

Experimental Analyses of Heuristics for Horsefly-type Problems

Gaurish Telang

Contents

	Page
<i>I Overview</i>	4
1 Descriptions of Problems	5
2 Installation and Use	8
<i>II Programs</i>	10
3 Overview of the Code Base	11
3.1 Source Tree	11
3.2 The Main Files	12
3.3 Support Files	14
4 Classic Horsefly	15
4.1 Module Overview	15
4.2 Module Details	15
4.3 Algorithm: Dumb Exact	21
4.4 Algorithm: Greedy - Nearest Neighbor	21
4.5 Algorithm: K2means	22
4.6 Algorithm: Greedy - Incremental Insertion	22
5 Segment Horsefly	23
6 Fixed Route Horsefly	24
7 One Horse, Two Flies	25
8 Reverse Horsefly	26
9 Watchman Horsefly	27

Appendices	28
A Index of Files	29
B Index of Fragments	30
C Index of Identifiers	31
D Man-page for <code>main.py</code>	32

Part I

Overview

Chapter 1

Descriptions of Problems

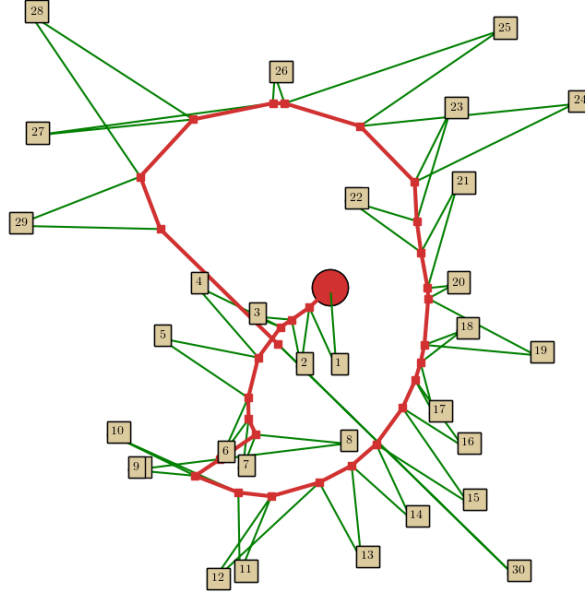


Figure 1.1: An Example of a classic Horsefly tour with $\varphi = 5$. The red dot indicates the initial position of the horse and fly, given as part of the input. The ordering of sites shown has been computed with a greedy algorithm which will be described later

The Horsefly problem is a generalization of the well-known Euclidean Traveling Salesman Problem. In the most basic version of the Horsefly problem (which we call “**Classic Horsefly**”), we are given a set of sites, the initial position of a truck(horse) with a drone(fly) mounted on top, and the speed of the drone-speed φ .^{1 2}

The goal is to compute a tour for both the truck and the drone to deliver package to sites as quickly as possible. For delivery, a drone must pick up a package from the truck, fly to the site and come back to the truck to pick up the next package for delivery to another site.³

¹ The speed of the truck is always assumed to be 1 in any of the problem variations we will be considering in this report.

² φ is also called the “speed ratio”.

³ The drone is assumed to be able to carry at most one package at a time

Both the truck and drone must coordinate their motions to minimize the time it takes for all the sites to get their packages. Figure 1.1 gives an example of such a tour computed using a greedy heuristic for $\varphi = 5$.

This suite of programs implement several experimental heuristics, to solve the above NP-hard problem and some of its variations approximately. In this short chapter, we give a description of the problem variations that we will be tackling. Each of the problems, has a corresponding chapter in Part 2, where these heuristics are described and implemented. We also give comparative analyses of their experimental performance on various problem instances.

Classic Horsefly This problem has already described in the introduction.

Segment Horsefly In this variation, the path of the truck is restricted to that of a segment, which we can consider without loss of generality to be $[0, 1]$. All sites, without loss of generality lie in the upper-half plane \mathbb{R}_+^2 .

