THESIS

IMPROVE THE SAFETY AND EFFECTIVENESS OF FIRELINES USING DIJKSTRA'S ALGORITHM: MODEL DEVELOPMENT AND IMPLEMENTATION DURING INCIDENT

Submitted by

Zhiwei Liu

Department of Forest and Rangeland Stewardship

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2019

ABSTRACT

IMPROVE THE SAFETY AND EFFECTIVENESS OF FIRELINES USING DIJKSTRA'S ALGORITHM: MODEL DEVELOPMENT AND IMPLEMENTATION DURING INCIDENT

Multiple spatial indices have been developed by past research to evaluate fire management effectiveness and safety concerns and identify fire suppression operation opportunities. In this study, we will use those indices to directly support fireline construction during fire incident management. To identify firelines that can ensure both suppression effectiveness and responder safety, we introduced a raster-surface-based spatial optimization model based on Dijkstra's algorithm. We modified and implemented this algorithm on a network generated from a spatial index derived raster cost surface, and identified paths with the lowest accumulated cost. We also applied indices specified thresholds to avoid locations that may be too risky to conduct fireline construction. We use a portion of northwest Sierra National Forest in California, the USA, as our study site for demonstration, since this area has been well studied in previous research and has abundant data. We tested our model by improving some of the proposed event lines during the 2018 Ferguson Fire suppression for better safety and effectiveness. Testing results confirmed that our model can achieve management objectives. At the end of this study, we identified the limitations of our model, and proposed future works to improve this model to better support fire management.

ACKNOWLEDGEMENTS

I would like to thank Christopher “Kit” O’Connor for helping generate the terrestrial SDI raster layer used in this research.

DEDICATION

I would like to dedicate this thesis to my family.

TABLE OF CONTENTS

ABSTRACT ii
ACKNOWLEDGEMENTS iii
DEDICATION iv

Chapter 1 Introduction 1

Chapter 2 Methods 7
2.1 Designing methodology 7

Chapter 3 Study site and test cases 15
3.1 Study site description 15
3.2 Model implementation and test cases 15

Chapter 4 Results 26

Chapter 5 Conclusion and discussion 34

References 36

Appendix A Programming code examples 43

Chapter 1
Introduction

During the past several decades, wildfires have burned substantially larger areas in the U.S. and with highly variable annual areas burned; both contributed to the dramatically increased firefighting expenditures and escalated concerns of public safety and fire responder safety (Calkin et al., 2005; Liang et al., 2008; Gebert et al., 2012). In the early 2000s, the average annual wildfire suppression expenditure in the U.S. has exceeded US$1 billion, comparing with the less than US$400 million annual average from 1970 to 2000 (under the same currency purchase value) (Calkin et al., 2005; Liang et al., 2008). Land management agencies responsible for fire management are concerned about cost and budget issues and looking for methods to lower fire suppression cost and improve suppression efficiency (Calkin et al., 2012). In addition, since fire responder and public safety have become the primary concern in wildland fire management programs (Calkin et al., 2012; NWCG, 2014), policymakers and fire managers also need to select wildfire suppression strategies in compliance with safety requirements.

Determining the suitable location to construct fireline is an important component of wildfire suppression strategies and tactics (NWCG, 2014). However, it has been a challenge in both fire research and practical realms. A successful fireline often has three characteristics: easy access and dispatch firefighting resources; having a higher chance of line-holding on wildfire; and ensuring the fire responder safety. Fire behavior simulation can provide useful information to help managers decide fireline locations. Some fire models used in simulating fire behavior and spread patterns can provide potentially useful suppression suggestions. For example, one module in FARSITE (Finney, 1998) can simulate suppression locations by comparing fireline production rate against fire growth rate. This module also considers spatial heterogeneity in fuels and topography, which will impact fire growth rate and fireline production rate, and outputs spatially varying suppression location suggestions (Finney et al., 2002). BehavePlus (Andrews et al., 2008) models fire behavior, fire effects and the fire environment in a spatially explicit manner. Its CONTAIN module, which is based on the algorithm of Fried and Fried (1996), estimates fire suppression resources necessary for containment of a fire ignites and spreads from a point source (Andrews, 2014). FSPro (US Forest Service, 2009) stochastically simulates and ensembles many fire footprints, which include burn probability for a specified time period in each raster cell. Combining with spatial information such as critical infrastructure and natural resources that require protections (e.g. RAVAR (Calkin and Hyde, 2004)), FSPro results can inform fire managers the proper management decisions in selecting fireline construction locations (Calkin et al., 2011; O'Connor et al., 2017).

Although modules supporting suppression decisions have been built into many of the current fire simulation systems, limitations often exist when executing those modules. For example, the limitation of spatial resolution on model input files may omit roads and some other narrow features that can limit fire spread by creating discontinuity of fuel patterns and provide access for fire crews and equipment deployment (Narayanaraj and Wimberly, 2011). This may also lead to overlooking the inhibition effects of natural or artificial barriers on fire spread route and cause the simulated fireline construction location not to align with the actual fire perimeters, which mostly follow the natural or artificial barriers (O'Connor et al., 2017). Another example of limitation is that some key factors which potentially influence fire suppression decisions are overlooked when selecting fireline construction locations. In FARSITE the horizontal fireline production rate for each fuel model is set to a fixed value defined by available suppression resources (Finney et al., 2002) without considering factors which may affect the spatial heterogeneity other than topography and fuel. In BehavePlus fireline production rate is also defined as a constant (Andrews, 2014), in spite of the difficulty of fireline construction varies across the landscape due to factors such as transportation conditions and vegetation density, etc. Locations easier for fireline construction generally require less suppression efforts and maybe safer for fire responders to engage a fire front. These locations should be categorized as more suitable operational locations to improve suppression effectiveness and responder safety.

Recent researchers have developed a series of spatial indices to evaluate fire management effectiveness and safety concerns and identify fire suppression operation opportunities. For example, Suppression Difficulty Index (SDI) is an index that integrates multiple weighted sub-indices including the simulated fire behavior (fire energy release) and landscape derived features (road network accessibility, equipment mobility off-road, firefighter traveling penetrability, fireline production rate and aerial resources availability) (Rodríguez y Silva et al., 2014). SDI partially quantizes the difficulty of performing suppression actions during a fire incident in relation to potential fire intensity. It also provides decisionmakers and fire managers the priority ranking information for deploying existing firefighting resources to increase suppression effectiveness (Rodríguez y Silva et al., 2014; O'Connor et al., 2017). SDI reflects responder safety concerns by mapping the locations that are not safe for engaging fire, conducting direct attack tactics, scouting or doing ground transportation (O'Connor et al., 2016; Dunn et al., 2019).

These spatial indices have been incorporated in many other wildland fire research to identify suppression opportunities or support fire suppression decisions. Mitsopoulos et al. (2016) reproduced and expanded the concept of SDI by mapping fire suppression difficulty with fine-scale fuel sampling on very high-resolution fuel maps across three different ecosystems in the Eastern Europe region. Another related wildland fire research is the implementation of the potential wildfire operational delineation (POD) concept. PODs are landscape polygons used as fire management units, whose boundaries are delineated by fire containment relevant features (Thompson et al., 2016; 2017). PODs are useful for summarizing fire risks, suppression opportunities, and response strategies within their perimeters (Thompson et al., 2016; 2017; Wei et al., 2019), and can be used to anchor fire management operations by constructing firelines on POD boundaries when only limited treatment opportunities exist across the landscape (Thompson et al., 2017). Besides these, PODs can be used to prioritize fuel treatment locations across a landscape (Thompson et al., 2017), and to design optimal large fire containment strategies during a real-time incident (Wei et al 2018; 2019). Thompson et al. (2016) suggested indices such as SDI could be used to characterize fire suppression difficulty within PODs along with fire risk and help fire managers decide operation priority among all PODs. Thompson et al. (2016) also pointed out that some PODs boundaries, which delineated using coarse approximations like watersheds or with more refined fire operation features, may not become effective barriers to stop fire spread or safe locations for firefighters. Taking consideration of this issue, SDI thresholds were applied by Thompson et al. (2018) to exclude boundaries associated with high suppression difficulty when delineating PODs. None of the above research, however, uses the surfaces of those spatial indices to directly support fireline construction.

Shortest path algorithms have been applied to support suppression activities by simulating fire spread based on different network structures. For example, Cova et al. (2005) constructed a shortest-path tree on a fire spread network and built an evacuation trigger buffer for early warning system. Stepanov et al. (2011) estimated minimum travel time paths and fire arrival times from ignition points by applying shortest path algorithm on a Delaunay triangulation network, which represents fire spread within a heterogeneous landscape. Inspired by those studies, we aim at implementing the shortest path algorithm to achieve our objective by using the spatial indices associated with suppression difficulty, safety and effectiveness (e.g. SDI). The objective is to construct firelines that are potentially cost less, safer to build and having higher probability of holding fire in position. More specifically, we have designed a raster-surface-based spatial optimization model using the Dijkstra's algorithm to identify potential paths of firelines weighted by selected spatial indices to decrease suppression difficulty on the paths and improve path holding probabilities, while minimizing the construction cost. We tested our model in a scenario from real-world fire incident; then we explain the strengths and limitations of this modeling method based on model performance and discuss where this model still needs to be improved in the future research.

Chapter 2
Methods

2.1 Designing methodology

We implemented the Dijkstra's algorithm in our raster-surface-based spatial optimization model. Dijkstra's algorithm is one of the most commonly used shortest path algorithms (Yu et al., 2003; Eldrandaly et al., 2014). It was designed for tracing the shortest path between a source node to a destination node within a network with nodes connected by weighted links (Dijkstra, 1959; Yu et al., 2003; Antikainen, 2013; Eldrandaly et al., 2014). Raster maps representing different cost surfaces can be used as inputs of the Dijkstra's algorithm in finding least-cost paths on a continuous surface. The core idea is to find the ideal path with the least accumulated cost between two locations on a raster cost surface (Eldrandaly et al., 2014).

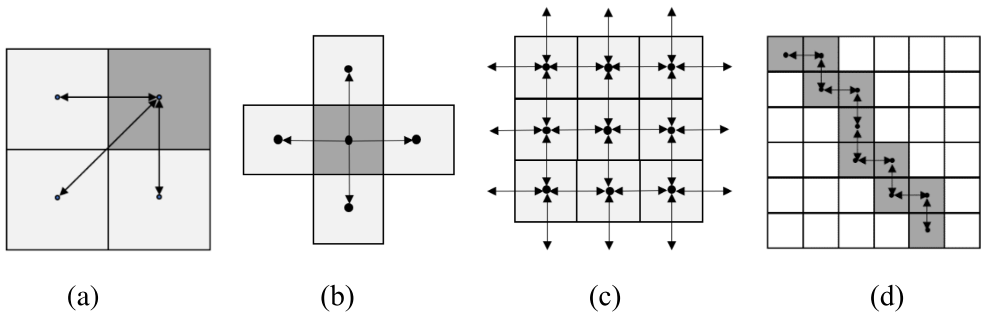


Fig. 1. Virtual network on raster map for Dijkstra's algorithm implementation. (a). A demonstration of nodes and links on the virtual network. (b). The orthogonal (aka. Rook's) pattern. (c). Network created based on orthogonal pattern. (d). Stair-stepped route caused by orthogonal pattern.

To implement Dijkstra's algorithm on a raster map, a virtual network needs to be constructed. Every raster cell center is treated as a node; the length between two neighboring cell centers can be viewed as links (Fig. 1a) (Xu and Lathrop, 1994; Yu et al., 2003).

