12-2008

Two Highly Diverse Studies in Computing: A Vitruvian Framework for Distribution and a Search Approach to Cancer Therapies

Brian G. Smith
Utah State University

TWO HIGHLY DIVERSE STUDIES IN COMPUTING: A VITRUVIAN FRAMEWORK FOR DISTRIBUTION AND A SEARCH APPROACH TO CANCER THERAPIES

by

Brian Smith

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Stephen W. Clyde Major Professor	Nicholas S. Flann Committee Member
Scott Cannon Committee Member	Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

Two Highly Diverse Studies in Computing: A Vitruvian

Framework for Distribution and A Search

Approach to Cancer Therapies

by

Brian Smith, Master of Science

Utah State University, 2008

Major Professor: Dr. Stephen W. Clyde
Department: Computer Science

Solid cancer tumors must recruit new blood vessels for growth and maintenance. Discovering drugs that block this tumor-induced development of new blood vessels (angiogenesis) is an important approach in cancer treatment. However, the complexity of angiogenesis and the difficulty in implementing and evaluating medical changes prevent the discovery of novel and effective new therapies. This paper presents a massively parallel computational search-based approach for the discovery of novel potential cancer treatments, using a high fidelity simulation of angiogenesis. Discovering new therapies is viewed as multi-objective combinatorial optimization over two competing objectives: minimizing the medical cost of the intervention while minimizing the oxygen provided to the cancer tumor by angiogenesis. Results show the effectiveness of the search process in finding simple interventions that are currently in use and more interestingly, discovering some new approaches that are counterintuitive yet effective.

Distributed systems are becoming more prevalent as the demand for connectivity increases. Developers are faced with the challenge of creating software systems that meet these demands and adhere to good software practices. Technologies of today aid developers in this, but they may cause applications to suffer performance problems and require developers to abandon basic software concepts, such as modularization, performance, and maintainability. This work presents the Vitruvian framework that provides solutions to common distribution goals, and distributes applications using replication and transparency at varying stages of application development.

(70 pages)

ACKNOWLEDGMENTS

To my wife and son, who fill my life with light, hope, and love.

To Dr. Clyde and Dr. Flann, who are bright, humble, and honorable men.

To all those who encouraged and believed in me.

Brian Smith

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis consists of two publications in disparate areas of computer science. With multiple authors on each publication, it is necessary to clarify my specific contributions.

The first publication, which comprises Chapter 1, "An Orthogonal Approach to Distribution: An Introduction to the Vitruvian Framework," introduces the Vitruvian framework and explains how distribution can be accomplished through orthogonal services in a service-oriented architecture. I started development of the Vitruvian framework in March of 2007, and have been the sole developer. This work has since been verified and validated under the direction of Dr. Stephen Clyde. Dr. Clyde assisted with the formulation of examples and the writing of the paper.

The primary contribution of this work is the identification of a method that provides distribution with replication while maintaining location and access transparency. These contributions also include providing a foundation for the continued research of replication strategies, object migration, object fragmentation, reliability, cross-platform integration and cooperation, and distributed service-based software systems.

The second publication, which comprises Appendix 1, "Discovering Novel Cancer Therapies: A Computational Modeling and Search Approach," explains a method for finding possible therapies for cancer by using a search engine. This was accomplished with a high-fidelity computational model to generate a metric for the search engine, which probed the search space for parameters that reduced the delivery of oxygen to a tumor. Arthur W. Mahoney and Dr. Nicholas S. Flann researched and implemented the search engine. I researched and implemented the computational model under Dr. Flann's direction. Gregory J. Podgorski verified the biological fidelity of the model and results.

The preliminary work for the computational model began in *CS6890, ST: Computational Biology*, offered Summer 2007 at Utah State University, with Dr. Flann as the principle instructor. In this class, I led a team of four researchers that prototyped a model using COMPUCELL. My team, which consited of Ranjitha Dhanas, Kamath Puru, Vineela Kalluru, and myself, displayed this work was displayed as a poster presentation at Utah State University. This prototype demonstrated angiogenesis emerging from a VegF diffusion gradient.

After the class, and under the continued direction of Dr. Flann, I extended this work to create an enhanced model that met the biological fidelity required by the search engine. I augmented the model to include the ECM fibers and stromal cells, which are necessary to facilitate the emergence of anastomosis (loop formation in blood vessels); this was implemented shortly after the preliminary work was presented to the Breast Cancer Research Group at Huntsman Cancer Research Center, wherein anastomosis was identified as a crucial and missing part of the model. Once the model included anastomosis, I attempted to correctly identify the vessel loops formed as a result of this process, which eventually led to a pressure diffusion model that governed the secretion of oxygen at the cell level. I extended the model to include oxygen diffusion, which is the secretion of oxygen from existing vasculature and endothelial cells contained in vessel loops. Finally, I added the tumor cells to the model, and the resulting metric of oxygen was computed as the amount of oxygen available to the tumor. My primary contribution to this work, under the direction of Dr. Flann, was the research and implementation of the computational model exhibiting angiogenesis and anastomosis, which provides the oxygen metric utilized by the search engine described in the publication. From a computer science perspective this first paper enhances the knowledge of complex systems created by emergent behaviors and properties.

Those involved in both of these publications perceive them to be avenues to future research, and I am pleased to have been a contributing factor to them.

CHAPTER 2
AN ORTHOGONAL APPROACH TO DISTRIBUTION: AN INTRODUCTION TO THE VITRUVIAN FRAMEWORK[1]

Abstract

Distributed systems are becoming more prevalent as the demand for connectivity increases. Developers face the challenge of creating software systems that meet these demands, while still trying to achieve basic quality goals, such as good modularization, performance, and maintainability. The challenge is even greater when requirements for distribution are introduced late in the development cycle. This paper introduces a development framework, called Vitruvian, that allows programmers to create software systems with class hierarchies that are close to the problem domain and independent of distribution decisions. Then, at any point in the development cycle, developers can declare what is distributed, and they can do so at virtually any level of granularity from high-level objects down to individual attributes. To minimize impact on the development life cycle, Vitruvian supports access and location transparency, plus a wide range of replication strategies.

2.1 Introduction

The world is rapidly becoming more connected, causing an increase in demand for software systems that are more interoperable and distributed, and that provide better opportunities for collaboration. The shift from stand-alone, centralized systems to grid-based applications presents challenges to software developers on three fronts: retro-fitting

1 Co-authored by Brian Smith, and Stephen Clyde of Computer Science Department, Utah State University.

existing applications with distribution capabilities, designing new distributed systems from the ground up, and accommodating new requirements for distribution in mid-development stream. This paper focuses on the latter situation, wherein developers are well into a project and management announces that the product is now going on the "grid."[2] To this end, the current project presents a framework in a way that minimizes developer effort to distribute objects, without compromising functionality, extensibility, good modularization, performance, or maintainability.

In the last few years, there has been an explosion of frameworks, toolkits, papers, and books for developing distributed or grid-based applications. Compilations of such resources are maintained by several worldwide collaborations, including CERN [1], the Globus Alliance [2], Unicore [3], and GridBus [4]. The most predominate technologies focus on abstractions of communications, typically through remote procedure calls (RPCs), remote method invocation (RMIs), or web services. In general, these *communication-centric* technologies improve the development of new massively distributed applications without comprising functionality or extensibility. However, when it comes to adding distribution to an existing application, they suffer from three common problems related to modularization, performance, and maintainability.

The first problem comes from having to restructure the class hierarchies of an application when introducing distribution. With communication-centric technologies, the structure of the application depends heavily on decisions about which services are local

2 Here, the term "grid" does not refer to a specific technology; rather, it refers to the general notion of a distributed, service-oriented environment.

and which are remote. These decisions need to be made early in the design process to minimize development efforts.

For example, consider a photo-library application that allows users to browse large image collections by date, photographer, or subject. An initial analysis, without any consideration to distribution, would reasonably identify *Photo Collection, Photo, Original Image, Thumbnail Image,* and *Photographer as key problem domain objects* (see Figure 2-1).

A subsequent design could parallel this conceptual model rather closely. Assume that the system needs to be distributed using a communication-centric framework, such as .NET Remoting [5]. Remote objects in .NET must specify *MarshalByRefObject.* Since all of the key objects in this application can be either remote or local, the developer must extend the design to include a parallel class structure for remote objects. To allow some transparency for remote and local objects, the developers could introduce interfaces for the original classes. Figure 2-2 shows a snippet of a design for this application consistent with principles of .NET Remoting. The two original classes of *Photo* and *Photographer* plus their association have been replaced by five classes, two interfaces, and two associations. Not only is the design more complex, it drifts away from the underlying conceptual model, weakening the program's modularization and maintainability.

A second problem occurs when distribution technologies try to provide access and location transparency without replication. Since transparency frees developers from having to think about which objects are local or remote, operations and the data that they use can end up unknowingly and unnecessarily distanced. Simply substituting remote

Figure 2-1. Photo library class structure.

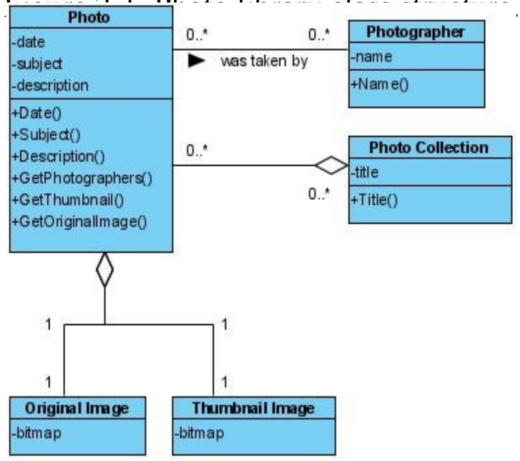
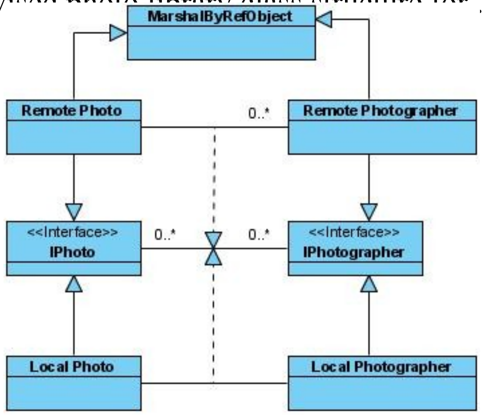


Figure 2-2. Revised photo library class structure for .NET remoting.



service invocations in place of local method calls can devastate an application's performance. Object replication can solve this problem by keeping copies of data local to computationally intense or time-sensitive operations. Unfortunately, in communication-centric frameworks, the addition of replication to a design adds yet another dimension of complexity and drives it further away from the original conceptual model.

A third problem deals with performance and maintainability in the presence of changing runtime traffic or load patterns. For example, if object X initially resides on node A, and after a while, the objects on node B start accessing X more than objects local to A do, X should migrate to B to reduce network traffic and improve throughput. However, with communication-centric frameworks, distribution decisions tend to be static, unless the developers have explicitly added dynamic replication and migration capabilities – yet another level of complexity that can reduce understandability and maintainability.

To address these problems in the context of late distribution requirements, a framework must satisfy the following goals:

- Minimize changes to existing class hierarchies
- Encapsulate distribution decisions (location transparency)
- Simplify access to distribution objects (access transparency)
- Allow developers to make distribution decisions at virtually any stage of development
- Allow developers to declare or adjust replication and migration strategies without altering the core design

- Automatically tune performance by either replicating or migrating objects in response to changing traffic and load patterns, according to declared replication and migration strategies.

