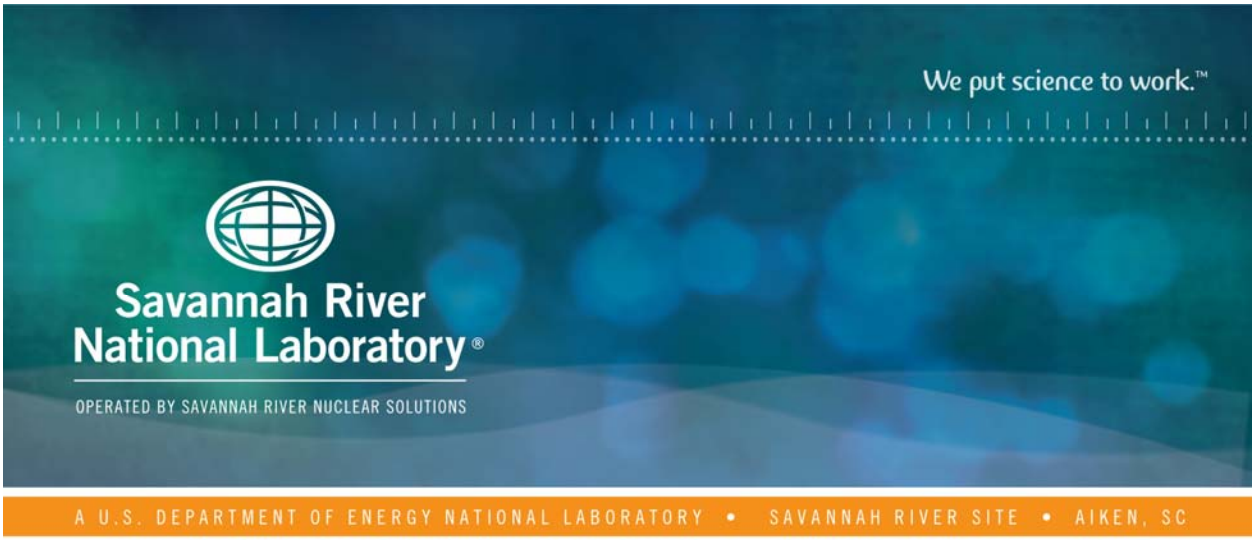
Contract No:

This document was prepared in conjunction with work accomplished under Contract No. DE-AC09-08SR22470 with the U.S. Department of Energy (DOE) Office of Environmental Management (EM).

Disclaimer:



A Monte Carlo Rectangle Packing Algorithm for Identifying Likely Spatial Distributions of Final Closure Cap Subsidence in the E-Area Low-Level Waste Facility

T. L. Danielson

October 2019

SRNL-STI-2019-00440, Revision 0

DISCLAIMER

Printed in the United States of America

Prepared for
U.S. Department of Energy

SRNL-STI-2019-00440
Revision 0

Keywords: *Risk*
Performance Assessment
Stochastic Algorithms

Retention: *Permanent*

A Monte Carlo Rectangle Packing Algorithm for Identifying Likely Spatial Distributions of Final Closure Cap Subsidence in the E-Area Low-Level Waste Facility

T. L. Danielson

October 2019

Prepared for the U.S. Department of Energy under contract number DE-AC09-08SR22470.

Savannah River National Laboratory®
OPERATED BY SAVANNAH RIVER NUCLEAR SOLUTIONS

REVIEWS AND APPROVALS

AUTHORS:

_____

T. L. Danielson, SRNL, Environmental Modeling Date

TECHNICAL REVIEW:

_____

J. A. Dyer, SRNL, Environmental Modeling Date

APPROVAL:

_____

D. A. Crowley, Manager Date
SRNL, Environmental Modeling

_____

A. P. Fellinger, Acting Director Date
SRNL, Environmental Restoration Technologies

ACKNOWLEDGEMENTS

The author acknowledges Solid Waste Management for providing insights into the logistical operations of waste disposal at the E-Area Low-Level Waste Facility.

EXECUTIVE SUMMARY

A Monte Carlo rectangle packing algorithm, summarized in Figure ES-1, has been developed to provide insights into the likely spatial distribution of subsided areas or holes in the final closure cap for the E-Area Low-Level Waste Facility (ELLWF) resulting from the placement of non-crushable waste containers within slit and engineered trench disposal units. Using historical records provided by Solid Waste Management as an indicator of future waste streams, a probability distribution of unique waste containers was constructed, and Monte Carlo sampling was used to sequentially select waste containers for placement in a rectangular region bounded by the disposal unit footprint. Within this Monte Carlo approach, the probability of any container type being selected is weighted by the fractional abundance relative to all other container types. For each Monte Carlo realization, the frequency of non-crushable waste containers appearing at each location on a spatially discretized mesh is updated. After many realizations, the location(s) with the highest frequencies of non-crushable waste containers are regarded as the most likely locations to experience subsidence (i.e., for a hole to form). Utilizing this approach has enabled the down-selection of a computationally intractable set of subsidence hole combinations to be explored in three-dimensional finite element groundwater flow and contaminant transport models. Such an approach can be applied to a number of different packaging problems.

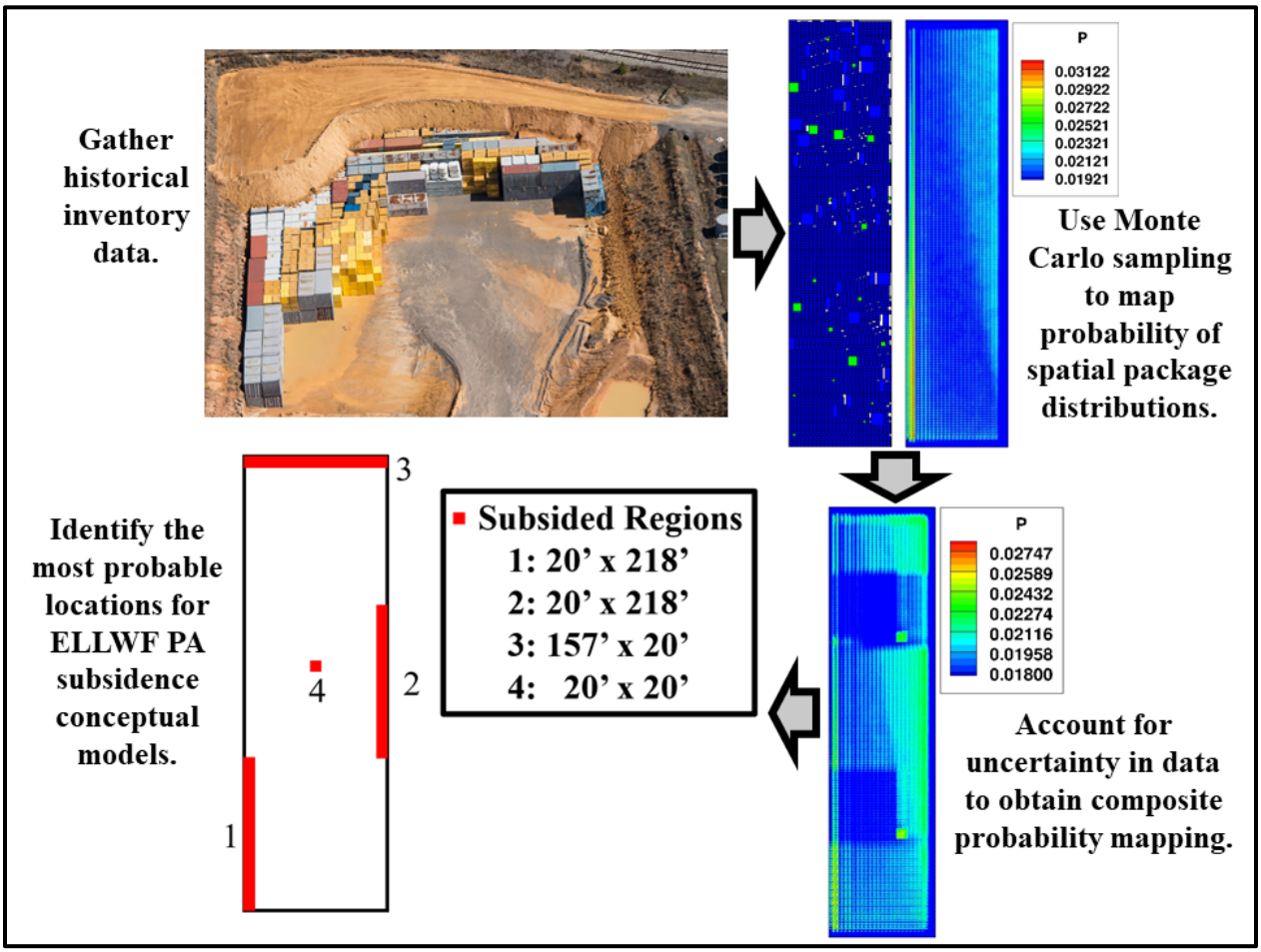


Figure ES-1. Summary of the monte Carlo rectangle packing algorithm workflow.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CP	Crushable package
DOE	Department of Energy
ELLWF	E-Area Low-Level Waste Facility
ET	Engineered Trench
MC	Monte Carlo
NCP	Non-crushable package
PA	Performance Assessment
SRNL	Savannah River National Laboratory
ST	Slit Trench

1.0 Introduction

Per Department of Energy (DOE) Order 435.1-1 (DOE, 1999), a performance assessment (PA) is required to provide reasonable assurance of compliance regarding the potential impacts of a low-level waste facility on water resources a thousand years after institutional controls and maintenance are lifted and final closure of the facility has been carried out. To fully accomplish this objective, it is necessary to construct models that adequately account for the degradation of any engineered barriers. In the context of the E-Area Low-Level Waste Facility (ELLWF), one such degradation scenario manifests from settling, and ultimately subsidence, of portions of the final closure cap covering slit and engineered trenches (STs/ETs) where non-dynamically compacted (or non-crushable) waste containers are located. In locations of the final closure cap where subsidence has occurred, an elevated water infiltration rate is expected to be observed, which more quickly releases contaminants from the waste disposal unit and often results in higher peak doses.

In the next revision of ELLWF PA, groundwater flow and contaminant transport models will be developed using the finite-element method and employ a three-dimensional spatially discretized mesh with water infiltration rates applied as boundary conditions along the top boundary (i.e., ground surface) of the model. In previous studies of the ELLWF, Hamm et al. (Hamm, 2018) showed that for a few key radionuclide species (e.g., Sr-90), the combination of partition coefficients and half-lives is such that a cap-averaged subsided infiltration rate (i.e., a length-weighted average of the intact and subsided regions' infiltration rates) applied along the entire cap boundary produces conservative results (i.e., produces the highest peak flux to the water table). However, in most cases, a discrete hole, where only a well-defined portion of the cap is given a subsided water infiltration rate as a boundary condition (and the rest of the cap is given an intact infiltration rate), yields conservative results. Because it is difficult to determine, *a priori*, the conceptual infiltration model that will produce the most conservative flux to the water table profile for each species in the suite of radionuclides of interest, both implementations are carried out and the most conservative result is used. Therefore, to implement a conceptual model that addresses the case of closure cap subsidence, it is necessary to identify a well-defined spatial location(s) where higher water infiltration rate boundary conditions should be applied.

The methodology for computing bounding infiltration rate estimates for intact and subsided conditions was developed and implemented by Dyer and Flach (Dyer, 2018), but this effort did not include an explicit attempt to account for the spatial distribution of subsidence holes (i.e., size of the hole, number of holes, or location of holes). The portion of the finite-element mesh corresponding to the closure cap might consist of around 300 to 500 elements. Therefore, if only one element makes up a subsided hole location, there would be up to 500 different spatial locations where the hole could exist. While this number of simulations is not necessarily limited by computational resources, a multi-subsided-hole investigation would have exponentially more combinations rendering any logical identification of best-estimate or bounding cases computationally intractable.

Access to historical data on typical crushable and non-crushable waste streams entering the ELLWF allows for the construction of probability distributions and, therefore, utilization of a stochastic approach to understand the spatial distribution, size, and number of subsided holes in the final closure cap. The following sections describe the development of a Monte Carlo rectangle packing algorithm designed to provide insights for developing conceptual models that will be used to investigate subsidence scenarios in the next revision of the ELLWF PA.

2.0 Monte Carlo Rectangle Packing Algorithm

To illustrate the abstract problem that will be approached by this algorithm, consider a two-dimensional square grid with uniform spacing, l, bounded by a square with a side length, L, and an infinite inventory of only one rectangle type that has a length and width that match the grid spacing. If rectangles are selected

from the inventory at random for sequential placement along the discrete locations of the grid, it can be said that there will be exactly L^2 rectangles after L^2 placements with no grid space left unfilled. Now consider that 25% of the inventory is made up of a second rectangle with side length, *2l,* and width, *l*. Rectangles are again selected at random for sequential placement where the probability of selection is weighted by the percentage of the inventory made up by that rectangle. One possible final configuration (i.e., once no more rectangles can be placed) is shown in Figure 2-1. Notably, with a non-square rectangle present in the distribution, orientation is also a factor and is governed by the surrounding rectangles that have already been placed. If no neighboring constraints exist, the rectangle can be placed in either direction with equal probability. However, as is illustrated in Figure 2-2, certain configurations remove one possible orientation, and in some cases, the possibility of placing the package altogether. After analyzing many final configurations, it may be that some grid locations are occupied by a specific rectangle-type more frequently than others, thereby providing insights to an average (or most probable) final configuration. Adding more rectangles to the distribution, as will be done in the current work, creates even more possible configurations. Monte Carlo sampling is an ideal technique for analyzing the configuration space of such a problem.

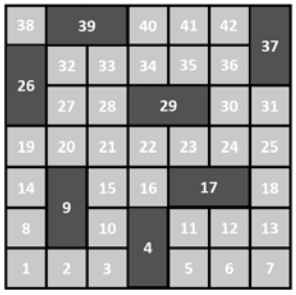


Figure 2-1. One possible configuration of rectangles where the number indicates the order in which the rectangle was selected and placed.

