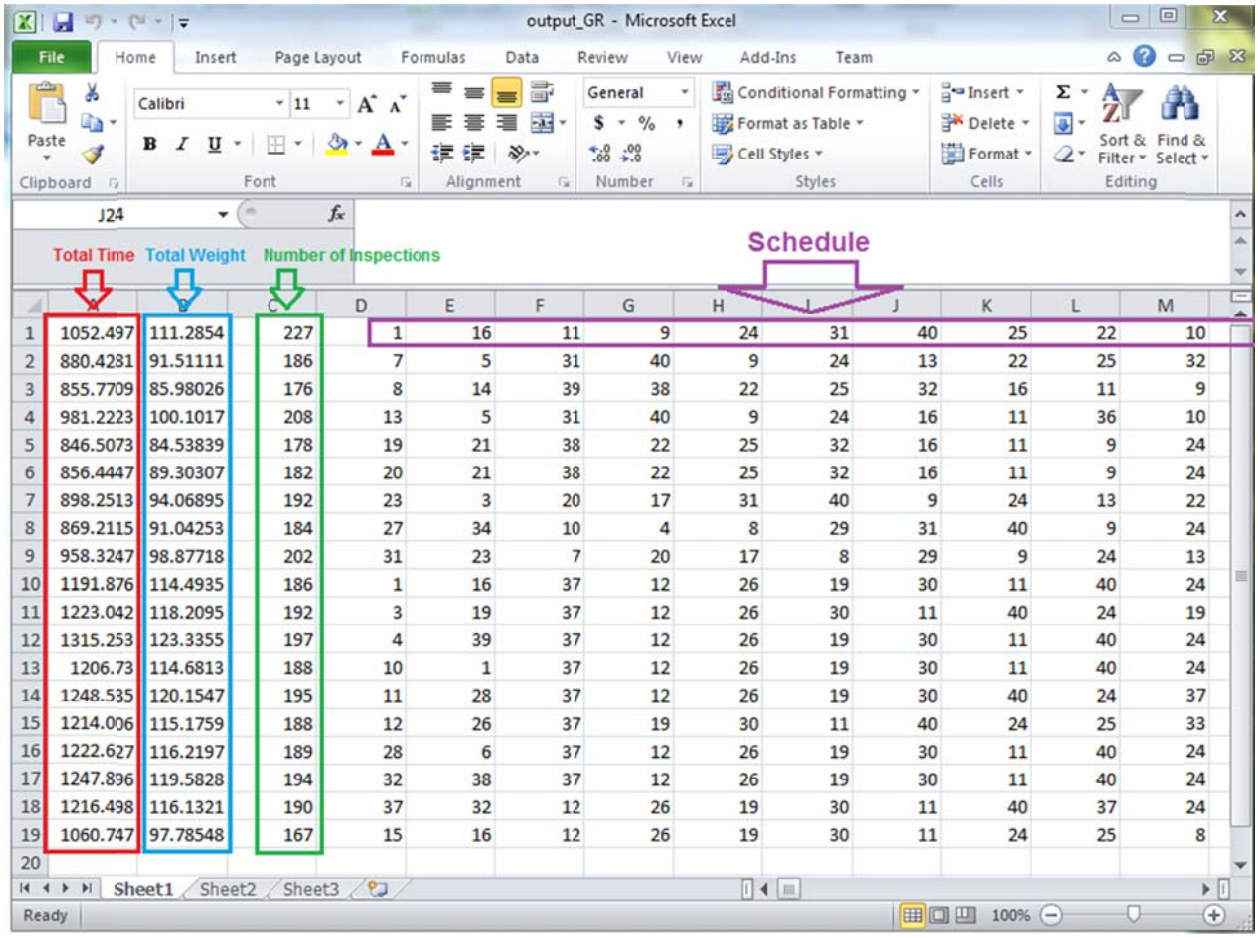




G.1.2. Output Parameters of the Algorithms

The outcome of both of the algorithms will be an Excel file, named "outcome_GA" and "outcome_GR" for the genetic algorithm and the greedy heuristic algorithm, respectively. Outcome file includes a matrix, in which, each row corresponds to an inspection schedule and its details. Each row contains: the total time of the inspection schedule in the first column, the total weight of the inspection schedule in the second column, the total number of inspections completed in the third column, and the sequence of tracks inspected in the remaining columns. Figure 9 demonstrates an output file's details.

Figure 9: Excel Screenshot for an Excel Output file



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1052.497	111.2854	227	1	16	11	9	24	31	40	25	22	10
2	880.4231	91.51111	186	7	5	31	40	9	24	13	22	25	32
3	855.7709	85.98026	176	8	14	39	38	22	25	32	16	11	9
4	981.2223	100.1017	208	13	5	31	40	9	24	16	11	36	10
5	846.5073	84.53839	178	19	21	38	22	25	32	16	11	9	24
6	856.4447	89.30307	182	20	21	38	22	25	32	16	11	9	24
7	898.2513	94.06895	192	23	3	20	17	31	40	9	24	13	22
8	869.2115	91.04253	184	27	34	10	4	8	29	31	40	9	24
9	958.3247	98.87718	202	31	23	7	20	17	8	29	9	24	13
10	1191.876	114.4935	186	1	16	37	12	26	19	30	11	40	24
11	1223.042	118.2095	192	3	19	37	12	26	30	11	40	24	19
12	1315.253	123.3355	197	4	39	37	12	26	19	30	11	40	24
13	1206.73	114.6813	188	10	1	37	12	26	19	30	11	40	24
14	1248.535	120.1547	195	11	28	37	12	26	19	30	40	24	37
15	1214.006	115.1759	188	12	26	37	19	30	11	40	24	25	33
16	1222.627	116.2197	189	28	6	37	12	26	19	30	11	40	24
17	1247.896	119.5828	194	32	38	37	12	26	19	30	11	40	24
18	1216.498	116.1321	190	37	32	12	26	19	30	11	40	37	24
19	1060.747	97.78548	167	15	16	12	26	19	30	11	24	25	8
20													

G.2. Genetic Algorithm Description and User Guidelines

The genetic algorithm uses the parameters listed in Table 21 as the input excel files and gives "outcome_GA" as the output excel file. In order to use this algorithm, the Matlab 2014a function file, named *Inspection_Planner_GA.m* should be opened and executed. After running, the following questions will be asked inside the Matlab Command Window:

1. What is the number of tracks?

2. What is the Time Horizon that you want to schedule within?
3. What is the number of non-improved parents to stop the algorithm?

