

Dual-scale Landscape for LANDIS-II Concepts & Design

James B. Domingo
Robert M. Scheller
Catherine Ravenscroft

University of Wisconsin-Madison

Last Revised: March 21, 2007

Table of Contents

1	INTRODUCTION	3
1.1	References.....	3
1.2	Acknowledgements.....	3
2	DUAL-SCALE LANDSCAPE	4
2.1	Motivation.....	4
2.2	Relationship Between Scales	4
2.3	Design Goals.....	5
2.3.1	<i>Use Fewer Resources than Whole Landscape at Fine-scale</i>	5
2.3.2	<i>Avoid One-off Software</i>	5
2.3.3	<i>Just One Version of LANDIS-II in the Long Run</i>	5
2.3.4	<i>Allow Extensions Written for One Scale</i>	5
3	DESIGN A – TWO SIZES OF SITES	6
3.1	Site Locations	6
3.2	A Site’s Neighbors.....	7
3.3	Modifications to Current Design of Site Data Type	8
3.3.1	<i>Scale Property</i>	8
3.3.2	<i>IsActive Property</i>	8
3.3.3	<i>IsSubdivided Property</i>	9
3.3.4	<i>Area Property</i>	9
3.3.5	<i>Parent Property</i>	9
3.3.6	<i>GetEnumerator Method</i>	9
3.3.7	<i>AllChildren Property</i>	9
3.3.8	<i>GetChild Method</i>	10
3.3.9	<i>IsMutable Property</i>	10
3.4	Site Variables.....	10
3.4.1	<i>Large Subdivided Sites</i>	11
3.5	Landscape	12
3.5.1	<i>Active Site Indexer</i>	12
3.5.2	<i>GetSite Method</i>	13
3.5.3	<i>IsValid Method</i>	13
3.5.4	<i>Default Site Iterator</i>	13
3.5.5	<i>AllSites Property</i>	13
3.5.6	<i>AllLargeSites Property</i>	14
3.5.7	<i>ActiveLargeSites Property</i>	14
3.6	Output Maps	14
4	DESIGN B – FINE-SCALE DATA SHARING	16
4.1	Landscape	17
4.1.1	<i>Default Site Enumerator</i>	17
4.1.2	<i>Distinct Active Sites Enumerator</i>	18
4.1.3	<i>All Sites Enumerator</i>	18
4.1.4	<i>Block Row Buffer</i>	18
5	INPUT DATA	20

5.1	Dual-Scale in Scenarios	20
5.1.1	<i>New DualScale Parameter</i>	20
5.1.2	<i>Broad-scale Input Maps</i>	21
5.2	Initial Communities Map	21
5.2.1	<i>Fine-scale Data Sharing</i>	21
5.3	Input Maps for Extensions	22
6	OUTPUT EXTENSIONS	24
6.1	Age Output Extension.....	24
6.1.1	<i>Two Site Sizes</i>	24
6.1.2	<i>Fine-scale Data Sharing</i>	24
6.1.3	<i>Correcting Performance Penalty</i>	25
6.2	Reclass Extensions.....	26
6.2.1	<i>Two Site Sizes</i>	27
6.2.2	<i>Fine-scale Data Sharing</i>	27
7	SUCCESSION	28
7.1	Succession Component (Library).....	28
7.1.1	<i>Seeding</i>	28
8	DISTURBANCES	29
8.1	AgeCohort.IDisturbance Interface	29
9	WIND EXTENSION	30
9.1	Event Spreading	30
9.1.1	<i>Fine-scale Data Sharing</i>	30
9.2	Severity Maps	31
9.2.1	<i>Two Site Sizes</i>	31
10	HARVEST EXTENSION	32

1 Introduction

The purpose of this document is to describe the concepts and design of dual-scale landscapes for the LANDIS-II model. The primary audience for this document is the team which will be developing this functionality for LANDIS-II. Since that team includes ecologists and developers, this document must be understandable by both types of team members. The document assumes the reader is familiar with the conceptual description of the LANDIS-II model.

1.1 References

LANDIS-II Model v5.1 Conceptual Description.

LANDIS-II Model User Guide.

(TNC proposal?) Simulating the Effects of Climate Change and Disturbance on Forest Composition and Rates of Change in a Northeastern Minnesota Landscape Using a Spatially Dynamic Model., White, M.A. et al.

1.2 Acknowledgements

Funding for this work was provided by ...

2 Dual-scale Landscape

2.1 Motivation

Currently, the landscape in the LANDIS-II model is represented by a 2-dimensional grid of equally-sized squares called **sites** or **cells** (see chapter 3 in the *LANDIS-II Conceptual Model Description*).

The motivation for the dual-scale landscape is to allow the user to subdivide selected areas of the landscape into smaller sites (see Figure 1). This functionality would allow the user to investigate the model's behavior in response to spatial variability in topography, soils, climate, etc. at a finer scale in selected regions of the landscape.

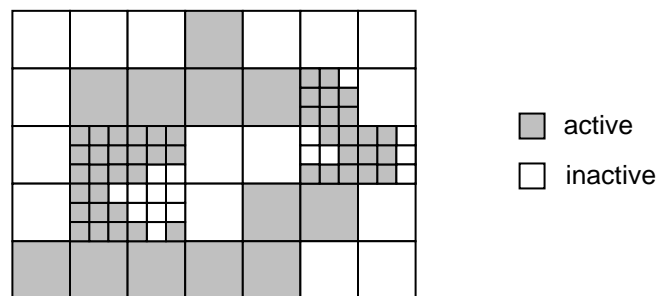


Figure 1 – Dual-scale landscape: selected sites are subdivided.