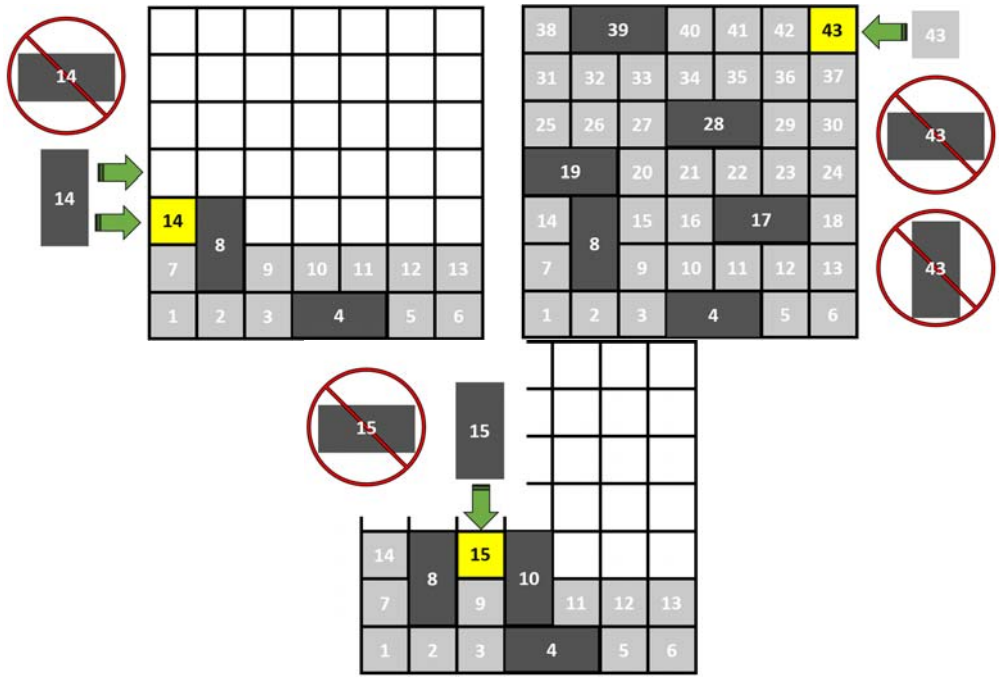


Figure 2-2. Configurations where the orientation of the selected rectangle is dictated by the neighboring rectangles. The yellow squares indicate the current grid location where a rectangle is to be placed.

The Monte Carlo (MC) technique is applied by randomly selecting events, each having a probability of occurrence, p_i, such that:

$$\sum_{i=1}^{N} p_i = 1 \tag{2.1}$$

for N unique event types. In the context of the current rectangle packing application, events correspond to the sequential selection of rectangular objects from an inventory of rectangular object types, each having a unique length, l, and width, w, and the subsequent non-overlapping placement inside a region defined by a larger rectangle. Note that two rectangle types in the inventory may have the same area but a different overall areal footprint (i.e., different length and width), and are therefore considered unique. The probability, p_b, of any unique rectangular object being placed is weighted by its relative abundance in the inventory such that:

$$p_b = \frac{n_b A_b}{\sum_{i=1}^{N} n_i A_i} = \frac{n_b A_b}{A_T} \tag{2.2}$$

where n_b is the number of rectangular objects of type b in the inventory, A_b is the area of rectangular objects of type b, and A_T is the summed total area of all rectangular objects in the inventory.

The placement of rectangles continues until the entire space is filled or no more rectangles in the inventory will fit in any remaining open space, such that:

$$\sum_{i=1}^{N} A_i \approx A_R \tag{2.3}$$

where the summation is taken across all rectangular objects that have been placed inside of the large rectangle with area A_R.

In the following subsections, this mathematical formulation will be applied to the problem of placing rectangular crushable and non-crushable waste packages in STs and ETs of the ELLWF.

2.1 Algorithm

The current application is concerned with identifying possible subsidence hole distributions across the final closure cap in the ELLWF. The MC algorithm relies on a distribution of rectangular packages, each having a unique areal footprint defined by its length, l, and its width, w. Because subsidence is most likely to occur in areas where non-crushable packages are placed, two additional package classifiers, crushable (CP) and non-crushable (NCP), are used. Starting from Eqn. (2.1):

$$P_C + P_{NC} = 1 \tag{2.4}$$

where P_C and P_{NC} are the probability of selecting a crushable and non-crushable package, respectively. Then, expressing CPs and NCPs as discrete probability distributions of different package types in the sets N_C and N_{NC}:

$$P_C = \sum_{i=1}^{N_C} p_i \tag{2.5}$$

and

$$P_{NC} = \sum_{j=1}^{N_{NC}} p_j \tag{2.6}$$

so that,

$$\sum_{i=1}^{N_C} p_i + \sum_{j=1}^{N_{NC}} p_j = 1 \tag{2.7}$$

where p_i and p_j are the probabilities for selecting crushable package i and non-crushable package j, respectively.

Substituting Eqn. (2.2) into Eqn. (2.7) and rearranging, the number of each package type in the inventory distribution is obtained from the relationship:

$$\sum_{i=1}^{N_C} n_i A_i + \sum_{j=1}^{N_{NC}} n_j A_j = A_T \tag{2.8}$$

If the algorithm is sampling the package distributions appropriately, the following conditions should, on average, hold true across any number of MC realizations:

$$\frac{\sum_{i=1}^{N_C} n_i A_i}{A_T} \approx P_C \tag{2.9}$$

$$\frac{\sum_{j=1}^{N_{NC}} n_j A_j}{A_T} \approx P_{NC} \tag{2.10}$$

The inputs and assumptions for assembling the CP and NCP distributions are discussed in subsection 2.2.

The MC algorithm begins by reading a specified input file containing the data sets of CPs and NCPs, where each package is specified as a length, width, a tag, "C" or "NC," for CPs and NCPs, and a total number of that package type in the distribution. Note that for non-square packages, a default orientation of the rectangular package is also embedded in the data point at the time of reading.

Once the inventory has been gathered, a user-specified number, N_R, of MC realizations begins. At the start of each MC realization, an empty (i.e., no packages have been placed) dictionary of occupied locations is initialized. The large rectangular region representing the ST or ET is subdivided with a user-specified resolution along the length, L, and width, W, such that all grid points along the length and width fall on the intervals $[0, \frac{L}{R}]$ and $[0, \frac{W}{R}]$, where R is the resolution. In general, each grid point is initialized with a numerical value of one, signifying no occupation. A package is selected by subdividing the inventory dictionary into an ordered list of interval bounds based on the total number of each package type as $[X_1, X_2, ..., X_N]$ such that

$$X_1 = B_1 \tag{2.11}$$

and

$$X_N = \sum_{j=1}^{N-1} X_j + B_N \tag{2.12}$$

where B_N is the total number of package type N. An integer, r, is selected at random from the interval $[1, X_N]$ such that if

$$X_{M-1} + 1 \le r \le X_M \tag{2.13}$$

then the selected package belongs to type *M*, which has a unique length, *l*, and width, *w*. To determine if the package can be placed, the algorithm iterates through each row comparing the total sum in the row to the grid-normalized package width (*w*/*R*). The package can be placed if

$$\sum_{i=0}^{\frac{W}{R}} f(i,j) \geq \frac{w}{R} \tag{2.14}$$

where *j* is the position of the row on the interval $[0,\frac{L}{R}]$, *i* is the position along the row that falls on the interval $[0,\frac{W}{R}]$, $f(i,j)$ is one if the current grid location is not occupied by a package and zero if it is occupied by a package, *w* is the width of the package, and *R* is the grid resolution.

If (2.14) is satisfied, the algorithm iterates through the row, computing a sequential sum in search of a location where

$$\sum_{i}^{i+\frac{w}{R}-1} f(i,j) = \frac{w}{R} \tag{2.15}$$

If a location is found where (2.15) is satisfied, the algorithm computes the sum across all grid coordinates that the package area would span to check the condition

$$\sum_{j}^{j+\frac{l}{R}-1} \sum_{i}^{i+\frac{w}{R}-1} f(i,j) = \frac{lw}{R^2} \tag{2.16}$$

If (2.16) is also satisfied, the package is placed at location (i,j) and each grid coordinate on the interval from (i,j) to $(i+\frac{w}{R}-1, j+\frac{l}{R}-1)$ is given a value of 0 and added to an occupations dictionary with a flag to indicate whether the package is a crushable or a non-crushable package.

If any condition expressed by (2.14), (2.15), or (2.16) is not satisfied, the package cannot fit in its default orientation; therefore, the package is rotated by 90 degrees so that each of the conditions can be checked again as:

$$\sum_{i=0}^{\frac{L}{R}} f(i,j) \geq \frac{l}{R} \tag{2.17}$$

$$\sum_{i}^{i+\frac{l}{R}} f(i,j) = \frac{l}{R} \tag{2.18}$$

$$\sum_{j}^{j+\frac{w}{R}-1} \sum_{i}^{i+\frac{l}{R}-1} f(i,j) = \frac{lw}{R^2} \tag{2.19}$$

If any condition expressed by (2.17), (2.18), or (2.19) is not satisfied, the algorithm continues to the next row and begins checking the criteria from (2.14) through (2.19) until a suitable location is found. Conversely, if the criteria fail in all rows, the selected package will not fit anywhere in the trench. If the package is unable to fit in the trench, it is removed from the inventory. To avoid significant skewing of the inventory statistics, if greater than 90% of the total area represented by all packages in the inventory is removed as a result of a failed package placement, the realization is stopped, and the occupations dictionary

is returned as output. After each MC realization, the occupations of all grid locations are collected so that the frequency that any location is occupied by an NCP is compiled across all realizations.

It is important to sample the distribution enough times such that the sum of all MC realizations is statistically representative of the most likely configurations. The minimum number of Monte Carlo realizations that should be performed can be estimated based on the number of realizations for each package to have an opportunity to be placed on each grid location at least one time. Eqn. (2.2) gives the probability of selecting any package from the distribution, and in any given realization, the total area of the ST or ET occupied by any given package is proportional to the probability of selecting the package. Therefore, the number of any type of package per realization is estimated as:

$$n = \frac{A_{Trench} p_b}{A_b} \tag{2.20}$$

where A_{Trench} is the area of the ST or ET, p_b is the probability of selecting the package, and A_b is the area of the package. Knowing the number of grid points, the number of realizations for each package to have the opportunity to be placed on each grid location at least one time is:

$$N_R = \frac{nA_{Trench}}{R^2} \tag{2.21}$$

where R is the grid resolution. The maximum of all calculated numbers of realizations for each package type should be used.

The full source code for the MC rectangle packing algorithm is provided in Appendix A.

2.2 Assumptions and Inputs

An inventory list of all non-crushable packages that have been placed (or are to be placed) in the ELLWF, and the area associated with each package, has been provided by Solid Waste Management and is shown in Appendix B. While the inventory list provides the area, it does not provide the length and width of the packages and the specific dimensions were not readily available. Therefore, non-crushable packages are initially assumed to be square and have been binned based only on the area of the package giving 14 unique package types with a total areal footprint of 13,446 ft^2. Table 2-1 shows the assumed percentage of the distribution of all crushable waste types that has been used when considering STs or ETs. Estimations of the different waste type percentages for ETs and STs are based on Phifer (2010) and communications with Solid Waste Management.

Table 2-1. Percentage breakdown of crushable waste types.

Trench Type	Percent Bulk Waste	Percent Boxed Waste	B-25 Percentage of Boxed Waste	Sealand Percentage of Boxed Waste	Percentage of Sealands that are 20' long	Percentage of Sealands that are 40' long
ET	5%	95%	95%	5%	95%	5%
ST	67%	33%	95%	5%	95%	5%

For the next revision of the ELLWF PA, Dyer (Dyer, 2019) identified several relevant "percent subsidence scenarios" based on the number and size of non-crushable containers located in ST/ETs. Each of these scenarios was investigated by the MC rectangle packing algorithm to identify how the spatial distribution changes as a function of the percentage of the distribution made up of non-crushable containers. The areal footprint of the total distribution made up of crushable and non-crushable containers is shown in Table 2-2.

The ten percent non-crushable scenario was not identified as a relevant case by Dyer (2019) but was still tested as a bounding scenario.

Table 2-2. Total areal footprints of non-crushable and crushable packages making up the waste distribution.

Percent Subsidence	Non-Crushable Waste Areal Footprint (ft^2)	Crushable Waste Areal Footprint (ft^2)
0.54%	13,446	2,476,554
2%	13,446	658,854
3.6%	13,446	360,054
4.9%	13,446	260,962
10%	13,446	121,014

The assumed dimensions of each waste type are shown in Table 2-3 and have been rounded to the nearest whole number so that the grid resolution can be expressed in whole feet.

Table 2-3. Assumed dimensions of crushable waste types in the distribution.

Bulk Waste Dimensions		***B-25 Dimensions***		***20' Sealand Dimensions***		***40' Sealand Dimensions***	
Length (ft)	**Width (ft)**	**Length (ft)**	**Width (ft)**	**Length (ft)**	**Width (ft)**	**Length (ft)**	**Width (ft)**
20	20	4	6	9	20	9	40

Based on the percentage of each waste type that is to be represented in the distribution, Eqn. (2.9) was used to calculate the number of waste packages for each trench type. The number of each type of package represented in the distributions for STs and ETs is shown in Table 2-4.

Along with the package distributions and dimensions, the dimensions of the rectangular trench (e.g., ET or ST) in which the packages are placed influences the packing of the packages. In the current work, an ET's dimensions are 157 feet wide by 656 feet long and a ST's dimensions are 20 feet wide by 656 feet long (notably, this would only correspond to one trench segment of an actual ST).

Table 2-4. Total square footage and number of each crushable package type represented in the package inventory distributions for each percentage subsidence in STs and ETs.

Percent Subsidence	Trench Type	Waste Type	Total Area (ft^2)	Number
0.54%	*ET*	Bulk	123,827.7	310
		B-25	2,235,089.985	93,129
		20’ Sealand	111,754.499	621
		40’ Sealand	5,881.816	16
	ST	Bulk	1,651,036	4,128
		B-25	784,242.1	32,677
		20’ Sealand	39,212.105	218
		40’ Sealand	2,063.795	6
2%	*ET*	Bulk	32,942.7	82
		B-25	594,615.735	24,776
		20’ Sealand	29,730.787	165
		40’ Sealand	1,564.778	4
	ST	Bulk	439,236	1,098
		B-25	208,637.1	8,693
		20’ Sealand	10,431.855	58
		40’ Sealand	549.045	2
3.6%	*ET*	Bulk	18,002.7	45
		B-25	324,948.735	13,540
		20’ Sealand	16,247.437	90
		40’ Sealand	855.128	4
	ST	Bulk	240,036	600
		B-25	114,017.1	4,751
		20’ Sealand	5,700.855	32
		40’ Sealand	300.045	1
4.9%	*ET*	Bulk	13,048.108	33
		B-25	235,518.352	9,813
		20’ Sealand	11,775.918	65
		40’ Sealand	619.785	2
	ST	Bulk	173,974.776	435
		B-25	82,638.018	3,443
		20’ Sealand	4,131.901	23
		40’ Sealand	217.468	1
10%	*ET*	Bulk	6,050.7	15
		B-25	109,215.135	4,551
		20’ Sealand	5,460.757	30
		40’ Sealand	287.408	1

3.0 Analysis of Results

In the following subsections, the validation of the algorithm is presented, and a base case was selected and compared with several test cases to explore how the percentage of NCPs in the inventory, the shape of NCPs, and the trench geometry impact the likely spatial distribution of subsidence for ETs and STs. One million MC realizations were performed for the base case and between 100,000 and 500,000 MC realizations were used for all subsequent test cases, which proved sufficient to establish relevant trends.