There are multiple methods to define neighboring cells and assign nodes and links on a raster layer. The orthogonal (aka. Rook's) pattern is one common way to define nodes neighborhood relationship (Fig. 1b), where a defined node only connects to its four direct neighboring nodes with links. This neighborhood pattern creates a network with only horizontal or vertical arcs (Fig. 1c) (Church et al., 1992; van Bemmelen et al., 1993). However, with the orthogonal neighborhood pattern, a few geometric distortions will be produced. For example, an actual diagonal straight-line path will be distorted to a stair-stepped route (Fig. 1d) (van Bemmelen et al., 1993). Running the algorithm under this assumption will output a best route on the network that is significantly different from the ideal route in the real landscape (potentially longer paths). Thus, alternative approaches have been developed to reduce this distortion by increasing the number of directions of arcs emanating from each node (Church et al., 1992; Xu and Lathrop, 1994). These alternatives include both orthogonal and oblique neighborhood patterns. For example, a defined node connects not only to its four direct neighboring nodes but also to its four diagonal neighboring nodes (aka. Queen's pattern) (Fig. 2a). We can further add links similar to the knight's move in chess to the result of the Queen's pattern, then there will be a 16-direction connection neighborhood pattern (aka. Knight's pattern) (Fig. 2b) (Goodchild, 1977; Church et al., 1992; van Bemmelen et al., 1993; Xu and Lathrop, 1994; Yu et al., 2003). This logic can be expanded to obtain a 32-direction or more connected network (van Bemmelen et al., 1993). Although adding connection directions of neighboring nodes can reduce geometric distortions and create better paths, it also has several disadvantages. For example, more connections will increase the total computation effort of the model, since there will be more alternative routes for the model to choose from. Another disadvantage is that the processes of assigning cost value to a link become more complicated. A link may pass more than four raster cells and may be split into multiple segments by the raster cells (Fig. 2b), comparing with only two segments in Queen's pattern and Rook's pattern. To get the cost of the link, we need to calculate the cost of each link segment using the cost weight of the cell where the segment locates in and sum up the costs of all segments along each link. This also could result in longer processing time. To balance the advantages and disadvantages of different neighborhood patterns, we chose the Queen's pattern to construct our virtual network when implementing the Dijkstra's algorithm.

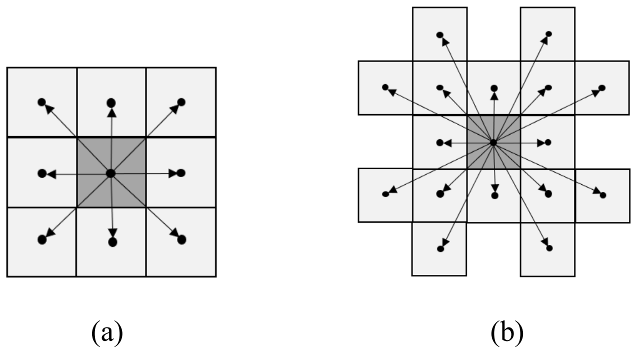


Fig. 2. Alternative neighborhood patterns. (a). Queen's pattern. (b). Knight's pattern.

We started from building a network using given raster map layers. We assumed each node in the network represents the center of a raster cell in the map. Multiple attribute values were assigned to each node. The first attribute is the spatial coordinate (row index and column index) of each raster cell. The relative positions of the eight neighboring nodes of each center node are represented as ((1, -1) (1, 0) (1, 1) (0, -1) (0, 1) (-1, -1) (-1, 0) (-1, 1)) following the Queen's pattern. To implement the Dijkstra's algorithm, the cost of the link between each pair of nodes will be stored as a node attribute. We split one link between two adjacent nodes into two segments at the edge (or corner) of raster cells (Fig. 3a), calculate the cost associated with each segment and store the cost to the node corresponding to the segment. The neighborhood topology between nodes will also be stored to the related nodes along with the cost of each link segment.

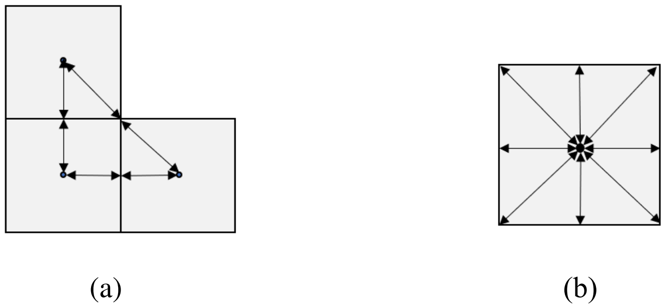


Fig. 3. (a). Each link is split into two segments. (b). The eight link segments connect to a node.

For horizontal or vertical (we will call it rectangular) links, the associated cost of a link segment is calculated as equation (1); and for diagonal links, the cost of a link segment is calculated as equation (2).

$$Cost_{\text{rectangular}} = \frac{1}{2} \cdot l_{cell} \cdot w \tag{1}$$

$$Cost_{\text{diagonal}} = \frac{\sqrt{2}}{2} \cdot l_{cell} \cdot w \tag{2}$$

$$w = m + 1 \tag{3}$$

where l_{cell} is the unit length of all raster cells, and w is the node specified multiplier used when calculating the weighted cost of link segments connected to the node. We derived the value of w from the node attribute value m extracted from the underlying raster cell attribute by applying equation (3); the constant 1 is added in the equation (3) to avoid the situation when an extremely small m may unrealistically cause the cost approaching 0. Sometimes a threshold of certain node attribute value can be applied to distinguish between viable or inviable link between adjacent nodes. If any node has its attribute values strictly falls out of the allowed range defined by the threshold, this node will be considered as inaccessible through the corresponding link and will be assigned a very large segment cost value. Every node would have eight weighted cost value attributes representing the weighted costs of the eight link segments connecting to itself (Fig. 3b).

After formulated a complete network with all possible links between nodes, we applied Dijkstra's algorithm on the network to find desired path to build firelines or management unit boundaries. However, unlike the path of minimum total length described in Dijkstra (1959), the ideal path with the least accumulated cost identified from our model doesn't necessarily reflect to the shortest path based on Euclidean distance. It is possible that our model's results include significantly longer and winding paths as exchange for safer or less difficult to build firelines or management unit boundaries. A flowchart demonstrated in Fig. 4 shows how we designed our model and where we implemented Dijkstra's algorithm in our model.

Our objective is to identify potential paths of firelines in a studied landscape by connecting the nodes on the network one by one. This process begins with identifying and sorting the potential nodes that can serve as the "source nodes" of new paths, i.e. the designated starting point of a segment of existing fireline that requires rework. The second step is to identify a list of nodes that

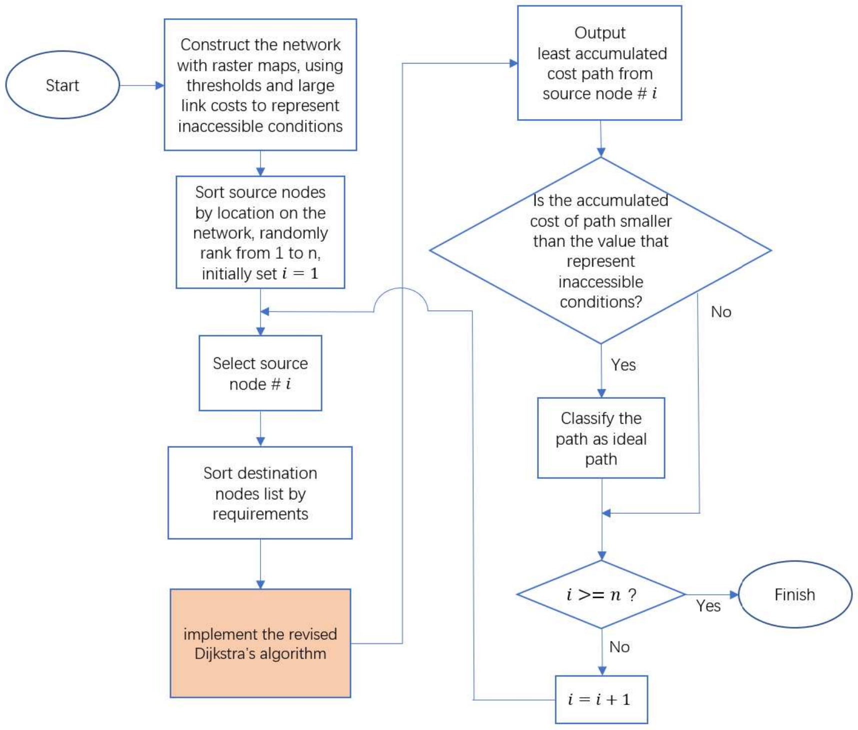

Start
Construct the network with raster maps, using thresholds and large link costs to represent inaccessible conditions
Sort source nodes by location on the network, randomly rank from 1 to n, initially set $i = 1$
Select source node # i
Sort destination nodes list by requirements
implement the revised Dijkstra's algorithm
Output least accumulated cost path from source node # i
Is the accumulated cost of path smaller than the value that represent inaccessible conditions?
No
Yes
Classify the path as ideal path
$i >= n$?
Yes
Finish
No
$i = i + 1$

Fig. 4. Flowchart of model implementation, the highlighted box shows Dijkstra's algorithm.

could serve as the "destinations" for each source node to connect to. Unlike the traditional shortest path study which only has single fixed destination node, our model needs to adapt to the situation of multiple possible destination nodes. The third step is to implement Dijkstra's algorithm on the network to connect the identified source nodes and their corresponding destination nodes. The pseudocode of our revised Dijkstra's algorithm based on the coding structure of Bertrand (2014) is shown in Algorithm 1. This algorithm will enumerate and calculate the least accumulated costs recursively by nodes from the source node until it successfully reaches one candidate destination node and forms a path. The fourth step is to evaluate if the identified path passes any inaccessible conditions. If so, the path emanating from this source node will be abandoned, otherwise it will be classified as an ideal path. The second to fourth steps will be repeated for every source node until all the identified source nodes have been processed. Finally, all the ideal paths will be digitized back to map format for further analysis and evaluation.

Algorithm 1: Pseudocode of the revised Dijkstra's algorithm:

```
This program calculates and exports the Least accumulated cost path
from the defined source node to one of the candidate destination nodes

define an empty list called visited
define an empty dictionary called cost
define an empty dictionary called predecessors
define Dijkstra's algorithm (network graph, source node, destination nodes, visited, cost, predecessors):
    # Ending condition:
    if source node falls in destination nodes:
        define an empty list called path
        pred = source node
        while pred != none:
            append path with pred
            from predecessors get the last predecessor and name it pred, using key=pred
            if this key doesn't exist, get none here
        The least accumulated cost path can be found in path
        The least accumulated cost can be found by cost [source node]

    else:
        In the initial run, initializes the accumulated cost for the source node and save in cost
        cost [source node] = 0

        for neighbors of the source node in network graph:
            if a neighbor hasn't been visited:
                new cost = cost [source node] + network graph [source node][neighbor] (link cost
                between source node and this neighbor)
                if new cost < cost.get (neighbor, infinite):
               # from cost get the cost value using key=neighbor, if this key doesn't exist, get infinite
               here.
                    cost [neighbor] = new cost
                    predecessors [neighbor] = source node
        # Mark the source node as visited
        visited.append (source node)

        # After all neighbors have been visited, select the non-visited nodes
        define an empty dictionary called nonvisited
        for nodes in network graph:
            if nodes not in visited:
                nonvisited [nodes] = cost.get (node, infinite)
                # from *cost* get the cost value using key=node, if this key doesn't exist, get infinite here.
        find the node x in *nonvisited* with the smallest *cost* [x]

        # run Dijskstra algorithm with source node = x
        **Dijskstra algorithm** (*network graph*, x, *destination nodes*, *visited*, *cost*, *predecessors*)
```

Chapter 3
Study site and test cases

3.1 Study site description

We tested our model in a study area at the northwest portion of the Sierra National Forest in California with a total area of 670 km^2. About half of the study area is located within 2018 Ferguson Fire perimeter (Fig. 5). To improve fireline construction, we selected a few fireline segments during the Ferguson Fire suppression (the July 15th proposed event line) to demonstrate how they can be improved (Fig. 6). These selected segments are parts of the proposed firelines when the fire had not escaped from the initial attack efforts; those proposed firelines locate along the directions where fire finally escaped. The selected segments were anchored directly to some existing road features. We want to test whether these proposed fireline segments are the most effective and the safest and whether there might be potentially better layouts for those firelines. This represents a test in a scenario from real-world fire incident to use our model to provide better real-time line construction suggestions.