Without this functionality, the user has to model the whole landscape at the finer scale, which requires significantly more computational resources than modeling the landscape at the broader scale. For example, simulating a landscape at the smaller resolution of 28.5 m (0.08 ha per site) requires approximately 36 times the memory and processor time than simulating it at the larger resolution of 171 m (2.9 ha per site).

2.2 Relationship Between Scales

Like the larger sites, the smaller sites are equally-sized squares. Furthermore, a large site is subdivided into complete small sites (i.e., there are no partial small sites). Therefore, the cell length of a large site is an integer multiple of the cell length of a small site:

$$\text{Cell length}_{\text{large-scale}} = k \times \text{Cell length}_{\text{fine-scale}}$$

where k is integer ≥ 2

Allow $k = 1$? Would be single-scale essentially.

2.3 Design Goals

2.3.1 Use Fewer Resources than Whole Landscape at Fine-scale

With the current version of LANDIS-II, the user can model the whole landscape at the finer scale. But that requires a lot of computational resources (memory and processor time). The goal is to use fewer resources by allowing some areas of the landscape to be represented at the broader scale. This means less memory for site-specific data and hence, less time to do site-specific computations.

2.3.2 Avoid One-off Software

We want to maximize the chances that this new functionality becomes a permanent part of the LANDIS-II software. We want to avoid it becoming a variation or a fork of the code base that is just used for the RJ Kose grant, and then abandoned. We'll be investing a lot of time into developing this functionality, and we want to maximize the return on that investment by maximizing the software's longevity.

2.3.3 Just One Version of LANDIS-II in the Long Run

Although we may create a separate version of LANDIS-II during the development of this new functionality, we don't want to distribute and support multiple variations of LANDIS-II in the long run. We don't want the hassle for us or the users of dealing with single-scale LANDIS-II versus dual-scale LANDIS-II. In other words, we want to eventually fold this new functionality into the current code base so there is only one version of LANDIS-II that we maintain and distribute. In essence, this one version of LANDIS-II would support single-scale and dual-scale simulations.

2.3.4 Allow Extensions Written for One Scale

We want to allow developers to write extensions without having to consider the dual-scale behavior. In other words, they can write extensions for a single scale, and the extensions would function correctly and efficiently with a dual-scale landscape. Therefore, in order to get existing extensions to work with dual-scale landscapes, we want to make no or very minimal changes to their source code.

3 Design A – Two Sizes of Sites

In this design, the two sizes of sites (large and small) are represented explicitly. In other words, a site has a new property called “Scale” which indicates the site’s scale or size.

3.1 Site Locations

In the current single-scale landscape, a site is identified by its location (row, column). In a dual-scale landscape, a site’s location is expressed in terms of rows and columns that are based on the site’s scale. So, a large site’s location is expressed in terms of large-scale rows and columns, while a small site’s location is expressed in terms of small-scale rows and columns.

Consider the example dual-scale landscape in Figure 2. The location of the large subdivided site F is (2, 2) while the location of the small site F7 is (6, 4).

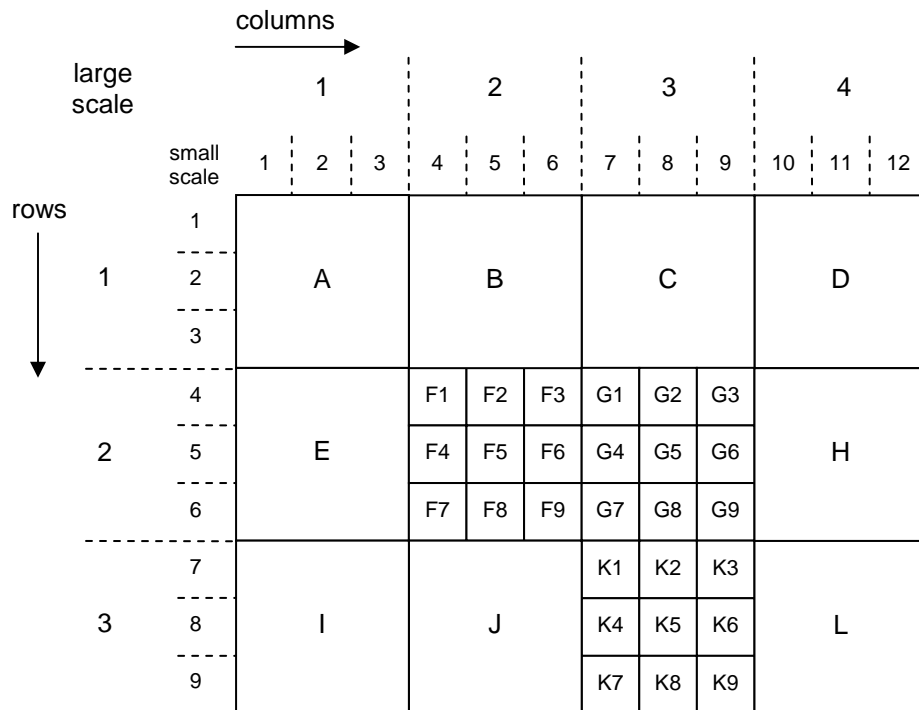


Figure 2 – Rows and columns at two scales

As a consequence of adding a second scale to the landscape, a site cannot be identified solely by its location. On a dual-scale landscape, a location and a scale are required to identify or retrieve a site. For example, in Figure 2, retrieving the site at “(3, 2), large scale” yields site J, while retrieving the site at “(5, 9), small scale” yields site G6.