For the first question, the user needs to put the number of tracks to be inspected. The number of tracks must be equal to the size of the row vector parameters given in Table 21. For question two, the length of the inspection period should be given. Please consider that unit of metric used for the length of the inspection period, T, must be equal to unit of metric used for travelling time (tt), inspection time (t), and the time between two consecutive inspections (tau). For example, if inspection time is given in hours, for instance 2.7 hours to inspect track 1, then, the length of the inspection period should also be given in terms of hours. For the third question, it is recommended to put "2". Although having a bigger number may help to find more Pareto efficient inspection schedules, it takes considerably more time for the algorithm to terminate.
The main file can be seen in the following Matlab code:

```
%% ======================= Inspection Planner GA ==========================
clc;clear;
%% ========================== Step 1: Input Data ==========================
[n,t,tt,L,tau,W,T,I,pop_numbers,matrix_length,number_mutation,...
   pareto_front_data,mean_weight_time,non_improve,parent_number]=create_inputs();
%% ========================== Step 2: GA Code==============================
output=GA_TISP(n,t,tt,L,tau,W,T,I,pop_numbers,matrix_length,number_mutation,...
   pareto_front_data,mean_weight_time,non_improve,parent_number);
%% ==================== Step 3: saving final Solution =====================
xlswrite('output_GA.xlsx', output);
%% ======================= Step 4: Drawing figures ========================
scatter(output(:,1),output(:,2),30,'c','MarkerEdgeColor','b','MarkerFaceColor','b');
xlabel('Objective 1: Total Time (TT)','FontSize',10);
ylabel('Objective 2: Total Weight (TW)','FontSize',10);
title('Pareto Front','FontSize',10,'FontWeight','bold');
%%
=========================================================================
```

Inside the main file for GA algorithm stated above (Inspection_Planner_GA.m), two functions are included: (1) "create_inputs" function, and (2) "GA_TISP" function. In the next two subsections, each one of them is discussed. In addition, in this main file, the outcome is saved as "outcome_GA", an Excel file, and the figure of Pareto Front is drawn.

G.2.1. "create_inputs" function

This function reads parameters, which are introduced in Table 21, from the same directory of GA codes. First, it is asking about the number of tracks (n), the length of the inspection period (T) and the number of non-improved parents to stop (these questions were discussed above). The following shows the code for this function.

```
function [n,t,tt,L,tau,W,T,I,pop_numbers,matrix_length,number_mutation,...
    pareto_front_data,mean_weight_time,non_improve,parent_number]=create_inputs()
fprintf('\n ---> ');
prompt1='What is the number of tracks?';
n=input(prompt1); % Getting number of tracks
pop_numbers=n;

%% Importing parameters t,tt,tau,L,W
% t: a Vector that shows Inspection Time
% tt: A matrix that shows Traveling Time between tracks
% tau: a Vector that shows time for consecutive inspection of tracks
% L: a Vector that shows Minimum required inspection for each track
% W: Importance of Tracks
a=xlsread('t.xlsx');
b=xlsread('tt.xlsx');
c=xlsread('tau.xlsx');
d=xlsread('L.xlsx');
e=xlsread('W.xlsx');
t=a(1:n);
tt=b(1:n,1:n);
tau=c(1:n);
L=d(1:n);
W=e(1:n);

fprintf('\n ---> '); % a function of number of tacks
prompt1='What is Time Horizon That you want to schedule within?';
T=input(prompt1);
I=(1:n); % set of tracks

matrix_length=ceil(max(t)*sum(L));
% we are using that to create a matrix to store chromosome the matrix has a
% lot of zeros and just a part of each row will be filled by non-zero elements
number_mutation=10; % Getting number of mutations
% number of mutations can be a function of sum(L), i.e. number_mutation=0.005*sum(L)
pareto_front_data=zeros(10*pop_numbers,2,1);
% A matrix to save the objective value of non-dominated solution in each
% iteration
mean_weight_time(1,1)=0;
% come up with another way to represent the improvement through iterations
parent_number(1,1)=0;
fprintf('\n What is the number of non-improved parents to stop the algorithm? ...');
prompt2='\n (Please note that 2 would be sufficient!)';
non_improve=input(prompt2); % Getting number of non-improved parents to stop the
```

```
algorithm
end
```

G.2.2. “GA_TISP” function

Using the outcome of “create_inputs” function, this function is performing genetic algorithm operations to find a set of Pareto efficient inspection schedules for the track inspection problem. The code for “GA_TISP” is as follows:

```
% ================================TIPP GA ================================
function output=GA_TISP(n,t,tt,L,tau,W,T,I,pop_numbers,matrix_length,number_mutation,...
pareto_front_data,mean_weight_time,non_improve,parent_number)
%% ======================= 1. Initial Population ========================
fprintf('\n ========================================');
fprintf('\n Generating Initial Solutions:');
[pop,pop_length,inspections_matrix,spent_time_matrix]=...
initiate_population(n,t,tt,L,tau,I,matrix_length,pop_numbers);
fprintf('\n Initial Solutions is generated');
fprintf('\n ========================================');
PARENT=[];
%% ========================== 2. Iteration Loop ========================
iter=0;
non_improved_parents=1;
while non_improved_parents<non_improve
iter=iter+1;
% ================= Pareto-Front and Parent Selection =================
[parent,parent_inspections_matrix,pareto_front_data,...
parent_length,parent_spent_time_matrix,mean_weight_time,parent_number]=...
pareto_front_selection(n,pop,pop_length,inspections_matrix,W,...
spent_time_matrix,pareto_front_data,iter,mean_weight_time,parent_number);

if isequal(parent,PARENT)

non_improved_parents=non_improved_parents+1;
fprintf('\n ========================================');
fprintf('\n \t Same Non-Dominated Chromosomes is generated');
fprintf('\n \t \t Non-Dominated Chromosomes are being mutated');
% ====================== Mutating Population ======================
[pop,pop_length,inspections_matrix,spent_time_matrix]=...
mutation(n,t,tt,tau,L,W,T,I,parent,parent_length,parent_inspections_matrix,...
parent_spent_time_matrix,matrix_length,pop_numbers,number_mutation);
% ================================================================
fprintf('\n \t \t New population is generated');
fprintf('\n ========================================');
else
```

```
non_improved_parents=1;
PARENT=parent;
      fprintf('\n ========================================');
      fprintf('\n \t New Non-Dominated Chromosomes is generated');
      fprintf('\n \t \t Non-Dominated Chromosomes are being mutated');
      % ==================== Mutating Population ======================
      [pop,pop_length,inspections_matrix,spent_time_matrix]=...
mutation(n,t,tt,tau,L,W,T,I,parent,parent_length,parent_inspections_matrix,...
         parent_spent_time_matrix,matrix_length,pop_numbers,number_mutation);
      % ========================================================================
      fprintf('\n \t \t New population is generated');
      fprintf('\n ========================================');
   end
   fprintf('\n Iteration: %2d', iter);
   fprintf('\n Non-Improved Parents: %2d', non_improved_parents);
end
output=[parent_spent_time_matrix,parent_inspections_matrix*W',parent_length,parent];
end
```

Inside this, the following functions are included, for which the details are explained next.

