ADAPTIVE SIMULATION MODEL CONFIGURATION

by

Randall Gray, B.A. (Flinders University of South Australia)

Submitted in fulfilment of the requirements
for the Degree of Doctor of Philosophy

Discipline of Mathematics
University of Tasmania
December, 2018

I declare that this thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the thesis, and, to the best of my knowledge and belief, no material previously published or written by another person, except where due acknowledgement is made in the text of the thesis; nor does the thesis contain any material that infringes copyright.

Dr Simon Wotherspoon is listed as a contributor to the papers included as Chapters 2 and 3, his contribution to each of the papers was predominantly as a sounding board and a source of encouragement. His direct input into the papers was primarily concerned with the structure of the paper and its ordering rather than its content.

Signed:

Date: December, 2018

The publishers of the papers comprising Chapters 2 and 3 hold the copyright for those chapters, and access to the material should be sought from the respective journals. The remaining non-published content of the thesis may be made available for loan and limited copying and communication in accordance with the *Copyright Act 1968*

Signed:

Date: December, 2018

Statement of Co-Authorship

School of Natural Sciences, Mathematics — Randall Gray (Candidate)
University of Tasmania — Dr Simon Wotherspoon (Supervisor)

Author details and their roles:

Paper 1, *Increasing model efficiency by dynamically changing model representations*

Located in Chapter 2.

The candidate, Randall Gray, was the primary author and was responsible for the conception, design, implementation, preparation of graphics, analysis of the work described in the paper, and writing the paper itself. Dr. Wotherspoon contributed guidance as to how to improve the presentation of the work and make it accessible to a broader audience before submission, and provided advice on addressing the concerns of the reviewers.

Paper 2, *Adaptive submodel selection in hybrid models*

Located in Chapter 3.

The candidate conceived the ring-like structure used in paper, established the algebraic properties of the structure and proved the relevant theorems. The candidate also designed the thought model described in the paper, prepared the tables and graphics and was wrote the paper. Again, Dr. Wotherspoon contributed by acting as an informal reviewer, highlighting those parts of the paper that were inadequately explained or difficult to follow, and after acceptance by providing advice on addressing reviewers' comments.

Signed:

Dr Simon Wotherspoon
Supervisor
Institute for Marine and Antarctic Studies

Date:

Signed:

Prof Mark Hunt
Head of School
School of Natural Sciences

Date:

ABSTRACT

Models of complex systems may be improved in fidelity, efficiency or both by allowing the way they represent component parts to change as the state of the model and its components move through their state-spaces.

A simple model demonstrates that representation changes for a population encountering contaminants may perform better than either conventional form. Here, there was a substantial decrease in runtime relative to a purely *i*-state configuration (individual-based) model, with comparable fidelity, while the purely *p*-state (population-based) version exhibited error arising from the ̀blurring" of contaminant contact through the distributed population. This example demonstrates the utility of model representations maintain important state data across representations so that previous representations can be recovered with minimal error.

Triggers for changing representations of components is addressed in a paper exploring a possible set of dynamics associated with a simple, hypothetical model of a seven component ecosystem. The components of the system are described as *i*-state configuration models or as *p*-state models, and a mechanism for determining when to change representations is outlined.

To support the analysis and selection of representations, a metric-space with the properties of a commutative ring is defined. The elements of the metric-space are trees that can encode the structural character of a set of submodels which comprise the model of a system, and to provide a metric in analysis.

Finally, a framework is developed with an example model that closely follows the hypothetical example. It was designed as a reference model, and is freely available on `https://github.com/snarkypenguin/Mutans.git` This implementation demonstrates dynamic configuration management, maintenance of the states of superceded submodel representations, and the support structures needed to implement models of this type.

ACKNOWLEDGEMENTS

My deepest thanks go to my very patient wife, Anne, who coped with far too many years with me firmly attached to my keyboard. Thanks also go my supervisors: Simon, who was never short of optimism and encouragement, and both Vincent Lyne and Beth Fulton who shared many years of modelling on the cliff-face with me. I would also like to acknowlege my debt and gratitude to the late Dr William Cornish of Flinders University, who kindled my love of mathematics in 1985.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Modellers of complex systems are often faced with the task of coupling components which may operate at quite different scales with the goal of creating a useful tool for study these systems. This thesis puts forward the argument that better[1] models may be built if we allow the representation of component parts of a model to change according to

their states,
the nature of their interactions with other components,
the needs and states of the other components

and

the state of the model as a whole.

The majority of the discussion will be couched in the context of biological or environmental systems, but it is not limited to this area: the context was chosen for its familiarity, its generality, and because models of ecosystems often incorporate significant environmental, physical and biochemical submodels such as in ATLSS [DeAngelis et al., 1998], Atlantis [Fulton et al., 2011b], and InVitro [Gray et al., 2014].

A mechanism for deciding how close complex mixtures of agents are to nominative "good" or "bad" configurations is necessary, so a mathematical structure is developed which can be used to encode information about the model's constituents and their relationships. This structure is a normed vector-space, and can be used to assess the relative utility of a configuration by comparison with a corpus of configurations which are either preferred or deprecated.

The assessment and conversion of components from one form to another is a potentially burdensome process; to alleviate this, and make the approach more useful, a framework for constructing models, Remodel, is developed in Chapter 5. It treats models as being composed of niches which are analogous to environmental niches. Much as an environmental niche may be occupied by members of a number of different species, a niche in the framework may

[1] Here, *better* can mean *'less error, faster, more consistent with field data* or whatever the model in question is configured to optimise. This objective may also change through the life of the simulation.

be populated by instances of different types of agents. In an environment, the succession of occupants in a niche is driven by the state of the system, and organisms which are more *fit* for the state replace those which perform less well. The same sort of principle may be applied in Remodel. While it can be used in much the same way as other modelling frameworks, such as NetLogo [Wilensky, 1999] or Swarm [Minar et al., 1996], the framework provides support for modellers to indicate what conditions are favourable for different representations which fill the niches in the models and to automatically replace agents which are ill-suited to their context with more appropriate representations when the need arises. The framework is explored using the *thought-model* of Chapter 3 as a notional equivalent of an artist's mannequin.

Such a strategy has a number of potential benefits:

- we can make many simple representations, submodels, for a niche (*sensu* Gray et al. [2006] and Gray et al. [2014]) in the model, each of which deals well with a particular part of the submodel's domain;
- the comparative simplicity of these representations effectively reduces the number of potential code paths within a model at any given moment, since representations do not have to cope with edge cases, they merely indicate that they are entering a marginal or inappropriate domain;
- we can use analytic representations which are more efficient at representing large numbers of entities;
- we can use individual-based representations which capture the fine-scale dynamics that dominate when we are dealing with discrete events or low numbers of entities;
- we can choose representations that make the best use of available data within the model, or can ask for better representations in the ensemble;
- it is simple to incorporate code to track information about representation changes, relative execution speed, and cumulative error into the modelling system;
- we may include agents that identify the emergence of perverse dynamics within the system;

and

- we can decouple the production of the results from processes which simulate the systems and subsystems being modelled.

The work will explore issues such as managing components with different time-steps, the preservation of data across various representations of a component of the model, the sequencing of agents and maintaining temporal consistency, and the fundamental issues regarding how the transition from one model mix to another should be decided upon and effected. It also makes it

simple to address the questions "*How do we deal with situations where the assumptions that underpin our representation no longer hold?*" and "*How do we manage the execution of submodels which simulate systems or entities with multi-modal behaviour?*"

Consider this example: rather than a single representation for the population of a coastal city, we may have a number of different submodels with different levels of aggregation and different temporal or spatial scales. In a simulation of a cyclone season, we may start with a simple single age-histogram representation. As a tropical storm builds we may disaggregate the histogram, appropriately distributing the population to finer age-histograms associated with localities throughout the region. As it approaches the coastline, the essentially static representations which lie in areas likely to suffer damage are converted into agents instances of running submodels which represent households. Shortly before the cyclone reaches the point where damage occurs, we may resolve the representation further, instantiating emergency response agents and converting households agents to individuals at risk. In the aftermath, aggregation may occur in regions where there are no acute effects, but other parts of the system may remain finely resolved.

In this example, we change the representations to deal with both of the questions above. We initially assumed that for most of the purposes of the simulation, our population could be treated as relatively homogeneous, and may have kept aggregate information about population distribution, wealth and demographic characteristics, but as the storm hits, the simulation needs to change the state of a portion of the population (those with damaged property, for example), and we have to refine our representation. Similarly, the behaviours following the storm are modally different: those who live in protected areas are mostly free to carry on with essentially the same normal representation, but those who are living in damaged areas must engage in quite different activities, and may have quite a different exposure to risks.

Adaptive approaches commonly occur in techniques for numerical approximation (regression, root finding, and parameter estimation for example), numerical solutions for systems of differential equations, feature detection and recognition, control systems and route planning. Often these approaches involve adjusting the size of the domain considered (subdivisions or step size), or the rates associated with a process. In the case of route planning, sets of routes may be marked as impassable as data becomes available, triggering a reassessment of the set of possible routes. More broadly, domain adaptation describes a general approach where a model or system adjusts itself to the data it works with one of the canonical examples is a Bayesian spam filter which includes a users assessment of whether email is spam or not in its subsequent assessments. A common trait these adaptive techniques share is that the algorithm which processes the data remains essentially the same.

Huston et al. [1988] argued that individual-based models are able to incorporate dynamics across scales and that the fine scale dynamics experienced by individuals plays a significant role in many of the population scale patterns observed in ecological studies. This is a reasonable position to take, and

individual-based modelling is now a common approach. Individual-based or super-individual-based models are not a panacea. These types of models may become costly as the number of individuals or the interactions between individuals or super-individuals grows, and small discrepancies between the behaviour or parameterisation of the modelled entities and their real counterparts may produce large discrepancies at the population level. Many of the parameters that may influence the life-history of organisms at an individual level are difficult or impossible to estimate in situ, and so the effects of individuals modelled behaviours or processes may not scale well when incorporated in larger systems.

Discrete changes in the behaviour of a system, or part of a system, are commonplace. The scales of systems that exhibit switching behaviour range from a molecular level, such as the behaviour of freezing liquids, through the intermediate scales to the changing climate. Evidence for a broad recognition that systems dynamics can (and do) switch rapidly from one ostensibly stable mode to another is that the discussion of tipping points has increased dramatically in the last decade [Bhatanacharoen et al., 2011].

1.1 Motivation

There are a number of situations where the nature of a system changes to such a degree that the fundamental dynamics we might normally use to simulate the system are inappropriate. In models of animal populations, it may be that the number of individuals may have climbed to the point that inaccuracies in individual-based models begin to dominate the dynamics of the system, or that the number of individuals has declined to the point that an analytic representation is unable to capture the dynamics of a small population. We ought to be able to reduce our model error by choosing appropriate representations for the conditions, and to limit the propagation of error into components that depend on other submodels. Such partitioning also makes error estimation more straightforward.

We can also find examples systems that are subject to serious perturbation arising from small deviations from their "nominal" condition. Traffic flow, for example, can be dramatically constrained by obstructions (sudden or otherwise) on the roadway with consequences that spread quite a long way from the source. Another potential source of perturbation which may have significant effects is a change in behaviour in some subset of components of the system. In the context of biological systems, this is particularly so if the change impinges on the viability of individuals or their ability to reproduce.

Frameworks have been constructed to assist the development of agent-based models, notably Swarm [Minar et al., 1996], MASON [Luke et al., 2004], NetLogo [Wilensky, 1999], RePast [Collier et al., 2003], and AnyLogic [AnyLogic, 2001]. These frameworks or toolkits were the first major software packages designed to make the process of constructing agent based simulation models simpler. Many of them also support the blending of modelling paradigms

(systems dynamics, discrete event simulations, and individual-based models, for example). These toolkits have a broad user base and include support for essential facilities such as generating graphic output or interaction with GIS systems: their focus is on making robust tools available to modellers across a range of research domains. They have, however, no explicit support for models where the representations of entities may actually change their form. As mentioned above, we are comfortable with real-world systems where things undergo dramatic changes: water freezes, some insects go from egg to nymph and nymph to adult as part of their life-cycle: it seems a natural conclusion that simulations of these things may benefit from an analogous process.

1.1.1 Implications of the effects of behaviour, vulnerability and change

Migrations are common in many populations, often for reasons such as breeding, resource scarcity, and the encroachment of competing species, such as Homo sapiens. These examples are relatively predictable, and typically involve a homogeneous response from the population. Similarly, acute exogenous factors, such as changing environmental conditions, such as the seasonal variations, weather, or the availability of water or prey, can significantly alter the behaviour of organisms. The responses to these stressors are not necessarily uniform in the population. Ward and Krebs [1985] discuss the behaviour of lynxes in response to declining prey populations. In this study, two distinct behavioural responses prevailed: some animals choose a nomadic lifestyle as a means of optimising their likelihood of hunting success, while others remain in their own territory. Though Ward & Kreb's sample size was small, the distinct responses suggest that lynx populations respond to prey scarcity in a heterogeneous way and, as a result, may be less amenable to modelling as a single population. Models of the effect of trapping on the lynx population which do not represent these two behaviourally distinct populations differently would have difficulty producing results which were consistent with field studies.

Zala and Penn [2004] present a useful review of the effects of behaviour disrupting contaminants in over thirty different species of vertebrates studied in the literature. In this review, nearly half of the noted behavioural changes amongst the species could impair their reproductive capacity, and a number of them involved impairment of cognitive abilities, movement or ability to foraging. Simple models of systems with contaminant uptake, and its non-lethal effects may be unable to capture important dynamics in the populations when contact with contaminants may be both indirect and behaviour dependent at several trophic levels. If a species is particularly vulnerable to low levels of contaminants, the flow-on effect of trophic transport may be very difficult to model in any way other than with an individual-based model. Swan et al. [2006] has found that very low levels of the anti-inflammatory drug diclofenac are fatal to old world (*Gyps spp*) vultures which acquire it by scavenging carcasses of dead cattle. In cases where there may be altered behaviour or fatal toxicity at low levels, traditional analytic approaches to modelling the transport of contaminants through the food chain may be inadequate since the behaviours exhibited by the populations are no longer homogeneous, and alter-

native representations for affected and unaffected portions of the populations are indicated.

When an altered behaviour or biological efficiency is associated with the spread or reproduction of organisms, the scope for destructive feedback is increased, and the dynamics can diverge rapidly from representations that are adequate for an unperturbed system. The effects of *Toxoplasma gondii* on rats [Berdoy et al., 2000] is an ideal example: rats inoculated with the parasite (usually by contact with infected cat faeces) lose their innate fear of cats. The positive reinforcement on the spread of T. gondii afforded by this leads to greater potential for the pathogen to infect more cats, and hence, more rats. The behaviour of the rats is clearly not consistent with the behaviour one would usually encode in a model of a rat population. In a similar paper, van Dobben [1952] observes that roaches (*Rutilus spp*) infected with *Lingua intestinalis* were three to five times more numerous in cormorant catches than in the roach population of the IJsselmeer[2] (as estimated from commercial catches), suggesting that something in the fishs behaviour makes them more susceptible to capture. In these cases, special steps would be needed to to model the populations, either analytically or in simulation, to avoid underestimating both the mortality of the prey species and the presence of the pathogens in the definitive host.

Dobson [1988] contains a brief summary of literature that addresses parasitic infection which alters the hosts behaviour to benefit the parasites reproduction. Particularly useful are the analyses of the consequence of the interaction for both the definitive host that parasites breed in, and intermediate hosts which are used to reach the definitive host[3]. In a broader study, Poulin [1994] assesses the effect on host behaviour in a number of host-parasite pairings, and found that the parasites had a significant effect on the behaviour of their hosts.

These papers concerning the effect of parasites on host behaviour suggest that the influence of parasitism on populations may be greater than we might expect. Lafferty et al. [2006] argue that parasitism is the dominant mode of trophic interaction, and that their role is significantly absent in the literature. The implication is that for trophic models of any sort, there needs to be a body of robust techniques to manage the transition of parts of a population from uninfected to infected and some means of dealing with them which preserves their properties appropriately. While Dobson provides an analytic example, it is clearly limited by the requirement of sufficient population sizes to make the analytic representation tenable its application in situations where local extinction is a possible outcome may not be tenable.

It is also possible for populations which are afflicted by behaviour modifying

[2] A large lake in the Netherlands.

[3] The author also presents several systems of differential equations which describe the population trajectories of different ways the parasitism occurs, their points of equilibrium, and he also discusses the derivation of these systems. Equations like these can be used as the basis for interaction between different levels of aggregation (see Section 5.7.2 for an example using integral equations to make gape limited predation by individual on populations select the correct sub-population).

parasites to *also* be vectors for contaminants which affect their predators. Managing the resulting cascade of influence may have particular relevance when considering strategies for the management of vulnerable species. An appropriate expression of behaviour influences reproduction and predation dynamics and is thus essential in simulation models.

When a significant portion of a population is infected with these types of parasites, we can no longer treat the population as homogeneous, and the rates for reproduction and mortality vary significantly between infected and uninfected sub-populations. In this scenario, there it is a reasonable assertion that infected portions of the population should be treated separately from the uninfected population.

1.1.2 Implications of the effects of social dysfunction

The situation is more complex when there are endogenous reasons for fundamental changes in an organisms basic dynamics. Unnatural situations can induce radical, even pathological, changes in behaviour. Social animal populations may behave in quite strange ways when their population density grows too large, there are stressors which are alien to them, or the populations social profile is disrupted. A seminal (and *grim*) example of this is described in Calhoun [1973]. Calhoun recounts an experiment in which mice are confined in a domain where all their physical needs were met, all possible sources of mortality apart from senescence and death by injury were excluded, and there was no possibility of emigration. Social and behavioural disintegration began to manifest in the third generation (day 315), and the population went into terminal decline after 560 days, and for all practical purposes the social organisation had collapsed utterly.

Calhoun discounted the population density as the cause of the social disintegration, rather attributing the collapse to the inability of young adults to engage in normal roles due to high competition for the *social niches* which were filled by older, dominant mice. The behavioural changes attending the social collapse did not revert to more normal behaviour when population levels dropped (past 560 days). This type of response to social conditions may be pertinent for other species, and it has clear implications for simulation models of social animals. Behaviour associated with social settings is, almost by definition, context dependent. This paper illustrates that, at least in social animals, dramatic changes in models of behaviour may be required, and that those changes may be dependent on the *social context* rather than the physical environment. The social connections maintained may be as important to the survival of an individual as access to shelter or food in some circumstances.

1.2 Comparable work

In environmental modelling, there has been gradual trend to increase the representational correspondence between models and their biophysical analogues,

both in terms of basic biological activity and in their interactions with other elements in the simulated domain. This comes at a cost. Tracking the fine scale modal properties more closely requires decision or selection strategies which ensure that the models behave in ways that are appropriate for the context.

Many individual-based models incorporate environmental characteristics that influence the behaviour of the individuals simulated. Botkin et al. [1972a,b] and DeAngelis [1978] are important early examples: Botkin et al. dealt with systems that modelled the effect of spatially explicit environmental conditions on simulated trees (rather than stands or coupes) in a mixed species population in North America; in the case of DeAngelis, the model was used to explore the distribution of fish (modelled as individuals) in a speculative body of water with a known distributions of temperature and food availability. Both of these models simulated the dynamics resulting from the physical conditions real plants and animals might encounter and they reflected observed patterns to a striking degree. Many processes occur over short time spans, small domains or both, and these models provided a means of simulating these processes over appropriate scales. These models were successful because they they increased the fidelity of their representation, and were able to generate results that were more intuitively accessible to non-experts because they reflected observable processes. They also represent the scientific realisation that traditional population-level models relying on a significant degree of homogeneity may not be appropriate in heterogeneous environments. What they demonstrate is that the form a model should take ought to be sensitive to the conditions and dynamics which prevail in the system being simulated.

Hu and Edwards [2005] describe a model of crayfish is constructed in the DEVS framework using component submodels which competitively assess various behaviours and select one or more of these behaviours based on threshold values as an appropriate response to the environment and the state of the crayfish agent. An essentially similar strategy for behaviour selection was used in in Lyne et al. [1994b] for unimodal behaviour selection and in Gray et al. [2006, 2014] for restricted multi-modal behaviours. These works base the execution (or not) of particular branches of code model on assessments comprised of their own state and that of their environment. In this way, they bear a similarity to the model selection described in subsequent chapters; while this approach to nuanced simulation is only focussed on the behaviours evinced by the agents, it foreshadows modal representation dictated by dynamic assessment.

The coastal marine ecosystem models based on the InVitro framework [Gray et al., 2006] used different representations for the organisms to based on their life-stages. As organisms that comprised the benthic habitat matured, their representations would change to suit their niche in the system. In this case the sequence of transitions was determined before compilation of the model. This is an early attempt to select a representation for niche which was appropriate for the dynamics it was expected to demonstrate, and the computational aspects it was to optimise (such as run-time or fidelity). This model was in most other ways similar to conventional agent-based modelling of the time.

Bobashev et al. [2007] describes a model of epidemic simulation in which the representations of populations or portions of populations are decided based on the number of infected individuals relative to a nominated trigger value. This model demonstrates that there is a demonstrable advantage to changing representation in terms of computational efficiency and the fidelity of the model. The published model in Chapter 2 is similar: when individuals with contaminant loads in Chapter 2 leave the region where contamination is possible, they are subsumed back into the population and their data is incorporated so that the individual contaminant profiles are maintained and subject to depuration. Both of these models are an improvement on traditional individual-based and equation-based methods: local conditions are able to affect the outcome for individuals, and the computational load of individual-based simulation is reduced when appropriate. The aim is to preserve the fidelity of the individual-based data, while benefiting from the speed and mathematical tractability of an analytic or numerical model. In both of these models, the triggers for change are fairly basic – numbers of infected individuals or presence within the zone of potential contamination.

Much like Bobashev et al. and the model in Chapter 2, the model described in Wallentin and Neuwirth [2017] is a hybrid predator-prey model that exhibits model switching. The model simulates a lake with fish and plankton, switching between equation-based lake-wide representations and agent-based representations for fish at either a sub-region level or as individuals, and plankton were represented either by a global model or by local cellular-automata. Again, changes in representation were by transitions past nominated trigger values. These swapping strategies are ideal for models which focus on a limited set of simple dynamics, but whether this is adequate for more complex ensembles is unclear.

The pedestrian detection and tracking system of Zhang et al. [2016] selects the algorithms and parameters to be used in the analysis of segments of data based on the nature of the data it is given. Their approach has qualitative similarities to the approach suggested in this work, since the whole method of evaluation changes based on its input, rather than adjusting the scales or domains. They accomplish this by training the selection system off-line on known data. Their selection system corresponds to a member of particular class in Remodel called a *monitor*. The *monitor* agents are fundamental to the automatic control of the mix of representations in the running model: they interrogate sets of agents and are able to initiate changes in representation. *Monitors* are able to make use of a corpus of sets containing *known-good* and *known-bad* configurations: a situation which corresponds closely with Zhang et al.. The *monitors* also have the ability to adjust this corpus according to the dynamics at play within the simulation and so may be reactive as well as adaptive.

Vincenot et al. [2011] provides a good overview of many of the features and issues attending models which combine representations from different paradigms. The authors argue that rather than treating the discrete and the analytic (or near analytic) models as opposing paradigms, they have the potential to model common domains more accurately with a more broadly accessible form, and possibly with greater efficiency. In spite of the fact that awkward issues may

remain[4], it provides a road-map with signposts to aid newcomers to the domain.