Note: retrieving a site with a small-scale location may result in a large site. For example, in Figure 2, retrieving the site at “(1,4), small scale” yields the large site B.

3.2 A Site’s Neighbors

The neighbors of a site are specified by their relative location to the site. A relative location is a pair of offsets (row, column). Each offset can be positive, zero, or negative.

In a dual-scale landscape, the offsets in a relative location are interpreted in terms of the site's scale. So a relative location of a large site's neighbor is interpreted in terms of large-scale rows and columns. Similarly, a relative location of a small site's neighbor is interpreted in terms of small-scale rows and columns.

Note: When a neighbor is retrieved for a large site, the neighbor is always a large site. When a neighbor is retrieved for a small site, the neighbor may be either another small site or a large site.

Here are some neighbors for some of the sites in the example dual-scale landscape in Figure 2.

<u>Site</u>	<u>Relative Location</u>	<u>Neighbor</u>
E	(-1 , 2)	C
E	(0 , 2)	G
E	(1 , 2)	K
F9	(1 , 0)	J
F9	(1 , 1)	K1
F9	(1 , 2)	K2
F9	(1 , 3)	K3
F9	(1 , 4)	L
F9	(1 , 5)	L
F9	(1 , 6)	L

3.3 Modifications to Current Design of Site Data Type

3.3.1 Scale Property

This new property indicates the site's scale. Value: Large or Small.

3.3.2 IsActive Property

This existing property indicates whether the site is active or not. Value: true or false.

By definition, a large subdivided site is active if at least one of its small sites is active.

3.3.3 IsSubdivided Property

This new property indicates whether the site is subdivided into smaller sites. Value: true or false.

This property is always false for small sites.

3.3.4 Area Property

This new property is the area of site. Value: number > 0. Units: m².

This may be useful for extensions that need to know this. Currently, the core provides extensions with two properties: CellLength and CellArea. Perhaps these should be moved to the Landscape class?

3.3.5 Parent Property

This new property is the site's parent: the large-scale site that encompasses the site. Value: a large-scale site.

By definition, the parent of a large-scale site is itself.

Alternatives to "Parent" as a name? "EnclosingLargeSite", "MyLargeSite"

3.3.6 GetEnumerator Method

This new method will be added to allow the developer to iterate through the small active sites in a large divided site.

```
foreach (Site smallSite in largeSite) {  
    ...  
}
```

If the site is a small site, this method returns an enumerator which iterates through one site: the small site itself. If the site is a large undivided site, this method returns an "empty" enumerator which doesn't iterate through any small sites.

3.3.7 AllChildren Property

This new property returns an enumerator that iterates through all the small sites (active and inactive) for a large subdivided site. If the site is a large undivided site, then this property returns an "empty" enumerator. If the site is a small site, then the enumerator iterates through just one site: the small site itself.

Alternative name: "AllSmallSites"

3.3.8 GetChild Method

This new method returns a small site within a large subdivided site. The parameter is the small site's location within the large site. Therefore, if the parameter is location (1,1), then the upper-left small site is returned. If the location is out-of-bounds (row or column is not between 1 and k), then the method returns a null site.

Null site is returned by the default constructor of the new Site value type. Its location is (0, 0) (returned by the Location's default constructor), and its landscape is null. Would a new boolean property called IsNull be useful? Alternative names: IsZero, IsValid, Exists

3.3.9 IsMutable Property

This existing property will be removed. It's needed because sites are implemented as objects (i.e., a reference type), and site iterators (enumerators) use mutable versions of site objects to minimize object creation.

The development of various extensions has demonstrated the need for immutable versions of site objects. But according to the .NET guidelines for value types, sites should be implemented as a value type (structs in C#). Since value types are stack-based and utilize copy semantics for assignment operations, they are immutable.

Changing the Site type from a reference type to a value type would eliminate the need to implement an ImmutableSite type and would allow the elimination of existing mutable classes.

This change would break binary compatibility for 5.1 extensions. However, source compatibility may still be possible (source compatibility = no changes to source code, just need to recompile the extension).

3.4 Site Variables

Site variables are data structures for storing site-specific information. A site variable has a distinct value for each active site on the landscape. A site variable handle inactive sites in one of two ways: 1) all the inactive sites share a single common value, or 2) each inactive site has its own distinct value. The former variation uses less memory and is used much more frequently by extensions.

3.4.1 Large Subdivided Sites

How are large subdivided sites handled? A site variable doesn't allocate memory for large subdivided sites; it only allocates memory for small active sites and large active undivided sites. In other words, if a site variable is indexed with a large subdivided site, what happens?

One option is to raise an `InvalidOperationException`. With this option, any code using a site variable has to check for large subdivided sites before indexing the site variable.