3.2 Model implementation and test cases

For our test, we implemented the following analysis processes. We selected a minimum rectangle shaped area that fully encloses our study area. This selected area is rasterized into 1177×1473 cells (column by row), and the size of each cell is 30×30m. Feature layers in vector format are converted

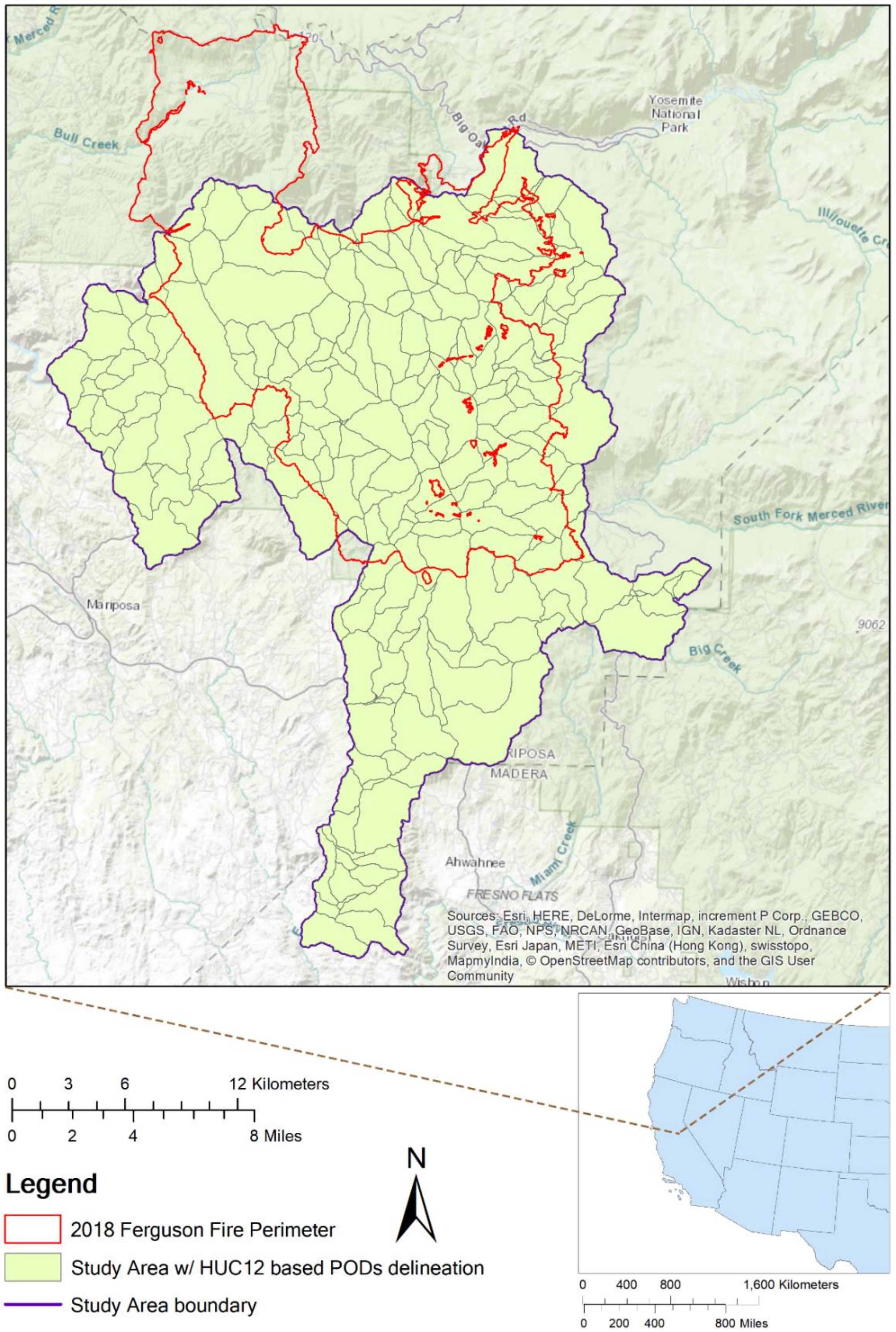

Merced R
Bull Creek
Big Oak Rd
120
Yosemite National Park
Illilouette Cr
South Fork Merced River
Mariposa
9062
Big Creek
MARIPOSA
MADERA
Ahwahnee
Miami Creek
FRESNO FLATS
Sources: Esri, HERE, DeLorme, Intermap, increment P Corp., GEBCO, USGS, FAO, NPS, NRCAN, GeoBase, IGN, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), swisstopo, MapmyIndia, © OpenStreetMap contributors, and the GIS User Community
0 3 6 12 Kilometers
0 2 4 8 Miles
Legend
2018 Ferguson Fire Perimeter
Study Area w/ HUC12 based PODs delineation
Study Area boundary
N
0 400 800 1,600 Kilometers
0 200 400 800 Miles

Fig. 5. Study area location including POD boundaries delineated based on the HUC12 watersheds, the 2018 Ferguson Fire perimeter and the general location of the study site in the western continental United States.

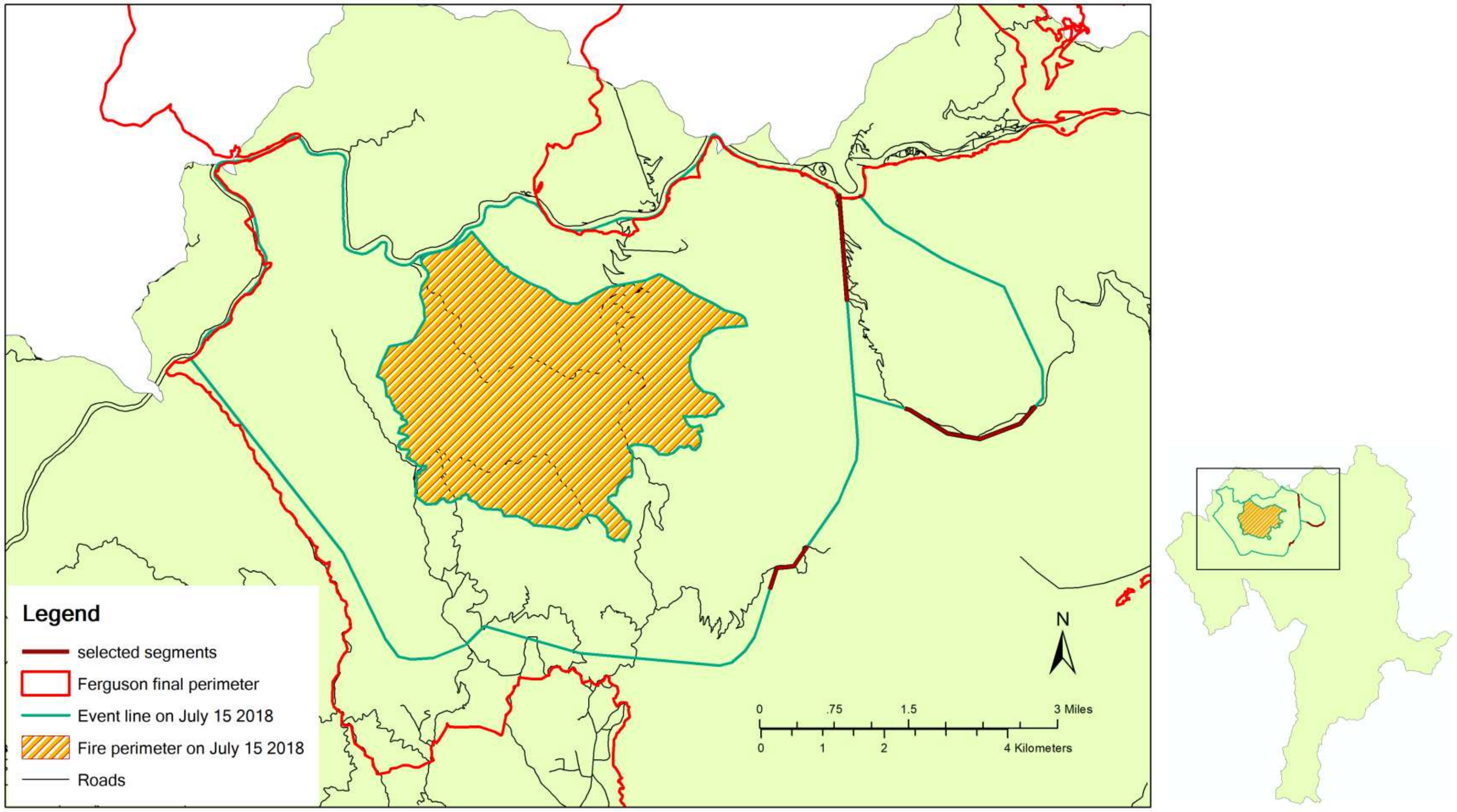


Fig. 6. Part of July 15th proposed event line of Ferguson Fire, along with the fire perimeter on that date and its relative position to the final fire perimeter.

to raster map format. We then combined all the information retrieved from the raster layers and the converted feature layers into a three-dimensional array: the x-dimension represents row index; the y-dimension represents column index; the z-dimension is used to store multiple raster cell attributes that will be used in later analysis. The specific structure arrangement of the three-dimensional array is shown in Table 1.

Table 1: Three-dimensional array (referenced as *Array*[ ][ ][ ] later) structure and metadata:

x-dimension:	Row index: x
y-dimension:	Column index: y
z-dimension:	• *Category 1*: Raster value attributes retrieved from spatial index raster map layers. The values in this category will be primarily used to identify inaccessible nodes by defined thresholds. The 1st value in this category will also be used in calculating the node specific multiplier w for the weighted cost of link segments between nodes. In the array the z-dimension indices of those values are denoted as $z_1, z_2, z_3 \dots z_i$. • *Category 2*: Node type: The values in this category will be used to identify the types of the nodes which determines how the node will be categorized in the network. For example, we used this to classify source nodes, destination nodes, the nodes that are neither source nor destination, or nodes that are out of our study area. In the array the z-dimension indices of those values are denoted as $z_{i+1}, z_{i+2}, z_{i+3} \dots z_{i+j}$. • *Category 3*: Weighted cost of link segments associated to a node: When applying Queen's pattern to construct network, the link segments around a node are facing its neighboring nodes at these directions: N, NE, E, SE, S, SW, W, NW. In the array the z-dimension indices of those values are denoted as $z_{i+j+1}, z_{i+j+2}, \dots z_{i+j+8}$ corresponding to the order of the eight directions.

After populated the three-dimensional array, we formulated a network based on information of the array. Each node is represented by a unique pair of spatial coordinates, so there are 1177×1473=1,733,721 nodes in total within our network. For a node at location (x, y), the weighted cost of links between itself and its eight neighbors are calculated as Fig. 7 and Table 2. After the network was built, ideally, we could run the revised Dijkstra's algorithm to identify the least accumulated cost paths between nodes.

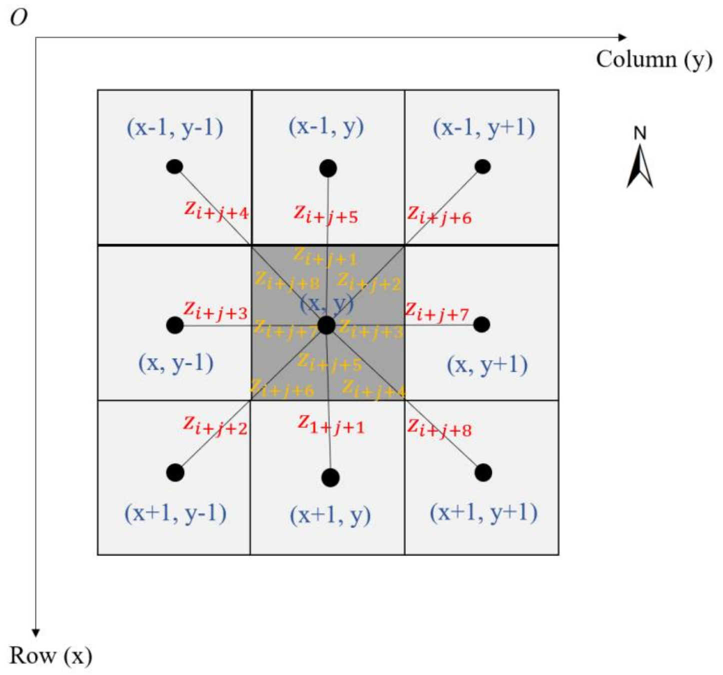


Fig. 7. The node (x, y) and its eight neighbors on the network, along with the array index of the link segments.

