NASA HERC Strategic Tool Creation

Josie Caroline Hodges

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
NASA HERC Strategic Tool Creation

by

Josie Caroline Hodges

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

of

The University of Alabama in Huntsville

4/10/2023

Honors Capstone Director: Mr. David Fikes

Lecturer of Mechanical and Aerospace Engineering

Student 4/13/2023

David H Fikes 04/13/2023

Director Date

William Wilkerson

Honors College Dean Date
Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

____________________________
Student Name (printed)

Josie Hodges

____________________________
Student Signature

Josie Hodges

4/13/2023

Date
Table of Contents

Abstract .............................................................................................................................................. 4
Introduction ........................................................................................................................................ 5
Background Research ...................................................................................................................... 6
Programming Outline .................................................................................................................... 9
Program Development ................................................................................................................. 10
Conclusion ...................................................................................................................................... 16
References ...................................................................................................................................... 17
Appendix .......................................................................................................................................... 18
Abstract

The National Aeronautics and Space Administration’s (NASA) Human Exploration Rover Challenge (HERC) includes a half mile course. With various obstacles and tasks along its length, riders of rovers must navigate the course within an eight-minute time limit, assessing the varying point values for each obstacle and task. There are varying point values for each obstacle and task. All challenges along the course are optional as riders must use their best judgment to determine which events are worth the time expense required for an attempt. This judgement or decision presents an optimization problem to be solved for any given competition situation. In order to achieve the best possible outcome, one must end the event with the highest amount of obtained points for a time below the given course time limit. The aim of this project was to create a tool for use in preparing riders in training their brains for optimized decision making. The tool was developed using the high-level programming language, Python. In the future, iterations on this project could be implemented on a rover itself for an instant decision-making addition for the use of its riders.
Introduction

NASA’s HERC Competition is an international competition that takes place every year at the Marshal Space Flight Center in Huntsville Alabama. The University of Alabama in Huntsville has a Senior Design class in which students have the opportunity to construct both a rover and a tool for completing course tasks for use in the competition. Students are given a school-year of preparation time, with a guidebook from NASA, describing the competition requirements. “The primary objective of HERC is for teams of students to design, develop, build, and test human-powered rovers capable of traversing challenging terrain and a task tool for completion of various mission tasks.” [2]

A crucial piece of the HERC competition that can be overlooked is the decision-making process on the actual course. This is long after most of the hands-on engineering work has been completed. Riders must decide upon which obstacles and tasks maximize their point gain, without exceeding the competition time limit. The problem can be solved efficiently with an optimization tool. A programmed tool can take the input of the area of the course a rider has reached and use that information to determine the best course of action regarding which obstacles and tasks to attempt before heading to the end of the course. This engineering problem, paired with statistical analysis of gathered data, ensures the programmed tool design is unique to a specific rover and task tool device.
Background Research

For each obstacle and task on the HERC course there is an assigned point value. Completing the task grants a team those points, however, a team also gains partial credit for simply attempting an obstacle or task. The NASA Guidebook lists the point values for each challenge. When practicing these obstacles and challenges, the University of Alabama in Huntsville’s team determined their rover’s success rate at each obstacle and their task tool’s success rate at each task. The team also measured the average time it took the rover to navigate each obstacle as well as the average time required to complete each task. The below table shows the assigned point values of each challenge, as well as the average time of each challenge and success rate of each challenge gathered from testing.

Table 1. Challenge Point and Time Data

<table>
<thead>
<tr>
<th>Obs/Chal</th>
<th>Description</th>
<th>Points</th>
<th>Time (s)</th>
<th>%Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>Undulating</td>
<td>3</td>
<td>12.9</td>
<td>70</td>
</tr>
<tr>
<td>O2</td>
<td>Crater</td>
<td>3</td>
<td>23.9</td>
<td>60</td>
</tr>
<tr>
<td>O3</td>
<td>Transverse</td>
<td>4</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>O4</td>
<td>Martian Terrain</td>
<td>6</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>O5</td>
<td>Ravine</td>
<td>4</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>T1</td>
<td>Cave Sample</td>
<td>10</td>
<td>102</td>
<td>0.05</td>
</tr>
<tr>
<td>T2</td>
<td>Underground</td>
<td>14</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>O6</td>
<td>Crevasse</td>
<td>5</td>
<td>7.2</td>
<td>70</td>
</tr>
</tbody>
</table>
From the information in Table 1, an expected value can be calculated for each obstacle and task. This provides a clearer representation of the points a team can expect to obtain from attempting a challenge on the course. The expected values of each of the tasks along with the average times provide the information needed to complete an optimization tool to determine the highest combined expected value of tasks for a combined time less than the maximum time allowed. The information discussed is represented in the below table.