Another option is to have two new delegate properties which are called whenever a large subdivided site is used with a site variable.

```
class SiteVariable
{
    static class Delegates
    {
        delegate T GetSubdividedSite<T>(Site site);
        delegate void SetSubdividedSite<T>(Site site,
                                           T value);
    }
}

interface ISiteVar<T>
: SiteVariable
{
    Delegates.GetSubdividedSite<T> GetSubdividedSite
    { get; set; }
    Delegates.SetSubdividedSite<T> SetSubdividedSite
    { get; set; }
}

// example of use

static class FooUtil
{
    static Foo GetSubdividedSite(Site site)
    {
        // Maybe average or total the values for
        // the site's small sites.
    }

    static void SetSubdividedSite(Site site,
                                  Foo value)
    {
        // Maybe assign the value to all the site's
        // small sites. Or, partition the value
        // among the small sites.
    }
}
```

```
}  
  
ISiteVar<Foo> fooVar;  
fooVar = Model.Core.Landscape.NewSiteVar<Foo>();  
fooVar.GetSubdividedSite = FooUtil.GetSubdividedSite;  
fooVar.SetSubdividedSite = FooUtil.SetSubdividedSite;
```

By default, when a site variable is created on a dual-scale landscape, these two new delegate properties point to methods that simply raise an `InvalidOperationException`.

3.4.1.1 Limitations

The technique works satisfactorily for site variables that store simple data types (e.g., numbers) for which there are clearly-defined ways to generate a broad-scale value from fine-scale value. But issues arise with site variables that store more complex data types like groups of cohorts at a site. For these data types, it may be really difficult (or impossible) to define a technique for generating a broad-scale value from fine-scale values.

For example, the succession extension creates a site-variable to store a group of cohorts for each active site. With a dual-scale landscape, this site variable has a group of cohorts for each large active undivided site and for each small active site.

Now consider the existing “Max Species Age” output extension which generates maps for maximum cohort age for selected species. This extension iterates through the active broad-scale sites on the landscape, gets the group of cohorts at each site from the cohort site-variable, and passes each cohort group to a utility function that computes the maximum cohort age at the site.

In order to avoid changing this extension’s source code, somehow the cohort site-variable would need return a broad-scale cohort object when the variable is indexed with a broad-scale site that’s divided. Not sure if it’s possible to define such a broad-scale group of cohorts from the groups of cohorts at the fine-scale sites.

3.5 Landscape

3.5.1 Active Site Indexer

The parameter is the site’s location.

Should this location be interpreted at the large scale? This would mean that existing source code would be treated as operating at the large scale.

3.5.2 GetSite Method

The parameter is the site's location.

Should this location be interpreted at the large scale? This would mean that existing source code would be treated as operating at the large scale.

Need to define an overload of this method that takes a second parameter which is a scale value. The second parameter indicates which scale to use for interpreting the location.

3.5.3 IsValid Method

This existing method determines if a location is valid for the landscape.

Should this location be interpreted at the large scale? This would mean that existing source code would be treated as operating at the large scale.

Need to define an overload of this method that takes a second parameter which is a scale value. The second parameter indicates which scale to use for interpreting the location.

3.5.4 Default Site Iterator

Currently, with a single-scale landscape, the GetEnumerator method of a landscape returns an enumerator that iterates over all the active sites in the landscape.

With a dual-scale landscape, should this enumerator iterate over all active sites (large and small)? Or should it iterate over just the large active sites? The latter approach would mean that existing source code would be treated as iterating over active sites at the broad scale.

To avoid changing the source code for existing single-scale extensions, this enumerator would iterate over all active sites, both large and small.

3.5.5 AllSites Property

This existing property returns an enumerator that iterates through all the sites (active and inactive) on a landscape. To avoid changing the

source code for existing single-scale extensions, this enumerator would iterate over all the sites at fine scale, as described in section 3.6.

For a dual-scale landscape, should this enumerator iterate over all sites (large and small)? Or should it iterate over just the large sites (active and inactive)?

3.5.6 AllLargeSites Property

This new property would return an enumerator that iterates through all the large sites on a landscape.

3.5.7 ActiveLargeSites Property

This new property would return an enumerator that iterates through all the active large sites on a landscape.

3.6 Output Maps

How does an extension generate an output map at fine-scale? Currently, to generate a single-scale output map, the extension iterates through all the sites (active and inactive) on the landscape, so it can generate pixel values for the output map.

```
// Open the output map for writing
foreach (Site site in Model.Core.Landscape.AllSites)
{
    // compute a pixel value for the site
    // write the pixel value to the map
}
```

What would the loop for a fine-scale map look like? Specifically, what sort of iterator do we need for the landscape? It would appear that we would need an iterator that goes through all the fine-scale sites on the landscape. Problem: the landscape is not divided into fine-scale sites everywhere.

Consider the sample dual-scale landscape in Figure 2. In order to generate pixel values for a fine-scale output map, we need to access the large undivided sites multiple times:

Pixel Location	Site
(1 , 1)	A
(1 , 2)	A
(1 , 3)	A

(1 , 4)	B
(1 , 5)	B
(1 , 6)	B
...	...
(1 , 11)	D
(1 , 12)	D

This same sequence of sites would be repeated for the pixel locations in rows 2 and 3. For row 4, this is the sequence of sites accessed:

<u>Pixel Location</u>	<u>Site</u>
(4 , 1)	E
(4 , 2)	E
(4 , 3)	E
(4 , 4)	F1
(4 , 5)	F2
(4 , 6)	F3
(4 , 7)	G1
(4 , 8)	G2
(4 , 9)	G3
(4 , 10)	H
(4 , 11)	H
(4 , 12)	H

One approach: we create an iterator that returns all sites in a landscape at fine scale, recognizing that we will get back both large and small scale sites.

Second approach: we create an iterator for all the site locations on a landscape at fine scale, and then use the landscape's GetSite(location, "Fine scale") method to get each site in turn.