Fixed Route Horsefly This is the obvious generalization of Segment Horsefly, where the path which the truck is restricted to travel is a piece-wise linear polygonal path.⁴ Both the initial position of the truck and the drone are given. The sites to be serviced are allowed to lie anywhere in \mathbb{R}^2 . Two further variations are possible in this setting, one in which the truck is allowed reversals and the other in which it is not.

One Horse, Two Flies The truck is now equipped with two drones. Otherwise the setting, is exactly the same as in classic horsefly. Each drone can carry only one package at a time. The drones must fly back and forth between the truck and the sites to deliver the packages. We allow the possibility that both the drones can land at the same time and place on the truck to pick up their next package.⁵

Reverse Horsefly In this model, each site (not the truck!) is equipped with a drone, which fly *towards* the truck to pick up their packages. We need to coordinate the motion of the truck and drone so that the time it takes for the last drone to pick up its package (the “makespan”) is minimized.

Bounded Distance Horsefly In most real-world scenarios, the drone will not be able to (or allowed to) go more than a certain distance R from the truck. Thus with the same settings as the classic horsefly, but with the added constraint of the drone and the truck never being more than a distance R from the truck, how would one compute the truck and drone paths to minimize the makespan of the deliveries?

⁴More generally, the truck will be restricted to travelling on a road network, which would typically be modelled as a graph embedded in the plane.

⁵In reality, one of the drones will have to wait for a small amount of time while the other is retrieving its package. In a more realisting model, we would need to take into account this “waiting time” too.

Watchman Horsefly In place of the TSP, we generalize the Watchman route problem here.

⁶ We are given as input a simple polygon and the initial position of a truck and a drone. The drone has a camera mounted on top which is assumed to have 360° vision. Both the truck and drone can move, but the drone can move at most euclidean distance ⁷ R from the truck.

We want every point in the polygon to be seen by the drone at least once. The goal is to minimize the time it takes for the drone to be able to see every point in the simple polygon. In other words, we want to minimize the time it takes for the drone (moving in coordination with the truck) to patrol the entire polygon.

⁶ although abstractly, the Watchman route problem can be viewed as a kind of TSP

⁷The version where instead geodesic distance is considered is also interesting

Chapter 2

Installation and Use

To run these programs you will need to install Docker, an open-source containerization program that is easily installable on Windows 10¹, MacOS, and almost any GNU/Linux distribution. For a quick introduction to containerization, watch the first two minutes of https://youtu.be/_dfL0zuIg2o

The nice thing about Docker is that it makes it easy to run softwares on different OS'es portably and neatly side-steps the dependency hell problem (https://en.wikipedia.org/wiki/Dependency_hell.) The headache of installing different library dependencies correctly on different machines running different OS'es, is replaced **only** by learning how to install Docker and to set up an X-windows connection between the host OS and an instantiated container running GNU/Linux.

A. [*Get Docker*] For installation instrutions watch

GNU/Linux <https://youtu.be/KCckWweNSrM>

Windows <https://youtu.be/ymlWt1MqURY>

MacOS <https://youtu.be/MU8HUV1JTEY>

B. [*Download customized Ubuntu image*] `docker pull gtelang/ubuntu_customized`
2

C. [*Clone repository*] `git clone gtelang/horseflies_literate.git`

D. [*Mount and Launch*]

For GNU/Linux Open up your favorite terminal emulator, such xterm and then

- Copy to clipboard the output of `xauth list`

¹You might need to turn on virtualization explicitly in your BIOS, after installing Docker as I needed to while setting Docker up on Windows. Here is a snapshot of an image when turning on Intel's virtualization technology through the BIOS: https://images.techhive.com/images/article/2015/09/virtualbox_vt-x_amd-v_error04_phoenix-100612961-large.idge.jpg

²The customized Ubuntu image is approximately 7 GB which contains all the libraries (e.g. CGAL, VTK, numpy, and matplotlib) that I typically use to run my research codes portably. On my home internet connection downloading this Ubuntu-image typically takes about 5 minutes.