There is clearly a growing set of models which demonstrate that adaptive hybrid models are both feasible and worthwhile. However, the mechanisms behind changes in representation in the literature are both simple and largely based on local state information. The exploration of the issues associated with the maintenance of state information across transitions is also largely absent. There is no systematic support for transitions in response to the needs of other components of the systems, or mechanisms that enable systematic collection of data regarding the performance of the model. Most importantly, there are no fundamental code-bases from which model development can proceed. The work that follows addresses these deficits and constructs a more comprehensive framework that is able to support models with complex conditions related to representational transitions, and to support the maintenance of state data across transitions.

1.3 Structure

Models of complex systems usually incorporate alternative code paths or expressions in a component to deal with situations where there are fundamentally different dynamics or properties by testing for these conditions at each potential fork in the code. In many cases, this sort of control flow may decide between a number of paths, but the selection protocol and consequent paths are all hard-coded within the model. Such a structure can engender a complicated network of potential execution paths through the model. In contrast, the approach discussed in this thesis addresses this problem by constructing the models as an ensemble of agents which can cede their role to another representation which is more suitable when the need arises. A resident population might be represented by a single *population* agent, a set of *individual-based* agents, *super-individual* agents or by some mixture of these representations.

Models of complex systems usually incorporate alternative code paths or expressions in a component to deal with situations where there are fundamentally different dynamics or properties by testing for these conditions at each potential fork in the code. In many cases, this sort of control flow may decide between a number of paths, but the selection protocol and consequent paths are all hard-coded within the model. Such a structure can engender a complicated network of potential execution paths through the model. In contrast, the approach discussed in this thesis addresses this problem by constructing the models as an ensemble of agents which can cede their role to another representation which is more suitable when the need arises. A resident population might be represented by a single population agent, a set of individual-based agents, super-individual agents or by some mixture of these representations.

The decision to change the representation of a submodel occupying a niche

[4]Such as might be posed by situations analogous to gape limited predation between equation-based and individual-based entities in marine ecosystems

in the model should be based on the state of the system, the capabilities (or incapability) of the agents in the system and the objectives of the modeller in configuring a model, we might prioritise speed over accuracy for a real-time simulation, a computer game or a combat training simulator, and for a scientific extrapolation of the state of a harbour for each of a number of development scenarios, we might choose accuracy over speed.

Representing the state of the model, either as a whole or of its constituent parts, is not simple: not only may the submodel mix in the niches of the model vary through time, but the dependences of submodels on other components and niches may change as the state of the model changes. A model which contains a whale spotting tourism venture may follow the activities of whales at particular times of the year, but be utterly indifferent to the whales at times when there is little likelihood of their presence in an accessible location. Similarly, the association of entities represented by a population-based model may need to be maintained if the population disaggregates into agents based on super-individuals (small cohorts) or agents representing individuals.

The interplay of factors like these make a simple vector-based encoding mechanism for the states of a model and its components awkward, thus we turn to a metric space whose elements are trees with a finite number of weighted, labelled nodes. This simplifies the comparison of possible ways to fill the niches in the model for a given global state, and makes available any algorithms (particularly useful are clustering) which depend only on the properties of a metric space.

The decision to change representations can be made by an agent that recognises that it is unable to continue in the conditions in which it finds itself (akin to the code-path decisions in more traditional models), or by a similar assertion from some higher agency (a monitor in the discussion which follows) which assesses states more broadly. In the case of an agent determining that it needs to change, such as a penguin moving from the nestling submodel to the juvenile submodel, this can be effected directly, though the more general (and in this case, burdensome) strategy would for a monitor to flag the desired state change and then act on it when appropriate.

The models explored in this work bear a resemblance to a multitasking operating system, but, unlike an operating system, the models kernel must maintain temporal ordering of agents' start times within the agent queue, and this order must be strictly non-decreasing. Interactions between agents are largely mediated by the kernel and new agents may be created or removed with relative ease. Choosing this as an organisational template means that there are many patterns to serve as templates for further development.

All of the submodels in the example model presented in Chapter 5 are able to act to some degree as a kernel themselves in a sense, models can be nested.

1.4 Scales

The natural time step or spatial scales of a model may change if one or more of its constituent submodels changes its representation. This seems like an obvious statement, but many models are structured with quite carefully chosen time steps and spatial scales, and they may behave poorly when these scales are changed. Examples of this, such as Lyne et al. [1994a], Xu et al. [2001], Zhang and Montgomery [1994], model systems where the appropriate time-step is intimately linked to the temporal scale of the system being studied.

Component submodels of the models in Lyne et al. [1994a] (and of Gray et al. [2006] and Gray et al. [2014]) were tested during development for the effect of length of an individual's time-step on the component's dynamics. The results of the first of these models prompted the use floating point variables for time in subsequent models, and time-steps with intervals determined by a continuous function based on the individual's conditions, activity and interactions. In Lyne et al. [1994a], the smallest time-step was conditioned by the interval where a simulated organism's contaminant uptake was reasonably stable: steps which were too long either over-estimated or under-estimated potential uptake, and steps which were too short suffered both from computational inefficiency, and accumulated error in the movement of the modelled individuals. Xu et al. [2001] found that simulations of mesoscale convective systems were sensitive to the size of the time-step, even though the model remained numerically stable. They found that the predicted precipitation could vary by as much as 50% when the time-step was reduced from 225s to 50s, and attributed the change to the dependence on the time-step size of the calculations associated with the horizontal diffusion. Zhang and Montgomery [1994] found that grid-size for digital elevation maps in hydrological models had a significant effect on the results of simulations.

Fulton et al. [2004b] examines the effect of spatial resolution of the dynamics of marine populations using two models of a large bay. The domain is partitioned into regions (boxes) filling the domain both vertically and horizontally. Two particularly important observations are made: coarse spatial resolutions in the model lead to a simplification of the trophic web relative to fine-scale simulations, and the spatial resolution must reflect the dynamics of scale of the dominant gradients and processes in the system. This sentiment is also reflected by a conclusion in a paper produced by the FAO,

> At the end of this process the necessary components will need to be represented at the appropriate scales in prototype or final model(s). It is important to reemphasise here that there is no one single right model. All models have problems and it is best (where possible) to use a range of models that can address the question in different ways.

Food and of the United Nations [FAO], p. 78. While these sentiments are made with conventional modelling strategies in mind, they apply equally well to the constitution of a model as it runs.

Generally, representations of individual organisms are likely to require much smaller temporal and spatial scales than representations at a population level, so a model which seeks to accommodate both possibilities must be able to accommodate the scales that are important at that point in the simulation. For temporal scales this means providing the infrastructure to 'buffer' the activity of agents with longer time-steps, and to ensure that – as far as possible – the temporal discrepancy amongst the agents is minimised. Agents with long time-steps will necessarily often be ahead or behind agents with shorter time-steps. Changing spatial resolution seems relatively straightforward, but the errors that accompany spatial misregistration or integration over an inappropriately interpolated domain can become significant.

Changes in temporal scale can be more problematic, however. Time influences causality in a way that space does not. Deciding how to manage the flow of time in an ecological simulation model is one of the first decisions in its design. Many ecological models have been constructed as a large set of arrays containing state variables which are inspected and updated in the body of an event loop (or many loops). Some models achieve temporal optimisation by dividing the arrays into various groups of fast-stepping and slower-stepping variables, only dealing with the necessary parts of the system at each time step (*variable speed splitting* as in Walters et al. [2000], for example). Gray et al. [2006] and Gray et al. [2014] allow agents to dynamically determine their own time step based on their state – time steps may be truncated, or changed for their next turn in response to their situation. There is a trade-off in this: with a variable speed splitting approach, we can calculate all values based on a temporally coherent set of data, and update them all in one pass; in contrast, the dynamic time stepping approach means that each interaction is essentially conducted in isolation, and the consequences of a set of interactions may be dependent on the order in which the interactions occurs. Both variable speed splitting and dynamic time step selection are flexible enough to support representational changes for entities, but the greatest advantage comes from constructing the submodels to be robust with respect to arbitrary time steps over a reasonable domain. If a model is consistently run with time steps which are too long or too short, the model or system needs to be able to initiate a change to a more appropriate representation.

Inappropriate or incommensurate time steps can pose a real problem: while the interactions between submodels with short time steps and submodels with long time steps may be managed, at least to some degree, by accumulating changes to the slower model and applying them during the slower model's time step, this is not an ideal solution. One of the major risks this approach poses is a of distortion of resource availability which is dependent on the order in which agents are executed. This sort of error can artificially inflate or deplete apparent resources in a seemingly random fashion, and render the results of the simulation useless.

The principles which have guided the coupling of models remain salient, particularly those aspects associated with issues of coherence in time and space. While matching time steps isnt essential, the discrepancy between the time steps of interacting models should be limited by the magnitude of the changes

which may occur as a result of interactions large changes may call for small time steps.

1.5 Outline

The paper[5] which forms the body of Chapter 2 develops a model of organisms that periodically move through a region subject to plumes of contaminant. The model is capable of modelling the organisms either with a population-based representation or with an individual-based representation. This model is run in three configurations: purely population based, purely individual-based and as a hybrid where the individual-based representation is used when it is possible for any of the population represented to come into contact with the contaminant, and with the population-based elsewhere. An essential notion that was treated lightly in this paper is developed much more fully, namely that for a model to allow an oscillation between representations, additional data must be passed between them, maintained and possibly adjusted in order to preserve consistency across transitions.

The purpose of the model is to demonstrate the feasibility and utility of maintaining the contaminant loads (more generally *state data*) while running in a different representation and to compare both the execution speed of the simulations and the fidelity of the simulation with respect to the contaminant loads of the simulated population. The study found that the trials which alternated representations based on the proximity to the contaminated region was consistent with the individual-based trials, but performed with a computational speed of the same order as the population-based trials.

Chapter 3 was published[6] in a special issue of *Frontiers in Environmental Science*. It considers a small three species ecosystem and explores the properties needed for a more complex evaluation of possible model-swapping configurations of a running model, and develops a thought-model as a platform for discussion. To support the dynamic assessment and selection of model configurations, the paper introduces a metric space based on a tree structure. The metric space allows us to calculate distances between configurations and to reduce our potential search spaces by identifying clusters of representations that are largely similar in their constitution.

Chapter 4 presents a modified version of the appendix included in Chapter 3. The modifications make the structures easier to manipulate, and code to perform mathematical operations on these objects forms the basis for the selection process in the realised modelling framework and the example model discussed in Chapter 5.

There are many consequences which arise from the premise that the agent or set of agents which represent part of a simulation may change and that the synthesis of what they represented may be represented by a different set of

[5] https://doi.org/10.1016/j.envsoft.2011.08.012

[6] https://doi.org/10.3389/fenvs.2015.00058

agents. The discussion in Chapter 5 seeks to highlight and explain these consequences and the response to them, referring both to the Remodel framework and to the example model that was described in Chapter 3 and built within the framework. Currently the framework is functional, but still requires a more fully developed set of transition mechanisms, basic classes (for things other than animals, trees and monitors), and is lacking in a broad set of example models. Many of the functional niches in Remodel have more than one possible representation, and these representations can change in response to their own state, the states of other agents, or even the requirements of other agents. Where possible, the options for model selection differ in an important and fundamental way: they span the range from individual-based representations, through intermediates which represent a number of individuals, to conceptually continuous models. These changes are, in some sense, analogous to the transition from the set of integers to the set of real numbers. A model which demonstrated the utility of adaptive representations without changing the "cardinality" of the system would have been much simpler, but it would have missed some of the most important parts of the problem.

— — —

The corpus of code in the framework is a little under 28,500 lines of Scheme code and is freely available at

`http://github.com/snarkypenguin`

The interpreter/compiler used in this work is Gambit developed by Marc Feeley. It has a thriving community, excellent support and integrates well with C and C++, and is available from the GitHub repository

https://github.com/gambit

While the model does not require SLIB by Aubrey Jaffer, SLIB provide a broad range of useful functions. SLIB is available at

http://people.csail.mit.edu/jaffer/SLIB.html

———

Dr Simon Wotherspoon was credited in the papers which comprise Chapters 2 and Chapter 3, in acknowledgement of his salient advice to me on how to approach the task of writing papers both for journals and for a broader audience, and how to avoid getting distracted by the little stuff.

The corpus of code in Remodel is a little under 28500 lines of Scheme code and is freely available at http://github.com/snarkypenguin/Remodel.git

The interpreter/compiler used in this work is Gambit developed by Marc Feeley. It has a thriving community, excellent support and integrates well with C and C++, and is available from the GitHub repository https://github.com/gambit

While neither the framework nor the model requires SLIB by Aubrey Jaffer, SLIB provides a broad range of useful functions.

SLIB is available at http://people.csail.mit.edu/jaffer/SLIB.html

CHAPTER 2

Increasing model efficiency by dynamically changing model representations

2.1 Prologue to the paper

The body of this chapter is a (verbatim) paper published in the refereed journal *Environmental Modelling and Software* (Gray and Wotherspoon [2012]). The model discussed in this chapter is built using an older, much simpler body of code than the framework that is explored in Chapter 3. While the primary purpose of the paper is indicated by its title, a significant contribution is its development of the basic mechanisms for changing representations.

The source-code can be obtained from

https://github.com/snarkypenguin/Model-Efficiency.git.

The paper explores a model of marine organisms that periodically migrate through an intermittent plume of some contaminant. The uptake and depuration of the contaminants in the organisms are modelled, and the paper compares the results and the run-time for the system in three forms in order to establish how useful using model switching may be in terms of run-time and fidelity. The three configurations tested are

- a purely analytic representation of the migrating population – In this configuration a population whose relative locations follow a Gaussian distribution is moved around the migratory circle, and the uptake of contaminant is calculated using a Runge-Kutta4 algorithm
- a purely individual-based model – Individuals move through the contaminant zone, integrating their contact with the plume and generating an uptake level appropriately

- either a population (as described), a set of individuals (also as described), or as a mix with part of the cohort represented as individuals within the risk zone, and the balance represented as a population – Here, individuals are generated (and removed from the population) as the population disk encroaches on the contaminant zone. As individuals leave the contact zone, they are subsumed by the population and their individual contaminant level is decayed appropriately.

The analytic submodel is the simplest of the representations, consisting largely of a value (or vector) which records the contaminant load, and the number of individuals which are represented. A single instance of the analytic submodel is present in both the purely analytic model and in the switching model. The individual-based model incorporates a number of state variables, such as velocity, contaminant load and the instantiated agents are independent of the analytic representation.

In order to avoid confounding the results, the contaminant is inert since including toxicity effects would have the potential to alter both the number of entities modelled and their behaviour (such as movement rates); since slower individuals would take longer to move through contaminated regions, their likelihood of contact would be higher. The scenario in the model would be substantially similar to simulating the uptake of isotopes associated with particular geographic locations.

Given the very predictable dynamics of the modelled entities, the inclusion of multiple contaminants or multiple sources, such as in Gray et al. [2006, 2014], seemed unlikely to do anything surprising.[1]

[1] incorporated mortality or morbidity associated with contact, the mortality strategy used in Gray et al. would have been an appropriate choice.

Increasing model efficiency by dynamically changing model representations

Randall Gray[2]

CSIRO Division of Marine and Atmospheric Research

Simon Wotherspoon

University of Tasmania

Abstract

There are a number of strategies to deal with modelling large complex systems such as large marine ecosystems. These systems are often comprised of many submodels, each contributing to the overall trajectory of the system. The balance between the acceptable modelling error and the run-time often dictates the form of these submodels. There may be scope to improve the position of this balance point in both regards by structuring models so that submodels may change their algorithmic representation and state space in response to their local state and the state of the model as a whole.

This paper uses an example system consisting of a single population of animals which periodically encounters a diffuse contaminant in a localised region as an example of such a system, and discusses the key issues that arise from the approach.

2.2 Introduction

There is a body of literature stretching back several decades which discusses individual-based modelling as a useful alternative to classical models. Early examples modelled forest canopy dynamics, notably JABOWA and its derivatives Botkin et al. [1972b,a]. The number of significant papers and books has steadily increased since the 1980s. These works describe the use of individual-based models across a broad range of systems, and the relative strengths and weaknesses of the approach (such as Huston et al. [1988], DeAngelis and Gross [1992] and Grimm and Railsback [2005]). Classical models exploring populations and ecological systems are usually associated with modelling the dynamics of large groups and arguably appeared at the end of the eighteenth century with Malthus's ***An Essay on the Principle of Population*** (1798). The

[2]Published in *Environmental Modelling and Software*, 2012
Corresponding author: `Randall.Gray@limnal.net` (Randall Gray),
`Simon.Wotherspoon@utas.edu.au` (Simon Wotherspoon)

properties of these models are well understood and their state variables usually correspond to measurable quantities. Often, they are much faster than individual-based counterparts, and the analysis of model error may be much more straightforward. Classical and individual-based approaches represent the ends of a spectrum of aggregation in time, space and membership. Representations lying between these extrema, such as described by Scheffer et al. [1995], capitalise on the process-fidelity of an individual-based representation and gain some of the computational efficiency of a more aggregated classical approach, but an adaptive exploitation of the strengths of different representations is possible and worth exploring.

Ecosystem models are becoming broader in scope (Rose et al. [2010], DeAngelis and Gross [1992], Harvey et al. [2003] **?**Fulton et al. [2004a], Gray et al. [2006], Gray et al. [2014]) and include more species with richer environments. The environmental response to climate change has also made anthropogenic pressure an important feature in many of these models. As this trend grows it seems less likely that a single model drawn from any particular region of this spectrum will be able to address all members and processes equally well. Simulation models often embed their subject in an "environment" comprised of primary data and other models and these components may occupy many places in the spectrum of representations. The model's actual implementation may be anything from a set of distinct models which are coupled together but retain their independence, to a corpus of code with the submodels so integrated that there is no real distinction between one "model" and the next.

The dynamics associated with biological and ecological systems can depend on the distributions and states of individuals in ways which are not amenable to equation-based modelling. The individual-based models described in Farolfi et al. [2010], and de Almeida et al. [2010] deal with systems of this sort. Versions of these models could be embedded in a common simulation environment in order to address more complex problems which span traditional domain boundaries, and such a model could address broader questions, such as how mosquito control strategies may best adapt to evolving agricultural practices and watershed conditions. Thiele and Grimm [2010] describes an extension to NetLogo which allows modellers to incorporate calls to R functions to aid in configuring the model to meet desirable mathematical conditions, to provide ongoing analysis, and to display the model's state through its run. This interface between R and NetLogo could be extended to support incorporating mathematical decision models written in R into the model's decision tree. The fusion of these three elements would form a system capable of simulating possible trajectories for the management of watersheds and human health in ways which would not be possible with a traditional monolithic modelling approach.

Models are including more functional groups and the interactions between components are becoming more detailed. It is costly in terms of computational load to address this increased demand for detail: individual-based models of populations may be very good at capturing vulnerability to exceptional events, but such simulations take a long time. Much of this time may be spent with the model in a largely unchallenging or uninteresting part of its state-space.

This paper explores the technique of changing the representation of a component of a model based on its location in its state-space. Modellers already do this to some degree: time-steps or spatial resolutions are changed, particular code paths may be by-passed to avoid pointless work, or additional calculations might be performed to reduce the error when the state is changing rapidly. These optimisations are largely optimisations of the *encoding* of the model or submodels, rather than an actual change in representation.

Vincenot et al. [2011] make a clear case for considering what the authors term "hybrid-models." They present four reference cases which they use to describe ways in which equation-based models and individual-based models might be coupled to increase their utility. Their categories of hybrid-models are: individual-based models interacting with a single system dynamics model, system dynamics models embedded in individual-based models, individual-based models interacting with a number of system dynamics models, and models in which the representation swaps between individual-based and an equation-based form. They argue that a hybrid approach may provide a means of increasing the speed and accuracy of our models; Gray et al. [2006], and **?** have demonstrated that large models of ecosystems can be modelled this way. Vincenot et al. note that they found relatively few models which use both individual-based and equation-based submodels, and they present no existing models representing their fourth reference case. This final case, where models swap from equation-based to individual-based, is briefly described in general terms and is clearly intended to encompass models like the model of this paper.

This "mutating" or "switching" approach to the problem of managing complex simulations was developed using the experience from making several large scale human-ecosystem interaction models (Lyne et al.; Gray et al. [2006]; and a current, larger study of Ningaloo coastal region (*work in progress*)). In each of these studies a significant component of the model focused on simulating the interaction between organisms and contaminant plumes, though there is nothing that inherently limits the techniques to these sorts of studies. Lyne et al. assessed the potential of contaminants originating in industrial waste percolating through the food chain into commercially exploited fish stocks. Gray et al. developed a regional model to assess management strategies for human activity which interacts with the biological systems along the Northwest Shelf of Australia.

Simulating contaminant interactions in an ecosystem is expensive in terms of run-time and memory use. The models described by Gray et al. and Lyne et al. include contaminant transport, uptake and depuration modelling, with behavioural sensitivity to contaminants. In Gray et al., the time taken to run a simulation with contaminants increased by roughly an order of magnitude, and in both studies a large amount of time was spent in regions where no interaction with contaminant plumes was possible. Monte [2009] presents a lucid discussion of analytic *contaminant migration-population effects* models. These models incorporate the movement of populations and their internal distribution, the transport of contaminants through the system via biotic and abiotic pathways, and the changes in behaviour and population dynamics associated

with contamination. Monte discusses a method of coupling the equations which govern contaminant dispersion with the equations for population dynamics and migration. The technique depends on the equations of the location and the dispersion of members of a population satisfying an independence condition with respect to time and location which must hold. He states that the class of systems where the "movement of animals, the death and birthrates of individuals in $\mathbf{x}$ [location] at instant t [time] depend on previously occupied positions" is not generally amenable to the approach and suggests that repeated simulations of many individuals is an appropriate way of dealing with this situation.

It is unnecessary to run a complex model and carry the burden of maintaining its state when a simple model may perform better. If representations are switched appropriately, there is potential for improvements in run-time and accuracy. We need to consider four basic questions to do this:

1. What data need to persist across representations?
2. When should a model change representation?
3. How is the initial state for a new representation constructed?
4. How should the error associated with the loss of state information be managed?

The answer to these questions is specific to the set of submodels in question. Before expending resources and effort on a large scale model there needs to be a demonstration that the notion is worth pursuing, and some indication of how it might be accomplished. The aim of this paper is to provide this demonstration rather than to develop a comprehensive body of techniques supporting the approach. Many systems may benefit from similar techniques; obvious candidates are models of marginal populations, and the population dynamics of animals with behaviour where short periods of time have a significant influence on population levels (Wolff [1994] and Elderd et al. [2008], for example).

2.3 Overview: an *ODD* model description

The *ODD* protocol [Grimm et al., 2006] is used to describe the example model. We discuss the issues associated with making such a system, strategies and the reasons behind them in the Discussion section.

2.3.1 Purpose

This example model plays two roles. Its first is as an explicit demonstration, and the second is as a tool to explore the larger subject of changing a model's representation in response to its state. This example is overly simple, but it

shares a number of features with plausible models and the analysis and development of the mutating model should be a reasonable template for other systems.

The model simulates organisms moving along a simple migratory path which intersects a region containing a field of fluctuating contamination (see Figure 2.3.1). This model exhibits fundamental attributes of larger studies of pollutant/ecosystem interactions (Lyne et al. and Gray et al.) and, while it is not intended to accurately represent any particular system, it might loosely correspond to some body of water influenced by contaminant loads associated with terrestrial runoff resulting from intense rainfalls.