Drawbacks of 2nd approach is that inefficiency. Since we're explicitly getting each site, we can't construct an enumerator that remembers where it is in the landscape, and hence retrieving the next site would be quicker. *(May need to investigate this closer to confirm)*

4 Design B – Fine-scale Data Sharing

In this design, there is only one scale for sites: fine scale. Therefore, there is no need for a new “Scale” property for sites; they are always fine scale.

The broad scale is simulated by aggregating the data for a block of adjacent fine-scale sites. In other words, the block of fine-scale sites share a single data value in a site variable. This is how the current design of LANDIS-II handles inactive sites across the whole landscape.

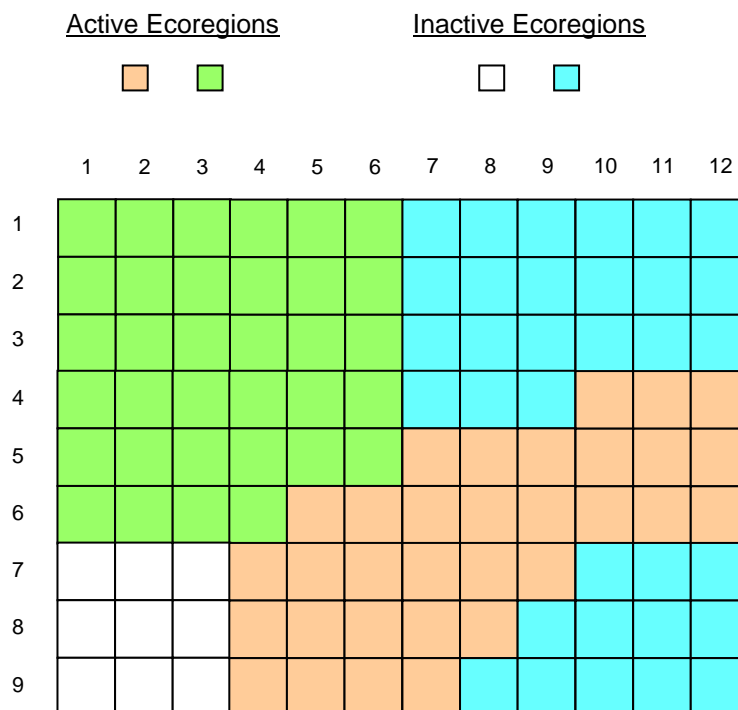


Figure 3 – Sample ecoregion map

Consider the input map of ecoregions in Figure 3. Two ecoregions are active (green and tan); the other two ecoregions (white and blue) are inactive. If the broad-scale factor is 3, then each 3-by-3 block of active sites are aggregated if all the sites are in the same ecoregion.

Therefore, if all the 9 sites in a block (i.e., a broad-scale site) are in the same active ecoregion, then all the sites share the same data value in site variables.

Figure 4 shows the data indexes assigned to each site if 3-by-3 aggregation is applied to the ecoregion map in Figure 3.

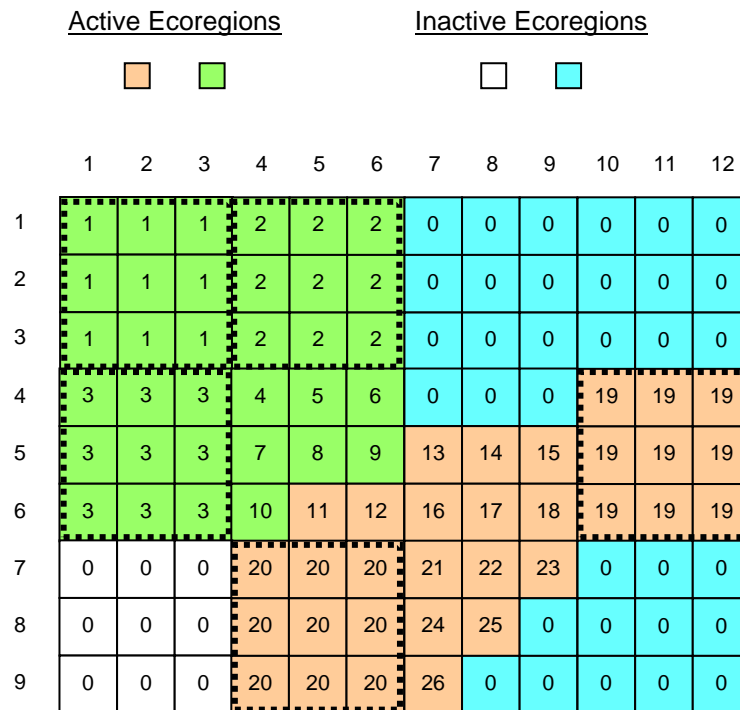


Figure 4 – Data indexes for sites based on 3-by-3 aggregation

So although there are 66 active sites on the landscape, they only require memory to store 26 data values in each site variable.

4.1 Landscape

4.1.1 Default Site Enumerator

For consistency with the current single-scale LANDIS-II design, the default site enumerator for a dual-scale landscape in this design would iterate through all the active sites.

4.1.2 Distinct Active Sites Enumerator

This new type of site enumerator for a landscape would iterate through all the distinct active sites. This would ensure that data values shared by active sites are visited only once.