- `cd horseflies_literate`
- `docker run -it --name horsefly_container --net=host -e DISPLAY -v /tmp/.X11-unix -`
- `cd horseflies_mnt`
- `xauth add <paste-from-clipboard>`

For Windows I had to follow the instructions in <https://dev.to/darksmile92/run-gui-app-in-linux-Docker-container-on-windows-host-4kde> to be able to run graphical user applications

- E. [*Run experiments*] If you want to run all the experiments as described in the paper again to reproduce the reported results on your machine, then run ³,
`python main.py --run-all-experiments.`

If you want to run a specific experiment, then run
`python main.py --run-experiment <experiment-name>.`

See Index for a list of all the experiments.

- F. [*Test algorithms interactively*] If you want to test the algorithms in interactive mode (where you get to select the problem-type, mouse-in the sites on a canvas, set the initial position of the truck and drone and set φ), run `python main.py --<problem-name>.` The list of problems are the same as that given in the previous chapter. The problem name consists of all lower-case letters with spaces replaced by hyphens.

Thus for instance “Watchman Horsefly” becomes `watchman-horsefly` and “One Horse Two Flies” becomes `one-horse-two-flies`.

To interactively experiment with different algorithms for, say, the Watchman Horsefly problem, type at the terminal `python main.py --watchman-horsefly`

If you want to delete the Ubuntu image and any associated containers run the command ⁴
`docker rm -f horsefly_container; docker rmi -f ubuntu_customized`

That’s it! Happy horseflying!

³ Allowing, of course, for differences between your machine’s CPU and mine when it comes to reporting absolute running time

⁴the ubuntu image is 7GB afterall!

Part II

Programs

Chapter 3

Overview of the Code Base

NOTE: The style of presentation in this chapter has been adapted from Chapter 2 of the Nuweb reference manual <http://nuweb.sourceforge.net/nuweb.pdf>

Almost all of the code has been written in Python 2.7 and tested using the standard CPython implementation of the language. In some cases, calls will be made to external C++ libraries (mostly CGAL and VTK) using SWIG (<http://www.swig.org/>). This is either for speeding up a slow routine or to use a function that is not available in any existing Python package.

Source Tree

```
..  
|-- horseflies.pdf -> tex/horseflies.pdf  
|-- main.py  
|-- scrap.test  
|-- src  
|   |-- expts  
|   |-- lib  
|   |   `-- problem_classic_horsefly.py  
|   |-- Makefile  
|   `-- tests  
|-- tex  
|   |-- directory-tree.tex  
|   |-- horseflies.pdf  
|   |-- horseflies.tex  
|   `-- standard_settings.tex  
`-- webs  
    |-- descriptions-of-problems.web  
    |-- horseflies.web  
    |-- installation-and-use.web
```

```
|-- overview-of-code-base.web
|-- problem-classic-horsefly.web
|-- problem-fixed-route-horsefly.web
|-- problem-one-horse-two-flies.web
|-- problem-reverse-horsefly.web
|-- problem-segment-horsefly.web
|-- problem-watchman-horsefly.web
`-- weave-tangle.sh
```

6 directories, 20 files

There are three principal directories

webs/ This contains the source code for the entire project written in the nuweb format along with documents (mostly images) needed during the compilation of the \LaTeX files which will be extracted from the `.web` files.

src/ This contains the source code for the entire project “tangled” (i.e. extracted) from the `.web` files.

tex/ This contains the monolithic `horseflies.tex` extracted from the `.web` files and a bunch of other supporting \LaTeX files. It also contains the final compiled `horseflies.pdf` (the current document) which contains the documentation of the project, interwoven with code-chunks and cross-references between them along with the experimental results.

The files in `src` and `tex` should not be touched. Any editing required should be done to the `.web` files, which should then be weaved and tangled using the script `weave-tangle.sh` in the `webs` directory.