2.2 Background and Related Work

Technologies and frameworks that aid developers in creating distributed applications are not new. One early solution was Orca [6], which introduced a language that used replication and migration to overcome performance costs of remote-procedure calls. CORBA and DCOM emerged in the mid 1990's with the promise of providing open standards for distributed objects [7, 8, 9]. Today, J2EE [10, 11], Webservices [12], Java RMI [13, 14], and .NET Remoting [5] are some of the most common tools of the trade, each using forms of remote-procedure calls but with differing abstractions for building distributed applications.

Many distribution methods and strategies build on these technologies, but they are subject to the same underlying issues. For example, one method, suggested by [15], augments the Java RMI distribution model by using aspects to add access and location transparency, creating a seamless integration of local and remote objects. The class hierarchy changes are minimized by post-compiling distribution into the application at the very end of the development cycle. This satisfies the first three goals listed above but not the last three goals, because the underlying distribution model does not offer any control over replication or migration. This deficiency can actually make the transparency a detriment to the application's performance. Consider the following function, which draws an object in a continuous loop to create a simple animation.

```
public void DisplayChangingShape(Shape A)
{
  // continuously draw the changing A
  while (true)
  {
    // draw A using A’s color
    Draw(A, A.Color);

    // Do other things, such as change
  // the state of A and sleep a certain
    // amount of time to control the
    // frame-rate
    ...(snip)...
  }
}
```

When *A* is a local object, accessing its color is a fast operation. However, if *A* is a remote object, *A.Color* is similarly a remote-procedure, requiring a round-trip message across a network with communication delays that will result in unpredictable and unsatisfactory performance. The problem is not that location or access transparency are bad, but that by themselves they do not represent a complete solution to the basic distribution goals listed above.

Transparency must be coupled with replication to ensure application performance. Replication increases performance by bringing data and operations as close together as possible, which decreases access costs [6] and eliminates performance side effects otherwise hidden by transparency.

When using communication-centric technologies, the speed of the network carrying the round-trip message governs the performance of the distributed application. This model of distribution only yields significant performance increases by making the transmission speeds faster, which means that the developer cannot directly affect performance. .NET Remoting and Java RMI address this problem by introducing asynchronous communication patterns [5, 13], but this clutters the object interface and undermines transparency.

2.3 Overview of the Vitruvian Framework

Vitruvian is a distribution framework that successfully addresses the goals listed above by supporting:

- Access and location transparency
- Control over the granularity of the distribution units
- Distribution with replication
- A rich suite of replication strategies, plus the ability for developers to write their own
- Control over which replication strategy each distribution unit uses

In essence, the Vitruvian framework gives developers the ability to make and encapsulate all major distribution decisions independent of the application's core design. In doing so, it incorporates the three principles introduced by the Roman architect Vitruvius in the earliest work on architecture, De architectura [16]. These principles are firmitatis (durability), utilitatis (utility), and venustatis (beauty). In the context of software systems, providing durability means ensuring correct, reliable, and secure execution in the face of change. Changes can come from new requirements, modifications to the design or code, or changes to the underlying platform or related software systems. Utility relates to reuse at multiple levels. Systems with good utility can be leveraged through interoperability, design reuse, or code reuse. Although there are varying opinions on what comprises software “beauty,” symmetry and simplicity play a key role in creating elegant systems.

The Vitruvian framework is a service-oriented architecture (SOA), that focuses on encapsulating functional units of software (i.e., services) to improve reuse, parallel development activities, and rapid product packaging or repackaging in both stand-alone and distributed applications. The heart of the framework is its service registry, which facilitates the location and initialization of local and remote services. Section 2.3.1 explain this component in more detail. Because it is a SOA, the Vitruvian framework works best with an execution pattern that follows well understood object-oriented principles, such as low coupling, high cohesion, and data encapsulation [17]. Section 2.3.2 describes one common execution pattern for Vitruvian and shows how the service registry is populated, initialized, and utilized. Section 2.4 shows a staged integration of the framework into a sample application.

2.3.1 Service Registry

The Vitruvian framework includes a symmetrical singleton, called the service registry, that provides access to the services comprising an application. A service implements a simple interface that

- Gives the service an identity
- Defines symmetrical methods for starting and stopping the service.

A service's identity is globally unique. The current implementation relies on .NET's guide to create that identity. Service identities are one way, but not the only way, to look up services in the service registry; the registry also supports look-up by service type.

The starting and stopping methods allow the service registry to automatically manage the service life lines so programmers do not have to. Typically, the start method resolves interservice dependencies (through the registry), and the stop method releases resources.

Services follow natural boundaries of encapsulation that overflow into classes that have smaller footprints and are easier to understand and maintain.
Although the developer builds most of the services for an application, Vitruvian includes a number of useful orthogonal services [18], like logging, persistence, and session management. It even includes a plug-in service for dynamically loading extension classes, which in turn could instantiate and register new services at runtime.

2.3.2 General Execution Pattern

The Vitruvian framework uses an execution pattern that centers on the service registry as the hub for object discovery. Using this pattern, an application first populates the service registry with services and then initializes the services through the registry. It looks up services and executes until an exit point is reached. Finally, the application cleans up the service registry, which in turn stops all remaining services. This pattern increases a developer's ability to maintain and understand a software system, because the functionality is encapsulated in the set of services contained in the service registry.

Below is a typical main routine for a program based on the Vitruvian framework.

```
static class Program
{
  static void PopulateServices()
  {
    XmlFramework.Deserialize(“myService.xml”,
                               ServiceRegistry.Services);
  }

   static void Main()
   {
    PopulateServices();
    ServiceRegistry.Init();

    // Start up a form window & block
    UIService form = ServiceRegistry.GetService<UIService>();
    Application.Run(form.AppForm);
    ServiceRegistry.Cleanup();
   }
}
```

In this example, the *Main* method first calls *PopulateServices*, which loads all the application's services from an XML file. For brevity, we only show a snippet of the XML file below.

```
<item type="Vitruvian.Windows.UIService, Vitruvian.Windows">
  <property name="MyAppForm" type="MyApp.MainForm, MyApp”>
    <property name="Text" value="My Form” />
    <property name="Width" value="800"/>
    <property name="Height" value="600"/>
  </property>
</item>
```

This item in the XML file defines a UIService consisting of a window of type *MainForm*, an initial title of “My Form”, and size of 800x600.

After populating the service registry, the main routine initializes the services by calling the registry's *Init* method. At this point, the registry is ready for use. Next, the main routine looks up a UIService in the registry and passes its form to the *Run* method of the *Application,* a .NET component. This method blocks until the form window closes. Finally, the main routine stops all of the services by calling the *Cleanup* method of the registry.

Although Vitruvian supports other execution patterns, this approach encapsulates the essence of the application in the services. One of its benefits is that it offers a recursive view of the software system with multiple levels of detail. In particular, it enumerates the services in the service registry, thus providing an overview of the system. For example, consider the photo management application introduced above. Listing the services in the registry allows one to discover that the application supports collections, photographs, photographers, thumbnails, and original images. By looking at the interfaces for this services, one could readily discover the above components' capabilities and relationships. This technique introduces new developers to the functionality of the application, and even helps overcome communication barriers between software developers and management.

2.4 Staged Integration of Vitruvian into a Sample Application

This section illustrates a step-wise integration of the Vitruvian framework into an ant colony simulation. Section 4.1 describes this basic application and shows the key parts of a nondistributed implementation, which is representative of an SOA that a typical developer might create. Section 4.2 shows a conversion of the stand-alone application into a SOA, using the Vitruvian framework. Section 4.3 uses additional, built-in services of the framework to distribute the application, and Section 4.4 applies various replication strategies to tune the application's performance.

This four-step sample scenario mimics a typical, yet simple, development cycle and demonstrates the migration of a centralized legacy system to a distributed system.

The ant simulation is relatively simple, but it still provides sufficient opportunity to illustrate the fundamental distribution techniques.

2.4.1 Step 1: Local Ant Colony Simulation

The stand-alone application created in this first step focuses on the movement and behavior of ants. An ant is part of a colony and can explore the ground around the colony's nest. The ant's goal is to locate food, transport it to the nest, and by using pheromones, direct other ants to the location of the food source. Multiple colonies compete in the gathering of food, but they are not aggressive towards each other. Figure 2-3 shows classes and association that model these simple concepts.

The main program creates the ground and a single ant colony, and populates it with ants. We run the simulation for a set period of time, while continuously allowing the ants to move. The resulting application is concise and true to its purpose, namely, the movement and behavior of ants.

```
static class Program
{
  // main entry point
  static void Main()
  {
    Ground ground;
    Colony colony;

    // Create and setup the ground
    ...(snip)...

    // Create and populate the colony
    ...(snip)...

    // Add the colony to the ground
    ground.Add(colony);

    // Run simulation for 10,000 steps
    for (int i = 0; i < 10000; i++)
    {
      // Move the ants
      foreach (Ant ant in colony)
```

```
            ant.Move();

          // Display the ground, ants, etc.
          Display();
        }
      }

      // output a simple display of the
      // ground, and for each colony,
      // each ant’s position on the ground
      static void Display()
      {
        ...(snip)...
      }
    }
```

2.4.2 Step 2: Converting to SOA

This step uses the Vitruvian framework to create better encapsulations, provide an easy way to extend the application for multiple colonies, and facilitate the integration of a replaceable user-interface. The main functional units, or services, of the application are the ground, the colony, and the user-interface. The colony now encapsulates the movement of the ants by creating and managing a new thread. The ground locates colonies by using the service registry, which now allows multiple colonies just by adding new colony services to the service registry. Figure 2-4 gives an overview of the program’s services.

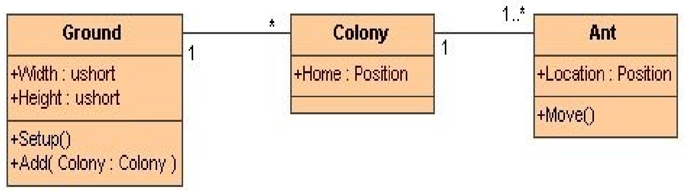


Figure 2-3. Object relationships in ants simulation.

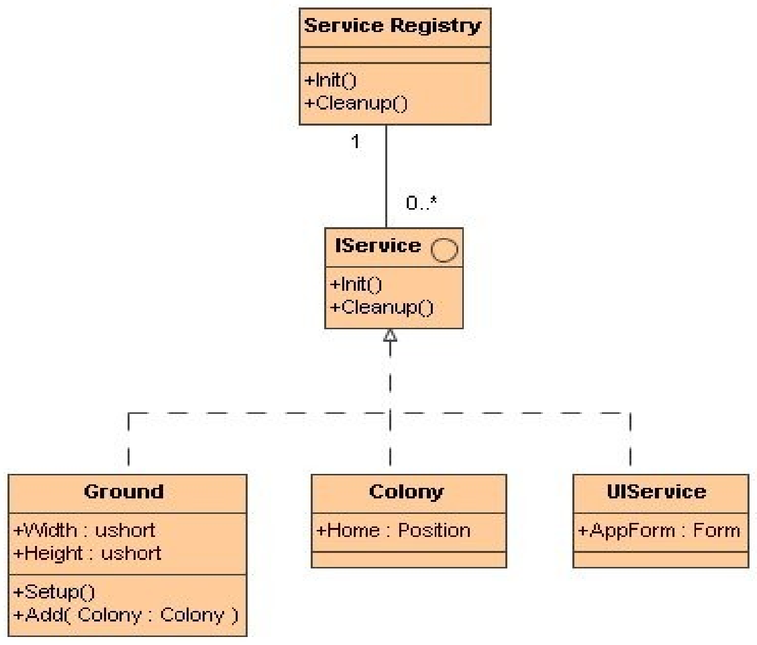


Figure 2-4. Services in ants simulation.