G.2.2.1. "initiate_population" function

This function is creating an initial population consisting of chromosomes. To generate a feasible chromosome, “create_chromosome” function is executed. This function has been executed until it returns a feasible chromosome; this process continues until it generates the number of feasible solutions that are targeted. The code for this function is as follows:

```
%% ============================ Input ================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% L: a vector of size n that shows the minimum required inspections of tracks

%% ============================ Output ===============================
% pop: a matrix that holds chromosomes of population
% pop_length: a vector that each element of it shows the total number of
% inspection in each solution
% inspections_matrix: a matrix that contains inspection vector of each solution
% spent_time_matrix: a matrix that contains spent time for each solution in
% population
%%
========================================================================
```

```
function [pop,pop_length,inspections_matrix,spent_time_matrix]=...
   initiate_population(n,t,tt,L,tau,I,matrix_length,pop_numbers)

% creating a zero matrix to store populations and other parameters
pop=zeros(pop_numbers,matrix_length);
pop_length=zeros(pop_numbers,1);
inspections_matrix=zeros(pop_numbers,n);
spent_time_matrix=zeros(pop_numbers,1);

for i=1:pop_numbers
   feasibility=0;
   while feasibility==0
      [feasibility,chrom,inspections,spent_time]=...
         create_chromosome(n,t,tt,tau,L,I);
   end
   % Storing feasible chromosome in the matrix, as well as the length
   % of it and inspections matrix and spent_time
   pop(i,1:length(chrom))=chrom;
   pop_length(i,1)=length(chrom);
   inspections_matrix(i,:)=inspections;
   spent_time_matrix(i,1)=spent_time;
end
end
```

G.2.2.2. “create_chromosome” function

This function will help create chromosomes. The idea is that starting from an empty chromosome; it adds feasible genes until satisfying minimum required inspections for all tracks. The code is as follows:

```
%% ============================== Input ============================
% n: number of tracks
% t: Inspection Time
% tt: Traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% L: A vector that shows minimum required inspection for each track
% chrom: the solution that we want to find its properties
%% ============================= Output ============================
% feasibility: a binary variable that shows the generated chromosome is feasible or not
% chrom: The created chromosome
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom
%% =================================================================
function [feasibility,chrom,inspections,spent_time]=...
```

```
    create_chromosome(n,t,tt,tau,L,I)
%% ======================= Initialing the Solution ==============
feasibility=1;

inspections=zeros(1,n); % actual inspections
LST=-tau;
fistgenetrials=randperm(n);

x=fistgenetrials(1);
chrom=x; % generate the first gene
spent_time=t(chrom);
gene=1;
inspections(chrom)=inspections(chrom)+1;
LST(chrom)=0;
% gtl (genetrial left): a set of remaining inspections
gtl=I(L>inspections); % Set of left inspections to satisfy the required inspections
gtl=gtl(randperm(length(gtl))); % randomize order of set
% gtd (genetrial done): a set of done inspections
gtd=I(L<=inspections); % Set of done inspections
gtd=gtd(randperm(length(gtd))); % randomize order of set
%% ================= While Loop to Create Feasible Chromosome ======
while sum(L>inspections)>0 % this should change % spent_time<=T
    y=gtl(1); % Creating second gene
    EST=spent_time+tt(chrom(gene),y); % updating earliest start time possible
    if EST-LST(y)>=tau(y)
        chrom=[chrom,y]; % adding the gene
        LST(y)=EST; % updating the latest start time
        spent_time=EST+t(y); % updating the total time spent
        inspections(y)=inspections(y)+1; % updating the actual inspections
        gene=gene+1; % next gene
        % ============== updating gtl and gtd ====================
        gtl=I(L>inspections); % Set of left inspections to satisfy the required inspections
        gtl=gtl(randperm(length(gtl))); % randomize order of set

        gtd=I(L<=inspections); % Set of done inspections
        gtd=gtd(randperm(length(gtd))); % randomize order of set
        % ========================================================
    else
        gtl(1)=[]; % here, we could not add the current gene, so next potential gene will not
be the same
        if isempty(gtl) % if we have tried every potential gene and could not add, then we
are infeasible; hence, start the chromosome over
            gtl=gtd(1); % picking new genes from gtd
            gtd(1)=[]; % removing that gene from gtd
            if isempty(gtd)
                fistgenetrials(1)=[];
```

```
            if isempty(fistgenetrials)
               % we have possible infeasibility
               feasibility=0;
               chrom=[];
               inspections=L;
            else
               inspections=zeros(1,n); % actual inspections
               EST=0;
               LST=-tau;
               x=fistgenetrials(1);
               chrom=x; % generate the first gene
               spent_time=t(chrom);
               gene=1;
               inspections(chrom)=inspections(chrom)+1;
               LST(chrom)=0;
               gtl=I(L>inspections); gtl=gtl(randperm(length(gtl)));
               gtd=I(L<=inspections); gtd=gtd(randperm(length(gtd)));
            end
         end
      end
   end
end
end
```

G.2.2.3. "pareto_front_selection" function

This function finds non-dominated solutions. It compares every two solutions inside the population. If one solution dominates another one in all objectives, then the dominated solution is removed. Then it sorts solutions based on their total time. The code is as follows:

```
%% ================ Finding Non-Dominated Set of Solutions ================
function [parent,parent_inspections_matrix,pareto_front_data,...
  parent_length,parent_spent_time_matrix,mean_weight_time,parent_number]=...
   pareto_front_selection(n,pop,pop_length,inspections_matrix,W,...
   spent_time_matrix,pareto_front_data,iter,mean_weight_time,parent_number)

objectives=[inspections_matrix*W',spent_time_matrix];
pop_counter=1;

while pop_counter<=size(objectives,1)-1
   index_check=pop_counter+1;
   while index_check<=size(objectives,1)
      if objectives(pop_counter,1)>=objectives(index_check,1) && ...
            objectives(pop_counter,2)<=objectives(index_check,2)
         objectives(index_check,:)=[];
```

```
            pop(index_check,:)=[];
            pop_length(index_check,:)=[];
            inspections_matrix(index_check,:)=[];

        else
            if objectives(pop_counter,1)<=objectives(index_check,1) && ...
                    objectives(pop_counter,2)>=objectives(index_check,2)

                objectives(pop_counter,:)=[];
                pop(pop_counter,:)=[];
                pop_length(pop_counter,:)=[];
                inspections_matrix(pop_counter,:)=[];
                % we need to update indices after this. Because removing
                % the first chromosome without restarting the index of
                % second chromosome can leave some compares unchecked!
                pop_counter=pop_counter-1;
                index_check=size(objectives,1)+1;

            else
                index_check=index_check+1;
            end
        end
    end
    pop_counter=pop_counter+1;
end

%% ========================= Output Information ============================
%pareto_front_data(:,:,iter)=zeros(pop_numbers,2);
pareto_front_data(1:size(objectives,1),:,iter)=objectives;

parent=pop;
parent_inspections_matrix=inspections_matrix;
parent_length=pop_length;
parent_spent_time_matrix=objectives(:,2);

mean_weight_time(1,iter)=mean(objectives(:,1)./objectives(:,2));
% max_weight_time(1,iter)=max(objectives(:,1)./objectives(:,2));

parent_number(1,iter)=size(parent,1);

%% ================== sorting data based on spent_time ====================
b=size(parent,2);

parent=[parent,parent_spent_time_matrix];
parent_inspections_matrix=[parent_inspections_matrix,parent_spent_time_matrix];
parent_length=[parent_length,parent_spent_time_matrix];
```

```
parent=sortrows(parent,b+1);
parent_inspections_matrix=sortrows(parent_inspections_matrix,n+1);
parent_length=sortrows(parent_length,2);
parent_spent_time_matrix=sortrows(parent_spent_time_matrix,1);

parent=parent(:,1:b);
parent_inspections_matrix=parent_inspections_matrix(:,1:n);
parent_length=parent_length(:,1);
end
```

