2014

Connecting Landscapes to People: Assessing the Distribution of Ecosystem Service Flows Using the SPAN Approach

Gary Wayne Johnson

University of Vermont

Follow this and additional works at: https://scholarworks.uvm.edu/graddis

Part of the Computer Sciences Commons, and the Natural Resources Management and Policy Commons

Recommended Citation


https://scholarworks.uvm.edu/graddis/301

This Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks @ UVM. It has been accepted for inclusion in Graduate College Dissertations and Theses by an authorized administrator of ScholarWorks @ UVM. For more information, please contact donna.omalley@uvm.edu.
CONNECTING LANDSCAPES TO PEOPLE: ASSESSING THE DISTRIBUTION OF ECOSYSTEM SERVICE FLOWS USING THE SPAN APPROACH

A Dissertation Presented

by

Gary W. Johnson, Jr.

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
Specializing in Computer Science

October, 2014
Accepted by the Faculty of the Graduate College, The University of Vermont, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

Dissertation Examination Committee:

__________________________  Advisor
Robert R. Snapp, Ph.D.

__________________________
Byung S. Lee, Ph.D.

__________________________
Donna M. Rizzo, Ph.D.

__________________________  Chairperson
Joshua C. Farley, Ph.D.

__________________________  Dean, Graduate College
Cynthia J. Forehand, Ph.D.

Date: August 25, 2014
The Service Path Attribution Network (SPAN) framework provides a novel, user-centric, connectivity-based approach to ecosystem service assessment and valuation (ESAV). Ecosystem services are delivered to users through the simulated flow of some service medium (i.e., matter, energy, or information) from the ecosystems in which it originates (sources) to the people or assets which it affects (users). Along the way, the service medium may be absorbed by intervening landscape features (sinks) or captured by rival users.

Crucially, the service medium is not itself an ecosystem service or benefit but rather an agnostic transport mechanism which establishes connectivity between sources, sinks, rival users, and nonrival users within a delimited study region. Each user then receives benefits or harm from the encountered service medium depending on their specific relationship with it. For example, if surface water is the simulated service medium, it may increase productivity at a hydropower plant but damage farmers in floodplains by drowning their crops.

In the SPAN terminology, sources provide provisioning ecosystem services to users with a beneficial relationship with the service medium. Similarly, sinks provide preventive ecosystem services to users with a detrimental relationship with the service medium by reducing the amount flowing to their locations. Notably, within a single SPAN analysis, both sources and sinks may provide ecosystem services given a sufficiently heterogeneous pool of users.

The results of a SPAN ESAV analysis are myriad, totalling up to 30 output maps for some services. Taken together, these maps tell the story of which sources provide services to which users, which sinks protect users from harm, which users compete for the same resources (and who wins), and how all of the sources, sinks, rival users, and nonrival users affect one another. Additionally, a SPAN simulation produces maps of the flow paths taken by the service medium from sources to users as well as where and by how much the flow strength is reduced by sinks. Studying these flow paths can help decision makers identify those locations at which management actions would be maximized or minimized depending on their specific development goals.

A crowning achievement of this work is that for most ecosystem services the SPAN algorithm’s complexity is guaranteed to be linear $O(n)$ in both time and space with respect to the number of discrete locations analyzed. This makes it a viable option for high resolution landscape level ESAV studies using no more than commodity hardware.

This dissertation explores the SPAN framework in depth, from its novel conceptual terminology and computational algorithms through to the intended interpretation of its results. In addition to describing the conceptual and mathematical components of this system in detail, this work also provides a complete Literate Program demonstrating the application of the SPAN framework to an assessment of the scenic beauty ecosystem service in Chittenden County, Vermont.
Acknowledgements

No dissertation is ever completed in isolation, and for me it was no different. First and foremost, I must thank my advisors, Ferdinando Villa and Robert Snapp, for their enormous contributions to my work. Without Ferdinando’s research guidance, I may have never taken on the problem of computational ecosystem service assessment, and without Robert’s endless patience and brilliant instruction, I would have sorely lacked the necessary knowledge to accomplish my task.

Kenneth Bagstad and Brian Voigt both provided countless man-months of discussion and testing around my SPAN framework, helping me to sand down its rough edges and extend it to fit the many diverse ecosystem service examples they were able to dream up. In our years of working together, their good-natured camaraderie and honest criticisms were doled out in equal measure, and my work was most certainly the better for it.

Ioannis Athanasiadis provided me with a much-needed critical eye when it came to model development and helped me to solve some of the more challenging aspects of the channeled flow problem. Perhaps even more importantly, he reminded me of the importance of letting go of old algorithms – “killing your darlings” so to speak – regardless of how clever I found them. His encouragement in this regard gave me the courage to abandon years of older SPAN models that had become dead ends performance-wise and take on the daunting project of redesigning everything from scratch.

Not to be overlooked, Joshua Farley taught my first class on ecosystem services, introducing me to my initial glimpses of the field’s terminology, conceptual models, and wide ranging environmental applications. Also standing as the chair of my defense committee, his participation in these two roles has served to rather poetically bring my research work full circle.

Although invited somewhat belatedly to the party, Byung Lee and Donna Rizzo were both extremely gracious in agreeing to join my defense committee during the final year of my work. Despite initially being only peripherally familiar with my research, both were quick to pick it up, ask pointed questions, and help me finally take my work the last mile to the finish line.

On a more personal note, I would like to thank my employer, David Saah, for granting me a partially funded leave of absence for the eight months I needed to complete my research and write this dissertation. Although not directly involved in its production, his patience, financial support, and encouraging words have gone a long way towards getting me over this last hurdle.

Finally, I would like to thank my family and friends for their support and encouragement over all the years that I have been toiling away at this project. First and foremost among them, my love and appreciation goes out to my partner, Hanna Howard, for never giving up on me, always being available to listen whenever I needed a receptive ear, taking care of our home when I was so buried in work that I couldn’t see my way out, and knowing just when to remind me to stop and watch the sunset.
DEDICATION

To nature, who awed me with wonders unimaginable, inspired me to always dig deeper for answers and for strength, and provided comfort and solace when the road seemed hard and unending.

To technology, for keeping my curiosity ablaze since early childhood, for teaching me that few things are more powerful than a well-placed abstraction, and for showing me that solving even the hardest problems is often just about finding the right incantation.

To the very strange and yet completely familiar path of my life, which led me to that place at which these two blend, amplify, and illuminate one another, giving me both a purpose for my work and the tools to pursue it.
# Table of Contents

Acknowledgements ........................................................................ ii
Dedication .................................................................................. iii
List of Tables ................................................................................ vi
List of Figures ................................................................................ viii

1 Introduction ................................................................................. 1
  1.1 Services Delivered Not Services Produced ................................. 3
  1.2 Common Ecosystem Service Metrics ....................................... 4
  1.3 Connecting Specific Producers and Users ................................. 5
  1.4 Strategic Planning around Flow Paths ..................................... 6

2 The SPAN Methodology .............................................................. 8
  2.1 Flow Path Topologies ........................................................... 12
    2.1.1 In situ Flow ................................................................. 12
    2.1.2 Proximal Flow ............................................................ 13
    2.1.3 Channeled Flow ......................................................... 15
    2.1.4 Diffusive Flow ............................................................ 16
    2.1.5 Global Flow ............................................................... 17
  2.2 Mapping the SPAN Components ............................................ 17
  2.3 Flow Surfaces and Routing Layers ......................................... 20
    2.3.1 Proximal Flow Surface ................................................ 21
    2.3.2 Channeled Flow Surface ............................................. 22
    2.3.3 Diffusive Flow Surface ............................................... 24
  2.4 The Flow Distribution Algorithm ......................................... 26
    2.4.1 Ordering the Serviceshed Cells by Outlet Distance ............. 28
    2.4.2 Propagating the Service Medium Downstream .................. 30
    2.4.3 Backpropagating Source and Sink Impacts Upstream .......... 36
  2.5 Understanding the SPAN Outputs .......................................... 41

3 Scenic Beauty in Chittenden County, VT ..................................... 45
  3.1 Scenic Beauty As an Ecosystem Service .................................. 47
  3.2 Setting Up the PostGIS Database ......................................... 48
  3.3 Identifying the Study Area ..................................................... 49
  3.4 Source Model ........................................................................ 51
    3.4.1 Slope Data .................................................................. 52
    3.4.2 Land Cover Surprisal Data ......................................... 54
    3.4.3 Scenic Beauty Source Data ......................................... 62
  3.5 Sink Model ........................................................................... 64
    3.5.1 Scenic Beauty Sink Data .............................................. 66
  3.6 Use Model .............................................................................. 67
    3.6.1 Scenic Beauty Use Data ............................................... 68
3.7 Flow Model ................................. 71
3.7.1 Setting Up the Clojure Environment .......... 71
3.7.2 Transferring the PostGIS Layers to Clojure .......... 72
3.7.3 Computing the Scenic Beauty Flow Surface ........... 75
3.7.4 Accounting for Distance Decay and Visibility .......... 93
3.7.5 Performing the Serviceshed Ordering Phase .......... 103
3.7.6 Performing the Forward Propagation Phase .......... 105
3.7.7 Performing the Backpropagation Phase .......... 110
3.7.8 Cleaning Up the Output Maps .................. 114
3.7.9 Reviewing Our Single Serviceshed Results .......... 115
3.7.10 Merging the Servicesheds of All Users .......... 120

4 Discussion and Conclusions 134
4.1 Performance Considerations .................. 134
4.2 Valuing the Service Medium .................. 138
4.3 Land Management Scenarios .................. 141
4.4 Flow Path Analysis .......................... 143
4.5 Final Thoughts .............................. 146
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Basic SPAN terminology</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Some ecosystem services with in situ flow</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Some ecosystem services with proximal flow</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Some ecosystem services with channeled flow</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Some ecosystem services with diffusive flow</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>Some ecosystem services with global flow</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>Cell values for each of the SPAN input maps</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Results of the SPAN forward propagation phase</td>
<td>36</td>
</tr>
<tr>
<td>2.9</td>
<td>Results of the SPAN backpropagation phase</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>Software necessary to evaluate the code in this chapter</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic metadata for the VT county boundaries dataset</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Basic metadata for the USGS NED DEM24 dataset</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Basic metadata for the USGS NLCD dataset</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>NLCD land cover classes present in Chittenden county</td>
<td>59</td>
</tr>
<tr>
<td>3.6</td>
<td>Self information per land cover class</td>
<td>60</td>
</tr>
<tr>
<td>3.7</td>
<td>NLCD land cover codes used as scenic beauty sinks</td>
<td>64</td>
</tr>
<tr>
<td>3.8</td>
<td>Basic metadata for the VT outdoor recreation sites dataset</td>
<td>68</td>
</tr>
<tr>
<td>3.9</td>
<td>Matrix dimensions of raster layers imported from Postgresql to Clojure</td>
<td>74</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Spatial relationships in the SPAN abstraction</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>In situ flow examples</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Proximal flow example</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Channeled flow example</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Diffusive flow example</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>Global flow example</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>Some possible value types for the SPAN input maps</td>
<td>20</td>
</tr>
<tr>
<td>2.8</td>
<td>Unidirectional and multidirectional flow types</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>Nested servicesheds and their outlet points</td>
<td>28</td>
</tr>
<tr>
<td>2.10</td>
<td>Nested servicesheds with a global ordering</td>
<td>29</td>
</tr>
<tr>
<td>2.11</td>
<td>Service medium path through a cell during forward propagation</td>
<td>33</td>
</tr>
<tr>
<td>2.12</td>
<td>Relative flow contributions from local and upstream cells</td>
<td>37</td>
</tr>
<tr>
<td>2.13</td>
<td>Relative flow reductions by local and upstream cells</td>
<td>40</td>
</tr>
<tr>
<td>3.1</td>
<td>VT county boundaries</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Chittenden county elevation in meters at 30 meter resolution</td>
<td>54</td>
</tr>
<tr>
<td>3.3</td>
<td>Chittenden county slope in degrees at 30 meter resolution</td>
<td>55</td>
</tr>
<tr>
<td>3.4</td>
<td>Chittenden county land cover at 30 meter resolution</td>
<td>58</td>
</tr>
<tr>
<td>3.5</td>
<td>Self information per undeveloped land cover class (NLCD 2001)</td>
<td>60</td>
</tr>
<tr>
<td>3.6</td>
<td>Chittenden county land cover surprisal in nats at 30 meter resolution</td>
<td>62</td>
</tr>
<tr>
<td>3.7</td>
<td>Scenic beauty source histogram</td>
<td>63</td>
</tr>
<tr>
<td>3.8</td>
<td>Scenic beauty source map for Chittenden county at 30 meter resolution</td>
<td>65</td>
</tr>
<tr>
<td>3.9</td>
<td>Scenic beauty sink histogram</td>
<td>66</td>
</tr>
<tr>
<td>3.10</td>
<td>Scenic beauty sink map for Chittenden county at 30 meter resolution</td>
<td>68</td>
</tr>
<tr>
<td>3.11</td>
<td>Scenic beauty use map for Chittenden county at 30 meter resolution</td>
<td>70</td>
</tr>
<tr>
<td>3.12</td>
<td>Scenic beauty source map after downsampling and thresholding</td>
<td>76</td>
</tr>
<tr>
<td>3.13</td>
<td>Scenic beauty sink map after downsampling and thresholding</td>
<td>77</td>
</tr>
<tr>
<td>3.14</td>
<td>Scenic beauty nonrival use map after downsampling and thresholding</td>
<td>78</td>
</tr>
<tr>
<td>3.15</td>
<td>Scenic beauty elevation map after downsampling and thresholding</td>
<td>79</td>
</tr>
<tr>
<td>3.16</td>
<td>Distance (in meters) to potentially visible points within 25km of a user</td>
<td>80</td>
</tr>
<tr>
<td>3.17</td>
<td>Inward trajectory from each point within 25km of a user</td>
<td>82</td>
</tr>
<tr>
<td>3.18</td>
<td>Trajectories snapped to their nearest compass direction</td>
<td>85</td>
</tr>
<tr>
<td>3.19</td>
<td>Octagonal partitioning of the serviceshed ordered by steps from the user</td>
<td>89</td>
</tr>
<tr>
<td>3.20</td>
<td>Proximal flow surface for a user with a 25km view radius</td>
<td>92</td>
</tr>
<tr>
<td>3.21</td>
<td>Sight lines using independent trajectories vs the merged flow surface</td>
<td>93</td>
</tr>
<tr>
<td>3.22</td>
<td>Derivation of the scenic beauty distance decay function</td>
<td>95</td>
</tr>
<tr>
<td>3.23</td>
<td>Scenic beauty distance decay function</td>
<td>95</td>
</tr>
<tr>
<td>3.24</td>
<td>Viewshed distance decay map for a user with a 25km view radius</td>
<td>97</td>
</tr>
<tr>
<td>3.25</td>
<td>Sight line obstructions by intervening topography</td>
<td>98</td>
</tr>
<tr>
<td>3.26</td>
<td>Viewshed visibility map for a user with a 25km view radius</td>
<td>101</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Human beings, no matter where they live or what they understand about the world, are all dependent on the natural systems and processes around them. There is no doubt that human ingenuity and labor, both within and beyond the boundaries of economic markets, can contribute significantly to the pursuit of well-being, happiness, and quality of life. Unfortunately, goods and services produced by natural systems with little or no intervention by people are not necessarily valued as consistently as their anthropogenic counterparts, leading to a market system biased toward undervaluing these “ecosystem services”. The field of ecosystem service assessment and valuation (ESAV) aims to counter this bias by making clear, economic arguments that demonstrate the many ways in which these services contribute to our health, wealth, and long-term prosperity. (Daily, 1997, Millennium Ecosystem Assessment (MA), 2005, National Research Council (NRC), 2005)

However, despite the wealth of publications and professional conferences now centered around this subject, its adherents are largely drawn from non-technical fields, such as ecology, conservation biology, natural resource management, and ecological economics. As a result, the field’s terminology remains both predominantly qualitative and frequently disputed, and its computational science branch is far from mature.

In the past decade, software supporting ESAV has begun to appear from the academic,
governmental, and business sectors. (Villa et al., 2014, Daily et al., 2009, Coffin et al., 2012, Troy and Wilson, 2006) This is quite encouraging, but due to the field’s lack of a robust mathematical foundation, no two programs seem to generate the same metrics. Somewhat unsurprisingly, this has led to a situation in which those who wish to use ESAV software to support land planning decisions are often unclear as to which tools are most applicable to their questions. As a result, research articles claiming to compare different tools side by side remain in high demand among both developers and users. (Bagstad et al., 2013c,b)

The Service Path Attribution Network (SPAN) framework presented in this dissertation aims to provide a solid mathematical foundation upon which to base quantitative ecosystem service assessments. Importantly, the SPAN approach is largely agnostic with respect to the final valuation method chosen and should therefore be seen by those in the valuation field as a tool for augmenting their existing techniques rather than replacing them outright.

For developers of ESAV software, the SPAN algorithms presented here may be (and hopefully will be) freely incorporated into existing works. This is made possible because SPAN is not merely a program that one uses for ESAV analyses (although it does have a working software implementation) but rather a mathematical framework for describing spatial connectivity relationships between ecosystems which provide services and human users which benefit from them.

In crafting the SPAN algorithms, a great deal of care was put into making them both accurate and efficient. As a result, for most ecosystem services a full ESAV analysis is guaranteed to be linear $O(n)$ in both time and space with respect to the number of discrete locations studied. This makes SPAN capable of high resolution landscape level modeling using no more than commodity hardware.

In the remainder of this chapter, we outline four key ways in which the SPAN approach aims to enhance and extend the computational techniques currently in use in the field of
ESAV analysis. Later, in Chapter 2, we will introduce the Service Path Attribution Network (SPAN) framework in detail. Following this, Chapter 3 provides a complete Literate Program demonstrating an ecosystem service assessment for the scenic beauty service in Chittenden County, Vermont. Finally, Chapter 4 will review the main contributions of this research and expand upon their potential for informing sustainable land management decisions.

1.1 Services Delivered Not Services Produced

A common approach to ESAV, known as “benefit transfer”, translates some environmental, infrastructural, or demographic features of a region (e.g., land use type, population density, or affluence) into monetary value ranges per area per time unit (e.g., dollars per hectare per year). The overall ecosystem service value of a region assessed in this manner is the sum of the value contributed by each distinct feature set weighted by its area. The monetary value ranges used are generally calculated from a meta-analysis of previously published ESAV studies which share certain similarities with the ESAV site being analyzed. One ESAV tool that uses this approach is the Natural Assets Information System (NAIS) developed by Spatial Informatics Group, LLC.(Costanza et al., 1997, 2006, Troy and Wilson, 2006, Wilson and Hoehn, 2006)

Another spatially static, yet non-monetary, approach involves applying an “ecological production function” to each areal unit within the study region (e.g., cells on a raster grid) for each ecosystem service of interest. Such a function typically estimates the biophysical values that each cell would produce in a given time period (e.g., surface water runoff in mm$^3$/yr, CO$_2$ sequestered in tons/yr, crop yield in tons/yr). This approach to ESAV is used in software tools like the Integrated Valuation of Environmental Services and Tradeoffs (InVEST) program created by the Natural Capital Project.(Kaiser and Roumasset, 2002, Ricketts et al., 2004)
Because both the “benefit transfer” and “production function” approaches rely on *in situ* calculations, they estimate ecosystem service values in terms of the amount of goods and services that may potentially be produced by a location rather than by the fraction which is ultimately delivered to human beneficiaries. (Nelson et al., 2009) We improve upon these spatially static methods by designing a framework for modeling ecosystem services in terms of flows of some form of matter, energy, or information, called the “service medium”, which is produced by an ecosystem and represents value or damage when encountered by humans or their assets.

When the service medium represents a beneficial flow of goods or services (e.g., fresh water or scenic beauty), ecosystem service values are assigned to locations based on the amount of the medium they produce which is captured by beneficiaries in the study region. When the service medium represents a detrimental flow of matter or energy (e.g., floodwater or wildfire), ecosystem service values are assigned to locations which absorb or capture the medium, thereby reducing or preventing damage done to vulnerable individuals or assets.

The details of this novel ESAV framework, called the Service Path Attribution Network (SPAN), forms the foundation of all the research work presented in this dissertation. (Johnson et al., 2010, 2012)

1.2 **Common Ecosystem Service Metrics**

Many ESAV exercises involve estimating values for more than one service within the same region. The total (lower-bound) value for that area is then calculated by combining the individual service values into a “service bundle” or “service portfolio”. (Raudsepp-Hearne et al., 2010) The specific methods used in this multi-service aggregation stage present their own challenges, particularly when services are dependent on one another. However, regardless of the approach taken, some common set of metrics must be used for all services combined in this way.
When each one is modeled using different methods and assumptions, both aggregation and comparison across services (e.g., to assess changes in their distribution under different scenarios) can become challenging if not impossible. One way in which this apparent incommensurability may be resolved is to convert all values to monetary units. However, the approaches used in estimating these monetary values may introduce their own uncertainty into the modeling process. (Plummer, 2009)

As an alternative solution, we again propose the use of the SPAN ESAV framework. One of its core design principles has been sufficient generalizability to express many different ecosystem services and produce common output metrics for each. This enables direct comparison between or aggregation across multiple services without needing to resort to monetary valuation techniques. However, should such techniques be deemed appropriate for a particular application, SPAN’s outputs may be easily converted to monetary values on a service specific basis.

Since 2008, SPAN implementations have been created for eight distinct ecosystem services, and the conceptual elements necessary to implement an additional 10 services have already been published. This work was developed as part of the NSF-funded Artificial Intelligence for Ecosystem Services (ARIES) project. (Villa et al., 2009, 2011, Bagstad et al., 2011, 2013a, Villa et al., 2014)

1.3 Connecting Specific Producers and Users

In order to determine the amount of impact that each service-producing ecosystem has on each human beneficiary, a SPAN model simulates the flow of the service medium across the study region. The flow paths that it traces out form a directed acyclic graph, which may be partitioned into subgraphs specific to particular service producers or users.

Thus, one may perform a SPAN assessment for a large watershed and see the distribution of ecosystem services to all the users within that watershed just as easily as one may ask
SPAN to provide the distribution of ecosystem services to a single user or any collection of users. Similarly, without rerunning the flow simulation, the SPAN graph may be traversed upstream from any user to those ecosystems which provide its services or downstream from any ecosystem to the users which it benefits.

This enables both rapid, high-level interrogation of the result space (e.g., “Where are this user’s ecosystem services produced?” or “Which users receive the ecosystem services produced by this land?”) as well as more detailed numerical analyses (e.g., “How much service does each location provide to this user?”).

Combining these graph walking techniques with spatial queries over non-SPAN-related datasets can enable quite sophisticated inquiries, such as: “Which upland forests provide at least 5% of the total ecosystem service value received by farmers in floodplains?” By making this level of detail available to land managers, we hope to significantly expand both the appeal and the usefulness of ESAV techniques in informing decision making processes.

1.4 Strategic Planning around Flow Paths

At the heart of every SPAN analysis is the notion of a flow path or corridor. Unlike in situ approaches that consider only one location at a time or even spatially dynamic approaches that propagate some quantity (e.g., water) over a landscape in timesteps to their final destination, the SPAN framework models ecosystem service delivery as the creation of flow paths through space from points of production to points of impact and assesses both their possible and actual throughput at each location on the landscape.

This connectivity-based paradigm makes it possible to answer such questions as:

1. By what routes are ecosystem services conveyed from the landscapes generating them to the people benefitting from them?

2. Are there places where many such routes converge? How do management impacts in
these locations differ from those in areas with less concentrated flows?

3. Are there beneficiaries with only one or a few routes to them? How does this flow path scarcity affect the value of ecosystem services delivered along these routes? In which situations should flow corridors be considered substitutable or non-substitutable?

4. Where would landscape management actions have the greatest impact (for better or worse) on ecosystem service delivery? Is it possible to increase or decrease flows of ecosystem services to people without directly affecting the places where they are produced?

Whereas every one of these questions presents a potential stumbling block to existing ESAV approaches, the SPAN framework has been designed from the ground up to address precisely these kinds of connectivity-related issues. In much the same way that the field of wildlife conservation has shifted its conversation to one of migration corridors and habitat fragmentation in recent decades, we contend that ecosystem services must also be seen through this lens in order to avoid the unintended consequences of flow path fragmentation.

In the end, if the service medium produced by an ecosystem has no path to people or their assets, then no ecosystem services can be realized.
 Chapter 2

The SPAN Methodology

Connectivity.

Without it, conversations fail, actions have no impacts, and goods and services cannot be delivered. Instead, we have only isolated, independent entities unrelated to and unaware of one another. Add connectivity back in, and our static world suddenly leaps to life, filled with endless opportunities for complex and often unexpected interactions.

This is as much the case for ecosystem services as for any other complex system that one may wish to study. However, many existing approaches to ecosystem service assessment and valuation focus their energies on in situ estimates of landscape productivity rather than on the delivery of these services to their final users.(Costanza et al., 1997, 2006, Troy and Wilson, 2006, Wilson and Hoehn, 2006, Kaiser and Roumasset, 2002, Ricketts et al., 2004, Nelson et al., 2009) This may seem perfectly reasonable for services which are used in the same place that they are produced. Unfortunately, this approach is fundamentally near-sighted since for a great many ecosystem services, the human recipients may be located far away from these source points.