The Main Files

3.2.1 Each of the files with prefix `problem-*` contain implementations of algorithms for one specific problem. Thus `problem-watchman-horsefly.py` contains algorithms for approximately solving the Watchman Horsefly problem.

All such files are in the directory `src/lib/`

3.2.2 Similarly, each of the files with prefix `expt-*` contain code for testing hypotheses regarding a problem, generating counter-examples or comparing the experimental performance

of the algorithm implementations for each of the problems. Thus `expt-watchman-horsefly.py` contains code for performing experiments related to the Watchman Horsefly problem.

All such files are in the directory `src/expt/`

3.2.3 The file `main.py` in the top-level folder is the *entry-point* for running code. Its only job is to parse the command-line arguments and pass relevant information to the handler functions for each problem and experiment.

Each problem or experiment has a handler routine that effectively acts as a kind of “main” function for that module that does some house-keeping duties by parsing the command-line arguments passed by main, setting up the canvas by calling the appropriate graphics routines and calling the algorithms on the input specified through the canvas.

"../main.py" 13≡

```
import sys
import problem_classic_horsefly as chf
import problem_segment_horsefly as shf
import problem_one_horse_two_flies as oh2f

if __name__=="__main__":
    # Select algorithm or experiment
    if (len(sys.argv)==1):
        print "Specify the problem or experiment you want to run"
        sys.exit()

    elif sys.argv[1] == "--problem-classic-horsefly":
        chf.run_handler()

    elif sys.argv[1] == "--problem-segment-horsefly":
        shf.run_handler()

    elif sys.argv[1] == "--problem-one-horse-two-flies":
        oh2f.run_handler()

    else:
        print "Option not recognized"
        sys.exit()
```

◇

Support Files

3.3.1 These files contain common utility functions that will be useful for manipulating data-structures, common plotting and graphics routines for all horsefly-type problems. All such files have the prefix `utils-*`

All such files are in the directory `src/lib/`

3.3.2 To automate testing of code during implementations, tests for various routines across the entire code-base have been written in files with prefix `test-*`.

Each of the main files have a corresponding test file. Tests for functions in the support files and experimental files have all been implemented in the files `test-utilities.py` and `test-experiments.py` respectively.

All such files are in the directory `src/test/`

Chapter 4

Classic Horsefly

Module Overview

4.1.1 All algorithms to solve the classic horsefly problems have been implemented in the file `problem_classic_horsefly.py`. Here is a high-level view of the module. The `run_handler` function acts as a kind of main function for this module. This function is called from `main.py` which then processes the command-line arguments and runs the experimental or interactive sections of the code.

"../src/lib/problem_classic_horsefly.py" 15a≡

```
⟨ Relevant imports for classic horsefly 15b ⟩  
def run_handler():  
    ⟨ Define key-press handler 16a ⟩  
    ⟨ Set up interactive canvas 20 ⟩  
    ⟨ Algorithms for classic horsefly 21a, ... ⟩  
◇
```

Module Details

4.2.1

⟨ Relevant imports for classic horsefly 15b ⟩ ≡

```
import sys  
import argparse  
import pprint as pp  
import matplotlib.pyplot as plt
```

```

import matplotlib as mpl
from matplotlib import rc
from colorama import Fore
from colorama import Style
import os
import randomcolor
import numpy as np
import classic_horsefly as chf
import algo_utils
import graphics_utils
◇

```

Fragment referenced in [15a](#).

4.2.2 The key-press handler function detects the keys pressed by the user when the canvas is in active focus. This function allows you to set some of the input parameters like speed ratio φ , or selecting an algorithm interactively at the command-line, generating a bunch of uniform or non-uniformly distributed points on the canvas, or just plain clearing the canvas for inserting a fresh input set of points.