Table 2: The weighted cost of links between a node (x, y) and its eight neighbors:

Center node	neighbor node	weighted cost of links between these two nodes
(x, y)	(x-1, y)	$Array[x][y][z_{i+j+1}] + Array[x-1][y][z_{i+j+5}]$
	(x-1, y+1)	$Array[x][y][z_{i+j+2}] + Array[x-1][y+1][z_{i+j+6}]$
	(x, y+1)	$Array[x][y][z_{i+j+3}] + Array[x][y+1][z_{i+j+7}]$
	(x+1, y+1)	$Array[x][y][z_{i+j+4}] + Array[x+1][y+1][z_{i+j+8}]$
	(x+1, y)	$Array[x][y][z_{i+j+5}] + Array[x+1][y][z_{i+j+1}]$
	(x+1, y-1)	$Array[x][y][z_{i+j+6}] + Array[x+1][y-1][z_{i+j+2}]$
	(x, y-1)	$Array[x][y][z_{i+j+7}] + Array[x][y-1][z_{i+j+3}]$
	(x-1, y-1)	$Array[x][y][z_{i+j+8}] + Array[x-1][y-1][z_{i+j+4}]$

However, because the Dijkstra's algorithm requires large amount of temporary data to be saved in the RAM, we found out that our computer hardware does not have enough memory to process all the 1,733,721 nodes selected in the network at once. To overcome the computer memory limitation, we use sectors of a circle to define "searching ranges" to limit the size of network. Each searching range (sector) is defined by three parameters: orientation (α, the direction of an angular bisector, in compass degrees), angle between the two radial boundaries (θ) and searching radius (r) from the identified source node. Those parameters can be adjusted. With fixed computer memory availability, if we want to connect a source node to destination nodes that are far away, we may need to increase the value of r, and meanwhile, decrease θ; if we want to search for a broader angular range, we can increase θ, but meanwhile, decrease r and select an α that faces the direction where the potential connection path would most likely be along; if we have no pre-knowledge of the probable direction of potential connection path, we can select θ=360° and select a small r. A demonstration of this situation is shown in Fig. 8. After the searching range is defined, only nodes falling in or on the edge of the searching range will be selected to form the network to run Dijkstra's algorithm in our model.

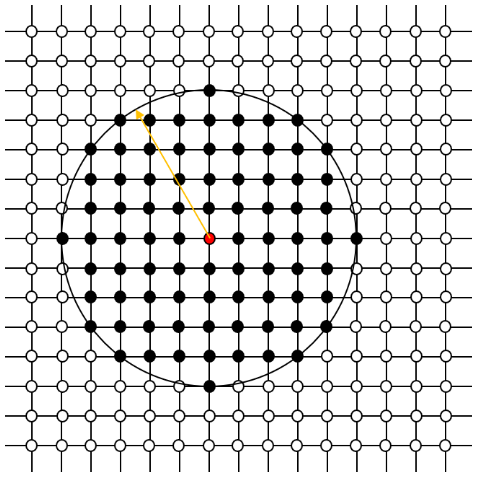


Fig. 8. The selected nodes when parameters of searching range are assigned as: $\theta = 360°$, $r = 5$.

To test our model, we used SDI raster layer as both the cost surface and the threshold surface to improve fireline construction location selections. SDI was originally calculated using equations (4) and (5) based on the methods described in Rodríguez y Silva et al. (2014). However during generating the SDI raster layer, aerial resources data were not available for calculation. Thus instead of using the original SDI, we used a slightly modified equation (6) by removing aerial resources sub-index from the equation (4), and rename the generated spatial index as terrestrial SDI (tSDI). The index tSDI was developed from SDI after excluded factors influencing the aerial suppression resources efficiency (O'Connor et al., 2017).

$$SDI = [\frac{\sum I_{ce}}{\sum(I_a+I_m+I_p+I_{ar}+I_c)}] \tag{4}$$

$$I_{ce} = \sum[\left(2 \times FL_i \times \frac{HUA_i}{FL_i+HUA_i}\right) \times \left(\frac{A_i}{A_t}\right)] \tag{5}$$

$$tSDI = [\frac{\sum I_{ce}}{\sum(I_a+I_m+I_p+I_c)}] \tag{6}$$

(I_{ce}: energy behavior sub-index; I_a: accessibility sub-index; I_m: mobility sub-index, I_p: penetrability sub-index; I_{ar}: aerial resources sub-index; I_c: fireline opening sub-index.)

In this case, the sub-indices of tSDI we used were generated under 80th percentile weather data (10 mph windspeed; most frequent wind direction 250-degree azimuth; 1-hr fuel moisture 4%, 10-hr fuel moisture 5%, 100-hr fuel moisture 7%, herbaceous fuel moisture 30%, woody fuel moisture 60%) of El Portal weather station. The generated tSDI raster layer for the study area is shown in Fig. 9.

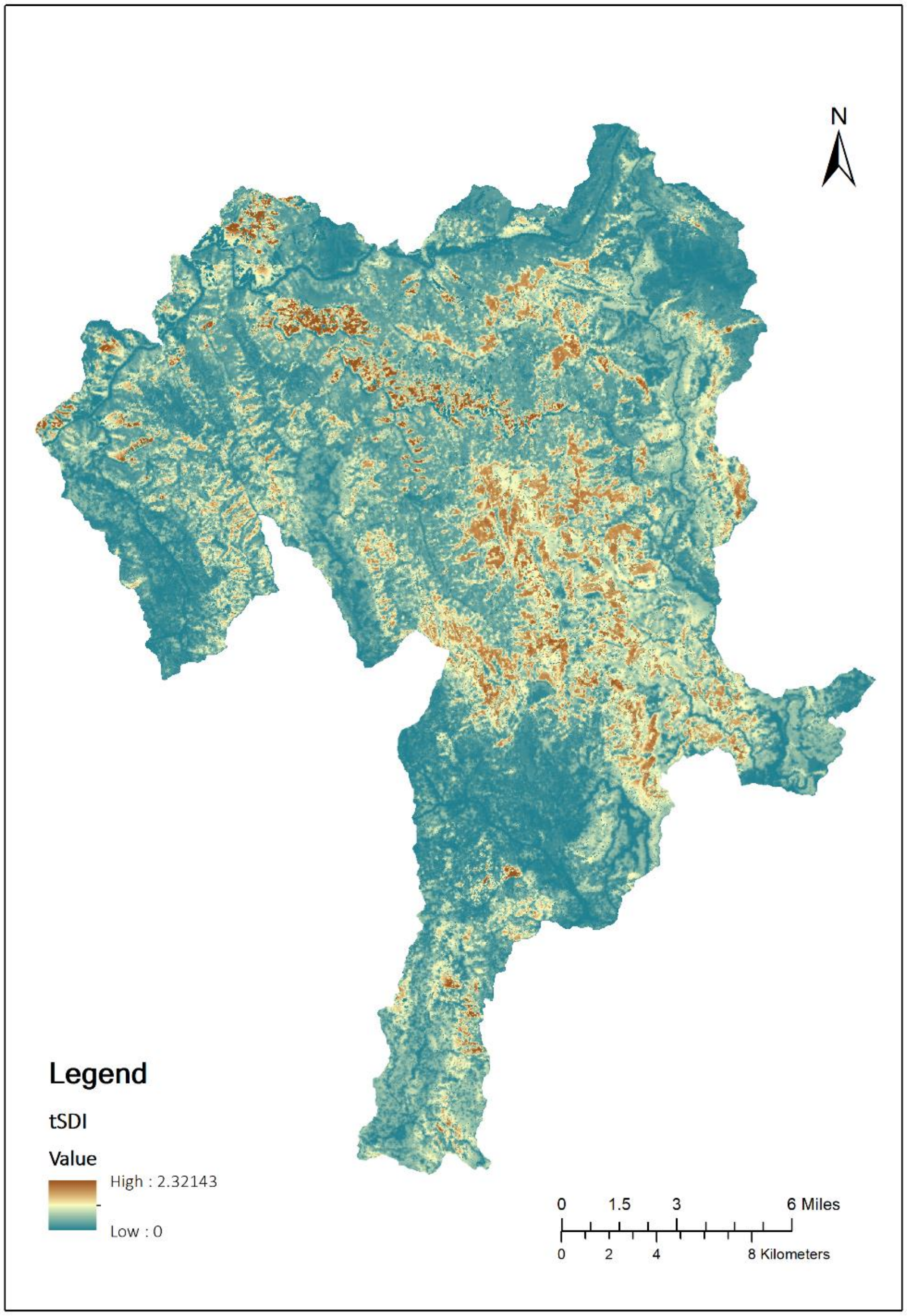


Fig. 9. The generated tSDI raster layer for the study area.

For convenience, we first assigned a serial code to each of the selected proposed fireline segments (Fig. 10). We then conducted a set of tests to connect the end nodes of those proposed lines either using the shortest Euclidian distance or the least accumulated cost weighted by tSDI. Euclidian distance will be called "unweighted cost" in this study for distinction with tSDI weighted costs. We kept a similar connectivity pattern in the newly identified fireline construction location in order to make comparison with the original one: we tested to connect the south end of proposed fireline segment 1 with the west end of proposed fireline segment 3, the north end of proposed fireline segment 2 to the west end of proposed fireline segment 3, and the east end of proposed fireline segment 3 to proposed fireline segment 4, using Dijkstra's algorithm. The tested searching ranges to find ideal paths for connecting the proposed fireline segments are highlighted in Fig. 11 and marked from A to F. Notice that the orientation of each searching range is not perfectly align with the connection between the ending points of proposed fireline segments for simplification purpose. The "wedge-shaped" searching range has a narrower part at one end. This would likely omit the optimal paths not entirely included in the searching range. To somewhat ease this concern, we also conducted conjugate runs between each pair of nodes. The searching ranges of both runs are oriented oppositely so that the broader section of one searching range can cover another's narrower section. If each of the paired runs successfully find an ideal path, we will compare them and keep the path with a better performance (lower accumulated cost, lower threshold, etc.). The parameters of each searching range are selected to accommodate our computer hardware capacity (Table 3).

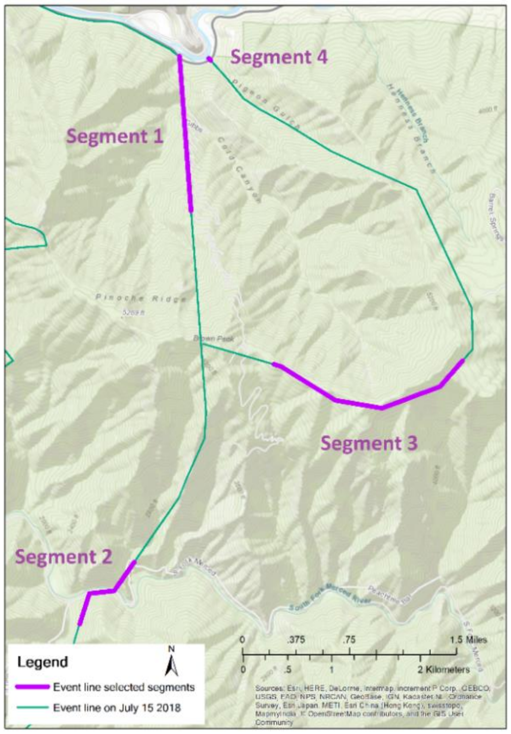


Fig. 10. Selected proposed fireline segments and serial codes.