The Millennium Ecosystem Assessment report has defined ecosystem services most simply as “benefits people obtain from ecosystems.”(Millennium Ecosystem Assessment (MA), 2005) Still, this definition lacks a mechanism of action, so we must dive a bit deeper to see if
such an agent can be found. Consider, for example, a patch of chanterelles growing in a New England forest. The fungus of which they are a part performs several important ecological functions in its immediate environment, which are of interest to biologists, ecologists, and of course, the nearby trees with which they exchange nutrients. However, ecological functions do not by themselves give rise to ecosystem services. To make this next step, a relationship must be established with a human beneficiary.

Thus, when a mushroom hunter comes ambling through the forest and spots the chanterelles, the ecosystem service of “food provision” suddenly materializes. Alternatively, had the visitor been a mycologist searching for spore print samples, she would have received the very different ecosystem service of “genetic resources”. Going still further, perhaps a nature photographer might have decided that a snapshot of these mushrooms would be a worthy addition to his collection. Here the informational ecosystem service of “scenic beauty” is realized.

What these three quite different instances should illustrate is that chanterelles are not an ecosystem service nor do they produce ecosystem services by themselves. Instead, different services develop based upon each person’s unique relationship with them. The mushrooms then are what we will call, in the SPAN terminology, the service medium.

For each ecosystem service, its associated service medium will be some form of matter, energy, or information, which is produced by an ecosystem and often travels through space and time to people, whom it ultimately helps or harms. Every ecosystem service must have at least one service medium, although many different services may share the same one in common. We call ecosystems which produce the service medium sources and humans or their assets which it may affect users. The route connecting any source to any user is known as a flow path.

For some services, ecosystems which can absorb the service medium, called sinks, may lie along a flow path between sources and users, thereby reducing the amount flowing from
one point to the other. Finally, when users interact with the service medium, they may do so either destructively or nondestructively. We call the former group *rival users* and the latter *nonrival users*.

Figure 2.1 shows the spatial relationships between each of these landscape elements. Note that sinks and rival users reduce the service medium along flow paths passing through them while nonrival users do not.

![Figure 2.1: Spatial relationships in the SPAN abstraction](image)

As the number of sources, sinks, rival users, and nonrival users increases in any region, their flow paths will come to form a rich network of branching and merging routes along which the service medium’s flow determines the ecosystem service value of that area.

We call such a model of ecosystem services a Service Path Attribution Network (SPAN) because we attribute the final ecosystem service values to those path elements whose positive or negative impacts on the flow strength generate benefits for people. When the service medium represents a beneficial flow of goods or services (e.g., fresh water or scenic beauty), ecosystem service values are attributed to sources based on the amount of the medium they produce which is captured by users in the study region. When the service medium
represents a detrimental flow of matter or energy (e.g., floodwater or wildfire), ecosystem service values are attributed to sinks which absorb or capture the medium, thereby reducing or preventing damage done to vulnerable individuals or assets.

In the SPAN terminology, sources provide *provisioning ecosystem services* to users with a beneficial relationship with the service medium. Similarly, sinks provide *preventive ecosystem services* to users with a detrimental relationship with the service medium by reducing the amount flowing to their locations. Notably, within a single SPAN analysis, both sources and sinks may provide ecosystem services given a sufficiently heterogeneous pool of users. For example, if surface water is the simulated service medium, it may increase productivity at a hydropower plant but damage farmers in floodplains by drowning their crops. Here the power plant would value those upstream sources which generate the most runoff, but the farmers would value sinks, such as wetlands, that slow or halt the flow of floodwater to them.

We summarize the terminology introduced in this section in Table 2.1. In the next section, we will explore the topologic characteristics of SPAN flow paths in greater depth.

Table 2.1: Basic SPAN terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Medium</td>
<td>Matter, energy, or information which flows from sources to users, establishing connectivity and the possibility of ecosystem service delivery.</td>
</tr>
<tr>
<td>Source</td>
<td>A location which produces the service medium.</td>
</tr>
<tr>
<td>Sink</td>
<td>A location which absorbs the service medium.</td>
</tr>
<tr>
<td>Rival User</td>
<td>A user who benefits by removing the service medium from its flow path.</td>
</tr>
<tr>
<td>Nonrival User</td>
<td>A user who is affected by simply encountering the service medium.</td>
</tr>
<tr>
<td>Flow Path</td>
<td>A route between source and user along which the service medium flows.</td>
</tr>
<tr>
<td>Provisioning Service</td>
<td>An ecosystem service in which sources provide beneficial flows to users.</td>
</tr>
<tr>
<td>Preventive Service</td>
<td>An ecosystem service in which sinks reduce detrimental flows to users.</td>
</tr>
</tbody>
</table>
2.1 Flow Path Topologies

Now that we have established the idea that ecosystem services are delivered from sources (or sinks) to users by means of flow paths, the next logical question to ask is how exactly these flow paths are to be defined. While the number of distinct ecosystem services is potentially quite large, fortunately their means of transmission are relatively few. Based on the spatio-temporal characteristics of their service media, we group ecosystem services broadly into the following five general flow categories.

2.1.1 In situ Flow

Services of this type exhibit the shortest flow paths. In order for an ecosystem service to be realized, a user (whether rival or nonrival) must visit or inhabit the source location, and only sinks co-located with sources may impact service delivery. Figure 2.2 shows the four configurations of sources (S), sinks (K), rival users (R), and nonrival users (N) in which *in situ* flows can occur.

![Figure 2.2: In situ flow examples](image)

All of the chanterelle-derived ecosystem services discussed in the previous section belong to this category as do most which are based on the extraction of natural resources, including agriculture. One particularly interesting service to consider is recreation. This ecosystem service is really just the delivery of a particular enjoyable experience to the user, leaving them (unexpected injuries notwithstanding) with only memories and possibly some health
benefits afterwards, depending on the activities involved. If the direct experience of any sensory perception related to a recreational activity is limited to a very near distance, then we may effectively consider this an in situ service. Table 2.2 lists some ecosystem services with this flow type.

Table 2.2: Some ecosystem services with in situ flow

<table>
<thead>
<tr>
<th>Provisioning</th>
<th>Preventive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Food</td>
<td>Pest control</td>
</tr>
<tr>
<td>Biomass fuels</td>
<td></td>
</tr>
<tr>
<td>Building materials</td>
<td></td>
</tr>
<tr>
<td>Minerals</td>
<td></td>
</tr>
<tr>
<td>Genetic resources</td>
<td></td>
</tr>
<tr>
<td>Medicinal resources</td>
<td></td>
</tr>
<tr>
<td>Ornamental resources</td>
<td></td>
</tr>
<tr>
<td>Recreational enjoyment</td>
<td></td>
</tr>
</tbody>
</table>

2.1.2 PROXIMAL FLOW

This flow type is characterized by independent, nonrival use of a resource, whose accessibility to or impact on the user attenuates with distance. As a result, no shared flow surface exists for any two non-spatially-colocated users. Instead each user imposes their own flow surface on their immediately surrounding environment, within which they are the only user and all flow paths point directly towards them like the spokes of a wheel. Source and sink values are weighted by a distance decay function such that those located further from the user will have less of an impact on service medium production or absorption than nearer ones.

Figure 2.3 illustrates an example of this flow path topology. In the center of the circle is a single nonrival user (N) surrounded by several sources (S) and sinks (K). All sources project flow paths to the user, but the only sinks which affect ecosystem service delivery are those which fall along one of these paths.

For proximal flow services, the process of creating an inward-facing flow surface must be repeated for each use location in the study area. A separate SPAN simulation is then
run over each such flow surface, and all of the results are finally combined together into a single set of output maps via summation. Note that the result maps generated in this way show the total service medium amount produced (by sources), absorbed (by sinks), or received (by users). These should not be confused with maps of value, in which case each user would first weight the service medium received by some value function prior to summing these results across all the users. This approach is discussed in greater detail in Section 4.2. Alternatively, one could combine all of the output results by averaging rather than summing. The only requirement for the combination function used is that it maintains the mathematical relationships between the outputs layers described in Section 2.5.

The proximal flow type is most appropriate for sensory-based services, such as scenic beauty within a viewshed (i.e., range of sight) or noise mitigation within a soundscape (i.e., range of hearing). Table 2.3 lists some services belonging to this category.

*Table 2.3: Some ecosystem services with proximal flow*

<table>
<thead>
<tr>
<th>Provisioning</th>
<th>Preventive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenic beauty</td>
<td>Reduced light pollution</td>
</tr>
<tr>
<td>Natural soundscapes</td>
<td>Noise mitigation</td>
</tr>
<tr>
<td>Olfactory enjoyment</td>
<td>Odor management</td>
</tr>
</tbody>
</table>

*Figure 2.3: Proximal flow example*
2.1.3 **Channeled Flow**

In this category, a network of distance-independent flow paths channel the service medium from sources through sinks to rival and/or nonrival users. Flow surfaces of this type are generally derived from the study region’s underlying topography.

Figure 2.4 shows an example of this flow type. In this diagram, sources (S) with no downstream users, (non)rival users (N/R) with no upstream sources, and sinks (K) lacking either upstream sources or downstream users are excluded from the flow network.

![Figure 2.4: Channeled flow example](image)

Services that fall under this heading include terrestrial hydrologic services, such as surface water supply (both quantity and quality) or sediment transport (for erosion and deposition-related benefits), as well as those involving animal migration corridors related to hunting, fishing, or ecotourism. Table 2.4 lists some services exhibiting channeled flow.

<table>
<thead>
<tr>
<th>Provisioning</th>
<th>Preventive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water supply</td>
<td>Water purification</td>
</tr>
<tr>
<td>Sediment deposition</td>
<td>Avoided erosion</td>
</tr>
<tr>
<td>Hydropower</td>
<td>Flood mitigation</td>
</tr>
<tr>
<td>Hunting migratory species</td>
<td>Landslide interception</td>
</tr>
<tr>
<td>Fishing migratory species</td>
<td>Avalanche prevention</td>
</tr>
<tr>
<td>Ecotourism of migratory species</td>
<td>Disease regulation (by water)</td>
</tr>
</tbody>
</table>
2.1.4 Diffusive Flow

This grouping represents a hybrid of the channeled and proximal flow types. Here, service media radiate outward from sources, weakening with distance as in the proximal flow case. However, two distinct differences separate this category from its sibling. First, only one flow surface exists across all users, and as a result, intersecting flow paths will interact with one another. Second, non-decaying or even accelerating corridors may exist in the flow path network, which may serve to propagate the service medium beyond its distance decay limited range of effect. Figure 2.5 shows an example of this flow type.

Figure 2.5: Diffusive flow example

These features make the diffusive flow type particularly well suited for describing ecosystem services transmitted by flows of energy, such as wildfire, lightning strikes, and wave energy. Additionally, it may be applied in cases of stochastic outward movement, such as for crop pollination by animals or wind. Table 2.5 lists some services belonging to this category.

Table 2.5: Some ecosystem services with diffusive flow

<table>
<thead>
<tr>
<th>Provisioning</th>
<th>Preventive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tidal energy</td>
<td>Storm surge protection</td>
</tr>
<tr>
<td>Wind pollination</td>
<td>Wildfire control</td>
</tr>
<tr>
<td>Animal pollination</td>
<td>Lightning strike interception</td>
</tr>
<tr>
<td></td>
<td>Disease regulation (by animals)</td>
</tr>
</tbody>
</table>
2.1.5 Global Flow

On the opposite end of the spectrum from in situ flow are those ecosystem services with a global flow distribution. In this category, all sources are absorbed by all sinks and the remaining service medium is transmitted (possibly unevenly) to all users. Figure 2.6 illustrates this simplest of ecosystem service flow types.

![Figure 2.6: Global flow example](image)

This best describes the dynamics of atmospheric effects, such as global warming or ozone depletion, as well as so-called “existence values”, such as the peace of mind some people experience by knowing that polar bears are not extinct despite the fact that they will never see or interact with one in person. Table 2.6 lists some services with global flow.

<table>
<thead>
<tr>
<th>Provisioning</th>
<th>Preventive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existence values</td>
<td>Carbon sequestration</td>
</tr>
<tr>
<td></td>
<td>Climate regulation</td>
</tr>
<tr>
<td></td>
<td>Avoided sea level rise</td>
</tr>
<tr>
<td></td>
<td>Reduced storm intensity and frequency</td>
</tr>
<tr>
<td></td>
<td>Ultraviolet light protection</td>
</tr>
</tbody>
</table>

2.2 Mapping the SPAN Components

To operationalize the ideas presented so far in this chapter, we must next put them in a form that may be analyzed by computer. As the intended application of our SPAN ESAV modeling is to better inform sustainable land management decisions, all of our calculations
will be designed to both consume and produce raster maps. This allows us to smoothly integrate with the GIS-based workflows already in use by land planners and easily make our results available for further spatial analysis by trained GIS technicians.

The key choice when working with geospatial datasets is whether to adopt a raster or vector model of space. Vectors represent spatial information as a collection of geometric objects – generally points, linestrings, and polygons – with any number of attributes assigned to each. Rasters, on the other hand, store information as a matrix of grid cells, each containing a single value sampled at that point in space.

Although many of the flow path topologies described in Section 2.1 seem to have their most natural expression in a vector model, we choose the raster model for our SPAN analyses because it provides us an efficient representation of what are frequently highly spatially heterogeneous datasets. The regularity of the underlying grid structure also provides us with a lattice upon which to define our calculations in terms of network flow propagation. Finally, much of the environmental data used as inputs for SPAN models is most easily acquired in raster format. When vector datasets are needed (as is often the case for anthropocentric inputs), well-tested rasterization procedures are available in most common GIS packages.

A limiting factor associated with working within a raster model of space is that the choice of resolution can significantly affect the model’s cost, performance, and results. Acquiring and warehousing high resolution raster datasets typically requires greater financial and computational resources than for equivalent low resolution data. These costs may be mitigated somewhat by the increasing prevalence of servers supporting the Open Geospatial Consortium (OGC) standard web mapping services (e.g., WCS, WFS, WMS). (Alameh, 2003) However, completely avoiding any local data warehousing still appears to be a long way off.

From a performance perspective, the SPAN algorithms have all been designed to guarantee linear $O(n)$ time and space complexity with respect to the number of grid cells in the
input raster maps. So although the computational requirements for a SPAN analysis will increase at higher resolutions, their relatively slow rate of increase should enable landscape level modeling even on commodity hardware.

The most important factor after cost in determining the appropriate resolution for a SPAN model should then be the amount of flow path misdirection, or “aliasing”, experienced as the resolution is reduced. As the resolution becomes coarser, services belonging to the channeled flow type are particularly prone to having their flow paths misdirected (e.g., water flowing uphill or overland rather than in a river channel). If this aliasing becomes sufficiently pronounced, the SPAN results may need to be discarded. Determining the minimum and maximum resolution within which the underlying flow path shape (from vector space) may be successfully anti-aliased (i.e., reconstructed from the raster samples) for each ecosystem service is related to the Nyquist-Shannon sampling theorem and still remains an important open problem in this area. (Shannon, 1949)

Ultimately, after identifying the service medium associated with the ecosystem service of interest and selecting a suitable grid resolution, the next step in developing a SPAN model is to produce raster maps of its sources, sinks, rival users, and nonrival users in the chosen study area. Although no specific functions are prescribed for calculating each of these map layers, their cells must contain the values listed in Table 2.7.

<table>
<thead>
<tr>
<th>Map Layer</th>
<th>Cell Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>production per cell in units of the service medium</td>
</tr>
<tr>
<td>Sink</td>
<td>absorption capacity per cell in units of the service medium</td>
</tr>
<tr>
<td>Rival Use</td>
<td>demand per cell in units of the service medium</td>
</tr>
<tr>
<td>Nonrival Use</td>
<td>1 if nonrival users are present in a cell; 0 otherwise</td>
</tr>
</tbody>
</table>

If desired, these map values may be probability distributions (Bayesian or frequentist) of the respective service medium production, absorption, or demand per cell rather than deterministic values. Alternatively, one could use interval ranges per cell if information about the probability distributions is found lacking. These uncertainties may then be prop-
agated through the SPAN flow distribution algorithm using either Monte Carlo simulation,
variance propagation, or interval arithmetic as appropriate. (Metropolis, 1987, Goodman,
1960, Sunaga, 2009) These alternative approaches are easily implemented within the SPAN
framework because its calculations are defined using only the arithmetic (+, −, *, /) and
relational (=, <, >, ≤, ≥, min, max) operators. Therefore, any value space for which these
functions are well defined may be used at the modeler’s discretion. Figure 2.7 illustrates
some of the possible value types allowed as SPAN inputs.

![Figure 2.7: Some possible value types for the SPAN input maps](image)

Finally, as an optional filtering mechanism, one may define thresholds for each of the
source, sink, and rival use layers (θ_S, θ_K, θ_R) below which any cell values in their respective
maps are set to zero. By increasing or decreasing these thresholds, one can attempt to focus
their SPAN analyses on only the most significant landscape elements which are likely to
affect ecosystem service flows. Since any zero-valued cells will be ignored by the SPAN flow
distribution algorithm, this can also lead to reductions in the model runtime.

### 2.3 Flow Surfaces and Routing Layers

Once the source, sink, rival use, and nonrival use layers have been created for the chosen
ecosystem service, the final piece of the puzzle is to generate a flow surface for the study
area. In a SPAN model, the flow surface is another raster map in which each cell contains an
encoded direction pointing to its next downstream cell. In this context, “upstream” means
nearer to the source point at the head of a flow path and “downstream” means nearer to
the use point at its end.

The arrangement of these arrows will be determined according to the flow path topology associated with the ecosystem service being studied (see Section 2.1). Importantly, neither in situ flow nor global flow services have a defined flow surface, since their calculations either happen independently by grid cell (in situ flow) or act strictly on the summed totals of the raster surfaces (global flow). For the remaining three flow path topologies, their construction usually proceeds as follows.

2.3.1 PROXIMAL FLOW SURFACE

Since no two users of proximal flow services interact within a SPAN simulation, we must create a separate flow surface for each one. To begin, we center a circle on the user’s location with a radius determined by both the user’s sense range and the environment’s ability to conduct the service medium. For example, if the ecosystem service is scenic beauty, the service medium is therefore visual information transmitted by light to the user’s eyes. If she has excellent vision and the air is clear, then the maximum view radius may be quite large. However, if her sight is poor and/or the air is hazy (e.g., from fog or smog), then this radius may be much smaller. Naturally, this value may vary from user to user across the landscape.

Once the region of effect has been established, every point within the circle is assigned a direction which points directly towards the user, called the trajectory. Unfortunately, since we are working within a raster model, movement between cells is only possible in the eight compass directions (N, E, S, W, NE, SE, SW, NW). To address this issue, we simply begin with the outermost ring of the circle and assign to each cell the nearest compass direction to their trajectory. For each such cell, we then calculate the difference between its compass direction’s angle and its trajectory, called the angular offset, and record it on the next downstream cell to which it points. Once the outermost ring of cells is complete,
we move inward to the next ring of cells and repeat the previous operation. However, for these cells and all cells nearer to the user, we first average all of the angular offsets received from their immediately upstream cells and add this to the base trajectory before snapping to the nearest compass direction and recording new angular offsets further downstream.

When this process has been completed for all cells within the region of effect, we should now have a unidirectional flow surface, which could best be seen as a funnel, streaming the service medium directly inward from sources on the flow surface to the single user at its center. For an in-depth look into the algorithmic steps behind creating a proximal flow surface, see Section 3.7.3 in Chapter 3.

### 2.3.2 Channeled Flow Surface

Whereas the proximal flow surface can be computed via a predictable trigonometric algorithm, ecosystem services with channeled flow types generally require additional location-specific information, called *routing layers*, to generate their flow surfaces. Fortunately, since users do interact in the presence of channeled flows, we need only create this raster layer once per service.

As an example, consider routing surface water over a landscape. Starting with a somewhat simplified model of hydrology, we may point each cell towards its neighboring cell with the lowest elevation. This embodies the idea that, given the opportunity, water always flows downhill. Refining this further, we might decide that all cells within standing bodies of water (e.g., streams, rivers, lakes) will always flow into their nearest aquatic neighbors with the lowest elevation. This helps us to keep water from rising out of streambeds if our grid resolution is too coarse. Should we feel it is appropriate for our study site, we might also set cells in flat areas to point in the same direction as their immediately upstream neighbors, thereby keeping our local elevation-based assignments from accidentally creating a loop in the flow surface. Note that for these hydrologic examples, the routing layers would
be two raster maps: 1) elevation and 2) the presence of water bodies.

In any case, the decisions that go into creating a channeled flow surface are left to the discretion of the modeler but should be sufficient to capture the spatial dynamics in question. Note, of course, that a unidirectional surface is not the only option for this flow type. Should multidirectional flows seem more appropriate, then each cell should be assigned a distribution over the eight compass directions, indicating what percentage of incoming flows should proceed to each of its neighboring cells. Finally, if flow direction is dependent on flow strength (as in the case of flooding), a conditional flow surface may be created. Here each cell contains a conditional expression of the form:

```
if test1 then direction1 else if test2 then direction2 ... else if testN then directionN
```

where test1 through testN are predicate functions of the service medium amount flowing out of each cell. Figure 2.8 provides examples of both unidirectional and multidirectional flow.

![Figure 2.8: Unidirectional and multidirectional flow types](image)

Regardless of the choice of a unidirectional, multidirectional, or conditional flow surface, all channeled flow surfaces must describe a directed acyclic graph (DAG). Otherwise, the SPAN flow distribution algorithm may never terminate. By representing the paths taken by the service medium with a DAG, all of the SPAN calculations may be performed in linear $O(n)$ time. As a corresponding disadvantage, feedback loops may not be captured by a single SPAN simulation. However, they may be modeled by running the SPAN flow distribution algorithm multiple times with different flow surfaces on each iteration.
2.3.3 Diffusive Flow Surface

The most difficult flow surfaces to construct are those belonging to the diffusive flow type. First, we project a circle around each source point out to its maximum range of effect and assign inward facing arrows to the cells within each circle using the trajectory + angular offset approach described earlier for creating proximal flow surfaces. However, since the flows emanating from diffusive sources may interact with one another, our second step involves combining all of these flow surfaces together.

We begin by choosing one source point and expanding into all of the neighboring cells which point at it along the current source’s flow surface. This expansion step is then repeated once for every other source in the study region. In the event that any cell is entered more than once during this operation, we combine each of the compass directions assigned to its intersecting flow surfaces into a new value as follows:

1. If all directions are identical, simply assign this direction once to the cell. This preserves the simplicity and computational efficiency of unidirectional flow for this cell.

2. If two directions oppose one another, combine them into a single multidirectional flow distribution with equal weights assigned to each. This is preferable to recording no outgoing direction because it represents the collision without creating a discontinuity in the flow surface.

3. If two directions are different but non-opposing, combine them into a single multidirectional flow distribution, along with every other compass direction that occurs in the sub-180° angle between them, using an equal weight distribution. This translates the collision into a fan effect, which once again ensures that no discontinuity will be created in the flow surface.

Once this local merging step is complete, we continue to expand each non-merged cell as before. These expanding and merging steps are then repeated until the maximum radius
of effect is reached for each source point. Merging the intersecting flow surfaces in this way ensures that we will create a DAG that includes all cells falling within the union of the original circular regions of effect assigned to each source point. Furthermore, all cells should lie along a continuous path from a source point to either a cell on the edge of its maximum range of effect or one containing a direct collision with another flow surface (see Step 2 above).

Finally, if the service medium is known to prefer some paths to others, we may overlay one or more additional flow paths on the merged flow surface using a set of routing layers as we did in the channeled flow case above. To combine these channeled paths with the merged proximal paths, we may either simply replace the old values with the new ones or merge them together into a multidirectional or conditional flow distribution.

Ultimately, the flow surface generated in this way is meant to represent the spatial distribution of unpredictable service media using what might be considered a mean field approach. So rather than interpreting the results of a diffusive flow model as showing the paths always taken by the service medium, they should instead be seen as an estimate of its expected movement patterns.

As an example, let us imagine that we are interested in the ecosystem service of coastal storm protection. The service medium might be wave energy generated by a hurricane located off the coast of our study area. The hurricane is the source of the service medium, and mangrove swamps along the coast act as wave energy sinks. People and their homes along the coast act as nonrival users of this service. Note that since the users will have a detrimental relationship with the service medium (i.e., more wave energy = worse quality of life), then the value of greatest interest from such an analysis is the amount of protection (i.e., preventive ecosystem service) that each user will receive from the mangrove swamps.

This is a clear case of diffusive flow since the hurricane emits wave energy in all directions, which may weaken with distance from its center. However, if the coastline contains one or
more river outlets, then the wave energy may be able to surge up these rivers unimpeded much further than it can over land. Therefore, creating this composite flow surface requires two steps:

1. The proximal flow surface generation algorithm is used to create a circular region centered on the hurricane source point with a maximum range of effect given by the wave energy’s expected distance decay.

2. If any coastal river outlets fall within the region created in Step 1, their associated river channels are overlayed on top of the circular flow surface with a distance decay function appropriate for riverine wave propagation.

To capture the stochasticity inherent in modeling unpredictable service media like the one described here, we may choose to incorporate Monte Carlo simulation into our analysis. For the hurricane example explored in this section, we might run multiple SPAN simulations, each with a different location and magnitude (i.e., source value) for the hurricane. Aggregating the results of such an experiment could provide a spatial distribution for the amount of preventive ecosystem services provided by the coastal mangrove swamps as well as the amount of damage expected at each use location.