**Table 2. Expected Values of Each Challenge Alongside Average Times**

<table>
<thead>
<tr>
<th>Obs/Chal</th>
<th>Description</th>
<th>Time (s)</th>
<th>Exp. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>Undulating</td>
<td>12.9</td>
<td>2.1</td>
</tr>
<tr>
<td>O2</td>
<td>Crater</td>
<td>23.9</td>
<td>1.8</td>
</tr>
<tr>
<td>O3</td>
<td>Transverse</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
An optimization tool is a common requirement in engineering problem solving. “Optimization solvers help improve decision-making around planning, allocating and scheduling scarce resources. They embed powerful algorithms that can solve mathematical programming models, constraint programming and constraint-based scheduling models.” [3] Normally, during the HERC competition, riders rely on their own cognitive flexibility to determine at any given point, the next best course of action. “Cognitive flexibility is important because if a decision maker wanted to achieve both a reasonably high level of accuracy and low effort, he or she...
would have to use a repertoire of strategies, with selection contingent upon situational demands.”

[1] Depending on one’s own cognitive flexibility, however, is unreliable and frequently an inaccurate tool. A prewritten program can run through every possible strategy, compare them, and choose the best strategy that fulfills the requirements of a given problem.

**Programming Outline**

The below table displays the outline used as a basis for developing the HERC competition strategic optimization tool. The table contains a list of the information already provided to the program that can be altered each competition year as NASA alters the requirements of the course. Also shown is a list of information that must be provided as an input to the code in order for it to determine what point in the HERC course a rover has reached. Finally, a list of outputs required from the code is shown. This determines which specific information the tool should provide the HERC team with.

**Table 3. Programming Outline**

<table>
<thead>
<tr>
<th>Information Type</th>
<th>Information Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given</strong></td>
<td>Maximum Time</td>
</tr>
<tr>
<td></td>
<td>Obs/Chal Point Values</td>
</tr>
<tr>
<td></td>
<td>Obs/Chal Average Times</td>
</tr>
<tr>
<td></td>
<td>Obs/Chal Success Rates</td>
</tr>
<tr>
<td></td>
<td>Course Distance</td>
</tr>
<tr>
<td></td>
<td>Average Rover Speed</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Time Elapsed</td>
</tr>
</tbody>
</table>
Program Development

This strategic optimization tool was developed using the programming language, Python. The first step of writing the code was to define every required piece of given information. This information was gathered from the NASA HERC handbook and from data gathered during testing. The below figure shows the handbook information added to the Python script from the 2022-2023 HERC competition.

```
# Constant Variables from Given Information and Gathered Data
TimeMaximum = 8*60 #seconds
TotalDistance = .5*5280 #feet
SpeedAverage = 8.65333 #feet per second
```

**Figure 1.** Given Information from NASA HERC handbook

Taking the success rate information from testing as well as the point values from the NASA HERC handbook, the expected value of each obstacle and challenge were determined
using the below equation. E represents the expected value, $S_{rate}$ represents the success rate, and P represents the point value.

$$E = S_{rate} \times P \quad \text{Eq. [1]}$$

After this calculation was run for each obstacle and challenge, the solutions as well as the average time of each obstacle and challenge were added to the Python script as shown in the below figures.