The XML file defining these services is shown below.

```
<item type="Ants.Ground, Ants">
   <property name="Width" value="36" />
   <property name="Height" value="36" />
</item>
<item type="Ants.Colony, Ants">
   <property name="NumAnts" value="15" />
   <property name="Color"
value="Black" />
</item>
<item type="Ants.Colony, Ants">
   <property name="NumAnts" value="10" />
   <property name="Color" value="Red" />
</item>
<item type="Vitruvian.Windows.UIService, Vitruvian.Windows">
   <property name="AppForm" type="Ants.DisplayForm, Ants">
     <property name="Text" value="Ants"/>
     <property name="Width" value="760"/>
     <property name="Height" value="800"/>
   </property>
</item>
```

The developer nextapplies the execution pattern described above to initialize all of the services and open the *AppForm.* The user-interface displays the ground, position of the colony nests, and positions of the ants by doing three things: finding the ground in the service registry, enumerating the colonies and ants, and drawing the objects.

Figure 2-5 shows the sequence of events resulting from the execution pattern described above, and the interaction of the services in the simulation.

2.4.3 Step 3: Distributing the Simulation

Using the Vitruvian framework, this step distributes colonies by adding the distribution service to the service registry and by marking the colony with attributes that indicate the replication strategy to use when synchronizing data. The ant object contains

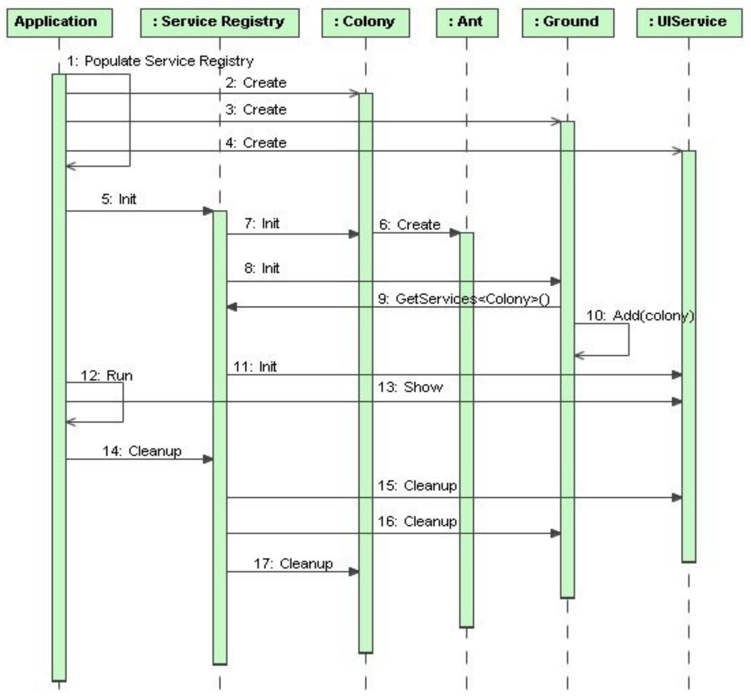


Figure 2-5. Execution sequence in ants simulation.

attributes specifying synchronization patterns because it can be reached through the colony. For simplicity in demonstration, this example uses a classic client / server distribution model, but the concepts are not restricted to this model.

```
public class Ant
{
  [SyncPattern("RPC")]
  public Position Location
  ...(snip)...

  [SyncPattern("RPC")]
  public Colony Colony
  ...(snip)...
}

public class Colony
{
  [SyncPattern("RPC")]
  public Position Home
  ...(snip)...

  [SyncPattern("RPC")]
  public List<Ant> Ants
  ...(snip)...
}
```

So far, the only synchronization pattern specified for the ant and colony is the RPC pattern which, as previously discussed, is the pattern prominent distribution technologies use. The RPC synchronization pattern makes remote calls to the remote objects every time the property or method is accessed. Under this model, the application is unresponsive, because the drawing loop of remote colonies and ants is too slow. It is obvious that other synchronization patterns are more appropriate, such as communicating the colony's nest only once because it does not change over the life of the application.

2.4.4 Step 4: Tuning Performance

The Vitruvian framework gives the developer the ability to easily tune the application by changing the synchronization patterns, and thus control the replication strategy. In this step, changing the RPC pattern to other synchronization patterns improves the example application performance. The tuned example follows:

```
public class Ant
{
   [SyncPattern("Poll")]
   public Position Location
   ...(snip)...

   [SyncPattern("Constant")]
   public Colony Colony
   ...(snip)...
}

public class Colony
{
   [SyncPattern("Constant")]
   public Position Home
   ...(snip)...

   [SyncPattern("Constant")]
   public List<Ant> Ants
   ...(snip)...
}
```

The two new synchronization patterns used are poll and constant. Poll asks for the remote value of the property at a given frequency, while constant is communicated across the network only once. The performance of the application increases, because these method calls return a local value when invoked.

The list of ants in the colony is a constant pattern, which means that once the colony of ants is populated, modifications will not be replicated. It is easy to imagine how changing design requirements could mandate that the distributed list of ants reflect these changes, which only requires changing the synchronization pattern. The Vitruvian framework empowers the developer to manage changing distribution requirements throughout the application's development cycle by facilitating the extension of the replication strategy, while improving application performance.

2.5 Distribution in the Vitruvian Framework

This final section describes the details of the orthogonal distribution service. It meets the distribution goals stated in Section 2.1 by integrating replication strategies and synchronization patterns in a way that supports location and access transparency without compromising performance. The distribution service connects to other applications, exchanges services, and provides access and location transparency to remote services and the objects reached through them. The distribution service acts orthogonally to other services in the application, and is the only difference between the stand-alone version of an application and the distributed version. The service can be added or removed at any stage of development, or even after deployment, which provides extreme flexibility in distribution decisions.

The following requirements drove the design of the Vitruvian distribution service:

- Connects to other applications
- Discovers the services contained in the remote service registry
- Provides a mechanism to replicate and synchronize the services and objects discovered through the remote application
- Maintains access and location transparency to remote services and objects
- Handles errors induced by remote connections

The distribution service fulfills the aforementioned requirements unobtrusively by generating dynamic types and instantiating proxy objects from those types the first time an object crosses the application boundary. The dynamic type is a specialization of the original object's type. This ensures access transparency because the application does not differentiate between the new type and the base type. The new type dynamically delegates the invocation of the method and properties to the specified synchronization pattern.

The Vitruvian framework accomplishes this during the encoding process by exchanging objects for object markers, which consists of two things: a unique identifier for the object, and the object's type. The session gives a message to the encoder, which encapsulates encoding and decoding of messages. The serializer contained by the encoder replaces objects for object markers, and then serializes the object marker instead of the object; the serializer does not exchange primitive types for object markers. The object broker that is utilized by the serializer manages the mappings between objects, object ids, and object markers (see Figure 2-6).

The encoder reverses the process when decoding a message, and exchanges object markers for objects with an additional step that creates proxies for unknown object ids (see Figure 2-7). When the deserializer locates an object marker, it deserializes the marker and requests the object, using the identifier in the marker, from the broker. When the identifier is unknown to the broker, the deserializer uses the proxy generator to create a proxy that derives from the type specified in the object marker, which is then added to the object broker to be used in subsequent exchanges. The proxy generator emits a new type that derives from the object's type, caches the new type for future use, and uses the new type to instantiate a proxy. The new emitted type delegates invocations of methods and properties to the synchronization pattern, which is specified by the developer through attributes: the entire distribution process is encapsulated in the

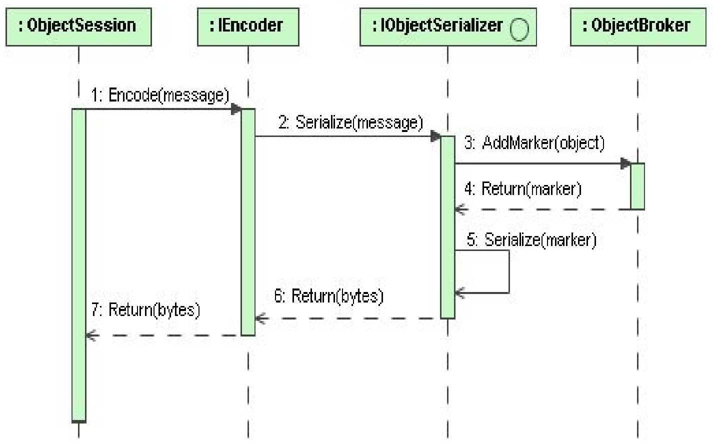


Figure 2-6. Object → object marker exchange.

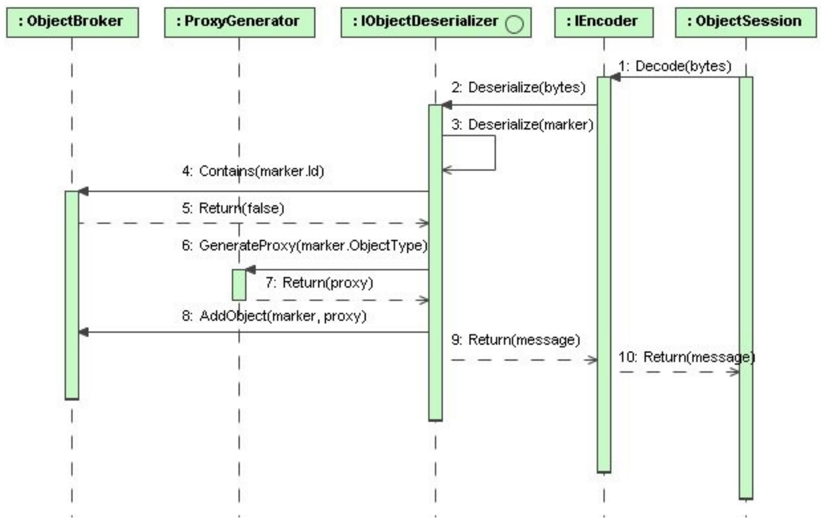


Figure 2-7. Object marker → object proxy exchange.

distribution service, and the synchronization pattern provides a way for the developer to control and extend the replication strategy.

The synchronization pattern extends the distribution framework and encapsulates a replication strategy, which completes the components necessary to provide distribution and replication with transparency. The developer gains the freedom to control the replication strategy either by reusing previously implemented patterns or by writing patterns specific to the application's needs. The synchronization pattern has access to the proxy object, the session, and the service registry (See Figure 2-8).

The distribution process in the Vitruvian framework includes a few simple steps: add the distribution service to the service registry, mark the methods and properties of objects with an attribute that specifies the desired synchronization pattern, and possibly write custom synchronization patterns. These steps must be done before distributing the objects, but they can be done at any stage of development.