3.1 Verification and Validation of the Monte Carlo Rectangle Packing Algorithm

A simple test of the algorithm was performed to ensure, in an ideal situation, that the expected results are obtained. A 20-foot-long by 30-foot-wide region was subdivided into one-foot by one-foot grid cells. An inventory distribution with two NCPs and 98 CPs, each one-foot by one-foot, was supplied as input to the MC algorithm and 100,000 realizations were performed. A contour map showing the probability that any grid location is occupied by an NCP is shown in Figure 3-1. As expected, no free space remained in any final configuration, the heat map is almost perfectly uniform, and the mean non-crushable occupancy of the fictitious trench is 2.002% (i.e., ~0.1% different from the expected 2% value), indicating that MC sampling is being performed correctly.

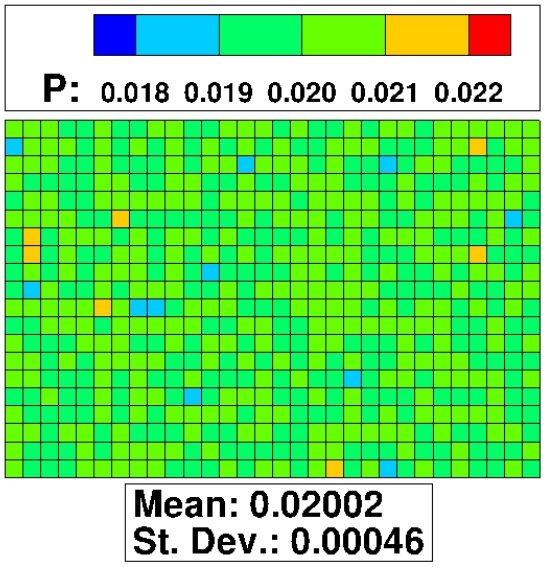


Figure 3-1. Contour map of the probability of grid locations being occupied by a non-crushable package.

3.2 Base Case

The two percent subsidence scenario with square non-crushable packages was selected as the base case and used as a benchmark for comparing and understanding how the shape and percentage of NCPs impact the likely spatial distribution of subsidence. Initially, one million MC realizations were performed for the ST and ET geometries. The final configurations from one realization performed within each geometry are shown in Figure 3-2. Based on historical photos from the ELLWF photo vault, the algorithm appears to perform well at producing realistic package placement patterning that might result from the Solid Waste Area's standard operations.

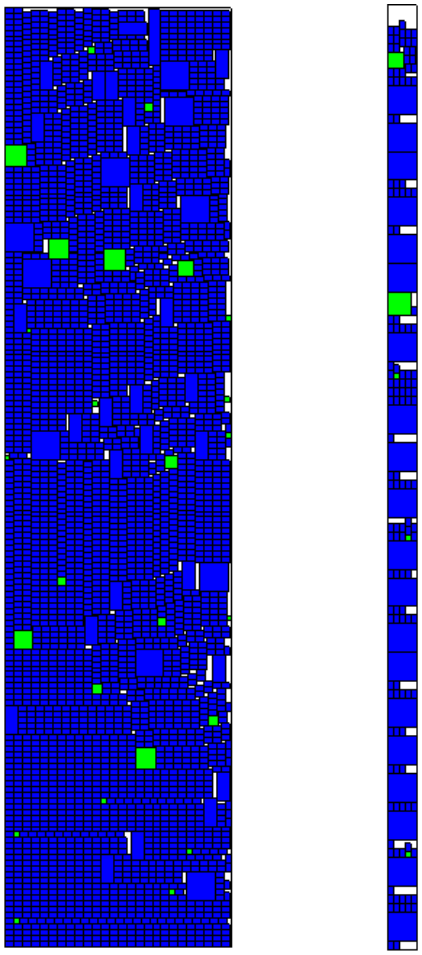


Figure 3-2. Final ET (left) and ST (right) configuration of a single Monte Carlo realization. Blue rectangles are crushable packages, green rectangles are non-crushable packages, and white indicates free space where no package is placed.

The frequency at which a non-crushable package is positioned on any given grid cell across the total number of MC realizations is an indicator of spatial regions that are most likely to receive NCPs. Figure 3-3 shows contour maps of the probability that any grid location is occupied by an NCP across one million realizations of filling the ET and ST geometries. The mean areal percentage of the total ET footprint occupied by NCPs was within ~5% of the target percent subsidence across the one million realizations, even without perfect packing (i.e., there is free space left in the final configuration). Packing was less efficient (i.e., more free space left in the final configuration) in STs due to the high probability of selecting bulk waste; however, the mean occupancy was still within 10%, offering further validation that MC sampling is adequately representing the statistics even without perfect packing. The most frequently occupied cells in the ET geometry lie along the left-most ~20 feet of the trench with a tendency toward the lower left corner. The most frequently occupied grid cells in the ST geometry are located near the bottom edge, yet the frequency of occupations is relatively uniform across the entire length.

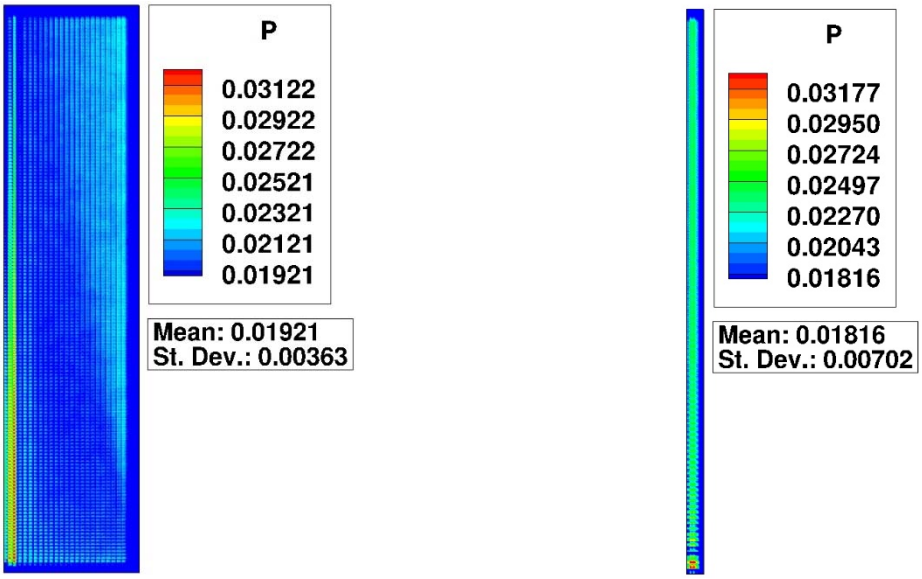


Figure 3-3. Contour map of the probability that any grid location is occupied by a non-crushable package across one million Monte Carlo realizations for ET (left) and ST (right).

3.3 Alternative Percent Non-Crushable Test Cases

The inventory distributions for the four additional percent non-crushable cases shown in Table 2-4 were tested to understand how the percentage of the inventory distribution that is non-crushable might change the spatial distribution of subsidence. Contour maps showing the probability of any grid location being occupied by an NCP are shown in Figure 3-4 through Figure 3-7 for the ET and ST geometries. Notably, changing the percentage of NCPs in the inventory distribution introduces no significant qualitative changes to the spatial distribution of NCPs, as can be seen by comparing each scenario to Figure 3-3.

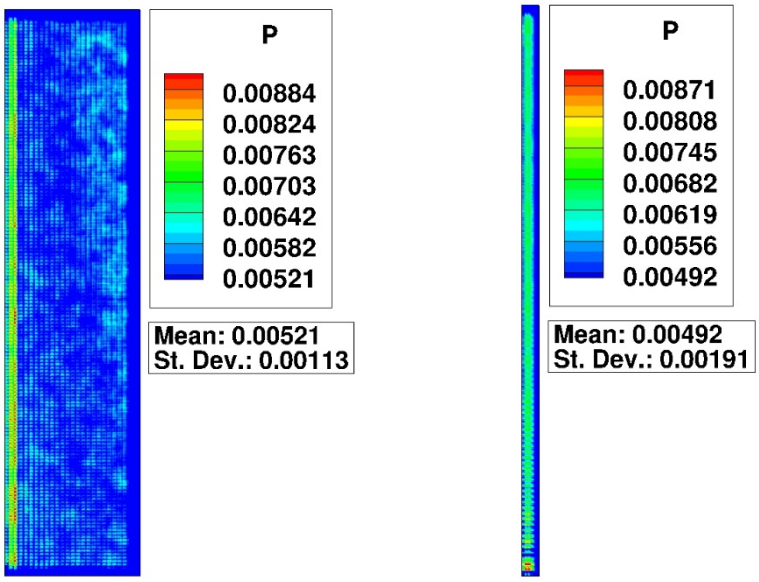


Figure 3-4. Contour map of the probability that any grid location is occupied by a non-crushable package with 0.54 percent non-crushable packages in the inventory distribution for ET (left) and ST (right).

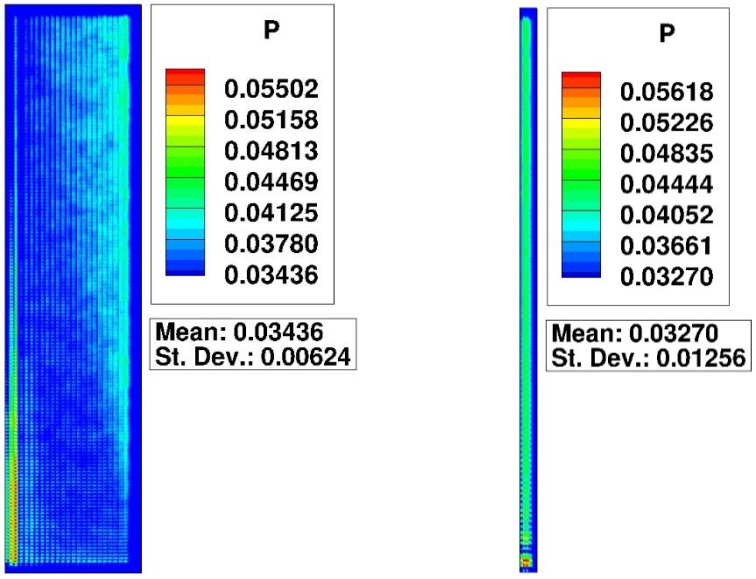


Figure 3-5. Contour map of the probability that any grid location is occupied by a non-crushable package with 3.6 percent non-crushable packages in the inventory distribution for ET (left) and ST (right).

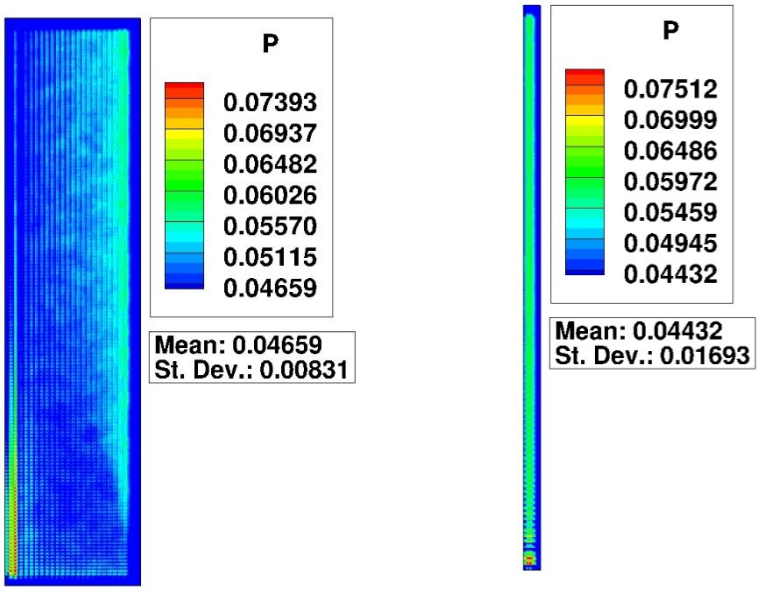


Figure 3-6. Contour map of the probability that any grid location is occupied by a non-crushable package with 4.9 percent non-crushable packages in the inventory distribution for ET (left) and ST (right).

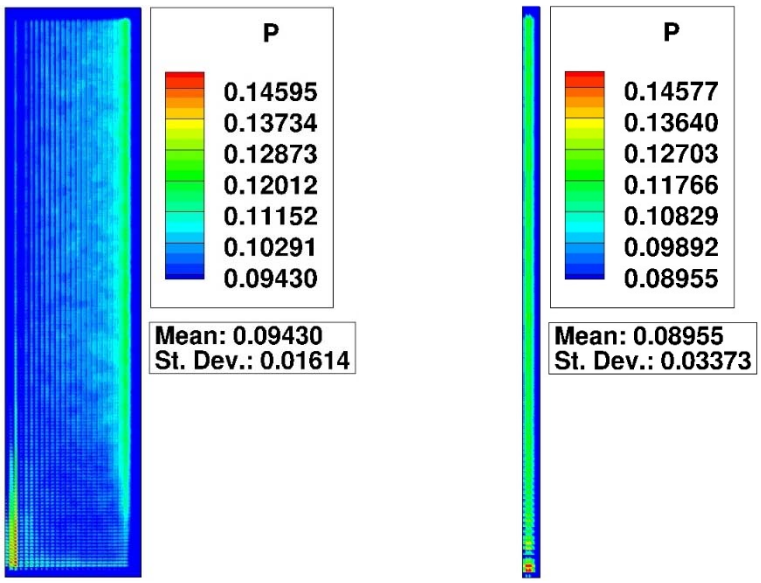


Figure 3-7. Contour map of the probability that any grid location is occupied by a non-crushable package with 10 percent non-crushable packages in the inventory distribution for ET (left) and ST (right).