For the example landscape shown in Figure 4, the distinct-active-site enumerator would return this sequence of sites:

```
( 1 , 1 )
( 1 , 4 )
( 4 , 1 )
( 4 , 4 )
( 4 , 5 )
( 4 , 6 )
( 4 , 10 )
( 5 , 4 )
( 5 , 5 )
( 5 , 6 )
( 5 , 7 )
( 5 , 8 )
( 5 , 9 )
( 6 , 4 )
( 6 , 5 )
( 6 , 6 )
( 6 , 7 )
( 6 , 8 )
( 6 , 9 )
( 7 , 4 )
( 7 , 7 )
( 7 , 8 )
( 7 , 9 )
( 8 , 7 )
( 8 , 8 )
( 9 , 7 )
```

Before implementing this new site enumerator, we need to determine if there's a need for it.

4.1.3 All Sites Enumerator

This enumerator is typically used for reading input maps and generating output maps. It would return every site in the landscape in row-major order, as is currently done in LANDIS-II.

4.1.4 Block Row Buffer

The following chapters describe algorithms for this landscape design. Several of those algorithms need to store data for a single row of blocks, i.e., a broad-scale row on the landscape. A block row buffer

has one data value for each broad-scale column in the landscape. Each data value is identified by its broad-scale column using the same notation used to specify an element in an array:

```
blockRowBuffer[blockLocation.column] = dataValue
```

For example, a block row buffer for the landscape in Figure 4 would have 4 data values. These 4 data values are identified by the column values 1, 2, 3 and 4.

4.1.4.1 Implementation

A block row buffer is essentially an array; therefore, an array can be used to implement it. The only difference is that the buffer indexes start at 1, where array indexes start at 0. Buffer indexes start at 1 in order to keep the algorithms simple. There are two possible approaches to implementing block row buffers with arrays:

- Allocate an array of k elements for each buffer where k is the number of broad-scale columns in the landscape. When a buffer is indexed with a column, compute the array index on the fly:

```
buffer[column] → array[column - 1]
```

- Allocate an array of $k + 1$ elements for each buffer where k is the number of broad-scale columns in the landscape. When a buffer is indexed with a column, simply use the column as the array index:

```
buffer[column] → array[column]
```

The space-vs-time tradeoff with this approach: the first element in the array (index = 0) is not used, in exchange for not having to compute the array index for each access.

5 Input Data

5.1 Dual-Scale in Scenarios

- How does the user specify that a scenario uses a dual-scale landscape?
- How does the user specify the resolution of the two scales?
- How does the user indicate which large sites are subdivided?
- How does the user indicate which small sites are active?

Current idea is to have the ecoregions map at the fine scale. Then the user uses a new parameter to denote the integer multiplier for the broad scale (k in Equation 1).

5.1.1 New DualScale Parameter

This is a new parameter for the scenario file (see chapter 4 in the *LANDIS-II Model User Guide*). It comes after the `CellLength` parameter. The parameter is optional to maintain backward compatibility. If the parameter is not present, then the landscape is single-scale.

If the parameter is present, then the resolution of the ecoregions map is the fine scale, and the parameter's value represents the multiplier for computing the broad scale.

```

LandisData      Scenario
...
Ecoregions      path/to/ecoregions.txt
EcoregionsMap   path/to/ecoregions/map.gis

CellLength      100  << meters

DualScale       5    << broad-scale multipiler

InitialCommunities      path/to/init-communities.txt
InitialCommunitiesMap   path/to/init-communities.gis
...

```

If the parameter is present, the initial-communities map is expected to be at the same resolution of the ecoregions map (i.e., fine-scale).

*Maybe this parameter should be named `BroadScaleMultiplier?`
`BroadScaleFactor?`*

Should this parameter be located before the `Ecoregions` parameter?

5.1.2 Broad-scale Input Maps

Idea for having the ecoregions and initial-community maps at broad scale. Map codes would indicate whether a broad-scale site is divided or not and how it's divided. For example, one map code may represent that the site belongs to ecoregion X and is undivided. Another map code may represent that the site is divided with its leftmost column of small sites all belonging to ecoregion X and all the other small sites belonging to ecoregion Y.

The number of map codes needed would depend the broad-scale multiplier (k , i.e., the number of small sites per large site), and how many combinations exist on the landscape.

Need to describe this idea more fully. It'd likely be more work for the modeler to prepare these type of input maps, but they'd be smaller files. Just wanted to record the idea.

5.2 Initial Communities Map

5.2.1 Fine-scale Data Sharing

With this design, the initial communities map is fine scale. The succession extension reads this map to initialize the cohorts at each active site. Active sites with unique data indexes are processed in the same manner as currently done in LANDIS-II: the initial community associated with a site is used to initialize that site's set of cohorts.

Active sites that share data indexes are handled differently. If all the sites in a block (broad-scale site) share a single data index, then they will share a single set of cohorts. One initial community must be identified to initialize this set of cohorts.

That community is determined by majority rule. The community that is assigned to the most sites in the block in the map is the community that's used to initialize the shared set of cohorts. If two or more communities appear the most frequently in the block, then one of these communities is randomly chosen for cohort initialization.

5.2.1.1 Algorithm

In order to record how frequent initial communities appear in a block, this algorithm uses a block row buffer of dictionaries. Each dictionary records which initial communities appear in its associated block, and how frequently. The keys used to index the dictionary are the map codes of the initial communities. The value associated with a key (community's map code) is the # of times that the community appears in the block.

```

with initial-communities map:
  foreach site in landscape.AllSites:
    read pixel from map
    if site.IsActive:
      communityMapCode = pixel.Band0
      if site.SharesData:
        blockLocation = site.BroadScaleLocation
        dictionary = buffer[blockLocation.Column]
        if dictionary.HasKey(communityMapCode):
          dictionary[communityMapCode] += 1
        else
          dictionary[communityMapCode] = 1

        lowerRight = Location(landscape.BlockSize,
                              landscape.BlockSize)
        if site.LocationInBlock = lowerRight:
          // last site in the block, so process the
          // dictionary
          community = mostCommonCommunity(dictionary)
          initializeCohorts(site, community)
          dictionary.RemoveAll()

      else
        // the same initialization as currently done
        // in LANDIS-II for an active site with an
        // unique data index
        community = lookup(communityMapCode)
        initializeCohorts(site, community)

```

The **highlighted** parts in the algorithm above represent changes to the current LANDIS-II design.