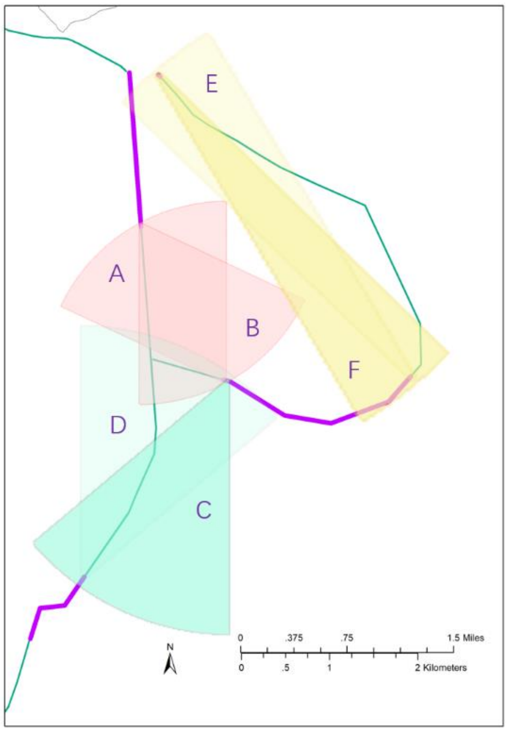


Fig. 11. Searching ranges following the original connectivity pattern.

Table 3. Searching range parameters:

Searching range	Orientation α (°)	Angle θ (°)	Searching radius r (cell length, 30m)
A	330°	60°	70
B	150°	60°	70
C	202.5°	45°	100
D	22.5°	45°	100
E	322.5°	15°	160
F	142.5°	15°	160

We started our initial tSDI upper bound threshold from 0.8 for all predefined searching ranges and ran the revised Dijkstra's algorithm. If there is not an ideal path found using the current threshold level, we would increase the threshold by 0.1 and rerun the algorithm. The highest tolerable tSDI threshold was set to be 1.2 for all predefined searching ranges.

Chapter 4
Results

After we processed all the 6 searching ranges, we found that searching range A, B, C and F successfully identified ideal paths. Searching range C and F identified the paths (summarized in path set C and path set F along with manager proposed firelines) when tSDI threshold was set to 0.8; searching range B identified the paths (summarized in path set B along with manager proposed firelines) when tSDI threshold was set to 0.9; searching range A identified the paths (summarized in path set A along with manager proposed firelines) when tSDI threshold was set to 1.0. Since searching range A and B are a pair of conjugates, and identified paths in path set B was processed under a lower tSDI threshold than identified paths in path set A, we considered path set B has a better performance and aborted path set A. Fig. 12 is a schematic map showing this model suggested new fireline paths along with the manager proposed firelines during the active fire incident. From Fig. 12, we also noticed that a segment of road connects proposed fireline segment 1 and 3, but was neither selected by fire manager as part of proposed firelines, nor selected by our model as a new fireline path in path set B. In order to make comparison in tSDI performance, we will also have this road segment included in path set B and test it along with other paths in the same path set.

Projecting model suggested new fireline paths along with manager proposed firelines on a 3D view

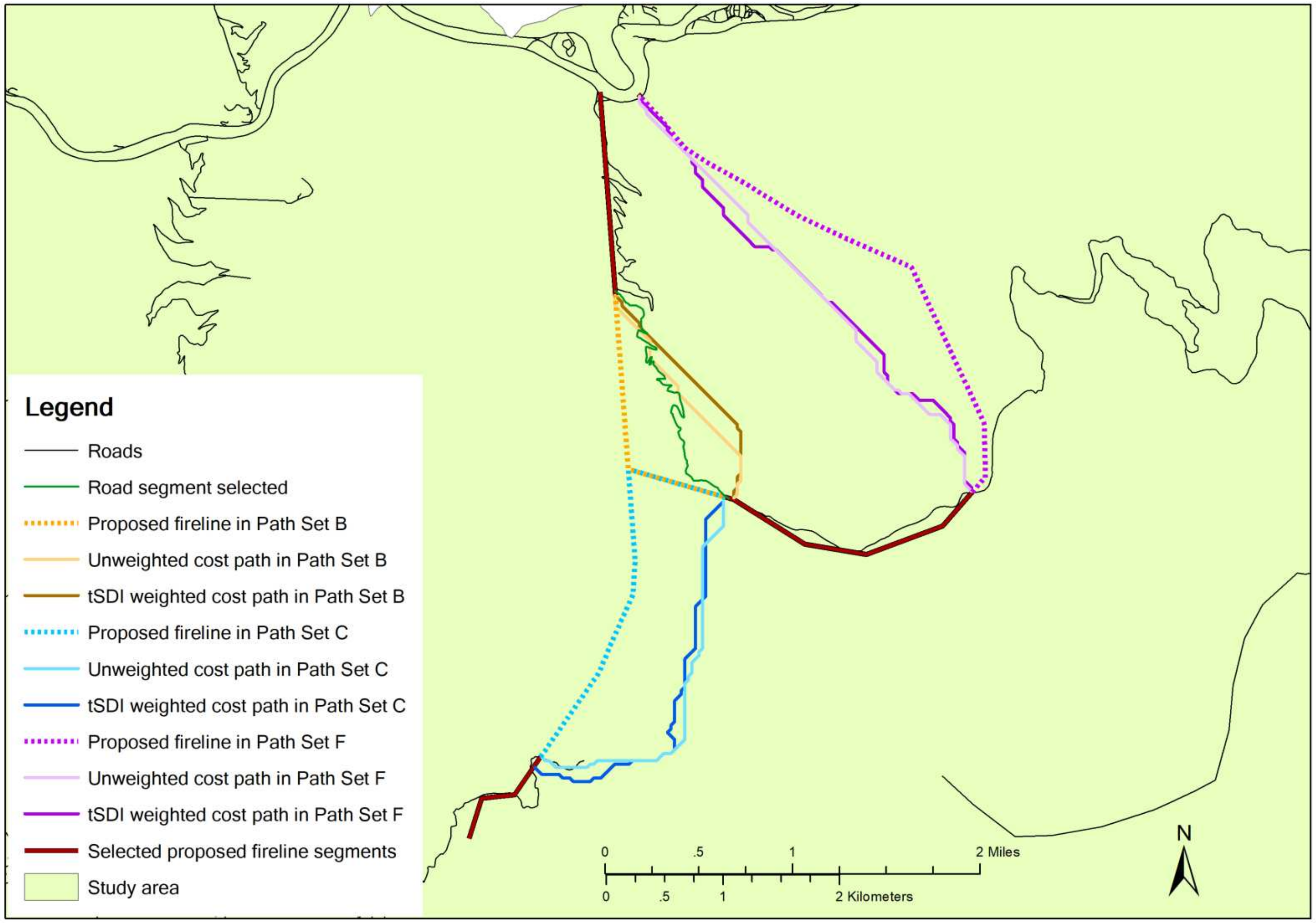
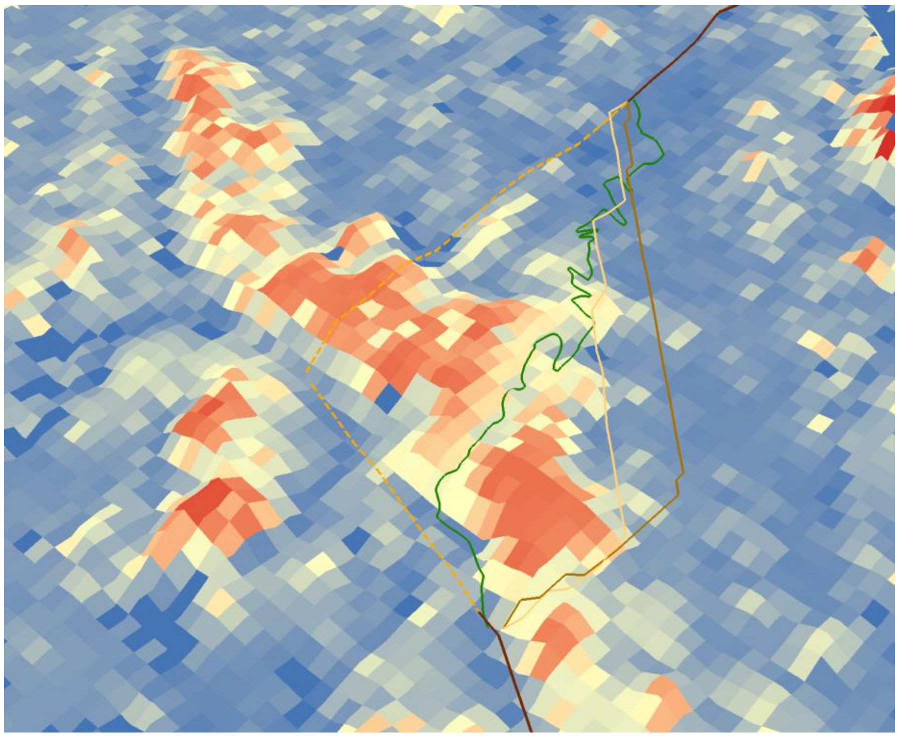
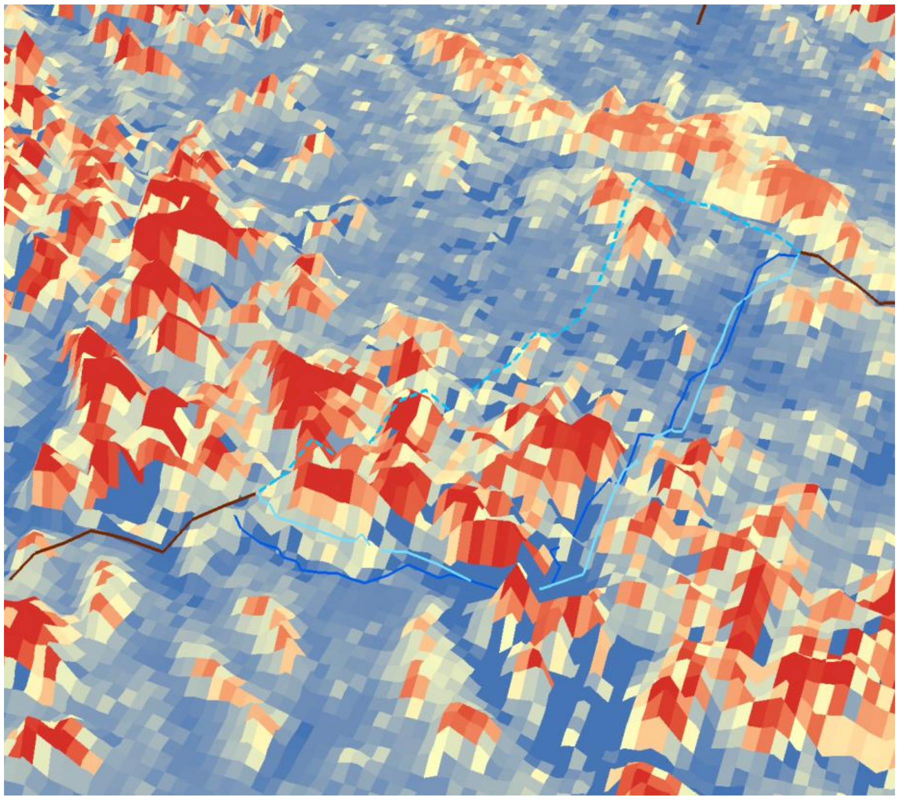

Legend
Roads
Road segment selected
Proposed fireline in Path Set B
Unweighted cost path in Path Set B
tSDI weighted cost path in Path Set B
Proposed fireline in Path Set C
Unweighted cost path in Path Set C
tSDI weighted cost path in Path Set C
Proposed fireline in Path Set F
Unweighted cost path in Path Set F
tSDI weighted cost path in Path Set F
Selected proposed fireline segments
Study area
0 .5 1 2 Miles
0 .5 1 2 Kilometers
N

Fig. 12. Comparison between new fire control line paths and manager proposed fire control lines.