3.4 Effects of Package Shape and Orientation

As discussed in Section 2.0, the shape of the packages can influence the packing pattern in the trench. Because the length and width of the NCPs were not provided (only the total areal footprint), there is uncertainty in the non-crushable package dimensions. Therefore, to understand how non-square packages might influence the packing and the likely spatial distribution of NCPs, two additional tests were performed where the NCPs were stretched along their axes to give an aspect ratio of 4:1 and 1.778:1. As an example of how this might impact the results, when given a 4:1 aspect ratio for placement in STs, some NCPs become wider than the trench in one direction and, therefore, the orientation of these packages will be dictated by their dimensions. The contour maps of the probability of any grid location being occupied by an NCP within the ET and ST geometries are shown in Figure 3-8 and Figure 3-9. Notably, as the NCPs become more elongated, the most likely placement increasingly moves to the top boundary of the trench for the ET geometry. Once again, no significant changes were observed for the ST geometry.

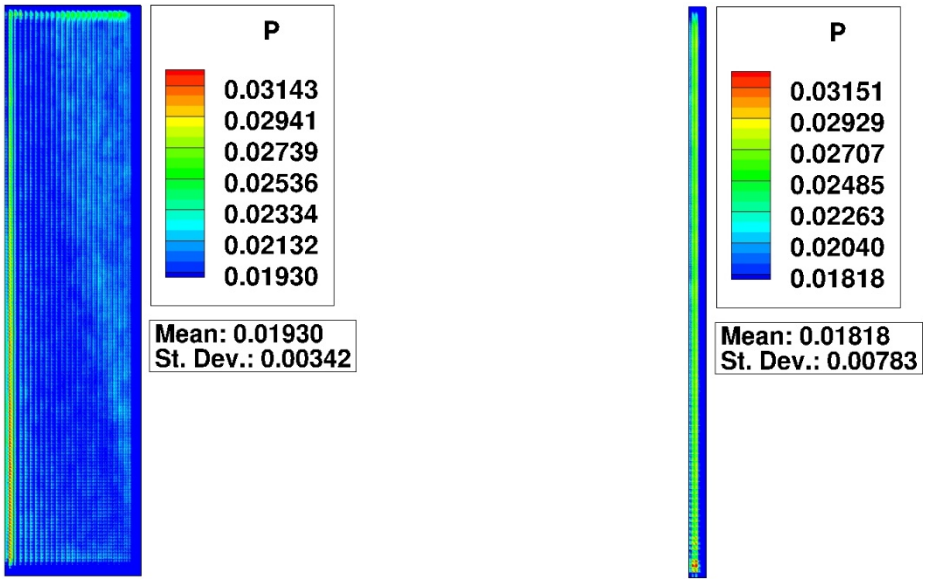


Figure 3-8. Contour map of the probability that any grid location is occupied by a non-crushable package when non-crushable packages are given a 1.778:1 aspect ratio and the long edge is oriented parallel to the length (long side) of the trench for ET (left) and ST (right).

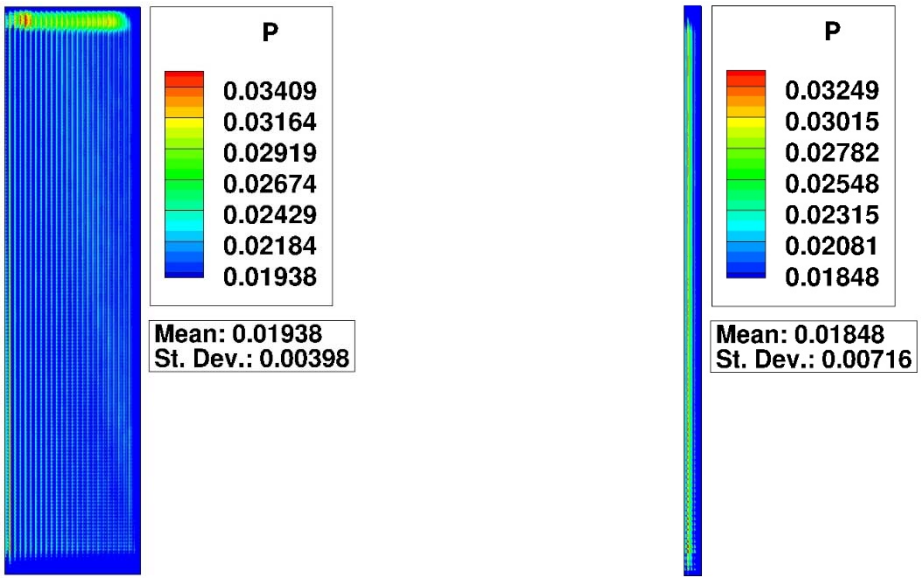


Figure 3-9. Contour map of the probability that any grid location is occupied by a non-crushable package when non-crushable packages are given a 4:1 aspect ratio and the long edge is oriented parallel to the length (long side) of the trench for ET (left) and ST (right).

The rectangular packages in the inventory distribution that are supplied as input to the MC rectangle packing algorithm have a default orientation where Figure 3-8 and Figure 3-9 have the long edge oriented parallel to the length (long side) of the trench. As discussed in Section 2.1, the algorithm tries to first place the package in its default orientation, and if that is not possible, only then is the package rotated. Therefore, to discover if there are any biasing effects introduced by having a default orientation, the elongated NCPs were rotated by 90 degrees. The contour maps of the probability that any location is occupied by an NCP in the rotated configuration are shown in Figure 3-10 and Figure 3-11.

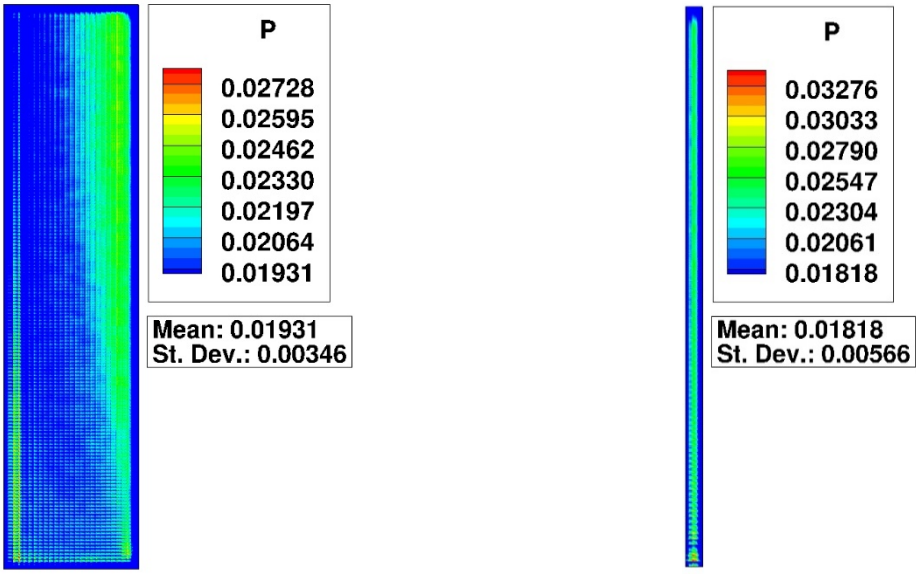


Figure 3-10. Contour map of the probability that any grid location is occupied by a non-crushable package when non-crushable packages are given a 1.778:1 aspect ratio and the long edge is oriented parallel to the width (short side) of the trench for ET (left) and ST (right).

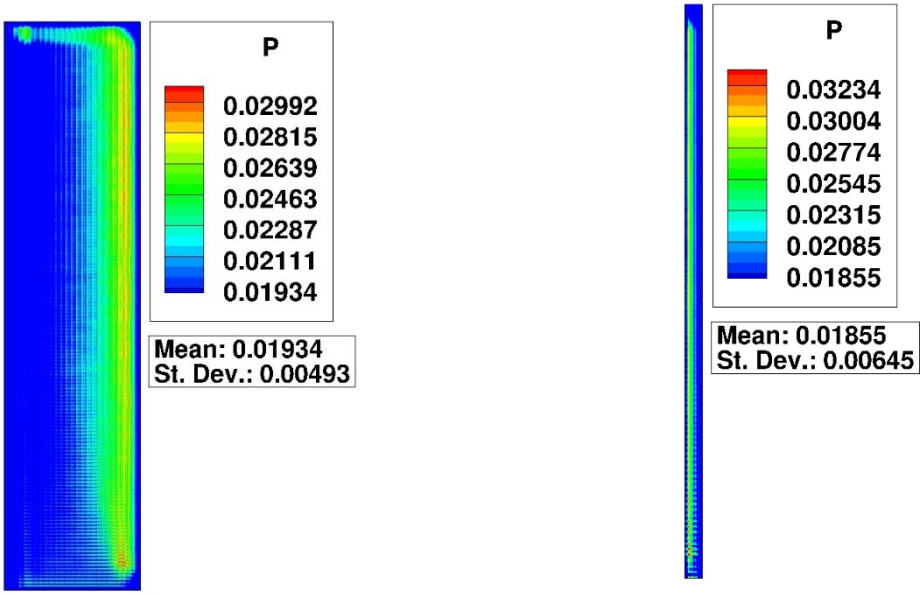


Figure 3-11. Contour map of the probability that any grid location is occupied by a non-crushable package when non-crushable packages are given a 4:1 aspect ratio and the long edge is oriented parallel to the width (short side) of the trench for ET (left) and ST (right).

Notably, there are no significant impacts for the ST geometry, while the highest probability locations shift from the top edge to the right edge for the ET geometry. To understand this result, consider the number of possible ways any individual ET grid location can become occupied as a function of the inventory. That is, each grid location can be occupied by any package from the inventory that does not extend beyond either boundary from that location. In addition, locations can be occupied by a package that was placed on a location below or to the left (i.e., previously placed packages) and whose dimensions overlap the location of interest. This number for each grid location can change as a function of the inventory (package shapes,

sizes, orientations, etc.) and location in the trench. Such a calculation for the ET geometry is shown in Figure 3-12 through Figure 3-14. Note the qualitative similarities to the MC simulations. When NCPs are specified as square, the number of ways a package can be placed on a grid cell is relatively uniform throughout, with a slight biasing toward the top of the trench. Only a slight deviation from the square box scenario was found when applying a 1.778:1 aspect ratio. Stretching the boxes to 4:1 leads to a substantial difference and matches well with the MC results.

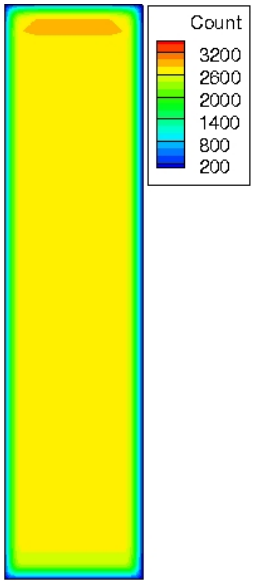


Figure 3-12. Number of possible ways each grid cell within the ET geometry can become occupied by any package from an inventory where NCPs are square-shaped.

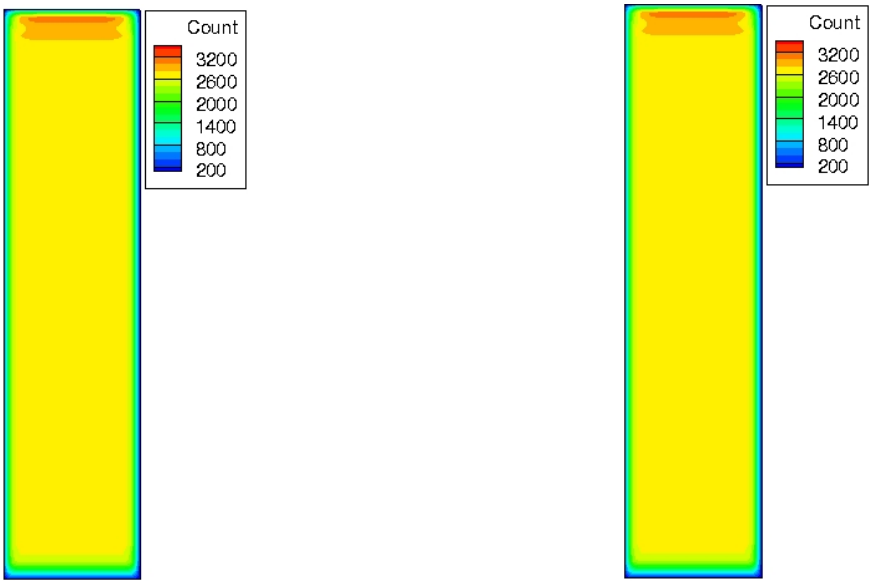


Figure 3-13. Number of possible ways each grid cell within the ET geometry can become occupied by any package from an inventory where NCPs are given an aspect ratio of 1.778:1 and the long edge is oriented parallel to the length of the trench (left) and the width of the trench (right).

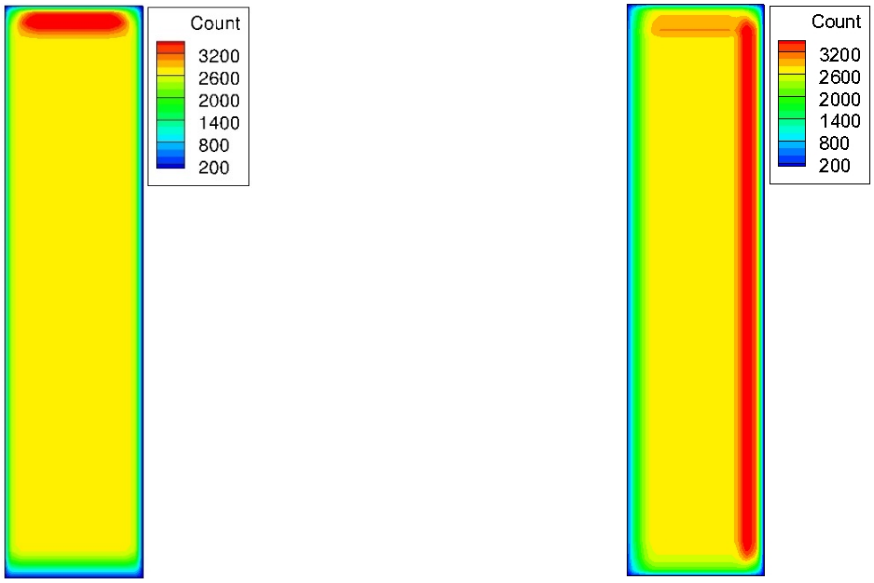


Figure 3-14. Number of possible ways each grid cell within the ET geometry can become occupied by any package from an inventory where NCPs are given an aspect ratio of 4:1 and the long edge is oriented parallel to the length (left) of the trench and the width of the trench (right).