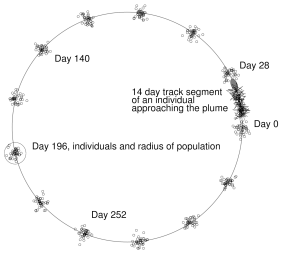

Figure 2.1: Snapshots of individuals' locations at 28 day intervals superimposed on the migratory path. The plume's contact domain is marked by a grey ellipse near the position of individuals at day 28, with the track of a single individual approaching it. The domain of a population is is circumscribed around the individuals at day 196 for comparison.

The test models are composed of one or more submodels which run within a simple time-sharing system. Each submodel runs for a nominated period of time and passes control to the next submodel, very much like tasks running in many modern computer operating systems. In a *mutating* configuration, a trial will have different models take turns representing components of the system.

The population-based and individual-based submodels have been kept as similar as practicable in order to minimise the sources of divergence.

2.3.2 State variables and scales

There are essentially three distinct submodels in the simulation: an individual-based representation of the migrating group, population-based representation of the group, and a contaminant uptake-depuration model. We can think of the models which take the role of the group as candidates for filling a *niche*, which we can think of as the "sub-model shaped hole" in the middle of the program. Because the individual-based and population based models have fundamentally different spatial representations, each of these models include mechanisms to evaluate their contact with a plume as they move through their environment. The spatial domain of the whole model system is a circular region with an arbitrary radius of somewhat more than 100km which encom-

passes both the area influenced by the contaminant source and the annual migratory path of the organisms. The plume can be viewed as a forcing function in the model and it has a maximum footprint area of approximately $43km^2$ which may be circular or elliptical and is centered on a point of the migratory circle. Both the elliptic and circular variants of the plume have the same area, and their intensities are adjusted so that the integral of the contaminant concentration over the region is the same.

The individual-based representation maintains a contaminant load associated with contact with the plume, a location, a direction and the next time at which it is scheduled to run. The population-based representation treats the group as homogeneous with respect to all state variables other than the contaminant load, and maintains only a record of its next time-to-run and an indication of contaminant load in the population. In the straight population-based representation, this is a single value, but in the mutating system the submodel maintains a list of contaminant loads which correspond to the non-zero loads of individuals. The plume model is deterministic with respect to time and location and maintains no state variables.

2.3.3 Process overview and scheduling

Simulations were run with 90 minute time-steps for a period representing twelve years. At each time-step, each instance of a submodel is rostered in a priority queue sorted on the "time-to-run" state variable, and when it comes to the top of the queue it executes.

Populations operate in a straightforward way: their path is deterministic, exposure to contaminants is calculated, and the resulting values are fed through the uptake-depuration equation. Individuals calculate their path (a segment of a directed random walk which follows the path of migration) and contact for the time-step and an apply the uptake-depuration equation. At the end of a time-step in non-mutating configurations, data is accumulated for output and each submodel reinserts itself in the priority queue. Otherwise, a heuristic is used to choose an appropriate representation for the niche in next time-step and that is inserted into the queue. Randomisation within a time-step is unnecessary, since the individual's or the population's contaminant updates are resolved for the contaminant contact across their time-step and are not dependent on the state of any other agents.

2.4 Design concepts

The central reason for the model is the mutability of the representation of the simulated organisms. Individuals and populations in the model are profoundly simple: no real scope is present for any of the trait categories mentioned in Grimm et al. [2006], apart from their interaction with the contaminant plume, though this interaction is completely deterministic with respect to their path through the plume. In place of these traits, we have the basic heuristics

associated with triggering a change from a population-based representation to individuals and a corresponding heuristic which indicates when an individual should join (or become) a population. The actual mechanism which turns a population into individuals or its converse is not necessarily a property of those models. Since the objective is to examine the impact of changing model representation in a fairly narrow situation, no attempt is made to optimise the submodels in the "non-contact" areas which constitute most of the model domain.

2.5 Details

2.5.1 Initialisation

Individuals and populations initially begin with no contaminant load, and individuals are positioned according to the two-dimensional normal distribution which characterises the population's assumed distribution. When a population mutates into an appropriate set of individuals, the individuals are positioned in the same fashion (centered on the centre of the population) with their corresponding contaminant loads either taken from the list of non-zero contaminant loads maintained by the population or initialised to be zero should the population's list fall short.

2.5.2 Input

Several characteristic features of the model are determined by the time and location represented. The contaminant intensity (and hence extent) at any point, $\mathbf{r}$, relative to the centroid of the plume, $\mathbf{m}_{plume}$, at a time, t, by the equation

$$I(t, \mathbf{r}) = \frac{1}{2}(1 + \cos(2\pi t/p)) \exp(-\psi\phi(\mathbf{r}, \mathbf{m}_{plume}))$$

where p is the period of 34 days, $\psi = 0.05$ is a decay exponent. We take ϕ to be a distance function, either $\phi(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}|$, for a circular plume, or $\phi(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b}) \cdot (\sqrt{2}, \sqrt{1/2})}$, for an elliptical plume. The effective radius of the circular plume in the model is about 3.7% of the circular migratory path of the populations and individuals. The intensity of the elliptical plume is adjusted by scalar multiplication so that the integral of I for the two plumes over their domain is the same.

The individual-based and population-based models follow a circular migratory path about the origin. The path is traced annually and its location at any given time follows the equation $\mathbf{l}(t) = 10^5 (\cos(2t\pi/365.25), \sin(2t\pi/365.25))$.

2.5.3 Submodels

Individual-based representation

Individuals follow a directed random walk around the migratory circle described in the previous section. At each time-step the stride the individual takes is calculated according to its proximity to the "target" on the migratory path. There are a number of parameters associated with the movement of the individuals presented in Table 2.1.

Table 2.1: Parameters associated with individual movement

Parameter	Value	Description
$\overline{V}$	4	A "variability" parameter associated with a Poisson-like process
q	0.5	A magnitude control parameter on directional change
μ_δ	1 day	Notional interval over which we calibrate individual's movement
μ	20km	Indicates the radius which is likely in a period of μ_δ
s	4ms^{-1}	nominal speed of the individuals

If we take δ to be the length of the current time step, and v to be a realisation of an event in a Poisson-like process with a mean of $\overline{V}$, we can take

$$Q = \left[1 - \exp\left(\frac{v}{\overline{V}} \log\left(1 - q\right)\right)\right]$$

to be a "variation" scalar which we use to evaluate an effective radial speed,

$$\nu_s = \left|-1 + \sqrt{1 + 4sQ^2 \frac{\delta}{\overline{V}}}\right| / 2Q^2.$$

Large values of Q correspond to long stretches of time without a change in direction, so we include Q in the calculation of α, the partial change in the individual's direction vector, by setting it to $\alpha = \pi rnd\left(-Q, Q\right)$. We can take their effective displacement over the 90 minute interval to be determined by a weighted sum of the normalised vector which joins them to their "target" location on the migratory path and a direction vector of length ν_s which is deflected by α.

Population-based representation

The population-based model assumes that a radially symmetric, normal distribution of individuals is an appropriate representation. Trials using the move-

ment model of the individual-based model were run, and the positions of individuals relative to their "target" on the migratory circle at each time-step closely matched a 2D-normal distribution with a $\sigma^2 = 3136.25^2$. Using this value, we define the density of the population at the point $\mathbf{p} = (p_x, p_y)$, relative to the population's centre, to be

$$\rho(\mathbf{p}) = S_L \frac{1}{2\pi\sigma^2} \exp\left(-\frac{p_x^2 + p_y^2}{2\sigma^2}\right)$$

$S_L = 1.015$ is a scaling parameter chosen so that the integral over the population's effective disk, $\mathbf{D} = \left\{\mathbf{q} \in Domain\,(\rho) : |\mathbf{q}| \leqslant 3\sigma^2\right\}$, gives

$$\int_{\mathbf{D}} \rho\,(\mathbf{p})\, d\mathbf{p} = 1.$$

Contaminant handling

Initially a contact value is calculated for the time step. For an individual, this value is the integral of the contaminant level over its path. Population contact is calculated in an analogous way over the domain of the population and it represents the average contact of the members of the population.

The mass of contaminant which is available for uptake, or contact is, for individuals, taken to be the result of integrating the intensity of the plume over its path, $\mathbf{P}_t$ to $\mathbf{P}_{t+\delta}$. Namely,

$$M = \int_{\mathbf{P}_t}^{\mathbf{P}_{t+\delta}} I(\mathbf{p}) \|\mathbf{p}\| d\mathbf{p}$$

where our variable $\mathbf{p}$ is a vector with time and location and we assume that the motion from $\mathbf{P}_t$ to $\mathbf{P}_{t+\delta}$ is along a straight line segment. We take $\|\mathbf{p}\|$ to be the speed at which the individual is moving.

Population's contact occurs across its domain and we calculate the definite integral

$$M = \int_{\mathbf{P}_t}^{\mathbf{P}_{t+\delta}} 2 \int_{\mathbf{\Omega}} I(\mathbf{p} + \mathbf{!})\rho(\mathbf{!}) d\mathbf{!} d\mathbf{p}$$

where $\mathbf{\Omega}$ is an area over which we assess the effective area of the population and $\mathbf{p} + \omega$ denotes the area $\mathbf{\Omega}$ translated so that its centroid corresponds to $\mathbf{p}$. The contact equations are solved using a simple adaptive quadrature routine. This value corresponds to the most likely mean contact in the population.

For both models of our organisms, uptake and depuration is modelled by the ordinary differential equation

$$dC/dt = uM - \lambda C$$

where $u = 0.02$ is the uptake rate, a decay rate which is approximately λ =0.0059. The equation is solved numerically with a fourth order Runge-Kutta algorithm for the value of C given a contact mass, M, and an initial contaminant value or vector of values for C

Mutating sub-models

The individual-based representation requires no change to run in a mutating configuration, but the population-based representation must maintain a list of contaminant loads which are processed in exactly the same way a scalar might be processed in one of the simple configurations. In the mutating configuration, each instance of a model is assessed at the end of its time-step to determine whether a change in representation is appropriate.

When a population dis-aggregates into individuals, the set of individuals with contaminant loads corresponding to the entries in the list are created. Their locations are normally distributed within the population disk. Any shortfall in numbers is handled by creating individuals which have no contaminant load and positioning them in the same fashion. Once the individuals are created, the population model is allowed to terminate.

An individual joins a population by having its contaminant load added to the list the population maintains. The first step in the process is to determine if there is a population close enough to the individual. If not, an empty population is created. Once a population's contaminant load has been inserted into the population the individual is allowed to terminate. The population model itself does not really play a part in this transaction: the "import" call is never used directly by the population, rather it is the supervising scheduler which organises the transfer to and from individuals and populations.

2.6 Results

The data presented in section 2.6.1 are based on two sets of simulations representing forty individuals. The first set uses a circular plume and the second an elliptical plume. These data sets allow a comparison of run-times, the equivalence (or lack of equivalence) amongst the submodels, and that provide data to examine the robustness of the representations to changes in the configuration of the plume. To ensure that run-time comparisons are meaningful all of the simulations in the first set of trials were run on the same computer.

The first set is comprised of forty trials of the homogeneous individual-based model, corresponding trials of the mutating model, and a single run of the population-based model. The second set is comprised of eighty trials of the mutating model and a single run of the population-based model. The data in the first set of trials establishes the equivalence of the homogeneous individual-based model and the mutating model. We pool the data from the mutating and homogeneous individual-based runs from the first set to match the eighty runs in the second to compare the effect of the plume's configuration. The results with an elliptical plume were not consistent across the model representations.

The individual-based representation produces a time series of contaminant levels for each individual, while the population submodel produces a "mean load" across a group of entities. The mutating submodel sits between the two, sometimes producing individual time series and sometimes mean time series

for varying parts of the population. We denote representations by a subscript $r \in \{i, m, p\}$, so that $C_{rkj}(t)$ is the contaminant load at t in time series, C, associated with individual j in trial k of representation r, $C_{rk}(t)$ is the mean at a time t over all the groups simulated in the indicated representation and trial, and $C_r(t)$ denotes the mean of $C_{rk}(t)$ across the k trials for the indicated representation. To compare the dynamics of the system we generate mean time series for each of the k trials in the individual-based and mutating sets, $C_{ik}(t)$ and $C_{mk}(t)$. We are careful to generate the correct mean in the mutating submodel from time steps which have a mixture of individual trajectories and mean trajectories from population-based representations. Each of the mean time series, $C_{rk}(t)$, corresponds to the mean contaminant load of the population, $C_p(t)$, produced by the population submodel; averaging them, that is constructing

$$C_r(t) = \frac{1}{k} \sum_{j=1}^{k} C_{rj}(t),$$

where r is one of i or m, is equivalent to running many stochastic trials and averaging to fit the population submodel. Using $C_{ik}(t)$, $C_{mk}(t)$ and $C_p(t)$ we find the maximum value attained for each representation, $\hat{C}_r$. We are also interested in the mean value across time of each representation,

$$\bar{C}_r = \frac{1}{T} \sum_{t \in T} C_r(t)$$

2.6.1 Contaminant load correspondence between representations

Both sets, $\bar{C}_r$ and $\hat{C}_r$, are presented in Table 2.2.

Table 2.2: Maxima and Means

$Series_r$	$\hat{C}_r$	$\bar{C}_r$
C_i	0.1787	0.0390
C_m	0.1821	0.0392
C_p	0.1387	0.0350

These data suggest that the mutating representation is consistent with the homogeneous individual-based representation. The population-based representation seems to present markedly different mean and maximum values.

2.6.2 Contaminant load variability

We calculated measures of variability in the time series using the aggregated time series $C_{ik}(t)$ and $C_{mk}(t)$ and their respective means across the k trials,

$C_i(t)$ and $C_m(t)$. We will take T to be the total number of time steps taken, and we take

$$\hat{\sigma}_{ab} = \max_{t\in[1,T]}\left[\frac{1}{k}\sum_{j=1}^{k}\left(C_{ak}(t) - C_b(t)\right)^2\right]^{1/2}$$

and

$$\bar{\sigma}_{ab} = \left[\frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{k}\sum_{j=1}^{k}\left(C_{ak}(t) - C_b(t)\right)^2\right]\right]^{1/2},$$

to be the maximum root mean square error and the average root mean square error. Clearly we can write $\bar{\sigma}_{rr}$ as $\bar{\sigma}_r$ without introducing ambiguity, and similarly for $\hat{\sigma}_r$. The values for these measure of variability are presented in Table 2.3.

Table 2.3: Deviations amongst the model runs with respect to a given mean

r.m.s.e.	$r = i$	$r = m$	$r = p$
$\widehat{\sigma}_{ir}$	0.0083	0.0084	0.0534
$\bar{\sigma}_{ir}$	0.0024	0.0024	0.0096
$\widehat{\sigma}_{mr}$	0.0090	0.0090	0.0538
$\bar{\sigma}_{mr}$	0.0024	0.0024	0.0096

The data here indicate that the variability about the mean is consistent in the two representations which use simulated individuals to estimate contact and uptake. This is what we would expect since the mechanisms of uptake and contact are the same. In contrast, the population's values suggest that the contact and uptake are quite different, and that this model does not perform in quite the same way.

2.6.3 Sensitivity to the shape of the plume

We will use the same notation as Section 2.6.2 for the data derived from the circular plumes, while we will add a prime symbol to the data derived from the elliptical plumes. Thus, the mean value time series for the mutating submodel with elliptical plumes would be denoted C'_m and the mean value of that time series is $\bar{C}'$.

There is a good correspondence between the means and deviations associated with the mutating model in the circular and elliptical plume scenarios, but there is much poorer correspondence in the population based results in the two scenarios. The data for the circular plume and for the elliptical plume are presented in Tables 2.4 and 2.5 respectively.

The population based model is clearly more sensitive to the shape of the plume than the mutating model. It seems likely that the major driver of this difference

Table 2.4: Circular plume results

$Series_r$	$\hat{C}_r$	$\bar{C}_r$	StdDev	$r = m$	$r = p$
C_m	0.1738	0.0392	$\widehat{\sigma}_{mr}$	0.0088	0.0535
C_p	0.1387	0.0350	$\bar{\sigma}_{mr}$	0.0024	0.0098

Table 2.5: Elliptical plume results

$Series_r$	$\widehat{C'}_r$	$\overline{C'}_r$	StdDev	$r = m$	$r = p$
C'_m	0.1856	0.0394	$\widehat{\sigma}'_{mr}$	0.0087	0.0616
C'_p	0.1763	0.0445	$\bar{\sigma}'_{mr}$	0.0025	0.0092

is that the long axis of the plume (a region where the net contact will be higher) remains in close proximity to the centroid of population where the population density is greatest.

2.6.4 Run-time

Each run collected data regarding the amount of time spent in different parts of the submodel; predictably, most of the effort is in calculating contact and updating contaminant loads.

The optimisation of suppressing the contact calculations when a population is outside the area of potential contact seemed to make very little difference to the run-time of population submodel (about 3%). It seems unlikely to make a great deal of difference to the mutating submodel. In the case of the purely individual-based submodel, this sort of optimisation is likely to play a much bigger role; any penalty would be multiplied by the number of animals simulated.

The population submodel ran for 98.7 cpu seconds. This submodel is deterministic and the amount of cpu time used is very stable, so only a single run is considered for comparison. The purely individual-based submodels took just over a mean time of 4205 cpu seconds with a standard deviation of approximately 16 seconds and the mean of the mutating submodel's run time was 1157 cpu seconds with a standard deviation of slightly over 11 cpu seconds.

2.7 Discussion

In the example our objective is to produce time-series data associated with the contaminant load of the group. Our individual-based model is taken as the best model for capturing the contact that real organisms have with an intermittent plume, and the population based representation has a computational efficiency that the individuals lack. The case for swapping in the example model is reasonably clear: there is a distinct improvement in run-time with no apparent deterioration in the fidelity of the dynamics. It seems likely that the naïve population distribution may be introducing a systematic divergence from what we see as an accurate, but computationally intense, individual-based model.

The general case is not limited to the polar extremes of switching between individual-based models and populations. A niche may have many representations, each of which has a particular set of strengths and weaknesses. This adaptive approach would present the same scope for improvement in purely equation-based models, where rules of thumb might be replaced by first-order approximations, or by complex systems of differential equations. In a purely individual-based example, the depth of the representation of the individual might vary from a simple mass and location through to a level of detail which included the individual's metabolic rates and breeding characteristics.

2.7.1 State spaces

To make our population-based model compatible with the individual-based model we have to extend the population's state space and maintain additional information to preserve the essential parts of the individual representation that makes it valuable to us. This is basically posing the first of the enumerated question from section 2.2. The *significant* information which the individual-based representation possesses is embodied in the contaminant loads amongst the individuals which comprise the group. We assume that the role of their relative locations about the population's centre is not important over a large portion of the global state-space and that we can discard it when we move from individuals to populations.

The union of submodels' state-spaces can generally be decomposed into *processing sets* of state-variables. Partitioning the state variables in this way – particularly in advance – makes it easier to analyse and minimise the boundary effects associated with the transition from one representation to another. Within a representation, the state variables which are unique to it form a special subset. The subset can be divided into the variables which need to be maintained by other representations, which we call V_r, and the variables which do not which we will call U_r. In principle, the variables in V_r might be maintained by a routine which is common to them all. The variables in U_r are more complex: when some other representation is mutating to representation r, the values assigned to the variables in U_r should reflect the state implied by the state of the old representation.

In the example model, an individual's relative location is a member of this set. Variables which are maintained and used by more than one representation are the third major group. This group can be divided into the set which is used consistently across the submodels (W_r), and the group of variables which have different dynamics in the various representations (X_r). In our example case, the contaminant load level of an individual would belong to the set V_{ind}. Its location and velocity would be in U_{ind}, the current time for both individuals and populations belongs to W_r, and a list of contaminant loads for populations belongs to V_{pop}.

2.7.2 Heuristics

The example model has very simple dynamics: the plumes are always in the same place, the migration is very predictable, and the spatial domain an individual may explore is well contained. Implementing a heuristic for the model which efficiently decides when to move from one representation to another is very straightforward: *If we are close enough that an individual might encounter the plume if the plume were at its maximum, switch a population to a group of individuals. Conversely, if there is no chance that an individual heading straight toward the plume (backwards) will encounter it, move the individual to a "close enough" population, or create a new population to accommodate the individual.* The model was constructed so that any number of populations could be run, and the heuristic was framed so that there were no assumptions about the number of population agents and the size of the groups they represented.

In this model, the decision process was shared between the representations themselves and the controlling scheduler. The representations reported their "robustness" to the scheduler which would then decide how to act on the advice, either triggering a change in representation or not.

The goal is to have a complex ensemble of niches which will change representations under the aegis of the scheduler to optimise the global outcome. For this more general approach additional information is needed: the scheduler would incorporate information about what properties each of the current representations required from other niches in order to decide what the mix of submodels filling the niches ought to be.

2.7.3 Transitions

The transition from individuals to population and population to individuals involves the loss and reconstruction of fine-scale position data, which may be a source of error. In our example, we assume that we can reconstruct a plausible position for each individual from the population's distribution function because we know the typical distribution of individuals and there is no behavioural change associated with contaminant load. A contaminant that made an organism sluggish would skew the distributions of both the population as a whole and the distribution of intoxicated organisms within the population.

In the example model, individuals were randomly located in this way with enough time to randomise their velocities and blur any artifacts resulting from the selection of their locations. This corresponds to the perception that the velocities and relative locations of the individuals are comparatively unimportant except as they related to the distribution of the population. When values of state variables are generated in the process of changing to another representation, they need to conform to the distributions of the representation they are leaving. If for some strange reason (like behavioural change) the distribution of individuals, for example, does *not* conform to the distribution associated with a coherent population, then additional steps need to be taken accommodate this when changing to a population-based representation.

In section 2.7.2, transitions between representations in the example model are mediated by the scheduler. Decisions to change representation must be based, in part, on whether the transition will increase or decrease the efficacy of the suite of representations as a whole. If the example model were more than a pedagogic tool, it would have been useful to assess the mean error introduced in the transition from population to individual and individual to population. Our simulation was aimed at producing contaminant load results, but in this context our aim would be to track the mean position, variance and extrema of the distribution of individuals relative to the population. To do this we would perform a comparison similar to that of section 2.6 but using positional data rather than contaminant load. The results would indicate if there might be significant transition effects associated with the change in representation. This sort of testing should ideally be performed at a number of scales (temporal or spatial, for example) since the knowledge of how long it takes for the transition boundary effects to settle (if they do) should feed into the high-level managing scheduler. In a sophisticated system, a representation might be spun up in advance so that the boundary effects have settled before it takes over from a less efficient representation.

2.7.4 Errors

Estimating error and confidence is extremely hard in complex models. Not only are the abstract processes deeply connected, but there may be hundreds of thousands of lines of code[3] which may introduce error of their own. Confronted with the code of a large-scale marine ecosystem model, a naïve modeller might turn around and contemplate joining a monastery rather than try and track the error propagation through the system. When we look at making an aggregate model out of niches, we can make the set of each of the representations which fill the niche simpler than some chimera which tries to take the best bits of each of the candidate representations. With well isolated transition mechanisms, we side-step nests of conditional code-paths and can contain the potential sources of error we must analyse. Since transitions can be recorded

[3]Ningaloo-InVitro is currently more than 233000 lines of C++, NWS-InVitro is slightly more than 118000 lines of C++ and Atlantis is more than 145000. The size of the code-base for the InVitro models was a significant goad toward developing an approach to model construction which would make error control, tracking and estimation more feasible.