![Table 1: Average Times](image1)

![Table 2: Expected Values](image2)

**Figure 2.** Average Times From Testing  **Figure 3.** Expected Values from Testing

To prompt riders for information each time they ran the program, a while loop was used to contain the main body of the optimization tool. The loop contained each question requiring an input and the equations used to run calculations based on that information. The below equations were carried out to determine the point reached in the rover course by a rider.
\[ T_{\text{left}} = T_{\text{max}} - T_{\text{elapsed}} \]  
Eq. [2]

\[ D_{\text{left}} = D_{\text{total}} - D_{\text{traveled}} \]  
Eq. [3]

\[ T_{\text{challenges}} = T_{\text{left}} - \frac{D_{\text{left}}}{S_{\text{average}}} \]  
Eq. [4]

The included figure below shows the input section of the Python script within the overall while loop.

```python
#While Loop Setup
a = []
again = "y"
while again == "y":
    print("\nLet’s determine our strategy")
    TimeElapsed = input("How much time has elapsed? (seconds):")
    TimeElapsed = int(TimeElapsed)
    DistanceTraveled = input("How far have you traveled? (feet):")
    DistanceTraveled = int(DistanceTraveled)
    Challenge1 = input("Is obstacle 1 remaining? (y/n):")
    Challenge2 = input("Is obstacle 2 remaining? (y/n):")
    Challenge3 = input("Is obstacle 3 remaining? (y/n):")
    Challenge4 = input("Is obstacle 4 remaining? (y/n):")
    Challenge5 = input("Is obstacle 5 remaining? (y/n):")
    Challenge6 = input("Is task 1 remaining? (y/n):")
    Challenge7 = input("Is task 2 remaining? (y/n):")
    Challenge8 = input("Is obstacle 6 remaining? (y/n):")
    Challenge9 = input("Is obstacle 7 remaining? (y/n):")
    Challenge10 = input("Is task 3 remaining? (y/n):")
    Challenge11 = input("Is obstacle 8 remaining? (y/n):")
    Challenge12 = input("Is task 4 remaining? (y/n):")
    Challenge13 = input("Is task 5 remaining? (y/n):")
    Challenge14 = input("Is obstacle 9 remaining? (y/n):")
    Challenge15 = input("Is obstacle 10 remaining? (y/n):")
```

**Figure 4.** Input Section of Python Script
From the input information, a list was created with every obstacle and task remaining in the course. After this was completed, the equations displayed were used to determine the amount of time riders had left to complete those remaining challenges. To determine which combination of challenge attempts should be used, a nested for loop was initially used to find all possible combinations of challenges with a combined average time less than the time remaining for challenges. This code is shown in the below figure.

```python
p = [[]]
fml = [[],a]

for i in range(len(a)):
    for j in range(i+1,len(a)):
        p[-1].append([i,j])

for i in range(len(a)-3):
    p.append([])
    for m in p[-2]:
        p[-1].append(m+[m[-1]+1])

for i in p:
    for j in i:
        n = []
        for m in j:
            if m < len(a):
                n.append(a[m])
            if n not in fml:
                fml.append(n)

for i in a:
    if [i] not in fml:
        fml.append([i])

#print(fml)  #fml is list of possible combinations
```

**Figure 5.** Code Finding All Possible Combinations of Challenges
In the below figure, the time is calculated for each combination. If that time sum is less than the maximum allotted time for the challenges, it is included in possibilities moving forward. This narrows down the strategic options.

```
for i in fn1:
    if sum(i)<ChallengeTimeLeft:
        fol.append(i)
        fol2.append(i)
        fml.append(sum(i))
```

**Figure 6.** Code Eliminating Unuseable Combinations

The next step in the script is to determine, for all possible combinations, the one which has the maximum combined expected value of points. This is accomplished with the following code where the values inside the main list are replaced with the expected values.

```
Values = []
for i in y:
    Values.append(sum(i))
    #print("Sum of Values of Combinations: ",Values)
Value = max(Values)
    #print(Value)
    number = Values.index(max(Values))
    #print(number)
    Time = fml[number]
```

**Figure 7.** Code Determining Maximum Expected Value Combination

After the optimization piece of the code was completed, the only remaining requirements of the tool were the output delivery of the obstacles and tasks to attempt, the total time left for those challenges, the amount of time those challenges would take, and the expected value of that combination of challenges. The below piece of the Python script in Figure 8, accomplishes this requirement.
Figure 8. Code Printing Output Solutions