Using this same technique for the ST geometry shows no significant changes for any of the inventory modifications. Because this calculation only considers the dimension of packages in the inventory, and not the relative quantities (or probability of selection) of each type of package, this implies that the primary factor governing the most likely spatial distributions of subsidence in the ST geometry is the trench geometry itself.

The final test case concerning orientation accounts for the B-25 boxes in the inventory. Because B-25 boxed waste is represented by a non-square rectangle and makes up a large percentage of both ET and ST inventories, the default orientation of B-25s was rotated 90 degrees. This produced insignificant changes to either geometry's probability contour map, and therefore bias is not introduced through the default orientation of B-25 boxes.

3.5 Impact of Trench Geometry

So far, this work has considered a ST to be only 20 feet wide (i.e., a single trench within the 157-foot wide multi-trench ST disposal unit). The ST design includes five 20-foot wide trench segments, creating an overall footprint that is approximately the same as an ET. For the next revision of the ELLWF PA, it is desirable to have a generic model geometry that represents both ETs and STs. Waste distribution and trench dimensions are significantly different between a ST and an ET (i.e., a ST contains a much greater fraction of bulk waste than boxed waste); therefore, a MC simulation was performed where the ET geometry was instead filled assuming the waste inventory distribution for a ST. A contour map showing the probability that any grid cell is occupied by an NCP is shown in Figure 3-15, where the most probable spatial locations of NCPs are along the bottom and right edges. Figure 3-16 and Figure 3-17 highlight that assuming a different aspect ratio and rotating the default orientation do not introduce any significant change in the most probable spatial distribution of NCPs. This is likely because the bulk waste form dominates the inventory distribution and, therefore, plays a dominant role in impacting the spatial distribution of NCPs.

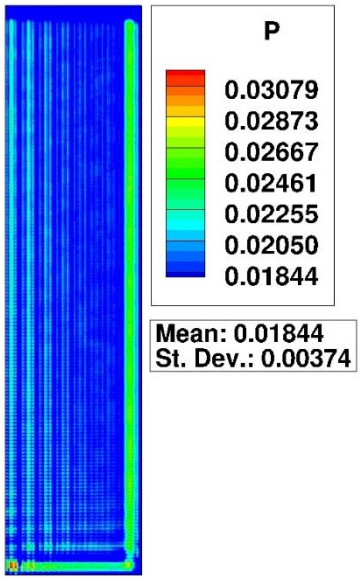


Figure 3-15. Contour map of the frequency that a non-crushable package occupied each grid location across 500,000 Monte Carlo realizations when the inventory distribution for a ST was used to fill the ET geometry.

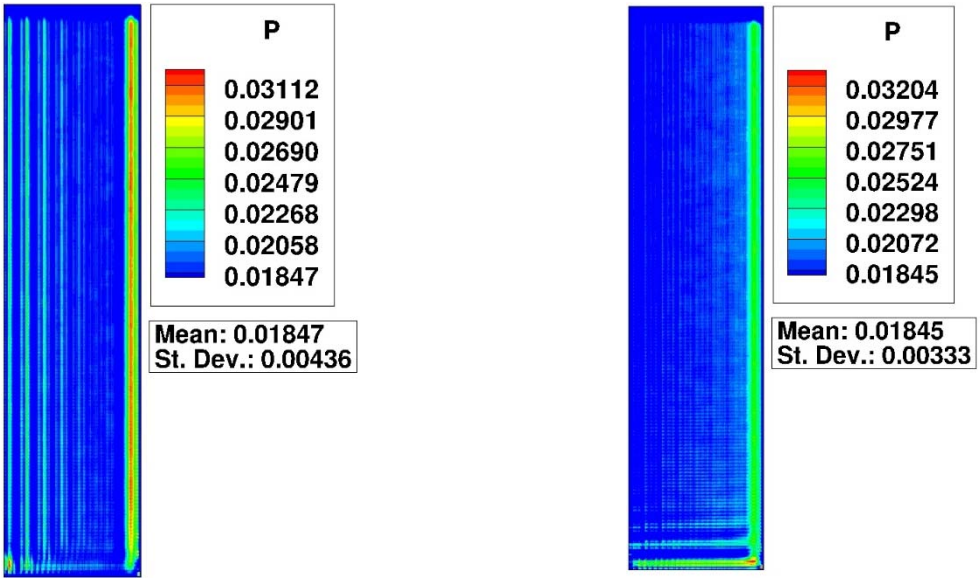


Figure 3-16. Contour map of the frequency that a non-crushable package occupied each grid location when the inventory distribution for a ST was used to fill the ET geometry and NCPs are given a 1.778:1 aspect ratio and default orientation with the length of the NCPs oriented parallel to the long edge of the ET (left) and the short edge of the ET (right).

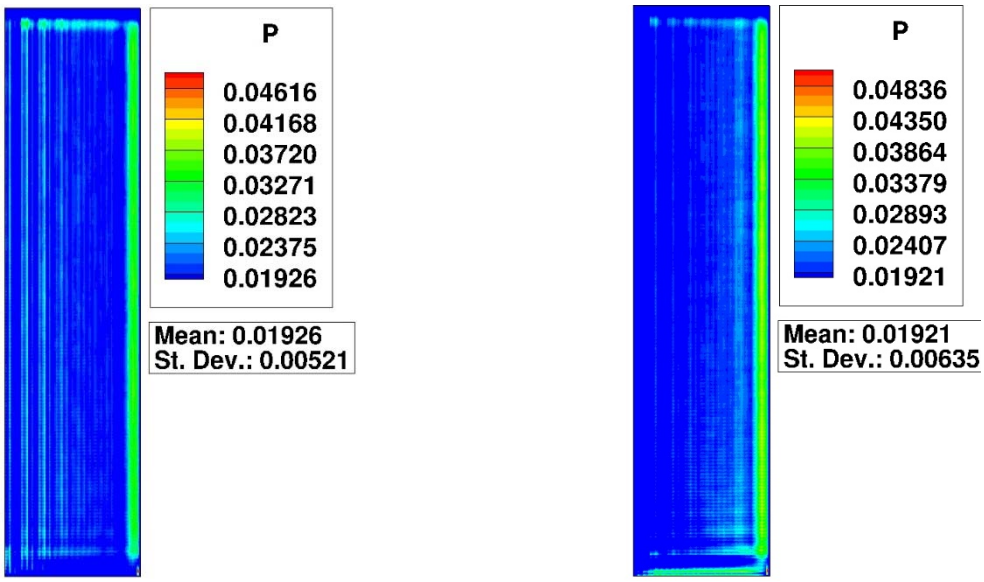


Figure 3-17. Contour map of the frequency that a non-crushable package occupied each grid location when the inventory distribution for a ST was used to fill the ET geometry and NCPs are given a 4:1 aspect ratio and default orientation with the length of the NCPs oriented parallel to (left) the long edge of the ET and (right) the short edge of the ET.

The next set of test cases will explore the possibility of an irregularity in the central portions of the trench geometry. For example, ETs in the ELLWF occasionally use sump pumps to assist with drainage, which could effectively alter the geometry of the trench because a non-negligible portion of the trench would no longer be occupied by waste packages, or the filling pattern could be temporarily altered. Because the sump pump can influence the pattern of package placement, a MC simulation was initialized with a 100-foot by 100-foot section occupied to represent the presence of a sump pump. Each of the NCP geometries (i.e., square, elongated, rotated elongated) were tested for three different sump pump locations. The probability that different grid locations in the ET are occupied is shown in Figure 3-18. Interestingly, for any of the inventory NCP geometries, the most likely location always fell in nearly the same spatial region neighboring the sump pump with a probability that is greater than a factor of four higher than the mean. Notice that adding a sump pump effectively adds another edge to the geometry and, based on all prior tests, NCPs have a higher probability of falling along edges due to their irregular size. This is partially attributed to the algorithm's filling methodology, which is intended to mimic standard operations. As the algorithm reaches the region with the sump pump, if the package cannot fit, it moves to the next row and the next open location where it can be placed.

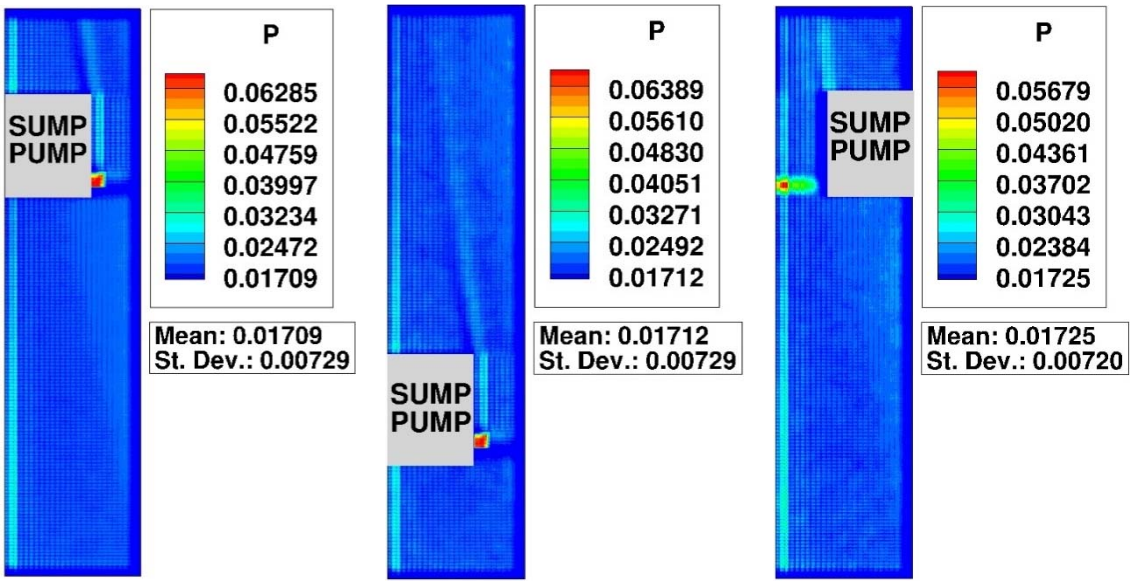


Figure 3-18. Contour map of the probability that an NCP occupies grid locations in the ET geometry when a sump pump is present in the trench.

3.6 Composite Results

The MC rectangle packing algorithm was utilized to investigate several test cases, each varying in the inventory distribution or trench geometry. Modifying the shape (i.e., aspect ratio and default orientation) of NCPs produced the most substantial differences in the most probable spatial distribution when compared to altering the percentage of NCPs. Because there is uncertainty embedded in the shape of past and future NCPs, the probability contour maps for several cases were combined to produce a composite probable spatial distribution of NCPs as shown in Figure 3-19. The composite includes the following test cases (all derived from the two percent NCP base case):

- Base case
- NCPs elongated with 1.778:1 aspect ratio
- NCPs elongated with 4:1 aspect ratio
- NCPs elongated with 1.778:1 aspect ratio and default orientation rotated
- NCPs elongated with 4:1 aspect ratio and default orientation rotated
- ST inventory placed in ET geometry with square packages
- B-25 boxed waste default orientation rotated
- All sump pump cases

In general, the most probable locations fall along the edges and corners of the ET geometry. In addition, the sump pump test cases also show an increased probability of finding NCPs in central regions of the trench.

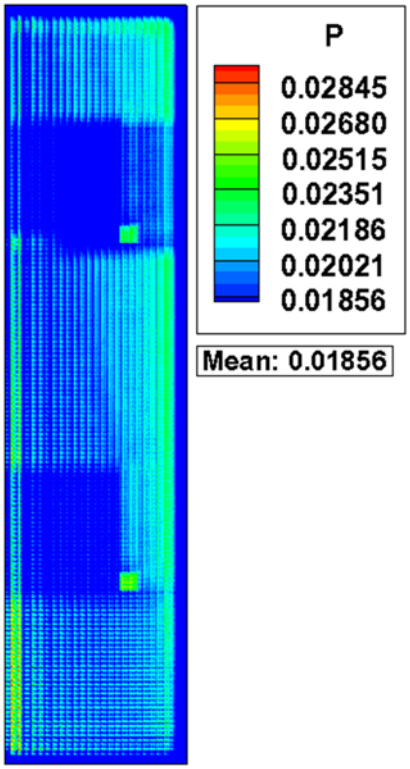


Figure 3-19. Composite heat map of the probability that an NCP is located at any given location in the ET geometry.

4.0 Conclusions

The MC rectangle packing algorithm developed for this study used historical ELLWF inventory data for NCPs disposed in ETs and STs to generate the most probable spatial distributions of those packages. In the process, new insights were gained that will assist in developing conceptual models for subsidence in the next revision of the ELLWF PA. While several assumptions were made, the algorithm appears to reproduce, in a reasonable way, a logistical filling pattern that may result from normal operations of Solid Waste Management. Several test cases revealed that the most probable spatial distribution of NCPs within an ET footprint resides along the edges of the trench. In contrast to the 157-foot-wide ET, the 20-foot-wide ST geometry was shown to be essentially insensitive to any modifications of the waste inventory distribution.

5.0 Recommendations, Path Forward, and Future Work

For the next revision of the ELLWF PA, the MC rectangle packing algorithm can be applied in multiple ways. One such way of applying the algorithm may be to select several subsidence cases based on the composite probability heat map of those cases identified as most likely. Using the proposed composite probability heat map that is shown in Figure 3-19 and simplifying for the symmetry of the ET geometry, one possible proposed list of relevant subsidence cases is shown in Figure 5-1. This proposed list of subsidence geometries accounts for all high probability subsidence locations around the edges of the disposal unit and the possibility of finding centrally located subsided holes (e.g., as were manifested in the "sump pump" test cases). If new information is obtained pertaining to NCPs in the ELLWF inventory, a new set of test cases should be evaluated to investigate any differences that may arise.