G.2.2.4. "mutation" function

In this function, first a random number less than the length of a chromosome is generated and the function selects a part of chromosome that has a length less than this number, starting from the beginning of the chromosome. Then, the function uses the "chrom_properties" function to find the $inspections$, $spent\_time$, LST for this part. Using "partial_mutation" function, the function fills the rest of the chromosome while maintaining the feasibility. Mutation generates reserved chromosome by the "initial_population" function. The new population is the combination of the parents, the children generated though mutation and the reserved chromosomes. The code for this function is as follows:

```
function [pop,pop_length,inspections_matrix,spent_time_matrix]=...
   mutation(n,t,tt,tau,L,W,T,I,parent,parent_length,parent_inspections_matrix,...
   parent_spent_time_matrix,matrix_length,pop_numbers,number_mutation)

% ======================== Mutating Population ========================
k=size(parent,1);

pop_mutate=[];
pop_mutate_length=[];
inspections_mutate_matrix=[];
spent_time_mutate_matrix=[];

for j=1:k

   [pop_mutated,pop_mutated_length,...
      inspections_mutated_matrix,spent_time_mutated_matrix]=...

partial_mutation(n,t,tt,tau,L,W,T,I,parent(j,:),parent_length(j,1),matrix_length,number_mutation)
;

   if ~isempty(pop_mutated_length)
      pop_mutate=[pop_mutate;pop_mutated];
      pop_mutate_length=[pop_mutate_length;pop_mutated_length];
```

```
        inspections_mutate_matrix=...  [inspections_mutate_matrix;inspections_mutated_matrix];
        spent_time_mutate_matrix=[spent_time_mutate_matrix;spent_time_mutated_matrix];
    end
end
% ======================= Reserved Population ========================
[pop_reserve,pop_length_reserve,inspections_matrix_reserve,spent_time_matrix_reserve]=...initiate_population(n,t,tt,L,tau,I,matrix_length,pop_numbers);
% ======================= New Population ==============================
pop=[parent;pop_mutate;pop_reserve];
pop_length=[parent_length;pop_mutate_length;pop_length_reserve];
inspections_matrix=[parent_inspections_matrix;inspections_mutate_matrix;inspections_matrix_reserve];
spent_time_matrix=[parent_spent_time_matrix;spent_time_mutate_matrix;spent_time_matrix_reserve];
end
```

G.2.2.5. “partial_mutation” function

This function first divides the length of the chromosome to the number of mutations. Doing this enables finding the length of the parts to be mutated. Each part is mutated once for minimizing TT and once for maximizing TW, and the results are saved in a matrix. Inside this function, “repair_mutation” function is used. The code is as follows:

```
%% ============================== Input ================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% W: a vector that shows the importance of each track
% T: Time horizon
% I: a vector between 1 and n that constraints track set
% pop: population member that we want to mutate
% pop_length: length of that pop member
% matrix_length: length of matrix we are saving value into
% number_mutation: number of times we mutate every population member
%% ============================= Output ============================
% inspections_mutated_matrix: Inspection matrix (Which tracks are being inspected
% spent_time_mutated_matrix: total time of inspection in a particular solution
% pop_mutated: the result of mutation over a population member (children)
% pop_mutated_length: length of children
function [pop_mutated,pop_mutated_length,...
    inspections_mutated_matrix,spent_time_mutated_matrix]=...
    partial_mutation(n,t,tt,tau,L,W,T,I,pop,pop_length,matrix_length,number_mutation)
part_length=floor(pop_length/number_mutation);
pop_mutated=zeros(1,matrix_length);
```

```
pop_mutated_length=[];
inspections_mutated_matrix=[];
spent_time_mutated_matrix=[];
counter=1;
for i=1:number_mutation
   if i<number_mutation
      mutated_gene=randi([(i-1)*part_length+1,i*part_length]);
   else
      mutated_gene=randi([(i-1)*part_length+1,pop_length-1]);
   end
   % A=inspections; B=spent_time; C=LST;D=chrom;
   chrom=pop(1:mutated_gene);
   [A,B,C]=chrom_properties(n,t,tt,tau,chrom); D=chrom;

   feasibility=0;
   while ~feasibility
      [feasibility,chrom,inspections,spent_time]=...
         repaire_mutation_TT(n,t,tt,tau,L,T,I,D,C,B,A);
   end
   if spent_time<=T
      pop_mutated(counter,1:length(chrom))=chrom;
      pop_mutated_length(counter,1)=length(chrom);
      inspections_mutated_matrix(counter,:)=inspections;
      spent_time_mutated_matrix(counter,1)=spent_time;
      counter=counter+1;
   end
end
for i=1:number_mutation
   if i<number_mutation
      mutated_gene=randi([(i-1)*part_length+1,i*part_length]);
   else
      mutated_gene=randi([(i-1)*part_length+1,pop_length-1]);
   end
   % A=inspections; B=spent_time; C=LST;D=chrom;
   chrom=pop(1:mutated_gene);
   [A,B,C]=chrom_properties(n,t,tt,tau,chrom); D=chrom;
   feasibility=0;
   while ~feasibility
      [feasibility,chrom,inspections,spent_time]=...
         repaire_mutation_TW(n,t,tt,tau,L,W,I,D,C,B,A);
   end
   if spent_time<=T
      pop_mutated(counter,1:length(chrom))=chrom;
      pop_mutated_length(counter,1)=length(chrom);
      inspections_mutated_matrix(counter,:)=inspections;
      spent_time_mutated_matrix(counter,1)=spent_time;
```

```
        counter=counter+1;
    end
end
end
```