2.6 Summary

The Vitruvian framework provides a foundation of orthogonal services that enhances application development, and gives a consistent development model that a range of applications, from stand-alone to massively distributed. An orthogonal service

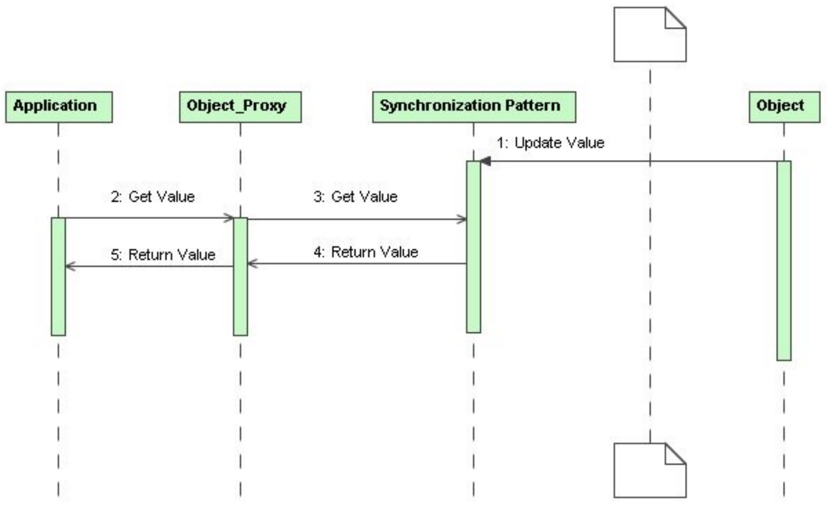


Figure 2-8. Synchronization pattern override.

that facilitates replication and transparency provides distribution. Synchronization patterns encapsulate replication strategies, of which some have been identified, but there are many more to discover and analyze. The Vitruvian framework provides mechanisms to dynamically adjust to changing network loads through object fragmentation and object migration, but this has not been thoroughly explored or implemented as of yet. The granularity of the replication strategy is currently applied to methods and properties, but it could also be applied to entire objects and namespaces. Another readily achievable objective is port the Vitruvian framework to multiple platforms, thus providing cross-platform distribution of services.

References

[1] "LCG - LHC Computing Grid." URL: http://www.cern.ch/lcg, last access on 2008/4/2008

[2] I. Foster, "Globus Toolkit version 4: Software for service-oriented systems," *IFIP International Conference on Network and Parallel Computing*, vol. 3779. pp. 2-13.

[3] M. Romberg, "The UNICORE Grid infrastructure," *Sci. Program*, vol. 10, no. 2, pp. 149-157.

[4] R. Buyya, and S. Venugopal, "The Gridbus toolkit for service oriented grid and utility computing: an overview and status report," *Grid Economics and Business Models*, GECON, 23 April 2004, pp. 19-66.

[5] ".NET Remoting Overview," *.NET Framework Developer's Guide,* http://msdn2.microsoft.com/en-us/library/kwdt6w2k%28VS.71%29.aspx, access on 4/24/2008

[6] H.E. Bal, M.F. Kaashoek, A.S. Tanenbaum, "Orca: A Language for Parallel Programming of Distributed Systems," *IEEE Transactions on Software Engineering*, vol. 18, no. 3, Mar 1992, pp. 190-205.

[7] R. Orfali, D. Harkey, and J. Edwards, *Instant CORBA*, John Wiley & Sons, 1997

[8] Catalog of OMG CORBA/IIOP Specifications, Object Management Group, http://www.omg.org/technology/documents/corba_spec_catalog.htm, April, 2008

[9] M. Horstmann and M. Kirtland, "DCOM Architecture," *Microsoft Developer Network*, Microsoft, July 23rd, 1997

[10] K. Walrath, *The J2EE Tutorial*, Addison-Wesley, 2002

[11] D. Alur, J Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, Sun Microsystems Press, 2001

[12] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "Web Service Description Language", W3C, http://www.w3.org/TR/wsdl/

[13] K. Falkner, P. Coddington and M. Oudshoorn. "Implementing Asynchronous Remote Method Invocation in Java," *Proc. Parallel and Real-Time Systems (PART'99)*, Melbourne, December, 1999, http://citeseer.ist.psu.edu/falkner99implementing.html

[14] D. Curtis, "Java, RMI, and CORBA," Object Management Group, 1997, http://www.omg.org/news/whitepapers/wpjava.htm

[15] M. Ceccato, and P.Tonella, "Adding distribution to existing applications by means of aspect oriented programming," *Fourth IEEE International Workshop*, 2004, pp. 107-116.

[16] Vitruvius: De Architectura, translated by Frank Granger; Cambridge (Mass.): Harvard University Press, 1931.

[17] S. Schach, *Object-oriented and Classical Software Engineering*, McGraw Hill Companies, Inc., 2007.

[18] C. Ghezzi, and M. Jazayeri, *Programming Language Concepts*, John Wiley, New York, NY, 1982.

CHAPTER 3

CONCLUSION

This thesis incorporates two highly diverse studies in computing. However, they are two studies that may well enhance each other in the future. The realization of individualized cancer therapies may someday require the unification of these disparate computer science topics. The model and search approach used to discover and validate therapies provides a process that may prove useful at an individual scale, while the distribution framework harnesses the needed computational power.

The computational model and search approach utilizes a massively parallel algorithm, but it is currently limited to the processing power of a single machine. The solutions that were discovered in this project would benefit from massively distributing the application to explore the search space, and to increase the number of model runs with each parameter set. The Vitruvian framework reduces the complexity of distribution and, hence, would be a natural candidate for this task.

The same model and search paradigm could be applied to individualized therapies. This requires that the initial conditions are representative of the individual, after which known general solutions could be tested to determine the effectiveness of the therapy for that individual. Such an approach would require many model runs to increase the confidence of the suggested therapy, and the timeliness of the suggested therapy would be crucial. A system capable of these computational requirements is most likely realized in a massively distributed computing environment.

These individual topics are diverse and have significant application in their respective areas. This project ended before unifying these topics, which leaves this possibility as future work.

APPENDICES

Appendix 1:

DISCOVERING NOVEL CANCER THERAPIES: A COMPUTATIONAL MODELING AND SEARCH APPROACH[3]

Abstract

Solid cancer tumors must recruit new blood vessels for growth and maintenance. Discovering drugs that block this tumor-induced development of new blood vessels (angiogenesis) is an important approach in cancer treatment. However, the complexity of angiogenesis and the difficulty in implementing and evaluating medical changes prevent the discovery of novel and effective new therapies. This paper presents a massively parallel computational search-based approach for the discovery of novel potential cancer treatments using a high fidelity simulation of angiogenesis. Discovering new therapies is viewed as multi-objective combinatorial optimization over two competing objectives: minimize the medical cost of the intervention while minimizing the oxygen provided to the cancer tumor by angiogenesis. Results show the effectiveness of the search process in finding simple interventions that are currently in use and more interestingly, discovering some new approaches that are counter intuitive yet effective.

A.1 Introduction

Cancer has become the leading cause of death for Americans between the ages of 40 and 74 [6]. With survival rates at approximately 50%, new, more effective therapeutic treatments are urgently needed. However, given the complexity of cancer development,

3 Co-authored by Arthur Mahoney, Brian Smith, and Nicholas Flann of Computer Science Department, Utah State University, and Gregory Podgorski of the Biology Department and Center for Integrated Bio Systems, Utah State University.

discovering interventions through purely medical and experimental techniques is woefully slow. This paper introduces a massively parallel computational search-based technique that shows promise in rapidly discovering potential therapeutic interventions.

Many kinds of cancers involve solid tumors that during their development must recruit new blood vessels from the host tissues in order to grow and remain viable. These new blood vessels sprout from existing vessels and grow towards the tumor to provide needed nutrition and oxygen that enable the tumor to grow rapidly. Blocking this process of tumor-induced angiogenesis (the formation of new blood vessels) has been an important approach in cancer treatment [2]. This paper introduces a computational approach to search for novel intervention strategies that disrupt angiogenesis induced by solid tumors. In order to automate the computational search, a high fidelity simulation model of angiogenesis has been developed that is sufficiently abstract to be computationally feasible yet sufficiently detailed to identify specific medically-relevant intervention targets. Running this model correctly simulates the early stages of angiogenesis, when new blood vessels grow towards the tumor, form loops and allow blood to flow, thus secreting oxygen and feeding the tumor.

The simulation system integrates: (a) a cellular Potts model (CPM) that realistically captures mechanisms of endothelial cell growth, cell adhesion, extra cellular matrix (ECM) fiber adhesion and degradation, and tip cell chemotaxis and haptotaxis [5]; (b) a continuous model of vascular endothelial growth factor (VegF) secretion from the tumor, diffusion through the stroma (host tissue), and endothelial cell uptake and activation; (c) a discrete flow model that estimates blood flow through the irregular network of vessels that emerge during angiogenesis; and (d) a continuous model of oxygen secretion from vessel loops, diffusion through the stroma and uptake by the tumor. This model captures behaviors, such as vessel branching, loop formation (anastomosis), progression and termination of tip movement, and activation and growth of new vessels. All these complex behaviors emerge from interactions among the simpler, biologically relevant component mechanisms of the model.

The state of the art in angiogenesis-blocking drugs is reviewed in [11]. Current drugs take a simplistic and reductionist approach to disrupting angiogenesis through interference with the VegF system, which diffuses from the tumor cells to the existing vascular, triggering the formation of new vessel growth and guiding the newly growing sprouts towards the tumor. These drugs either bind to the protein VegF ligand, thereby slowing its diffusion through the supporting tissue and interfering with its receptor binding, or binding to the VegF receptor on the endothelial cells, thus preventing activation and growth. While demonstrating some effectiveness at slowing the progression of the tumor, these drugs target only one obvious component of the angiogenesis process. To identify more effective medical interventions, it is necessary to consider disrupting other component mechanisms of angiogenesis and in addition, combinations of those mechanisms. However, the complexity of angiogenesis presents a multitude of component mechanisms that could be disrupted in multiple ways, presenting a large combinatorial space of possible therapies that is infeasible to search using laboratory-based biological methods. By utilizing a high fidelity simulation of angiogenesis, this large combinatorial space can be efficiently searched and new potential therapies discovered.

The rest of this paper is organized as follows: First, the Monte Carlo-based search engine is described in Section A.2, followed by a description of the cellular Potts-based angiogenesis simulation system in Section A.2. Section A.3 describes the potential effective cancer therapies found by the search engine. Finally, Section 2A4 concludes the paper.

A.2 Search Method

The search engine uses an extension of the COMPUCELL [5] tissue simulation system that takes as input a vector of 23 parameters, denoted as $\vec{P}$, runs a simulation of angiogenesis (blood vessel growth), and outputs a measure of oxygen absorbed by a cancer tumor. Based on literature and simulation studies of standard blood vessel morphologies, each of the 23 parameters has been assigned a normative value, which represents the untreated condition [1]. The normative parameter vector, $\vec{P}_N$, contains the normative values for each parameter and is given in Table A-1. Running the simulation with the normative parameter vector produces the expected progression of blood vessel growth in tissues adjacent to a tumor without treatment.

An effective therapy disrupts the normal progression of blood vessels, thereby reducing the oxygen provided to the tumor and arresting its growth. Each potential therapy is represented as a vector of deviations $\Delta\vec{P}$ to the normative parameter values $\vec{P}_N$, where $\vec{P} = \vec{P}_N + \Delta\vec{P}$. The search for effective therapies can be viewed as a combinatorial optimization over the space of parameter deviations, with fitness of each

Table A-1. The Parameters Governing the Model in This Work.