2.4 THE FLOW DISTRIBUTION ALGORITHM

Once all of the raster maps (i.e., source, sink, rival use, nonrival use, and flow surface) have been created, the following SPAN flow distribution algorithm is applied to these layers to derive the connectivity relationships between them. In Sections 2.4.1 – 2.4.3, we walk through each of the algorithm’s three phases in detail, accomplishing the following tasks:

1. Order the directed acyclic graph described by the flow surface.
2. Propagate the service medium from sources through sinks to users.
3. Attribute the sink and use impacts to upstream sources.

For the remainder of this chapter, whenever we mention *theoretical* source, sink, rival use, or nonrival use values, we are referring to the original source, sink, rival use, and nonrival use maps defined in Section 2.2. Except for nonrival use, which contains boolean values, these represent the maximum production, absorption, or consumption values that each cell can contribute toward increasing or decreasing the service medium’s flow. However, it is rarely the case that these theoretical values will be fully realized in any SPAN simulation. Instead, by applying the SPAN flow distribution algorithm, we separate these theoretical values into their *inaccessible, possible, blocked, captured*, and *actual* constituents, as defined in Section 2.5.

As a reminder, proximal flow services must repeat this algorithm once for each user in the study region since each one produces its own independent flow surface. The result maps from these analyses will then be combined by summation (or averaging) per cell. However, since no shared flow surface exists for proximal flow services, the combined theoretical and inaccessible flow maps – lacking any definition – will be excluded from the final result set.

Additionally, both proximal flow and diffusive flow services have a distance decay component as part of their descriptions (see Sections 2.3.1 and 2.3.3). The design of this decay function is at the discretion of the modeler, but please note that its application is slightly different in these two cases. For proximal flow, the input source and sink values should both be weighted by their distance from the current user being analyzed (see Section 3.7.6 for an example). These will be the values used whenever theoretical source or theoretical sink are referenced in the following sections’ equations. For diffusive flow, we simply apply the decay function (which should be based on distance from the corresponding source) to the outgoing flow values for each cell on the flow surface.
2.4.1 Ordering the Serviceshed Cells by Outlet Distance

Because ecosystem service assessment in the SPAN framework is user-centric, we begin by identifying all of the cells which have a flow path to at least one downstream user. For each use cell on the landscape, we call the spatially contiguous grouping of its upstream cells its serviceshed. The use location at the end of each serviceshed is called its outlet point. Any cells which do not belong to at least one serviceshed may be safely ignored as they have no flow path to any user on the landscape and can therefore neither contribute to nor detract from the generation, delivery, or uptake of the service medium under study.

Figure 2.9 illustrates a hypothetical region containing three nested servicesheds, each associated with the use location (i.e. outlet point) at its end.

\[ \text{Figure 2.9: Nested servicesheds and their outlet points} \]

Once all of the servicesheds have been identified, each of their cells is labeled by the stepwise distance along the flow surface to their most downstream user. This provides a global ordering for the union of the servicesheds, which will be used during the forward propagation phase presented in Section 2.4.2.

Figure 2.10 shows such an ordering for the hypothetical servicesheds shown in Figure 2.9. Here, the horizontal lines separate groups of cells that share the same stepwise distance to the most downstream user.

For services with in situ flow, single cell servicesheds will necessarily only exist in grid
Figure 2.10: Nested servicesheds with a global ordering

cells containing both sources and users. For global flow services, the union of all source, sink, rival use, and nonrival use cells forms a kind of abstract serviceshed with all users acting as outlet points. However, for both of these flow path topologies, the ordering procedure described above may be disregarded.

In the case of proximal flow, each user produces its own flow surface in which all cells within the maximum range of effect are considered part of its serviceshed, and it is, of course, the only outlet point. As a result, each user’s flow surface must be ordered independently of the others (see Section 2.3.1).

Channeled flow surfaces may give rise to nested or overlapping servicesheds depending on the shape of their flow paths and placement of their users. If the flow surface is defined to be a unidirectional directed acyclic graph, then servicesheds may be nested but may not partially overlap. If the flow surface is multidirectional, then both nesting and overlapping may occur. In the case of a conditional flow surface, each cell may be treated as though it contains a multidirectional flow distribution for the purposes of ordering. When the forward propagation phase (see Section 2.4.2) is reached, the flow strength information will then be used to prune out the unnecessary branches in the flow graph.

Finally, diffusive flow surfaces are structured in such a way that many users’ flow paths are likely to merge on the way to the same source. Hence, overlapping servicesheds should be a common occurrence in all but the most trivial cases. Additionally, serviceshed nesting will
occur whenever any two users’ flow paths are collinear. Note that – in contrast to proximal and channeled flow surfaces – when traversing a diffusive flow surface, following the encoded directions in each cell moves one upstream (i.e., nearer to a source) while following them in reverse moves one downstream (i.e., nearer to a user) along an ever-branching path.

2.4.2 Propagating the Service Medium Downstream

The second step in the SPAN flow distribution algorithm is to propagate the service medium produced in the source locations along the ordered flow surface toward their downstream users. We begin with the cells furthest from their serviceshed outlet point(s), add their source values (if any) to the available service medium along their flow paths, subtract their local sink and rival use values, and propagate the remainder (if any) on to their next downstream cells. At no point will the service medium value become negative.

If the flow surface is multidirectional, then the service medium is distributed among the downstream cells according to the percentages locally assigned to each compass direction. If a conditional flow surface is used, the remaining flow value is passed as input to the test expressions, which should select one direction from the eight allowed. For a diffusive flow surface, note that the downstream direction is defined to branch even though a unidirectional flow surface is used. In this case, the remaining service medium in each cell may simply be evenly distributed among its downstream cells.

Once these calculations have been completed for the first set of cells, the current outlet distance is decremented by one step, and the local service medium calculations are repeated for all cells at the new distance. Once all the flow paths have been traversed to their final outlet point(s), we end the forward propagation part of our simulation.

Along the way, we compute the amount of the service medium absorbed (by sinks), consumed (by rival users), or experienced (by nonrival users) at each location. We also record the total upstream production (from sources), absorption (by sinks), and consumption (by
rival users) for each cell as well as its total incoming and outgoing flow amounts so as to maintain a history that we can use later. This information is then used to derive several new map layers, as presented below.

Let $S$ denote source values, $K$ denote sink values, $R$ denote rival use values, $N$ denote nonrival use values, and $F$ denote flow values. In the following formulas, the subscript $T$ means Theoretical, $I$ means Inaccessible, $P$ means Possible, $B$ means Blocked, $C$ means Captured, $A$ means Actual, $\Leftarrow$ means Incoming, and $\Rightarrow$ means Outgoing. Finally, let $c$ represent any grid cell in the serviceshed and $\text{parents}(c)$ denote the set of cells immediately adjacent to $c$ which flow directly into it.

The Incoming Flow to each cell $c$ is the sum of the Theoretical Source at $c$ and the Outgoing Flow amounts provided by each of its immediately upstream neighbors.

$$F_{\Leftarrow}(c) = S_T(c) + \sum_{p \in \text{parents}(c)} F_{\Rightarrow}(p, c)$$

Possible Sink is the amount absorbed by each sink location given the service medium quantity encountered. We use “possible” for this attribute rather than “actual” because the amount absorbed by any sink is only the upper bound for its impact on downstream users rather than the actual impact that will occur given the structure of the flow network.

$$K_P(c) = \min(F_{\Leftarrow}(c), K_T(c))$$

Actual Flow is the amount available to users after sink effects are applied.

$$F_A(c) = F_{\Leftarrow}(c) - K_P(c)$$

Actual Nonrival Use is equal to Actual Flow for any cells containing nonrival users since
they lack a limiting demand value in the SPAN framework.

$$\mathcal{N}_A(c) = \begin{cases} \mathcal{F}_A(c) & \text{if } \mathcal{N}_T(c) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Actual Rival Use is the amount captured by rival users after sink effects are applied.

$$\mathcal{R}_A(c) = \min(\mathcal{F}_A(c), \mathcal{R}_T(c))$$

Outgoing Flow is the service medium amount remaining after sink and rival use impacts have been subtracted. This value will be distributed among cell $c$’s downstream neighbors – here denoted by $\text{children}(c)$ – depending on the flow surface type (e.g., unidirectional, multidirectional, conditional).

$$\mathcal{F}_{\Rightarrow}(c) = \mathcal{F}_A(c) - \mathcal{R}_A(c)$$

$$\mathcal{F}_{\Rightarrow}(p) = \sum_{c \in \text{children}(p)} \mathcal{F}_{\Rightarrow}(p, c)$$

Figure 2.11 illustrates the flow relationship between sources ($\mathcal{S}_T$), sinks ($\mathcal{K}_T$), rival users ($\mathcal{R}_T$), and nonrival users ($\mathcal{N}_T$) within a single cell. The thick arrow emanating from $\mathcal{S}_T$ represents the cell’s Incoming Flow $\mathcal{F}_\leftarrow$. The thinner arrow leaving $\mathcal{K}_T$ is the Actual Flow $\mathcal{F}_A$, which retains the same value as it passes through $\mathcal{N}_T$. The final arrow exiting the cell is the Outgoing Flow $\mathcal{F}_{\Rightarrow}$, which then becomes one of the small arrows entering its next downstream cell.

Using the results of these formulas, we now derive 13 more properties for cell $c$.

Inaccessible Sink is the unused absorption capacity due to insufficient upstream production.

$$\mathcal{K}_I(c) = \mathcal{K}_T(c) - \mathcal{K}_P(c)$$
Blocked Flow is the total reduction in flow strength due to local and upstream sink effects.

\[ \mathcal{F}_B(c) = \kappa_P(c) + \sum_{p \in \text{parents}(c)} \mathcal{F}_B(p) \]

Captured Flow is the total reduction in flow strength due to upstream rival users.

\[ \mathcal{F}_C(c) = \sum_{p \in \text{parents}(c)} (\mathcal{R}_A(p) + \mathcal{F}_C(p)) \]

Possible Flow is the amount available to users in the absence of sinks and upstream rival users.

\[ \mathcal{F}_P(c) = \mathcal{F}_A(c) + \mathcal{F}_B(c) + \mathcal{F}_C(c) \]

Inaccessible Flow is the subset of the flow surface which does not lie on any path between source and use points.

\[ \mathcal{F}_I(c) = \begin{cases} \mathcal{F}_T(c) & \text{if } \mathcal{F}_P(c) = 0 \\ 0 & \text{otherwise} \end{cases} \]

Possible Nonrival Use is the amount that could be experienced by nonrival users in the
absence of sinks and upstream rival users.

\[ N_P(c) = \begin{cases} 
  \mathcal{F}_P(c) & \text{if } N_T(c) > 0 \\
  0 & \text{otherwise}
\end{cases} \]

Blocked Nonrival Use is the additional amount that would have been experienced by nonrival users in the absence of local and upstream sinks. For preventive services, this may be interpreted as protection from harm due to upstream sinks.

\[ N_B(c) = \begin{cases} 
  \mathcal{F}_B(c) & \text{if } N_T(c) > 0 \\
  0 & \text{otherwise}
\end{cases} \]

Captured Nonrival Use is the additional amount that would have been experienced by nonrival users in the absence of upstream rival users. For preventive services, this may be interpreted as protection from harm due to rival users.

\[ N_C(c) = \begin{cases} 
  \mathcal{F}_C(c) & \text{if } N_T(c) > 0 \\
  0 & \text{otherwise}
\end{cases} \]

Inaccessible Nonrival Use is a boolean presence/absence layer which shows those nonrival users with no upstream sources.

\[ N_I(c) = \begin{cases} 
  N_T(c) & \text{if } \mathcal{F}_P(c) = 0 \\
  0 & \text{otherwise}
\end{cases} \]

Possible Rival Use is the amount that could be captured by rival users in the absence of sinks and upstream rival users.

\[ \mathcal{R}_P(c) = \min(\mathcal{F}_P(c), \mathcal{R}_T(c)) \]
Blocked Rival Use is the additional amount that could have been captured by rival users in the absence of local and upstream sinks. For provisioning services, this may be interpreted as unmet demand due to sinks.

\[
R_B(c) = \begin{cases} 
\frac{F_B(c)(R_P(c) - R_A(c))}{F_B(c) + F_C(c)} & \text{if } F_B(c) > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Captured Rival Use is the additional amount that could have been captured by local rival users in the absence of upstream rival users. For provisioning services, this may be interpreted as unmet demand due to upstream competition for the service medium.

\[
R_C(c) = \begin{cases} 
\frac{F_C(c)(R_P(c) - R_A(c))}{F_B(c) + F_C(c)} & \text{if } F_C(c) > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Inaccessible Rival Use is the unmet demand due to insufficient upstream production.

\[
R_I(c) = R_T(c) - R_P(c)
\]

Since ecosystem services with \textit{in situ} flow lack downstream connections between cells, we simply treat all cells with both sources and users as their own distinct servicesheds. As a result, simply performing the local service medium calculations described above once for each of these cells effectively completes this phase of the SPAN flow distribution algorithm in one step.

For services with global flow paths, we create one abstract serviceshed cell containing the map sums of the source, sink, rival use, and nonrival use layers respectively. The local service medium calculations are then simply applied to this one cell, generating all of the forward propagation results.

Table 2.8 lists the 19 new map layers produced by the SPAN forward propagation phase.
Table 2.8: Results of the SPAN forward propagation phase

<table>
<thead>
<tr>
<th>Result Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_i$ Incoming Flow</td>
<td>Sum of local source and immediately upstream outgoing flows</td>
</tr>
<tr>
<td>$K_s$ Possible Sink</td>
<td>Amount locally absorbed by sinks</td>
</tr>
<tr>
<td>$F_A$ Actual Flow</td>
<td>Amount available to users after local sink impacts</td>
</tr>
<tr>
<td>$N_A$ Actual Nonrival Use</td>
<td>Amount experienced by nonrival users after local sink impacts</td>
</tr>
<tr>
<td>$R_A$ Actual Rival Use</td>
<td>Amount captured by rival users after local sink impacts</td>
</tr>
<tr>
<td>$F_o$ Outgoing Flow</td>
<td>Amount available after local sink and rival use impacts</td>
</tr>
<tr>
<td>$F_B$ Blocked Flow</td>
<td>Amount unavailable to users due to sink impacts</td>
</tr>
<tr>
<td>$N_B$ Blocked Nonrival Use</td>
<td>Amount missed by nonrival users due to sink impacts</td>
</tr>
<tr>
<td>$R_B$ Blocked Rival Use</td>
<td>Amount not captured by rival users due to sink impacts</td>
</tr>
<tr>
<td>$F_C$ Captured Flow</td>
<td>Amount unavailable to users due to upstream rival use</td>
</tr>
<tr>
<td>$N_C$ Captured Nonrival Use</td>
<td>Amount not captured by local rival users due to upstream rival use</td>
</tr>
<tr>
<td>$R_C$ Captured Rival Use</td>
<td>Amount not captured by rival users without sinks or upstream rival use</td>
</tr>
<tr>
<td>$F_P$ Possible Flow</td>
<td>Amount available to users without sinks or upstream rival use</td>
</tr>
<tr>
<td>$N_P$ Possible Nonrival Use</td>
<td>Amount experienced by nonrival users without sinks or rival use</td>
</tr>
<tr>
<td>$R_P$ Possible Rival Use</td>
<td>Amount captured by rival users without sinks or upstream rival use</td>
</tr>
<tr>
<td>$F_I$ Inaccessible Flow</td>
<td>Flow surface cells that do not lie between sources and users</td>
</tr>
<tr>
<td>$K_I$ Inaccessible Sink</td>
<td>Unused absorption capacity due to insufficient upstream production</td>
</tr>
<tr>
<td>$N_I$ Inaccessible Nonrival Use</td>
<td>Nonrival users with no upstream sources</td>
</tr>
<tr>
<td>$R_I$ Inaccessible Rival Use</td>
<td>Unmet demand due to insufficient upstream production</td>
</tr>
</tbody>
</table>

2.4.3 Backpropagating Source and Sink Impacts Upstream

In the third and final phase of the SPAN flow distribution algorithm, we follow the flow surface backwards from the outlet point(s) to the furthest source locations. Along the way, we assign the service medium values captured by sinks, rival users, and nonrival users throughout the serviceshed(s) to the sources which produced them and the sinks which prevented their delivery. By the end of this phase, we will have produced an additional 8 maps, increasing our result count to 27 layers total. However, we may discard 2 of these maps (Incoming Flow and Outgoing Flow) from this set as they are no longer needed after the steps in this section are complete.

Backpropagating the results of the forward propagation phase requires two impact distribution calculations in each cell, one for upstream source effects and the other for upstream sink effects. The following two sections cover these operations in detail.
Distributing the Source Impacts

Recall that the Incoming Flow $F_{\text{in}}(c)$ is the total service medium amount available to be sunk, experienced, or consumed in cell $c$. Let the relative contribution from each cell $p$ to cell $c$’s Incoming Flow be defined as follows:

$$W_S(p, c) = \begin{cases} 
S_T(c) & \text{if } p = c \\
F_{\text{in}}(p, c) & \text{if } p \in \text{parents}(c) \\
0 & \text{otherwise}
\end{cases}$$

Figure 2.12 illustrates a sample set of these weights (in yellow) for one cell with three parents. Note that the weights sum to 1 and that parent cells with higher Outgoing Flow values (and hence thicker outgoing flow lines) have correspondingly larger weights.

To properly allocate source effects, we multiply these weights by $c$’s Possible Sink $K_P(c)$, Actual Rival Use $R_A(c)$, and Actual Nonrival Use $N_A(c)$ values independently and assign the results to each of the contributing cells. This task of division by relative flow contribution is then repeated recursively all the way up the serviceshed, with each cell first combining its downstream Possible Sink $K^-_P(c)$, Actual Rival Use $R^-_A(c)$, and Actual Nonrival Use $N^-_A(c)$ values with its local values prior to applying the weighting formula. Notably, $K_P(c)$ and $R_A(c)$ values each combine additively while the most recent $N_A(c)$ value simply replaces more downstream ones. These mathematical relationships are shown below for a cell $p$ with.
$c \in \text{children}(p)$:

$$
\mathcal{K}_P(p) = \begin{cases} 
\mathcal{W}(p, c)\mathcal{K}(c) & \text{if } \mathcal{N}_A(c) > 0 \\
\mathcal{W}(p, c)(\mathcal{K}(c) + \mathcal{K}_P(c)) & \text{otherwise}
\end{cases}
$$

$$
\mathcal{R}_A(p) = \mathcal{W}(p, c)(\mathcal{R}_A(c) + \mathcal{R}_A(c))
$$

$$
\mathcal{R}_A(p) = \begin{cases} 
\mathcal{W}(p, c)(\mathcal{R}_A(c) + \mathcal{R}_A(c)) & \text{if } \mathcal{N}_A(c) > 0 \\
\mathcal{W}(p, c)\mathcal{R}_A(c) & \text{otherwise}
\end{cases}
$$

$$
\mathcal{N}_A(p) = \begin{cases} 
\mathcal{W}(p, c)\mathcal{N}_A(c) & \text{if } \mathcal{N}_A(c) > 0 \\
\mathcal{W}(p, c)\mathcal{N}_A(c) & \text{otherwise}
\end{cases}
$$

Whenever a source location is encountered (i.e., $\mathcal{S}_T(c) > 0$), we define the Actual Source $\mathcal{S}_A(c)$, Blocked Source $\mathcal{S}_B(c)$, Captured Source $\mathcal{S}_C(c)$, Possible Source $\mathcal{S}_P(c)$, and Inaccessible Source $\mathcal{S}_I(c)$ values in terms of these recursive definitions as follows:

$$
\mathcal{S}_A(c) = \begin{cases} 
\mathcal{W}(c, c)\mathcal{N}_A(c) & \text{if } \mathcal{N}_A(c) > 0 \\
\mathcal{W}(c, c)(\mathcal{N}_A(c) + \mathcal{R}_A(c) + \mathcal{R}_A(c) - \mathcal{R}_A(c)) & \text{otherwise}
\end{cases}
$$

$$
\mathcal{S}_B(c) = \begin{cases} 
\mathcal{W}(c, c)\mathcal{K}(c) & \text{if } \mathcal{N}_A(c) > 0 \\
\mathcal{W}(c, c)(\mathcal{K}(c) + \mathcal{K}_P(c)) & \text{otherwise}
\end{cases}
$$

$$
\mathcal{S}_C(c) = \mathcal{W}(c, c)(\mathcal{R}_A(c) + \mathcal{R}_A(c))
$$

$$
\mathcal{S}_P(c) = \mathcal{S}_B(c) + \mathcal{S}_A(c)
$$

$$
\mathcal{S}_I(c) = \mathcal{S}_T(c) - \mathcal{S}_P(c)
$$

The Actual Source value indicates the total amount of the produced Theoretical Source that is either consumed by rival users or non-destructively experienced by nonrival users.

The Blocked Source value at cell $c$ indicates the amount of the Theoretical Source produced at $c$ that is captured by sinks prior to reaching and impacting users.

The Captured Source value indicates the amount of the generated Theoretical Source
that is consumed by downstream rival users.

Possible Source is the amount of the Theoretical Source that could reach users if no sinks intervened.

The Inaccessible Source map shows the amount of the Theoretical Source that has no route to any user. Alternately, it may indicate a surplus of production that remains untapped by downstream rival users due to a lack of demand.

Note that whenever a service medium flow reaches a nonrival user, all sinks and rival users downstream of that point are ignored in the Blocked Source and Actual Source calculations in order to avoid double-counting.

Realize also that for services with in situ or global flows, the source weight $W_S(p,c)$ simplifies to the following definition:

$$W_S(p,c) = \begin{cases} 1 & \text{if } p = c \\ 0 & \text{otherwise} \end{cases}$$

Additionally, the downstream recursive values (i.e., $K_P(c), R_A^-(c), R_A^-(c)t, N_A^-(c)$) will always be 0 due to the fact that there are no downstream relationships between cells in these two flow path topologies.

**Distributing the Sink Impacts**

Recall that the Blocked Flow $F_B(c)$ is the total service medium amount blocked from reaching users in cell $c$ due to local and upstream sink effects. Let the relative contribution from each cell $p$ to cell $c$’s Blocked Flow be defined as follows:

$$W_K(p,c) = \begin{cases} \frac{K_P(c)}{F_B(c)} & \text{if } p = c \\ \frac{F_B(p)}{F_B(c)} & \text{if } p \in \text{parents}(c) \\ 0 & \text{otherwise} \end{cases}$$
Figure 2.13 illustrates a sample set of these weights (in yellow) for one cell with three parents. Note that the weights sum to 1 and that parent cells with higher Blocked Flow values (and hence thinner outgoing flow lines) have correspondingly larger weights.

![Diagram of weights](image)

Figure 2.13: Relative flow reductions by local and upstream cells

To allocate these sink effects, we multiply the weights by c’s Blocked Rival Use $R_B(c)$ and Blocked Nonrival Use $N_B(c)$ values independently and assign the results to each of the contributing cells. As in the source impact distribution case described above, this division by relative flow blockage is repeated recursively up the serviceshed, with each cell combining its downstream Blocked Rival Use $R_B(c)$ and Blocked Nonrival Use $N_B(c)$ values with its local values before applying the weighting formula. Here, the maximum of $R_B(c)$ and $R^{-}_B(c)$ prevails as does the maximum of $N_B(c)$ and $N^{-}_B(c)$. These mathematical relationships are shown below for a cell $p$ with $c \in \text{children}(p)$:

$$R_B^{-}(p) = \max(R_B(c), R^{-}_B(c))W_K(p, c)$$
$$N_B^{-}(p) = \max(N_B(c), N^{-}_B(c))W_K(p, c)$$

Whenever a sink location is encountered (i.e, $K_P(c) > 0$), we define the Actual Sink
$\mathcal{K}_A(c)$, Blocked Sink $\mathcal{K}_B(c)$, and Captured Sink $\mathcal{K}_C(c)$ values as follows:

$$
\mathcal{K}_A(c) = \max(\mathcal{R}_B(c), \mathcal{R}_B^-(c), \mathcal{N}_B(c), \mathcal{N}_B^-(c)) W_K(c, c)
$$

$$
\mathcal{K}_B(c) = \mathcal{K}_P(c) - \mathcal{K}_A(c)
$$

$$
\mathcal{K}_C(c) = \max(\mathcal{R}_B(c), \mathcal{R}_B^-(c)) W_K(c, c)
$$

The Actual Sink value is the amount of the Possible Sink that meaningfully reduces the available service flow to all downstream users.

The Blocked Sink value at cell $c$ is the amount of $c$’s Possible Sink that has no impact on downstream users, whether rival or nonrival.

The Captured Sink value is the amount of the Possible Sink that meaningfully reduces the available service flow to downstream rival users.

Realize that for services with \textit{in situ} or global flows, the sink weight $W_K(p, c)$ simplifies to the following definition:

$$
W_K(p, c) = \begin{cases} 
1 & \text{if } p = c \\
0 & \text{otherwise}
\end{cases}
$$

Additionally, the downstream recursive values (i.e., $\mathcal{R}_B^-(c), \mathcal{N}_B^-(c)$) will always be 0 due to the fact that there are no downstream relationships between cells in these two flow path topologies.

Table 2.9 lists the 8 new map layers produced by SPAN’s backpropagation phase.