(like other useful data), we can generate an indication of the likely level of confidence based on our understanding of the representations used in a simulation.

2.8 Conclusion

Interesting and unanticipated results have come from this experiment. The discrepancy between the data concerning elliptical and circular plumes suggests that, at least in contaminant work, we need to pay closer attention to the movement dynamics of individuals and the density functions of populations. More predictably, the run-times show that mutating configurations provide a reasonable means of increasing computational speed without sacrificing fidelity in appropriate situations. The simple example demonstrated that a model which changes the representation of the system according to its location in its state space could provide much better computational efficiency than a model with a constant representation with no loss of accuracy.

Increasing population and resource use often reduce our environment's resilience and there is a growing need to model larger, more complex parts of the system we live in. Techniques for this have been iteratively moving toward a more systematic approach: many models will optimise their run-time and accuracy by suppressing unnecessary calculation, models will split their time-steps according to what they are simulating, or perhaps even adaptively set an appropriate time-step or spatial scale.

This paper proposes that actually changing the representations to suit the different regions of the state space of the model could provide a better balance between computational efficiency and error.

In our experience of large scale marine ecosystem modelling, the size of the system considered is growing much faster than computational capacity. Even for small systems the possibility of adjusting the representation of submodels to optimise the accuracy of the model as a whole has great appeal. Mutating models may provide an effective means of concentrating the use of computational capacity where it is most needed.

The authors would like to thank the three reviewers whose advice and suggestions have made this paper much clearer and to the point. Their care and insight are deeply appreciated.

2.9 Epilogue to the paper

This paper shows that two of the basic problems discussed in Chapter 1, namely situations where the mixing assumption fails, and improving the run-time efficiency of a model without sacrificing fidelity, can be resolved by models which change their representations according to the state of the agents. The initial motivation for considering this kind of model was the recurring need to incorporate contaminant modelling in marine environments Lyne et al. [1994a], Gray et al. [2006, 2014].

The decision strategy in this model was unusually straightforward and simple; this was primarily to avoid obscuring the the central idea, since a more realistic treatment might both obscure the basic simplicity of the idea, and need a significantly more capable modelling system for the discourse. At the time, the mathematical and programmatic resources I could bring to bear on the matter were primitive.

A soon after this paper was submitted I became involved in what seemed to be unrelated work which sought to incorporate social survey data into systems-management models. This work lead to the development of a tree structure which is used in Chapter 3 and fully defined, later, in Chapter 4. The configuration of trees can closely follow the structural characteristics of the surveys and provide a robust mathematical means of allowing simulated individuals with different attitudes, organisations or events to exert influence (positively or negatively) the attitudes of others in the system. The mathematical work presented later arose from that project, and during its development it became evident that this could provide a means of encoding complex relationships within a modelling system.

CHAPTER 3

Adaptive submodel selection in hybrid models

3.1 Prologue to the paper

This chapter is a (verbatim) inclusion of a paper that I was invited to submit for a special topic issue of the refereed journal *Frontiers in Environmental Science*. The issue's intent was to focus attention on hybrid models which couple component-models (submodels) from across the range of modelling paradigms, and to encourage the development of this sort of model in the context of ecological research. The paper develops a thought model which demonstrates a high level mechanism for governing the mix of representations used in the model based on the state of the system and the states of its components.

Simple rules can be used to govern transitions from one representation to another (Chapter 2), but these local transitions may degrade the quality of the simulation as a whole. The biomass of grass a farm's paddocks is probably quite adequately represented by a single number if there is little or no grazing, but with more livestock we might need to represent each paddock's biomass individually, ... and possibly each paddock as a finely resolved field showing where the animals graze most intensely.

The appendix to the paper is an early version of the mathematical machinery developed in Chapter 4. Since publication, the basic mathematical structure has changed somewhat. Other, related trees have been explored in an attempt to simplify calculations involving elements of the vector space; initial attempts were based on the notion of treating the scalar term of the lable in a node as its weight, but each of these formulations resulted in a loss of one or another of the algebraic properties which are useful to us. The algebraic structure described in the next chapter shares many of the properties of the trees defined in the appendix to this chapter, but differs in ways which simplify operations and make the interpretation of their effect somewhat more intuitive.

Here, the propositions and assertions in this chapter are presented largely

without proof[1], though the corresponding propositions for the work in Chapter 4 are proved in the material in Chapter 4.

The *Supplementary Material* mentioned in the chapter is present as Appendix C.

[1]Proofs of these propositions and assertions are available, but they haven't been included since the structure has been superceded.

Adaptive submodel selection in hybrid models

Randall Gray[2]
University of Tasmania
Simon Wotherspoon
University of Tasmania

Abstract

Hybrid modeling seeks to address problems associated with the representation of complex systems using "single-paradigm" models: where traditional models may represent an entire system as a cellular automaton, for example, the set of submodels within a hybrid model may mix representations as diverse as individual-based models of organisms, Markov chain models, fluid dynamics models of regional ocean currents, and coupled population dynamics models. In this context, hybrid modelers try to choose the best representations for each component of a model in order to maximize the utility of the model as a whole.

Even with the flexibility afforded by the hybrid approach, the set of models constituting the whole system and the dynamics associated with interacting models may be most efficient only in parts of the global state space of the system. The immediate consequence of this possibility is that we should consider adaptive hybrid models whose submodels may change their representation based on their own state and the states of the other submodels within the system.

This paper uses a simple example model of an artificial ecosystem to explore a hybrid model which may change the form of its component submodels in response to their local conditions and internal state relative to some putative optimization choices. The example demonstrates the assessment and actions of a "monitor" agent which adjusts the mix of submodels as the model run progresses. A simple mathematical structure is also described and used as the basis for a submodel selection strategy, and alternative approaches are briefly discussed.

3.2 Introduction

The case has been made for developing systems with submodels that change their representation according to their state. Vincenot et al. [2011] identify ref-

[2]Published in *Frontiers in Environmental Science*, 20/8/2015
Corresponding author:`Randall.Gray@limnal.net` (Randall Gray),
`Simon.Wotherspoon@utas.edu.au` (Simon Wotherspoon)

erence cases describing the major ways system dynamics models (*SD*) and individual-based models (*IB*) can be coupled. Their final case, *SD-IB* model swapping, is exemplified in the models described by both Bobashev et al. [2007] and Gray and Wotherspoon [2012]. These papers argue that we can improve on conventional hybrid models, in terms of efficiency, fidelity, model clarity or execution speed by using an approach that allows the submodels themselves to change during a simulation. The last two papers implement simple models which demonstrate the approach, with correspondingly simple mechanisms to control transitions between different submodels.

Some authors argue that the explicit coupling of *SD* models and *IB* models may provide greater clarity and resolution in modeling [Vincenot et al., 2011, Fulton, 2010]: parts of a model that are most clearly the result of aggregate processes are likely to be better suited to modeling with a *SD* approach. In contrast, the parts of a system where individuals have a significant influence on their neighbors [Botkin et al., 1972b] are better suited to an *IB* approach. This argument is closely tied to the notion of model fidelity. Following Del-Sole and Shukla [2010], we take *fidelity* to be the degree to which a model's trajectory is compatible with real trajectories. If our immediate goal is to maximize the utility of the set of submodels within a model as it runs, this must include the fidelity of the system in the decision process.

Measuring or estimating execution speed and numerical error are comparatively straight-forward, but determining model fidelity is not. Models with a high degree of fidelity should produce results which are consistent with observed data from real instances of the system they model across both a wide range of starting conditions and under the influence of ad hoc perturbations, such as fires through a forested domain. Model fidelity is addressed by Del-Sole and Shukla [2010] in the context of seasonal forecasting models. They explore the relationship between fidelity and skill using an information-theoretic approach. They describe *skill* loosely as the ability to reproduce actual trajectories, and they describe *fidelity* as measuring the difference between the distribution of model results and the distribution of real world results. They highlight the attractiveness of mutual information and relative entropy as measures (or at least indices) of skill and fidelity, but they observe that in their domain, climate modeling, the necessary probability distributions are unknown.

The issues of fidelity and the attendant cost/benefit balance are central to the discussion in Bailey and Kemple [1992]. This paper assesses the costs and benefits of three different upgrades to an existing model designed to help determine the best mix of types of radios used in a military context; their objective is to prioritize implementation of the refinements of their model. The fundamental issues they address are substantially the same as issues that influence dynamic model selection.

The paper by Yip and Marlin [2004] compares three models used for real-time optimization of a boiler network: simple linear extrapolation from the system's current state, quadratic prediction with the coefficients based on historical data and updated at every step, and a detailed process model that corresponds closely with the physical elements of the modeled system. Their conclusion

correlates the fidelity of the model with its ability to control the real-time optimization of the system. They explicitly note that there are real costs associated with the increased fidelity. These costs include model development and the need for more expensive sensors. They note that increasing fidelity in the model enabled the system to adapt to changing fuel more efficiently, and that when there were frequent changes in fuel characteristics the simpler models performed poorly.

The projects described in Little et al. [2006] and Fulton et al. [2011a] both used hybrid models as a means of decreasing the run-time, and increasing the fidelity of the modeled contaminant uptake in simulated organisms. This was accomplished by mixing individual-based submodels and regional population-based systems models. Gray and Wotherspoon [2012] explicitly used changes in the representation of agents to improve the execution speed of a contamination tracking model, without losing the fidelity of the individual based uptake model. In this paper we will develop a more general strategy which may be appropriate for more complex systems.

3.3 Model organization

For clarity, we will take the term *niche* to refer to something in the model which could be modeled in several ways: a "porpoise" niche could be filled by many instances of an individual-based model, models of pods, or a regional *SD* model of the porpoises. This is essentially the same as the term *component* in Vincenot et al. [2011]. The motivation for departing from this convention arose from confusion resulting from inadvertently using *component* both in a technical and non-technical sense. The close analogy between the nature of a niche in an ecosystem and the nature of a component as discussed in Vincenot et al. [2011] suggested the choice of *niche*.

Each of the alternative ways of representing a niche can be viewed as a *submodel*, and the word *representation* will be used to reflect a particular choice of submodel within a niche. An explicit instance of a submodel (such as a specific pod or an *SD* model) will be referred to as an *agent*. The *configuration* of the model at any moment consists of the particular set of submodels which fill the niches that comprise the model as a whole. For an adaptive hybrid model, there may be a large number of possible configurations and the selection of a "best" configuration is a complex matter.

Each agent running in a model must necessarily have data which can serve to characterize it for these assessments. This data would typically be some subset of its state variables, but the data alone may not be enough to base an assessment on: there may also be extrinsic data which play a role in a particular submodel's or agent's activity and impinges on its suitability. Then, the characterization of an agent – its *state vector* – is an amalgam of its own state and the state of other niches it interacts with, and it can be regarded as a point in the state space which the submodel is defined over.

A corresponding set of data characterizes a niche in the model; here, it is typ-

ically some appropriate aggregation of agent-level state variables (a biomass-by-size distribution, for example), relative rankings of the suitability of agents and alternative submodels, and indications of what extrinsic support all of the various alternatives require. This niche-level state vector provides the data needed for optimizing the configuration globally, and for managing the configuration when niche-wide effects become significant, for example, for an incipient epidemic.

Thus, there are three distinct levels of organisation which may influence the considerations regarding the current configuration, and inform any decision about what may need to change, namely

1. agent-level data need to be examined to determine how well suited each agent is to its current state and the context provided by the agents it interacts with,
2. a niche-level assessment which compares the utility of each of its current agents within a niche with their alternative submodels, and
3. a model-wide assessment which determines whether there are cross-agent conflicts or unmet needs arising from a particular configuration.

The state vectors which form the domains of submodels and niches are loci in appropriate state spaces and can be encoded as an elements in appropriate vector spaces. The mathematical tools to manipulate these state vectors can then be applied to calculate the distances between two states, the similarity of loci which represent models or niches, or to identify trends or clusters.

3.3.1 Implications of changing configurations

At a basic level, hybrid models are designed to represent entities or processes in the real world in a way which brings more clarity, efficiency, or fidelity that may be possible with more traditional approaches. Adaptive hybrid models, implicitly acknowledge that the appropriate representation may change through time. An important consequence is that when a submodel in a niche changes, it may trigger changes in representation elsewhere in the model.

We might consider an example where an *SD* submodel which represents the prey for an *SD* based predator changes to *IB* submodels. It seems reasonable to expect the representation of the predator might follow suit. This may change the spatial resolution, the fineness of the "quantities" represented, and possibly the time steps associated with the predators and prey. Disparities in either of the first two are simple enough to deal with: modelers routinely use interpolation as a means of removing inappropriate edges, or generating subscale data, for example. Changes in an agent's time step can have a dramatic causal influence on the subsequent simulation.

Chivers [2009] discusses how individual-based models are sensitive to when state variables are updated. In his discussion, the issue arises as a result of when the probability of a predator-prey interaction is calculated relative to

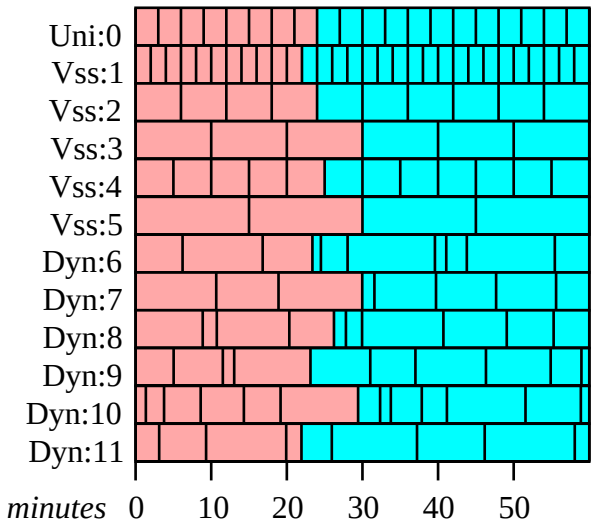

Figure 3.1: Time scheduling strategies. Red boxes represent time steps that have already passed, blue boxes represents scheduled time steps that have not yet been run. "Uni:" and "Vss:" submodels are members of a uniform or variable speed splitting submodels and require uniform time steps, and "Dyn:" submodels have adaptive time steps.

when the prey are removed from the system, though similar effects are also likely to occur in other contexts. The temporal sensitivity of submodels' interactions needs careful examination in order to construct submodels that proceed through time coherently and interact correctly.

Multi-agent models must have strategies to manage the agents as they step from the start of the simulation to its end. The simplest method is to make everything within the model use the shortest time step required. This is computationally inefficient in a heterogeneous model.

A better approach is the technique of variable speed splitting, such as in Walters and Martell [2004] and many others. (Figure 3.3.1) This approach allows models to step through time in different intervals by dividing the largest interval required into smaller steps that are more appropriate for the submodels with naturally shorter time scales. While models with uniform time steps are a trivial example of this approach, variable speed splitting is almost as simple and much more efficient. This technique can keep the subjective times of a set of agents moderately consistent, but ad hoc stepping changes would still seem to be awkward or difficult.

Both of these strategies may be subject to artifacts arising from the sequence in which agents are given their time step. The general class of model errors of the sort described in Chivers [2009] arise as a consequence of structure of the processing across the set of agents in a simulation. *IB* models which process agents species-by-species will be particularly vulnerable to these sorts of artifacts, since there will be an implicit advantage or disadvantage to being

early in the list. Similarly, advantage or disadvantage can arise when there is a change in representation, perhaps from an *SD* submodel to an *IB* submodel; a shorter time step in this situation may introduce a great many small time steps which agents may exploit. This kind of problem can be overcome by introducing a randomizing process within each time step. Early versions of the variable speed splitting model in Lyne et al. [1994a] suffered from predator-prey artifacts arising from a naïve introduction of predators and prey into the list of agents, and such randomizing was introduced to minimize the effects. In situations where the time steps of the interacting agents differ, implementing a randomization strategy may require a significant increase in the complexity of the system to accommodate irregular stepping through the lists of agents, or a significant change in the basic structure of the model.

Gray et al. [2006] and Fulton et al. [2009] describe models that have a well developed approach to coordinating agents using adaptive time steps. In these models agents may set their own time steps to intervals that are suitable for their current activity or role. This strategy can readily incorporate submodels with uniform time steps, or collections that employ a variable speed splitting strategy. When agents interact, they either explicitly become synchronous before interaction occurs by setting their time steps appropriately and waiting, or they implicitly acknowledge that there is a temporal mismatch. (Figure 3.3.1)

While some agents should be given execution priority (such as an agent which models ocean currents), most agents will have their execution order within a time step randomized, effectively preventing a large class of execution order dependent artifacts. The associated overhead in the most recent work, Gray et al. [2006], Gray and Wotherspoon [2012], is marginally higher than one would expect from single-stepping or variable speed stepping systems, but the advantages arising from the ability to ensure synchrony and change time steps in response to environmental stimulus outweigh the small computational overhead. This last approach seems likely to be the most appropriate for a general hybrid model that supports swapping models.

General adaptive hybrid models must have a mechanism for scheduling each agent's execution which keeps the cohort of agents roughly synchronous, and it should able to handle changes in an agent's time step when the agent changes its representation; where possible, agents should also be designed so that they may run at other time steps as well as their own preferred time step so they can become synchronous and interact at the appropriate temporal scale with other agents.

3.3.2 Systematically adjusting the model configuration

A model's configuration should only change when there is an overall benefit in the efficiency or fidelity of the system. A straightforward way of determining this is to have a monitoring routine that runs periodically, polling the agents, and ranking likely configurations according to their relative benefit or cost. This means that each submodel would need a way to provide, to the monitor, a measure of its current suitability, and to indicate what it needs from other

niches.

Algorithm 1 Basic processing pass for the monitor

```
for all niches do
    for all submodels in the niche do
        for all agents in the submodel do
            generate agent state vector
            generate the submodel state vector
                note extrinsic requirements
        end for
    end for

    generate niche state vector
end for

Run niche-level assessment
Flag any whole of model issues
for all candidate configurations do
    Deprecate untenable configuration
    Adjust for unavoidable extrinsic
        requirements
end for

Select best indicated configuration
```

The last step in Algorithm 1 is deliberately vague.

Algorithm 1 illustrates a possible assessment pass for a monitor, though how appropriate it may be is an open question. Configuration ranking for the example model will be cast in terms of evaluating an objective function based on elements of the vector space of tree elements described in the Appendix.

A monitor may have large number of potential candidate configurations, but we would like to keep the actual number quite low. The example model described below has a global domain associated with a particular representation, along with local domains (subregions of the global domain) which are associated with finer scale representations of the modeled entities. The set of potential candidate trees could be quite large; in practice we reduce the number by casting the candidate trees in a more general way – including trees representing particularly good representations and particularly poor representations: the first to steer the configuration toward good choices, and the second to drive it away from poor choices. We can use the hierarchical organisation (whole-model, niche, submodel, agent) to help limit our search space, as well as the geographic context of the agents (whole-domain, local cell, immediate-locus).

The sets of candidate trees which are associated with particular configurations will need to be crafted carefully as a part of the model design. These trees reflect the modelers understanding of the strengths and weaknesses of each of the submodels (or sets of different submodels) which may be employed.

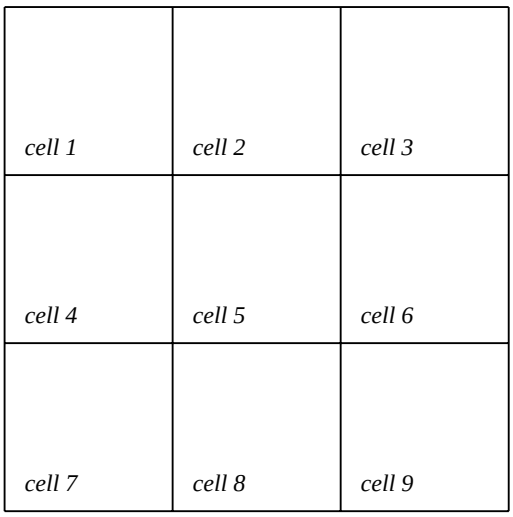

Figure 3.2: The model domain is divided into nine cells. An *SD* agent is associated with each of these cells and with the domain as a whole. Any *IB* agents which are created during the simulation will be associated with one cell at any given time.

Exactly how a monitoring routine is integrated into the model framework is a subjective choice best left to the team implementing the models, but one very attractive option is to implement the monitor as an agent in the system. This would allow the monitor to assess its own performance and the needs of other agents with respect to its own suitability with the option of swapping itself our for a montitor which implements some alternative strategy.

3.4 The example model

The purpose of the example model described below, is to provide a context for a discussion of the dynamics associated with a hypothetical simulation using this model. The ends of the spectrum between *SD* models and *IB* models are represented, and the environment is unrealistically simple in order to keep us from being swamped by detail.

The model consists of a spatially explicit environment that is partitioned into nine cells (Figure 3.4). The biotic elements consist of plants, fruit, seeds, herbivores, and carnivores. The herbivores feed on the plants and their fruit; and carnivores prey upon juvenile herbivores. The plants and herbivores are interdependent: fruit is the sole diet for juvenile herbivores and the plants need juvenile herbivores to make the seeds viable by eating the fruit.

The representations are equation-based *SD* models of the interactions between the plants and animals and *IB* models for plants and animals. The *SD* submodels model the biomass with respect to size, for plants and animals, or sim-

ply numeric quantities for fruit and seeds, and they can operate at either the global or cell-sized scale. Modeling biomass in this way makes it possible to minimize the loss of fidelity incurred by swapping from *IB* agents to *SD* agents and visa-versa, since we preserve more of the essential nature of the populations. A more detailed description of the *SD* agents is presented in the *Supplementary Material*.

Fruit and seeds

Fruit and seeds are treated somewhat differently to the rest of the niches. They exist principally as numbers of entities that are updated as a result of the activities of other, more explicit *SD* or *IB* models. There are explicit routines that deal with uniquely "fruit" and "seed" processing to handle spoilage and germination, respectively.

For fruit and seeds we have the following relationships

$$dN_{\mathrm{F}}(t) = Production - Spoilage - FruitEaten$$

and

$$dN_{\mathrm{S}}(t) = s * FruitEaten - \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right) Germ.$$

where $N_{\mathrm{P}}(t)$ is the biomass of plants at time t, and K_{P} is the carrying capacity of the pertinent domain (either global or cell-based). The processing for fruit

Algorithm 2 Basic processing pass for fruit

$N_{\mathrm{F}} \leftarrow N_{\mathrm{F}} - (Spoilage_{\mathrm{F}} \cdot N_{\mathrm{F}})$

is quite simple and consists only of applying "spoilage"; no reference to other agents in the system is required, and only the number of fruit is adjusted as a result (Algorithm 2). Seed models will adjust their "seed count" as well as

Algorithm 3 Basic processing pass for seeds

$NewTreeCount \leftarrow Germination \cdot SeedCount$
$SeedCount \leftarrow SeedCount - (NewTreeCount + Spoilage_{\mathrm{S}} \cdot SeedCount)$
generate $NewTreeCount$ new plant agents and introduce them into the system

the biomass distribution for plants in their time step, according to the level of germination. Germination is probabilistic as is the size of the plant a germinated seed becomes in its pass, though the distribution of possibly sizes is quite restrained (Algorithm 3).