Param.	$\vec{P_N}$	Description
ϕ_t	1.0	τ_t growth rate
α_t	10^{-4}	τ_t VegF activation level
ϕ_s	1.0	τ_s growth rate
α_s	10^{-4}	τ_s VegF activation level
ε_t	2.0	Elasticity of τ_t
ε_s	2.0	Elasticity of τ_s
$J_{t,t}$	16	Adhesion between τ_t and τ_t
$J_{t,s}$	0	Adhesion between τ_t and τ_s
$J_{s,s}$	1	Adhesion between τ_s and τ_s
$J_{t,e}$	2	Adhesion between τ_t and τ_e
$J_{s,e}$	2	Adhesion between τ_s and τ_e
$J_{t,st}$	10	Adhesion between τ_t and τ_{st}
$J_{s,st}$	10	Adhesion between τ_s and τ_{st}
μ_c	35000	VegF chemotactic force on τ_t
μ_h	2000	ECM haptotactic force on τ_t
k_t	0.1	ECM degradation by τ_t
k_{t0}	0.05	ECM degradation adjacent to τ_t
η	0.25	ECM to fluid threshold
λ_V	10^{-5}	VegF degradation rate
D_V	0.045	VegF diffusion rate
S_V	0.035	VegF secretion rate
β_V	10^{-4}	VegF binding and absorbtion rate
γ	1.0	VegF binding efficiency

potential therapy being evaluated using two competing minimization objective functions: (a) the estimated cost of medically implementing those changes in the patient, represented as *Cost*($\Delta\vec{P}$), and (b) the estimated oxygen provided to the tumor by the simulated blood vessels formed when COMPUCELL is run under the changed parameter values, represented as *O_2*($\Delta\vec{P}$). The cost objective is described in Section A.1, the oxygen objective is described in Section A.2.

Optimization occurs in the deviation space, *X*, defined as a subset of the $\mathbb{R}^{23}$ vector space, where each $\Delta\vec{P} \in X$ contains 23 parameters deviations (dimensions) bound by ranges. The deviation range for each parameters is limited to what changes are physically valid in the model and may be medically feasible to change in the future. Table A-1 gives details of each parameter, which are summarized below, grouped by subsystem:

$$\vec{P} = (\underbrace{p_1 \ldots p_4}_{\text{growth}}, \underbrace{p_5 \ldots p_{13}}_{\text{adhesion}}, \underbrace{p_{14}, p_{15}}_{\text{forces}}, \underbrace{p_{16}, p_{17}, p_{18}}_{\text{ECM}}, \underbrace{p_{19} \ldots p_{23}}_{\text{VegF}})$$

Equation A-1.

The subsystems are integrated into the cellular Potts model (CPM) and include endothelial cell growth, cell adhesion, the forces of chemotaxis and haptotaxis, the ECM (extra-cellular matrix fibers in the host tissue) adhesion and degradation, and finally a model of vasculoendothelial growth factor (VegF) secretion from the tumor, diffusion through the host tissue, and endothelial cell uptake and activation.

A.2.1 Improving Monte Carlo Search

The search engine uses a naive, improving Monte Carlo search algorithm to discover novel and potentially counterintuitive therapies in *X* that require the least medical cost for the largest average decrease in oxygen supplied to the tumor. To maintain medical practicality, the Monte Carlo search samples $\vec{P}$ vectors with at most three parameters deviated from nominal. The improving Monte Carlo search is an effective approach to this optimization problem because it maintains a set of promising $\vec{P}$ vectors while monotonically decreasing their estimated cost of clinical implementation.

The improving Monte Carlo search algorithm, A*lgorithm 1*, used in this study takes as input a seeded list, *S*, of Monte Carlo deviation vector samples from *X*, the maximum number, *k*, of active (non-zero) parameters in each sample, and returns a list, *L**, containing promising deviation vectors found during the search. See the following algorithm.

```
MONTE_CARLO($S$, $k$)
$K^{(0)} \leftarrow S$
$k^{(0)} \leftarrow k$
for $i$ = 0 to $k$ - 1 do
    for $j$ = 1 to $n$ do
        $C[j] \leftarrow Cost(K_j^{(i)})$
        $O[j] \leftarrow O_2(K_j^{(i)})$
    end for
    $L \leftarrow Optimal(K^{(i)}, C, O)$
    $L^* \leftarrow L$
    $K^{(i+1)} \leftarrow Sample(L, k^{(i)} - 1, n)$
    $k^{(i+1)} \leftarrow k^{(i)} - 1$
end for
return $L^*$
```

Let S be a list of n parameter deviation vectors sampled from X. For $1 \leq j \leq n$, each deviation vector $\Delta \vec{P}_j \in S$ is defined as $\Delta \vec{P}_j = (\delta_{p1}, \ldots, \delta_{pi};, \ldots, \delta_{p23})$. If the deviation $\delta_{pi} \neq 0$, then the ith parameter is said to be "active". $A(\Delta \vec{P})$ denotes the number of active parameters in the deviation vector $\vec{P}$.

The improving Monte Carlo search takes as input the list, *S*, and the maximum number of active (non-zero) parameter deviations, *k*, then searches over the combinatorial space, *X*. The algorithm maintains a list of promising deviation vectors, *K*. *K* is initially seeded with *S*, where each parameter deviation vector in *S* has the same number of active parameters, and $A(\vec{P}_j) = k$ for each deviation vector $\Delta \vec{P}_j \in S$. Our Monte Carlo search operates in a "top-down" fashion. That is, at each iteration, it systematically reduces the number of active parameters in each $\Delta \vec{P}_j \in K$ until each vector has no active parameters remaining.

Our improving Monte Carlo search proceeds in an iterative fashion as follows. At the i^{th} iteration, the estimated cost and the amount of amount of oxygen supplied to the tumor are first evaluated for each $\Delta\vec{P} \in K^{(i)}$ using *Cost()* and *O_2()*. The results are stored in the data structures C and O, respectively. The function, *Optimal($K^{(i)}$, C, O)* is then applied to $K^{(i)}$. *Optimal()* returns a list, L, of deviation vectors that demonstrate good effectiveness and have low cost. These promising deviation vectors are then given to the *Sample()* function. *Sample(L, k - 1, n)* uniformly samples the list of promising deviation vectors in L and returns a new list with each selected $\vec{P}$ vector reduced by exactly one active parameter. This reduction of a $\Delta\vec{P}$ randomly selects one of the active parameters in $\Delta\vec{P}$ and sets its deviation to 0, thus rendering that parameter inactive. Each promising deviation vector is then perturbed by adding uniform noise to its active parameters while keeping the parameter deviations within the parameter range constraints. This set of perturbed and reduced deviation vectors becomes $K^{(i+1)}$ for the next iteration. During each iteration, the deviation vectors in L are appended to the list, L^*, which contains every promising vector deviation discovered by the search thus far. The algorithm is described in pseudo-code in Algorithm 1.

A.2.1.1 Evaluating the Cost of Each Deviation Vector. Given a deviation vector defined as $\Delta\vec{P} = (\delta_{p1}, \ldots, \delta_{pi}, \ldots, \delta_{p23})$, let the range of parameter i be $R_i = H_i - L_i$. Then, the cost of a deviation vector is estimated by summing the weighted individual errors:

$$Cost(\Delta\vec{P}) = \sum_{1}^{23} \omega_i * (\frac{\delta p_i}{R_i})^2$$

$$\omega_i = \begin{cases} 2.0 & \text{if cost of } p_i \text{ is low} \\ 4.0 & \text{if cost of } p_i \text{ is medium} \\ 8.0 & \text{if cost of } p_i \text{ is high} \end{cases}$$

Equation A-2.

The relative costs of changing parameters ω_i is determined by reviewing the literature on known medical intervention strategies [11] and considering the complexity of the biological subsystem to be manipulated. For example, changing the diffusion properties (via the ligand) or binding efficiency of VegF is already commercially viable and is, therefore, given a low cost, while changing the adhesive or elasticity of specific cell types is complex and poorly understood and is, therefore, given a high cost.

A.2.1.2 Parallel Evaluation of Oxygen Supplied to the Tumor for Each Deviation Vector. The COMPUCELL model simulates angiogenesis, beginning with the secretion of VegF by the tumor. This induces the budding and growth of new blood vessels that sometimes form loops, enabling blood flow and oxygen delivery. The stochastic nature of the simulation generates different morphologies based on an initial random seed tied to each individual simulation. Due to the differing morphologies, the search engine evaluates 256 random initializations for a given $\Delta\vec{P}$. The metric $O_2(\Delta\vec{P})$ is the average O_2 absorbed by the tumor in each run.

For each simulation run, the domain is first initialized, as illustrated in Figure

A-1, with initial blood vessel at the top, the tumor on the bottom, and the stroma randomized with ECM fibers and stromal cells. The parameters used by COMPUCELL are set from $\vec{P} \leftarrow \vec{P}_N + \Delta\vec{P}$, and it is run for 1,500 Monte Carlo Steps (MCS). The O_2 score of this run is calculated by accumulating the O_2 absorbed by the tumor at each MCS.

The COMPUCELL simulations performed while evaluating $O_2(\Delta K_j^{(i)})$ during each iteration are independent from each other. This implies that the improving Monte Carlo search is what is known as an embarrassingly parallel algorithm [3] [13]. Provided enough processors, the COMPUCELL simulations required for individual $O_2()$ evaluations, as well as multiple $O_2()$ evaluations themselves, can be performed concurrently and executed in nearly the same amount of time as a single simulation. On a modern processor, one simulation can be completed in five to seven minutes. Because the Monte Carlo search requires the execution of several thousand simulations, the algorithm

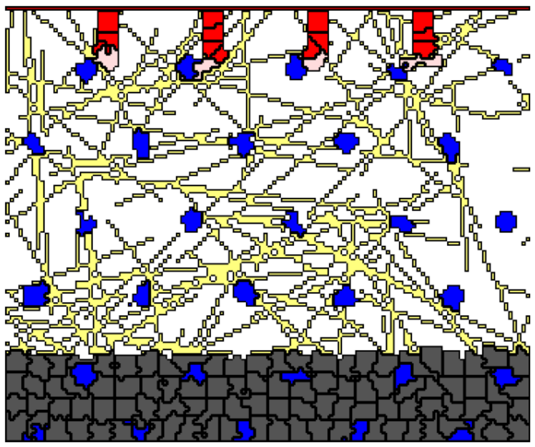


Figure A-1. The initial conditions of the angiogenesis simulation of the cell type domain σ[,]. The colors used are given in Table A-2.

Table A-2. Cell Types Used in Simulation.

Cell Types		
Symbol	Description	Color in Figures
τ_m	Medium	White
τ_{ev}	Existing Vasculature	Dark red
τ_{tu}	Tumor	Black
τ_s	Endothelial Stalk	Red
τ_t	Endothelial Tip	Pink
τ_{st}	Stromal Cell	Blue
τ_e	ECM Fiber	Yellow

cannot be feasibly performed in a reasonable amount of time without the use of a large set of processors. The improving Monte Carlo search used in this study was implemented using the common master/slave paradigm and was executed on a 2,312-processor AMD Opteron^{TM} cluster maintained by the Arctic Region Supercomputing Center.