2.5 **Understanding the SPAN Outputs**

Prior to running the SPAN flow distribution algorithm, we had created 5 maps (i.e., source, sink, rival use, nonrival use, and flow surface) per ecosystem service in our study area. After applying it to these layers (and discarding the intermediate results), we now have
Table 2.9: Results of the SPAN backpropagation phase

<table>
<thead>
<tr>
<th>Result Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_I$ Inaccessible Source</td>
<td>Theoretical Source with no route to a user</td>
</tr>
<tr>
<td>$S_P$ Possible Source</td>
<td>Theoretical Source amount which could reach users without sinks</td>
</tr>
<tr>
<td>$S_B$ Blocked Source</td>
<td>Theoretical Source amount unused due to absorption by sinks</td>
</tr>
<tr>
<td>$S_C$ Captured Source</td>
<td>Theoretical Source amount captured by rival users</td>
</tr>
<tr>
<td>$S_A$ Actual Source</td>
<td>Theoretical Source amount which reaches all users</td>
</tr>
<tr>
<td>$K_B$ Blocked Sink</td>
<td>Possible Sink amount that does not affect users</td>
</tr>
<tr>
<td>$K_C$ Captured Sink</td>
<td>Possible Sink amount that affects rival users</td>
</tr>
<tr>
<td>$K_A$ Actual Sink</td>
<td>Possible Sink amount that affects all users</td>
</tr>
</tbody>
</table>

25 additional maps, bringing us to a round 30 as our final total. With such a plethora of information coming out of this system, it is easy to become overwhelmed and confused as to how all of these results relate to one another.

Fortunately, despite the rather convoluted mathematics introduced in Section 2.4, the final relationships between these maps are actually quite simple. Realize, in fact, that the SPAN calculations merely partition the original input values in each grid cell into the fractions that actually deliver ecosystem services and those that don’t (and why).

For each of the source and sink maps, the following relationships should hold in every grid cell:

\[
P = B + A
\]

\[
T = I + P
\]

\[
= I + B + A
\]

\[
C \leq A
\]

These are the partitioning relationships for rival use:

\[
P = B + C + A
\]

\[
T = I + P
\]

\[
= I + B + C + A
\]

42
For the nonrival use and flow maps, these are the relationships that bind their results together:

\[ P = B + C + A \]

\[ I = \begin{cases} T & \text{if } P = 0 \\ 0 & \text{otherwise} \end{cases} \]

Remember that the input maps to the SPAN flow distribution algorithm are the theoretical values in these formulas. Each of the grid cell values in these maps has then been separated into the following 5 subcomponents:

**Actual** amount that is received by users (source), taken away from users (sink), available to users (flow), experienced by nonrival users (nonrival use), or consumed by rival users (rival use)

**Blocked** amount that is lost to sinks (source, flow, nonrival use, rival use) or does not affect users (sink)

**Captured** amount that is consumed by rival users (source), taken away from rival users (sink), or lost to upstream rival users (flow, nonrival use, rival use)

**Possible** amount that is received by users in the absence of sinks (source), absorbed from the flow path (sink), available to users in the absence of sinks and upstream rival use (flow), experienced by nonrival users in the absence of sinks and upstream rival use (nonrival use), or consumed by rival users in the absence of sinks and upstream rival use

**Inaccessible** amount that exceeds downstream demand (source), exceeds upstream production (sink, rival use), has no route from source to use (flow), or has no upstream source (nonrival use)
Each of these result maps may be used independently or in combination with others to answer any number of land management questions, which we will explore later in Section 4.3. However, the Actual maps should be of greatest interest to those engaged in ESAV because they let us know what fraction of the Theoretical values do, in fact, provide ecosystem services to users. For provisioning services, ecosystem service values should be assigned to source locations based on their Actual Source values. Likewise, for preventive services, the ecosystem service value assigned to each sink should be based on its Actual Sink value. Section 4.2 explores the options for integrating SPAN outputs into ecosystem service valuation procedures in greater detail.

This concludes our discussion of the SPAN framework’s terminology, algorithms, and results. In the next chapter, we will take a look at a Literate Program that implements the SPAN flow distribution algorithm for the proximal flow type and applies it to an assessment of the scenic beauty ecosystem service in Chittenden County, Vermont. By following along with the examples provided, we hope that this demonstration chapter will help to make the abstractions presented here far more tangible and intuitive.
CHAPTER 3

SCENIC BEAUTY IN CHITTENDEN COUNTY, VT

This chapter presents a polyglot Literate Program written in the spirit of Reproducible Research. (Knuth, 1984, Claerbout, 1992, Buckheit and Donoho, 1995) In these pages, you will not only find the results of numerous geospatial analyses but also the program code that generated them and the rationale behind each step taken. Furthermore, all of the datasets used in the following case study are accompanied by URLs whence they may be freely downloaded from the Vermont Center for Geographic Information (VCGI: http://vcki.vermont.gov). In this way, the SPAN methodology for modeling ecosystem services is made immediately available to the reader.

To follow along with the examples in this chapter, the reader will need to install several pieces of software, all of which are open source and/or freely available for all major operating systems. These programs are listed in Table 3.1 along with their minimum required versions and URLs from which they may be downloaded.

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgresql</td>
<td>9.3+</td>
<td><a href="http://www.postgresql.org">http://www.postgresql.org</a></td>
</tr>
<tr>
<td>PostGIS</td>
<td>2.1+</td>
<td><a href="http://postgis.net">http://postgis.net</a></td>
</tr>
<tr>
<td>Gnuplot</td>
<td>4.6+</td>
<td><a href="http://www.gnuplot.info">http://www.gnuplot.info</a></td>
</tr>
<tr>
<td>QGIS</td>
<td>2.2+</td>
<td><a href="http://qgis.org">http://qgis.org</a></td>
</tr>
<tr>
<td>Java Development Kit</td>
<td>1.7+</td>
<td><a href="http://www.java.com">http://www.java.com</a></td>
</tr>
<tr>
<td>Leiningen</td>
<td>2.4+</td>
<td><a href="http://leiningen.org">http://leiningen.org</a></td>
</tr>
</tbody>
</table>

Table 3.1: Software necessary to evaluate the code in this chapter
Postgresql (along with the PostGIS spatial extensions) will be used to load the input map layers obtained from VCGI, co-align them spatially, clip them to our study area boundary, and finally compute the source, sink, and nonrival use maps that will be fed into the SPAN flow distribution algorithm.

Note: A command line shell (e.g., bash, tcsh, cmd.exe) is needed to execute the database creation and data import commands. Simply use whichever shell is most readily available for your operating system. All SQL code can be entered into the shell prompt provided by Postgresql’s psql client package.

Gnuplot is used to create plots from database queries and to illustrate functions used in the SPAN analysis. For standalone plots, simply copy the gnuplot code provided into its shell prompt. Whenever a plot depends on a SQL query however, copy the output of this query into a file (e.g., “outfile.dat”) and replace the plot data . . . line in the subsequent gnuplot code with plot “outfile.dat” . . . in order to reproduce those plots.

QGIS is used to generate map visualizations from our PostGIS geodatabase. To view these map layers on your own computer, open QGIS and select Database → DB Manager from the dropdown menu. Establish a connection to the PostGIS database containing your maps, and then you should be able to select them from the tree view on the left side of the screen, right click, and choose Add to canvas.

The SPAN flow distribution algorithm is written in the Clojure programming language, which is a modern dialect of Lisp hosted on the Java Virtual Machine.(Hickey, 2008) As a result, a Java Development Kit is required to compile and run the SPAN code.

Finally, the Clojure build tool, Leiningen is used to download required libraries and provide a code evaluation prompt (a.k.a. REPL) into which we will enter the code making up the SPAN algorithm.

License Notice: All code presented in this chapter is solely the work of the author (Copyright 2014 Gary W. Johnson, Jr.) and is hereby made freely available under the terms
of the GNU General Public License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later version. See http://www.gnu.org/
licenses/gpl.html for more details.

3.1 Scenic Beauty As an Ecosystem Service

Although the SPAN methodology is intended to be sufficiently general to model many
ecosystem services, we only present the scenic beauty ecosystem service in detail for the
following reasons:

1. As a service belonging to the proximal flow type, scenic beauty requires a superset of
the simpler functionality used in the channeled flow models introduced in Section 2.1.3
and shares much of its functionality with the diffusive flow type described in Section
2.1.4. Therefore, after working through the examples presented in this chapter, one
should be capable of implementing services belonging to both of these categories. This
is left then as an exercise for the astute reader.

2. Both the in situ and global flow types are so simple to implement that a detailed
explanation is unlikely to be necessary once one has read Chapter 2.

3. From a software engineering perspective, all of the code related to the SPAN flow
distribution algorithm – arguably the heart of the system – is the same for every
ecosystem service modeled in the SPAN framework. Therefore, by demonstrating
its application to the scenic beauty service, everything about its operation is made
available to the reader.

Let us now proceed with some definitions.

The scenic beauty ecosystem service will be modeled as a flow of desirable visual infor-
mation from sources of beauty on the landscape to the eyes of one or more human users.
Landscape features deemed unattractive, or “visual blight”, act as sinks in this model. Because we are transmitting an informational service medium, these sinks detract from the quality of the views (reducing the desirability of the transferred information) rather than from their accessibility to users. As a model assumption, sinks that do not interfere with views of a source are ignored in the final scenic beauty value assigned to each user.

To account for landscape-level impedances to view access, we will weight the impact of each source and sink on each user by its visibility to (as determined by intervening topography) and distance from each user. As scenic beauty is not easily measured in concrete units (e.g., m$^3$ of water or tons of CO$_2$), the service medium in this model is represented with a real-valued unitless ranking $M \in [0, 1]$.

Finally, as views are a nonrival resource, users do not compete with one another in this model. Therefore, depending on the study area’s size and topography, each user may potentially be affected by views of every source and sink location without interfering with other users’ views of the same. In our software implementation, we accomplish this by running the SPAN flow distribution algorithm once for each user with all other users removed from the landscape. Once complete, we merge all 30 of the SPAN output maps from each run by summing them together across all of the users. In this way, we are able to calculate the total ecosystem service distribution values for our study area.

### 3.2 Setting Up the PostGIS Database

We begin by creating a new PostGIS database, in which we will perform all of our initial geospatial operations. When installing Postgresql, we should have been prompted to create an initial superuser called `postgres`, who has full permissions to create new databases and roles. We can log into the Postgresql server as this user with the following `psql` command.

```
psql -U postgres
```
Once logged in, we issue the following commands to create a new user account with our system login name (in my case, this is gjohnson). We then create a new database called span_vt for our analysis and import the PostGIS spatial extensions into it.

```
CREATE ROLE gjohnson WITH LOGIN CREATEDB;
CREATE DATABASE span_vt WITH OWNER gjohnson;
CREATE EXTENSION postgis;
```

From this point on, we will enter all the SQL code presented in this chapter into the psql prompt after connecting to database span_vt with our new user (e.g., gjohnson) as follows:

```
psql -U gjohnson -d span_vt
```

Now that we have created a database, we need to organize its contents into schema (i.e., groups of tables, sequences, and so on). To keep our scenic beauty analysis separate from any other ecosystem services we might wish to study in the future with this database, we store the scenic beauty models and results in the scenic_beauty schema.

```
CREATE SCHEMA scenic_beauty;
```

Because the datasets used in our scenic beauty models may be of use to other ecosystem service models, we will store them in two separate database schemas, called raster and vector respectively.

```
CREATE SCHEMA raster;
CREATE SCHEMA vector;
```

This concludes our PostGIS database setup.

### 3.3 Identifying the Study Area

In order to make this analysis more interesting to its most likely audience, we will choose Chittenden County, VT to be our study area. The Vermont county boundaries may be
obtained as a polygon dataset from the Vermont Center for Geographic Information (VCGI) at the following URL:

http://maps.vcgi.org/gisdata/vcgi/packaged_zips/BoundaryCounty_CNTYBNDS.zip

<table>
<thead>
<tr>
<th>Data Type</th>
<th>polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extent</td>
<td>VT statewide</td>
</tr>
<tr>
<td>Coordinate Reference System</td>
<td>Vermont State Plane meters (NAD83)</td>
</tr>
</tbody>
</table>

To load this dataset into PostGIS, we must first determine its SRID (Spatial Reference Identifier). The following SQL query looks it up in the PostGIS `spatial_ref_sys` table.

```sql
SELECT srid, substring(srtext from 1 for 50) || '...' AS srtext
FROM spatial_ref_sys
WHERE srtext LIKE '%Vermont%'
  AND proj4text LIKE '%%units=m%';
```

<table>
<thead>
<tr>
<th>srid</th>
<th>srtext</th>
</tr>
</thead>
<tbody>
<tr>
<td>2852</td>
<td>PROJCS['NAD83(HARN) / Vermont',GEOGCS['NAD83(HARN)...</td>
</tr>
<tr>
<td>3684</td>
<td>PROJCS['NAD83(NSRS2007) / Vermont',GEOGCS['NAD83(NS...</td>
</tr>
<tr>
<td>32145</td>
<td>PROJCS['NAD83 / Vermont',GEOGCS['NAD83',DATUM['Nor...</td>
</tr>
</tbody>
</table>

Since no reference to HARN or NSRS2007 can be found in the metadata for this dataset, we assume that the intended SRID is **EPSG:32145**. Assuming that we unzipped it into a directory named “../gis_data/vt_counties/”, we can now load this dataset into PostGIS with the following `shp2pgsql` command.

```bash
shp2pgsql -s 32145 -g geom -I
../gis_data/vt_counties/Boundary_CNTYBNDS_poly.shp
vector.vt_counties | psql -U gjohnson -d span_vt
```

Opening table `vector.vt_counties` in QGIS displays the VT county boundaries contained in the newly imported dataset. Figure 3.1 shows this map with Chittenden county highlighted.
3.4 Source Model

We define the source function $S$ to be the maximum of elevation change and land cover rarity, normalized to the range $[0, 1]$. We represent change in elevation with degrees of slope $\theta \in [0, 90]$. We estimate the rarity of each land cover type $\omega \in \Omega$ with the self-information function from information theory, also called the “surprisal” of an event. (Shannon, 1948)

$$I(\Omega) = \log \left( \frac{1}{P(\Omega)} \right) = -\log(P(\Omega))$$

where $\Omega$ is a discrete random variable with probability mass function $P(\Omega)$.

In order to restrict the source function’s range $S \in [0, 1]$, we normalize $\theta$ by dividing by 90 and $I(\omega)$ by dividing by the maximum self-information $I(\Omega)_{\text{max}} = \log(N)$, where $N$ is
the number of grid cells in the study region. Thus the final source function $S_i$ for any cell $1 \leq i \leq N$ is defined as follows:

$$S_i = \max\left(\frac{\theta_i}{90}, I(\omega_i) \log(N)\right)$$

### 3.4.1 Slope Data

We will generate the slope data for this analysis from the USGS National Elevation Database (NED) based DEM24 data at 30m resolution. This dataset can be downloaded from VCGI at the following url:


**Table 3.3: Basic metadata for the USGS NED DEM24 dataset**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>raster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>30 meters</td>
</tr>
<tr>
<td>Extent</td>
<td>VT statewide</td>
</tr>
<tr>
<td>Value Units</td>
<td>feet</td>
</tr>
<tr>
<td>Coordinate Reference System</td>
<td>Vermont State Plane meters (NAD83)</td>
</tr>
</tbody>
</table>

Since the coordinate reference system is the same as our VT county boundaries dataset from earlier, we already know that its SRID is **EPSG:32145**. Thus, provided that we have unzipped the elevation layer into a directory named “../gis_data/vt_elevation_30m/”, we can simply import it into PostGIS with the `raster2pgsql` command shown below. To speed up our calculations, we tile this raster dataset into 15km x 15km square regions and create a GiST (Generalized Search Tree) spatial index on the raster column.

```
raster2pgsql -d -s 32145 -t 500x500 -I -C -
../gis_data/vt_elevation_30m/dem_24/dem_24
`raster.elevation30m` | psql -U gjohnson -d span_vt
```

Because we are restricting our analysis to Chittenden county, we can clip out just the portion of our elevation dataset that covers this region and store it for later use. While we
are at it, we also translate the elevation units from feet to meters to simplify our future slope calculations.

```sql
CREATE TABLE scenic_beauty.chittenden_elev AS
    WITH chittenden_county AS
        (SELECT geom
         FROM vector.vt_counties
         WHERE cntyname = 'CHITTENDEN'),
    chittenden_elev AS
        (SELECT ST_Union(ST_Clip(rast,geom),1) AS rast
         FROM raster.elevation30m CROSS JOIN chittenden_county
         WHERE ST_Intersects(rast,geom))
    SELECT ST_MapAlgebra(rast, '16BSI', '((rast.val)*0.3048)::integer', -32768) AS rast
    FROM chittenden_elev;
SELECT AddRasterConstraints('scenic_beauty'::name, 'chittenden_elev'::name, 'rast'::name);
```

Let’s see how many elevation cells (i.e., raster grid cells) exist in the imported dataset for Chittenden county.

```sql
SELECT real_valued_cells,
    total_cells - real_valued_cells AS nodata_cells,
    total_cells,
    100 * (total_cells - real_valued_cells) / total_cells AS nodata_percentage
FROM (SELECT ST_Count(rast) AS real_valued_cells,
    ST_Count(rast, false) AS total_cells
     FROM scenic_beauty.chittenden_elev) AS cell_counts;
```

<table>
<thead>
<tr>
<th>real_valued_cells</th>
<th>nodata_cells</th>
<th>total_cells</th>
<th>nodata_percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1788007</td>
<td>1297793</td>
<td>3085800</td>
<td>42</td>
</tr>
</tbody>
</table>

Over 3 million cells is quite a lot, but fully 42% of those are nodata cells. Let’s visualize this elevation map to better understand the landscape in which we are working. Figure 3.2 shows the results of opening table `scenic_beauty.chittenden_elev` in QGIS.

Of course, the scenic beauty source function we defined above relies on slope in degrees, not elevation in meters. Thus, we will need to derive the slope values using PostGIS’ `ST_Slope` function.
The results of this slope calculation can be seen in Figure 3.3, which we produce by visualizing table `scenic_beauty.chittenden_slope` in QGIS. This concludes our work in generating the slope data for our scenic beauty source model.

### 3.4.2 Land Cover Surprisal Data

For land cover data, we use the USGS’s 2001 National Land Cover Database (NLCD) at 30 meter resolution. This can be downloaded from VCGI at the following url:
Figure 3.3: Chittenden county slope in degrees at 30 meter resolution


Table 3.4: Basic metadata for the USGS NLCD dataset

<table>
<thead>
<tr>
<th>Data Type</th>
<th>raster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>30 meters</td>
</tr>
<tr>
<td>Extent</td>
<td>VT statewide</td>
</tr>
<tr>
<td>Value Units</td>
<td>land cover code</td>
</tr>
<tr>
<td>Coordinate Reference System</td>
<td>Vermont State Plane meters (NAD83)</td>
</tr>
</tbody>
</table>

Once again, we know that the SRID is **EPSG:32145**, so we can proceed directly to the `raster2pgsql` import step. To speed up our calculations, we tile this raster dataset into 15km x 15km square regions and create a GiST (Generalized Search Tree) spatial index on the raster column. In this case, we have unzipped the NLCD dataset into a directory named “../gis_data/vt_land_cover_30m/”.

```
raster2pgsql -d -s 32145 -t 500x500 -I -C \
```
Because we are restricting our analysis to Chittenden county, we can clip out just the portion of our land cover dataset that covers this region and store it for later use.

```sql
CREATE TABLE scenic_beauty.chittenden_land_cover AS
WITH chittenden_county AS
  (SELECT geom
   FROM vector.vt_counties
   WHERE cntyname = 'CHITTENDEN')
SELECT ST_Union(ST_Clip(rast,geom),1) AS rast
FROM raster.land_cover30m
CROSS JOIN chittenden_county
WHERE ST_Intersects(rast,geom);

SELECT AddRasterConstraints('scenic_beauty'::name,
 'chittenden_land_cover'::name,
 'rast'::name);
```

Before moving forward with our analysis, we should verify that the land cover dataset is correctly aligned with the slope dataset from the previous section. The following query shows the relevant georeferencing metadata for each of these layers.

```sql
SELECT 'slope' AS layer, upperleftx::integer,
  upperlefty::integer, width, height, scalex, scaley
FROM (SELECT (ST_MetaData(rast)).* FROM scenic.beauty.chittenden_slope) AS slope
UNION ALL
SELECT 'land cover' AS layer, upperleftx::integer,
  upperlefty::integer, width, height, scalex, scaley
FROM (SELECT (ST_MetaData(rast)).* FROM scenic.beauty.chittenden_land_cover) AS land_cover;
```

<table>
<thead>
<tr>
<th>layer</th>
<th>upperleftx</th>
<th>upperlefty</th>
<th>width</th>
<th>height</th>
<th>scalex</th>
<th>scaley</th>
</tr>
</thead>
<tbody>
<tr>
<td>slope</td>
<td>431505</td>
<td>247233</td>
<td>1480</td>
<td>2085</td>
<td>30</td>
<td>-30</td>
</tr>
<tr>
<td>land cover</td>
<td>431508</td>
<td>247242</td>
<td>1480</td>
<td>2086</td>
<td>30</td>
<td>-30</td>
</tr>
</tbody>
</table>

Unfortunately, it would appear that the land cover dataset was not correctly aligned by the source authority (i.e., VCGI) in this case. To rectify this situation, we must resample one layer to match the same grid alignment as the other. Since the land cover layer covers slightly more geographic area, we resample it to match the extent and resolution of the slope layer.
CREATE TABLE scenic_beauty.chittenden_land_cover_realigned AS
WITH land_cover_resampled AS
  (SELECT ST_Resample(land_cover.rast,
   slope.rast,
   'NearestNeighbor') AS rast
   FROM scenic_beauty.chittenden_slope AS slope
   CROSS JOIN scenic_beauty.chittenden_land_cover AS land_cover)
SELECT ST_Intersection(land_cover.rast,
   slope.rast,
   'BAND1') AS rast
FROM land_cover_resampled AS land_cover
CROSS JOIN scenic_beauty.chittenden_slope AS slope;
SELECT AddRasterConstraints('scenic_beauty'::name,
   'chittenden_land_cover_realigned'::name,
   'rast'::name);

We can now repeat the previous metadata comparison using our new land cover layer to verify that the re-alignment worked correctly.

As all dimensions now appear to match, we will proceed with visualizing this data layer by opening table scenic_beauty.chittenden_land_cover_realigned in QGIS. The resulting image can be seen in Figure 3.4.

We now turn our attention to the land cover classes appearing in this dataset. The following code matches up the numeric codes used in the map with the NLCD class names which provide their meaning. This encoding can be seen in Table 3.5.
We are now ready to calculate the self-information \( I(\Omega) \) of each land cover class in our
Table 3.5: NLCD land cover classes present in Chittenden county

<table>
<thead>
<tr>
<th>value</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Open Water</td>
</tr>
<tr>
<td>21</td>
<td>Developed, Open Space</td>
</tr>
<tr>
<td>22</td>
<td>Developed, Low Intensity</td>
</tr>
<tr>
<td>23</td>
<td>Developed, Medium Intensity</td>
</tr>
<tr>
<td>24</td>
<td>Developed, High Intensity</td>
</tr>
<tr>
<td>31</td>
<td>Barren Land (Rock/Sand/Clay)</td>
</tr>
<tr>
<td>41</td>
<td>Deciduous Forest</td>
</tr>
<tr>
<td>42</td>
<td>Evergreen Forest</td>
</tr>
<tr>
<td>43</td>
<td>Mixed Forest</td>
</tr>
<tr>
<td>52</td>
<td>Shrub/Scrub</td>
</tr>
<tr>
<td>71</td>
<td>Grassland/Herbaceous</td>
</tr>
<tr>
<td>81</td>
<td>Pasture/Hay</td>
</tr>
<tr>
<td>82</td>
<td>Cultivated Crops</td>
</tr>
<tr>
<td>90</td>
<td>Woody Wetlands</td>
</tr>
<tr>
<td>95</td>
<td>Emergent Herbaceous Wetlands</td>
</tr>
</tbody>
</table>

study area. For the purposes of this model, we will only consider undeveloped landscapes as sources of scenic beauty. Therefore, we exclude land cover codes 22, 23, and 24 when computing the distribution of cells. Because Vermonters are known to be fond of agriculture, we include codes 81 and 82 in this analysis.