SD representations

Each of the niches has an integral equation expressing the change in biomass for a given size; an animal's equation is of the form[3]

$$\begin{aligned} dN_{\mathrm{A}}(t,x) =& Growth\&Starv + Repr \\ & - PredMort - NatMort. \end{aligned}$$

We do not include migration terms in the *SD* models, since that will be addressed by the *IB* forms. The assumption is that the *SD* representation is most appropriate when population levels are moderately high, and there is adequate food; under these conditions, we will assume that the net migration associated with a domain will be close to zero.

Plants are represented by similar equations, namely

$$\begin{aligned} dN_{\mathrm{P}}(t,x) =& \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right)[Growth + Germ] \\ & - PredMort - NatMort \end{aligned}$$

where $N_{\mathrm{P}}(t,x)$ is the biomass of plants of size x at time t.

The important state variables for the *SD* are, for each domain, the biomass-by-size distributions for plants, herbivores and carnivores, and the raw numbers of fruit and viable seeds.

Algorithm 4 Basic processing pass for the *SD* models

```
for all agents in this domain do
    Incorporate quantities that are
        controlled in other agents
    Run Runge-Kutta4
    Update only quantities that are
        controlled by this agent
end for
```

The system of equations described in the *Supplementary Material* is evaluated using a fourth order Runge-Kutta algorithm; the numbers of fruit and seeds, and both the global and cell-based biomass distributions for plants and animals are updated at the end of the calculation. The model will adjust the values in the global and cell-based models to allow data from models running with better resolution (usually more localized models) (Algorithm 4) to take precedence.

Most of the important parameters and many of the functions associated with the life history of the modeled entities are not specified. This way we may consider possible trajectories without being tied to a particular conception or parameterization of the system.

[3] See the *Supplementary Material* for a more detailed set of equations.

IB representations

Individual-based representations for plants, herbivores and carnivores follow the pattern in Little et al. [2006]; fruit and seeds are only modeled in the *SD* representation, though their numbers are modified by the activities of the herbivores irrespective of how those herbivores are represented.

3.4.1 *IB* Plants

Plants maintain a reference to their cell, their location, a mass and a peak mass. If a plant's mass drops below a certain proportion ($\mathtt{P}_{M\Omega}$) of its peak mass, it dies — this provides a means for the herbivores to drive the plant population to local extinction.

We will suppose that plants grow according to a sigmoidal function with some reasonable asymptote and intermediate sharpness; fruiting occurs probabilistically as in the *SD* representation.

Algorithm 5 Basic processing pass for plants

if $(Mass \geq \mathtt{P}_{Mature}) \wedge (\mathtt{P}_{Fruits} \geq \mathrm{rnd}_{0,1})$ **then**
  ADDFRUIT($\mathtt{P}_{\rho} Mass^{\frac{2}{3}}$)
end if
if $(Mass \leq \mathtt{P}_{\mathtt{M}\Omega}\, PkMass) \vee (\Omega_{\mathrm{ind}\mathtt{P}} < \mathrm{rnd}_{0,1})$ **then**
  DIE
else
  $Mass \leftarrow \Gamma_{\mathtt{P}}(\delta t, Mass)$
  if $Mass > PkMass$ **then**
    $PkMass \leftarrow Mass$
  end if
end if

The plant agent goes through the steps in Algorithm 5 in each of its time steps. In the algorithm, $\Gamma_{\mathtt{P}}(\delta t, mass)$ is an analogue of the probability of a plant growing from one size to another from the *SD* representation, $\mathtt{P}_{Mature}$ is the parameter that indicates the mass a plant must be before it fruits, $\mathtt{P}_{Fruits}$ is the probability of a mature plant fruiting, and $\mathtt{P}_{\rho}$ is the amount of fruit relative to the fruiting area. The routine ADDFRUIT updates the models representing fruit in the domain.

3.4.2 *IB* Animals

Like the plants, animals maintain a reference to their cell, their location, and a mass. They also maintain several variables that are associated with foraging or predation, namely the amount of time until they need to eat (*Sated*), and the amount of time they have been hungry (*Hungry*).

Animals will grow while they do not need to eat and will only forage when

they are hungry. Reproduction happens in a purely probabilistic way once the animal is large enough, and the young are not cared for by the parents.

Animal movement is constrained so that they will tend to stay within their nominated home cell, only migrating (changing their home cell to an adjacent cell) when food becomes scarce or if the population exceeds some nominated value and causes crowding.

The analogues of the mechanisms for growth and starvation in the *SD* representation are quite different to those of the *IB* version. In the *SD* models, starvation and growth occur as a result of the relative population levels of the consumer and the consumed rather than the local availability of food.

There are no real programmatic differences between the *IB* representations of herbivores and carnivores; their differences lie in their choices of food and the way their "time-to-eat" variable is initially managed. InDiViduAl-based, newborn carnivores begin with a long time till they need to eat. This reflects a reliance on some unmodeled foodstuff until they are large enough to prey on the juvenile herbivores. In contrast, the juvenile herbivores must begin eating fruit immediately, and only switch to foraging on plants when they are larger (but before they can reproduce). For both species, if the amount of time they have been hungry exceeds a particular value, H_Ω or C_Ω, the individual dies.

Algorithm 6 Basic processing pass for herbivores and carnivores

if $(\Omega_{\mathrm{ind}\mathtt{A}} > \mathrm{rnd}_{0,1}) \vee (Hungry \geq \mathtt{A}_\Omega)$ **then**
  DIE
end if
$PreyList \leftarrow \textsc{PreyPresent}_{\mathtt{A}}(Locus, Mass)$
if $Sated \geq 0$ **then**
  $Mass \leftarrow Mass + \textsc{Growth}_{\mathtt{A}}(mass, \delta t)$
else if $(Hungry \geq 0) \wedge (\mathrm{len}(PreyList) > 0)$ **then**
  $Sated \leftarrow \textsc{Eat}(PreyList, \mathtt{A}_{EatLimit}, mass)$
  $Hungry \leftarrow 0$
  $ForageCt \leftarrow 0$
else if $((Hungry \geq 0) \wedge len(PreyList) = 0)$ **then**
  FORAGE
  $ForageCt \leftarrow ForageCt + 1$
else if $(Hungry \geq \mathtt{A}_{moveT}) \vee \textsc{Crowded}_{\mathtt{A}}$ **then**
  $\textsc{Migrate}_{\mathtt{A}}(Locus)$
else
  if $(mass \geq \mathtt{A}_{RepSize}) \wedge (\mathtt{A}_{RepP} \geq \mathrm{rnd}_{0,1})$ **then** $\textsc{Reproduce}_{\mathtt{A}}(Locus)$
  end if
end if

So, if we take `A` to represent either carnivores (`C`) or herbivores (`H`) below, then the processing pass for an animal is shown in Algorithm 6, where $\mathtt{A}_{moveT}$ is the amount of time an animal can be hungry before it migrates, $\mathtt{A}_\Omega$ is the amount of time it takes for the animal to starve, $\mathtt{A}_{EatLimit}$ is the most the animal can eat as a proportion of its mass, $\mathtt{A}_{RepSize}$ is the minimum size an animal may breed at

and $\mathtt{A}_{RepP}$ is the probability of reproducing. The routines $\textsc{PreyPresent}_{\mathtt{H}}$ and $\textsc{Eat}_{\mathtt{H}}$ have different cases for juvenile and adult herbivores, since juveniles prey upon fruit, and the seeds from the fruit they eat need to be accounted for in the appropriate places. There is a similar issue with juvenile carnivores. Their *preylist* will always be set to a value that indicates that they may eat as much as they like, and the corresponding call to $\textsc{Eat}_{\mathtt{C}}$ will handle this value appropriately.

3.4.3 The monitor and model dynamics

The following may be typical of the types of situations that could or should cause changes in the configuration:

- *Low population* – If, in an *SD* representation, the number of individuals filling a niche (either explicitly taken from a distribution, or estimated using a mean and a biomass) drops below a nominated value, then the biomass in that niche should be converted to *IB* agents representing those individuals. This type of change is motivated by the observation that at low population levels the assumption that we can treat the population as having uniform access to resources (or be uniformly available to predators) breaks down;
- *High population* – If a niche in a cell is represented by *IB* agents and the number of individuals exceeds a (higher) nominated value, the biomass those agents represent should be subsumed by the distribution in the local *SD* submodel. The change in representation is attractive here for two reasons: an equation-based representation will be much faster, and *SD* submodels are arguably simpler to calibrate;
- *Starvation risk* – If the mean amount of time an animal in a cell spends *hungry* in a cell exceeds half of $\mathtt{A}_{\omega}$ (or some other nominated time), the prey biomass must convert to *IB* agents if it isn't already so (bearing in mind that this isn't pertinent for fruit). This mean is calculated by averaging the means of each animal in the cell. If this is triggered, it indicates that the biomass of the prey species is sparse enough that homogeneity assumption is unlikely to hold;
- *Relative biomass* – If the biomass available for predation is represented in a local *SD* agent and its density drops below some proportion of the minimum required to support the predators in the domain, the prey species should convert its biomass into *IB* agents and, if the predator is represented by a *SD* agent, it should also convert to an *IB* form. If the biomasses are such that the effective predation rate is unsustainable, the mixing assumption is unlikely to hold.

The pertinent data for conditions will be periodically reported to the monitor through a set of status trees. The trees are able to represent single entities, nested entities and aggregates equally well, and can preserve structural information which may also be used in the comparison of these trees. One of the

basic elements we can easily incorporate into a submodel's status tree is the agent's own assessment of its competence relative to its state-vector and its local conditions. This measure of "self-confidence" can probably be maintained at little computational cost for most agents, and may be the most significant component in a monitor's assessment. The *high* and *low* population level conditions can clearly be determined by the agent in question; it can set its level of self-confidence upward or downward as appropriate. *Starvation* can also be encoded in the relevant node of an agent's status tree, but since starvation alone may not indicate a problem with the way the entity is represented, it probably wouldn't reduce the value for its confidence.

A starvation trigger may usually arise as a natural consequence of the population dynamics, but it may also occur when there is a mismatch in representations which has not been adequately addressed in the design stage. The final condition based on the relative biomasses is one which properly lies in the realm of the monitor – it would be quite inefficient for each of the candidate animals to be querying their prey for available biomass, summing the result, and then noting the need for change.

The monitor will primarily use the confidence values associated with agents and their niches, and the distance from trees which describe the state of the model or its set of submodels to trees which describe "known good" configurations. With data obtained directly from the agents in the system and from alternative representations it generates status trees,

- $\bar{\tau}^{\Sigma}_{sn}$, is a candidate status tree tied to a specific configuration. The serial number, sn, ties it to a configuration with that serial number,
- $\check{\tau}^{\Sigma}_{d}$, is a candidate tree which represents the current state of a domain,

  τ^{Σ}_{t}, an aggregate tree for the whole domain at time t,

  $\tau^{\Sigma}_{SD(n),t}$, aggregate trees for each cell, $n \in \{1, \ldots, 9\}$,

  $\tau_{\mathsf{R}(i),t}$, specific status trees for each agent,

  $\tau_{R,t}$, specific status trees for a representation R for each representation associated with a niche,

  and

  $\hat{\tau}_{\mathsf{R}(i),t}$, candidate trees for all possible representations of each agent i,

at the beginning of each of its steps. The model may have a mix of *SD* and *IB* representations, and some of the trees will have to incorporate data from many agents (τ^{Σ}_{t}, any of the $\hat{\tau}_{\mathsf{R}(i),t}$, and $\tau_{R,t}$, for example). A candidate tree is a status tree which represents an alternative submodel in a niche, and candidate trees are generated for specific agents and for each niche. When the monitor begins to generate status or candidate trees for a given agent, it first looks to see if it has generated an appropriate tree already. If it finds one, it incorporates

or adjusts the tree appropriately; perhaps by incorporating the agent's biomass and size into the tree's data. We will also denote the configuration of a domain (global or local) with $\check{\boldsymbol{\tau}}^{\Sigma}_{c}$ where c identifies the domain in question.

The monitor assesses the trees by calculating aggregate values of particular attributes, comparing the trees' divergences from allegedly ideal configurations, and by looking how uniform groups are – groups of individuals that are all very similar are good candidates for simpler representations.

We can calculate the average confidence value from any of these trees by evaluating

$$\frac{(\!|\overline{\mathsf{mask}}(\,\boldsymbol{\tau}, \mathit{confidence}, 0)|\!)}{\mathrm{supp}(\overline{\mathsf{mask}}(\,\boldsymbol{\tau}, \mathit{confidence}, 0))},$$

for example. The trees and functions to manipulate them are described in the Appendix.

Now let us consider what a simulation might look like. Figure 3.3 provides an overview of the configuration of the system as our hypothetical simulation runs. The model begins with eleven agents (not counting the monitor). The monitor runs its first step generating the status trees: $\boldsymbol{\tau}^{\Sigma}_{0}$, which characterizes the model in aggregate, $\boldsymbol{\tau}^{\Sigma}_{SD(0),0}, \ldots, \boldsymbol{\tau}^{\Sigma}_{SD(9),0}$, which record the aggregate state of the ten *SD* submodels, the aggregate status tree for the *IB* agent, $\boldsymbol{\tau}^{\Sigma}_{IB(0),[}9]$, status trees for the *SD* submodels: $\boldsymbol{\tau}_{\mathsf{SD}(0),0} - \boldsymbol{\tau}_{\mathsf{SD}(10),0}$, the status tree for the lone carnivore, $\boldsymbol{\tau}_{IB(11),0}$, followed by the trees which represent alternative agents: $\hat{\boldsymbol{\tau}}_{SD(0),0} - \hat{\boldsymbol{\tau}}_{SD(10),0}$ and $\hat{\boldsymbol{\tau}}_{IB(11),0}$. As mentioned earlier, there is only the single tree for agent 11 (the carnivore) since its alternative representation is embodied in $\hat{\boldsymbol{\tau}}_{SD(10),0}$. During the simulation a simulated fire will occur.

The first steps which must be taken before ranking of potential configurations is to find the sets of candidate trees which best approximate the current configuration at both the global and cell levels. We do this by calculating a similarity index or a distance which indicates how close each of the candidate trees are to the configuration of each of the domains. There are many ways we could do this: for an index which only considers structural similarity we might use something like the simple function

$$\mathrm{ssim}(\,\boldsymbol{c}, \boldsymbol{\tau}_{\boldsymbol{d}}) = \frac{\mathrm{overlap}(\,\boldsymbol{c}, \boldsymbol{\tau}_{\boldsymbol{d}})}{\max(\|c\|_{\top}, \|\tau_{d}\|_{\top})},$$

but for a more comprehensive treatment which factors values which are incorporated into the candidate and status trees we might apply the $\Delta(,)$ or d functions described in the Appendix. The d function is a well-defined distance over the vector space of trees, while the Δ(,f)unction is an index of similarity that incorporates structural characteristics as well as the numerical distance between compatible subtrees. To refine such an analysis we could apply mask and $\overline{\mathsf{mask}}$ to select only the relevant parts of the candidate and status trees.

So to assess the configuration of a domain, we would use our chosen measure to construct a set of the results of applying an optimisation function, *opt*, to each of the candidate trees and their similarity to the current configuration. So

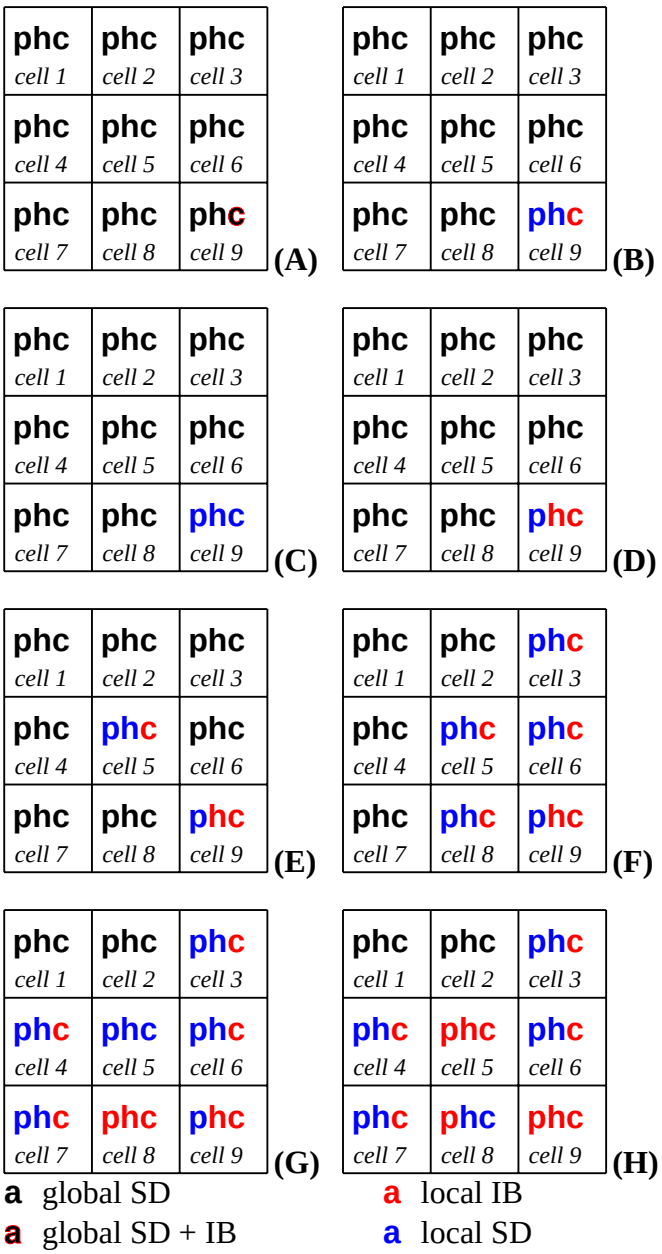

Figure 3.3: The color of the **p**,**h** and **c** indicate an agent's current representation within a cell at various points in the description of a simulation. In each, a black symbol indicates that the biomass of plants (**p**), herbivores (**h**) or carnivores (**c**) is modeled with the global *SD* agent, a blue symbol indicates that the biomass is modeled with a cell's *SD* agent, and red indicates that an *IB* model is being used. Symbols composed of two colors indicate that more than one representation is currently controlling portions of the relevant biomass.

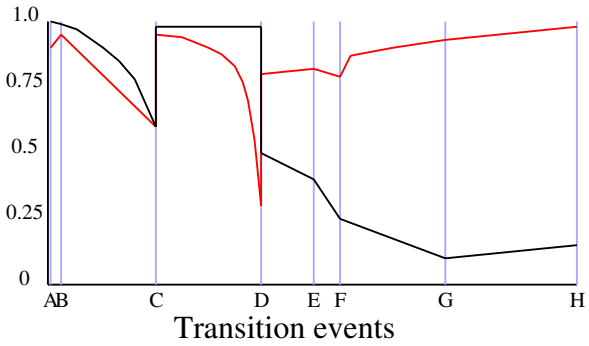

Figure 3.4: Normalized indexes of execution speed (black) and fidelity (red) the against configuration changes through time associated with Figure 3.3

if $\boldsymbol{S}$ is the set of all serial numbers for candidates, $\check{\boldsymbol{\tau}}_d^{\Sigma}$ is the status tree fo the current domain, and is the, we calculate

$$(C) = \{(\delta(\check{\boldsymbol{\tau}}_d^{\Sigma}, \bar{\check{\boldsymbol{\tau}}}_i^{\Sigma}), i) : \forall i \in \boldsymbol{S}\},$$

and this is used to generate

$$\boldsymbol{C}^* = \{(\text{opt}(\bar{\check{\boldsymbol{\tau}}}_i^{\Sigma}), c, \bar{\check{\boldsymbol{\tau}}}_i^{\Sigma}, i) : \forall (c, i) \in \boldsymbol{C}\}$$

where δ stands for our chosen measure of similarity.

The elements in $\boldsymbol{C}^*$ are then assessed by the monitor, and the best permissible candidate is selected. If there is only a small improvement on the current configuration, $\check{\boldsymbol{\tau}}_d^{\Sigma}$, the monitor will leave the configuration as it is; otherwise, the monitor would then manage the creation of new agents to replace less optimal representations and manage the exchange of state data.

So the early phase of our simulation might begin like so:

1. Both of the aggregate trees $\boldsymbol{\tau}_0^{\Sigma}$ and $\boldsymbol{\tau}_{SD(9),0}^{\Sigma}$ indicate that there is an *IB* agent in their domain and that their *SD* representation does not perform well for the indicated biomass. Both the status and candidate trees for agent 11, $\boldsymbol{\tau}_{11(0),}$, $\boldsymbol{\tau}_{IB(11),0}$ and $\hat{\boldsymbol{\tau}}_{IB(11),0}$, indicate that it is confident that it can represent the biomass, and that there are no immediate unmet requirements from other agents. Figure 3.3 (**A**)

2. The monitor assesses the trees against a prepared set of configurations: each of the alternative configurations (including the current configuration) is compared to a set of prepared, "efficient" configurations. The configuration of cell 9, $\check{\boldsymbol{\tau}}_9^{\Sigma}$, notes global *SD* representations for plants and herbivores. This configuration is ranked lower than the alternative which has an individual based model for carnivores and a local *SD* submodel for the other entities in the cell. The monitor makes this change in configuration, and informs the global *SD* agent that it is no longer controlling the biomasses in cell 9. Figure 3.3 (**B**)

3. The model may run for some time without any change in configuration. Both the herbivores and carnivores breed. The increased execution speed between **C** and **D** in Figure 3.4 is a result of a change in representation: the number of carnivores, recorded in $\boldsymbol{\tau}^{\Sigma}_{SD(C),t_4}$, reaches a point that prompts the monitor to convert them to an *SD* form. Figure 3.3 (**C**)

4. The biomass of carnivores has increased significantly by the time the model reaches **D** in Figure 3.4, and they are now eating all the young herbivores; as a result the carnivore population is now prey-limited, and the *Relative biomass* condition is triggered. Both the carnivore and herbivore populations are converted to *IB* representations. Notice that dynamics in the fidelity in Figure 3.4 around **D** arise from the collapse of the carnivore's prey, followed by the increase in fidelity after the representation change at **D**. Figure 3.3 (**D**)

5. A carnivore, agent 43, has been hungry ($Hungry \geq \mathtt{A}_{moveT}$) and has migrated to the cell 5 (noted in $\boldsymbol{\tau}_{IB(43),t_5}$). As occurred in cell 9 at step 2, the monitor converts plants and herbivores in cell 5 to a local *SD* representation, with *IB* carnivores. Figure 3.3 (**E**)

6. A lot of activity has occurred in this monitor interval: a *Starvation risk* is triggered in cell 9 because too many of the carnivores are hungry (many of the $\boldsymbol{\tau}_{IB(n),t_7}$ trees indicate that the elapsed time without eating is greater than *Hungry*). There has been more migration to cells 5,6 and 8 from cell 9 (more of the $\boldsymbol{\tau}_{IB(n),t_7}$ trees indicate residence in new cells), and a chance migration has introduced a carnivore into cell 3 from cell 5. Cells 3,6 and 8 are converted to local *SD* and *IB* representations as happened in step 3. Figure 3.3 (**F**)

7. The population of carnivores in cell 9 crashes as a result of migration and the scarcity of prey, (reported in $\check{\boldsymbol{\tau}}^{\Sigma}_{9}$) The *IB* juvenile herbivores are patchy and harder to find, so only a few carnivores are getting enough to eat. There will be many $\boldsymbol{\tau}_{IB(n),t_{10}}$ which indicate hunger or death due to starvation. The monitor cleans up the dead agents. There are chance migrations from cell 5 into cells 4 and 7 (in $\boldsymbol{\tau}^{\Sigma}_{SD(4),0}$ and $\boldsymbol{\tau}^{\Sigma}_{SD(7),0}$). A fire begins in cell 8, moving through cell 5: biomass loss in all niches causes all niches to shift to *IB* representations. Figure 3.3 (**G**)

8. Juvenile herbivores are reappearing in cell 9, but the available plant biomass (recorded in $\boldsymbol{\tau}_{\mathsf{SD}(9),t_{11}}$) has dropped due to reduced germination rates, triggering the *Relative biomass* condition in cell 9 causing the plants to convert to an *IB* representation. The fire in cell 8 has killed all animal biomass in the cell; they *do not* return to the global *SD* representation because their status trees diverge by too much. Instead, they convert to local *SD* representations (which represent zero biomass quite efficiently). Plants remain as *IB* agents The fire spreads to cell 5. Figure 3.4 shows a modest increase in execution speed between **G** and **H** due to the population losses associate with the fire. Figure 3.3 (**H**)

- ... the simulation continues

3.5 Discussion

Adaptive hybrid models *can* be constructed so that each submodel is aware of its other representations and is able to change form as appropriate [Gray and Wotherspoon, 2012]. This approach requires each model to have a reasonably close coupling with its alternative representations, and the burden of instrumenting (and maintaining) the necessary code quickly becomes untenable in complex models. Worse, it removes the possibility of more subtle configuration management that can accept poor performance in one part of a system in exchange for much better performance elsewhere. It seems that a guiding principle should be that in an adaptive hybrid model, each representation should know only as much about the rest of the model as it *must* know, and no more. The facility for a submodel to delve into the workings of other submodels, or the workings of the model as a whole, decreases the clarity that hybrid modeling makes possible, and opens avenues for unwanted, unanticipated behavior.