G.2.2.6. "chrom_properties" function

This function will help repair a chromosome after mutation. Given a part of chromosome, one can calculate the following properties of the chromosome by this function: (1) inspections vector to compare with L, (2) $spent\_time$, and (3) LST (Latest start time). The code is as follows:

```
%% ============================== Input ==================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% chrom: the solution that we want to find its properties
%% ============================== Output ==============================
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom
% LST: a vector that shows the latest start time of inspection for the
% tracks
%% ============================ Algorithm ================================
% First we collect the first gene in the chromosome and update the LST as -tau.
% Second, we update the earliest start time to zero.
% Third, The latest start time of that gene will be zero.
% Finally, we constitute a loop on the chromosome to update the three
% variables to the end of chromosome
%%
=======================================================================
function [inspections,spent_time,LST]=chrom_properties(n,t,tt,tau,chrom)

m=length(chrom); % calculating the length of chromosome (How many inspections)

inspections=zeros(1,n); % initiate inspection matrix
LST=-tau; % initiate Latest start time
spent_time=t(chrom(1)); % initial value of total time of the solution
LST(chrom(1))=0; % updating Latest Start time for the solution i.e. chromosome
inspections(chrom(1))=inspections(chrom(1))+1;
for i=1:(m-1)
   EST=spent_time+tt(chrom(i),chrom(i+1));
   LST(chrom(i+1))=EST;
   spent_time=EST+t(chrom(i+1));
   inspections(chrom(i+1))=inspections(chrom(i+1))+1;
end
end
```

G.2.2.7. “repair_mutation” function

There are two different functions that are repairing chromosomes after mutation. The whole process that they are following is similar to creating a feasible chromosome. However, instead of finding a random feasible gene, these functions add a feasible gene with minimum inspection and traveling time in one function (repair_mutation_TT) and a feasible gene with maximum weight in the other function (repair_mutation_TW).

The code for “repair_mutation_TT” function is as follows:

```
%% ============================== Input ================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% T: Time horizon
% I: a vector between 1 and n that constrains track set
% Chrom: chromosome that we want to repair
% LST: Latest start time of each track in chromosome
% spent_time: spent time in chromosome
% inspections: inspection vector in chromosome to compare to L

%% ============================== Output ===============================
% feasibility: a binary variable that shows the generated chromosome is feasible or not
% chrom: The created chromosome
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom
function [feasibility,chrom,inspections,spent_time]=...
   repaire_mutation_TT(n,t,tt,tau,L,T,I,chrom,LST,spent_time,inspections)
a=chrom;
b=LST;
c=spent_time;
d=inspections;
feasibility=1;
gtl=I(L>d); % Set of left inspections to satisfy the required inspections
gtl=gtl(randperm(length(gtl))); % randomize order of set
gtd=I(L<=d); % Set of done inspections
gtd=gtd(randperm(length(gtd))); % randomize order of set
% ================== Create a Feasible fistgenetrials ==================
fistgenetrials=[];
for i=1:n
   EST=spent_time+tt(chrom(end),i);
   if EST-LST(i)>=tau(i)
      fistgenetrials=[fistgenetrials,i];
```

```
    end
end
fistgenetrials=fistgenetrials(randperm(length(fistgenetrials)));
% ========================================================================
x=fistgenetrials(1); chrom=[a,x];
inspections=d; inspections(x)=inspections(x)+1;
LST=b; EST=c+tt(a(end),x); LST(x)=EST;
spent_time=EST+t(x);
gene=length(chrom);
%% ================ While Loop to Create Feasible Chromosome ==============
while sum(L>inspections)>0 % this should change % spent_time<=T
    y=greedy_selection_TT(t,tt,tau,T,LST,spent_time,chrom,gtl,gtd); % This must be a gready selection along with the following process
    if ~isempty(y)
        % Since, it is already feasible
        EST=spent_time+tt(chrom(gene),y); % updating earliest start time possible
        chrom=[chrom,y]; % adding the gene
        LST(y)=EST; % updating the latest start time
        spent_time=EST+t(y); % updating the total time spent
        inspections(y)=inspections(y)+1; % updating the actual inpsections
        gene=gene+1; % next gene
        % ============= updating gtl and gtd ====================
        gtl=I(L>inspections); gtl=gtl(randperm(length(gtl)));
        gtd=I(L<=inspections); gtd=gtd(randperm(length(gtd)));
        % ========================================================
    else
        fistgenetrials(1)=[];
        if isempty(fistgenetrials)
            % we have possible infeasibility
            feasibility=0;
            chrom=a;
            LST=b;
            spent_time=c;
            inspections=d;
        else
            gtl=I(L>d); gtl=gtl(randperm(length(gtl)));
            gtd=I(L<=d); gtd=gtd(randperm(length(gtd)));
            x=fistgenetrials(1); chrom=[a,x];
            inspections=d; inspections(x)=inspections(x)+1;
            LST=b; EST=c+tt(a(end),x); LST(x)=EST;
            spent_time=EST+t(x);
            gene=length(chrom);
        end
    end
end
end
```

The code for “repair_mutation_TW” function is as follows:

```
%% ============================== Input ==================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% W: a vector of weights of tracks
% I: a vector between 1 and n that constrains track set
% Chrom: chromosome that we want to repair
% LST: Latest start time of each track in chromosome
% spent_time: spent time in chromosome
% inspections: inspection vector in chromosome to compare to L

%% ============================== Output ================================
% feasibility: a binary variable that shows the generated chromosome is feasible or not
% chrom: The created chromosome
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom

function [feasibility,chrom,inspections,spent_time]=...
   repaire_mutation_TW(n,t,tt,tau,L,W,I,chrom,LST,spent_time,inspections)

a=chrom;
b=LST;
c=spent_time;
d=inspections;
feasibility=1;
gtl=I(L>d); % Set of left inspections to satisfy the required inspections
gtl=gtl(randperm(length(gtl))); % randomize order of set
gtd=I(L<=d); % Set of done inspections
gtd=gtd(randperm(length(gtd))); % randomize order of set
% ================== Create a Feasible fistgenetrials ==================
fistgenetrials=[];
for i=1:n
   EST=spent_time+tt(chrom(end),i);
   if EST-LST(i)>=tau(i)
      fistgenetrials=[fistgenetrials,i];
   end
end
fistgenetrials=fistgenetrials(randperm(length(fistgenetrials)));
% ======================================================================
x=fistgenetrials(1); chrom=[a,x];
inspections=d; inspections(x)=inspections(x)+1;
LST=b; EST=c+tt(a(end),x); LST(x)=EST;
```

```
spent_time=EST+t(x);
gene=length(chrom);
%% ================ While Loop to Create Feasible Chromosome ==============
while sum(L>inspections)>0 % this should change % spent_time<=T
   y=greedy_selection_TW(tt,tau,W,LST,spent_time,chrom,gtl,gtd); % This must be a gready
selection along with the following process
   if ~isempty(y)
      % Since, it is already feasible
      EST=spent_time+tt(chrom(gene),y); % updating earliest start time possible
      chrom=[chrom,y]; % adding the gene
      LST(y)=EST; % updating the latest start time
      spent_time=EST+t(y); % updating the total time spent
      inspections(y)=inspections(y)+1; % updating the actual inpsections
      gene=gene+1; % next gene
      % ============== updating gtl and gtd =====================
      gtl=I(L>inspections); gtl=gtl(randperm(length(gtl)));
      gtd=I(L<=inspections); gtd=gtd(randperm(length(gtd)));
      % ========================================================
   else
      fistgenetrials(1)=[];
      if isempty(fistgenetrials)
         % we have possible infeasibility
         feasibility=0;
         chrom=a;
         LST=b;
         spent_time=c;
         inspections=d;
      else
         gtl=I(L>d); gtl=gtl(randperm(length(gtl)));
         gtd=I(L<=d); gtd=gtd(randperm(length(gtd)));
         x=fistgenetrials(1); chrom=[a,x];
         inspections=d; inspections(x)=inspections(x)+1;
         LST=b; EST=c+tt(a(end),x); LST(x)=EST;
         spent_time=EST+t(x);
         gene=length(chrom);
      end
   end
end
end
```