The following code summarizes the number of cells per land cover class and the associated self-information values (in nats) that are derived from this distribution. These relationships can be seen in Table 3.6.

```sql
WITH value_counts AS
  (SELECT (ST_ValueCount(rast)).* FROM scenic_beauty.chittenden_land_cover_realigned),
value_percents AS
  (SELECT value, (count::double precision)/sum(count) OVER () AS percent FROM value_counts WHERE value NOT IN (22,23,24)),
value_information AS
  (SELECT value, -ln(percent) AS self_information FROM value_percents)
SELECT name, count, round(100*percent::numeric,1) AS percent, round(self_information::numeric,2) AS self_information FROM scenic_beauty.chittenden_land_cover_classes
INNER JOIN value_counts USING (value)
INNER JOIN value_percents USING (value)
INNER JOIN value_information USING (value)
ORDER BY value;
```
Table 3.6: Self information per land cover class

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>percent</th>
<th>self_information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Water</td>
<td>239019</td>
<td>14.1</td>
<td>1.96</td>
</tr>
<tr>
<td>Developed, Open Space</td>
<td>81827</td>
<td>4.8</td>
<td>3.03</td>
</tr>
<tr>
<td>Barren Land (Rock/Sand/Clay)</td>
<td>2285</td>
<td>0.1</td>
<td>6.61</td>
</tr>
<tr>
<td>Deciduous Forest</td>
<td>581897</td>
<td>34.4</td>
<td>1.07</td>
</tr>
<tr>
<td>Evergreen Forest</td>
<td>138035</td>
<td>8.2</td>
<td>2.51</td>
</tr>
<tr>
<td>Mixed Forest</td>
<td>224159</td>
<td>13.2</td>
<td>2.02</td>
</tr>
<tr>
<td>Shrub/Scrub</td>
<td>24796</td>
<td>1.5</td>
<td>4.22</td>
</tr>
<tr>
<td>Grassland/Herbaceous</td>
<td>3716</td>
<td>0.2</td>
<td>6.12</td>
</tr>
<tr>
<td>Pasture/Hay</td>
<td>249501</td>
<td>14.7</td>
<td>1.91</td>
</tr>
<tr>
<td>Cultivated Crops</td>
<td>92451</td>
<td>5.5</td>
<td>2.91</td>
</tr>
<tr>
<td>Woody Wetlands</td>
<td>45431</td>
<td>2.7</td>
<td>3.62</td>
</tr>
<tr>
<td>Emergent Herbaceous Wetlands</td>
<td>9696</td>
<td>0.6</td>
<td>5.16</td>
</tr>
</tbody>
</table>

For ease of understanding, we can show the self-information per land cover class as a bar chart.

```
reset
set term postscript solid color eps enhanced
set title 'Self-Information per Undeveloped Land Cover Class' \
  '(NLCD 2001) in Chittenden County, VT'
set xlabel 'Land Cover Class'
set xtics nomirror rotate by -90 offset 1,0
set ylabel 'Self-Information (nats)'
set yrange [0:]
set style data histogram
set style histogram clustered
set style fill solid 1.0 border lt -1
set grid
plot data using 4:xtic(1) notitle
```
Finally, we can use PostGIS' ST_Reclass function to replace the land cover codes in our base map with their calculated self-information, or surprisal, values in nats. The following code performs this reclassing operation and stores the results in table `scenic_beauty.chittenden_surprisal`.

```
CREATE TABLE scenic_beauty.chittenden_surprisal AS
SELECT ST_Reclass(rast,
1,
'11:1.95760387453894,' ||
'21:3.02953965872250,' ||
'[22-24]:0.0,' ||
'31:6.60778089687245,' ||
'41:1.06785346509428,' ||
'42:2.50663964502993,' ||
'43:2.02179129973908,' ||
'52:4.22346457137894,' ||
'71:6.12149910026702,' ||
'81:1.91468399801904,' ||
'82:2.90746814676633,' ||
'90:3.61795222980314,' ||
'95:5.16243349189165',
'32BF'),
-1.0) AS rast
```
FROM scenic_beauty.chittenden_land_cover_realigned;
SELECT AddRasterConstraints('scenic_beauty'::name,
   'chittenden_surprisal'::name,
   'rast'::name);

Opening the generated table in QGIS produces the surprisal map shown in Figure 3.6. This concludes our work in generating the self-information data for our scenic beauty source model.

![Chittenden County Land Cover Surprisal](image)

*Figure 3.6: Chittenden county land cover surprisal in nats at 30 meter resolution*

### 3.4.3 Scenic Beauty Source Data

Now that we have calculated the slope and relative rarity (i.e., self-information/surprisal) of the undeveloped land cover classes in Chittenden county, we are ready to apply the source function from Section 3.4 to generate the scenic beauty source map.
CREATE TABLE scenic_beauty.chittenden_source AS
SELECT ST_MapAlgebra(slope.rast, surprisal.rast, 'greatest([rast1.val]/90, [rast2.val]/ln(1692813))', '32BF') AS rast
FROM scenic_beauty.chittenden_slope AS slope
CROSS JOIN scenic_beauty.chittenden_surprisal AS surprisal;
SELECT AddRasterConstraints('scenic_beauty'::name, 'chittenden_source'::name, 'rast'::name);

Note: The number of undeveloped cells \( N \) in the surprisal map is 1692813.

The following plot shows the distribution of the scenic beauty source values computed over our study area.

SELECT value, (count::double precision)/sum(count) OVER () AS percent
FROM (SELECT (ST_ValueCount(rast)).*
FROM scenic_beauty.chittenden_source) AS value_counts
ORDER BY value;

reset
dot set title 'Scenic Beauty Source Histogram for Chittenden County, VT'
dot set xlabel 'Scenic Beauty Source Value'
dot set xrange [0:1]
dot set xtics 0.1 out nomirror
dot set ylabel 'Percent Coverage'
dot set yrange [0:]
dot set grid
plot data using 1:2 with impulses linwidth 8 notitle
Finally, we visualize the scenic beauty source map in Figure 3.8 by opening table `scenic_beauty.chittenden_source` in QGIS. This concludes our work in generating the scenic beauty source data.

### 3.5 Sink Model

We define the sink function $\mathcal{K}$ to be a simple linear function of the amount of development in each grid cell. Table 3.7 shows the three land cover codes excluded from our source model and their associated amounts of impervious surface.

<table>
<thead>
<tr>
<th>code</th>
<th>name</th>
<th>percent impervious</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Developed, Low Intensity</td>
<td>(20,50)</td>
</tr>
<tr>
<td>23</td>
<td>Developed, Medium Intensity</td>
<td>(50,80)</td>
</tr>
<tr>
<td>24</td>
<td>Developed, High Intensity</td>
<td>(80,100)</td>
</tr>
</tbody>
</table>
We will use the midpoint of each code’s imperviousness range to represent the average development in a cell with that label. In order to keep our landscape sinks from completely dominating our views, we then scale these values down by a factor of 10. While this choice is guided largely by the author’s previous experiences with such models, please note that for a non-demonstration model, it would be necessary to calibrate such coefficients according to known user preferences within the study region. The sink function $\mathcal{K} \in [0, 0.1]$ is therefore defined as follows:

$$\mathcal{K}_i = \begin{cases} 
0.035 & : \text{code} = 22 \\
0.065 & : \text{code} = 23 \\
0.090 & : \text{code} = 24 
\end{cases}$$
3.5.1 Scenic Beauty Sink Data

Since we have already loaded in a land cover dataset for Chittenden county, we can generate the sink layer with a single call to PostGIS’ ST_Reclass function.

```sql
CREATE TABLE scenic_beauty.chittenden_sink AS
SELECT ST_Reclass(rast, 1,
                     '[11-21]:0.0, 22:0.035, 23:0.065, 24:0.090, [31-95]:0.0',
                     '32BF', -1.0) AS rast
FROM scenic_beauty.chittenden_land_cover_realigned;
SELECT AddRasterConstraints('scenic_beauty'::name,
                              'chittenden_sink'::name,
                              'rast'::name);
```

The following plot shows the distribution of the scenic beauty sink values computed over our study area. Note that we use a semilog scale here.

```sql
SELECT value, (count::double precision)/sum(count) OVER () AS percent
FROM (SELECT (ST_ValueCount(rast)).*
        FROM scenic_beauty.chittenden_sink) AS value_counts
ORDER BY value;
```

```bash
reset
set term postscript solid color eps enhanced
set title 'Scenic Beauty Sink Histogram for Chittenden County, VT'
set xlabel 'Scenic Beauty Sink Value'
set xrange [0.0:0.1]
set xtics 0.01 out nomirror
set ylabel 'Percent Coverage'
set yrange [0.001:1]
set grid
set logscale y
plot data using 1:2 with impulses linewidth 8 notitle
```
Opening table `scenic_beauty.chittenden_sink` in QGIS produces the map shown in Figure 3.10. This concludes our work in generating the scenic beauty sink data.

### 3.6 Use Model

Since scenic beauty is a nonrival ecosystem service, we define the use function $\mathcal{N}$ to be a simple presence/absence function that returns 1 for grid cells containing potential beneficiaries of scenic views and 0 otherwise. This trivial boolean function $\mathcal{N} \in \{0, 1\}$ may be written as follows:

$$
\mathcal{N}_i = \begin{cases} 
0 & : \text{user absent} \\
1 & : \text{user present} 
\end{cases}
$$

For this example, we use Vermont’s public outdoor recreation sites as our view beneficiaries, since higher quality views may increase the attractiveness of these sites to visitors.
3.6.1 Scenic Beauty Use Data

To spatially locate our potential view beneficiaries, we use a point dataset of Vermont outdoor recreation sites provided by the Vermont Forest and Parks Department. This can be downloaded from VCGI at the following url:

http://anrmaps.vermont.gov/websites/vgisdata/layers_anr/PrePackaged_Shapefiles/TourismRecreation_RECSITES.zip

<table>
<thead>
<tr>
<th>Table 3.8: Basic metadata for the VT outdoor recreation sites dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
</tr>
<tr>
<td>Extent</td>
</tr>
<tr>
<td>Coordinate Reference System</td>
</tr>
</tbody>
</table>
Since we already know that the SRID for this dataset is **EPSG:32145**, we can load it into our PostGIS database with the following `shp2pgsql` command. In this case, we unzipped it first into a directory named “..gis_data/vt_recreation_sites/”.

```sql
shp2pgsql -s 32145 -g geom -I \
  ../gis_data/vt_recreation_sites/Tourism_RECSITES_point.shp \
  vector.vt_recreation_sites | psql -U gjohnson -d span_vt
```

Because we are restricting our analysis to Chittenden county, we can clip out just those sites contained within it to avoid unnecessary calculations in the future.

```sql
CREATE TABLE scenic_beauty.chittenden_rec_sites AS
WITH chittenden_county AS
  (SELECT geom
   FROM vector.vt_counties
   WHERE cntyname = 'CHITTENDEN')
SELECT r.site_name, r.geom
FROM vector.vt_recreation_sites AS r
CROSS JOIN chittenden_county AS c
WHERE ST_Within(r.geom, c.geom);
```

Since the SPAN flow distribution model demonstrated later in this chapter does not accept vector maps as inputs, we need to rasterize this point dataset to the same 30 meter resolution as our source and sink datasets. Fortunately, PostGIS' `ST_AsRaster` and `ST_MapAlgebra` functions may be readily applied to accomplish this task.

```sql
CREATE TABLE scenic_beauty.chittenden_use AS
WITH merged_users AS
  (SELECT ST_Collect(geom) AS geom
   FROM scenic_beauty.chittenden_rec_sites),
rasterized_users AS
  (SELECT ST_AsRaster(geom,rast,'1BB') AS rast
   FROM merged_users
   CROSS JOIN scenic_beauty.chittenden_source)
SELECT ST_MapAlgebra(use.rast,
  source.rast,
  '1',
  'BBSI',
  'SECOND',
  '0',
  '1',
  -128) AS rast
FROM rasterized_users AS use
CROSS JOIN scenic_beauty.chittenden_source AS source;
SELECT AddRasterConstraints('scenic_beauty'::name,
  'chittenden_use'::name,
  'rast'::name);
```

69
As a final quirk, it will take us two steps to create a proper visualization of our nonrival use map. Opening table `scenic_beauty.chittenden_use` in QGIS does provide us access to the map we have just created. However, because the resolution of this dataset is so high, it can be very challenging to see the few, scattered user cells against the background color. To give us a clearer image of their locations, we need to overlay the vector layer of recreation sites stored in table `scenic_beauty.chittenden_rec_sites`. Figure 3.11 shows the combined result map. This concludes our work in generating the scenic beauty use data.

![Chittenden County Scenic Beauty Nonrival Use](image)

*Figure 3.11: Scenic beauty use map for Chittenden county at 30 meter resolution*
3.7 Flow Model

To route the lines of sight between sources of scenic beauty and our chosen use locations, we employ an implementation of the SPAN proximal flow model written in the Clojure language. Although PostGIS has thus far provided us with all the necessary tools to generate our source, sink, and (non)rival use maps, it lacks an expressive programming model for dynamic simulations. By using Clojure for this, we can write succinct and expressive code to describe our flow model that should also be highly performant due to its compilation to Java bytecode. Additionally, we can make use of Clojure’s powerful parallel and concurrent programming features to get an even greater speedup on multicore hardware.

3.7.1 Setting Up the Clojure Environment

Because Clojure is implemented on the Java Virtual Machine (JVM), we must explicitly list all of the libraries used by our program on the Java classpath. Fortunately, the Clojure build program, Leiningen, can handle downloading and storing these libraries as well as linking them to the Clojure process at runtime. However, in order for Leiningen to know which libraries are needed, we must first create its config file, called “project.clj”, and place it in the directory from which we will call our Clojure program. A minimal but complete project.clj for the SPAN implementation demonstrated in this chapter is shown below.

```clojure
(defproject org.clojars.lambdaтратonic/span "2.0.0"
  :description "SPAN 2.0 Framework"
  :dependencies [[org.clojure/clojure "1.6.0"]
                  [org.clojure/java.jdbc "0.3.5"]
                  [postgresql/postgresql "9.1-901-1.jdbc4"]
                  [net.mikera/core.matrix "0.22.0"]
                  [net.mikera/vectorz-clj "0.21.0"]
                  [org.clojars.lambdaтратonic/matrix-viz "0.1.7"]]
  :min-lein-version "2.0.0"
  :global-vars {*warn-on-reflection* true})
```
Once this file is created, we need to run Leiningen once from the same directory to
download these library dependencies.

```
lein deps
```

Now we are ready to begin working our way through the SPAN proximal flow model.
If you want to follow along with the program as it is presented in the next few sections,
simply run the following Leiningen command once from your shell to open a REPL (Read-
Evaluate-Print-Loop).

```
lein repl
```

By copying all the code snippets shown in the remainder of this chapter into the REPL,
you should see the same results presented below.

### 3.7.2 Transferring the PostGIS Layers to Clojure

To begin, we must copy the raster grids for our source, sink, and nonrival use layers from
PostGIS to our Clojure process. We also copy over the elevation layer to use in routing our
lines of sight. In order to reduce the number of grid cells that must be analyzed by the flow
simulation (because spatial distributions at less than 100m resolution will not be visible in
this chapter’s figures anyway), we perform two pre-processing steps on each of these raster
layers:

1. Downscaling their cell resolution from 30 meters to 100 meters.
2. Setting cell values to 0 if they are less than a provided threshold.

For this demonstration, we will use the following per-layer threshold values:

**Source Layer** $\theta_s = 0.15$

**Sink Layer** $\theta_K = 0.01$
Naturally, a threshold would have little purpose for the nonrival use layer, since its values represent a boolean presence/absence state. Also, thresholding the elevation map would only lead to erroneous sight line calculations later, so we pass it through unchanged as well. Finally, in order to efficiently store these matrices as 2D double arrays, we convert all nodata values to -1.0.

```clojure
(use 'clojure.core.matrix)
(use 'clojure.core.matrix.operators)
(set-current-implementation :vectorz)
(use 'clojure.java.jdbc)
(import 'org.postgresql.jdbc4.Jdbc4Array)

(defn postgis-raster-to-matrix
  "Send a SQL query to the database given by db-spec for a raster tile from
table table-name. Resample the raster to match resolution and set any values
below threshold to 0. Return the post-processed raster values as a Clojure
matrix using the core.matrix API."
[db-spec table-name resolution threshold]
(let
  [rescale-query (if resolution
                   (format "ST_Rescale(rast,%s,-%s,'NearestNeighbor')" resolution resolution)
                   "rast")
   threshold-query (if threshold
                   (format (str "ST_MapAlgebra(%s,NULL,
                                 'CASE WHEN [rast.val] < %s
                                 THEN 0.0 ELSE [rast.val] END')")
                               rescale-query threshold)
                   rescale-query)
   data-query (format "SELECT ST_DumpValues(%s,1) AS matrix FROM %s" threshold-query table-name)]
(with-db-transaction [conn db-spec]
  (->> (query conn [data-query])
       first
       :matrix
       #(.(getArray ^Jdbc4Array %))
       (emap #(or % -1.0))
       matrix)))

(def db-spec {:classname "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname "/localhost:5432/span_vt"
              :user "gjohnson")

(def source-layer (postgis-raster-to-matrix db-spec
                                          "scenic_beauty.chittenden_source" 100 0.15))

(def sink-layer (postgis-raster-to-matrix db-spec
                                           "scenic_beauty.chittenden_sink" 100 0.01))
```

73
Now that all four layers are loaded into our Clojure process, we can run the following code snippet to verify that their dimensions all match. Table 3.9 shows the results of this query.

\[
\text{Table 3.9: Matrix dimensions of raster layers imported from Postgresql to Clojure}
\]

<table>
<thead>
<tr>
<th></th>
<th>Source</th>
<th>Sink</th>
<th>Use</th>
<th>Elevation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows</td>
<td>626</td>
<td>626</td>
<td>626</td>
<td>626</td>
</tr>
<tr>
<td>Cols</td>
<td>444</td>
<td>444</td>
<td>444</td>
<td>444</td>
</tr>
</tbody>
</table>

Since we just downsampled and thresholded these layers, we should now take another look at them to make sure that the post-processed maps look right in Clojure space. Since the nonrival use map is a point layer, it will likely be very difficult to see the highlighted sites without some image preprocessing. To address this issue, we apply a neighborhood copying algorithm to each site, allowing its value to bleed out into its immediately adjacent cells. Note that this is only used to generate the visualization for the use layer and will not affect its values in the remainder of our analysis.
Figure 3.12 shows source values ranging from 0.0 to 0.58 as expected. Figure 3.13 shows that the sink layer also made it through the downsampling and thresholding process quite well with values ranging from 0.0 to 0.09. Unfortunately, because it was a rasterized point dataset, the nonrival use layer did not fare as well as these other two. While there were 317 outdoor recreation sites in our 30 meter resolution map of Chittenden county, Figure 3.14 shows that only 33 of these sites persisted through the downsampling process. Although not ideal, this should still be sufficient to illustrate the flow distribution algorithm presented in the remainder of this chapter. Finally, the elevation layer, shown in Figure 3.15, seems to have preserved much of its variation with values ranging from 22 to 1328 meters above sea level.

3.7.3 COMPUTING THE SCENIC BEAUTY FLOW SURFACE

Now that we have computed the source, sink, and nonrival use layers for scenic beauty, we must turn our attention to the flow surface(s) along which its service medium travels from the landscape to human users. Since many ecosystem services may be transmitted by service media with similar spatio-temporal dynamics, we group them broadly into five general flow categories, each represented by a unique flow surface calculation algorithm.

*In situ Flow* users must be co-located with sources

*Proximal Flow* source effects drop off with distance

*Channeled Flow* media are channeled along paths from sources to users

*Diffusive Flow* like proximal flow with channeling corridors

*Global Flow* all users benefit from all sources
Since the service medium in this case is scenic beauty transmitted by light, it will radiate out in all directions from the source points as far as light may travel. Thus, the limiting factors in determining which sources and sinks affect each user are the maximum distance that each user can see given the clarity of the air from their particular vantage point and the degree to which the immediately surrounding topography may intervene with a user’s sight lines.

This clearly highlights the scenic beauty service as belonging to the proximal flow category. This flow type is characterized by independent, nonrival use of a resource, whose accessibility to or impact on the user attenuates with distance. As a result, no shared flow surface exists for any two non-spatially-colocated users. Instead each user imposes their own flow surface on their immediately surrounding environment, within which they are the only outlet point and all locations within range flow directly towards the user.
For proximal flow services like this one (and also including soundscapes and walking access to open space), this process of creating an inward-facing flow surface is repeated for each user in our study area. An independent SPAN simulation is run over each such flow surface, and all of the results are combined together into a single set of output maps at the end of the analysis.

**Finding All Points in Range**

We begin our flow surface calculations for a proximal flow service by identifying all of the grid cells within range of each user. Given its location, the maximum distance of effect, and the dimensions, cell width, and cell height of the underlying raster grid, the following algorithm returns a map of these in-range cells to each one’s distance from the user.
Figure 3.14: Scenic beauty nonrival use map after downampling and thresholding

(defn in-bounds?  
  "Returns true if the point lies within the bounds [0,rows) by [0,cols)."  
  [rows cols [i j]]  
  (and (>= i 0)  
       (>= j 0)  
       (< i rows)  
       (< j cols)))

(defn neighborhood-points  
  "Returns a vector of the points within radius steps of the passed in point."  
  [point radius]  
  (let [side (+ (* 2 radius) 1)  
         offset (- radius)]  
    (+ point [offset offset] (index-seq-for-shape [side side]))))

(defn scale-by-cell-dims  
  "Returns the passed in point’s coordinates in units of the cell width  
   and cell height."  
  [cell-width cell-height [i j]]  
  [(+ i cell-height) (* j cell-width)])

(defn compute-point-distances-in-range
Figure 3.15: Scenic beauty elevation map after downsampling and thresholding

"Returns a map of neighboring points to their distances from central-point."
[central-point max-distance cell-width cell-height rows cols mask-layer]
(let [scaled-point (scale-by-cell-dims cell-width cell-height central-point)
  max-effect-radius (int (Math/ceil (/ max-distance
    (min cell-width cell-height)))))
(into {}
  (for [point (neighborhood-points central-point max-effect-radius)]
    (if (and (in-bounds? rows cols point)
      (not (neg? (apply mget mask-layer point))))
      (let [dist (distance scaled-point
        (scale-by-cell-dims cell-width
          cell-height
          point))]
        (if (<= dist max-distance)
          [point dist]))))))

To get an intuitive understanding of what this code does, let’s run it using a maximum view distance of 25 kilometers for one of our use locations in the western part of Chittenden county. Figure 3.16 shows the resulting map.
(def example-user [303 116])
(def max-view-distance 25000.0) ;; in meters

(def mask-layer (emap #(if (not (neg? %)) 1.0 0.0) source-layer))

(def point-distances (compute-point-distances-in-range example-user
  max-view-distance
  cell-width
  cell-height
  num-rows
  num-cols
  source-layer))

(let [flow-distances (apply-mask
  (compute-matrix [num-rows num-cols]
    (fn [i j] (point-distances [i j] 0.0))
    mask-layer
    -1.0)]
  (save-matrix-as-png :color 1 -1.0 flow-distances "pics/flow_distances.png"))

\textbf{Figure 3.16: Distance (in meters) to potentially visible points within 25km of a user}
Creating the Inward-Facing Trajectory Surface

Since scenic beauty is transmitted by light to the user’s eyes, we can most directly model its flow by projecting straight lines between the user and each point within the view radius. However, as we are working within a raster model, these lines must be fit to the underlying grid which defines our study area. Compounding this issue tremendously is the fact that our flow surface must contain all possible paths from our user to the the potentially visible points simultaneously in order for the SPAN flow distribution algorithm to function properly.

The approach that we take to resolve this problem contains several geometrical subproblems, which we describe and implement in code throughout the remainder of this section.

The first step in our journey is to assign to each point within range the angle which points directly from it towards the user at the center of the serviceshed. Since all angular coordinate systems require a reference angle, we set due East to be $0^\circ$ and use degrees as our unit of choice throughout this section. Please note that if a cell is due West of the use point, then its trajectory is due East.

The trajectory $\theta(p, u)$ from each in-range point $p_{x,y}$ to the user $u_{x',y'}$ is therefore defined as follows:

$$\theta(p, u) = 180 + \arctan\left(\frac{y - y'}{x - x'}\right)$$

Figure 3.17 shows the result of applying this function to each point within 25km of our example user.

```Clojure
(defn get-trajectory
  "Returns the angle corresponding to the line from pointA to pointB."
  [pointA pointB]
  (let [[dy dx] (- pointB pointA)]
    (+ 180.0 (Math/toDegrees (Math/atan2 dy dx))))
)
```

```Clojure
(defn compute-matrix
  [num-rows num-cols]
  (fn [i j]
    (let [[dy dx] (- example-user [i j])]
      (get-trajectory example-user [i j] 0.0))))
```

```Clojure
(save-matrix-as-png :color 1 -1.0
  (-> (compute-matrix [num-rows num-cols]
    (fn [i j] (if (and (point-distances [i j])
      (not= example-user [i j]))
      (get-trajectory example-user [i j])
      0.0))))
)```
Snapping to the Nearest Compass Direction

Although the trajectory surface does point accurately from each cell toward the user, we cannot traverse a raster grid along arbitrary angles. Only the 8 cardinal and ordinal directions (N, E, S, W, NE, SE, SW, NW) are available options for each step of our flow routing. The following code sets up functions for snapping angles to their nearest compass direction and looking up the corresponding angle, point offset, and numeric code associated with each.

For efficiency, we represent each of these 8 directions with an integer as follows:
(defn nearest-compass-direction
  "Returns a keyword in #{:E :SE :S :SW :W :NW :N :NE} corresponding to the nearest compass direction to angle."
  [angle]
  (condp >= angle
    22.5 :E
    67.5 :SE
    112.5 :S
    157.5 :SW
    202.5 :W
    247.5 :NW
    292.5 :N
    337.5 :NE
    360.0 :E))

(def compass-angle
  "Returns the angle pointing in the corresponding compass direction."
  {:E 360.0 :SE 45.0 :S 90.0 :SW 135.0 :W 180.0 :NW 225.0 :N 270.0 :NE 315.0})

(def compass-offset
  "[[NW N NE]
   [W E]
   [SW S SE]]"
  {:E [ 0 1]
   :NE [-1 1]
   :N  [-1 0]
   :NW [-1 -1]
   :W [ 0 -1]
   :SW [ 1 -1]
   :S  [ 1 0]
   :SE [ 1 1]})

(def flow-direction-code
  "Converts cardinal direction keywords to their numeric values in the flow directions layer."
  {{0 1} 1.0 ; E
  [-1 1] 2.0 ; NE
  [-1 0] 3.0 ; N
  [-1 -1] 4.0 ; NW
  [ 0 -1] 5.0 ; W
  [ 1 -1] 6.0 ; SW}
Applying the snapping function above to the trajectory surface divides the space into 8 wedges. Any point within the same wedge will thus snap to its corresponding compass direction. Figure 3.18 illustrates this partitioning of the serviceshed.

Ordering the Wedges into Lines

Of course, trying to follow the snapped trajectory surface directly will just lead to some very warped sight lines. Instead, we augment this octagonal partitioning with an additional binning by distance from the user.

Starting from the use point, we project a straight line in each of the eight compass directions out to the maximum distance of effect. We then take a step out along one of
Figure 3.18: Trajectories snapped to their nearest compass direction

these lines and project a second line perpendicular to the first. All cells within the same wedge that fall along this second line are labeled with the number of steps taken along the outward line thus far. This procedure is repeated all the way to the outermost cells in each wedge.

The following (lengthy) code block implements this wedge line ordering algorithm.