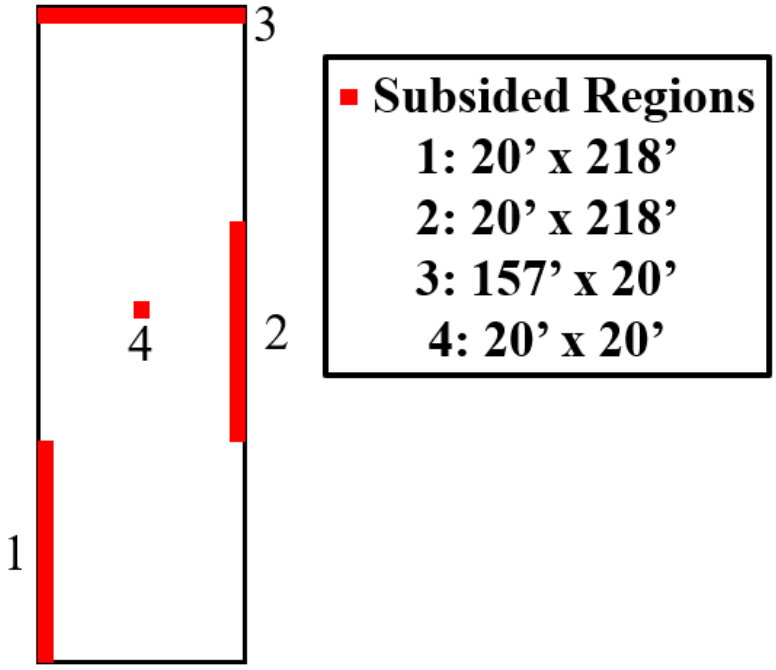


Figure 5-1. Relevant subsidence cases based on the inputs and assumptions of the current work.

Another potential application of the algorithm involves post-processing of flux to the water table profiles from subsidence geometries that were selected based on additional data that includes the package placement date of each NCP (i.e., geometries of interest that may or may not have been identified as likely from the MC algorithm). Such a data set, coupled with the opening and closing dates of each disposal unit, would provide insights to the possible location of the NCPs and whether they tend to arrive individually or in groups. For example, if a package (or set of packages) was placed halfway through the operational period of a trench, one might assume that the package is located somewhere in the middle of the trench. Once the subsidence geometries have been selected and groundwater radionuclide transport simulations have been performed for each case to produce flux to water table profiles for each subsidence geometry, the probability heat maps can provide weighting factors for combing flux to the water table results. Such a methodology would account for all relevant subsidence scenarios, while giving the most weight to the most likely of the selected scenarios. As an example of how this weighting methodology could be applied, consider the recommended cases in Figure 5-1 and the composite probability heat map shown in Figure 3-19. Assume that the average of the probability values in each of the four regions corresponds to the average probability values shown in Table 5-1. The flux weighting factors in the table are computed as:

$$W_i = \frac{P_i}{\sum_{i=1}^{N} P_i} \tag{5.1}$$

where, W_i is the weighting factor for subsided region i, P_i is the average probability of subsided region i (taken from the composite probability heat map), and the denominator is the summation taken across the average probability of each of the N subsided regions. Using this method, the radionuclide flux to the water table resulting at each time step from each subsided region would be blended to obtain a blended flux to the water table profile as:

$$F(t) = \sum_{i=1}^{N} W_i f_i(t) \tag{5.2}$$

where, W_i is the weighting factor for the subsided region i, $f_i(t)$ is the flux to the water table at time t resulting from a simulation that considers the subsided region i, and the summation is taken across all N subsided regions.

Table 5-1. Average probabilities and flux weighting factors for the subsided regions shown in Figure 5-1.

Subsided Region from Figure 5-1	Average Probability	Flux Weighting Factor
1	0.027	0.290323
2	0.022	0.236559
3	0.020	0.215054
4	0.024	0.258065

6.0 References

DOE, 1999. "Radioactive Waste Management Manual of 7-09-99", DOE M 435.1-1, U.S. Department of Energy, Washington, D.C. July 1999.

Dyer, 2018. J. A. Dyer and G. P. Flach, "Infiltration Time Profiles for E-Area LLWF Intact and Subsidence Scenarios", SRNL-STI-2018-00327, Rev. 0, Savannah River National Laboratory, Aiken, SC, 29801, July, 2018.

Dyer, 2019. J. A. Dyer, "Infiltration Data Package for the E-Area Low-Level Waste Facility Performance Assessment", SRNL-STI-2019-00363, Rev. 0. Savannah River National Laboratory, Aiken, SC (DRAFT).

Hamm, 2018. L. L. Hamm, S. E. Aleman, T. L. Danielson, and B. T. Butcher, "Special Analysis: Impact of Updated GSA Flow Model on E-Area Low-Level Waste Facility Groundwater Performance", SRNL-STI-2018-00624, Rev. 0, Savannah River National Laboratory, Aiken, SC, 29801, December, 2018.

Phifer, 2010. M. A. Phifer, "Slit Trench Waste Representation", SRNL-L6200-2010-00018, Savannah River National Laboratory, Aiken, SC, 29801, June, 2010.