5.3 Input Maps for Extensions

An input map can either be at the large scale or the fine scale.

The resolution for an input map for an extension is based on the scale at which the extension operates. If the extension has been developed to operate at the broader scale, then it will expect its input maps to be at the larger resolution. If the extension has been developed to operate at the finer scale, then it will expect finer-resolution input maps.

A broad-scale extension is responsible for handling pixel data for large sites that are subdivided. Depending upon the type of input data, the extension may replicate the large site's data value for all its small sites, or the extension may subdivide the large site's data value among all its small sites.

Conversely, a fine-scale extension is responsible for handling pixel data for large sites that are not subdivided. For a large undivided site, the extension must combine the pixel data for all the corresponding small sites in the input map into a single data value. Depending upon the type of input data, this combination may be a summation or an average.

6 Output Extensions

This chapter examines the design options in relation to various existing LANDIS-II output extensions.

6.1 Age Output Extension

The main processing loop in this extension is:

```
foreach species in selectedSpecies:
  open output map using species' name
  foreach site in landscape.AllSites:
    if site.IsActive:
      age = computeMaxAge(cohorts[site][species])
    else
      age = 0
      pixel.Band0 = age
      write pixel to map
  close map
```

6.1.1 Two Site Sizes

This design option can preserve the functionality of this main loop. In other words, if the AllSites enumerator works as described in section 3.5.5, then source code will produce the correct maps with a dual-scale landscape.

However, because large subdivided sites are returned multiple times by the AllSites enumerator, the maximum-age calculation is repeated unnecessarily for each active small site in a large subdivided site. In effect, the loop is executed as many times as it would be with a single-scale landscape at fine-scale. Hence the extension takes the same amount of time to execute as it would running over a whole landscape at fine-scale.

6.1.2 Fine-scale Data Sharing

This design option can also preserve the functionality of this main loop. Thus the extension will produce the correct maps with a dual-scale landscape.

However, this design also has the same issue with repeating the maximum-age calculation unnecessarily on the same set of site cohorts. Specifically, if a block of active sites shares a set of site cohorts, then the main loop will calculate the maximum age for those cohorts for each active site in the block. So the extension takes the same amount of time to execute as it would running over a whole landscape at fine-scale.

6.1.3 Correcting Performance Penalty

The performance penalty described in the previous two sections arises because the current design of the extension's main loop assumes that each active data value will only be accessed once. This assumption is not valid for the dual-scale landscapes.

In order to manifest the performance gain that is possible with either dual-scale designs, we must somehow save the maximum ages for those sites that share data values. Specifically, the extension would need to have a max-age site variable. The main loop would be split into two loops: the first loop which assigns this new site variable, and the second loop creating the output map.

```
maxAge.InactiveSiteValues = 0
foreach species in selectedSpecies:
  foreach active site in landscape: // default enumerator
    maxAge[site] = computeMaxAge(cohorts[site][species])

  open output map using species' name
  foreach site in landscape.AllSites:
    pixel.Band0 = maxAge[site]
    write pixel to map
  close map
```

6.1.3.1 Traversing the Landscape Once

Although the source code modification described in the previous section eliminates repetitions of the maximum-age calculations, it still traverses the landscape twice for each species. By using a couple new properties to the Site data type, the main loop can be modified to traverse the landscape once.

```
maxAge.InactiveSiteValues = 0
foreach species in selectedSpecies:
  open output map using species' name
  foreach site in landscape.AllSites:
    if site.IsActive:
      if site.SharesData:
        if site.LocationInBlock = (1,1):
          age = computeMaxAge(cohorts[site][species])
          maxAge[site] = age
        else:
          // already computed age for the block
          age = maxAge[site]
      else:
        age = computeMaxAge(cohorts[site][species])
    pixel.Band0 = age
    write pixel to map
  close map
```

Although the algorithm above generates an age map for a single species with one pass through the landscape, it uses a site variable for buffering purposes when a single block row buffer (section 4.1.4) is adequate.

```

foreach species in selectedSpecies:
  open output map using species' name
  foreach site in landscape.AllSites:
    if site.IsActive:
      if site.SharesData:
        blockLocation = site.BroadScaleLocation
        if site.LocationInBlock = (1,1):
          age = computeMaxAge(cohorts[site][species])
          buffer[blockLocation.Column] = age
        else:
          // already computed age for the block
          age = buffer[blockLocation.Column]
      else:
        age = computeMaxAge(cohorts[site][species])
    else:
      // site if inactive
      age = 0
    pixel.Band0 = age
    write pixel to map
  close map

```

Depending upon the size of landscape and the amount of aggregation possible, the difference in memory usage between a site variable and a block row buffer is very significant. For example, the ecoregion map for the TNC study area is 6,512 rows by 10,177 columns. Therefore, a row block buffer will need memory for 10,178 data elements (see section 4.1.4.1).