```clojure
(defn angular-width
  "Returns the number of degrees between theta1 and theta2."
  [theta1 theta2]
  (mod (- theta2 theta1) 360.0))

(defn left-of
  "Returns true if theta1 is counterclockwise from theta2."
  [theta1 theta2]
  (and (not= theta1 theta2)
       (< (angular-width theta1 theta2) 180.0)))

(defn right-of
  "Returns true if theta1 is clockwise from theta2."
  [theta1 theta2]
  (and (not= theta1 theta2)
       (> (angular-width theta1 theta2) 180.0))
```

85
(def corner-offsets
"Offsets to add to the center of a grid cell to reach its four corners."
(mapv #(∗ 0.5 (compass-offset %)) [:NE :SE :SW :NW]))

(defn get-bounding-angles
"Returns the bounding angles for any line originating at use-point
and passing through the point provided as [min max]."
[use-point point]
(when (not= point use-point)
(let [corner-angles (vec
(sor
(for [corner-point (map #(+ point %) corner-offsets)]
(let [[dy dx] (- corner-point use-point)]
(+ 180.0 (Math/toDegrees (Math/atan2 dy dx)))))))]
(if (left-of (corner-angles 0) (corner-angles 3))
[(corner-angles 0) (corner-angles 3)]
(let [[[low-angles high-angles] (split-with #(≤ % 180.0) corner-angles)]
[[apply min high-angles] (apply max low-angles)]))))

(defn wedge-step-offsets
"Returns four point offsets {:straight :left :right :zigzag} for
traversing a wedge on trajectory-matrix."
[use-point obstruction-point diagonal-wedge?]
(let [straight-dir (-> (get-trajectory use-point obstruction-point)
(#(mod (+ 180.0 %) 360.0))
nearest-compass-direction)
straight-angle (compass-angle straight-dir)
left-angle (mod (- straight-angle 90.0) 360.0)
right-angle (mod (+ straight-angle 90.0) 360.0)]
{:straight (compass-offset straight-dir)
:left (compass-offset (nearest-compass-direction left-angle))
:right (compass-offset (nearest-compass-direction right-angle))
:zigzag (when diagonal-wedge?
(compass-offset
case straight-dir
:NE :E
:SE :S
:SW :W
:NW :N))))}

(defn collect-wedge-lines
"Returns a sequence of vectors, each containing a line of points
running from left to right across the wedge hidden behind
obstruction-point when looking out from use-point."
[use-point obstruction-point diagonal-wedge?]
(rows cols cell-width cell-height max-distance]
(let [offsets (wedge-step-offsets use-point obstruction-point diagonal-wedge?)
[left-bounds right-bounds] (get-bounding-angles use-point obstruction-point)
max-steps (int (Math/ceil (/ max-distance (min cell-width cell-height))))]
(take-while seq
(for [steps (range (if diagonal-wedge?
                    (int (* (Math/sqrt 2) max-steps))
                    max-steps))]
  (let [center-point (+ obstruction-point
                   (if diagonal-wedge?
                      (if (even? steps)
                         (* (/ steps 2) (:straight offsets))
                         (+ (:zigzag offsets)
                            (* (/ (dec steps) 2)
                               (:straight offsets)))))
                      (* steps (:straight offsets))))]
  (filterv #(in-bounds? rows cols)
         (concat
          (reverse
           (take-while #(not (left-of (first (get-bounding-angles use-point %))
                                      left-bounds))
           (iterate #(+ (:left offsets) %) center-point)))
          (take-while #(not (right-of (second (get-bounding-angles use-point %))
                                        right-bounds))
           (rest (iterate #(+ (:right offsets) %) center-point))))))))

(defn get-obstruction-points
  "Returns a map of {compass-direction obstruction-point} for the eight points around use-point whose shadows perfectly partition all the cells on a raster grid into 8 wedges."
  [use-point]
  {:N (+ use-point (* 2 (compass-offset :N)))
   :E (+ use-point (* 2 (compass-offset :E)))
   :S (+ use-point (* 2 (compass-offset :S)))
   :W (+ use-point (* 2 (compass-offset :W)))
   :NE (+ use-point (compass-offset :NE))
   :SE (+ use-point (compass-offset :SE))
   :SW (+ use-point (compass-offset :SW))
   :NW (+ use-point (compass-offset :NW))})

(defn create-wedge-line-matrix
  "Assigns each point on a grid to belong to one of 8 compass-oriented wedges centered on use-point and then labels it with its step distance from use-point when traversing the grid in the compass direction associated with its wedge. This produces a matrix with numbers increasing outward from use-point along octagonal contour lines."
  [rows cols cell-width cell-height max-distance use-point point-distances]
  (let [wedge-line-matrix (zero-matrix rows cols)]
    (doseq [[compass-dir obstruction-point] (get-obstruction-points use-point)]
      (let [diagonal-wedge? (contains? #{:NE :SE :SW :NW} compass-dir)
To better visualize the result of this ordering algorithm, we run it over the serviceshed associated with our example user with a 25 kilometer view radius. Figure 3.19 illustrates this newly ordered octagonal view surface.

Applying Trajectories to the Ordered Wedges

Now that we have ordered all of the cells within each wedge by distance from the user, our flow surface begins to take shape. Remember that we are ultimately trying to project sight lines from every point within the serviceshed in towards our user and map them onto the underlying raster grid.

Each sight line which originates from a cell \( p \) on the surface will never pass through any cell \( p' \) with the same or higher wedge line number. Therefore, if we start at the outermost ordered line in each wedge, we can project the sight line from each of its cells on to its next step cell according to the compass direction given in the snapped trajectory surface. We then label each of these upstream cells with their outgoing flow direction.

As the final step, we record the difference between the snapped angle that was chosen for each cell and its original unsnapped trajectory to compensate for the error introduced by this process. After we finish assigning outgoing flow directions to each cell in this line,
we assign to each downstream cell an angular error $\theta_{\text{error}}$, which is the average of all the snapping errors experienced by its immediately upstream cells.

That is, if cells $p_1$ and $p_2$ both flow into cell $p_3$, then cell $p_3$’s angular error $\theta_{\text{error}}(p_3)$ will be set as follows:

$$\theta_{\text{error}}(p_3) = \frac{(\theta(p_1, u) - \theta_{\text{snapped}}(p_1, u)) + (\theta(p_2, u) - \theta_{\text{snapped}}(p_2, u))}{2}$$

When we move inward to the next wedge line, any cells with incoming sight lines add their angular error value to their trajectory prior to snapping it to the nearest compass direction. We then measure the snapping error associated with this adjusted angle and propagate it downstream as before. This procedure repeats recursively all the way down the wedge and is then repeated for every wedge in the serviceshed.
Finally, because none of these sight lines may travel further than their corresponding use point, the use cell is assigned a code indicating that it is a terminal point in the flow surface.

```
(defn downstream-point-by-trajectory
"Returns the next [point angular-offset] pair after taking one step in the nearest compass direction toward use-point adjusted by the current angular-offset."
[use-point point angular-offset]
(let [local-angle (get-trajectory use-point point)]
  (if (not= point use-point)
    (let [adjusted-angle (mod (+ local-angle angular-offset) 360.0)
          adjusted-dir (nearest-compass-direction adjusted-angle)]
      (+ point (compass-offset adjusted-dir))
    [- adjusted-angle (compass-angle adjusted-dir)]))))

(defn compute-proximal-flow-surface
"Assigns each point on a grid to belong to one of 8 compass-oriented wedges centered on use-point and then organizes them in lines perpendicular to the compass direction associated with its wedge. Each cell in such a line is then labeled with a flow direction code that merges its flow path into the cell in the next line nearer to the user that is closest to being on a straight line from the current point to the user. This produces a matrix with flow direction codes in each cell, which point in toward the user along an octagonal flow surface."
[rows cols cell-width cell-height max-distance use-point point-distances]
(let [flow-matrix (zero-matrix rows cols)
      count-matrix (zero-matrix rows cols)
      offset-matrix (zero-matrix rows cols)]
  (doseq [[compass-dir obstruction-point] (get-obstruction-points use-point)]
    (let [diagonal-wedge? (contains? #{:NE :SE :SW :NW} compass-dir)
          wedge-lines (collect-wedge-lines use-point obstruction-point
diagonal-wedge? rows cols
cell-width cell-height max-distance)]
      (doseq [line-points (reverse wedge-lines)
              [up-i up-j :as up-point] line-points]
        (when (point-distances up-point)
          (let [count (mget count-matrix up-i up-j)
                offset (mget offset-matrix up-i up-j)
                [down-i down-j :as down-point] down-offset]
            (downstream-point-by-trajectory use-point up-point
              (if (zero? count)
                0.0
                (/ offset count)))))
          (mset! flow-matrix up-j up-i
direction-code (- down-point up-point))))
          (mset! count-matrix down-i down-j
ing (mget count-matrix down-i down-j))
          (mset! offset-matrix down-i down-j))
```
The final proximal flow surface generated by this algorithm is shown in Figure 3.20. As before, we apply this to our example user with a 25 kilometer maximum view radius.

```clojure
(def flow-directions (compute-proximal-flow-surface num-rows num-cols cell-width cell-height max-view-distance example-user point-distances))
(save-matrix-as-png :color 1 -1.0 (apply-mask flow-directions mask-layer -1.0) "pics/proximal_flow_surface.png")
```

**Testing the Surface Quality**

One might understandably ask how valid is the flow surface generated by this wedge ordering algorithm. To test this, we will project 20 sight lines from randomly selected cells in the serviceshed using two different algorithms. In the first case, we simply use the error correcting recursive trajectory algorithm that was described in the previous section. However, since we are propagating independent lines, we will never need to average the snapping errors due to path intersections. In the second case, we simply follow our new proximal flow surface from the start point to its termination at the user.

```clojure
(let [start-points [[280 276] [312 90] [163 292] [116 248] [445 194] [156 246] [364 171] [343 179] [274 113] [420 53] [350 40] [201 87] [267 207] [375 226] [321 234] [284 69] [198 90] [395 208] [231 239] [378 179]]
  line-points (reduce into #{}))
Figure 3.20: Proximal flow surface for a user with a 25km view radius

```lisp
(for [point start-points]
  (->> [point 0.0]
       (iterate #(.apply downstream-point-by-trajectory example-user %))
       (take-while identity)
       (mapv first)))

surface-points (reduce into #{}
  (for [point start-points]
    (->> point
     (iterate
     #(if-let [offset (flow-offset (apply mget flow-directions %))]
        (+ % offset))
     (take-while identity))))

(save-matrix-as-png :color 1 -1.0
  (-> (compute-matrix
       [num-rows num-cols]
       (fn [i j] (if (line-points [i j]) 1.0 0.0))
       (apply-mask mask-layer -1.0))
    "pics/sample_sight_lines_by_trajectory.png")

(save-matrix-as-png :color 1 -1.0
  (-> (compute-matrix
       [num-rows num-cols]
       "pics/sample_sight_lines_by_trajectory.png")
```
Figure 3.21 shows the results of both test cases side by side.

As we can see, there is very little error introduced by the proximal flow surface algorithm presented in this section. Although slightly less precise than simply projecting independent lines between the user and each point in the serviceshed, our shared flow surface allows us to analyze all flow paths simultaneously with linear $O(n)$ time and space complexity compared to a significantly worse quadratic $O(n^2)$ complexity for the independent line case.

3.7.4 Accounting for Distance Decay and Visibility

In the previous section, we identified all of the cells within the maximum view radius of our user. However, the scenic beauty (or visual blight) in each of these cells will not impact
the user equally due to two spatially non-uniform weighting factors: **distance decay** and **visibility**.

Let $U$ denote the set of all use points. Let $P_u$ denote the set of all points within the maximum distance of effect $d_{\text{max}}$ of user $u \in U$. Let $d(u,p)$ denote the two-dimensional euclidean distance between points $u$ and $p$.

The distance decay associated with each point $p \in P_u$ is a real value $D(u,p) \in [0,1]$, which may vary non-monotonically with increasing $d(u,p)$ but is always 0 for all cells beyond the maximum distance of effect $d_{\text{max}}$. The function which determines this value may be swapped out according to the ecosystem service being analyzed and/or the expertise of the SPAN modeler. For our Chittenden County scenic viewshed case study, we will use the following formula for distance decay.

$$D(u,p) = \begin{cases} \frac{100}{2d(u,p) \tan(22.5^\circ)} & \text{if } u \in U \\ 0 & \text{otherwise} \end{cases}$$

The rationale behind this function proceeds as follows and is illustrated in Figure 3.22. Since we are operating on a raster grid, one can only move (or look) in one of the eight cardinal or ordinal directions. Each such direction would therefore sweep out 1/8 of the $360^\circ$ view space available to any scenic beauty user. Now, imagine that a user is facing directly into a wall which runs forever both to the left and right. As we move this wall away from the user, we reveal the triangular region swept out by the current $45^\circ$ viewing angle. At distance $d(u,p)$, the cross-sectional length of the wall within the viewing region is $2d(u,p) \tan(22.5^\circ)$.

Since each cell is 100 meters wide in our scenic beauty example, we can calculate the fraction of the view space to which it contributes at distance $d(u,p)$ by dividing the cell width by the corresponding wall length. This gives us the distance decay formula above, which weights each cell by its relative contribution compared to other visible cells at the
same distance.

To help visualize this distance decay curve, we will plot it with our working example's maximum view distance of 25 kilometers on semilog axes.

```plaintext
reset
define f(x)=100/(2*x*tan(22.5))
define g(x)=f(x)>1?0:f(x)
plot g(x) with lines notitle
```

*Figure 3.22: Derivation of the scenic beauty distance decay function*
Applying this distance decay formula to each cell within 25 kilometers of our example user generates the decay surface shown in Figure 3.24. Because these decay values drop off so quickly with distance (as evidenced by the previous function plot), we use a logscale color-ramp in this figure.

```clojure
(def distance-decay-coefficient (/ 100.0 (* 2.0 (Math/tan (/ Math/PI 8.0)))))

(defn distance-decay
  [distance]
  (if (pos? distance)
    (let [decay (/ distance-decay-coefficient distance)]
      (if (> decay 1.0) 0.0 0.8 decay)) 0.0))

(defn compute-distance-decay-layer
  "Returns a matrix of weights in [0,1] based on each point's distance from its user."
  [rows cols point-distances]
  (compute-matrix [rows cols]
    (fn [i j] ...)"
(if-let [dist (point-distances [i j])]
  (distance-decay dist) 0.0))}

(def distance-decay-layer (compute-distance-decay-layer num-rows num-cols point-distances))

(save-matrix-as-png :colorlog 1 -1.0
  (apply-mask distance-decay-layer mask-layer -1.0) "pics/distance_decay.png")

Figure 3.24: Viewshed distance decay map for a user with a 25km view radius

Up to this point, we have been working in a flat world with the assumption that our user can see everything within their maximum view radius. To move our model closer to reality, we must now incorporate the effects of topography on the user’s field of view. Starting from the use point, we follow the flow directions in reverse until we reach the maximum distance of effect. At each step, we use the elevation layer to compute the slope between the current
point and the user. If this slope is greater than those preceding it along the same sight line, then it is recorded as the new dominant sight slope for the remainder of that path. Additionally, we use the previous cell’s dominant sight slope to calculate the fraction of the current point’s elevation which is not hidden from the user by intervening topography. This value is stored in the visibility layer to later be used as a weighting factor for source and sink points along with distance decay.

The following plot illustrates a hypothetical topography for a sight line being projected from the user at location (0,0).

```plaintext
reset
set term postscript solid color eps enhanced
set title 'Sight Line Obstructions by Intervening Topography'
set xlabel 'Distance from User (m)'
set xrange [0:10000]
set xtics out nomirror
set ylabel 'Elevation (m)'
unset grid
f(x)=0.5*x*sin(0.5*x)
g(x)=f(x)>0?(x>4000?f(x)+500:f(x)): -0.5*f(x)
h(x)=0.5*x
i(x)=0.583*x
j(x)=x<6000?h(x):i(x)
set key horizontal top left
set style fill solid 1.0 border lt -1
plot j(x) with filledcurves x1 linetype 3 linecolor 5 title 'Obstructed Views', \
g(x) with filledcurves x1 linetype 4 linecolor 6 title 'Hills', \
h(x) with lines linetype 1 linewidth 4 title 'Sight Line for x<6000', \
i(x) with lines linetype 2 linewidth 4 title 'Sight Line for x>=6000'
```

98
The following code implements the sight line algorithm described above to compute the visibility values for all cells within the user’s maximum view radius.

```clojure
(defn on-bounds?
  "Returns true if the point is on an edge row or column."
  [[rows cols [i j]]
   (or (== 0 i)
       (== 0 j)
       (== (dec rows) i)
       (== (dec cols) j)))

(defn downstream-point
  "Returns the neighboring point into which the given point flows."
  [[flow-matrix point]
   (let [flow-direction-code (apply mget flow-matrix point)]
    (if (not= flow-direction-code NO-FLOW-DIRECTION)
     (+ point (flow-offset flow-direction-code))))]

(defn upstream-points
  "Returns a vector of points which flow directly into the given point. If the point is on the bounds of the flow matrix, it will be considered to have no upstream points."
  [[flow-matrix rows cols point]
   (if-not (on-bounds? rows cols point)
    (filterv #(if-let [p (downstream-point flow-matrix %)] 
               p))]
```


(defn depth-first-graph-update!
  "Traverses a graph in depth-first order, applying
  update-node-and-return-successors! to each node in the graph.
  Returns nil."
  [root update-node-and-return-successors!]
  (loop [open-list [root]
      closed-set (transient #{}))
    (when-first [node open-list]
      (if-let [children (seq (update-node-and-return-successors! node))]
        (recur (concat (remove closed-set children) (rest open-list))
            (conj! closed-set node))
        (recur (rest open-list)
            (conj! closed-set node)))))

(defn compute-visibility-layer
  "Returns a matrix of weights in [0,1] representing the percentage
  of each cell's elevation that is not hidden from the user by
  intervening topography."
  [[i j :as use-point] elev-layer point-distances flow-directions rows cols]
  (let [use-elev (mget elev-layer i j)
        visibility-layer (zero-matrix rows cols)]
    (depth-first-graph-update!
      [use-point nil]
      (fn [[[i j :as point] max-slope]]
        (let [new-max-slope
              (if (= point use-point)
                (do (mset! visibility-layer i j 1.0)
                    max-slope)
                (let [elev-gain (- (mget elev-layer i j) use-elev)
                        distance (point-distances point)
                        slope (/ elev-gain distance)]
                  (cond
                    (nil? max-slope) (mset! visibility-layer i j 1.0)
                    (pos? max-slope) (if (> slope max-slope)
                        (mset! visibility-layer i j
                          (- 1.0 (/ max-slope slope))))
                    :otherwise (if (> max slope)
                        (mset! visibility-layer i j
                          (repeat max-slope))))
                  (zipmap (upstream-points flow-directions rows cols point)
                    (repeat new-max-slope)))))
            visibility-layer)))

To aid in our graphical understanding of this visibility algorithm, we will visualize the map generated by running it for our example user with a maximum view radius of 25
kilometers. Figure 3.26 shows the results.

```lisp
(def visibility-layer (compute-visibility-layer example-user
elev-layer
point-distances
flow-directions
num-rows
num-cols))

(save-matrix-as-png :color 1 -1.0
(apply-mask visibility-layer mask-layer -1.0)
"pics/visibility.png")
```

![Viewshed visibility map for a user with a 25km view radius](pics/visibility.png)

*Figure 3.26: Viewshed visibility map for a user with a 25km view radius*

Finally, we can combine the distance decay $D(u, p)$ and visibility $V(u, p)$ values for each cell within the maximum view radius $\text{dist}_{\text{max}}$ to create a map of impact weights $I(u, p)$ as follows:

$$\forall u \in U, \forall p \in P_u : I(u, p) = D(u, p)V(u, p)$$
Recall that the distance decay function we are using scales cells by their relative contributions on a horizontal line. The visibility function we have defined scales cells by their relative contributions on a vertical line. Thus, by multiplying them together in the impact function, we create a scaling coefficient for each cell that describes the fraction of the two-dimensional view field that it fills for each user. The following code generates this map of impact values, which can be seen in Figure 3.27. Note that it also uses a logscale color-ramp.

```
(def impact-layer (* distance-decay-layer visibility-layer))
(save-matrix-as-png :colorlog 1 -1.0
  (apply-mask impact-layer mask-layer -1.0)
  "pics/impact.png")
```

Figure 3.27: Viewshed impact map for a user with a 25km view radius

We have now implemented all the functionality necessary to weight our source and sink
layers according to their potential impact on a single use location. In the next section, we will explore the SPAN flow distribution algorithm, which will finally help us to connect our users to those sources, sinks, and flow pathways that generate their ecosystem services.

3.7.5 Performing the Serviceshed Ordering Phase

For a refresher on the details behind the serviceshed ordering phase of the SPAN flow distribution algorithm, see Section 2.4.1.

Since the scenic beauty service belongs to the proximal flow category, this ordering is recomputed once for each use location, which resides at the center of its radially defined serviceshed. In this and the following two sections, we apply each phase of the SPAN flow distribution algorithm to only one user for ease of explanation. In Section 3.7.10, we then complete the analysis by running all three phases across all users and merging their results.

The following code identifies the outlet points in the serviceshed (in our example case, this is simply our solitary viewer) and then follows the flow directions upstream from those users until the edge of the serviceshed is reached. Along the way, each cell is labeled with the number of steps along the flow surface needed to reach its outlet point.

```clojure
(defn depth-first-graph-ordering
  "Traverses a graph in depth-first order, returning a map of the node values encountered to the number of steps they are from the root node."
  [root successors]
(loop [open-list [root]
       ordered-nodes (transient {root 0})]
  (if (empty? open-list)
    (persistent! ordered-nodes)
    (let [this-node (first open-list)
           next-level (inc (ordered-nodes this-node))
           children (seq (remove ordered-nodes (successors this-node)))
           (recur (concat children (rest open-list))
                   (reduce #(assoc! %1 %2 next-level) ordered-nodes children))]
      (recur (rest open-list)
             ordered-nodes))))

(defn order-upstream-points
  "Returns a map of {tree-depth -> [point1 point2 ... pointN]} for one serviceshed."
  [flow-matrix outlet-point]
  ...)
\[
\text{Figure 3.28 shows the results of running this serviceshed ordering algorithm on our example user with a 25 kilometer view radius.}
\]

```clojure
(let [rows (row-count flow-matrix)
      cols (column-count flow-matrix)
      outlet-distances (depth-first-graph-ordering
                        outlet-point
                        #(upstream-points flow-matrix rows cols %))]
  (group-by outlet-distances (keys outlet-distances))))

(defn has-downstream-user?
  "Returns true if this point has a downstream use point."
  [flow-matrix use-points point]
  (if-let [next-point (downstream-point flow-matrix point)]
    (if (contains? use-points next-point)
      true
      (recur flow-matrix use-points next-point))
    false))

(defn find-serviceshed-outlets
  "Returns all use points with no downstream users."
  [flow-matrix use-points]
  (remove #(has-downstream-user? flow-matrix use-points %) use-points))

(defn order-serviceshed-points
  "Returns a unified map of \{tree-depth -> [point1 point2 ... pointN]\} for all servicesheds."
  [flow-matrix use-points]
  (apply merge-with join
     (for [outlet-point (find-serviceshed-outlets flow-matrix use-points)]
       (order-upstream-points flow-matrix outlet-point outlet-point))))

(def serviceshed-bands (order-serviceshed-points flow-directions
                                                 #{example-user}))

(let [serviceshed-bands-layer (zero-matrix num-rows num-cols)]
  (doseq [band-num (keys serviceshed-bands)]
    [i j] (serviceshed-bands band-num)]
    (mset! serviceshed-bands-layer i j band-num))
  (save-matrix-as-png :color 1 -1.0
    (apply-mask serviceshed-bands-layer mask-layer -1.0)
    "pics/serviceshed_bands.png")
```
3.7.6 Performing the Forward Propagation Phase

For a refresher on the details behind the forward propagation phase of the SPAN flow distribution algorithm, see Section 2.4.2.