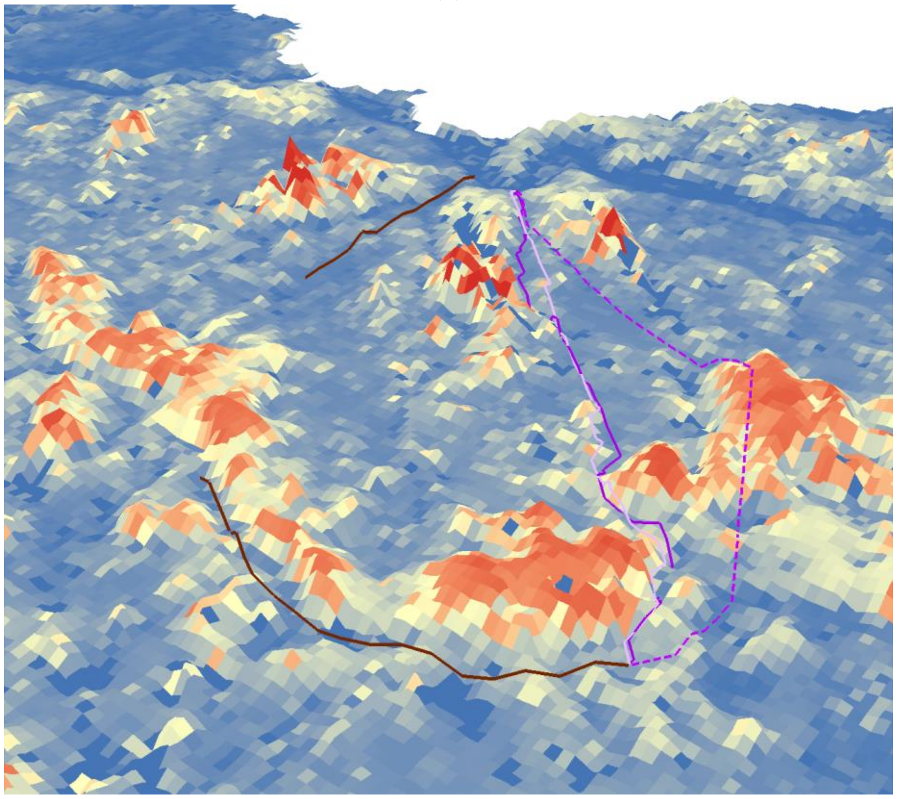
of the tSDI surface helps us visualize the effect of different fireline options (Fig. 13). The “ridges” marked in red represent the areas of high tSDI value as undesirable suppression conditions; the “valleys” marked in blue represent the areas of low tSDI value as more preferred suppression conditions. All maps show that firelines proposed by manager climb over at least one high tSDI “ridge”; in contrast, the model suggested new fireline paths bypassing the high tSDI areas. Comparing to the unweighted cost paths, paths with tSDI weighted cost tend to travel through lower tSDI area, however with a tradeoff of going through a more zigzagged route. In path set B only, the road segment which connects proposed fireline segment 1 and 3 avoids the high tSDI areas, but travels across medium range tSDI areas with a longer distance comparing with model suggested new fireline paths, and shows an extremely zigzagged shape (Fig. 13(a)).



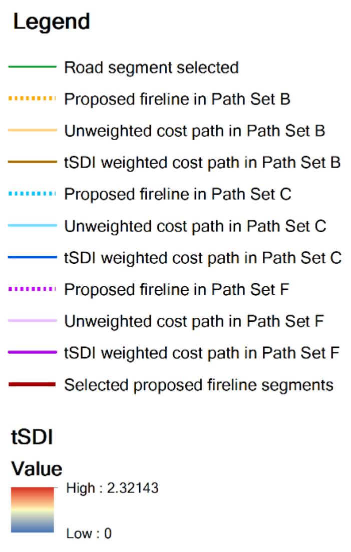
(a)



(b)



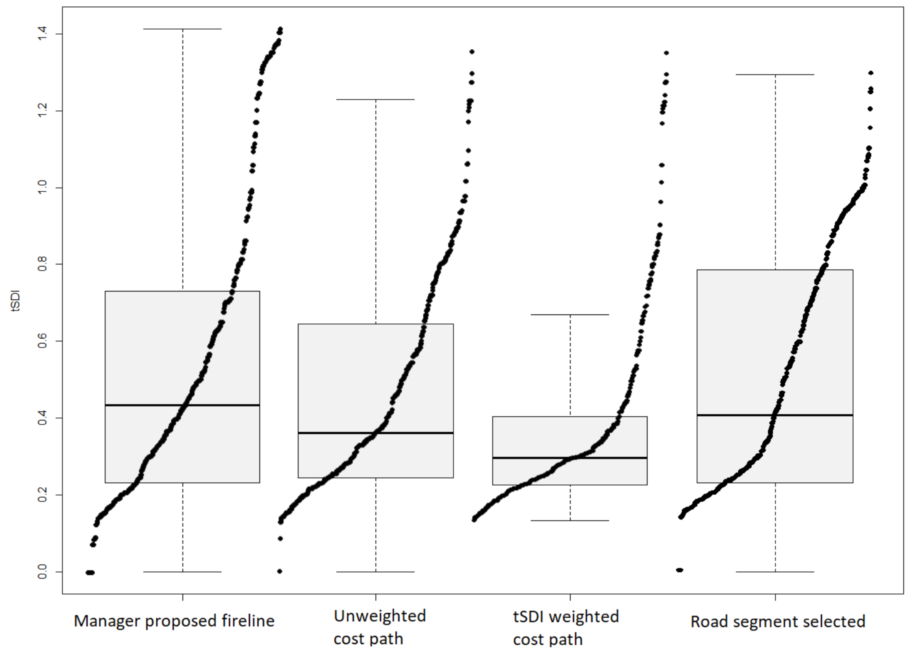
(c)


Legend
Road segment selected
Proposed fireline in Path Set B
Unweighted cost path in Path Set B
tSDI weighted cost path in Path Set B
Proposed fireline in Path Set C
Unweighted cost path in Path Set C
tSDI weighted cost path in Path Set C
Proposed fireline in Path Set F
Unweighted cost path in Path Set F
tSDI weighted cost path in Path Set F
Selected proposed fireline segments
tSDI
Value
High : 2.32143
Low : 0

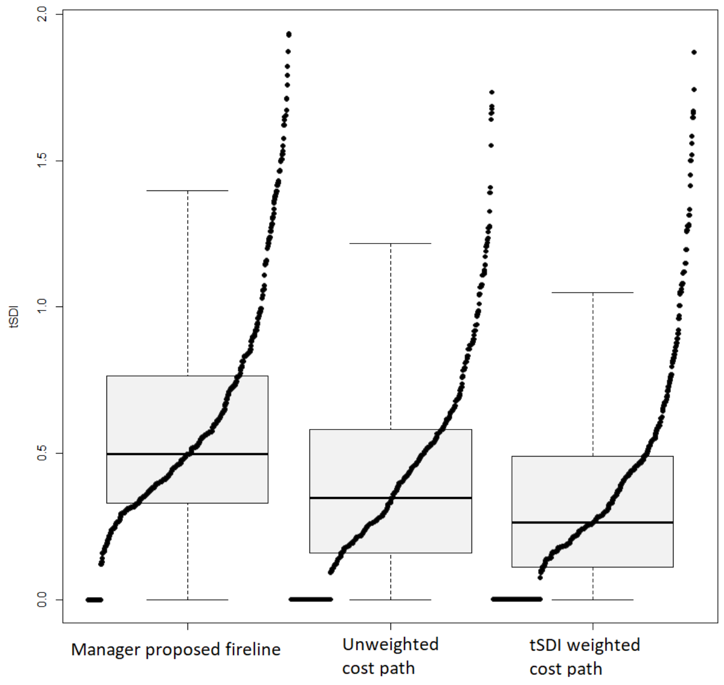
(d)

Fig. 13. Model suggested new fireline paths and manager proposed firelines projected on a 3D view of the tSDI raster. (a) Path set B including manager proposed fireline and road segment which connects proposed fireline segment 1 and 3. (b) Path set C including manager proposed fireline. (c) Path set F including manager proposed fireline. (d) Legend.

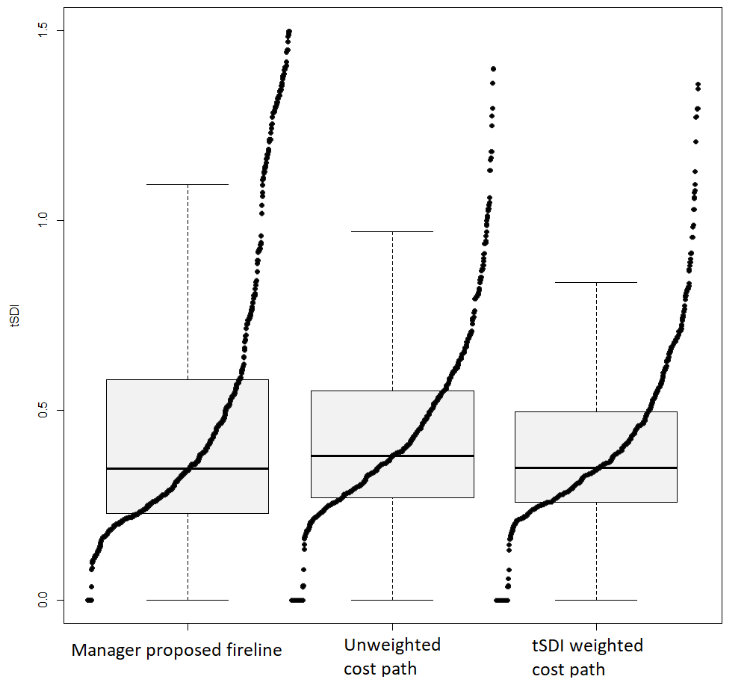
To summarize the difference between those paths through simple statistics, we created a 60m (2 cell length units) buffer along each path, and randomly generated 1000 points within each buffer. We then extracted tSDI values of these points and built a series with 3 sets of hybrid plots which include boxplot and cumulative stripplot (Fig. 14) to verify if the paths' adjacent corridors are also safe for conducting firefighting activities. In path set B and C, the sample's medians and highest values excluding outliers are significantly lower in both unweighted cost path and tSDI weighted cost path, in contrast with the manager proposed firelines; along model suggested new fireline paths in path set F, although the decrease in sample median is not obvious, the decrease in the highest values excluding outliers is significant. These results support our observation in Fig. 13. It demonstrates that model suggested new fireline paths traveling across areas with overall lower tSDI levels. By comparing the unweighted and weighted cost paths, we found the median of samples are slightly lower in tSDI weighted cost paths than the unweighted cost counterpart. In addition, for all three path sets, the tSDI weighted distance paths have a lower upper quantile within their sample distribution. These demonstrates that the adjacent corridors of tSDI weighted cost paths have even lower suppression difficulty levels. However, the tradeoff of this improvement is that it may increase the path's Euclidian distance and the chance of a zigzagged fireline. For example, both path set C and F demonstrate an increase of Euclidian distance by considering the tSDI weighted cost comparing with only using the unweighted cost (Table 4).



(a)



(b)



(c)

Fig. 14. Hybrid plots for 1000 random points within the 60m radius buffer of each path. (a) Path set B. (b) Path set C. (c) Path set F.

Table 4. The distance difference between paths in each path set (Unit: m):

Path set	Unweighted cost path	tSDI Weighted cost path	Proposed fireline	Road segment
B	2212.20	2212.20	2359.32	3321.64
C	3573.38	3745.22	3543.29	
F	4675.95	4816.54	4890.21	

Comparing with other paths in path set B, the selected road segment has a similar tSDI distribution to the manager proposed fireline, with slightly but not significantly lower median and the highest value; it also has a significant higher median and the highest value than the model suggested new fireline paths (Fig. 14a). With little improvement in tSDI within the adjacent corridor, significantly increased Euclidian distance (Table 4) and a zigzagged shape, there is no reason to choose this road segment for fireline construction based on tSDI surface. This comparison proves that our model does not always identify paths following existing roads, even if the selected spatial index is affected by road adjacency.