⟨ Define key-press handler 16a ⟩ ≡

```

# The key-stack argument is mutable! I am using this hack to my advantage.
def wrapperKeyPressHandler(fig,ax, run):
    def _keyPressHandler(event):
        if event.key in ['i', 'I']:
            ⟨ Start entering input from the command-line 16b ⟩
        elif event.key in ['n', 'N', 'u', 'U']:
            ⟨ Generate a bunch of uniform or non-uniform random points on the canvas 18 ⟩
        elif event.key in ['c', 'C']:
            ⟨ Clear canvas and states of all objects 19 ⟩
    ◇

```

Fragment referenced in [15a](#).

4.2.3

⟨ Start entering input from the command-line 16b ⟩ ≡

```

phi_str = raw_input(Fore.YELLOW + \
    "Enter speed of fly (should be >1): " +\
    Style.RESET_ALL)
phi = float(phi_str)

algo_str = raw_input(Fore.YELLOW + \
    "Enter algorithm to be used to compute the tour:\n Options are:\n" +\
    " (e)  Exact \n" +\
    " (t)  TSP \n" +\
    " (t1) TSP (using approximate L1 ordering)\n" +\
    " (k)  k2-center \n" +\
    " (k1) k2-center (using approximate L1 ordering)\n" +\
    " (g)  Greedy\n" +\
    " (g1) Greedy (using approximate L1 ordering) " +\
    Style.RESET_ALL)

algo_str = algo_str.lstrip()

# Incase there are patches present from the previous clustering, just clear them
clearAxPolygonPatches(ax)

if algo_str == 'e':
    horseflytour =
        run.getTour( chf.algo_dumb,
                    phi )
elif algo_str == 'k':
    horseflytour =
        run.getTour( chf.algo_kmeans,
                    phi,
                    k=2,
                    post_optimizer=chf.algo_exact_given_specific_ordering)
    print " "
    print Fore.GREEN, answer['tour_points'], Style.RESET_ALL
elif algo_str == 'k1':
    horseflytour =
        run.getTour( chf.algo_kmeans,
                    phi,
                    k=2,
                    post_optimizer=chf.algo_approximate_L1_given_specific_ordering)
elif algo_str == 't':
    horseflytour =
        run.getTour( chf.algo_tsp_ordering,
                    phi,

```

```

        post_optimizer=chf.algo_exact_given_specific_ordering)
elif algo_str == 'tl':
    horseflytour =
        run.getTour( chf.algo_tsp_ordering,
                    phi,
                    post_optimizer= chf.algo_approximate_L1_given_specific_ordering)
elif algo_str == 'g':
    horseflytour =
        run.getTour( chf.algo_greedy,
                    phi,
                    post_optimizer= chf.algo_exact_given_specific_ordering)
elif algo_str == 'gl':
    horseflytour =
        run.getTour( chf.algo_greedy,
                    phi,
                    post_optimizer= chf.algo_approximate_L1_given_specific_ordering)
else:
    print "Unknown option. No horsefly for you! ;-D "
    sys.exit()

#print horseflytour['tour_points']
chf.plotTour(ax,horseflytour, run.inithorseposn, phi, algo_str)
applyAxCorrection(ax)
fig.canvas.draw()
◇

```

Fragment referenced in [16a](#).

4.2.4 This chunk generates points uniformly or non-uniformly distributed in the unit square $[0, 1]^2$ in the Matplotlib canvas. I will document the schemes used for generating the non-uniformly distributed points later. These schemes are important to test the effectiveness of the horsefly algorithms. Uniform point clouds do not highlight the weaknesses of sequencing algorithms as David Johnson implies in his article on how to write experimental algorithm papers when he talks about algorithms for the TSP.