The major argument in favor of closely integrated representations for submodels is that it makes common (or at least similar) state variables easy to maintain across representations, even in the face of many representation changes. It is an attractive arguement, but the long term consequence is an ever growing burden of code maintenance.

Constructing hybrid models isn't significantly more complex than constructing traditional models. Adaptive hybrid models of the sort described in this paper will require a more significant investment in the design of a monitoring routine, and in the crafting of appropriate sets of candidate configurations. The transition dynamics such a model will exhibit depend on the sets of candidate configurations, and it seems likely that a combination of analysis and experimentation may be the most effective way to develop a set of useful configurations. The hybrid models associated with Lyne et al. [1994a], Little et al. [2006], Fulton et al. [2009] were built by extending the repertoire of ways of representing elements of the ecosystem or the anthropic components rather than wholescale redesign and replacement.

We can imagine an ideal adaptive hybrid model, where any state information which must be passed on is accompanied by an appropriate, opaque parcel of code to perform the maintenance. As long as the monitor knows what information each of these maintenance interfaces needs, they can be updated each time the monitor interrogates the agent which has control of the state data. This is a readily attainable ideal: many programming languages support first class functions with closures, and these features are precisely what we need to address this problem. *Scheme, Python, ML, Common Lisp, Lua, Haskell*, and *Scala* all have first order functions with closures and, hence, the capacity to build model systems with this capability.

The state vectors and their supporting maintenance procedures can be treated as data and passed in lists associated with the status trees. If a monitor decides to swap representations, the accumulated lists of maintenance functions may be passed on to the new representation. A new representation inherits a maintenance list with variables that are part of its native state, it can claim them as

its own and continue almost as though it had been running the whole time. In this way, a new representation doesn't need to know anything about its near kin, only that it must be able to run these black-box functions that come from other submodels, and to pass them on when required.

It may seem that this concentrates the global domain knowledge in the monitor, but this is not really the case. The monitor knows how to blindly query agents for state data and to the data in maintenance procedures. The monitor also knows how to recognize and rank characterizations of the states of the submodels or niches and to use those data to select a configuration.

The domain knowledge is encapsulated in the sets of targets the monitor matches the current configuration against, and in the heuristic triggers (such as *Starvation risk*) associated with a submodel or niche.

The essential problems any monitor is likely to deal with are problems of set selection (recognition, pattern matching…) and optimisation. These are common tasks: web searches, voice recognition, and route planning have become ingrained parts of modern society. Like route planning, the monitor needs to be able to reassess the "optimal" strategy as an ongoing process.

There are many options to choose from to rank configurations. A few of the likely candidates include

- using an objective function to evaluate each of the possible configurations,
- selecting a configuration based on decision trees,
- using neural nets to match model states and direct us to an appropriate configuration,
- using Bayesian networks to determine the most likely candidate,

and

- using support vector machines to select the target/configuration pairs.

In writing this paper, one of the vexing difficulties has been finding a suitable mathematical representation which would allow comparisons between configurations, submodel states and the states of niches. We need proxies that describe models and configurations of models in a way that we may readily understand, manipulate and reason about, and being able to deal with submodels which are, in themselves, adaptive hybrid models, seems to be a naturally desirable trait. The vector space of trees described in the Appendix has some nice properties, and may be directly useful with many of the options above: it forms a commutative ring (without necessarily having a unit), and would naturally inherit the body of techniques which only require the properties of such a ring.

3.6 Conclusion

There are still some major obstacles to developing a fully fledged adaptive hybrid model which is generic enough to tackle instances as varied as marine ecosystem modeling and urban planning. Foremost is a relative lack of real examples. The simulation of the hypothetical model[4] has tried to expose the character of an adaptive hybrid model which uses a monitor to manage the configuration of the system. There are parts of the description of the example system which are conspicuous by their absence; this is largely because they lie in almost wholly uncharted water. As a modeling community, we need to develop a wide range of approaches to how a model may assess the relative merits of a set of configurations. Many of the mechanisms we need for adaptive hybrid models already exist, but are found in domain specific models, and in wholly different domains, such as search engines and GPS navigation.

Establishing a suitable mathematical representation for model configurations which gives us access to well developed techniques for set selection, pattern recognition and component analysis would seem to be almost as urgent as adaptive hybrid examples of real systems.

Acknowledgments
The authors would like to thank two anonymous reviewers whose comments have improved the paper immeasurably. Thanks also go to a patient and understanding editor at Frontiers, and to Dr Tony Smith, who gave up a weekend to work a scientifically and grammatically fine toothed comb through the paper. The responsibility for any mistakes, awkward sentences, or places where it just does not make sense now rests completely with the lead author.

[4]The model described in this paper is currently under development and will be made freely available when it has been completed.

3.7 Appendix

3.7.1 Mathematical definitions

The trees we use are members of a normed vector space: we can add them, find out how far apart they are and interpolate between them. In principle, we can run clustering algorithms to find configurations that are similar, and identify when a model has left one cluster and entered another.

Definition 3.7.1. *Let $\mathcal{S}$ be a set of labels, and let $\mathbb{K}$be a field like the real or complex numbers. Then we define a node $\boldsymbol{n}$ as a triplet of the form $(s, v, \boldsymbol{E})$, with $v \in \mathbb{K}$, $s \subset \mathcal{S}$, and the set $\boldsymbol{E}$ is a (possibly empty) set of nodes of the same form with the restriction that no two nodes in $\boldsymbol{E}$ may have the same label in their first ordinate. We also define the triple $\mathbf{O} = (\varnothing, 0, \varnothing)$, which we will call the null tree, and define $\boldsymbol{T}$ to be the union of the set of all trees composed of a finite number of these nodes.*

The ordinates of $\boldsymbol{u} = (\boldsymbol{u}_\mathrm{P}, \boldsymbol{u}_v, \boldsymbol{u}_E)$ in $\boldsymbol{T}$ correspond to its label, value, *and* extension set. *An element of $\boldsymbol{u}_E$ will be called an* extension.

In our discussion, it will help to have a few more descriptive terms.

A node with an empty extension set is called a *simple* node, if this node happens to be a root node, then it is a simple tree.

Two trees, $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{T}$ are called *compatible* if either $\boldsymbol{u}_\mathrm{P} = \boldsymbol{v}_\mathrm{P}$ or at least one of $\boldsymbol{u}$ and $\boldsymbol{v}$ is $\mathbf{O}$.

We define the *depth* of a tree with:

$$\mathrm{depth}(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u} = (\varnothing, 0, \varnothing) = \mathbf{O} \\ 1 & \text{if } \boldsymbol{u} \text{ is a simple node} \\ 1 + \max(\{\mathrm{depth}(\boldsymbol{v}) : \forall\, \boldsymbol{v} \in \boldsymbol{u}_E\}) & \text{otherwise.} \end{cases}$$

We will also define for $\boldsymbol{u} \in \boldsymbol{T}$,

$$\mathrm{trim}(\boldsymbol{u}) = \begin{cases} \mathbf{O} & \text{if } \boldsymbol{u} = \mathbf{O} \\ \mathbf{O} & \text{if } \boldsymbol{u} \text{ is simple} \\ (\boldsymbol{u}_\mathrm{P}, \boldsymbol{u}_v, \{\mathrm{trim}(\boldsymbol{e}) : \\ \qquad \forall\, \boldsymbol{e} \in \boldsymbol{u}_E\} \setminus \{\mathbf{O}\}) & \text{otherwise.} \end{cases}$$

The cardinality of a tree is the number of nodes it contains. Specifically,

$$\|\boldsymbol{u}\|_\mathsf{T} = \begin{cases} 0 & \text{if } \boldsymbol{u} = \mathbf{O} \\ 1 + \sum_{\boldsymbol{e} \in \boldsymbol{u}_E} \|\boldsymbol{e}\|_\mathsf{T} & \text{otherwise.} \end{cases}$$

Simple nodes are the only nodes that have a cardinality of one, and $\mathbf{O}$ is the only node or tree with a cardinality of zero.

The support of a tree is the number of nodes which have a non-zero value.

$$\operatorname{supp}(\boldsymbol{u}) = \begin{cases} 0 + \sum_{e \in \boldsymbol{u}_E} \operatorname{supp}(\boldsymbol{e}) & \text{if } \boldsymbol{u}_v = 0 \\ 1 + \sum_{e \in \boldsymbol{u}_E} \operatorname{supp}(\boldsymbol{e}) & \text{otherwise .} \end{cases}$$

Related is the fundamental support of a tree, which only counts nodes with no zero valued nodes in their connection to the root node

$$\operatorname{fund}(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u}_v = 0 \\ 1 + \sum_{e \in \boldsymbol{u}_E} \operatorname{fund}(\boldsymbol{e}) & \text{otherwise .} \end{cases}$$

Clearly the support of a tree, supp $\boldsymbol{u}$, must lie in the domain $[0, \|\boldsymbol{u}\|_{\mathsf{T}}]$ and fund $\boldsymbol{u} \leq \operatorname{supp}(\boldsymbol{u})$.

The *overlap* between two trees is defined

$$\operatorname{overlap}(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} 0 & \text{if } \boldsymbol{u} = \mathbf{O} \text{ or } \boldsymbol{v} = \mathbf{O} \text{ or } \boldsymbol{u}_{\mathsf{P}} \neq \boldsymbol{v}_{\mathsf{P}} \\ 1 + \sum_{\substack{e \in \boldsymbol{u}_E \\ f \in \boldsymbol{v}_E}} \operatorname{overlap}(\boldsymbol{e}, \boldsymbol{f}) & \text{otherwise} \end{cases}$$

Clearly two trees, $\boldsymbol{u}$ and $\boldsymbol{v}$, are compatible if and only if $\operatorname{overlap}(\boldsymbol{u}, \boldsymbol{v}) \neq 0$; they will be said to *completely overlap* if $\|\boldsymbol{u}\|_{\mathsf{T}} = \|\boldsymbol{v}\|_{\mathsf{T}} = \operatorname{overlap}(\boldsymbol{u}, \boldsymbol{v})$.

We can now define scalar multiplication, and tree addition.

Definition 3.7.2. *Take $a \in \mathbb{K}$ and $\boldsymbol{u} \in \boldsymbol{T}$, then*

$$a\,\boldsymbol{u} = \begin{cases} \mathbf{O} & \text{if } \boldsymbol{u} = \mathbf{O} \\ \left(\boldsymbol{u}_{\mathsf{P}}, a\,\boldsymbol{u}_v, \{a\,\boldsymbol{f} : \boldsymbol{f} \in \boldsymbol{u}_E\} \setminus \{\mathbf{O}\}\right) & \text{otherwise.} \end{cases} \tag{3.1}$$

Definition 3.7.3. *Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be compatible elements of $\boldsymbol{T}$.Then taking the symbol + to be addition in the field $\mathbb{K}$, we extend it to addition in $\boldsymbol{T}$ so that for nodes $\boldsymbol{u}$ and $\boldsymbol{v}$,*

$$\boldsymbol{u} + \boldsymbol{v} = \begin{cases} \mathbf{O} & \text{if } \boldsymbol{u} = \boldsymbol{v} = \mathbf{O} \\ \boldsymbol{u} & \text{if } \boldsymbol{u} \neq \mathbf{O} \text{ and } \boldsymbol{v} = \mathbf{O} \\ \boldsymbol{v} & \text{if } \boldsymbol{u} = \mathbf{O} \text{ and } \boldsymbol{v} \neq \mathbf{O} \\ (\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v + \boldsymbol{v}_v, \varnothing) & \text{if } \boldsymbol{u}_E, \boldsymbol{v}_E = \varnothing \\ \Big(\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v + \boldsymbol{v}_v, \big(\{\boldsymbol{f} + \boldsymbol{g} : \boldsymbol{f} \in \boldsymbol{u}_E \text{ and } \boldsymbol{g} \in \boldsymbol{v}_E \text{ and } \boldsymbol{f}_{\mathsf{P}} = \boldsymbol{g}_{\mathsf{P}}\} & \\ \quad \cup \{\boldsymbol{f} : \boldsymbol{f} \in \boldsymbol{u}_E \text{ and } \boldsymbol{f}_{\mathsf{P}} \neq \boldsymbol{g}_{\mathsf{P}} \,\forall\, \boldsymbol{g} \in \boldsymbol{v}\} & \\ \quad \cup \{\boldsymbol{g} : \boldsymbol{g} \in \boldsymbol{v}_E \text{ and } \boldsymbol{g}_{\mathsf{P}} \neq \boldsymbol{f}_{\mathsf{P}} \,\forall\, \boldsymbol{f} \in \boldsymbol{u}\}\big) \setminus \{\mathbf{O}\}\Big) & \text{otherwise.} \end{cases} \tag{3.2}$$

Definition 3.7.4. *Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be compatible elements of $\boldsymbol{T}$.Then we define inner-multiplication between the two nodes*

$$\boldsymbol{u} \cdot \boldsymbol{v} = \left(\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v\,\boldsymbol{v}_v, \{\boldsymbol{f} \cdot \boldsymbol{g} : \boldsymbol{f} \in \boldsymbol{u}_E, \boldsymbol{g} \in \boldsymbol{v}_E \text{ and } \boldsymbol{f}_{\mathsf{P}} = \boldsymbol{g}_{\mathsf{P}}\} \setminus \{\mathbf{O}\}\right) \tag{3.3}$$

It can be shown that $\boldsymbol{T}$ with scalar multiplication and tree addition forms a vector space. This isn't quite enough to give us distances between trees, however, so we define a semi-norm

Definition 3.7.5. *Let $\boldsymbol{u}$ be an element of $\boldsymbol{T}$.Then we can define a semi-norm over $\boldsymbol{T}$*

$$\langle\!|\boldsymbol{u}|\!\rangle = \begin{cases} 0 & \text{if } \boldsymbol{u} = \mathbf{O} \\ |\boldsymbol{u}_v| & \text{if } \boldsymbol{u}_E = \varnothing \\ |\boldsymbol{u}_v| + \sum_{e \in \boldsymbol{u}_E} \langle\!|\boldsymbol{e}|\!\rangle & \text{otherwise.} \end{cases}$$

It is clear that $\langle\!|\boldsymbol{u}|\!\rangle$ will always be non-negative, and the only shortcoming is that we can have a node $\boldsymbol{u}$ with $\langle\!|\boldsymbol{u}|\!\rangle = 0$, but $\boldsymbol{u} \neq \mathbf{O}$. In order to turn this into a normed vector space we take the set $\mathfrak{O} = \{\boldsymbol{u} : u \in \boldsymbol{T} \text{ and } \langle\!|\boldsymbol{u}|\!\rangle = 0\}$ and we construct an equivalence relation on $\boldsymbol{T}$ by the rule $[\boldsymbol{u}] \equiv [\boldsymbol{v}]$ if and only if there exist $\boldsymbol{z_u}, \boldsymbol{z_v} \in \mathfrak{O}$ such that $\boldsymbol{u} + \boldsymbol{z_u} = \boldsymbol{v} + \boldsymbol{z_v}$. It can be shown that scalar multiplication, tree addition, and the semi-norm behave appropriately with respect to the equivalence classes. This means that if we identify elements of $\boldsymbol{T}$ with their equivalence class, then we can take $\langle\!|\boldsymbol{u}|\!\rangle$ to be a norm and that it induces a distance function

$$\mathrm{d}(\boldsymbol{u}, \boldsymbol{v}) = \langle\!|\boldsymbol{u} - \boldsymbol{v}|\!\rangle.$$

Definition 3.7.6. *We define the functions* mask *and* $\overline{\text{mask}}$ *that set the values associated with particular nodes in a tree to $v \in \mathbb{K}$. Specifically, if $\mathcal{L} \subset \boldsymbol{T}$, then*

$$\text{mask}(\boldsymbol{u}, \mathcal{L}, v) = \begin{cases} (\boldsymbol{u}_{\mathsf{P}}, v, \{\text{mask}(\boldsymbol{f}, \mathcal{L}, v) : \boldsymbol{f} \in \boldsymbol{u}_E\}) & \text{if } \boldsymbol{u}_{\mathsf{P}} \in \mathcal{L} \\ (\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v, \{\text{mask}(\boldsymbol{f}, \mathcal{L}, v) : \boldsymbol{f} \in \boldsymbol{u}_E\}) & \text{otherwise} \end{cases} \tag{3.4}$$

and

$$\overline{\text{mask}}(\boldsymbol{u}, \mathcal{L}, v) = \begin{cases} (\boldsymbol{u}_{\mathsf{P}}, v, \{\overline{\text{mask}}(\boldsymbol{f}, \mathcal{L}, v) : \boldsymbol{f} \in \boldsymbol{u}_E\}) & \text{if } \boldsymbol{u}_{\mathsf{P}} \notin \mathcal{L} \\ (\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v, \{\overline{\text{mask}}(\boldsymbol{f}, \mathcal{L}, v) : \boldsymbol{f} \in \boldsymbol{u}_E\}) & \text{otherwise.} \end{cases} \tag{3.5}$$

$\text{mask}(\boldsymbol{u}, \mathcal{L}, 0)$ *would return a tree similar to $\boldsymbol{u}$, but all its nodes that have labels in $\mathcal{L}$ would have values of zero.*

We also define several functions which prune or select a child from a tree's extension set. This function returns only the part of $\boldsymbol{u}$ which overlaps $\boldsymbol{p}$,

$$\text{excise}(\boldsymbol{u}, \boldsymbol{p}) = \begin{cases} (\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v, \{\text{excise}(\boldsymbol{f}, \boldsymbol{g}) : \boldsymbol{f} \in \boldsymbol{u}_E \text{ and } \boldsymbol{g} \in \boldsymbol{p} \text{ and } \boldsymbol{f}_{\mathsf{P}} = \boldsymbol{g}_{\mathsf{P}}\} \setminus \{\mathbf{O}\}) & \text{if } \boldsymbol{u}_{\mathsf{P}} = \mathcal{L} \\ \mathbf{O} & \text{if } \boldsymbol{u}_{\mathsf{P}} \neq \boldsymbol{p}_{\mathsf{P}} \\ (\boldsymbol{u}_{\mathsf{P}}, \boldsymbol{u}_v, \varnothing) & \text{otherwise,} \end{cases} \tag{3.6}$$

and this one either returns an appropriately labelled child from the extension set (a branch), or $\mathbf{O}$.

$$(_c\, \boldsymbol{u}, \ell) = \begin{cases} \boldsymbol{f} & \text{if } \boldsymbol{f}_{\mathsf{P}} = \ell \text{ and } \boldsymbol{f} \in \boldsymbol{u}_E \\ \mathbf{O} & \text{otherwise.} \end{cases} \tag{3.7}$$

We now finish with a definition of a function, $\Delta(,)$, that gives us a measure of the degree of divergence between two trees.

Definition 3.7.7. *The degree of deviation between two trees, $\boldsymbol{u}$ and $\boldsymbol{v}$ is given by the expression*

$$\Delta(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} \|u\|_\tau + \|v\|_\tau - 2\,\text{overlap}(\boldsymbol{u}, \boldsymbol{v}) + (\!|\boldsymbol{u} - \boldsymbol{v}|\!) & \textit{if } \boldsymbol{u} \textit{ and } \boldsymbol{v} \textit{ are compatible} \\ \|u\|_\tau + \|v\|_\tau & \textit{otherwise.} \end{cases} \tag{3.8}$$

The rationale behind this definition is that if trees $\boldsymbol{u}$ and $\boldsymbol{v}$ are identical, then $\Delta(\boldsymbol{u}, \boldsymbol{v})$ will be zero. We also want nodes that aren't common to both trees to count as differences.

3.8 Epilogue to the paper

The model described in this paper has been used as a template for the model used to frame the discussion in Chapter 5. The paper attempted to describe a credible "toy" model to provide a setting for such a discussion; when it came to implementation of the model the paper described, a few shortcomings came to light and some of the corresponding algorithms have been altered in the explicit model to correct them. The most notable example, and one which characterises the oversights well, is that adhering to the description effectively prevented cells which are stripped of living plants from being recolonised. As stated, the seeds resulting from the consumption of fruit were tied to the cell they were eaten in, because the animals excreted them immediately. The consequence of this was that there was no way for plants to recolonise empty cells.

This paper extended and generalised the simple state maintenance of Chapter 2 by passing closures rather than a vector of values and the confident knowledge that the closures embodied the correct process for update. By using closures, the actual machinery supporting the update of the state of a previous representation can be passed and both the mechanism and data made, in some sense, opaque to all but the representation it is associated with. Enforcing the notion that the code required for maintaining a state vector of a representation must be provided *by* that representation means that all of the code which changes the state of an agent is embodied within the agent's representation. This has practical appeal in that inappropriate modifications to that state data may cause errors that would be very difficult to isolate.

The trees described in the next chapter are used to characterise the distribution of agents within the model's spatial domain, and encode the different representations and numbers of those agents.

CHAPTER 4

A ring-like structure of trees

4.1 Introduction – a historical context

The project which triggered the development of the structure in this chapter considered using the modeling of the social dynamics associated with policies about, and responses to climate change and to predict the social and economic consequences of development arising from various scenarios. This was a significant change from previous representations of "public" participants, such as recreational fishers, tourists, accommodation and other small businesses, which were modelled in fairly simple ways (Fulton et al. [2011a], Gray et al. [2014]). The specific goal was to be able to incorporate a simulation of the way public opinion changes in response to policy actions and changes in the economy and environment. The starting point for this was a corpus of responses to a survey on attitudes associated with the topic of climate change (Boschetti et al. [2012]). The data consisted of (largely) numeric answers to individual questions which could represented as either distinct items, or as an aggregation of symbolic elements. Questions like "How much do trust the following individuals or organisations to tell you the truth about changing climate?" might be encoded in a symbolic way by aggregating the symbols climate_change, trust, information_source, AustPeng for the trustworthiness of the *Australian Penguins* as a source of information. The actual value marked could be encoded as a scalar, giving us an expression like "$4/5$ + climate_change + information_source + trust + AustPeng" – expressing it in this way suggested that working with more intricate relationships might be possible.