Appendix A. Source Code.

```
import random
from multiprocessing import Pool
import subprocess
import os
import timeit

def read_data(filename):
    """Read inventory (box dimensions and C/NC) from a file."""
    #Dictionary for storing the number of each package type.
    inventory_dict = {}
    with open(filename, 'r') as f:
        for line in f:
            x_dim = float(line.split()[0])
            y_dim = float(line.split()[1])
            C_NC = line.split()[2]
            #Check if package type is already in dictionary.
            try:
                #Increment count
                inventory_dict[(x_dim, y_dim, C_NC)] += 1
            except KeyError:
                #Add package type to dictionary.
                inventory_dict.update({(x_dim, y_dim, C_NC):1})

    return inventory_dict



def read_data2(filename):
    """Read inventory (box dimension, C/NC, n box type) from a file"""
    inventory_dict = {}
    with open(filename, 'r') as f:
        for line in f:
            x_dim = float(line.split()[0])
            y_dim = float(line.split()[1])
            C_NC = line.split()[2]
            n_box = int(line.split()[3])
            inventory_dict.update({(x_dim, y_dim, C_NC):n_box})
    return inventory_dict



def select_package(inventory_dict):
    """Randomly select a package from the inventory."""
    #Create a sequential list of numbers based on the number of
    #each type of package
    packages = []
    package_range_max = 0
    for key in inventory_dict.keys():
        package_range_max += inventory_dict[key]
        packages.append([package_range_max, key])


    #Select random number between 1 and the max val in packages
    p = random.randint(1, packages[-1][0])

    #Find the package corresponding to the random number
    for k in range(len(packages)):
        if packages[k][0] >= p:
            package = packages[k][1]
            break

    return package
```

```
def MC_realization(inventory_dict, DU_length, DU_width, resolution):
    """Select a crushable or non-crushable package of a given size
       and place it in an open location.  Crushable vs. non-crushable
       probability is based on the percent subsidence input and the
       package size is based on the distribution of package sizes.
    """
    #Make a copy of the inventory dictionary so that it remains unchanged
    #for the next realization.
    inventory_dict = dict(inventory_dict)
        #Keep track of the total area of the inventory as a reference if a
        #package type is removed.
    total_area_initial = 0
    total_area = 0
    for i in range(len(inventory_dict.keys())):
        area = (inventory_dict.keys()[i][0] *
                inventory_dict.keys()[i][1] *
                inventory_dict[inventory_dict.keys()[i]])
        total_area_initial += area
        total_area += area
    initial_len = len(inventory_dict)
    initial_num = sum(inventory_dict.values())
    finish = timeit.default_timer()

    #Dictionary to track if a package is occupying the current grid cell.
    occupations = {}

    #Total area occupied by crushable/non-crushable packages at each iteration.
    non_crush_area = 0
    crush_area = 0

    #Keeps record of all sequential placements -- keys = count,
    #values = lowest left corner to top right corner of package -- and is used
    #for writing tecplot zones to file in tecplot_tec_mcr.
    filling = {}
    count = 1

    #Identify the grid length and width (i.e., number of indices in i and j)
    grid_length = int(DU_length/resolution)
    grid_width = int(DU_width/resolution)
    DU_area = DU_width * DU_length

    #For accounting purposes
    package_count = 0
    crush_count = 0
    non_crush_count = 0

    #Set random seed if output is to be exactly reproducible.
#    random.seed(8)
    #Initialize grid as unoccupied (indicated by 1) everywhere.
    unoccupied = []
    for i in range(0, grid_length + 1):
        unoccupied.append([1] * (grid_width))

    #Initialize unallowed packages list as empty list since all packages are
    #allowed at the start.
    unallowed = []

    #Keeping track of rows in which packages are placed so that iteration is not
    #ALWAYS performed over the entire length of the DU.
    row = 0
    j_current = 0
    #Iteration performed across a "large number" that is greater than the number
    #of packages to be placed.
    for p in range(0, 100000):
        #Select a package.
        package = select_package(inventory_dict)
        package_placed = False
        #Check if "most" inventory is still allowed
        if total_area < 0.9 * total_area_initial:
            break
        elif package in unallowed:
```

```
    pass
else:
            #Identify package dimensions and the coordinates to be spanned.
    package_width = package[0]
    package_length = package[1]
    delta_i = int(package_width/float(resolution))
    delta_j = int(package_length/float(resolution))
    for j in range(row, len(unoccupied)):
        #If that package has enough total space
        if sum(unoccupied[j]) >= delta_i:
            #Check if enough sequential space is available
            consec = 0
            for i in range(len(unoccupied[j])):
                if unoccupied[j][i] == 1:
                    #Unoccupied location
                    consec += 1
                if unoccupied[j][i] == 0:
                    #Unoccupied sequence broken
                    consec = 0
                if consec == delta_i:
                    #Found enough space in x direction
                    loc = ((i+1) - delta_i)
                    break
            #Check if the y direction has enough empty space
            if consec == delta_i:
                consec = 0
                fits = True
                if j + delta_j <= len(unoccupied) - 1:
                    for jj in range(j, j + delta_j):
                        for i in range(loc, loc + delta_i):
                            if unoccupied[jj][i] == 1:
                                consec += 1
                            else:
                                fits = False
                                break
                        if not fits:
                            break
                    #Found enough space for package placement.
                    if consec == delta_i * delta_j:
                        ind = (loc, j)
                        for jj in range(j, j + delta_j):
                            for i in range(loc, loc + delta_i):
                                unoccupied[jj][i] = 0
                                if package[2] == 'NC':
                                    occupations.update({(i,jj):'N'})
                                else:
                                    occupations.update({(i,jj):'C'})
                        package_count += 1
                        #Keep track of total non-crushable material
                        if package[2] == 'NC':
                            non_crush_count += 1
                            non_crush_area += (package_width * package_length)
                        else:
                            crush_count += 1
                            crush_area += (package_width * package_length)
                        #Update filling dictionary
                        if package[2] == 'NC':
                            #"Lower left" (i.e. min) corner coords
                            filling.update({count:[
                                                    'N',
                                                    (ind[0], ind[1]),
                                                    (ind[0]+delta_i, ind[1]),
                                                    (ind[0]+delta_i, ind[1]+delta_j),
                                                    (ind[0], ind[1]+delta_j)
                                                   ]
                                            })
                        else:
                            #"Lower left" (i.e. min) corner coords
                            filling.update({count:['C',
                                                    (ind[0], ind[1]),
                                                    (ind[0]+delta_i, ind[1]),
```

```
                                        (ind[0]+delta_i, ind[1]+delta_j),
                                        (ind[0], ind[1]+delta_j)
                                      ]
                                  })
                        count += 1
                        package_placed = True
                        #Track rows and update if need be so that iteration
                        #is not always performed across the entire length.
                        if j != j_current:
                            if j - 3 >= 0:
                                row = j - 3
                            j_current = j
                        break
                #Package doesn't fit in vertical direction, try rotation
                else:
                    if sum(unoccupied[j]) >= delta_j:
                        #Check if enough sequential space is available
                        consec = 0
                        for i in range(len(unoccupied[j])):
                            if unoccupied[j][i] == 1:
                                #Unoccupied location
                                consec += 1
                            if unoccupied[j][i] == 0:
                                #Unoccupied sequence broken
                                consec = 0
                            if consec == delta_j:
                                #Found enough space in x direction
                                loc = ((i+1) - delta_j)
                                break
                        #Check if the y direction has enough empty space
                        if consec == delta_j:
                            consec = 0
                            fits = True
                            if j + delta_i <= len(unoccupied) - 1:
                                for jj in range(j, j + delta_i):
                                    for i in range(loc, loc + delta_j):
                                        if unoccupied[jj][i] == 1:
                                            consec += 1
                                        else:
                                            fits = False
                                            break
                                    if not fits:
                                        break
                                #Found enough space for package placement.
                                if consec == delta_i * delta_j:
                                    ind = (loc, j)
                                    for jj in range(j, j + delta_i):
                                        for i in range(loc, loc + delta_j):
                                            unoccupied[jj][i] = 0
                                            if package[2] == 'NC':
                                                occupations.update({(i,jj):'N'})
                                            else:
                                                occupations.update({(i,jj):'C'})
                                    package_count += 1
                                    #Keep track of total non-crushable material
                                    if package[2] == 'NC':
                                        non_crush_count += 1
                                        non_crush_area += (package_width * package_length)
                                    else:
                                        crush_count += 1
                                        crush_area += (package_width * package_length)
                                    #Update filling dictionary
                                    if package[2] == 'NC':
                                        #"Lower left" (i.e. min) corner coords
                                        filling.update({count:[
                                                               'N',
                                                               (ind[0], ind[1]),
                                                               (ind[0]+delta_j, ind[1]),
                                                               (ind[0]+delta_j,
ind[1]+delta_i),
                                                               (ind[0], ind[1]+delta_i)
```

```
                                                    ]
                                                })
                                        else:
                                            #"Lower left" (i.e. min) corner coords
                                            filling.update({count:['C',
                                                                   (ind[0], ind[1]),
                                                                   (ind[0]+delta_j, ind[1]),
                                                                   (ind[0]+delta_j,
ind[1]+delta_i),
                                                                   (ind[0], ind[1]+delta_i)
                                                                  ]
                                                            })
                                            count += 1
                                            package_placed = True
                                            #Track rows and update if need be so
                                            #that iteration is not always performed
                                            #across the entire length.
                                            if j != j_current:
                                                if j - 3 >= 0:
                                                    row = j - 3
                                                j_current = j
                                            break
                        #Rotate the package
                        elif sum(unoccupied[j]) >= delta_j:
                            #Check if enough sequential space is available
                            consec = 0
                            for i in range(len(unoccupied[j])):
                                if unoccupied[j][i] == 1:
                                    #Unoccupied location
                                    consec += 1
                                if unoccupied[j][i] == 0:
                                    #Unoccupied sequence broken
                                    consec = 0
                                if consec == delta_j:
                                    #Found enough space in x direction
                                    loc = ((i+1) - delta_j)
                                    break
                            #Check if the y direction has enough empty space
                            if consec == delta_j:
                                consec = 0
                                fits = True
                                if j + delta_i <= len(unoccupied) - 1:
                                    for jj in range(j, j + delta_i):
                                        for i in range(loc, loc + delta_j):
                                            if unoccupied[jj][i] == 1:
                                                consec += 1
                                            else:
                                                fits = False
                                                break
                                        if not fits:
                                            break
                                    #Found enough space for package placement.
                                    if consec == delta_i * delta_j:
                                        ind = (loc, j)
                                        for jj in range(j, j + delta_i):
                                            for i in range(loc, loc + delta_j):
                                                unoccupied[jj][i] = 0
                                                if package[2] == 'NC':
                                                    occupations.update({(i,jj):'N'})
                                                else:
                                                    occupations.update({(i,jj):'C'})
                                        package_count += 1
                                        #Keep track of total non-crushable material
                                        if package[2] == 'NC':
                                            non_crush_count += 1
                                            non_crush_area += (package_width * package_length)
                                            #"Lower left" (i.e. min) corner coords
                                            filling.update({count:[
                                                                   'N',
                                                                   (ind[0], ind[1]),
                                                                   (ind[0]+delta_j, ind[1]),
```

```
                                                    (ind[0]+delta_j, ind[1]+delta_i),
                                                    (ind[0], ind[1]+delta_i)
                                                ]
                                            })
                                    else:
                                        crush_count += 1
                                        crush_area += (package_width * package_length)
                                        #"Lower left" (i.e. min) corner coords
                                        filling.update({count:['C',
                                                    (ind[0], ind[1]),
                                                    (ind[0]+delta_j, ind[1]),
                                                    (ind[0]+delta_j, ind[1]+delta_i),
                                                    (ind[0], ind[1]+delta_i)
                                                ]
                                            })
                                    count += 1
                                    package_placed = True
                                    #Track rows and update if need be so that iteration
                                    #is not always performed across the entire length.
                                    if j != j_current:
                                        if j - 3 >= 0:
                                            row = j - 3
                                        j_current = j
                                    break
                #Package doesn't fit in trench, remove from inventory.
                if not package_placed:
                    unallowed.append(package)
                    del inventory_dict[package]
                    total_area = 0
                    for i in range(len(inventory_dict.keys())):
                        area = (inventory_dict.keys()[i][0] *
                                inventory_dict.keys()[i][1] *
                                inventory_dict[inventory_dict.keys()[i]])
                        total_area += area

    percent_non_crushable = non_crush_area / float(DU_length * DU_width)
    ratio_C_NC = non_crush_area / crush_area

    return occupations, percent_non_crushable, filling, \
             ratio_C_NC, package_count, crush_count, non_crush_count



def MC_realizations(n_realizations, trench_type, inventory_dict,
                            DU_length, DU_width, resolution,
                            OUTfilename, mov=False):
    """ Run n_realizations Monte Carlo realizations and output Tecplot
        formatted heat map showing the frequency that a given grid
        location is occupied by a non-crushable package.
    """
    NC_count = {}
    C_NC_balance = {}
    avg_percent_NC = 0
    avg_ratio = 0
    avg_crush = 0
    avg_NC = 0
    avg_count = 0
    for i in range(1, n_realizations + 1):
        print 'Realization #', i, '\n'
        #Run Monte Carlo realization
        start = timeit.default_timer()
        occupations, percent_non_crush, filling, ratio_C_NC, \
             package_count, crush_count, non_crush_count = \
                                                MC_realization(inventory_dict,
                                                               DU_length,
                                                               DU_width,
                                                               resolution)
        finish = timeit.default_timer()
        avg_percent_NC += percent_non_crush
        avg_ratio += ratio_C_NC
```

```
    avg_crush += crush_count
    avg_NC += non_crush_count
    avg_count += package_count
    print 'Average Ratio NC to C  :', avg_ratio / float(i)
    print 'Average % NC           :', 100 * (avg_percent_NC/float(i))
    print 'Avg #packages:         :', avg_count / float(i)
    print 'Avg #non_crush         :', avg_NC / float(i)
    print 'Avg #crush             :', avg_crush / float(i)
    print '\t\tMC Realization took', finish - start, 'seconds!\n\n'
    #Update the frequency dictionary
    for key in occupations.keys():
        if occupations[key] == 'N':
            try:
                NC_count[key] += 1
            except KeyError:
                NC_count.update({key:1})
            try:
                C_NC_balance[key] += 1
            except KeyError:
                C_NC_balance.update({key:1})
        if occupations[key] == 'C':
            try:
                C_NC_balance[key] -= 1
            except KeyError:
                C_NC_balance.update({key:-1})


#Write Tecplot file showing frequency of NC materials at all locations
filename = '%s_MC_%s_Realizations_%s_%s_Freq.tec' %(trench_type,
                                                    n_realizations,
                                                    DU_width,
                                                    DU_length)
filename2 = '%s_MC_%s_Realizations_%s_%s_Bal.tec' %(trench_type,
                                                    n_realizations,
                                                    DU_width,
                                                    DU_length)
with open(filename, 'w') as f1:
    with open(filename2, 'w') as f2:
        f1.write('TITLE = "E-Area Trench"\n')
        f1.write('VARIABLES = "x", "y" "Frequency"\n')
        f1.write('ZONE T="Grid", I=%s, J=%s, F=POINT\n'
                                                   %(int(width/resolution) + 1,
                                                     int(length/resolution) + 1))
        f2.write('TITLE = "E-Area Trench"\n')
        f2.write('VARIABLES = "x", "y" "Balance"\n')
        f2.write('ZONE T="Grid", I=%s, J=%s, F=POINT\n'
                                                   %(int(width/resolution) + 1,
                                                     int(length/resolution) + 1))
        for j in range(0, int(length/resolution) + 1):
            #Iterate across positions along the DU length.
            for i in range(0, int(width/resolution) + 1):
                try:
                    frequency = NC_count[(i, j)]
                except KeyError:
                    frequency = 0
                f1.write('%s\t\t%s\t\t%s\n' %(i * resolution,
                                              j * resolution,
                                              frequency))
                try:
                    balance = C_NC_balance[(i,j)]
                except KeyError:
                    balance = 0
                f2.write('%s\t\t%s\t\t%s\n' %(i * resolution,
                                              j * resolution,
                                              balance))

#Generate static and dynamic tecplot file outputs for final realization
tecplot_tec_mcr(filling, trench_type, OUTfilename, 3, movie=mov)

return NC_count
```

```
def tecplot_tec_mcr(filling, trench_type, OUTfilename, proc, movie=False):
    """Write Tecplot (.tec) output file, Tecplot macro file(s) (.mcr),
       and stitch the files together to create a movie.
    """
    #Make tmp directory if it does not already exist.
    if not os.path.exists('tmp'):
        os.makedirs('tmp')
    tecout = OUTfilename
    print tecout
    macros = []
    with open(tecout, 'w') as f1:
        #Write .tec header material
        #Function could be made more efficeint by writing separate .tec files,
        #but typically only used in isolated instances for diagnostics.
        f1.write('TITLE = "E-Area Trench"\n')
        f1.write('VARIABLES = "x", "y" "type"\n')
        f1.write('ZONE T="Grid", I=%s, J=%s, F=POINT\n'
                                                %(int(width/resolution) + 1,
                                                  int(length/resolution) + 1))

        #Iterate across positions along the DU width.
        for j in range(0, int(length/resolution) + 1):
            #Iterate across positions along the DU length.
            for i in range(0, int(width/resolution) + 1):
                f1.write('%s\t\t%s\t\t0\n' %(i * resolution, j * resolution))

        #Number of zones to go in each macro file (based on 4 subprocesses)
        zones = sorted(filling.keys())
        n_zones_total = len(zones)
        mcr_indicator = int(n_zones_total/float(proc))
        #Remaining zones to be added to the final .mcr file for processing.
        extra = n_zones_total - mcr_indicator*proc
        mcr_count = 1
        for key in sorted(zones):
            if movie:
                #Specify macro file name and write header material for each
                #range (of length of mcr_indicator) of keys
                if key == 1 or key % mcr_indicator == 0:
                    if key != 1:
                        #Write final line of .mcr file and open new .mcr file
                        f2.write('$!RemoveVar |MFBD|\n')
                        f2.close()
                    macro = 'TrenchFill_%s.mcr' %(mcr_count)
                    macros.append(macro)
                    f2 = open(macro, 'w')
                    mcr_count += 1
                    #Put remaining zones into the fourth file
                    if mcr_count == proc+1:
                        #10000 is an arbitrarily high number that
                        #should always be greater mcr_indicator
                        mcr_indicator += 10000
                    #Write .mcr header material
                    f2.write('#!MC 1200\n')
                    f2.write('# Created by Tecplot Focus build 12.2.0.9048\n')
                    MCRpath = os.getcwd()
                    f2.write("$!VarSet |MFBD| = '%s'\n" %(MCRpath))

                    #Remove header and adjust frame size
                    f2.write('$!FRAMECONTROL ACTIVATEBYNUMBER\n')
                    f2.write('  FRAME = 1\n')
                    f2.write('$!FRAMELAYOUT SHOWBORDER = NO\n')
                    f2.write('$!FRAMELAYOUT XYPOS{X = 0}\n')
                    f2.write('$!FRAMELAYOUT XYPOS{Y = 0}\n')
                    f2.write('$!FRAMELAYOUT WIDTH = 11\n')
                    f2.write('$!FRAMELAYOUT HEIGHT = 8.5\n')
                    f2.write('$!VIEW CENTER\n')

                    #Format Tecplot zones to account for variable lengths
                    f2.write('$!ACTIVEFIELDMAPS += [1-%s]\n' %(n_zones_total))
```

```
                f2.write('$!FIELDMAP [2-%s]  MESH{COLOR = BLACK}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  CONTOUR{COLOR = BLACK}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  CONTOUR{CONTOURTYPE =' \
                                                      'PRIMARYVALUE}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  VECTOR{COLOR = BLACK}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  SCATTER{COLOR = BLACK}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  EDGELAYER{COLOR = BLACK}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  EDGELAYER{LINETHICKNESS' \
                                                           ' = 0.1}\n'
                                                    %(n_zones_total))
                f2.write('$!FIELDMAP [2-%s]  EDGELAYER{EDGETYPE =' \
                                                 ' BORDERSANDCREASES}\n'
                                                    %(n_zones_total))
                f2.write('$!TWODAXIS AXISMODE = INDEPENDENT\n')
                f2.write('$!VIEW FIT\n')
                f2.write('$!ACTIVEFIELDMAPS -= [1-%s]\n' %(n_zones_total+1))

        #For coloring crushable or non-crushable packages in Tecplot
        C_NC = filling[key][0]
        if C_NC == 'C':
            C_NC = 1
        else:
            C_NC = 2
        #Coordinates of 4 corners of the rectangular package
        x1 = filling[key][1][0] * resolution
        y1 = filling[key][1][1] * resolution
        x2 = filling[key][2][0] * resolution
        y2 = filling[key][2][1] * resolution
        x3 = filling[key][3][0] * resolution
        y3 = filling[key][3][1] * resolution
        x4 = filling[key][4][0] * resolution
        y4 = filling[key][4][1] * resolution
        #Write zone coords.
        f1.write('ZONE T="%s", I=2, J=2, F=POINT\n' %(key))
        f1.write('%s\t\t%s\t\t%s\n' %(x1, y1, C_NC))
        f1.write('%s\t\t%s\t\t%s\n' %(x2, y2, C_NC))
        f1.write('%s\t\t%s\t\t%s\n' %(x4, y4, C_NC))
        f1.write('%s\t\t%s\t\t%s\n' %(x3, y3, C_NC))
        if movie:
            #Write active field maps in Tecplot
            f2.write('$!ACTIVEFIELDMAPS += [1-%s]\n' %(key+1))
            f2.write('$!EXPORTSETUP EXPORTFORMAT = PNG\n')
            #Format .png number by finding the length of the longest integer
            max_len = len(str(filling.keys()[-1]))
            zeros = '0' * (max_len - len(str(key)))
            PNGnum = zeros + str(key)
            PNGpath = os.getcwd() + '/tmp'
            f2.write("$!EXPORTSETUP EXPORTFNAME = '%s/%s'\n"%(PNGpath,
                                                    (PNGnum + '.png')
                                                        ))
            f2.write('$!EXPORT\n')
            f2.write('  EXPORTREGION = CURRENTFRAME\n')
    if movie:
        f2.write('$!RemoveVar |MFBD|\n')
        f2.close()
if movie:
    #Use multiprocessing to run Tecplot macro files and create .png files.
    if __name__ == '__main__':
        p = Pool(proc)
        p.map(run_tecMCR, macros)

    #Stich .png files together to create a movie.
    make_movie(trench_type, str(max_len))

return
```

```
def run_tecMCR(macro_file):
    """Run Tecplot macro files (*.mcr)."""
    tecfile = 'BoxedDistribution.lay'
    #Command line entry
    tecplot_comm = ["tecfocus", "-b", "-p", macro_file, tecfile]
    #Run tecplot macro on .tec file
    subprocess.call(tecplot_comm)
    return



def make_movie(trench_type, n_zeros):
    """Stitch *.png files together to create a movie."""
    outpath = 'tmp/' + trench_type + 'out.mp4'
    #Remove output path if it already exists.
    if os.path.exists(outpath):
        os.remove(outpath)
    #Stitch .png files together.
    ffmpeg = ("ffmpeg -i tmp/%0" + n_zeros
              + "d.png -pix_fmt yuv420p tmp/" + trench_type + "out.mp4")
    os.system(ffmpeg)
    #Remove .png files after stitching
    os.system("rm -r tmp/*.png")
    return



#Length and width of the trench
length = 656
width = 157
#length = 656
#width = 20
#Fractional resolution of grid cells in feet.
resolution = 1.
#Number of Monte Carlo realizations
n_realizations = 25000
#ET or ST
trench_type = 'ET'
#File from which inventory is read
inventory_filename = 'ET_C_NC_Inventory_6_4.dat'
#Static tecplot output filename (i.e., showing all boxes placed in trench)
OUTfilename = inventory_filename.split('.')[0] + '.tec'

#Read inventory and create dictionary for MC input
inventory = read_data2(inventory_filename)



#Running n_realizations realizations
MC_realizations(n_realizations, trench_type, inventory,
                        length, width, resolution,
                        OUTfilename, mov=False)
```

Appendix B. WITS Query Listing all Non-Crushable Containers in the ELLWF.