After the program was written, testing with the tool was conducted and analyzed to determine whether the optimization was completed correctly. Testing resulted in iterations on the original code until the most beneficial strategy was returned every time. The below figure displays a sample test for a randomly selected point in the rover course.

```python
print("Challenges Riders Should Attempt: ",Solution)  #Solution is the combination
print("Time of Challenges: ",Time," seconds")
print("Expected Point Value of Challenges: ", Value)
#Determine Expected point output!!

again = "n"
```

Figure 9. Python Program Input Prompts with Sample Inputs
The above sample data provides a rider with a clear strategy for a given point in the HERC course. The difference between the time left for the challenges and the time of the challenges also provides a rider with some factor of safety for their strategy in the case of inaccurate testing data.

**Conclusion**

Replacing the uncertainty in a rider’s decision-making on the NASA HERC course, the optimization tool developed for this project provides a clear path to course completion. As the NASA HERC competition has different requirements every year, this tool will need to be altered for future use. An improvement on this tool for future Senior Design students might include a more efficient challenge remaining input method. A secondary thought would be to display the tool on the rover itself for instant decision making on the course.
References


Appendix

Project Complete Python Script

```python
# -*- coding: utf-8 -*-

Created on Tue Apr 4 13:53:20 2023

@author: Josie Hodges

# Constant Variables from Given Information and Gathered Data
TimeMaximum = 8*60 # seconds
TotalDistance = .5*5280 # feet
SpeedAverage = 8.65333 # feet per second

#Average Times
o1 = 12.9
o2 = 23.9
o3 = 20
o4 = 10
o5 = 25
t1 = 102
t2 = 15
o6 = 12.8
o7 = 21.5
t3 = 8.1
o8 = 7.2
t4 = 30.326
t5 = 12.7
o9 = 18.8
o10 = 26.9

#Expected Values
exo1 = 2.1
exo2 = 1.8
exo3 = 0
exo4 = 6
exo5 = 1.6
ext1 = .5
ext2 = 4.2
exo6 = 3.5
exo7 = 3.6
ext3 = 7.5
exo8 = 2
ext4 = 9.6
ext5 = 8.4
exo9 = 2.5
exo10 = 2.4
```
# While Loop Setup

```python
a = []
while again == "y":
    print("\nLet's determine our strategy")
    TimeElapsed = input("How much time has elapsed? (seconds): ")
    TimeElapsed = int(TimeElapsed)
    DistanceTraveled = input("How far have you traveled? (feet): ")
    DistanceTraveled = int(DistanceTraveled)
    Challenge1 = input("Is obstacle 1 remaining? (y/n): ")
    Challenge2 = input("Is obstacle 2 remaining? (y/n): ")
    Challenge3 = input("Is obstacle 3 remaining? (y/n): ")
    Challenge4 = input("Is obstacle 4 remaining? (y/n): ")
    Challenge5 = input("Is obstacle 5 remaining? (y/n): ")
    Challenge6 = input("Is task 1 remaining? (y/n): ")
    Challenge7 = input("Is task 2 remaining? (y/n): ")
    Challenge8 = input("Is obstacle 6 remaining? (y/n): ")
    Challenge9 = input("Is obstacle 7 remaining? (y/n): ")
    Challenge10 = input("Is task 3 remaining? (y/n): ")
    Challenge11 = input("Is obstacle 8 remaining? (y/n): ")
    Challenge12 = input("Is task 4 remaining? (y/n): ")
    Challenge13 = input("Is task 5 remaining? (y/n): ")
    Challenge14 = input("Is obstacle 9 remaining? (y/n): ")
    Challenge15 = input("Is obstacle 10 remaining? (y/n): ")
    if Challenge1 == "y":
        Challenge1 = o1
    if Challenge2 == "y":
        Challenge2 = o2
    a.append(Challenge2)
    if Challenge3 == "y":
        Challenge3 = o3
    a.append(Challenge3)
    if Challenge4 == "y":
        Challenge4 = o4
    a.append(Challenge4)
    if Challenge5 == "y":
        Challenge5 = o5
    a.append(Challenge5)
    if Challenge6 == "y":
        Challenge6 = t1
    a.append(Challenge6)
    if Challenge7 == "y":
        Challenge7 = t2
    a.append(Challenge7)
    if Challenge8 == "y":
        Challenge8 = o6
    a.append(Challenge8)
    if Challenge9 == "y":
        Challenge9 = o7
    a.append(Challenge9)
    if Challenge10 == "y":
        Challenge10 = t3
    a.append(Challenge10)
    if Challenge11 == "y":
```