G.2.2.8. “greedy_selection” functions

The functions in this section are "greedy_selection_TT" and "greedy_selection_TT" functions. In these two functions, the search is biased towards less inspected tracks and the search initially ignores the tracks, which satisfy their minimum inspections. In order to do so, the functions first

choose from gtl (the set of tracks with remaining inspections) considering their minimum inspection plus traveling time and maximum importance weight. If gtl is empty, the functions then choose a track from gtd (set of tracks with no remaining inspections).

The code for "greedy_selection_TT" function is as follows:

```
% inputs: gtl, gtd, n, t, tt, tau, LST, chrom
% Output: greedy_track

function greedy_track=greedy_selection_TT(t,tt,tau,T,LST,spent_time,chrom,gtl,gtd)

COST=T; % set cost a huge number
greedy_track=[];

%% ================== Greedy Feasible Biased Track ======================
m=length(gtl);
for i=1:m
   EST=spent_time+tt(chrom(end),gtl(i));
   if EST-LST(gtl(i))>=tau(gtl(i))
      % Now, between them, we find the one with less cost
      cost=tt(chrom(end),gtl(i))+t(gtl(i));
      if cost<COST
         greedy_track=gtl(i); % select it as the track
         COST=cost; % updating the cost
      end
   end
end

%% ======================== Feasible Biased Track ========================
if isempty(greedy_track)
   m=length(gtd);
   for i=1:m
      EST=spent_time+tt(chrom(end),gtd(i));
      if EST-LST(gtd(i))>=tau(gtd(i))
         % Now, between them, we find the one with less cost
         cost=tt(chrom(end),gtd(i))+t(gtd(i));
         if cost<COST
            greedy_track=gtd(i); % select it as the track
            COST=cost; % updating the cost
         end
      end
   end
end

end
```

The code for "greedy_selection_TW" function is as follows:

```
% inputs: gtl, gtd, n, t, tt, tau, LST, chrom
% Output: greedy_track
function greedy_track=greedy_selection_TW(tt,tau,W,LST,spent_time,chrom,gtl,gtd)
COST=0; % set cost zero for TW, since we want to find a track with maximum weight
greedy_track=[];
%% ================== Greedy Feasible Biased Track ======================
m=length(gtl);
for i=1:m
   EST=spent_time+tt(chrom(end),gtl(i));
   if EST-LST(gtl(i))>=tau(gtl(i))
      % Now, between them, we find the one with less cost
      cost=W(gtl(i));
      if cost>COST
         greedy_track=gtl(i); % select it as the track
         COST=cost; % updating the cost
      end
   end
end
%% ======================= Feasible Biased Track ========================
if isempty(greedy_track)
   m=length(gtd);
   for i=1:m
      EST=spent_time+tt(chrom(end),gtd(i));
      if EST-LST(gtd(i))>=tau(gtd(i))
         % Now, between them, we find the one with less cost
         cost=W(gtd(i));
         if cost>COST
            greedy_track=gtd(i); % select it as the track
            COST=cost; % updating the cost
         end
      end
   end
end

end
```

G.3. Greedy Algorithm Description and User Guidelines

Greedy algorithm uses the parameters given in Table 21 as the input and gives “outcome_GR” as the output. In order to use this algorithm, the m file (Inspection_Planner_Greedy.m) should be opened and executed. After running, inside the MATLAB command window, it will ask the following question:

- What is the number of Tracks?

For this question, the number of tracks should be given. The number of tracks must be equal to the size of the row vectors parameters in Table 21. It is important to note that similar to the genetic algorithm, the unit of time horizon T, must be equal to unit of travelling time (tt), inspection time (t), and the time between consecutive inspections (tau).

Inside the main file of Greedy algorithm (Inspection_Planner_Greedy.m), three functions are used: (1) "create_inputs" function, (2) "generate_tour" function, and (3) "pareto_front_selection" function. In this section, instead of representing a solution as a chromosome, a solution is represented as a tour.. In the following subsections, each of these functions are explained. In addition, in this main file, we save the outcome as "outcome_GR"; an Excel file, and we draw a figure of Pareto Fronts of solutions.

In this main function, the outcome is saved as "outcome_GR", an Excel file, and a figure of the inspection schedules in the Pareto Front is drawn. The "Inspection_Planner_Greedy.m" for greedy algorithm is as follows:

```
%% ================== Inspection Planner Greedy ====================
clc; clear;
%% ========================= Step 1: Input Data =====================
[n,I,t,tt,tau,L,W,matrix_length]=create_inputs();

%% ====================== Step 2: Initialization ====================
tour_matrix=zeros(1,matrix_length);
tour_length_matrix=zeros(1,1);
spent_time_matrix=zeros(1,1);
inspections_matrix=zeros(1,n);

%% ====================== Step 3: Iteration Loop ====================
% Finding solution to minimize TT
for start_point=1:n
   feasibility=0;
   while feasibility==0
      [feasibility,tour,spent_time,inspections]=...
         generate_tour_TT(n,I,t,tt,tau,L,start_point);
   end
   % saving data into matrices
   tour_length_matrix(start_point,1)=length(tour);
   tour_matrix(start_point,1:length(tour))=tour;
   spent_time_matrix(start_point,1)=spent_time;
   inspections_matrix(start_point,:)=inspections;
   fprintf('\n A TT-Tour is generated starting from: %2d', start_point);
end
% Finding solution to maximize TW
for start_point=1:n
   feasibility=0;
   while feasibility==0
```

```
        [feasibility,tour,spent_time,inspections]=...
            generate_tour_TW(n,I,t,tt,tau,L,W,start_point);
    end
    % saving data into matrices
    tour_length_matrix(n+start_point,1)=length(tour);
    tour_matrix(n+start_point,1:length(tour))=tour;
    spent_time_matrix(n+start_point,1)=spent_time;
    inspections_matrix(n+start_point,:)=inspections;
    fprintf('\n A TW-Tour is generated starting from: %2d', start_point);
end
%% ============= Step 4: Finding Non-Dominated Solutions ============
[objectives,tour_matrix,tour_length_matrix,spent_time_matrix,...
    inspections_matrix]=pareto_front_selection(W,tour_matrix,...
    tour_length_matrix,spent_time_matrix,inspections_matrix);
%% ==================== Step 5: saving final Solution ======================
output=[objectives(:,2),objectives(:,1),tour_length_matrix,tour_matrix];
xlswrite('output_GR.xlsx', output);

%% ======================= Step 6: Drawing figures ========================
scatter(output(:,1),output(:,2),30,'c','MarkerEdgeColor','b','MarkerFaceColor','b');
xlabel('Objective 1: Total Time (TT)','FontSize',10);
ylabel('Objective 2: Total Weight (TW)','FontSize',10);
title('Pareto Front','FontSize',10,'FontWeight','bold');
%%
=========================================================================
```