The optimal block size for this map, in terms of the maximum data sharing (i.e., the smallest number of unique data indexes), has been determined to be 3-by-3. With this optimal block size, a site variable with one data value shared by all inactive sites will need memory for 7,830,886 data values – nearly 770 times the memory needed for a block row buffer.

6.2 Reclass Extensions

This section applies to both the base Reclass extension and the Biomass Reclass extension. The main processing loop in these extension is:

```
foreach mapDefinition in map definitions:
  open output map using map's name
  forestTypes = map definition's forest types
  foreach site in landscape.AllSites:
    if site.IsActive:
      forestType = computeForestType(site, forestTypes)
    else
      forestType = 0
    pixel.Band0 = forestType
    write pixel to map
  close map
```

6.2.1 Two Site Sizes

As with the Age Output extension, this design option can preserve the functionality of this main loop. The source code will produce the correct maps with a dual-scale landscape. However, it will suffer the same performance issue as described with the Age Output Extension.

6.2.2 Fine-scale Data Sharing

As with the Age Output extension, this design option can preserve the functionality of this main loop. The source code will produce the correct maps with a dual-scale landscape. However, it will suffer the same performance issue as described with the Age Output Extension.

7 Succession

This chapter examines the design options in relation to succession in LANDIS-II – the existing succession extensions and the component (library) that they share.

7.1 Succession Component (Library)

- Sufficient Light – Operates at the fine scale because the shade at a site is based on the cohorts present, and each small site has its own set of cohorts.
- Establishment – Operates at the fine scale, since each small site is associated with a particular ecoregion.

These two conditions affect various forms of reproduction.

7.1.1 Seeding

Mature Present – At large scale: a species is present at a large divided site if it's present at least one of the small sites.

8 Disturbances

This chapter examines the design options in relation to various existing LANDIS-II disturbances extensions.

How does dual-scale impact the disturbance interfaces in the cohort libraries, and the implementation of these interfaces in various disturbance extensions?

8.1 AgeCohort.IDisturbance Interface

How does this interface, in particular its CurrentSite property, work in regards to Fine-scale Data Sharing when a whole block of active sites is disturbed?

9 Wind Extension

This chapter examines the design options in relation to the wind disturbance extension.

9.1 Event Spreading

After the extension determines that a wind event starts at an active site, it spreads the event neighboring sites using this algorithm:

```
// sitesToConsider : list of neighboring sites that the
//                  event might spread to.
// getNeighbors : determines which neighbors of a site that
//                  an event may spread to. 9 neighbors are
//                  examined: the 8 nearest plus one neighbor
//                  in the second ring that lies down wind
//                  from the site. The function uses the
//                  event's wind intensity to stochastically
//                  determine which of the 9 sites are
//                  returned.
event.Area = 0
sitesToConsider = { initialSite }

while event.Area < event.DesiredArea and
      sitesToConsider is not empty:

    selectedSite = randomly selected site in sitesToConsider
    remove selectedSite from sitesToConsider
    add selectedSite to the event's list of sites
    siteVars.Event[selectedSite] = event

    if selectedSite.IsActive:
        damage cohorts[selectedSite]
        compute siteSeverity
        siteVars.Severity[selectedSite] = siteSeverity
        if siteSeverity > 0:
            sitesDamaged += 1

    event.Area += landscape.CellArea
    if event.Area < event.DesiredArea:
        for each neighbor in getNeighbors(selectedSite):
            if neighbor is not in sitesToConsider and
               siteVars.Event [neighbor] = null:
                add neighbor to sitesToConsider
```

9.1.1 Fine-scale Data Sharing

If an event spreads to an active site that shares data, then all the other active sites that share the same data index are also added to the event. In other words, if an event spreads into a block of active sites sharing data, then all the block's sites are added at once to the event. As a

consequence, the event's area may exceed the desired area by several cell sizes.

In the case where an event starts at an active site in a block of active sites sharing data, then all the block's sites are added together to the event.

```
// sitesToConsider : (see comment in previous section)
// getNeighbors : (see comment in previous section)
event.Area = 0
sitesToConsider = { initialSite }

while event.Area < event.DesiredArea and
  sitesToConsider is not empty:

  selectedSite = randomly selected site in sitesToConsider
  remove selectedSite from sitesToConsider
  add selectedSite to the event's list of sites
  siteVars.Event[selectedSite] = event

  if selectedSite.IsActive:
    damage cohorts[selectedSite]
    compute siteSeverity
    siteVars.Severity[selectedSite] = siteSeverity
    if siteSeverity > 0:
      if selectedSite.SharesData:
        sitesDamaged += landscape.SitesPerBlock
      else:
        sitesDamaged += 1

  if selectedSite.SharesData:
    event.Area += landscape.BlockArea
  else:
    event.Area += landscape.CellArea
  if event.Area < event.DesiredArea:
    if selectedSite.SharesData:
      // Use a variation of the getNeighbors function
      // that retrieves the fine-scale sites around a
      // block (i.e., broad-scale site).
      blockLocation = selectedSite.BroadScaleLocation
      neighbors = getNeighbors(blockLocation)
    else:
      neighbors = getNeighbors(selectedSite)
    for each neighbor in neighbors:
      if neighbor is not in sitesToConsider and
        siteVars.Event[neighbor] = null:
        add neighbor to sitesToConsider
```

9.2 Severity Maps

9.2.1 Two Site Sizes

How is site severity determined for a large divided site?

10 Harvest Extension

To do.