It became evident that the the mathematical structures could represent configurations of models, and that it might be able to incorporate the interdependencies between models and other information in a very simple way. It seemed possible to construct example configurations which were reasonable for particular conditions and to assess how close to "known-good" configurations the system was at any given point. It also seemed possible that it might allow strategies which were able to interpolate between "good" configurations making intermediate transitions between configurations feasible. The natural network of dependencies exhibited by some components of past models sug-

gested using tree structures to encode the configuration of a model as a starting point.

The structure described in this chapter is the structure behind the work in Chapter 5. The trees are comprised of nodes which have a label and a weight (and, of course, children). Initially, the labels were simply symbols associated with certain attitudes or semantic data which were taken from a nominated set. This had the advantages of simplicity and clarity, but their very simplicity made the multiplicative operator much more complicated. Much of the machinery associated with labels was really just a messy version of arithmetic in commutative ring-like structure of multinomials. Once labels were identified with multinomials, everything became clear. Using the field of rational multinomials may be productive, but this notion has not been seriously explored.

4.2 Conventions and preliminary definitions

The structure is, loosely speaking, a set comprised of disjoint subsets with a corresponding family of additive operators (one to each subset). Each subset is closed under its addition, and there is a multiplicative operator which is defined over the whole set.

Generally, we will use lower case, boldfaced symbols to denote a node (or tree), and upper case, boldfaced symbols to denote sets – particularly sets of nodes. Other symbols (such as x) will typically refer to numbers or multinomials. Elements of a node, $\boldsymbol{u}$ will be identified using an appropriate subscript, namely $\boldsymbol{u}_v$, for the node's value and $\boldsymbol{u}_\mathsf{P}$ for its label. Initially, the children of a node were thought of as refinements or children of an attitude, so the children of $\boldsymbol{u}$ were its children, and $\boldsymbol{u}_C$ denotes the set of children. We will take $\mathbb{K}[\boldsymbol{A}]$ to be the ring of multinomialsover the elements of a finite set of symbols $\boldsymbol{A}$. Here $\mathbb{K}$ would usually be some numeric field such as $\mathbb{Q}, \mathbb{R}$ or $\mathbb{C}$, for example.

Definition 4.2.1. *Given $\boldsymbol{A}$ and $\mathbb{K}[\boldsymbol{A}]$ and an arbitrary field, $\mathbb{K}$, we define the set, $\boldsymbol{T}^*$ of finite (acyclic) trees where each node is of the form $(\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C)$ where the value, $\boldsymbol{u}_v$, is a member of $\mathbb{K}$, it's label, $\boldsymbol{u}_\mathsf{P}$ is a member of $\mathbb{K}[\boldsymbol{A}]$ and the set of children, $\boldsymbol{u}_C$, contains leaf nodes with distinct labels.*

Nodes or trees with no children, $\boldsymbol{E} = \varnothing$, will be called *simple nodes, simple trees*, or *leaf nodes*, and simple nodes which also have scalar multinomials as their labels may be referred to as *scalar nodes* or *scalar trees*. The domain of trees, $\boldsymbol{T}^*$, is the collection of only those trees with a finite number of nodes. We will make use of a set of special elements in $\boldsymbol{T}^*$, $\mathbf{O} = (0, 0, \varnothing)$, which is an an analogue of zero that we will call the *zerotree*.

In this work, we will occasionally restrict ourselves to an arbitrary subset of $\boldsymbol{T}^*$, Once labels were identified with multinomials, everything became clear. We will take $\check{\boldsymbol{T}}_{p\mathsf{P}}$ to be the set of trees whose root nodes have a label equal to $\boldsymbol{p}_\mathsf{P}$. The element $\mathbf{O}_{p\mathsf{P}} = (0, \boldsymbol{p}_\mathsf{P}, \varnothing)$ will be shown to play the role of the additive identity in the set $\check{\boldsymbol{T}}_{p\mathsf{P}}$; we will use the symbol $\mathbf{O}$more generally to refer to

an appropriate member of the set of additive zero elements unless it creates ambiguity.

Note that the choice to use elements of the ring of multinomialsfor labels is, in a sense, arbitrary: elements of any commutative ring will serve, though we will see that if we use a commutative ring, $\boldsymbol{T}^*$ and the derived domains are also commutative; here, the ring of multinomialsprovide a simple example which is easily manipulated and printed. $\mathbb{K}$ is taken to be $\mathbb{R}$ or $\mathbb{Q}$.

Definition 4.2.2. *Two nodes or trees, $\boldsymbol{u}$, $\boldsymbol{v} \in \check{\boldsymbol{T}}_{p_{\mathsf{P}}}$ forsome $\boldsymbol{p}_{\mathsf{P}} \in \mathbb{K}[A]$ are said to be* compatible *if they have the same label.*

$$\boldsymbol{u} \sim \boldsymbol{v} \iff \boldsymbol{u} \in \boldsymbol{R}_{v_{\mathsf{P}}} \iff \boldsymbol{v} \in \boldsymbol{R}_{u_{\mathsf{P}}}.$$

Clearly, all the trees in $\check{\boldsymbol{T}}_{p_{\mathsf{P}}}$ are compatible, and we will denote the set of all nodes of the form $(0, \boldsymbol{p}_{\mathsf{P}}, \varnothing)$ with $\check{\mathbf{O}}$.

When there is no risk of ambiguity, we will use the same symbol to refer to a set, $\boldsymbol{T}^*$ for example, and a vector space based on that set. Compatibility (or lack thereof) is really only pertinent to the addition of trees, in the same way that having the same row- and column-rank is only necessary in matrix addition. There is no such constraint in the pairwise multiplication of trees.

First, the definitions for some basic tools for manipulating these trees. Some of the functions defined below are not used in this chapter, but play a role in the explicit model described in Chapter 5.

Definition 4.2.3. *The cardinality of a tree $\boldsymbol{u} \in \boldsymbol{T}^*$ is the number of nodes it contains. We define it formally as*

$$\|\boldsymbol{u}\|_{\mathsf{T}} = \begin{cases} 0 & \text{if } \boldsymbol{u}_v = 0 \text{ and } =_C \varnothing \\ 1 + \sum_{\boldsymbol{e} \in \boldsymbol{u}_C} \|\boldsymbol{e}\|_{\mathsf{T}}. \end{cases}$$

Simple nodes are the only nodes which have a cardinality of one, and $\mathbf{O}$is the only node or tree with a cardinality of zero.

Definition 4.2.4. *For any $\boldsymbol{u} \in \boldsymbol{T}^*$ we define the function for*

$$\text{depth}(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u}_v = 0 \text{ and } =_C \varnothing \\ 1 & \text{if } \boldsymbol{u} \text{ is a simple node} \\ 1 + \max(\{\text{depth}(\boldsymbol{v}) : \forall\, \boldsymbol{v} \in \boldsymbol{u}_C\}) & \text{otherwise} \end{cases}$$

which gives us the depth of the tree.

Definition 4.2.5. *We will also define for $\boldsymbol{u} \in \boldsymbol{T}^*$,*

$$\text{trim}(\boldsymbol{u}) = \begin{cases} \mathbf{O} & \boldsymbol{u}_v = 0 \text{ and } =_C \varnothing \\ \mathbf{O} & \text{if } \boldsymbol{u} \text{ is simple} \\ \left(\boldsymbol{u}_v, \boldsymbol{u}_{\mathsf{P}}, \{\text{trim}(\boldsymbol{e}) : \forall\, \boldsymbol{e} \in \boldsymbol{u}_C\} \setminus \{\check{\mathbf{O}}\}\right) & \text{otherwise.} \end{cases}$$

Obviously the depth is an indication of how many levels of nodes the tree possesses. Trimming essentially removes all simple nodes from the tree. A recursive application of trimming will be denoted trim_k, indicating that the tree $\boldsymbol{u}$ will be trimmed k times. Note that $\mathrm{trim}_{\mathrm{depth}(\boldsymbol{u})}\ \boldsymbol{u} = 0$ and $\mathrm{depth}(\mathrm{trim}_{\mathrm{depth}\ \boldsymbol{u}-1}\ \boldsymbol{u}) = 1$.

Definition 4.2.6. *The* overlap *between two trees is defined*

$$\mathrm{overlap}(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} 0 & \text{if } \boldsymbol{u} \in \check{\mathbf{O}} \text{ or } \boldsymbol{v} \in \check{\mathbf{O}} \text{ or } \boldsymbol{u}_{\mathrm{P}} \neq \boldsymbol{v}_{\mathrm{P}} \\ 1 + \sum_{\substack{\boldsymbol{e} \in \boldsymbol{u}_C \\ \boldsymbol{f} \in \boldsymbol{v}_C}} \mathrm{overlap}(\boldsymbol{e}, \boldsymbol{f}) & \text{otherwise} \end{cases}$$

Clearly two trees, $\boldsymbol{u}$ and $\boldsymbol{v}$, are compatible if and only if $\mathrm{overlap}(\boldsymbol{u}, \boldsymbol{v}) \neq 0$*; they will be said to* completely overlap *if* $\|\boldsymbol{u}\|_{\top} = \|\boldsymbol{v}\|_{\top} = \mathrm{overlap}(\boldsymbol{u}, \boldsymbol{v})$.

We may also make use of the relative overlap of two nodes, $\boldsymbol{u}$ and $\boldsymbol{v}$, given by

$$\mathrm{overlap}_r(\boldsymbol{u}, \boldsymbol{v}) = \frac{2\,\mathrm{overlap}(\boldsymbol{u}, \boldsymbol{v})}{\|\boldsymbol{u}\|_{\top} + \|\boldsymbol{v}\|_{\top}}.$$

The relative overlap of $\mathbf{O}$*with itself is not defined.*

Definition 4.2.7. *The* shadow *cast by a tree, $\boldsymbol{v}$, onto another tree, $\boldsymbol{u}$, is given by*

$$\mathrm{shadow}(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} \left(\boldsymbol{u}_v, \boldsymbol{u}_{\mathrm{P}}, \{\mathrm{shadow}(\boldsymbol{r}, \boldsymbol{s}) : \boldsymbol{r} \in \boldsymbol{u}_C, \boldsymbol{s} \in \boldsymbol{v}_C\} \setminus \{\check{\mathbf{O}}\}\right) & \text{if } \boldsymbol{u}_{\mathrm{P}} = \boldsymbol{v}_{\mathrm{P}} \\ \mathbf{O} & \text{otherwise} \end{cases}$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ are compatible.

This lets us restrict our attention to those parts of a tree which conform to some "template" tree. It is easy to see that $\mathrm{overlap}(\boldsymbol{u}, \boldsymbol{v})$ is precisely $\|(\|_{\top}\,\mathrm{shadow}(\boldsymbol{u}, \boldsymbol{v}))$.

Definition 4.2.8. *We will define a few useful notations for sets derived from sets of nodes in $\boldsymbol{T}^*$. Take $\boldsymbol{U}$ and $\boldsymbol{V}$ be such sets and $\boldsymbol{a}$ be a node in $\boldsymbol{T}^*$; then*

$$\boldsymbol{L}(\boldsymbol{U}) = \{\boldsymbol{e}_{\mathrm{P}} : \forall\, \boldsymbol{e} \in \boldsymbol{U}\}$$

$$\boldsymbol{U}|_{L(\boldsymbol{V})} = \{\boldsymbol{f} \in \boldsymbol{U} : \boldsymbol{f}_{\mathrm{P}} \in \boldsymbol{L}(\boldsymbol{V})\}$$
$$\boldsymbol{U}|_{\overline{L}(\boldsymbol{V})} = \{\boldsymbol{f} \in \boldsymbol{U} : \boldsymbol{f}_{\mathrm{P}} \notin \boldsymbol{L}(\boldsymbol{V})\}$$

and

$$a\boldsymbol{U} = \{a\,\boldsymbol{u} : \forall\, \boldsymbol{u} \in \boldsymbol{U}\}$$

Definition 4.2.9. *For convenience, we define analogues of several of the above relations for nodes to implicitly refer to the children of those nodes.*

Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be arbitrary nodes in $\boldsymbol{T}^*$. Then we define the following

$$\boldsymbol{L}(\boldsymbol{u}) \equiv \boldsymbol{L}(\boldsymbol{u}_C)$$
$$\boldsymbol{u}|_{L(\boldsymbol{v})} \equiv \boldsymbol{u}_C|_{L(\boldsymbol{v}_C)}$$
$$\boldsymbol{u}|_{\overline{L}(\boldsymbol{v})} \equiv \boldsymbol{u}_C|_{\overline{L}(\boldsymbol{v}_C)}$$

Some of these functions are not used in this chapter, but are presented because they may come into play in constructing software components used to assess and trigger changes in model configuration.

4.3 Scalar Multiplication and addition

The aim of this work is to be able to compare trees in a robust way and to manipulate them as though they were vectors: trees which form a vector space, or – better – a metric space, which can be compared and clustered. We will start by defining scalar multiplication of the trees in $\boldsymbol{T}^*$, and then we will define a few useful mappings which will help keep the expressions simple. Our aim, in this section, is to define addition, and to show that the defined scalar multiplication and addition make this a vector space. When there is no risk of ambiguity, we will use the same symbol to refer to both the set and a vector space based on that set.

4.3.1 Scalar multiplication and some convenience functions

Definition 4.3.1. *For all $a \in \mathbb{K}[\boldsymbol{A}]$ and $\boldsymbol{u} \in \boldsymbol{T}^*$, we define*

$$a\,\boldsymbol{u} = \begin{cases} \mathbf{O} & \text{if } a = 0 \text{ or } \boldsymbol{u} = \mathbf{O} \\ (a\,\boldsymbol{u}_v,\ \boldsymbol{u}_{\mathsf{P}},\ a\,\boldsymbol{u}_C) & \text{otherwise} \end{cases}$$

where $a\boldsymbol{A}$ indicates the element-wise product. Since the elements of $\mathbb{K}$ are a subset of $\mathbb{K}[\boldsymbol{A}]$ this also defines scalar multiplication of trees by elements of $\mathbb{K}$.

4.3.2 Addition

Now we define the addition of two trees,

Definition 4.3.2. *For compatible nodes $\boldsymbol{u}$ and $\boldsymbol{v} \in \boldsymbol{T}^*$,*

$$\boldsymbol{u} + \boldsymbol{v} = \begin{cases} \boldsymbol{u} & \text{if } \boldsymbol{v} = \mathbf{O} \\ \boldsymbol{v} & \text{if } \boldsymbol{u} = \mathbf{O} \\ \Big(\boldsymbol{u}_v + \boldsymbol{v}_v,\ \boldsymbol{u}_{\mathsf{P}},\ \big(\boldsymbol{u}|_{\overline{L}(\boldsymbol{v})} \cup \boldsymbol{v}|_{\overline{L}(\boldsymbol{u})} \\ \quad \cup \{\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}|_{L(\boldsymbol{v})} \text{ and } \boldsymbol{s} \in \boldsymbol{v}|_{L(\boldsymbol{u})} \text{ and } \boldsymbol{r}_{\mathsf{P}} = \boldsymbol{s}_{\mathsf{P}}\}\big) \setminus \{\mathbf{O}\}\Big) & \text{otherwise} \end{cases}$$

Definition 4.3.3 (Notation)**.** *We will use the following notation as a more concise representation of the addition of two sets, such as the sets of children in Eq. 4.3.2, by*

$$\boldsymbol{B} \boxplus \boldsymbol{C} = \big(\boldsymbol{B}|_{\overline{L}(\boldsymbol{C})} \cup \boldsymbol{C}|_{\overline{L}(\boldsymbol{B})} \cup \{\boldsymbol{r}+\boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{B}|_{L(\boldsymbol{C})} \text{ and } \boldsymbol{s} \in \boldsymbol{C}|_{L(\boldsymbol{B})} \text{ and } \boldsymbol{r}_{\mathsf{P}} = \boldsymbol{s}_{\mathsf{P}}\}\big) \setminus \{\mathbf{O}\}.$$

4.4 Properties of a vector space

In this section we prove that the elements of $\boldsymbol{T}$ and operations give us a vector space.

Proposition 4.4.1. *$\boldsymbol{T}$, with scalar multiplication and tree addition is a vector space.*

Proof. We will assume that $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \boldsymbol{T}$ and $a, b \in \mathbb{K}$.

Additive identity element This is explicit in the definition of addition between elements of $\boldsymbol{T}$.

Inverse elements with respect to addition The element $\mathbf{O}$ is its own inverse, since $\mathbf{O} + \mathbf{O} = \mathbf{O}$ by definition.

So, we consider the case of $\boldsymbol{u}$, where $\text{depth}(\boldsymbol{u}) = 1$, then

$$\begin{aligned}\boldsymbol{u} + -\boldsymbol{u} &= (\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) + (-1\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (\boldsymbol{u}_v + -\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= \mathbf{O}\end{aligned}$$

so for any simple node, $\boldsymbol{u}$, $-\boldsymbol{u}$ is its inverse.

Let $\boldsymbol{v}$ be a non-null node which is not simple, but has simple c, that is to say $\text{depth}(\boldsymbol{v}) = 2$. Then

$$\begin{aligned}\boldsymbol{v} + -\boldsymbol{v} &= (\boldsymbol{v}_v - \boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \{\boldsymbol{e} + -\boldsymbol{e} : \forall\, \boldsymbol{e} \in \boldsymbol{v}_C\} \setminus \{\mathbf{O}\}) \\ &= (0, \boldsymbol{v}_\mathsf{P}, \varnothing) \\ &= \mathbf{O}\end{aligned}$$

since the simple leaf nodes are all added to their own additive inverse.

So, we can generalise to trees with a depth greater than two. Assuming that the proposition holds for elements of $\boldsymbol{T}$ with depth n, we consider an element, $\boldsymbol{u}$, where $\text{depth}(\boldsymbol{u}) = n + 1$ added to the element $-\boldsymbol{u}$.

$$\boldsymbol{u} + -\boldsymbol{u} = (0, \boldsymbol{u}_\mathsf{P}, \{\boldsymbol{e} + -\boldsymbol{e} : \forall\, \boldsymbol{e} \in \boldsymbol{u}_C\})$$

Since the child nodes of the root node of $\boldsymbol{u}$ are, by assumption, added to their additive inverses, they then become

$$\begin{aligned}&= (\boldsymbol{v}_v + -\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \varnothing) \\ &= \mathbf{O}\end{aligned}$$

for each $\boldsymbol{v} \in \boldsymbol{u}_C$ and, by induction, our inverse holds for all members of $\boldsymbol{T}$.

Multiplicative identity element First observe that $1\,\mathbf{O} = (1 \cdot 0, 0, \varnothing) = \mathbf{O}$.

Now consider an arbitrary non-null tree in $\boldsymbol{T}$, $\boldsymbol{u}$; $\boldsymbol{u}$ is either simple, or it has child nodes. In the case of a simple $\boldsymbol{u}$, it is obvious that $1 \in \mathbb{K}$ acts as an identity.

$$\begin{aligned}\boldsymbol{u} &= (\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (1\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= 1(\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= 1\,\boldsymbol{u}\end{aligned}$$

Thus, for all leaf nodes on $\boldsymbol{u}$, 1 is the identity for scalar multiplication. Now we take $\boldsymbol{u}$ to be some non-simple node,

$$\begin{aligned}\boldsymbol{u} &= (\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C) \\ &= (1\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, 1\,\boldsymbol{u}_C) \\ &= (1\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \{(1\,\boldsymbol{e}_v, \boldsymbol{e}_\mathsf{P}, \boldsymbol{e}_C) : \forall\, \boldsymbol{e} \in \boldsymbol{u}_C\}),\end{aligned}$$

and, if the children are all simple nodes,

$$\begin{aligned}&= 1(\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C) \\ &= 1\,\boldsymbol{u}.\end{aligned}$$

so it is the case that 1 is the scalar multiplicative identity for nodes which have a depths of less than three.

Now suppose that 1 is the identity for scalar multiplication of all members of $\boldsymbol{T}$ with a depth of n or less for some natural number n, and we consider $\boldsymbol{v}$ which has a depth of $n+1$. Then

$$\begin{aligned}\boldsymbol{v} &= (\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \boldsymbol{v}_C) \\ &= (1\,\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, 1\,\boldsymbol{v}_C)\end{aligned}$$

.

But each of the elements in the set $1\,\boldsymbol{v}_C$ either has a depth of n or less and so the the children of $1\,\boldsymbol{v}$ is merely $\boldsymbol{v}_C$, and so $1\,\boldsymbol{v} = \boldsymbol{v}$.

By induction, we can demonstrate that 1 is the identity for scalar multiplication of nodes of arbitrary (finite) depth.

Commutativity Let us consider compatible nodes $\boldsymbol{u}$ and $\boldsymbol{v}$.

The commutativity of addition involving **O**is guaranteed by the definition of addition, so we first address the case where both addends are simple.

Take $\boldsymbol{u}$ and $\boldsymbol{v}$ to be simple nodes; then

$$\begin{aligned}\boldsymbol{u} + \boldsymbol{v} &= (\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) + (\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \varnothing) \\ &= (\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (\boldsymbol{v}_v + \boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= \boldsymbol{v} + \boldsymbol{u}.\end{aligned}$$

Now suppose that there is some number n for which $\text{depth}(\boldsymbol{u}) \leq n$ and $\text{depth}(\boldsymbol{v}) \leq n \implies \boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}$.

Then if we take $\boldsymbol{u}$ and $\boldsymbol{v}$ to be nodes with depths of $n+1$ or less,

$$\begin{aligned}
\boldsymbol{u} + \boldsymbol{v} &= (\boldsymbol{u}_v, \boldsymbol{u}_P, \boldsymbol{u}_C) + (\boldsymbol{v}_v, \boldsymbol{v}_P, \boldsymbol{v}_C) \\
&= \Big(\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_P, \big(\{\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}|_{L(\boldsymbol{v})} \text{ and } \boldsymbol{s} \in \boldsymbol{v}|_{L(\boldsymbol{u})} \text{ and } \boldsymbol{r}_P = \boldsymbol{s}_P\} \\
&\qquad \cup \{\boldsymbol{r} : \boldsymbol{r} \in \boldsymbol{u}_C \text{ and } \boldsymbol{r}_P \notin \boldsymbol{L}(\boldsymbol{v})\} \cup \{\boldsymbol{s} : \boldsymbol{s}_C \in \boldsymbol{v} \text{ and } \boldsymbol{s}_P \notin \boldsymbol{L}(\boldsymbol{u})\}\big) \setminus \{\mathbf{O}\}\Big) \\
&= \Big(\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_P, \big(\{\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}_C \text{ and } \boldsymbol{r}_P \in \boldsymbol{L}(\boldsymbol{v}) \text{ and } \boldsymbol{s} \in \boldsymbol{v}_C \text{ and } \boldsymbol{s}_P \in \boldsymbol{L}(\boldsymbol{u})\} \\
&\qquad \cup \boldsymbol{u}|_{\overline{L}(\boldsymbol{v})} \cup \boldsymbol{v}|_{\overline{L}(\boldsymbol{u})}\big) \setminus \{\mathbf{O}\}
\end{aligned}$$

So,

$$\begin{aligned}
\boldsymbol{u} + \boldsymbol{v} = \Big(\boldsymbol{v}_v + \boldsymbol{u}_v, \boldsymbol{u}_P, \big(\{\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}_C \text{ and } \boldsymbol{r}_P \in \boldsymbol{L}(\boldsymbol{v}) \text{ and } \boldsymbol{s} \in \boldsymbol{v}_C \text{ and } \boldsymbol{s}_P \in \boldsymbol{L}(\boldsymbol{u})\} \\
\cup \boldsymbol{u}|_{\overline{L}(\boldsymbol{v})} \cup \boldsymbol{v}|_{\overline{L}(\boldsymbol{u})}\big) \setminus \{\mathbf{O}\}
\end{aligned}$$

since addition in $\boldsymbol{T}$ is commutative. If we can demonstrate that the expression for the children is independent of order, then it must be the case that sum of the addends, $\boldsymbol{u}$ and $\boldsymbol{v}$, must also be order independent.