Challenges11 = o8
  a.append(Challenge11)
if Challenge12 == "y":
  Challenge12 = t4
  a.append(Challenge12)
if Challenge13 == "y":
  Challenge13 = t5
  a.append(Challenge13)
if Challenge14 == "y":
  Challenge14 = o9
  a.append(Challenge14)
if Challenge15 == "y":
  Challenge15 = o10
  a.append(Challenge15)

# Calculations from Given Variables
TimeLeft = TimeMaximum - TimeElapsed
DistanceLeft = TotalDistance - DistanceTraveled
ChallengeTimeLeft = TimeLeft - (DistanceLeft/SpeedAverage)

print(""
print("Time Left for Challenges: ",ChallengeTimeLeft)

p = [[]]
fnl = [[]],a

for i in range(len(a)):
  for j in range(i+1,len(a)):
    p[-1].append([i,j])

for i in range(len(a)-3):
  p.append([])
  for m in p[-2]:
    p[-1].append(m+[m[-1]+1])

for i in p:
  for j in i:
    n = []
    for m in j:
      if m < len(a):
        n.append(a[m])
      if n not in fnl:
        fnl.append(n)

for i in a:
  if [i] not in fnl:
    fnl.append([i])

# print(fnl)      # fnl is list of possible combinations
fol = []
fol2 = []
fml = []
for i in fn1:
    if sum(i)<ChallengeTimeLeft:
        fol.append(i)
        fol2.append(i)
        fml.append(sum(i))
# print("Possible Combinations: ",fol)  # fol and fol2 are list of possible combinations
# print(fml)  # fml is list of combination sums
y = []
for i in fol:
    yy = []
    for n in i:
        if n == 01:
            yy.append(exo1)
        if n == 02:
            yy.append(exo2)
        if n == 03:
            yy.append(exo3)
        if n == 04:
            yy.append(exo4)
        if n == 05:
            yy.append(exo5)
        if n == t1:
            yy.append(exo1)
        if n == t2:
            yy.append(exo2)
        if n == 06:
            yy.append(exo6)
        if n == 07:
            yy.append(exo7)
        if n == t3:
            yy.append(exo3)
        if n == 08:
            yy.append(exo8)
        if n == t4:
            yy.append(exo4)
        if n == t5:
            yy.append(exo5)
        if n == 09:
            yy.append(exo9)
        if n == o10:
            yy.append(exo10)
    y.append(yy)
# print("Exact Values of Combinations: ",y)

Values = []
for i in y:
    Values.append(sum(i))
# print("Sum of Values of Combinations: ",Values)
Value = max(Values)
# print(Value)
number = Values.index(max(Values))
# print(number)
Time = fml[number]
z = []
for i in fol2:
    zz = []
    for n in i:
        if n == o1:
            zz.append("Obstacle 1")
        if n == o2:
            zz.append("Obstacle 2")
        if n == o3:
            zz.append("Obstacle 3")
        if n == o4:
            zz.append("Obstacle 4")
        if n == o5:
            zz.append("Obstacle 5")
        if n == t1:
            zz.append("Task 1")
        if n == t2:
            zz.append("Task 2")
        if n == o6:
            zz.append("Obstacle 6")
        if n == o7:
            zz.append("Obstacle 7")
        if n == t3:
            zz.append("Task 3")
        if n == o8:
            zz.append("Obstacle 8")
        if n == t4:
            zz.append("Task 4")
        if n == t5:
            zz.append("Task 5")
        if n == o9:
            zz.append("Obstacle 9")
        if n == o10:
            zz.append("Obstacle 10")
    z.append(zz)

Solution = z[number]
print("Challenges Riders Should Attempt: ", Solution)  # Solution is the comm
print("Time of Challenges: ", Time, " seconds")
print("Expected Point Value of Challenges: ", Value)
# Determine Expected point output!!!

again = "n"