A.2.2 Angiogenesis Model

The CPM facilitates the combination of simpler submodels into a larger model that is biologically accurate and computationally feasible. One advantage of the CPM is that multi-cellular pattern formation is an emergent property of local interactions between simple subcellular components. Another advantage is that all cellular mechanisms including adhesion, growth, death, haptotaxis, and chemotaxis are easily and realistically implemented within the same architecture. The cell types used in this work interact with each other through the process of energy minimization to create emergent morphologies exhibiting simulations of angiogenesis and anastomosis.

A.2.2.1 Initial Conditions. In our simulation, the angiogenesis domain is modeled as a two-dimensional section bordered by the existing vasculature on the top and the tumor on the bottom, as a layer of four cells, as illustrated in Figure A-1. The distance from the existing vasculature to the surface of the tumor wall is 165 μm. The stromal area between the tumor and the vasculature is a mixture of medium, ECM fibers, and stromal cells. Each new sprout vessel is initially modeled as two cells. One cell is an endothelial tip cell, and the other is an endothelial-stalk cell that touches the existing vasculature.

A.2.2.2 Cellular Potts Model. The cellular Potts model [4] (CPM) is used to simulate the tissues and integrate the mechanisms in the angiogenesis process. The CPM represents the tissue domain as a two-dimensional array of lattice sites or pixels, each ≈ 2μm square. In all the simulations reported here, the tissue array is 100 by 120 pixels. Each cell within the tissue is represented as a set of contiguous pixels. Cell-cell contacts occur through adjacent pixels which belong to different cells. As the simulation is run, cells form new contacts and move with restrictions in size and in shape. All cell rearrangement is driven by a process of stochastic energy minimization.

The energy of a specific CPM tissue configuration is described by a Hamiltonian equation over the two-dimensional domain *s*, which comprises a set of pixels *z*, each with a designated cell id σ_z:

$$H_s = \sum_{z \in s, z' \in n(z)} J_{\tau_{\sigma_z}, \tau_{\sigma_{z'}}} + \sum_{\sigma \in s} \varepsilon_{\tau_\sigma} (a_\sigma - A_\sigma)^2$$

Equation A-3.

The first term implements differential adhesion based on the type of each cell, denoted by $\tau_{\sigma z}$, by estimating the total surface energy between all contacting cells σ_z and $\sigma_{z'}$. This is done by summing $J\,\tau_{\sigma z}, \tau_{\sigma z'}$ over all adjacent pixels z and z' where $\sigma_z \neq \sigma_{z'}$. There are 7 cell types used in the angiogenesis simulation and they are listed in Table A-2. The $J\,\tau, \tau'$ values are given in Table A-1.

The second term in Equation A-3 implements an area constraint on cells where a_σ is the actual area of a cell σ, and A_σ is σ's target area. The elasticity of a cell (the ease with which the cell can change its area) is controlled by the parameter ε_τ. Table A-1 gives the elasticity values for the endothelial tip and stalk cells. The target area of a cell is dependent on the cell-type. For those cell types that do not grow (tumor τ_{tu} and stromal τ_{st}), the target area is fixed. For those cell types that grow (endothelial tip τ_t and stalk τ_s), the target area A_σ is incremented each model iteration. When a growing cell's current target area reaches twice its original target area, it is split into two daughter cells, and the growing process repeats. In this work, all mitosis is symmetric, and the cleavage angle is chosen randomly. In angiogenesis, growth of endothelial cells is dependent on the presence of VegF and is controlled by the threshold value α_τ given in Table A-1.

Low energy cell arrangements are determined by repeatedly copying the state of one pixel $\sigma[x, y]$ at x, y to an adjacent pixel at x', y' for pixels belonging to different cells. Let s be the configuration before the copy and s' be the configuration after the copy; then $\Delta H_{s, s'} < 0$ is defined as $H_{s'} - H_s$. Further, if $\Delta H_{s, s'} < 0$, the state change is always accepted, and if $\Delta H_{s,s'} = 0$, the state change is accepted with probability 0.5. Otherwise, the state change is accepted with probability $e^{-(\Delta H_{s,s'})/T}$, where T is the temperature, representing the agitation of the cells [4].

In angiogenesis, the endothelial tip cell responds to haptotaxis [1] based on the local concentration of ECM in the tissue, and chemotaxis [7] based on the local concentration of VegF. To integrate these mechanisms into the CPM, two additional terms are added to the energy change function $\Delta H_{s,s'}$:

$$\Delta H_{s,s'} = (H_{s'} - H_s) + k_H(E[x', y'] - E[x, y]) + \mu_\sigma(V[x', y'] - V[x, y])$$

Equation A-4.

The level of VegF at pixel *x, y* is defined as *V [x, y]* with the strength of the chemotactic force controlled by parameter μ_σ given in Table A-1; the level of ECM at pixel *x, y* is defined as *E[x, y]* with the strength of the haptotactic force controlled by parameter k_H also given in Table A-1. How *V[x, y]* and *E[x, y]* are initialized and calculated is described in following sections.

To run the simulation forward from the initial conditions, a pixel and its neighbor are randomly selected, and a pixel state copy is considered as described above. One iteration of the simulation is termed an MCS (for Monte Carlo step) and comprises of repeatedly choosing a random pixel n times, where *n* is 16 times the number of the pixels in the domain, which is 100 x 120 in the simulation model described here. The remaining subsections describe each subsystem in detail.

A.2.2.3 Secretion and Uptake of VegF. The recruitment of vessels by tumor cells is accomplished through the secretion of VegF, and the resulting VegF diffusion gradient.

The VegF exerts a chemotactic force on the endothelial tip cells, and thus induces vessel growth towards the tumor. The level of VegF at location *x, y* is defined as *V[x, y]* and is controlled by the following reaction-diffusion equation:

$$\frac{dV[x,y]}{dt} = D_V \nabla^2 V[x,y] - \lambda_2 V[x,y] + S_V(x,y) - B(x,y,V[x,y])$$

Equation A-5.

There are four parameters that govern the VegF system. The coefficient of diffusion of VegF $D_V > 0$ is assumed to be homogeneous throughout the simulation domain. The degradation of VegF is also considered constant, at $\lambda_V > 0$. The two functions *S_V(x, y)* and *B(x, y, V[x, y])* describe the secretion and the absorption, respectively, of VegF in the domain. *S_V(x, y)* describes the secretion of VegF from tumor cells positioned on the right border of the domain, while *B(x, y, V[x, y])* describes the binding and uptake of VegF by the endothelial cells.

$$S_V(x,y) = \begin{cases} \phi & \text{if } \sigma[x,y].\tau = \tau_{tu} \\ 0.0 & \text{otherwise} \end{cases}$$

The secretion of VegF is at a constant rate o from the tumor cells, positioned along the bottom side of the domain.

$$B(x,y,V) = \begin{cases} \beta & \text{if } \beta \leq V \text{ and } \sigma[x,y].\tau = \tau_t \text{ or } \tau_s \\ V & \text{if } 0.0 \leq V \leq \beta \text{ and } \sigma[x,y].\tau = \tau_t \text{ or } \tau_s \\ 0.0 & \text{otherwise} \end{cases}$$

The binding and uptake of *V* by the endothelial cells is defined in *B(x, y, V)* and is limited to a maximum rate of *β* > 0.0 over the external surface of the endothelial cells. This is realistic since the capacity to bind and uptake VegF will saturate to a rate-limit.

A.2.2.4 Stroma. The ECM in the stroma exerts haptotactic forces on the endothelial tip cells. This force can cause the vessel paths to be diverted towards other vessels, resulting in anastomosis. The stroma is modeled using a combination of a discrete and a continuous model. The discrete model is over the [,] domain and represents stromal cells st and the ECM fibers denoted by e. The continuous model contains the concentration level of ECM fiber proteins and is stored in the E[,] domain.

A.2.2.5 Degradation of ECM fibers in E. The secretion of proteolytic enzymes by endothelial tip cells degrades the level of ECM protein when the endothelial cells are over or next to the ECM [12]. To model this process, the level of ECM at a point *x, y* is assumed to decay exponentially at those locations occupied by an endothelial tip cell, or directly adjacent to that cell. This effect is illustrated in Figure A-2. It is assumed that the

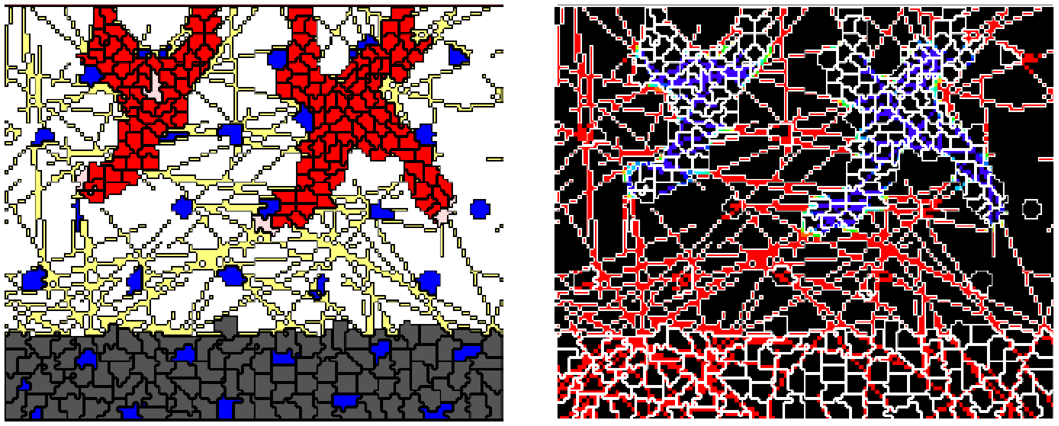

(a) [,] (MCS=600) (b) E[,] (MCS=600)

Figure A-2. *σ[,]* and *E[,]* domains where the τ_t cells create vessel paths exhibiting anastomosis. The τ_t cells are pink, and the τ_s cells are red. The degradation of τ_e fibers is evident in the *E* lattice, where red indicates high concentrations of the ECM protein and blue indicates low concentrations. The vessel paths also exhibit haptotactic influence from the ECM.

rate of decay of ECM proteins is higher when the tip cell is over the fiber than next to it. The behavior is captured in the following equations:

$$E[x,y] = E[x,y] * \begin{cases} e^{k_i} \text{ if } \sigma[x,y].\tau = \tau_t \\ e^{k_n} \text{ if } \sigma[x',y'].\tau = \tau_t \text{ and } x', y' \text{ adjacent to } x, y \\ 1.0 \text{ otherwise} \end{cases}$$

The physical effects of the ECM fibers are modeled within the CPM, where all the fibers are denoted $\sigma[x, y].\tau = \tau_e$ over the domain. τ_e is inelastic and initially arranged in random lines representing fiber bundles (see Figure A-2). By integrating the ECM fibers into the CPM domain, the sprout's morphological development is affected by the growing sprouts pulling along the surfaces of the fiber bundles and being turned by the fiber bundle obstructions.

Initially, the ECM fibers are allocated a level of ECM protein *E[x, y]* uniformly, then decayed according to Equation A-5 in the presence of endothelial tip cells. To model the removal of the physical fibers once the ECM protein has been sufficiently degraded, the fibers become interstitial fluid (medium) when *E[x, y]* drops below a fixed threshold η (given in Table A-1). This change in type is implemented by changing $\sigma[x, y].\tau$ from τ_e to τ_m.