The following (rather lengthy) code block implements this forward propagation algorithm, terminating once all the service medium flows have reached their final outlet points. All of the result matrices are then collected in a single hash-map for further analysis in the next section.

```clojure
(require '(clojure.core [reducers :as r]))

(defn sum-over
  "Adds up the results of applying f to each point."
  [points f]
  (reduce + 0.0 (r/map f points)))

(defn propagate-service-medium-downstream
```
"Traverses the serviceshed from the outermost band to its outlet point, propagating the service medium downstream along the flow surface. Along the way, sources increase the flow, sinks and rival users decrease it, and nonrival users record the amount encountered. Returns a map of {:result-name -> matrix} for the theoretical-source, theoretical-sink, theoretical-rival-use, theoretical-nonrival-use, theoretical-flow, upstream-production, upstream-absorption, upstream-consumption, incoming-flow, possible-sink, actual-flow, actual-nonrival-use, actual-rival-use, outgoing-flow, and outlet-points."

(let [theoretical-source theoretical-sink theoretical-rival-use theoretical-nonrival-use flow-matrix serviceshed-bands]
  (let [rows (row-count flow-matrix)
        cols (column-count flow-matrix)
        upstream-production (zero-matrix rows cols)
        upstream-absorption (zero-matrix rows cols)
        upstream-consumption (zero-matrix rows cols)
        incoming-flow (zero-matrix rows cols)
        possible-sink (zero-matrix rows cols)
        actual-flow (zero-matrix rows cols)
        actual-rival-use (zero-matrix rows cols)
        outgoing-flow (zero-matrix rows cols)]
    (doseq [band-num (range (apply max (keys serviceshed-bands)) -1 -1)
            [i j :as point] (serviceshed-bands band-num)]
      (let [local-upstream-points (upstream-points flow-matrix rows cols point)]
        (when (seq local-upstream-points)
          (mset! upstream-production i j
            (sum-over local-upstream-points
              (fn [[i j]] (+ (mget upstream-production i j)
                (mget theoretical-source i j))))))
        (when (pos? (mget upstream-production i j))
          (mset! upstream-absorption i j
            (sum-over local-upstream-points
              (fn [[i j]] (+ (mget upstream-absorption i j)
                (mget possible-sink i j))))))
        (mset! upstream-consumption i j
          (sum-over local-upstream-points
            (fn [[i j]] (+ (mget upstream-consumption i j)
              (mget actual-rival-use i j))))))
      (mset! incoming-flow i j
        (sum-over local-upstream-points
          (fn [[i j]] (mget outgoing-flow i j))))))
    (let [service-medium (+ (mget incoming-flow i j)
                              (mget theoretical-source i j))]
      (when (pos? service-medium)
        (let [possible-sink' (min service-medium (mget theoretical-sink i j))
              actual-flow' (- service-medium possible-sink')
              actual-rival-use' (min actual-flow'
                                 (mget theoretical-rival-use i j))
              outgoing-flow' (- actual-flow' actual-rival-use')]
          (mset! possible-sink i j possible-sink')
          (mset! actual-flow i j actual-flow'))))
  106)
For demonstration purposes, we will run this code now on the serviceshed associated with our example user with a 25 kilometer view radius. In order to ensure that the -1.0 values we have used to indicate nodata in the source and sink layers do not interfere with our calculations, we first set them to 0.0 prior to running this algorithm.
Note that for the scenic beauty service (and all other services with proximal flow dynamics), we first multiply the source and sink layers by the impact layer calculated in Section 3.7.4 prior to forward propagation. These impact-weighted source and sink layers can be seen in Figures 3.29 and 3.30 respectively. Because these impact weights drop off so quickly with distance from the user, we again use a \textit{logscale color-ramp} for these two figures to enable us to see more variability in the results.

\begin{figure}[h!]
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{Impact-weighted source values for a user with a 25km view radius}
\end{figure}

Once the forward simulation is complete, we now have all the information we need to generate most of the inaccessible, possible, blocked, and captured SPAN layers from their theoretical and actual counterparts. These relationships are defined in detail in Section 2.5.

The following code derives these additional results (see Section 2.4.2) from the outputs of our forward propagation simulation to increase the number of final SPAN output layers to 19.
Figure 3.30: Impact-weighted sink values for a user with a 25km view radius

(defn derive-intermediate-span-outputs
"Computes the inaccessible, possible, blocked, and captured matrices from the forward simulation results and appends these 13 new matrices onto intermediate-span-outputs."
{:keys [theoretical-sink theoretical-rival-use theoretical-nonrival-use theoretical-flow upstream-absorption upstream-consumption possible-sink actual-flow actual-rival-use]
  :as intermediate-span-outputs}
(let [blocked-flow (+ upstream-absorption possible-sink)
  captured-flow upstream-consumption
  possible-flow (+ actual-flow blocked-flow captured-flow)
  possible-rival-use (emap min possible-flow theoretical-rival-use)
  inaccessible-sink (- theoretical-sink possible-sink)
  inaccessible-rival-use (- theoretical-rival-use possible-rival-use)
  inaccessible-nonrival-use (emap (fn [f u] (if (pos? f) 0.0 u)) possible-flow theoretical-nonrival-use)
  inaccessible-flow (emap (fn [f d] (if (pos? f) 0.0 d)) possible-flow theoretical-flow)
  possible-rival-use possible-rival-use
  possible-nonrival-use (emap (fn [f u] (if (pos? u) f 0.0)) possible-flow theoretical-nonrival-use)
)
This concludes the forward propagation phase of the SPAN flow distribution algorithm. Although we just introduced a great many new map layers, we will wait until all of the SPAN outputs are produced in the following section before reviewing them as a whole. When presented in this way rather than individually, the relationships between these maps will hopefully become immediately apparent.

3.7.7 Performing the Backpropagation Phase

For a refresher on the details behind the backpropagation phase of the SPAN flow distribution algorithm, see Section 2.4.3.

The following code block implements the source weight distribution algorithm, which is used to compute the Blocked Source, Captured Source, and Actual Source maps.

```
(defn distribute-source-impacts!
  "Divide the possible-sink, actual-rival-use, and actual-nonrival-use
```

```
  (emap (fn [f u] (if (pos? u) f 0.0))
    blocked-flow theoretical-nonrival-use)
```

```
  (:captured-source-impacts! (emap (fn [f u] (if (pos? u) f 0.0))
    captured-flow theoretical-nonrival-use))
```
values from cell \([i, j]\) between the local cell based on its theoretical-source value and the parent-cells based on their outgoing-flow values.*

\([i, j]\) parent-cells blocked-source captured-source actual-source

{:keys [incoming-flow theoretical-source possible-sink
actual-rival-use actual-nonrival-use outgoing-flow]}

{:keys [downstream-possible-sink downstream-actual-rival-use
downstream-actual-nonrival-use]}

(let
[local-source (mget theoretical-source i j)
local-sink (mget possible-sink i j)
local-nonrival-use (mget actual-nonrival-use i j)
actual-total (+ local-source (mget incoming-flow i j))
affected-sink (if (pos? local-nonrival-use)
local-sink
(+ local-sink downstream-possible-sink))
affected-rival-use (+ (mget actual-rival-use i j)
downstream-actual-rival-use)
affected-rival-use' (if (pos? local-nonrival-use)
affected-rival-use
downstream-actual-rival-use')
affected-nonrival-use (if (pos? local-nonrival-use)
local-nonrival-use
downstream-actual-nonrival-use)]

(when (pos? local-source)
(let
[local-weight (/ local-source actual-total)]
(mset! blocked-source i j (* local-weight affected-sink))
(mset! captured-source i j (* local-weight affected-rival-use))
(mset! actual-source i j (* local-weight
(+ affected-nonrival-use
(- affected-rival-use affected-rival-use')))])

(for [[i j] parent-cells]
(let [upstream-weight (if (zero? actual-total)
0.0
(/ (mget outgoing-flow i j) actual-total))]
{:downstream-possible-sink (* upstream-weight affected-sink)
:downstream-actual-rival-use (* upstream-weight affected-rival-use)
:downstream-actual-rival-use' (* upstream-weight
affected-rival-use')
:downstream-actual-nonrival-use (* upstream-weight
affected-nonrival-use))})

This code block implements the sink weight distribution algorithm, which is used to compute the Captured Sink and Actual Sink maps.

(defun distribute-sink-impacts!
"Divide the blocked-rival-use and blocked-nonrival-use values from cell \([i, j]\) between the local cell based on its possible-sink value

111
and the parent-cells based on their blocked-flow values."
[[i j] parent-cells captured-sink actual-sink
{:keys [possible-sink blocked-flow blocked-rival-use
    blocked-nonrival-use]}
{:keys [downstream-blocked-rival-use downstream-blocked-nonrival-use]]
(let [local-sink (mget possible-sink i j)
    blocked-total (+ local-sink
        (sum-over parent-cells
            #(apply mget blocked-flow %))
    affected-rival-use (max (mget blocked-rival-use i j)
        downstream-blocked-rival-use)
    affected-nonrival-use (max (mget blocked-nonrival-use i j)
        downstream-blocked-nonrival-use)]
(when (pos? local-sink)
    (let [local-weight (/ local-sink blocked-total)
        affected-rival-use (max (mget blocked-rival-use i j)
            downstream-blocked-rival-use)
        affected-nonrival-use (max (mget blocked-nonrival-use i j)
            downstream-blocked-nonrival-use)]
        (let [local-weight (/ local-sink blocked-total)]
            (mset! captured-sink i j (* local-weight affected-rival-use))
            (mset! actual-sink i j (* local-weight
                (max affected-rival-use
                    affected-nonrival-use))))))
(for [[i j] parent-cells]
    (let [upstream-weight (if (zero? blocked-total)
        0.0
        (/ (mget blocked-flow i j) blocked-total))
        {:downstream-blocked-rival-use (* upstream-weight
            affected-rival-use)
        :downstream-blocked-nonrival-use (* upstream-weight
            affected-nonrival-use)]]))

The next code block traverses each outlet point’s serviceshed from bottom to top, performing the source and sink impact distribution calculations defined above. Upon completion, it derives the Inaccessible Source, Possible Source, and Blocked Sink layers from the results produced by this backpropagation step. Finally, the 8 new maps generated here are added to the hash-map of results produced during the forward propagation phase.

(defn backpropagate-source-and-sink-impacts-upstream
  "Traverses each outlet point’s serviceshed from bottom to top, distributing the service medium impacts on each cell to its upstream sources and sinks according to their relative flow contributions."
  [:keys [theoretical-source theoretical-flow possible-sink
      possible-flow outlet-points]
     :as intermediate-span-outputs]
  (let [rows (row-count theoretical-source)
        cols (column-count theoretical-source)
        blocked-source (zero-matrix rows cols)
        captured-source (zero-matrix rows cols)
        actual-source (zero-matrix rows cols)]
    ...)
To conclude this section, we will run this final phase of our SPAN flow distribution algorithm on our example user in Chittenden County with a maximum view radius of 25 kilometers.

```
(def backpropagation-results
  (-> forward-propagation-results
      derive-intermediate-span-outputs
      backpropagate-source-and-sink-impacts-upstream))
```
3.7.8 Cleaning Up the Output Maps

At this point, we have computed all 25 of the SPAN output maps. However, some intermediate results still need to be discarded from the result set, and the impact-weighted source and sink maps need to be replaced with their original input values. Finally, floating point rounding errors may have made some values that should be zero appear to be extremely small non-zero values.

```clojure
(defn discard-intermediate-results
  "Removes map layers left over from the SPAN flow distribution algorithm which are not part of the defined SPAN result set."
  [backpropagation-results]
  (dissoc backpropagation-results
    :upstream-production
    :upstream-absorption
    :upstream-consumption
    :incoming-flow
    :outgoing-flow
    :outlet-points))

(defn replace-impact-weighted-layers
  "Replaces the impact-weighted theoretical-source and theoretical-sink maps with their original unweighted values. Recalculates inaccessible-source and inaccessible-sink using the corrected values."
  [theoretical-source theoretical-sink
   {:keys [possible-source possible-sink] :as span-outputs}]
  (assoc span-outputs
    :theoretical-source theoretical-source
    :theoretical-sink theoretical-sink
    :inaccessible-source (- theoretical-source possible-source)
    :inaccessible-sink (- theoretical-sink possible-sink)))

(defn zero-within!
  "Sets values in matrix to 0.0 if they fall between epsilon and -epsilon."
  [epsilon matrix]
  (emap! #(if (and (< % epsilon)
                   (> % (- epsilon)))
    0.0
    %)
    matrix))

(defn fix-floating-point-rounding-errors!
  "Sets all extremely small values in the SPAN output matrices to 0.0."
  [span-outputs]
  (persistent!
    (reduce-kv (fn [acc label matrix]
                (assoc! acc label (zero-within! 1E-6 matrix)))
    114

After running these cleanup routines against the results of our backpropagation phase (and including the input maps for comparison), we will finally have all 30 SPAN output maps for our example scenic beauty user. These results are shown and analyzed in the following section.

```clojure
(def span-outputs
  (->> backpropagation-results
discard-intermediate-results
  (replace-impact-weighted-layers source-layer' sink-layer')
  fix-floating-point-rounding-errors!)

doseq [[label matrix] span-outputs]
  (when (some pos? (eseq matrix))
    (let [label' (name label)
      matrix' (if (.contains label' "use")
        (-> (apply-mask matrix mask-layer -1.0)
            (bleed-matrix 2 -1.0 pos?)
            (apply-mask matrix mask-layer -1.0))
      color-ramp (if (or (.contains label' "possible")
                          (.contains label' "blocked")
                          (.contains label' "actual"))
        :colorlog
        :color]
      (save-matrix-as-png color-ramp 1 -1.0 matrix'
        (str "pics/" label' "_single.png")))))

3.7.9 REVIEWING OUR SINGLE SERVICESHED RESULTS

The best way to understand the SPAN results for our scenic beauty user is to compare the theoretical, inaccessible, possible, blocked, and actual versions of each input map (e.g., source, sink, nonrival use, and flow) side by side. Remember, that ultimately what the SPAN flow distribution algorithm does is partition the input values into bins based on their role in delivering the service medium to the chosen users. For a refresher on the mathematical relationships between the different categories of output maps, see Section 2.5.

Please note that for the remainder of this section, all maps labeled Possible,
Blocked, or Actual are shown with logscale color-ramps in order to allow us to visualize the relatively minor variations between cells more easily.

Let’s begin with a review of the source layer definitions. The Theoretical Source map is the input layer that we initially extracted from PostGIS. Each cell contains the service medium quantity expected to be produced in that location during the simulation. We can split this total amount into two pieces: that which has a path to users (Possible Source) and that which lacks a path to users (Inaccessible Source). These outputs are shown in Figure 3.31.

![Figure 3.31: SPAN source results for a user with a 25km view radius (part 1)](image)

Going further, we can split the Possible Source values into the amount that is absorbed by sinks (Blocked Source) and the amount that reaches and affects users (Actual Source). These layers are shown together in Figure 3.32.

Next, we turn our attention to the sink layers. The Theoretical Sink map is another input layer that we originally extracted from PostGIS. It records the total absorption capacity of...
each cell on the landscape in units of the service medium. Splitting it into two pieces, we get the amount that each cell absorbs during the simulation (Possible Sink) and the amount of absorption capacity remaining when it is finished (Inaccessible Sink). The reason that possible is used here rather than actual is because the amount absorbed by any cell is the maximum amount which it may prevent from affecting a downstream user but not necessarily the true amount prevented. Figure 3.33 shows these sink maps.

As with sources, we may split the Possible Sink values into two parts: the amount absorbed which affects downstream users (Actual Sink) and the remaining amount which has no effect on them (Blocked Sink). In the case of scenic beauty, we have only nonrival users on the landscape, and each one will be affected by all changes to its incoming flow. Since Blocked Sink may only be positive in the presence of downstream rival users (due to the upper limit on flow impacts imposed by their demand value), the Blocked Sink layer should be zero everywhere for this ecosystem service. Figure 3.34 omits this empty Blocked
Sink map for simplicity. Make sure to notice that the Possible Sink and Actual Sink layers are identical as a result.

The nonrival use maps are by far the most simple to read. The Theoretical Nonrival Use map for our single serviceshed example contains only a single point with a 1 in it to indicate the location of our nonrival user. The Inaccessible Nonrival Use map would have a 1 in the same cell if this user receives no scenic beauty service from its serviceshed. As this is not the case for our example user, we omit this empty layer from Figure 3.35 for simplicity. Finally, the Possible Nonrival Use map contains the amount of the service medium which might reach the user if no sinks were present in its serviceshed.

Breaking the Possible Nonrival Use map into its subcomponents, we get the amount of the service medium which cannot reach the user because of intervening sinks (Blocked Nonrival Use) and the amount which finally reaches the user given the upstream interaction between source and sinks (Actual Nonrival Use). These are shown in Figure 3.36.
At last, we come to the flow layers. The Theoretical Flow map illustrates the directions that the service medium will travel if it is produced anywhere within the serviceshed. The Possible Flow map shows the service medium amount which would travel across this flow surface from sources to users if no sinks intervened. For our scenic beauty example, only a small portion of the theoretical flow surface is utilized because our nonrival user can only see a relatively small number of sources. Finally, the Inaccessible Flow map shows the flow surface depicted in the Theoretical Flow layer with all of the cells with a positive Possible Flow value masked out. This shows all the cells which do not lie on a path between a source and a user. Figure 3.37 shows these flow maps.

As with all the other maps shown in this section, we may split the Possible Flow map into its subcomponents as well. The service medium amount which passes through each cell on its way from sources to users is depicted in the Actual Flow map, and the amount which might have continued flowing without the intervention of sinks is shown in the Blocked Flow
Figure 3.35: SPAN nonrival use results for a user with a 25km view radius (part 1)

map. It is worth noting that for detrimental service media, the Blocked Flow map shows the amount of protection from this service medium that each cell receives from upstream sinks. This final set of maps can be seen in Figure 3.38.

This concludes our analysis of our single serviceshed scenic beauty example. In the next section, we will run the complete SPAN algorithm for all nonrival users on the landscape and merge their results into one final set of output maps.

3.7.10 Merging the Servicesheds of All Users

Now that we have seen a full SPAN analysis for a single user, we need to repeat this process for all of the users in our Chittenden County study area. The following code block runs this procedure once for each user, merging the result maps by summing. Note that Theoretical Flow and Inaccessible Flow lack a clear meaning when merged in this way, so these two
maps will be discarded from the final outputs.

```
(defn distribute-proximal-flow-for-single-user
  "Runs a SPAN proximal flow analysis for use-point, returning the 30 SPAN output maps for its serviceshed."
  [[i j :as use-point] rows cols cell-width cell-height max-distance
   source-matrix sink-matrix elev-matrix]
  (println "Use Point:" use-point)
  (let [point-distances (compute-point-distances-in-range use-point
    max-distance
    cell-width
    cell-height
    rows
    cols
    source-matrix)
    flow-directions (compute-proximal-flow-surface rows cols
      cell-width
      cell-height
      max-distance
      use-point point-distances)
    distance-decay (compute-distance-decay-layer rows
      cols
      point-distances)
    visibility (compute-visibility-layer use-point
      elev-matrix
      point-distances)
```
(a) Theoretical Flow (b) Inaccessible Flow (c) Possible Flow

Figure 3.37: SPAN flow results for a user with a 25km view radius (part 1)

```clojure
(defn points-where
  "Returns a set of the [i j] coordinates for each point in matrix which satisfies pred?."
  [pred? matrix]
  (into #{})
```

```clojure
(flow-directions rows cols)
(impact-matrix (* distance-decay visibility))
(source-matrix' (emap #(if (pos? %) % 0.0) source-matrix))
(sink-matrix' (emap #(if (pos? %) % 0.0) sink-matrix))
(->> (order-serviceshed-points flow-directions #{use-point}))
(propagate-service-medium-downstream (* impact-matrix source-matrix'))
(* impact-matrix sink-matrix')
(zero-matrix rows cols)
(doto (zero-matrix rows cols)
  (mset! i j 1.0))
(flow-directions)
(derive-intermediate-span-outputs)
(backpropagate-source-and-sink-impacts-upstream)
(discard-intermediate-results)
(replace-impact-weighted-layers source-matrix' sink-matrix')
(fix-floating-point-rounding-errors!)))
```
Figure 3.38: SPAN flow results for a user with a 25km view radius (part 2)

```clojure
(defn combine-span-results
  ([] {})
  ([r1] r1)
  ([r1 r2] (merge-with += r1 r2)))

(defn distribute-proximal-flow
  "Runs a SPAN proximal flow analysis for each user in nonrival-use-matrix and combines the result matrices from multiple runs by summing. Note: Theoretical Flow and Inaccessible Flow are omitted in proximal flow simulations, so only 28/30 results will be returned."
  [source-matrix sink-matrix nonrival-use-matrix elev-matrix
   rows cols cell-width cell-height max-distance]
  (let [use-points (points-where pos? nonrival-use-matrix)]
    (println "Analyzing" (count use-points) "use points...")
    (#(dissoc % :theoretical-flow :inaccessible-flow))))
```
After running the code above, we now have a new set of SPAN result maps that parallel those shown in Section 3.7.9. In the remainder of this section, we will examine what they can tell us about our Chittenden County case study. Please note that all maps labeled Possible, Blocked, or Actual are shown with logscale color-ramps in order to allow us to visualize the relatively minor variations between cells more easily.

Let’s begin with the source maps as before. The first thing to notice is that in Figure 3.39 the Theoretical Source values have been multiplied by the number of users $|U|$ on the landscape. This makes sense for services with proximal flow dynamics since each source point has the potential to affect each user exactly once, and the service medium (information) need not be conserved between users.

Next, we can see from the Possible Source map that although Chittenden County has no shortage of source cells, only a tiny fraction of them are visible to the 33 outdoor recreation sites which we selected as our nonrival users. This highlights one of the main purposes of
the SPAN model: to separate out locations which actually provide ecosystem services to people from those which only have the potential to provide them.

The following code snippet calculates what percentage of the overall scenic beauty available in Chittenden County (Theoretical Source) is either visible to users (Possible Source) or hidden from them (Inaccessible Source):

```lisp
(let [theoretical-source (esum (:theoretical-source proximal-flow-results))
      inaccessible-source (esum (:inaccessible-source proximal-flow-results))
      possible-source (esum (:possible-source proximal-flow-results))]
  (list 'hline
       (list "Percentage of Scenic Beauty Visible to Users"
            (format "%.4f" (/ possible-source theoretical-source)))
       'hline
       (list "Percentage of Scenic Beauty Hidden from Users"
            (format "%.4f" (/ inaccessible-source theoretical-source)))
       'hline))
```

<table>
<thead>
<tr>
<th>Percentage of Scenic Beauty Visible to Users</th>
<th>0.0003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of Scenic Beauty Hidden from Users</td>
<td>0.9997</td>
</tr>
</tbody>
</table>
Going further, we can see from the Blocked Source map in Figure 3.40 that our users are most definitely not all enjoying unspoiled views. The value in each Blocked Source cell tells us how much of its visible scenic beauty (Possible Source) is depreciated because sinks (i.e., visual blight) interfere with their lines of sight to users.

![Possible Source](image1)

![Blocked Source](image2)

![Actual Source](image3)

**Figure 3.40: SPAN source results for all outdoor recreation sites (part 2)**

The following code snippet calculates the percentage of the visible scenic beauty (Possible Source) that is lost to sinks (Blocked Source) and the remaining amount which directly benefits users (Actual Source):

```scheme
(let [possible-source (esum (:possible-source proximal-flow-results))
      blocked-source (esum (:blocked-source proximal-flow-results))
      actual-source (esum (:actual-source proximal-flow-results))]
  (list 'hline
        (list "Percentage of Visible Scenic Beauty Lost to Visual Blight"
              (format "%.2f" (/ blocked-source possible-source)))
        'hline
        (list "Percentage of Visible Scenic Beauty Available to Users"
              (format "%.2f" (/ actual-source possible-source)))
        'hline))
```

126
Next, we turn our attention to the sink maps shown in Figure 3.41. As with the sources, only a tiny fraction of the visual blight in Chittenden County (Theoretical Sink) is visible to our outdoor recreation sites (Possible Sink). Also in parallel with the source maps above, we should notice that the Theoretical Sink values have been multiplied by the number of users $|\mathcal{U}|$ on the landscape. The reasoning here is that once again each sink cell may interfere with each user’s view at most once.

The following code snippet calculates the percentage of the overall visual blight present in Chittenden County (Theoretical Sink) that is visible to users (Possible Sink):

```scheme
(let [theoretical-sink (esum (:theoretical-sink proximal-flow-results))
      inaccessible-sink (esum (:inaccessible-sink proximal-flow-results))
      possible-sink (esum (:possible-sink proximal-flow-results))]
  (list 'hline
    127)
```

*Figure 3.41: SPAN sink results for all outdoor recreation sites*
As discussed in Section 3.7.9, the lack of rival users on our landscape means that the Blocked Sink value will always be zero everywhere. Therefore, the Actual Sink map is identical to the Possible Sink map in this case.

This brings us to the nonrival use maps in Figure 3.42. As expected, Theoretical Nonrival Use simply shows a 1 in each cell containing an outdoor recreation site. Inaccessible Nonrival Use would then place a 1 at each of these sites that lacks a view of any scenic beauty sources. However, in our Chittenden county example, all values in this map are zero, indicating that all of our users benefit from scenic views. Finally, the Possible Nonrival Use map shows those sites which do have views of scenic beauty sources and labels each one with the sum of all the source values visible to it.