Chapter 5
Conclusion and discussion

In this paper, we demonstrated the use of a raster-surface-based spatial optimization model to help improve fireline safety and efficiency during real-world fire incident suppression operations. Using tSDI or other spatial indices as cost surfaces or threshold surfaces allows us to locate places that are easier and safer to construct fireline while avoid areas that might not be ideal for firefighting.

There are still some limitations of this approach that may lead to compromised solutions that need to be addressed in following up studies. For example, we used “searching ranges” to limit the network size to compensate for computer memory capacity. When the source and destination nodes locate far away from each other, we have to apply a narrower searching range. Future research needs to focus on optimizing memory management of the Dijkstra’s algorithm and increasing the searching range to identify better fireline routes and locations. Another possible limitation of the current approach is that our model may identify paths with zigzagged shape to avoid areas with higher tSDI values. Zigzag-shaped fireline is seldomly used in real world wildfire suppression scenario, because it will inevitably create multiple budges along its edge and cause some areas behind the fireline less safe for suppression operation due to the risk of fire breaching at the budges. A zigzag-shaped fireline is also less efficient to construct because the usage of heavy equipment is limited under this situation. In the future research we need to develop an algorithm to promote less zigzagged paths.

Our current model calculates optimal fireline paths based on many input raster surfaces generated using preselected weather and fuel conditions data. When used for incident fireline improvement, a key improvement would be using real time (or forecasted, depending on management context) dynamic weather conditions to update SDI or other related raster surfaces to support the fireline selections. Thus, a module that can dynamically integrate the most recent information and generate real time raster input layers need to be developed and embedded into the analytical process.

Beside this, in the future research, our model could be used to further delineate relatively large fire management units into smaller units to support more flexible fire management. When further delineating fire management units, instead of just considering one or multiple spatial indices, we can also consider other metrics that could be used to split a fire management unit, such as ecological conditions, resource values and distribution of critical infrastructures, fire benefits or losses, etc. For example, aggressive suppression may be suitable for units with high value resources or properties resided within, while low intensity wildfire might be beneficial in other units that contain fire adaptive ecosystems. Our expectation is to improve the applicability of our model through the proposed future research.

References

Andrews, Patricia L.; Collin D. Bevins and Robert C Seli. 2008. BehavePlus fire modeling system, version 4.0: User's Guide. Gen. Tech. Rep. RMRS-GTR-106WWW Revised. Ogden, UT: Department of Agriculture, Forest Service, Rocky Mountain Research Station. 116p.

Andrews, Patricia L. 2014. Current status and future needs of the BehavePlus Fire Modeling System. *International Journal of Wildland Fire* 2014, 23: 21–33. doi:10.1071/WF12167

Antikainen, Harri. 2013. Comparison of different strategies for determining raster-based least-cost paths with a minimum amount of distortion. *Transactions in GIS*, 2013, 17(1): 96–108. doi: 10.1111/j.1467-9671.2012.01355.x

Bertrand, Gilles. 2014. Dijkstra Algorithm: How to implement it with python (solved with all explanations)? Retrieved from http://www.gilles-bertrand.com/2014/03/dijkstra-algorithm-python-example-source-code-shortest-path.html

Calkin, David E. and Kevin D. Hyde. 2004. Break-even point: Suppression cost analyses in Montana weigh resource values as determined by tax records and available GIS data. *Wildfire Mag*. 13: 14-21.

Calkin, David E.; Krista M. Gebert; J. Greg Jones and Ronald P. Neilson. 2005. Forest Service Large Fire Area Burned and Suppression Expenditure Trends, 1970-2002. *Journal of Forestry* Vol. 103, No. 4, June 2005: 179-183.

Calkin, David E.; Matthew P. Thompson; Mark A. Finney and Kevin D. Hyde. 2011. A Real-Time Risk Assessment Tool Supporting Wildland Fire Decisionmaking. *USDA Forest Service / UNL Faculty Publications.* 359.

Calkin, David E.; Tyron Venn; Matthew Wibbenmeyer and Matthew P. Thompson. 2012. Estimating US federal wildland fire managers' preferences toward competing strategic suppression objectives. *International Journal of Wildland Fire* 22(2): 212-222. doi:10.1071/WF11075

Church, Richard L.; Scott R. Loban and Kristi Lombard. 1992. An interface for exploring spatial alternatives for a corridor location problem. *Computers & Geosciences* Vol. 18, No. 8 (1992): 1095-1105.

Cova, Thomas J.; Philip E. Dennison; Tae H. Kim and Max A. Moritz. 2015. Setting wildfire evacuation trigger points using fire spread modeling and GIS. *Transactions in GIS*, 2005, 9(4): 603–617.

Dijkstra, Edsger Wybe. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1 (1959): 269-271.

Dunn, Christopher J.; Christopher D. O'Connor; Matthew J. Reilly; Dave E. Calkin and Matthew P. Thompson. 2019. Spatial and temporal assessment of responder exposure to snag hazards in post-fire environments. *Forest Ecology and Management* 441 (2019): 202–214. doi:10.1016/j.foreco.2019.03.035

Eldrandaly, Khalid A.; Mohammed M. Hassan and Nabil M. AbdelAziz. 2015. A modified artificial bee colony algorithm for solving least-cost path problem in raster GIS. *Applied Mathematics & Information Sciences* 9, No. 1, 1-8 (2015). doi:10.12785/amis/090119

Finney, Mark A. 1998. FARSITE: Fire Area Simulator—Model Development and Evaluation. Research Paper RMRS-RP-4 Revised. Fort Collins, CO. Rocky Mountain Research Station. United States Department of Agriculture, Forest Service.

Finney, Mark A.; David B. Sapsis and Berni Bahro. 2002. Use of FARSITE for Simulating Fire Suppression and Analyzing Fuel Treatment Economics. Fire in California Ecosystems: Integrating Ecology, Prevention, and Management. Association for Fire Ecology Miscellaneous Publication No.1:121-136.

Fried, Jeremy S. and Burton D. Fried. 1996. Simulating Wildfire Containment with Realistic Tactics. *Forest Science* 42(3) 1996: 267-281.

Gebert, Krista M. and Anne E. Black. 2012. Effect of suppression strategies on federal wildland fire expenditures. *Journal of Forestry* March 2012: 65-73.

Goodchild, Michael F. 1977. An evaluation of lattice solutions to the problem of corridor location. *Environment and Planning A*, 1977 Volume 9: 727-738.

Liang, Jingjing; Dave E. Calkin; Krista M. Gebert; Tyron J. Venn and Robin P. Silverstein. 2008. Factors influencing large wildland fire suppression expenditures. *International Journal of Wildland Fire* 2008(17): 650–659.

Mitsopoulos, Ioannis; Giorgos Mallinis; Sergiy Zibtsev; Mehmet Yavuz; B. Saglam; O. Kucuk; V. Bogomolov; A. Borsuk and G. Zaimes. 2016. An integrated approach for mapping fire suppression difficulty in three different ecosystems of Eastern Europe. *Journal of Spatial Science* 2016. doi: 10.1080/14498596.2016.1169952

Narayanaraj, Ganapathy and Michael C. Wimberly. 2011. Influences of forest roads on the spatial pattern of wildfire boundaries. *International Journal of Wildland Fire* 2011, 20: 792–803.

National Wildfire Coordinating Group. 2014. Wildland Fire Incident Management Field Guide. National Interagency Fire Center, PMS 210, NFES 002943. (Boise, ID).

O'Connor, Christopher D.; Matthew P. Thompson and Francisco Rodríguez y Silva. 2016. Getting ahead of the wildfire problem: quantifying and mapping management challenges and opportunities. *Geosciences* 2016, 6, 35. doi:10.3390/geosciences6030035

O'Connor, Christopher D.; David E. Calkin and Matthew P. Thompson. 2017. An empirical machine learning method for predicting potential fire control locations for pre-fire planning and operational fire management. *International Journal of Wildland Fire* 2017, 26: 587–597. doi:10.1071/WF16135

Rodríguez y Silva, Francisco; Juan Ramón Molina Martínez and Armando González-Cabán. 2014. A methodology for determining operational priorities for prevention and suppression of wildland fires. *International Journal of Wildland Fire* 2014, 23: 544–554. doi:10.1071/WF13063

Stepanov, Alexander and James MacGregor Smith. 2012. Modeling wildfire propagation with Delaunay triangulation and shortest path algorithms. *European Journal of Operational Research* 218 (2012): 775–788. doi:10.1016/j.ejor.2011.11.031

Thompson, Matthew P.; Phil Bowden; April Brough; Joe H. Scott; Julie Gilbertson-Day; Alan

Taylor; Jennifer Anderson and Jessica R. Haas. 2016. Application of wildfire risk assessment results to wildfire response planning in the southern Sierra Nevada, California, USA. *Forests* 2016, 7, 64. doi:10.3390/f7030064

Thompson, Matthew P.; Karin L. Riley; Dan Loeffler and Jessica R. Haas. 2017. Modeling Fuel Treatment Leverage: Encounter Rates, Risk Reduction, and Suppression Cost Impacts. *Forests* 2017, 8, 469. doi:10.3390/f8120469

Thompson, Matthew P.; Zhiwei Liu; Yu Wei; Michael D. Caggiano. 2018. Analyzing wildfire suppression difficulty in relation to protection demand. In *Environmental Risks*; IntechOpen: London, UK, 2018. doi: 10.5772/intechopen.76937

US Forest Service. 2009. FSPro reference manual. Available online at https://wfdss.usgs.gov/wfdss/pdfs/fspro_reference.pdf; last accessed Sep. 12, 2019.

van Bemmelen, Joost; Wilko Quak; Marcel van Hekken and Peter van Oosterom. 1993. Vector vs Raster based Algorithms for Cross Country Movement Planning. *Proceedings, Auto-Carto* 11, 304-17.

Wei, Yu; Matthew P. Thompson; Jessica R. Haas; Gregory K. Dillon and Christopher D. O'Connor. 2018. Spatial optimization of operationally relevant large fire confine and point protection strategies: Model development and test cases. *Canadian Journal of Forest Research*, 2018, 48: 1–14. doi:10.1139/cjfr-2017-0271

Wei, Yu; Matthew P. Thompson; Joe H. Scott; Christopher D. O'Connor and Christopher J. Dunn. 2019. Designing Operationally Relevant Daily Large Fire Containment Strategies Using Risk Assessment Results. *Forests* 2019, 10, 311. doi:10.3390/f10040311

Xu, Jianping and Richard G. Lathrop Jr. 1994. Improving cost-path tracing in a raster data format. *Computers & Geosciences* Vol. 20, No. 10 (1994): 1455-1465.

Yu, Chaoqing; Jay Lee and Mandy J. Munro-Stasiuk. 2003. Extensions to least-cost path algorithms for roadway planning. *International Journal of Geographical Information Science*, 17 (2003): 361-376.