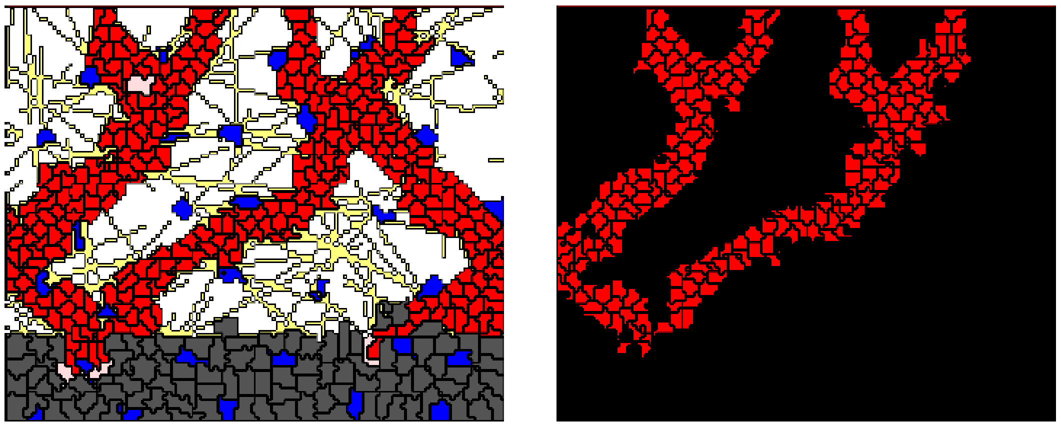
A.2.3 Emergent Properties

The new blood vessels shape, structure and network emerge from the complex interplay among the mechanisms of differential cell adhesion, VegF activated growth, chemotaxis, haptotaxis, tip cell-based ECM degradation, and the secretion, diffusion, and uptake of VegF.

A.2.3.1 Angiogenesis. The VegF chemical field is used as a growth signal to the τ_s cells. When the chemical level is above the threshold α_e, the cells grow. This method of growth, coupled with the uptake of VegF, the adhesion of cell types, and the chemotactic influence of the VegF gradient, produces the emergent property seen in tumor-induced blood vessel growth in animals. The τ_t cells climb the VegF gradient using the chemotactic force. The τ_t and τ_s cells uptake a portion of the VegF that is contained in their lattice pixels. The τ_s cells grow and divide in the presence of VegF and adhere to the τ_t cells. This creates a vessel growth pattern that follows the movement pattern of the τ_t cells. In effect, the τ_t cells pull the τ_s cells through the VegF gradient. The τ_s cells grow directly behind the τ_t cells, but they do not continue to grow as the cells get further from the τ_t cell. This is due to the low levels of VegF concentration that are left in the growth path due to accumulated uptake. Since the ECM is only degraded by the τ_t cells, the τ_s cells are often bounded by the remaining ECM fibers as they grow, forming regular width curving vessels under nominal conditions.

A.2.3.2 Anastomosis. The emergent property of anastomosis loop formation is critical to the usefulness of the simulations in evaluating cancer disruptions, since oxygen is only secreted by vessel loops. Anastomosis arises from the inclusion of complex stroma in the model, which causes the separate sprouts to bend and collide. The path of τ_t cells is a combination of many factors, including the haptotactic force provided by the ECM fibers, the chemotactic force of the VegF gradient, the adhesive force of stromal cells encountered, and the density of ECM fiber bundles which provide obstructions.

A.2.3.3 Endothelial Tip Cell Growth Termination. When two sprouts join together, it is common for one of the endothelial tip cells to continue growing, while the other becomes dormant. This desired behavior has been explicitly programmed into other models [9] [8], but in this system, the behavior naturally emerges from interactions of subcomponents. When two sprouts join together, the endothelial tip cell loser to the higher concentrations of the VegF gradient may often continue, while the other will stall out. It is common for the leading tip cell to continue growing towards the tumor through the chemotactic force, and thus consume the VegF available to both sprouts. The other endothelial tip cell will most likely terminate progression towards the tumor due to lack of VegF activation and gradient. The dominant endothelial-tip cell pattern recursively emerges to create an inverse tree-like vessel structure. This structure is consistent with biological observations and is illustrated in Figure A-3.



(a) Cells

(b) O_2 secreting cells only

Figure A-3. (a) Typical angiogenesis morphologies form an inverted tree-like structure with some tip cells being dominated by other sprout cells. (b) Only endothelial cells in loops that have sufficient blood flow secrete O_2.

A.2.4 Oxygen

The level of O_2 at location *x, y* is defined as $O_2[x, y]$ and is controlled by the following reaction-diffusion equation:

$$\frac{dO_2[x,y]}{dt} = D_2\nabla^2 O[x,y] - \lambda_2 O_2[x,y] + S(x,y) - U(x,y,O_2[x,y])$$

Equation A-6.

There are four parameters that govern the O_2 system. The coefficient of diffusion of O_2 $D_2 > 0$ is assumed to be homogeneous throughout the simulation domain. The degradation of O_2 is also considered constant at $\lambda_2 > 0$. The two functions *S(x, y)* and *U(x, y, O_2[x, y])* describe the secretion and the absorption, respectively, of O_2 in the domain. *S(x, y)* describes the secretion of O_2 from endothelial cells contained within loop structures in the resulting morphology (see below for an explanation), while *U(x, y, O_2[x, y])* describes the absorption of O_2 by the tumor cells.

$$S(x,y) = \begin{cases} \alpha & \text{if } \sigma[x,y].\tau = \tau_t \text{ or } \tau_s \text{ and } \sigma[x,y] \text{ is secreting} \\ 0.0 & \text{otherwise} \end{cases}$$

The secretion of O_2 is at a constant rate α from the surface of all endothelial cells that are within loops formed by anastomosis. It is realistic to assume that only those sprouts that successfully link with other sprouts and form loops will be able to secrete O_2, because they will be capable of maturing and carrying a flow of blood through the formed vessel. To determine which endothelial cells secrete O_2 during the angiogenesis simulation, a pressure diffusion model is calculated at the cell level to identify loops as contiguous paths along contacting endothelial cells that contact the original blood vessel at distinct locations. The flow (pressure difference) between cells is used to classify a cell as secreting ,thereby enabling the correct implementation of *S(x, y)* above when the O_2 diffusion equation given in Equation A-6 is solved.

$$U(x,y,O_2) = \begin{cases} \upsilon & \text{if } \upsilon \le O_2 \text{ and } \sigma[x,y].\tau = \tau_{tu} \\ O_2 & \text{if } 0.0 \le O_2 \le \upsilon \text{ and } \sigma[x,y].\tau = \tau_{tu} \\ 0.0 & \text{otherwise} \end{cases}$$

The absorbtion of O_2 by the tumor is defined in $U(x, y, O_2)$ and is limited to a maximum rate of $\upsilon > 0.0$ over the external surface of the tumor cells. This is realistic since the capacity to absorb oxygen will saturate to a rate-limit based on the maximum reaction rate on the membranes of the tumor cells.

Finally, the objective function $O_2(\Delta P)$ can be defined. First the parameters that control the simulations are set $\vec{P} \leftarrow \vec{P}_N + \Delta\vec{P}$, the domain initialized, and the simulation is run 1,500 Monte Carlo steps (MCS). Let $O_2[x, y]_t$ be the oxygen array at MCS *t*, then the oxygen score for this random initialization is:

$$O_2(\Delta P) = \sum_{t=0}^{1500} \sum_{x=1}^{100} \sum_{y=1}^{120} U(x,y,O_2[x,y]_t)$$

A.2.4.1 Summary of Angiogenesis Simulation. A CPM is used to model tissue and cells as a stochastic energy minimization process over an array of pixels. Chemical gradients create cell signaling. VegF is secreted by the tumor cells and absorbed by the endothelial cells, thereby forming a diffusion gradient from the tumor to the vessel sprouts. The endothelial tip cells respond to the VegF concentration by moving towards higher concentrations. The VegF also activates the endothelial sprout cells to grow. VegF is consumed by the endothelial cells, and eventually the concentration is low enough that the endothelial sprout cells become inactive. A vessel emerges that follows the chemotactic path of the endothelial tip cell.

The ECM fibers form obstructions to the endothelial cells. The tip cell degrades the ECM, until the protein concentration is low enough to become interstitial fluid. The tip cells adhere to the ECM fibers which causes a haptotactic force along contacting fibers. The ECM diverts the vessel sprouts, causing them to grow towards other sprouts. As anastomosis occurs, one of the tip cells may become dormant while the other tip cell continues to progress towards higher concentrations of VegF. This competition between tip cells results in blood vessels exhibiting an inverted tree structure, since this model only represents the early stages of angiogenesis and does not consider the formation of new tip cells. (see Figure A-4).

Oxygen begins to secrete from the endothelial cells as the difference in blood pressure across the created network increases above a threshold. The oxygen diffuses

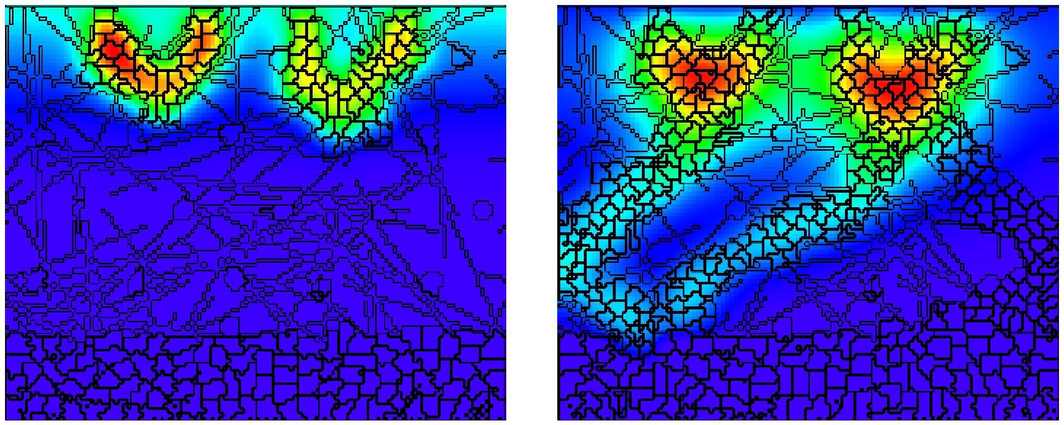


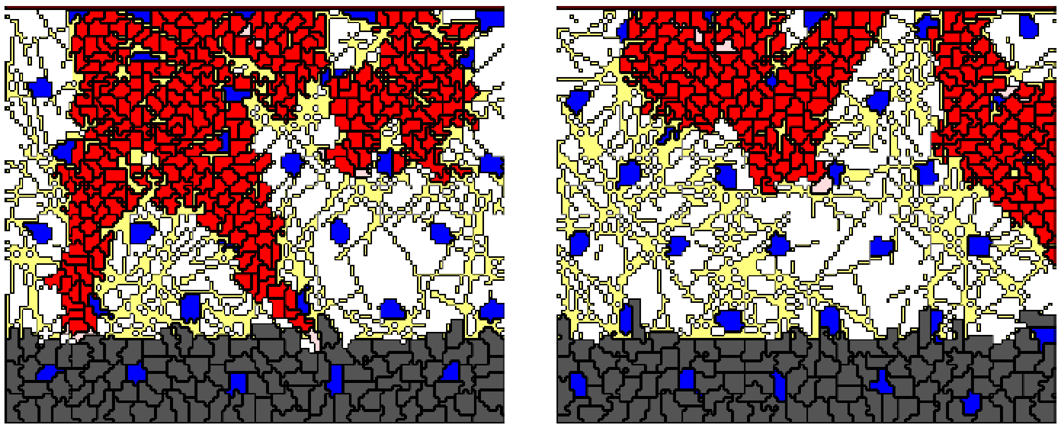
Figure A-4. O_2 values over the domain, where high concentrations of oxygen are red and low concentrations are blue. O_2 is secreted from vessels that form loops, diffuses through the stroma, and is absorbed by the tumor cells.

through the medium and reaches the tumor. The tumor cells consume the available oxygen, which is used as the principle metric for the search engine.