G.3.1. “create_inputs” function

This function is similar to the one coded for the genetic algorithm. This function reads parameters, which are introduced in Table 21, from the same directory with Matlab files and creates necessary inputs for the Greedy algorithm. First it is asking about the number of tracks (n). The following shows the code for this function.

```
function [n,I,t,tt,tau,L,W,matrix_length]=create_inputs()
fprintf('\n ---> ');
prompt1='What is the number of tracks?';
n=input(prompt1); % Getting number of tracks
%% Importing parameters t,tt,tau,L,W
% t: a Vector that shows Inspection Time
% tt: A matrix that shows Traveling Time between tracks
% tau: a Vector that shows time for consecutive inspection of tracks
% L: a Vector that shows Minimum required inspection for each track
% W: Importance of Tracks

a=xlsread('t.xlsx');
```

```
b=xlsread('tt.xlsx');
c=xlsread('tau.xlsx');
d=xlsread('L.xlsx');
e=xlsread('W.xlsx');

t=a(1:n);
tt=b(1:n,1:n);
tau=c(1:n);
L=d(1:n);
W=e(1:n);
I=(1:n); % set of tracks
matrix_length=ceil(max(t)*sum(L));
% we are using that to create a matrix to store chromosome the matrix has a
% lot of zeros and just a part of each row will be filled by non-zero
% elements
end
```

G.3.2. "generate_tour" functions

There are two different functions generating tours. The whole process that they are following is similar to the repairing after mutation in the genetic algorithm. A feasible gene is added with the minimum inspection plus the traveling time in one function (generate_tour_TT) and with the maximum weight in the other function (generate_tour_TW).

The code for "generate_tour_TT" function is as follows:

```
%% ============================== Input ==================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% L: A vector that shows minimum required inspection for each track
% I: a vector between 1 and n that constrains track set
% Start_point: A track that we want to start inspection from
%% ============================== Output ================================
% feasibility: a binary variable that shows the generated chromosome is feasible or not
% chrom: The created chromosome
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom

function [feasibility,tour,spent_time,inspections]=...
   generate_tour_TT(n,I,t,tt,tau,L,start_point)

T=sum(L)*(max(t)+max(max(tt)));
feasibility=1;
```

```
secondtracktrials=I;
secondtracktrials(start_point)=[];
stt=secondtracktrials(randperm(length(secondtracktrials)));
x=stt(1);
tour=[start_point,x];
inspections=zeros(1,n);
inspections(tour)=inspections(tour)+1;
LST=-tau;
LST(start_point)=0;
LST(x)=t(start_point)+tt(start_point,x);
spent_time=t(start_point)+tt(start_point,x)+t(x);

track=2;
%% ================ While Loop to Create Feasible Chromosome ==============
while max(L-inspections)>0
   y=greedy_selection_TT(n,t,tt,tau,L,T,tour(track),LST,spent_time,inspections); % Creating
second gene
   EST=spent_time+tt(tour(track),y); % updating earliest start time possible
   tour=[tour,y]; % adding the gene
   LST(y)=EST; % updating the latest start time
   spent_time=EST+t(y); % updating the total time spent
   inspections(y)=inspections(y)+1; % updating the actual inpsections
   track=track+1; % next gene

   if isempty(y) % if we have tried every potential gene and could not add, then we are
infeasible; hence, start the chromosome over
      stt(1)=[];
      if isempty(stt)
         % we have possible infeasibility
         feasibility=0;
         tour=[];
         inspections=L;
      else
         % Restart with another stt
         stt=stt(randperm(length(stt)));
         x=stt(1);
         tour=[start_point,x];
         inspections=zeros(1,n);
         inspections(tour)=inspections(tour)+1;
         LST=-tau;
         LST(start_point)=0;
         LST(x)=t(start_point)+tt(start_point,x);
         spent_time=t(start_point)+tt(start_point,x)+t(x);
         track=2;
      end
   end
```

```
end
end
```

The code for “generate_tour_TW” function is as follows:

```
%% ============================== Input ================================
% n: number of tracks
% t: a vector of size n that shows the Inspection Time for each track
% tt: n*n matrix that shows traveling time between tracks
% tau: allowed time between consecutive inspection of the same tracks
% L: A vector that shows minimum required inspection for each track
% I: a vector between 1 and n that constains track set
% Start_point: A track that we want to start inspection from

%% ============================== Output ===============================
% feasibility: a binary variable that shows the generated chromosome is feasible or not
% chrom: The created chromosome
% inspections: Inspection matrix (Which tracks are being inspected and how many times
% spent_time: total time of inspection in a particular solution i.e. chrom

function [feasibility,tour,spent_time,inspections]=...
    generate_tour_TW(n,I,t,tt,tau,L,W,start_point)

feasibility=1;
secondtracktrials=I;
secondtracktrials(start_point)=[];
stt=secondtracktrials(randperm(length(secondtracktrials)));
x=stt(1);
tour=[start_point,x];
inspections=zeros(1,n);
inspections(tour)=inspections(tour)+1;
LST=-tau;
LST(start_point)=0;
LST(x)=t(start_point)+tt(start_point,x);
spent_time=t(start_point)+tt(start_point,x)+t(x);
track=2;
%% ================ While Loop to Create Feasible Chromosome ===============
while max(L-inspections)>0 % this should change % spent_time<=T
    y=greedy_selection_TW(n,tt,tau,L,W,tour(track),LST,spent_time,inspections);
    EST=spent_time+tt(tour(track),y); % updating earliest start time possible
    tour=[tour,y]; % adding the gene
    LST(y)=EST; % updating the latest start time
    spent_time=EST+t(y); % updating the total time spent
    inspections(y)=inspections(y)+1; % updating the actual inpsections
    track=track+1; % next gene
```

```
   if isempty(y) % if we have tried every potential gene and could not add, then we are
infeasible; hence, start the chromosome over
      stt(1)=[];
      if isempty(stt)
         % we have possible infeasibility
         feasibility=0;
         tour=[];
         inspections=L;
      else
         % Restart with another stt
         stt=stt(randperm(length(stt)));
         x=stt(1);
         tour=[start_point,x];
         inspections=zeros(1,n);
         inspections(tour)=inspections(tour)+1;
         LST=-tau;
         LST(start_point)=0;
         LST(x)=t(start_point)+tt(start_point,x);
         spent_time=t(start_point)+tt(start_point,x)+t(x);
         track=2;
      end
   end
end
end
```