Appendix A
Programming code examples

Python code of three-dimensional array construction

```
import os
import numpy as np
from numpy import *

file1 = "F:/Research/Thesis/text_files/perimeter_sim/sdi_sa_wo_header.txt"
file2 = "F:/Research/Thesis/text_files/perimeter_sim/3D_array.txt"
file3 = "F:/Research/Thesis/text_files/perimeter_sim/fl_reclass_sa_wo_header.txt"
file4 = "F:/Research/Thesis/text_files/perimeter_sim/fl_mosaic_sa_wo_header.txt"

ncols=1177
# 0-1176
nrows=1473
# 0-1472

# Open the file with read only permit
f1 = open(file1)
f2 = open(file2, 'w')
grid = [line.split() for line in f1]
grid_g0 = np.array(grid)
print (grid_g0.shape)

grid_g1 = grid_g0[:, :, np.newaxis]
print (grid_g1.shape)

grid_g2 = np.zeros([nrows,ncols])
print (grid_g2.shape)

i=0
j=0
for i in range(0, nrows):
    for j in range(0, ncols):
        if float(grid_g0[i, j]) < 0:
            grid_g2[i][j] = 1
        else:
            if i == 0 or j == 0 or i == nrows-1 or j == ncols-1:
                grid_g2[i][j] = 2
            elif float(grid_g0[i - 1, j - 1]) < 0:
                grid_g2[i][j] = 2
            elif float(grid_g0[i - 1, j]) < 0:
                grid_g2[i][j] = 2
            elif float(grid_g0[i-1, j+1]) < 0 and i < nrows-1 and j < ncols-1:
                grid_g2[i][j] = 2
            elif float(grid_g0[i, j - 1]) < 0:
                grid_g2[i][j] = 2
            elif float(grid_g0[i, j+1]) < 0 and i < nrows-1 and j < ncols-1:
                grid_g2[i][j] = 2
            elif float(grid_g0[i+1, j-1]) < 0 and i < nrows-1 and j < ncols-1:
                grid_g2[i][j] = 2
            elif float(grid_g0[i+1, j]) < 0 and i < nrows-1 and j < ncols-1:
                grid_g2[i][j] = 2
            elif float(grid_g0[i+1, j+1]) < 0 and i < nrows-1 and j < ncols-1:
                grid_g2[i][j] = 2
            else:
                grid_g2[i][j] = 3

grid_g3 = np.dstack((grid_g1, grid_g2))
print (grid_g3.shape)

f3 = open(file3)
existingline = [line.split() for line in f3]
grid_g4 = np.array(existingline)
print (grid_g4.shape)

f4 = open(file4)
linesegment = [line.split() for line in f4]
grid_g5 = np.array(linesegment)
print (grid_g5.shape)

grid_g6 = np.dstack((grid_g3, grid_g4, grid_g5))

grid_g7 = np.zeros([nrows,ncols])
grid_g8 = np.zeros([nrows,ncols])
print (grid_g7.shape)
print (grid_g8.shape)

for p in range(0, nrows):
    for q in range(0, ncols):
        if   float(grid_g0[p, q])>= 0 and float(grid_g0[p, q]) < 0.8:
            # grid_g7[p][q] = 0.5
            grid_g7[p][q] = round(0.5 * (1 + float(grid_g0[p, q])),3)
            # grid_g8[p][q] = 0.707
            grid_g8[p][q] = round(0.707 * (1 + float(grid_g0[p, q])),3)
        else:
            grid_g7[p][q] = 4999
            grid_g8[p][q] = 4999

grid_final = np.dstack((grid_g6, grid_g7, grid_g8, grid_g7, grid_g8, grid_g7, grid_g8, grid_g7, grid_g8))
print "[tSDI, location, if is existing feature, feature segment No., N, NE, E, SE, S, SW, W, NW]"
print (grid_final.shape)

# Write the array to disk
# I'm writing a header here just for the sake of readability
# Any line starting with "#" will be ignored by numpy.loadtxt
f2.write('# Array shape: {0}\n'.format(grid_final.shape))

# Iterating through a ndimensional array produces slices along
# the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in grid_final:

    # The formatting string indicates that I'm writing out
    # the values in left-justified columns 7 characters in width
    # with 2 decimal places.
    np.savetxt(f2, data_slice, fmt="%s")

    # Writing out a break to indicate different slices...
    f2.write('# New slice\n')
```

Python code of network construction and processing Dijkstra's algorithm

```
import os
import sys
sys.setrecursionlimit(100000)
# to avoid: RuntimeError: maximum recursion depth exceeded
```

```
import numpy as np
from numpy import *

# Read the array from disk
# Suppress Scientific Notation in Numpy When Creating Array From Nested List
np.set_printoptions(suppress=True)

data_array = np.loadtxt("F:/Research/Thesis/text_files/perimeter_sim/3D_array.txt").reshape((1473,1177,12))
print data_array.shape

shortest_path = "F:/Research/Thesis/text_files/perimeter_sim/shortest_paths.txt"
f1 = open(shortest_path, 'w')

###################################################################################################

ncols=1177
# 0-1176
nrows=1473
# 0-1472

###################################################################################################

# Define Dijkstra method (Gilles Bertrand, 2014)

def dijkstra(graph, src, dest, visited=[], distances={}, predecessors={}):
# ending condition
    if src in dest:
        # We build the shortest path and display it
        path = []
        tempindex = dest.index(src)
        pred = dest[tempindex]
        while pred != None:
            path.append(pred)
            pred = predecessors.get(pred, None)
        print('shortest path: '+ '\n' + str(path) + " cost=" + str(distances[dest[tempindex]]))
        result = str(path)
        # print result
        f1.write('\n' + '\n' + 'shortest path: ')
        f1.write('\n' + str(path) + " cost=" + str(distances[dest[tempindex]]))

    else:
        # if it is the initial    run, initializes the cost
        if not visited:
            distances[src] = 0
        # visit the neighbors
        for neighbor in graph[src]:
            if neighbor not in visited:
                new_distance = distances[src] + graph[src][neighbor]
                if new_distance < distances.get(neighbor, float('inf')):
                    distances[neighbor] = new_distance
                    predecessors[neighbor] = src
        # mark as visited
        visited.append(src)
        # now that all neighbors have been visited: recurse
        # select the NON VISITED node with lowest distance 'x'
        # run Dijskstra with src='x'

        unvisited = {}
        for k in graph:
            if k not in visited:
                unvisited[k] = distances.get(k, float('inf'))
        x = min(unvisited, key=unvisited.get)
        dijkstra(graph, x, dest, visited, distances, predecessors)

################################################################################################

# Find the source node and destination node
# Create a nested dictionary based on array: {'center node':{'neighbor node': 'link cost between these two nodes'}}

if __name__ == "__main__":
    source = []
    p = 259
    q = 549
    # adjustable
    src = str(p) + ' ' + str(q)
    print ("source_node:")
    print src

    f1.write('\n' + '\n' + "source_node:")
    f1.write('\n' + src)

    main_dict = {}
    for i in range(0, nrows - 1):
        for j in range(0, ncols - 1):
            if data_array[i][j][1] > 1 and (i - p + 1.732)*(i - p + 1.732)+(j - q + 1)*(j - q + 1) <= 4900 and j <= 1.732 * i + q
- 1.732 * p + 2 and j >= q - 1:
                # Searching range
                main_dict[str(i) + ' ' + str(j)] = {}
                if j <= 1.732 * i + q - 1.732 * p + 2 and j >= q - 1:
                    main_dict[str(i) + ' ' + str(j)][str(i - 1) + ' ' + str(j)] = round(data_array[i][j][4] + data_array[i -
1][j][8], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i - 1) + ' ' + str(j + 1)] = round(data_array[i][j][5] + data_array[i -
1][j + 1][9], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i) + ' ' + str(j + 1)] = round(data_array[i][j][6] + data_array[i][j +
1][10], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i + 1) + ' ' + str(j + 1)] = round(data_array[i][j][7] + data_array[i +
1][j + 1][11], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i + 1) + ' ' + str(j)] = round(data_array[i][j][8] + data_array[i +
1][j][4], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i + 1) + ' ' + str(j - 1)] = round(data_array[i][j][9] + data_array[i +
1][j - 1][5], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i) + ' ' + str(j - 1)] = round(data_array[i][j][10] + data_array[i][j -
1][6], 3)
                    main_dict[str(i) + ' ' + str(j)][str(i - 1) + ' ' + str(j - 1)] = round(data_array[i][j][11] + data_array[i -
1][j - 1][7], 3)
    graph = main_dict
    print ("dictionary:")
    print graph

    dest = []
    for m in range(0, nrows - 1):
        for n in range(0, ncols - 1):
            if data_array[m][n][1] > 1 and data_array[m][n][2] > 0 and (m - p + 1.732)*(m - p + 1.732)+(n - q + 1)*(n - q
+ 1) <= 4900 and n <= 1.732 * m + q - 1.732 * p + 2 and n >= q - 1 and data_array[m][n][3] != data_array[p][q][3]:
                dest.append(str(m) + ' ' + str(n))
    print ("ending_node list:")
    print dest

    f1.write('\n' + '\n' + "ending_node list:")
    f1.write('\n' + str(dest))

    if len(dest):
        unvisited = {}
        dijkstra(graph, src, dest, [], {},{})
```

Python code of digitizing back to map format

```
import os
import numpy as np
from numpy import *

file1 = "F:/Research/Thesis/text_files/perimeter_sim/path.txt"
file2 = "F:/Research/Thesis/text_files/perimeter_sim/output/path_asc.txt"

f1 = open(file1)
lines = f1.read().splitlines()

# 2 digit by 2 digit (+2/+2, +9)
# print list[1:3]
# print list[4:6]
# print list[10:12]
# print list[13:15]

# 2 digit by 3 digit (+2/+3, +10)
# print list[1:3]
# print list[4:7]
# print list[11:13]
# print list[14:17]

# 3 digit by 2 digit (+3/+2, +10)
# print list[1:4]
# print list[5:7]
# print list[11:14]
# print list[15:17]

# 3 digit by 3 digit (+3/+3, +11)
print list[1:4]
print list[5:8]
print list[12:15]
print list[16:19]

# 3 digit by 4 digit (+3/+4, +12)
# print list[1:4]
# print list[5:9]
# print list[13:16]
# print list[17:21]

# 4 digit by 3 digit (+4/+3, +12)
# print list[1:5]
# print list[6:9]
# print list[13:17]
# print list[18:21]

nrows=1473
ncols=1177
xllcorner = 235789.27084673
yllcorner = 4134305.1496054
cellsize = 30
NODATA_value = -9999

grid = np.zeros([nrows,ncols])

# for i in range(0, 16):
    # 2 digit by 2 digit (+2/+2, +9)
    # m = int(list[1+9*i:3+9*i])
    # n = int(list[4+9*i:6+9*i])

    # 2 digit by 3 digit (+2/+3, +10)
    # m = int(list[1+10*i:3+10*i])
    # n = int(list[4+10*i:7+10*i])

    # grid[m][n] = 1

for j in range(0, 120):
    # 3 digit by 2 digit (+3/+2, +10)
    # m = int(list[1+10*j:4+10*j])
    # n = int(list[5+10*j:7+10*j])

    # 3 digit by 3 digit (+3/+3, +11)
    m = int(list[1+11*j:4+11*j])
    n = int(list[5+11*j:8+11*j])

    # 3 digit by 4 digit (+3/+4, +12)
```

```
        # m = int(list[1 + 12 * j:4 + 12 * j])
        # n = int(list[5 + 12 * j:9 + 12 * j])

        # 4 digit by 3 digit (+4/+3, +12)
        # m = int(list[1 + 12 * j:5 + 12 * j])
        # n = int(list[6 + 12 * j:9 + 12 * j])

        # special case:
        # grid[999][576] = 1
        # m = int(list[12 + 12 * j:16 + 12 * j])
        # n = int(list[17 + 12 * j:20 + 12 * j])
        grid[m][n] = 1

# https://stackoverflow.com/questions/24876331/writing-an-ascii-file-from-a-2d-numpy-array

from StringIO import StringIO

f2 = StringIO()
np.savetxt(f2, grid, fmt='%d')
f2.seek(0)
fs = f2.read().replace('0', '-9999', -1)
f2.close()

f3 = open(file2, 'w')
f3.write("ncols            " + str(ncols) + "\n")
f3.write("nrows            " + str(nrows) + "\n")
f3.write("xllcorner        " + str(xllcorner) + "\n")
f3.write("yllcorner        " + str(yllcorner) + "\n")
f3.write("cellsize         " + str(cellsize) + "\n")
f3.write("NODATA_value     " + str(NODATA_value) + "\n")
f3.write(fs)
f3.close()
```