⟨ Generate a bunch of uniform or non-uniform random points on the canvas 18 ⟩ ≡

```

numpts = int(sys.argv[1])
run.clearAllStates()
ax.cla()

```

```

    applyAxCorrection(ax)
    ax.set_xticks([])
    ax.set_yticks([])

    fig.texts = []

    import scipy
    if event.key in ['n', 'N']: # Non-uniform random points
        run.sites = algo_utils.bunch_of_random_points(numpts)
    else : # Uniform random points
        run.sites = scipy.rand(numpts,2).tolist()

    patchSize = (xlim[1]-xlim[0])/140.0

    for site in run.sites:
        ax.add_patch(mpl.patches.Circle(site, radius = patchSize, \
            facecolor='blue',edgecolor='black' ))

    ax.set_title('Points : ' + str(len(run.sites)), fontdict={'fontsize':40})
    fig.canvas.draw()
    ◇

```

Fragment referenced in [16a](#).

4.2.5

\langle Clear canvas and states of all objects 19 $\rangle \equiv$

```

    run.clearAllStates()
    ax.cla()

    applyAxCorrection(ax)
    ax.set_xticks([])
    ax.set_yticks([])

    fig.texts = []
    fig.canvas.draw()
    ◇

```

Fragment referenced in [16a](#).

4.2.6

⟨ *Set up interactive canvas 20* ⟩ ≡

```
fig, ax = plt.subplots()
run = chf.HorseFlyInput()
#print run

ax.set_xlim([xlim[0], xlim[1]])
ax.set_ylim([ylim[0], ylim[1]])
ax.set_aspect(1.0)
ax.set_xticks([])
ax.set_yticks([])

mouseClick = wrapperEnterRunPoints (fig,ax, run)
fig.canvas.mpl_connect('button_press_event' , mouseClick )

keyPress = wrapperkeyPressHandler(fig,ax, run)
fig.canvas.mpl_connect('key_press_event', keyPress )
plt.show()
◇
```

Fragment referenced in [15a](#).

Now that all the boring boiler-plate and handler codes have been written, its finally time for algorithmic ideas and implementations! Every algorithm is given an algorithmic overview followed by the detailed steps woven together with the source code.

Algorithm: Dumb Exact

4.3.1 Overview Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

4.3.2 Algorithmic Details

$\langle \textit{Algorithms for classic horsefly 21a} \rangle \equiv$

◇

Fragment defined by [21ab](#), [22ab](#).

Fragment referenced in [15a](#).

Algorithm: Greedy - Nearest Neighbor

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

$\langle \textit{Algorithms for classic horsefly 21b} \rangle \equiv$

◇

Fragment defined by [21ab](#), [22ab](#).

Fragment referenced in [15a](#).

Algorithm: K2means

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

⟨ Algorithms for classic horsefly 22a ⟩ ≡

◇

Fragment defined by [21ab](#), [22ab](#).

Fragment referenced in [15a](#).

Algorithm: Greedy - Incremental Insertion

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

⟨ Algorithms for classic horsefly 22b ⟩ ≡

◇

Fragment defined by [21ab](#), [22ab](#).

Fragment referenced in [15a](#).

Chapter 5

Segment Horsefly

Chapter 6

Fixed Route Horsefly

Chapter 7

One Horse, Two Flies

Chapter 8

Reverse Horsefly

Chapter 9

Watchman Horsefly

Appendices

Appendix A

Index of Files

"../main.py" Defined by [13](#).

"../src/lib/problem_classic_horsefly.py" Defined by [15a](#).

Appendix B

Index of Fragments

- ⟨ Algorithms for classic horsefly [21ab](#), [22ab](#) ⟩ Referenced in [15a](#).
- ⟨ Clear canvas and states of all objects [19](#) ⟩ Referenced in [16a](#).
- ⟨ Define key-press handler [16a](#) ⟩ Referenced in [15a](#).
- ⟨ Generate a bunch of uniform or non-uniform random points on the canvas [18](#) ⟩ Referenced in [16a](#).
- ⟨ Relevant imports for classic horsefly [15b](#) ⟩ Referenced in [15a](#).
- ⟨ Set up interactive canvas [20](#) ⟩ Referenced in [15a](#).
- ⟨ Start entering input from the command-line [16b](#) ⟩ Referenced in [16a](#).

Appendix C

Index of Identifiers

Appendix D

Man-page for `main.py`