A.3 Results

The Monte Carlo improving search method was run on the AMD OpteronTM cluster at the Arctic Region Supercomputing Center. The oxygen score of each parameter vector was evaluated by running it on 256 processors in parallel, using different random initial ECM fiber and stomal cell configurations, and different random seeds to the stochastic CPM. Each angiogenesis simulation took approximately five to seven minutes of cpu time. A total of 140 solutions were generated and evaluated for this preliminary study. Figure A-5 shows example morphologies form solutions c and j given in Figure 7.

Out of these 140 solutions generated, the best 22 near-pareto optimal solutions are given in Figure A-6 as a pareto-optimal frontier, and in detail in Figure A-7. Reviewing the results, it is clear that the Monte Carlo improving search is capable of discovering



(a) Solution *c* (b) Solution *j*

Figure A-5. Example morphologies from solutions c and j given in Figure A-7.

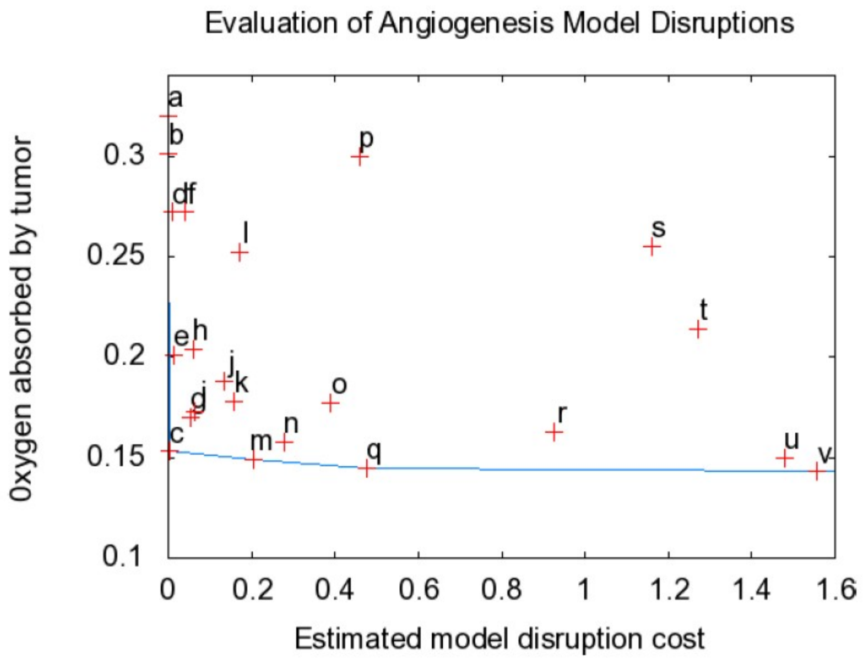


Figure A-6. Results: The Pareto optimal frontier and near optimal potential cancer therapies identified by the Monte Carlo reducing search process. The best solutions provide the minimum O_2 to the tumor (vertical axis) for the least estimated treatment cost (horizontal axis). The point labels correspond to the solutions shown in Figure 1-7.

novel and potentially important cancer therapies quickly. All the solutions presented reduce the oxygen to the tumor during early development, with the optimal solutions reducing oxygen by approximately 50%. This level of reduction will significantly slow the growth of the tumor and has the potential to improve the survivability rate of patients [10].

By reviewing the individual solutions in Figure A-7, it is satisfying to see that the currently medically viable strategy of disrupting VegF [11] is discovered in solutions b and m, which both reduce the effectiveness of uptake of VegF by the endothelial cells,

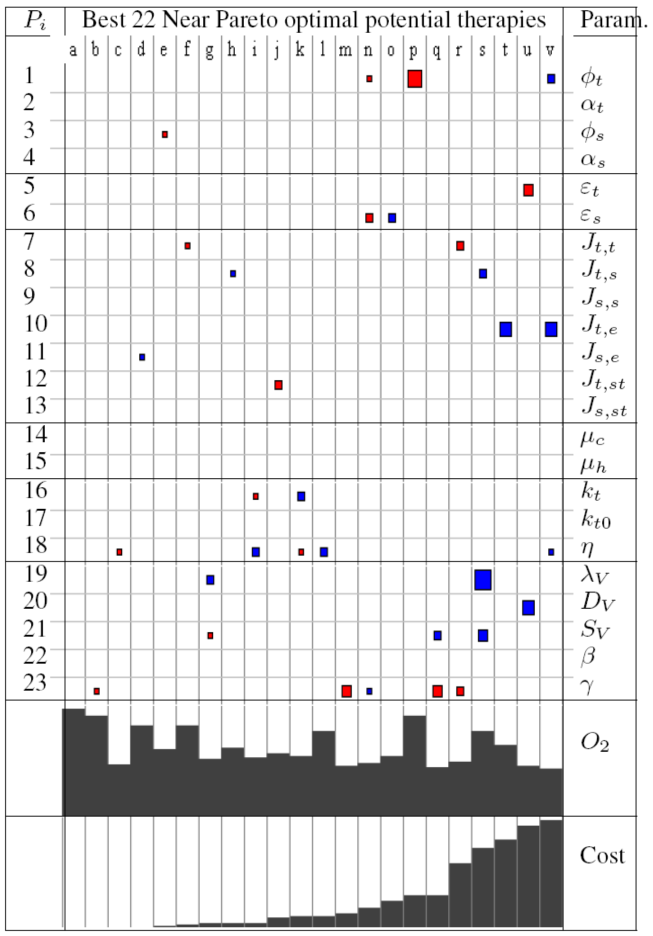


Figure 1-7. Results: Near Pareto optimal model disruptions found by the search engine. Each column represents one solution with the letter designations (across the top) corresponding to those in Figure 1-6. The parameter disruptions for each solution is given in the column, with white meaning no change, red a reduction and blue an increase. The size of the box represents the magnitude of the change. The average *O_2* provided to the tumor and cost is given in the last two rows.

and in solution g, which increases the degradation rate of VegF. Of more interest are the many solutions found that are quite novel and counter-intuitive, and while it is possible to form a post hoc explanation of each solution's effectiveness, it is very unlikely that these solutions could have been manually discovered. Consider one of the most effective solutions *c*, that works by slightly reducing the threshold parameter which controls when the degraded ECM fibers become medium. In other words, solution *c* suggests strengthening the ECM fibers in the tissues neighboring the tumor. A typical morphology for solution *c* is illustrated in Figure A-5(a) and shows clumped, stunted vessels intermixed with nondegraded ECM. Or consider solution *j* that suggests reducing $J_{t, st}$ thereby increasing the adhesive force between endothelial tip cells and stromal cells. A typical morphology for solution *j* is illustrated in in Figure A-5(b) and shows poorly formed thick vessels, with each of the four original tip cells stuck to a stromal cell that it encountered while moving up the VegF gradient. Solution *j* is an example of the power of this method to discover novel potential cancer therapies that disrupt the angiogenesis process in unexpected ways.

A.4 Conclusions

This work has described a massively parallel combinatorial search method for exploring the space of possible angiogenesis-blocking medical interventions for treating cancer. The method combines a high fidelity angiogenesis simulation system with a massively parallel Monte Carlo improving search process. In a preliminary study of only 140 sample model disruptions, many effective and medically feasible therapies were found. A few of the solutions were simple techniques that disrupt the VegF system and are currently being deployed. Significantly, many of the solutions were novel and, surprising, strongly suggest that this work could lead to the development of powerful new anti-cancer treatments.

A.5 Acknowledgments

This work was supported in part by a grant of high performance computing (HPC) resources from the Arctic Region Supercomputing Center at the University of Alaska Fairbanks as part of the Department of Defense High Performance Computing Modernization Program.

References

[1] Amy L. Bauer, Trachette L. Jackson, and Yi Jiang. A cell-based model exhibiting branching and anastomosis during tumor-induced angiogenesis. Biophys. J., 92(9):3105–3121, May 2007.

[2] P. Carmeliet and R. K. Jain. Angiogenesis in cancer and other diseases. Nature, 407(6801):249–257, September 2000.

[3] K. Esselink, L. D. J. C. Loyens, and B. Smit. Parallel monte carlo simulations. Phys. Rev. E, 51(2):1560–1568, Feb 1995.

[4] F. Graner and J. A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. Phys. Rev. Lett., 69:2013–2016, 1992.

[5] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel, S. A. Newman, and J. A. Glazier. Compucell, a multi-model framework for simulation of morphogenesis. Bioinformatics, 20(7):1129–1137, May 2004.

[6] Ahmedin Jemal, Elizabeth Ward, Yongping Hao, and Michael Thun.
Trends in the Leading Causes of Death in the United States, 1970-
2002. JAMA, 294(10):1255–1259, 2005.

[7] käfer, Jos , Paulien Hogeweg, and Athanasius F. Maree. Moving forward moving backward: Directional sorting of chemotactic cells due to size and adhesion differences. PLoS Computational Biology, 2(6):e56+, June 2006.

[8] S. R. McDougall, A. R. Anderson, and M. A. Chaplain. Mathematical modelling of dynamic adaptive tumour-induced angiogenesis: clinical implications and therapeutic targeting strategies. J Theor Biol, 241(3):564–589, August 2006.

[9] S. R. McDougall, A. R. Anderson, M. A. Chaplain, and J. A. Sherratt. Mathematical modelling of flow through vascular networks: implications for tumour-induced angiogenesis and chemotherapy strategies. Bull Math Biol, 64(4):673–702, July 2002.

[10] S. Shinkaruk, M. Bayle, G. La¨ın, and G. D´el´eris. Vascular endothelial cell growth factor (vegf), an emerging target for cancer chemotherapy. Current medicinal chemistry. Anti-cancer agents, 3(2):95–117, March 2003.

[11] Farbod Shojaei and Napoleone Ferrara. Antiangiogenic therapy for cancer: An update. The Cancer Journal, 13(6):345–348, November 2007.

[12] Stephen Turner and Jonathan A. Sherratt. Intercellular adhesion and cancer invasion: A discrete simulation using the extended potts model. Journal of Theoretical Biology, 216(1):85–100, May 2002.

[13] B. Wilkinson and A. Michael. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Upper Saddle River, NJ: Prentice Hall, second edition, 2005.

Appendix B:

Thursday, May 1, 2008

Hi Brian,

You have my approval to include the paper "Discovering Novel Cancer Therapies: A Computational Modeling and Search Approach" in your thesis.
Good luck!

Art

Appendix C:

Thursday, May 1, 2008

Dear Brian,

I am delighted to have my name appear as a co-author on the papers going into your thesis.

Best wishes,

Greg Podgorski