G.3.3. “greedy_selection” functions

Similar to "greedy_selection" functions in the genetic algorithm, the functions in this section are "greedy_selection_TT" and "greedy_selection_TW" functions. In these two functions, the search is biased toward less inspected tracks and the search initially ignores the tracks with no remaining required inspections. In order to do so, the functions first choose from gtl with the minimum inspection time plus traveling time and the maximum inspection importance weight. If gtl is empty, a track from gtl is selected.

The code for "greedy_selection_TT" function is as follows:

```
% inputs: gtl, gtd, n, t, tt, tau, LST, chrom
% Output: greedy_track
function greedy_track=...
   greedy_selection_TT(n,t,tt,tau,L,T,track,LST,spent_time,inspections)

COST=T; % set cost a huge number
greedy_track=[];
for i=1:n
```

```
    if i~=track % we have to make sure track is not equal to i
        f=(spent_time+tt(track,i))*min(1,max(0,L(i)-inspections(i))) ...
            -tau(i)-LST(i); % minimim required+gtl+consecutive
        if f>=0
            cost=tt(track,i)+t(i);
            if cost<COST
                greedy_track=i; % select it as the track
                COST=cost; % updating the cost
            end
        end
    end
end
if isempty(greedy_track)
    for i=1:n
        if i~=track % we have to make sure track is not equal to i
            f=spent_time+tt(track,i)-tau(i)-LST(i);
            if f>=0
                cost=tt(track,i)+t(i);
                if cost<COST
                    greedy_track=i; % select it as the track
                    COST=cost; % updating the cost
                end
            end
        end
    end
end
end
```

The code for "greedy_selection_TW" function is as follows:

```
% inputs: gtl, gtd, n, t, tt, tau, LST, chrom
% Output: greedy_track

function greedy_track=...
    greedy_selection_TW(n,tt,tau,L,W,track,LST,spent_time,inspections)
COST=0; % set cost a huge number
greedy_track=[];
for i=1:n
    if i~=track % we have to make sure track is not equal to i
        %dummy=min(1,max(0,L(i)-inspections(i)));
        %f=(spent_time+tt(track,i))*dummy -tau(i)-LST(i);
        f=(spent_time+tt(track,i))*min(1,max(0,L(i)-inspections(i))) ...
            -tau(i)-LST(i);
        if f>=0
            cost=W(i);
            if cost>COST
```

```
            greedy_track=i; % select it as the track
            COST=cost; % updating the cost
        end
    end
  end
end
if isempty(greedy_track)
  for i=1:n
    if i~=track % we have to make sure track is not equal to i
      %dummy=min(1,max(0,L(i)-inspections(i)));
      %f=(spent_time+tt(track,i))*dummy -tau(i)-LST(i);
      f=spent_time+tt(track,i)-tau(i)-LST(i);
      if f>=0
        cost=W(i);
        if cost>COST
          greedy_track=i; % select it as the track
          COST=cost; % updating the cost
        end
      end
    end
  end
end
end
```

G.3.4. "pareto_front_selection" function

Similar to the genetic algorithm, this function finds the non-dominated solutions for the Greedy algorithm. It compares every two solutions between tours. If one solution dominates another in all objectives, then the dominated solution is removed. Then, the function sorts the non-dominated solutions based on their total time. The code is as follows:

```
%% ================ Finding Non-Dominated Set of Solutions ================
function [objectives,tour_matrix,tour_length_matrix,spent_time_matrix,...
  inspections_matrix]=pareto_front_selection(W,tour_matrix,...
  tour_length_matrix,spent_time_matrix,inspections_matrix)

objectives=[inspections_matrix*W',spent_time_matrix];

tour_counter=1;

while tour_counter<=size(objectives,1)-1
  index_check=tour_counter+1;
  while index_check<=size(objectives,1)
    if objectives(tour_counter,1)>=objectives(index_check,1) && ...
          objectives(tour_counter,2)<=objectives(index_check,2)
```

```
        objectives(index_check,:)=[];
        tour_matrix(index_check,:)=[];
        tour_length_matrix(index_check,:)=[];
        inspections_matrix(index_check,:)=[];

    else
        if objectives(tour_counter,1)<=objectives(index_check,1) && ...
                objectives(tour_counter,2)>=objectives(index_check,2)

            objectives(tour_counter,:)=[];
            tour_matrix(tour_counter,:)=[];
            tour_length_matrix(tour_counter,:)=[];
            inspections_matrix(tour_counter,:)=[];
            % we need to update indeces after this. Because removing
            % the first chromosome withou restarting the index of
            % second chromosome can leave some compares unchecked!
            tour_counter=tour_counter-1;
            index_check=size(objectives,1)+1;

        else
            index_check=index_check+1;
        end
    end

  end
  tour_counter=tour_counter+1;
end
spent_time_matrix=objectives(:,2);
end
```

Appendix H. Remedial Actions for Detected Cracks

Table 25 below summarizes the remedial actions based on the length of cracks.

Table 25. Remedial Actions for Crack Sizes

Remedial Action (Office of Railroad Safety, 2011)					
Defect	Length of Defects (inch)		Percent of Head Cross –sectional area weakened by defect		If defective rail is not replaced, take*:
	More than	But not more than	Less than	But not less than	
Transverse fissure Compound fissure			70	5	B
			100	70	A2
				100	A
Detail fracture Engine burn fracture Defective weld			25	5	C
			80	25	D
			100	80	A2 [E and H]
				100	A [E and H]
Horizontal or Vertical split head Split web, Piped rail Head web separation	1	2			H and F
	2	4			I and G
	4				B
	Breakout in railhead				A
Bolt hole crack	$1/2$	1			H and F
	1	$1\ 1/2$			H and G
	$1\ 1/2$				B
	Breakout in railhead				A
Broken Base	1	6			D
	6				A or [E and I]
Ordinary Break					A or E
Damaged Rail					D
Flattened Rail	$Depth \geq 3/8$ and $Length \geq 8$				H

* For details, please refer to Office of Railroad Manual, Chapter 5 (2011).