The following code snippet counts the number of outdoor recreation sites (out of 33 total) with scenic views (Possible Nonrival Use) and without scenic views (Inaccessible Nonrival Use):

```
(let [view-sites (->> (:possible-nonrival-use proximal-flow-results) (points-where pos?) count) non-view-sites (->> (:inaccessible-nonrival-use proximal-flow-results) (points-where pos?) count)]

(list "Sites with Views of Scenic Beauty Sources" view-sites 'hline
(list "Sites without Views of Scenic Beauty Sources" non-view-sites 'hline))
```

| Sites with Views of Scenic Beauty Sources | 33 |
| Sites without Views of Scenic Beauty Sources | 0 |
Digging deeper into these nonrival use values, we see a set of maps in Figure 3.43 that very closely parallel the meanings behind the Possible Source, Blocked Source, and Actual Source maps shown above. Whereas the Possible Source map shows the amount of scenic beauty generated in each source cell that is visible to all users, the Possible Nonrival Use map shows the total amount of scenic beauty that is visible from each nonrival use cell. Similarly, the Actual Source map shows how much scenic beauty each source point provides to all users after sinks depreciate its value, and the Actual Nonrival Use map shows how much scenic beauty is available to each user after sinks reduce its value. In both cases, the Blocked map shows the difference between the Possible and Actual values. In our scenic beauty example, the Blocked Nonrival Use map shows how much scenic beauty depreciation each outdoor recreation site experiences due to visible visual blight.
The following code snippet calculates the percentage of the scenic beauty visible to users (Possible Nonrival Use) that is depreciated by sinks (Blocked Nonrival Use) and the remaining percentage which directly benefits users (Actual Nonrival Use):

```scheme
(let
  [possible-use (esum (:possible-nonrival-use proximal-flow-results))
   blocked-use (esum (:blocked-nonrival-use proximal-flow-results))
   actual-use (esum (:actual-nonrival-use proximal-flow-results))]
  (list 'hline
    (list "Percentage of Visible Scenic Beauty Lost to Visual Blight"
      (format "%.2f" (/ blocked-use possible-use)))
    'hline
    (list "Percentage of Visible Scenic Beauty Available to Users"
      (format "%.2f" (/ actual-use possible-use)))
    'hline))
```

<table>
<thead>
<tr>
<th>Percentage of Visible Scenic Beauty Lost to Visual Blight</th>
<th>0.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of Visible Scenic Beauty Available to Users</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Unsurprisingly, this is the same distribution shown with the source maps above.
Finally, we reach the flow maps shown in Figure 3.44. As we mentioned at the beginning of this section, the Theoretical Flow and Inaccessible Flow maps (which contain numerically encoded cardinal directions) lack a clear meaning when their values are summed across all users. Thus, for ecosystem services with proximal flow dynamics, these two layers can only be defined for a single user but not for any combination of users.

Instead, we must turn our attention to the Possible Flow map, which along with the Blocked Flow and Actual Flow maps is probably the most visually interesting of all the SPAN outputs. Here we see the amount of scenic beauty (our service medium) which passes through each cell on the landscape. Because our flow paths are lines of sight, these maps necessarily show view lines radiating out from each user to its visible sources like the spokes of a bicycle wheel. The brightness of each line indicates the visible scenic beauty value of its corresponding source point. Furthermore, wherever sight lines cross or converge, they become brighter due to the additive nature of service medium flows.

Figure 3.44: SPAN flow results for all outdoor recreation sites
The key to understanding these three maps is to remember that the Possible Flow map shows the sight line values without taking sinks into account, whereas the Actual Flow map does subtract sink effects from its sight lines. The Blocked Flow map shows the difference between these two layers, which is quite an interesting dataset in itself.

Each sight line in the Blocked Flow map originates from a sink and flows into a nonrival user. The value along this flow path may then be read in several different (but equivalent) ways:

1. Additional flow that would pass through cells in the absence of sinks.
2. A reduction in flow strength due to upstream sinks.
3. Protection received by cells downstream of sinks.

Taking all of these results into account, we can see that an ecosystem service assessment that ignores flow path connectivity information may be prone to significantly overestimating the value of scenic beauty to outdoor recreation sites in Chittenden County, VT. In fact, due to Vermont’s hilly topography, only a tiny fraction of the sources and sinks we calculated in Sections 3.4 and 3.5 are visible to these potential scenic beauty users. The SPAN Actual Source and Actual Sink maps show how much each location actually impacts users for better or worse, and it is these values that we should use when considering how to assign ecosystem service values to the landscape.

If we were in a position to address visual blight issues for these outdoor recreation sites, the Blocked Nonrival Use map immediately illustrates which use points are more affected by sinks than others, which could help in prioritizing management actions and resources. Not to be overlooked, the Actual Nonrival Use map shows us which sites have the best scenic views right now, the value of which to recreational tourists should be patently obvious.

Finally, the Actual Flow map shows us not only which sites have the best views, but in which direction those scenic vistas can be seen. Complementing this, if the goal is not necessarily to see the best views in the county but rather to simply avoid views of visual
blight during a recreational outing, the Blocked Flow map shows which sites have the most depreciated views and in which direction the offending visual blight lies. Of course, this information is not only useful to the recreational tourist but also to the land manager, who after prioritizing a site for improvement using the Blocked Use map can now use the Blocked Flow map to identify the sinks which need to be hidden or removed from view most urgently.

With that, we conclude our SPAN scenic beauty ecosystem service assessment for Chittenden County, VT. In the next and final chapter, we will discuss how the SPAN modeling framework may be used more broadly in the context of land management scenarios. It is our sincere hope that this demonstration chapter has helped to bring the abstractions presented in the previous chapter to life and has provided our readers with the necessary tools and inspiration to create their own SPAN assessments for those landscapes and ecosystem services which matter most to them.
CHAPTER 4

DISCUSSION AND CONCLUSIONS

At this point, we have been introduced to the SPAN framework’s terminology, conceptual models, and algorithms in detail. We have also seen them implemented in software and applied to an assessment of the scenic beauty ecosystem service in Chittenden County, Vermont. Now, in this chapter we will finally explore the implications of their results for influencing land planning and policy development. However, before we address this subject, we must first take a moment to explore the computational complexity (and hence scalability) of the SPAN approach since a mathematically correct but inefficient system is unlikely to be of much value in real world applications.

4.1 Performance Considerations

To be truly useful in land management decision making, a SPAN ESAV assessment must often be run over relatively large areas at high resolution. Just how large and how high will, of course, be determined by the questions being asked, but in all cases these choices will determine the most crucial factor affecting a SPAN model’s computational complexity: cell count. Fortunately, for all but the proximal flow services, all SPAN algorithms are designed with linear $O(n)$ time and space requirements, where $n$ is the number of cells in the study.
Consider the creation of the initial source, sink, rival use, and nonrival use layers as described in Section 2.2. As these maps are meant to capture local estimates of production, absorption, and consumption, they will generally be computed with functions that operate only on a single cell (or a neighborhood of cells) at a time, thus rendering them $O(n)$. Note that this should remain true regardless of the choice of deterministic or probabilistic value systems.

When constructing a flow surface (see Section 2.3), in situ and global flow services naturally take constant time and memory (specifically none), and channeled flow surfaces are usually created with neighborhood-based calculations (again $O(n)$). Although it may seem less obvious, proximal flow surfaces may also be created with linear time and memory. This is because the only fundamental difference between each user’s flow surface is its origin and radius (see Section 2.3.1 for more information).

Thus, to achieve linear time and memory usage for the proximal flow case, we can create one surface with origin $(0,0)$ and radius equal to the maximum radius associated with any user. To look up the flow direction at cell $(i,j)$ on the flow surface associated with the user in cell $(x,y)$, we use these steps:

1. Check whether the euclidean distance between $(x,y)$ and $(i,j)$ is less than or equal to the maximum range of effect associated with the user at $(x,y)$. If so, proceed to Step 2. If not, then $(i,j)$ is not part of this flow surface and thus has no assigned flow direction.

2. Look up the flow direction at cell $(i - x, j - y)$ in the shared flow surface with origin $(0,0)$. This is the flow direction for cell $(i,j)$.

This approach transforms an otherwise quadratic $O(n^2)$ operation back into a linear one. By carrying this idea over to the diffusive flow case, we can similarly create one proximal
surface and reuse it for each source point. The source field merging operation is already linear, and so is the channel burning step. Combined together as they are in sequence, they once again yield a linear algorithm in time and space.

Next, we move on to the SPAN flow distribution algorithm proper. In the serviceshed ordering step (Section 2.4.1), the directed acyclic graph defined by the flow surface is traversed from users to their most upstream cells. If one were to naively run an independent graph traversal for each use point, then this step would unfortunately be quadratic. However, in the channeled and diffusive flow cases, we can be much more clever about pruning this search space.

First, we select one use cell at random and walk upstream from it until the edge of the flow surface is reached. Each cell traversed in this way is then labeled with its stepwise distance to the use cell. Next, we select a different use cell at random from those remaining and again perform the upstream graph walking procedure. If any previously labeled cells are reached in this way (a label collision), their labels are recorded and the cells upstream of them are not explored.

Once the second use cell’s graph traversal is complete, we check to see if any label collisions were recorded. If not, we repeat this procedure for the next remaining use cell. If so, we repeat the traversal for the second use cell one more time but subtract one plus the maximum label collision value from each of its upstream cells’ new stepwise distance labels.

This algorithm is repeated for each of the remaining use cells and ultimately produces a single set of labels for all of the cells upstream of any user. Because each cell is traversed and labeled at most twice, this phase of the SPAN flow distribution algorithm is guaranteed to be linear $O(n)$.

In order to keep this phase linear for proximal flow services, we apply it only to the shared flow surface with origin $(0, 0)$ and then look up the generated flow surface labels using the cell index translation procedure described above.
As we move on to the forward propagation phase (Section 2.4.2), it should be easy to see that we are only touching each cell at most once. This is ensured because the algorithm partitions the cells by their stepwise distance labels and then runs through each grouping exactly once from largest to smallest. For channeled and diffusive flow, this naturally leads to linear time and space usage. However, for proximal flow services, this is the point at which they alone become quadratic $O(n^2)$ in their time requirements. This is because the linear procedure must necessarily be repeated once per user in the study region, and here no surface sharing tricks apply. Please note though, that since their result maps combine additively, we can still limit a proximal flow analysis to linear memory usage by running the users’ forward propagation phases sequentially and merging their results as we go.

Finally, in the backpropagation phase (Section 2.4.3), we are once again traversing the serviceshed(s) in the upstream direction. The local source and sink distribution calculations are constant time $O(1)$ per cell, and by simply reusing the stepwise distance partitioning in reverse, we can guarantee that all downstream cells will be processed prior to their upstream counterparts. This ensures that each cell will be entered at most once regardless of the use of a unidirectional, multidirectional, or conditional flow surface. However, as in the forward propagation case, proximal flow services will uniquely tend toward a quadratic time (but linear memory) solution because each user’s flow surface must be traversed independently.

In summary, all ecosystem services with in situ, channeled, diffusive, or global flow can be analyzed by the SPAN framework in linear $O(n)$ time and space with respect to the number of cells in the study region. Proximal flow services will require quadratic $O(n^2)$ time but only linear $O(n)$ space. Furthermore, by cleverly implementing shared surface-based approaches, many of the steps in a proximal flow assessment may be reduced to linear time as well. Ultimately, performance guarantees on this level should most definitely make the SPAN framework a viable option for high resolution landscape level analysis.
4.2 Valuing the Service Medium

Once we have generated the 30 result maps from a SPAN assessment, how might we apply them to the problem of ecosystem service valuation?

To begin, we strongly recommend that economic values be based on the Actual and Blocked layers rather than their Theoretical counterparts, which correspond more closely to the kinds of results expected in approaches based on *in situ* ecological production functions. For provisioning services, in which a user exhibits a beneficial relationship with the service medium, the Actual Rival Use and Actual Nonrival Use maps describe how much of the medium is received by each use location. For preventive services, in which the user benefits from avoiding the service medium, the Blocked Rival Use and Blocked Nonrival Use maps describe how much protection each user receives from upstream ecosystem sinks.

However, since economic values are not inherent in nature but rather imposed by human minds on their environments, we need to assign to each user type a utility function for the service medium based on their relationship with it. By then applying these utility functions to each use cell on the landscape, we can generate a map of value accrued for a single ecosystem service. The following figure illustrates a sample utility function that might be used in this way. Remember that for a provisioning service, we would use the Actual Rival Use or Actual Nonrival Use value per user, and for a preventive service, we would instead use the Blocked Rival Use or Blocked Nonrival Use as appropriate.
Of course, it is rare indeed for only a single ecosystem service to be assessed at a time. More common by far is the analysis of *service bundles*, in which the total (lower-bound) value for an area is calculated by combining together the individual values of multiple services. (Raudsepp-Hearne et al., 2010) The easiest way to do this – but also the most naive – is to simply calculate the values of each ecosystem service independently and then sum them together. A more nuanced approach employs multivariate utility functions, in which combinations of values from different services may produce non-additive results. The following plot shows a sample multivariate utility function for two hypothetical services.
Once the utility functions have been applied to generate value maps per service (or service bundle), we would next like to attribute these values back to the sources and sinks that produced them. Fortunately, we already have a means of accomplishing this: the backpropagation phase of the SPAN flow distribution algorithm. By replacing the Actual Rival Use, Actual Nonrival Use, Blocked Rival Use, and Blocked Nonrival Use values in each cell with the economic values generated by their respective utility functions, we can rerun this graph traversal operation once to create new Actual Source and Actual Sink maps, which will then show the economic contributions to all downstream users by all sources and sinks. Note that such an approach is only appropriate for multivariate utility functions if they separate provisioning and preventive services into separate bundles.

As a final note, in addition to the total ecosystem service value received by each user on the landscape, we might also consider what the spatial distribution of these values could tell us. Such information could be used to identify inequalities in service distribution, hotspots
of service flow convergence, or users which receive no ecosystem services at all. If the goals
of an ESAV study are focused around ensuring that minimum service delivery thresholds
are universally met, the SPAN output maps can, of course, highlight where this is or is
not being accomplished. Studying the flow path routes and densities in the Possible Flow,
Blocked Flow, and Actual Flow maps can also help to identify situations in which targeted
land management actions may increase or decrease flows to specific user communities as
desired.

All this is to say that the SPAN framework – although well prepared to handle ecosystem
service valuation problems – aims to provide a great deal more spatial information than
simply maps of current values produced and delivered. Far greater still is the insight it can
provide to sited actions and land management scenarios. In the following section, we dive
into this realm of costs and tradeoffs in greater detail.

4.3 Land Management Scenarios

It could be said that estimating the current distribution of ecosystem service values is only
half of the ESAV puzzle and that its much more challenging counterpart is predicting what
will happen given various changes to the underlying coupled human-natural system on which
they are based. In the previous section, we took on the first of these two. To address the
second, we need to once again introduce some new terminology.

When we run a SPAN ESAV analysis for one or more ecosystem services, we create a
set of maps that we will call the baseline results. As we have already seen, these establish
the initial levels of service medium production, absorption, consumption, and enjoyment
that we can expect from our study region. If we make changes to one or more parameters
that influence the values in the initial source, sink, rival use, nonrival use, or flow surface
maps, we refer to this set of changes as a scenario. Rerunning the SPAN ESAV analysis
under this scenario produces a new set of maps, called the scenario results.
Let us consider some examples of scenarios that could be incorporated in a SPAN model run. We might induce a change in land cover, such as deforesting a region or restoring a wetland complex. We might increase the number of users in an area, perhaps in line with the predictions of a population growth model. If we are interested in climate change, we might vary the temperature, humidity, and rainfall values in response to different microclimate weather models. If our concern is to observe development effects on flow path fragmentation, we might introduce new roadways or dam up rivers. Depending on the flow surfaces being used in our SPAN analysis, this may alternately increase flows to users, decrease them, or simply reroute them to different users than in the baseline case.

If our interest is instead focused on human interactions with service medium flows rather than their production or routing, perhaps we could vary the demand values found in the rival use maps. This technique might be used to show an increase in demand due to new technological innovations or a reduction in demand due to regulatory constraints. The SPAN scenario results associated with these changes could then be used to assess the effectiveness of different policy actions as well as their potentially unintended consequences.

For example, capping surface water extraction in the upper part of a watershed might provide more water to users in the lower watershed. However, SPAN’s network flow analysis might show that much of the additional water left untouched by the upstream rival users simply gets captured by sinks before reaching the downstream beneficiaries. In this case, it might be shown that reducing the intervening sink effects (or rerouting the water around them) would be of greater value in meeting the needs of downstream users than the original cap on extraction.

Regardless of the scenario chosen, the key to understanding the magnitude and kind of changes that it induces is to subtract each of the 30 baseline maps from their corresponding scenario results. The maps produced in this way are called the scenario change results. Since running the SPAN algorithm for each scenario is linear $O(n)$ in time and space and
subtracting their results from the baseline maps is also linear, the computational complexity of performing each additional scenario analysis is itself linear by definition.

Examining the scenario change results should quickly help us to determine which scenarios are helpful or harmful (and to whom) as well as which seem to have little or no effect on ecosystem service distributions. Stacking scenario change maps for multiple services under the same scenario can go even further towards showing us who wins or loses (and by how much) in each situation. It is expected that these kinds of analyses may generate the most interesting SPAN outputs for policy makers considering actions with multiple stakeholders. Finally, if scenario outcomes are better communicated in terms of economic gains or losses rather than service medium quantities, the techniques used in Section 4.2 may be used to accomplish this translation for the scenario change results as well.

4.4 Flow Path Analysis

In addition to spatially specific economic valuation and scenario analysis, the most innovative new approach that the SPAN framework makes possible is siting development actions based on flow path connectivity information.

Consider these common (but often extremely difficult) questions presented by those seeking ESAV information about their landscapes:

- Where are this user’s ecosystem services generated?
- Who receives the ecosystem services produced by this land?
- How much harm does this land cause to people in the region?
- Which users reduce service flows to other users?

Up until this point, we have dealt predominantly with SPAN result maps that relate all sources, sinks, and users with one another within the same study region. The questions presented above, however, require partitioning these complex flow networks into subgraphs
that connect to only specifically chosen sources, sinks, or users. We call this the *flow graph selection problem*, and once again the SPAN framework has an answer to even this rather tricky puzzle.

The key to the solution lies in our ability to traverse the flow surface both upstream and downstream from any cell all the way to its edge. Thus, when one asks where a user’s services are generated, we need only start from that user’s cell and follow the flow surface upstream to identify its associated serviceshed. If the user receives provisioning services from the landscape, then the Actual Flow map will contain positive values along all of the routes from the user to the sources in its serviceshed which contribute to its total ecosystem service value. If instead the user benefits from preventive services, theBlocked Flow map contains positive values leading from the use cell back to each of the sinks which protect it. Using these two layers as filters when traversing the flow surface enables us to carve out just those sources, sinks, and flow paths which generate and deliver services to our chosen user(s).

Similarly, if we want to know which users receive the ecosystem services produced by any piece of land, we need only follow the flow surface downstream from sources (for provisioning services) or sinks (for preventive services). Once again, the SPAN output maps can help us to prune the flow graph here. From sources, one need only travel downstream through cells with a positive Actual Flow value, and for sinks we only continue downstream as long as the Blocked Flow value is positive. The question of harm emanating from specific source or sink sites is solved in much the same way. In this case, we merely consider the Actual Flow from sources to be the result of a detrimental service medium and Blocked Flow from sinks to represent a reduction to a beneficial service medium flow.

Finally, when considering the problem of identifying competitive rival users, we may simply start from any rival or nonrival user and travel upstream along the flow surface as long as the Captured Flow map is positive to find those users which reduce service medium
flows to them. Likewise, by traveling downstream from a rival user as long as the Captured Flow map contains positive values, we can find all of the users which it detrimentally affects.

Perhaps even more novel than solving the flow graph selection problem is that our connectivity-based paradigm reveals the contributions to ecosystem service delivery from locations other than those containing sources, sinks, or users. Instead we focus on the flow paths directly and ask:

- By what routes are ecosystem services conveyed from the landscapes generating them to the people benefitting from them?

- Are there places where many such routes converge? How do management impacts in these locations differ from those in areas with less concentrated flows?

- Are there beneficiaries with only one or a few routes to them? How does this flow path scarcity affect the value of ecosystem services delivered along these routes? In which situations should flow corridors be considered substitutable or non-substitutable?

- Where would landscape management actions have the greatest impact (for better or worse) on ecosystem service delivery? Is it possible to increase or decrease flows of ecosystem services to people without directly affecting the places where they are produced?

As a group, these kinds of questions belong to a category we call flow path analysis. Consider the issue of converging paths on a flow surface. Since any impact on a path’s throughput propagates on to all downstream users, we can see right away that increasing source or sink values at these convergence points are more likely to affect greater numbers of users than siting changes anywhere else on the flow surface. We call such locations scenario hotspots.

The opposite concept might be called a scenario cold zone. This is any cell (or contiguous region of cells) which does not fall inside any user’s serviceshed. As a result, any scenario
changes to these locations other than the introduction of new users should not affect existing ecosystem service flows or values in the study region. All cells with non-zero values on the Inaccessible Flow map belong to this category.

Next, we come to the problem of flow path scarcity. If the Actual Flow value along any path delivering provisioning services is insufficient (or only barely sufficient) to meet the needs of its users, then we would consider this to be a critical flow path. The same would apply to preventive service paths with insufficient Blocked Flow. Increasing source values and decreasing sinks along critical provisioning service paths should then be noted as high priority when considering alternative sites for management actions. Likewise, decreasing source values and increasing sinks are the clear remedies to critical preventive service paths.

Finally, we reach the issue of flow path fragmentation. In much the same way that animals cannot move easily through fragmented habitats, service media cannot flow along breaks in the flow surface. Identifying such discontinuities and then acting to remedy them can sometimes be the most effective way of generating new flows of ecosystem services. For example, routing water via an irrigation canal from a productive river to a dry streambed provides services to users in a drought-stricken region. Such a management action is quite similar to building a connecting road between two highways to reroute traffic. Likewise, since every positive effect in the SPAN framework has its negative counterpart, we could attempt to break a flow path with an artificial sink to protect downstream users from detrimental flows.

4.5 Final Thoughts

This concludes our discussion of the SPAN framework for ecosystem service assessment and valuation. Its development has been both a struggle and a labor of love for many years now, and it is our sincere hope that it will be of value not only to the academic community around ESAV but far more importantly to those people charged with managing and developing our
land sustainably into the future.

By providing a single generalizable abstraction to cover the majority of all spatially quantifiable ecosystem services, we have aimed to create a unifying paradigm and set of metrics in which to conduct their assessments. Notably, a great deal of freedom is left to the developer of any SPAN model, from the choice of value system used to quantify service media to the decision to use unidirectional, multidirectional, or conditional flows. Moreover, since no specific functions are prescribed for defining the source, sink, rival use, or nonrival use layers, any approach that is deemed most appropriate by the modeler may be used, provided it creates final maps with the correct semantics.

Perhaps even more importantly, we hope through this work to convince more ESAV practitioners that landscapes should be valued according to the ecosystem services which they deliver to users rather than by spatially static ecological production functions that only estimate their potential for providing these services. Furthermore, by drawing attention to the flow paths directly – in addition to the sources, sinks, and users that fall along them – we intend to both raise awareness about the problems of flow path criticality and fragmentation as well as provide clear tools to target management actions for greatest effect.

As with all research endeavors, this work has highlighted a number of interesting open problems still to consider. First is how to automatically determine the minimum and maximum resolutions within which the underlying flow paths for any service medium may be successfully anti-aliased. Second is whether any performance gains might be achieved by translating the SPAN implementation presented here to use a vector model (rather than a raster model) for its input and output datasets. Finally, the subject of feedback loops, critical thresholds, ecological tipping points, equilibria, and resilience are of significant interest to those working in dynamical systems research, but as of yet, these concepts have no clear representation within the SPAN framework. I have suggested that a first step might be to iterate SPAN simulations with different flow surfaces in order to create feedback loops
or even to incorporate an embedded SPAN model into a larger dynamic model. Regardless of the specific method chosen, this particular problem looks to provide many interesting avenues for future research.

Finally, as a parting thought, it would be truly game changing for both the field of computer science and the environmental movement if more computer scientists could find it in themselves to tackle, with the enormous tools at their disposal, the environmental problems that present the greatest challenges of our time. With so much at stake, it is deeply disappointing to find so few technology experts on the front lines where their assistance is so sorely needed. Consider this then a call to action. Put down your e-commerce websites and mobile phone games, and use your skills to enact positive change in the world. Together we can make a difference, so come and join the revolution. Hack the planet!
References


Robert Costanza, Matthew Wilson, Austin Troy, Alexey Voinov, Shuang Liu, and John D’Agostino. The value of New Jersey’s ecosystem services and natural capital. *Gund Institute for Ecological Economics, University of Vermont and New Jersey Department of Environmental Protection, Trenton, New Jersey*, 13, 2006.


Gary W. Johnson, Kenneth J. Bagstad, Robert R. Snapp, and Ferdinando Villa. Service Path Attribution Networks (SPANS): Spatially quantifying the flow of ecosystem services from landscapes to people. In David Taniar, Osvaldo Gervasi, Beniamino


Mark L Plummer. Assessing benefit transfer for the valuation of ecosystem services.