2/1/2018 10:51:36 AM

Disposal Location of Non-Crushing Waste Forms

Package ID	Deviation Number	Area (ft2)
EAV	**ETRENCH2**	**1**
CBALX4962	SWMD-RFD-2003-0054,	14.60
CBALX5066	SWMD-RFD-2003-0054,	12.00
CBALX5083	SWMD-RFD-2003-0054	7.30
Count for 1: 3		33.90
Total for ETRENCH2: 3		33.90
EAV	**SLIT14**	**A**
AC10001497	WIR N/A, SWMD-RFD-2011-0001,	383.00
Count for A: 1		383.00
Total for SLIT14: 1		383.00
EAV	**SLIT2**	**1**
RD009462	SWMD-RFD-2002-0071	253.50
RD009463	SWMD-RFD-2002-0071	253.50
RD009464	SWMD-RFD-2002-0071	253.50
RD009465	SWMD-RFD-2002-0071	253.50
RD009466	SWMD-RFD-2002-0071	253.50
RD009467	SWMD-RFD-2002-0071	253.50
RD009468	SWMD-RFD-2002-0071	253.50
RD009469	SWMD-RFD-2002-0071	253.50
Count for 1: 8		2,028.00
EAV	**SLIT2**	**A**
WP24730001	SWMD-RFD-2003-0052	17.78
Count for A: 1		17.78
Total for SLIT2: 9		2,045.78
EAV	**SLIT3**	**A**
SR595025	SWMD-RFD-2004-0004	223.56
SR595026	SWMD-RFD-2004-0004	223.56
Count for A: 2		447.12
EAV	**SLIT3**	**B**
WP24740033	SWMD-RFD-2003-0064	9.30
Count for B: 1		9.30
EAV	**SLIT3**	**C**

Page 1 of 7

Package ID	Deviation Number	Area (ft2)
SR571645	SWMD-RFD-2004-0004	223.56
SR571683	SWMD-RFD-2004-0004	223.56
SR571687	SWM-RFD2004-0004	223.56
SR571688	SWMD-RFD-2004-0004	223.56
SR571691	SWMD-RFD-2004-0004	223.56
SR595027	SWMD-RFD-2004-0004	223.56
SR595032	SWMD-RFD-2004-0004	223.56
SR595033	SWMD-RFD-2004-0004	223.56
SR595035	SWMD-RFD-2004-0004	223.56
Count for C: 9		2,012.04
EAV	**SLIT3**	**D**
WP24740074	SWMD-RFD-2004-0013 & N-NCS-F-00098 (REV. 3)	29.84
Count for D: 1		29.84
EAV	**SLIT3**	**E**
FD00009092	SWMD-RFD-2004-0036	320.00
FD00009093	SWMD-RFD-2004-0036	320.00
FD00009094	SWMD-RFD-2004-0036	320.00
FD00009095	SWMD-RFD-2004-0036	320.00
WP24740095	SWMD-RFD-2004-0013 & N-NCS-F-00098 (REV. 3)	9.61
WP24740192	SWMD-RFD-2004-0063	124.02
WP24740193	SWMD-RFD-2004-0063	172.50
WP24740194	SWMD-RFD-2004-0063	240.00
WP24740195	SWMD-RFD-2004-0063	213.44
WP24740196	SWMD-RFD-2004-0063	140.00
WP24740283	SWMD-RFD-2005-0008	27.25
Count for E: 11		2,206.82
EAV	**SLIT3**	**F**
ET11000640	OBU-GSE-2004-00003	78.54
ET11000966	OBU-GSE-2004-00003	78.54
ET11001003	OBU-GSE-2004-00003	78.54
ET11001115	OBU-GSE-2004-00003	78.54
Count for F: 4		314.15
Total for SLIT3: 28		5,019.27

EAV	SLIT4	B
FC30S01844	SWMD-RFD-2006-0041	21.60
SD00005112	SWMD-RFD-2005-0045	185.00
SD00005113	SWMD-RFD-2005-0045	41.40
SD00005114	SWMD-RFD-2005-0045	41.40

Package ID	Deviation Number	Area (ft2)
SD00005118	SWMD-RFD-2005-0045	124.90
SD00005119	SWMD-RFD-2005-0045	124.90
SD00005120	SWMD-RFD-2005-0045	124.90
SD00005121	SWMD-RFD-2005-0045	124.90
SD00005124	SWMD-RFD-2005-0045	37.50
SD00005132	SWMD-RFD-2005-0045	12.50
SD00006358	SWMD-RFD-2005-0045	11.20
SD00006383	SWMD-RFD-2005-0045	11.20
SD00006384	SWMD-RFD-2005-0045	133.50
SD00006386	SWMD-RFD-2005-0045	118.00
Count for B: 14		1,112.90
EAV	**SLIT4**	**C**
WP24740185	N-NCS-F-00098 (REV. 3) & SWMD-RFD-2004-0063	29.20
WP24740186	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740187	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740188	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740189	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740190	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740191	SWMD-RFD-2004-0063 & N-NCS-F-00098 (REV. 3)	29.20
WP24740209	SWMD-RFD-2004-0067	24.80
WP24740281	SWMD-RFD-2005-0007	33.25
WP24740282	SWMD-RFD-2005-0007	33.25
Count for C: 10		295.70
EAV	**SLIT4**	**D**
FD00009089	SWMD-RFD-2004-0036	320.00
FD00009096	SWMD-RFD-2004-0036	320.00
FD00009101	SWMD-RFD-2004-0036	320.00
Count for D: 3		960.00
EAV	**SLIT4**	**E**
SR571676	SWMD-RFD-2004-0004	223.56
SR571679	SWMD-RFD-2004-0004	223.56
SR571681	SWMD-RFD-2004-0004	223.56
SR571682	SWMD-RFD-2004-0004	223.56
SR571685	SWMD-RFD-2004-0004	223.56
SR571689	SWMD-RFD-2004-0004	223.56
Count for E: 6		1,341.36
Total for SLIT4: 33		3,709.96
EAV	**SLIT5**	**A**

Package ID	Deviation Number	Area (ft2)
SWD041258	SWMD-RFD-2004-0038	14.00
SWD041259	SWMD-RFD-2004-0038	14.00
SWD041260	SWMD-RFD-2004-0038	14.00
SWD041261	SWMD-RFD-2004-0038	14.00
SWD041262	SWMD-RFD-2004-0038	14.00
SWD041263	SWMD-RFD-2004-0038	14.00
WP24740107	SWMD-RFD-2004-0049	28.75
WP24740197	SWMD-RFD-2004-0063	64.50
WP24740198	SWMD-RFD-2004-0063	62.35
WP24740199	SWMD-RFD-2004-0063	51.60
WP24740200	SWMD-RFD-2004-0063 RELOCATION ERROR	21.52
WP24740201	SWMD-RFD-2004-0063	28.69
Count for A: 12		341.41
EAV	**SLIT5**	**C**
CBALX4472	SWMD-RFD-2003-0054 SWE APPROVED	35.64
Count for C: 1		35.64
EAV	**SLIT5**	**E**
SD00005424	SWMD-RFD-2006-0009	81.80
Count for E: 1		81.80
EAV	**SLIT5**	**H**
FC30S01803	SWMD-RFD-2005-0049	23.80
Count for H: 1		23.80
EAV	**SLIT5**	**J**
WP24740212	SWMD-RFD-2004-0071 SWE APPROVED	31.50
WP24740214	SWMD-RFD-2004-0071 SWE APPROVED	31.50
WP24740223	N-NCS-F-00098 (REV. 3) & SWMD-RFD-2004-0071	11.25
Count for J: 3		74.25
Total for SLIT5: 18		556.90

EAV	SLIT6	A
SWD061506	SWMD-RFD-2006-0023	13.30
SWD061507	SWMD-RFD-2006-0023	13.30
SWD061508	SWMD-RFD-2006-0023	13.30
SWD061509	SWMD-RFD-2006-0023	13.30
SWD061510	SWMD-RFD-2006-0023	13.30
SWD061511	SWMD-RFD-2006-0023	13.30
SWD061512	SWMD-RFD-2006-0023	13.30
SWD061513	SWMD-RFD-2006-0023	13.30
SWD061514	SWMD-RFD-2006-0023	13.30

Package ID	Deviation Number	Area (ft2)
SWD061515	SWMD-RFD-2006-0023	13.30
SWD061516	SWMD-RFD-2006-0023	13.30
SWD061517	SWMD-RFD-2006-0023	13.30
SWD061518	SWMD-RFD-2006-0023	13.30
SWD061519	SWMD-RFD-2006-0023	13.30
SWD061520	SWMD-RFD-2006-0023	13.30
SWD061521	SWMD-RFD-2006-0023	13.30
SWD061522	SWMD-RFD-2006-0023	13.30
SWD061523	SWMD-RFD-2006-0023	13.30
SWD061524	SWMD-RFD-2006-0023	13.30
SWD061525	SWMD-RFD-2006-0023	13.30
SWD061526	SWMD-RFD-2006-0023	13.30
SWD061527	SWMD-RFD-2006-0023	13.30
SWD061528	SWMD-RFD-2006-0023	13.30
SWD061529	SWMD-RFD-2006-0023	13.30
SWD061530	SWMD-RFD-2006-0023	13.30
SWD061531	SWMD-RFD-2006-0023	13.30
SWD061532	SWMD-RFD-2006-0023	13.30
SWD061533	SWMD-RFD-2006-0023	13.30
SWD061535	SWMD-RFD-2006-0023	13.30
SWD061537	SWMD-RFD-2006-0023	13.30
SWD061538	SWMD-RFD-2006-0023	13.30
SWD061539	SWMD-RFD-2006-0023	13.30
SWD061540	SWMD-RFD-2006-0023	13.30
SWD061541	SWMD-RFD-2006-0023	13.30
SWD061542	SWMD-RFD-2006-0023	13.30
Count for A: 35		465.50
EAV	**SLIT6**	**B**
SD00005127	SWMD-RFD-2005-0045	12.10
SD00005129	SWMD-RFD-2005-0045	112.00
SD00005131	SWMD-RFD-2005-0045	112.00
Count for B: 3		236.10
EAV	**SLIT6**	**C**
CBALX4760	SWMD-RFD-2003-0054	115.90
CBALX4776	SWMFD-RFD-2003-0054	49.50
CBALX4783	SWMD-RFD-2003-0054	28.00
CBALX4845	SWMD-RFD-2003-0054	10.80
FCAN08001	SWMD-RFD-2008-00011, REV. 0	182.00
Count for C: 5		386.20
EAV	**SLIT6**	**D**

Package ID	Deviation Number	Area (ft2)
AC10001498	SWMD-RFD-2011-00002,	138.00
AC10001499	SWMD-RFD-2011-00002,	138.00
Count for D: 2		276.00
EAV	**SLIT6**	**F**
FC30S01824	SWMD-RFD-2006-0026	50.30
SD00005128	SWMD-RFD-2005-0045	112.00
SD00005130	SWMD-RFD-2005-0045	112.00
Count for F: 3		274.30
EAV	**SLIT6**	**G**
FCASROX100	SWMD-RFD-2009-00002,	228.00
Count for G: 1		228.00
Total for SLIT6: 49		1,866.10
EAV	**SLIT7**	**E**
RD00009839	SWMD-RFD-2006-0024	12.30
SWD061534	SWMD-RFD-2006-0023	13.30
SWD061536	SWMD-RFD-2006-0023	13.30
Count for E: 3		38.90
EAV	**SLIT7**	**F**
SWD061500	SWMD-RFD-2006-0023	13.30
SWD061501	SWMD-RFD-2006-0023	13.30
SWD061502	SWMD-RFD-2006-0023	13.30
SWD061503	SWMD-RFD-2006-0023	13.30
SWD061504	SWMD-RFD-2006-0023	13.30
SWD061505	SWMD-RFD-2006-0023	13.30
Count for F: 6		79.80
EAV	**SLIT7**	**H**
HC27001821	SWMD-RFD-2008-00001	51.00
Count for H: 1		51.00
EAV	**SLIT7**	**J**
ET11001154	SWE APPROVED. OBU-GSE-2004-00003	78.54
Count for J: 1		78.54
EAV	**SLIT7**	**K**
SD00006680	SWMD-RFD-2009-00007, REV. 1,	187.10
Count for K: 1		187.10
Total for SLIT7: 12		435.34
EAV	**SLIT8**	**B**

Package ID	Deviation Number	Area (ft2)
SD00006313	SWMD-RFD-2005-0045	12.90
Count for B: 1		12.90
Total for SLIT8: 1		12.90

Distribution:

timothy.brown@srnl.doe.gov
alex.cozzi@srnl.doe.gov
david.crowley@srnl.doe.gov
c.diprete@srnl.doe.gov
a.fellinger@srnl.doe.gov
samuel.fink@srnl.doe.gov
nancy.halverson@srnl.doe.gov
connie.herman@srnl.doe.gov
patricia.lee@srnl.doe.gov
Joseph.Manna@srnl.doe.gov
john.mayer@srnl.doe.gov
daniel.mccabe@srnl.doe.gov
Gregg.Morgan@srnl.doe.gov
frank.pennebaker@srnl.doe.gov
Amy.Ramsey@srnl.doe.gov
William.Ramsey@SRNL.DOE.gov
michael.stone@srnl.doe.gov
Boyd.Wiedenman@srnl.doe.gov
bill.wilmarth@srnl.doe.gov
sebastian.aleman@srnl.doe.gov
paul.andrews@srs.gov
dan.burns@srs.gov
tom.butcher@srnl.doe.gov
kerri.crawford@srs.gov
Thomas.Danielson@srnl.doe.gov
kenneth.dixon@srnl.doe.gov
James.Dyer@srnl.doe.gov
peter.fairchild@srs.gov
luther.hamm@srnl.doe.gov
thong.hang@srnl.doe.gov
daniel.kaplan@srnl.doe.gov
walt.kubilius@srnl.doe.gov
Dien.Li@srs.gov
steven.mentrup@srs.gov
verne.mooneyhan@srs.gov
ralph.nichols@srnl.doe.gov
Virgina.Rigsby@srs.gov
Jansen.Simmons@srs.gov
frank.smith@srnl.doe.gov
Ira.Stewart@srs.gov
Tad.Whiteside@srnl.doe.gov
Jennifer.Wohlwend@srnl.doe.gov
Records Administration (EDWS)