The set $\{\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}_C \text{ and } \boldsymbol{r}_P \in \boldsymbol{L}(\boldsymbol{v}) \text{ and } \boldsymbol{s} \in \boldsymbol{v}_C \text{ and } \boldsymbol{s}_P \in \boldsymbol{L}(\boldsymbol{u})\}$ must be order independent since each of the candidate $\boldsymbol{r}$ and $\boldsymbol{s}$ addends must have a depth of n or less. Since set union is commutative, the order of $\boldsymbol{u}|_{\overline{L}(\boldsymbol{v})}$ and $\boldsymbol{v}|_{\overline{L}(\boldsymbol{u})}$ doesn't affect the result, thus, addition must be commutative for all $\boldsymbol{u}$ where $\text{depth}(\boldsymbol{u}) \leq n+1$. By induction, this must be true for all $n \geq 0$.

Associativity Let us consider compatible nodes $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ in $\boldsymbol{T}$.

First consider the situation where the depths of $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ are all less than or equal to one. If they all have a depth of zero, the sum is trivially the null tree. Similarly, if any have depths of one or less is the zero tree, we also get a trivial result. So we take all of $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ to be simple. Then

$$\begin{aligned}
(\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w} &= ((\boldsymbol{u}_v, \boldsymbol{u}_P, \varnothing) + (\boldsymbol{v}_v, \boldsymbol{u}_P, \varnothing)) + (\boldsymbol{w}_v, \boldsymbol{u}_P, \varnothing) \\
&= (\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_P, \varnothing) + (\boldsymbol{w}_v, \boldsymbol{u}_P, \varnothing) \\
&= ((\boldsymbol{u}_v + \boldsymbol{v}_v) + \boldsymbol{w}_v, \boldsymbol{u}_P, \varnothing) \\
&= (\boldsymbol{u}_v + (\boldsymbol{v}_v) + \boldsymbol{w}_v), \boldsymbol{u}_P, \varnothing) \\
&= \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}).
\end{aligned}$$

Let us consider the case where these may be non-simple trees. Suppose there is an integer n such that associativity holds for any three trees $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$, whose depth is less than or equal to n, that is if $\text{depth}(\boldsymbol{u}) \leq n, \text{depth}(\boldsymbol{v}) \leq n$ and $\text{depth}(\boldsymbol{w}) \leq n$, then it must be the case that

$$(\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w} = \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}).$$

Now suppose one or more of these trees has a depth of $n+1$.

$$\begin{aligned}(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w} &= ((\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C)+(\boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{v}_C))+(\boldsymbol{w}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{w}_C)\\ &= (\boldsymbol{u}_v+\boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C \boxplus \boldsymbol{v}_C)+(\boldsymbol{w}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{w}_C)\end{aligned}$$

Recall that

$$\boldsymbol{u}_C \boxplus \boldsymbol{v}_C = \left(\boldsymbol{u}|_{\overline{L}(\boldsymbol{v})} \cup \boldsymbol{v}|_{\overline{L}(\boldsymbol{u})} \cup \{\boldsymbol{p}+\boldsymbol{q} : \boldsymbol{p} \in \boldsymbol{u}|_{L(\boldsymbol{v})} \text{ and } \boldsymbol{q} \in \boldsymbol{v}|_{L(\boldsymbol{u})} \text{ and } \boldsymbol{p}_\mathsf{P} = \boldsymbol{q}_\mathsf{P}\}\right) \setminus \{\mathbf{O}\}$$

so, letting

$$\boldsymbol{B} = \boldsymbol{u}_C \boxplus \boldsymbol{v}_C$$

we get

$$\begin{aligned}(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w} &= \left((\boldsymbol{u}_v+\boldsymbol{v}_v)+\boldsymbol{w}_v, \boldsymbol{u}_\mathsf{P}, \left(\boldsymbol{B}|_{\overline{L}(\boldsymbol{w})} \cup \boldsymbol{w}|_{\overline{L}(\boldsymbol{B})} \cup \boldsymbol{B} \boxplus \boldsymbol{w}_C\right) \setminus \{\mathbf{O}\}\right)\\ &= \left(\boldsymbol{u}_v+(\boldsymbol{v}_v+\boldsymbol{w}_v), \boldsymbol{u}_\mathsf{P}, \left(\boldsymbol{B}|_{\overline{L}(\boldsymbol{w})} \cup \boldsymbol{w}|_{\overline{L}(\boldsymbol{B})} \cup \boldsymbol{B} \boxplus \boldsymbol{w}_C\right) \setminus \{\mathbf{O}\}\right)\end{aligned}$$

since addition in $\boldsymbol{T}$ is associative for nodes with a depth of n or less

Notice that the elements of all the sets which comprise the children, those in $\boldsymbol{B}$ and in $\boldsymbol{w}_C$, must have a depth of n or less; any addition which occurs amongst the elements of these sets must be associative by our inductive assumption. Hence

$$(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w} = \boldsymbol{u}+(\boldsymbol{v}+\boldsymbol{w}).$$

Compatibility of scalar multiplication and multiplication in $\mathbb{K}$ Observe first that $a\,\mathbf{O} = \mathbf{O}$, $\forall a \in \mathbb{K}$. We also dispose with the case of simple nodes:

$$\begin{aligned}a(b\,\boldsymbol{u}) &= (a(b\,\boldsymbol{u}_v), \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= (ab\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= ((ab)\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= (ab)\,\boldsymbol{u};\end{aligned}$$

So assuming that multiplication is compatible with nodes with depths of n or less, we consider $\boldsymbol{u}$, where $\text{depth}(\boldsymbol{u}) = n+1$,

$$a(b\,\boldsymbol{u}) = a(b\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, b\,\boldsymbol{u}_C)$$

since $\text{depth}(\boldsymbol{e}) \leq n \forall\, \boldsymbol{e} \in \boldsymbol{u}_C$, multiplication of these elements is compatible, and

$$= (ab\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, a(b\,\boldsymbol{u}_C))$$

becomes

$$\begin{aligned}&= ((ab)\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, (ab)\,\boldsymbol{u}_C)\\ &= (ab)\,\boldsymbol{u}\end{aligned}$$

Thus the scalar and field multiplication operators are compatible.

Distribution of scalar multiplication with respect to vector addition Let us consider compatible trees, $\boldsymbol{u}$ and $\boldsymbol{v}$.

First, note that

$$\forall\, \boldsymbol{u} \in \boldsymbol{T}, a(\,\mathbf{O} + \boldsymbol{u}) = a\,\boldsymbol{u} = a\,\mathbf{O} + a\,\boldsymbol{u},$$

and that

$$\forall\, \boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{T}, 0(\,\boldsymbol{u} + \boldsymbol{v}) = \mathbf{O} = 0\,\boldsymbol{u} + 0\,\boldsymbol{v}.$$

The property holds for simple nodes,

$$\begin{aligned} a(\,\boldsymbol{u} + \boldsymbol{v}) &= a((\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) + (\,\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \varnothing)) \\ &= a\,(\,\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (a(\,\boldsymbol{u}_v + \boldsymbol{v}_v), \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (a\,\boldsymbol{u}_v + a\,\boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= (a\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) + (a\,\boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \varnothing) \\ &= a\,\boldsymbol{u} + a\,\boldsymbol{v} \end{aligned}$$

.

So, suppose that the equation $a(\,\boldsymbol{p} + \boldsymbol{q}) = a\,\boldsymbol{p} + a\,\boldsymbol{q}$ holds for all compatible nodes $\boldsymbol{p}$ and $\boldsymbol{q}$ such that $depth(\,\boldsymbol{p}) \leq k$, and depth$(\,\boldsymbol{q}) \leq j$.

Take $n = \min(j,k)$, $a \in \mathbb{K}$, and nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ such that depth$(\,\boldsymbol{u}) = n+1$, and depth$(\,\boldsymbol{v}) = n+1$. Note that n must be greater than zero since the property holds for simple nodes. Then

$$\begin{aligned} a(\,\boldsymbol{u} + \boldsymbol{v}) &= a((\,\boldsymbol{u}_v, \boldsymbol{u}_\mathsf{P}, \boldsymbol{u}_C) + (\,\boldsymbol{v}_v, \boldsymbol{v}_\mathsf{P}, \boldsymbol{v}_C) \\ &= a\Big(\,\boldsymbol{u}_v + \boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \big\{\,\boldsymbol{u}|_{\overline{L}(\,\boldsymbol{v})} \cup \boldsymbol{v}|_{\overline{L}(\,\boldsymbol{u})} \\ &\qquad \cup \big\{\,\boldsymbol{r} + \boldsymbol{s} : \boldsymbol{r} \in \boldsymbol{u}|_{L(\,\boldsymbol{v})} \text{ and } \boldsymbol{s} \in \boldsymbol{v}|_{L(\,\boldsymbol{u})}\big\}\big\} \setminus \{\,\mathbf{O}\}\Big) \\ &= \Big(a(\,\boldsymbol{u}_v + \boldsymbol{v}_v), \boldsymbol{u}_\mathsf{P}, \big\{\{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}|_{\overline{L}(\,\boldsymbol{v})}\} \cup \{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{v}|_{\overline{L}(\,\boldsymbol{u})}\} \\ &\qquad \cup \{a(\,\boldsymbol{r} + \boldsymbol{s}) : \boldsymbol{r} \in \boldsymbol{u}|_{L(\,\boldsymbol{v})} \text{ and } \boldsymbol{s} \in \boldsymbol{v}|_{L(\,\boldsymbol{u})}\}\big\} \setminus \{\,\mathbf{O}\}\Big)\Big) \\ &= \Big(a\,\boldsymbol{u}_v + a\,\boldsymbol{v}_v, \boldsymbol{u}_\mathsf{P}, \big\{\{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}|_{\overline{L}(\,\boldsymbol{v})}\} \cup \{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{v}|_{\overline{L}(\,\boldsymbol{u})}\} \\ &\qquad \cup \{a(\,\boldsymbol{r} + \boldsymbol{s}) : \boldsymbol{r} \in \boldsymbol{u}|_{L(\,\boldsymbol{v})} \text{ and } \boldsymbol{s} \in \boldsymbol{v}|_{L(\,\boldsymbol{u})}\}\big\} \setminus \{\,\mathbf{O}\}\Big)\Big) \end{aligned}$$

.

Notice that the component sets of the set of children to $\boldsymbol{u} + \boldsymbol{v}$, namely $\{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}|_{\overline{L}(\,\boldsymbol{v})}\}$, $\{a\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{v}|_{\overline{L}(\,\boldsymbol{u})}\}$ and $\{a(\,\boldsymbol{r} + \boldsymbol{s}) : \boldsymbol{r} \in \boldsymbol{u}|_{L(\,\boldsymbol{v})}$ and $\boldsymbol{s} \in \boldsymbol{v}|_{L(\,\boldsymbol{u})}\}$ can only contain nodes with a depth of n or less; Thus, we can proceed inductively, increasing the least upper bound, $\min(j,k)$, for the set of trees that cooperate with distribution of scalar multiplication over vector addition, to any value we wish.

Distribution of scalar multiplication with respect to addition in $\mathbb{K}$ The property is clearly true when $\boldsymbol{u} = \mathbf{O}$, since $(a+b)\,\mathbf{O} = \mathbf{O} = a\,\mathbf{O} + b\,\mathbf{O}$.

We first consider simple nodes:

$$\begin{aligned}(a+b)\,\boldsymbol{u} &= (a+b)((a+b)\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= ((a+b)\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= (a\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \varnothing) + (b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \varnothing)\\ &= a\,\boldsymbol{u} + b\,\boldsymbol{u}.\end{aligned}$$

Nodes with a depth of two are slightly more complicated,

$$\begin{aligned}(a+b)\,\boldsymbol{u} &= (a+b)((a+b)\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, (a+b)\,\boldsymbol{u}_C)\\ &= (a\,\boldsymbol{u}_v + b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \{(a+b)\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}_C\})\end{aligned}$$

but $\boldsymbol{u}_C$ is composed of simple nodes, so,

$$\begin{aligned}&= (a\,\boldsymbol{u}_v + b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \{a\,\boldsymbol{e} + b\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}_C\})\\ &= (a\,\boldsymbol{u}_v + b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, a\,\boldsymbol{u}_C) + (b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, b\,\boldsymbol{u}_C)\\ &= a\,\boldsymbol{u} + b\,\boldsymbol{u}.\end{aligned}$$

Now suppose the property holds for nodes with a depth of n. Then we consider node $\boldsymbol{u}$ with a depth of $n+1$:

$$\begin{aligned}(a+b)\,\boldsymbol{u} &= (a+b)(\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P},\ \boldsymbol{u}_C)\\ &= ((a+b)\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, (a+b)\,\boldsymbol{u}_C),\\ &= (a\,\boldsymbol{u}_v + b\,\boldsymbol{u}_v,\ \boldsymbol{u}_\mathsf{P}, \{a\,\boldsymbol{e} + b\,\boldsymbol{e} : \boldsymbol{e} \in \boldsymbol{u}_C\})\end{aligned}$$

since depth($\boldsymbol{e}$) = n

$$= a\,\boldsymbol{u} + b\,\boldsymbol{u}.$$

By induction, the property must hold for all $n >= 0$ □

4.5 Seminorms, norms and metrics

Now that we have a vector space, we can construct model configurations as linear combinations of basis configurations. In the context of models which change their configuration, we need a way for the model itself to combine basis configuration trees by choosing from a set of configurations that are known to exhibit suitable properties. To this end, we need a mechanism for judging how close or far a given configuration is from where it needs to be – we need a way to decompose an extant, running configuration into its basis elements, and then map these to some provably more appropriate configuration. To do this, our structure needs to be a metric space.

We will now construct a seminorm on the vector space $\boldsymbol{T}$. This will induce a norm on a quotient space of $\boldsymbol{T}$ which we can use as a tool for assessing the similarity of trees and, ultimately, provide both a means of clustering trees and selecting trees with particular properties.

4.5.1 T and its classwise seminorm

Definition 4.5.1. *We define the absolute value of a node to be*

$$\langle\!| \boldsymbol{u} |\!\rangle = \begin{cases} 0 & \text{if } \boldsymbol{u} = \mathbf{O} \\ |\boldsymbol{u}_v| & \text{if } \boldsymbol{u}_C = \varnothing \\ |\boldsymbol{u}_v| + \sum_{e \in \boldsymbol{u}_C} \langle\!| e |\!\rangle & \text{otherwise.} \end{cases}$$

The absolute magnitude is only based only on the values of the nodes of trees. Note that each node in a tree can only contribute a non-negative quantity to the absolute value of the tree, it is obvious that $\langle\!| \boldsymbol{u} |\!\rangle \geq 0$ for all $\boldsymbol{u} \in \boldsymbol{T}$ and that equality only occurs if the weight of each node in the tree $\boldsymbol{u}$ is zero.[1]

For any two trees (including those with dissimilar labels) we can define a function analogous to a distance.

Definition 4.5.2. *We will define a semi-norm on the trees:*

$$\langle \boldsymbol{u} \rangle = |\boldsymbol{u}_v|^2 + \sum_{v \in \boldsymbol{u}_C} \langle v \rangle \tag{4.1}$$

and we define the norm in terms of this with

$$\langle\!\langle \boldsymbol{u} \rangle\!\rangle = \sqrt{\langle \boldsymbol{u} \rangle}. \tag{4.2}$$

Both of these functions clearly have non-negative ranges, and they only take the value zero if all the node values and the node labels are zero. Since we restrict ourselves to finite, acyclic trees, we can argue that for any finite set of n trees, we can construct an equivalent vector-based representation of the set by serialising the trees. The defined norm is precisely equivalent to the usual length of vectors in this set. In the example model discussed in Chapter 5 we make use of both this norm and the distance function $\langle\!\langle \boldsymbol{u} - \boldsymbol{v} \rangle\!\rangle$.

Proposition 4.5.1. *For $a \in \mathbb{K}$ and $\boldsymbol{u} \in \boldsymbol{T}$, $|a| \langle\!| \boldsymbol{u} |\!\rangle = \langle\!| a\, \boldsymbol{v} |\!\rangle$.*

Proof. The magnitude of the empty tree is trivially zero, so $|a| \langle\!| \mathbf{O} |\!\rangle = \langle\!| a\, \mathbf{O} |\!\rangle = 0$.

Consider simple nodes in $\boldsymbol{T}$:

$$\begin{aligned} |a| \langle\!| \boldsymbol{u} |\!\rangle &= |a| (|\boldsymbol{u}_v| + 0) \\ &= |a|\, |\boldsymbol{u}_v| \\ &= \langle\!| a\, \boldsymbol{u} |\!\rangle. \end{aligned}$$

Now suppose that there is $n \geq 1$ such that the proposition is true for all trees with a depth of n or less. Then, taking $\boldsymbol{u} \in \boldsymbol{T}$ where $\text{depth}(\boldsymbol{u}) = n + 1$, we have

$$\begin{aligned} |a| \langle\!| \boldsymbol{u} |\!\rangle &= |a| \Big((|\boldsymbol{u}_v| + \sum_{e \in \boldsymbol{u}_C} \langle\!| e |\!\rangle) \Big) \\ &= |a|\, |\boldsymbol{u}_v| + \sum_{e \in \boldsymbol{u}_C} |a| \langle\!| e |\!\rangle \end{aligned}$$

[1] . . . Or, alternatively, the absolute value of every subtree is zero.

but all the elements in $\boldsymbol{u}_C$ have a depth of k or less

$$\begin{aligned} &= ||a|\,\boldsymbol{u}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{u}_C} (\!|a|\,\boldsymbol{e}|\!) \\ &= |a\,\boldsymbol{u}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{u}_C} (\!|a\,\boldsymbol{e}|\!) \\ &= (\!|a\,\boldsymbol{u}|\!). \end{aligned}$$

By induction, the proposition must be true for all $n \geq 0$. □

Proposition 4.5.2. *For* $\boldsymbol{u}$ *and* $\boldsymbol{v} \in \boldsymbol{T}$, $(\!|\boldsymbol{u} + \boldsymbol{v}|\!) \leq (\!|\boldsymbol{u}|\!) + (\!|\boldsymbol{v}|\!)$.

Proof. We start by considering trees of depths zero and one. The case for null trees is trivial: $(\!|\mathbf{O} + \mathbf{O}|\!) = |0 + 0| = 0$, and if only one of the trees has a depth of one, we get either $(\!|\boldsymbol{u} + \mathbf{O}|\!) = (\!|\boldsymbol{u}|\!)$ or $(\!|\mathbf{O} + \boldsymbol{u}|\!) = (\!|\boldsymbol{u}|\!)$.

For $\boldsymbol{u}$ and $\boldsymbol{v}$ with depths of one,

$$(\!|\boldsymbol{u} + \boldsymbol{v}|\!) = (\!|(\,\boldsymbol{u}_v + \boldsymbol{v}_v,\, \boldsymbol{u}_\mathsf{P}, \varnothing)|\!) = |\,\boldsymbol{u}_v + \boldsymbol{v}_v|.$$

Since $\boldsymbol{u}_v$ and $\boldsymbol{v}_v$ are scalars in $\mathbb{K}$, we must have $|\,\boldsymbol{u}_v + \boldsymbol{v}_v| \leq |\,\boldsymbol{u}_v| + |\,\boldsymbol{v}_v|$, so

$$|\,\boldsymbol{u}_v + \boldsymbol{v}_v| \leq |\,\boldsymbol{u}_v| + |\,\boldsymbol{v}_v| = (\!|\boldsymbol{u}|\!) + (\!|\boldsymbol{v}|\!).$$

We will now proceed by induction; let n be a positive integer for which the triangle inequality holds for all trees with a depth of k or less. Let's consider compatible trees, $\boldsymbol{u}$ and $\boldsymbol{v}$ whose depths are less than or equal to $n+1$. Then

$$\begin{aligned} (\!|\boldsymbol{u} + \boldsymbol{v}|\!) &= (\!|(\,\boldsymbol{u}_v + \boldsymbol{v}_v,\, \boldsymbol{u}_\mathsf{P},\, \boldsymbol{u}_C \boxplus \boldsymbol{v}_C)|\!) \\ &= \Big[\,|\,\boldsymbol{u}_v + \boldsymbol{v}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{u}_C\boxplus\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!)\Big]. \end{aligned}$$

Observe that $|\,\boldsymbol{u}_v + \boldsymbol{v}_v| \leq |\,\boldsymbol{u}_v| + |\,\boldsymbol{v}_v|$, and that each of the addends in

$$\sum_{\boldsymbol{e}\in\boldsymbol{u}_C\boxplus\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!)$$

has a depth of n or less, so

$$\sum_{\boldsymbol{e}\in\boldsymbol{u}_C\boxplus\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!) \leq \sum_{\boldsymbol{e}\in\boldsymbol{u}_C} (\!|\boldsymbol{e}|\!) + \sum_{\boldsymbol{e}\in\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!).$$

This implies that

$$(\!|\boldsymbol{u} + \boldsymbol{v}|\!) \leq \Big[\,|\,\boldsymbol{u}_v| + |\,\boldsymbol{v}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{u}_C} (\!|\boldsymbol{e}|\!) + \sum_{\boldsymbol{e}\in\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!)\Big];$$

rearranging we get

$$(\!|\boldsymbol{u} + \boldsymbol{v}|\!) \leq \Big[\,|\,\boldsymbol{u}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{u}_C} (\!|\boldsymbol{e}|\!)\Big] + \Big[\,|\,\boldsymbol{v}_v| + \sum_{\boldsymbol{e}\in\boldsymbol{v}_C} (\!|\boldsymbol{e}|\!)\Big]$$

and hence

$$(\!| \boldsymbol{u} + \boldsymbol{v} |\!) \leq (\!| \boldsymbol{u} |\!) + (\!| \boldsymbol{v} |\!).$$

□

Corollary 4.5.1. *The absolute value forms a seminorm on* $\boldsymbol{T}$.

Proof. Propositions, 4.5.1 and 4.5.2, are sufficient for the absolute value to be a seminorm on $\boldsymbol{T}$. □

At this point we should consider the elements $\boldsymbol{o} \in \boldsymbol{T}$ which are analogues of zero. We define the set $\mathfrak{O} = \{\, \boldsymbol{o} \in \boldsymbol{T} : (\!| \boldsymbol{o} |\!) = 0\}$, and observe that for any $\boldsymbol{e} \in \boldsymbol{T}$, and $\boldsymbol{o} \in \mathfrak{O}$ the equation $(\!| \boldsymbol{e} + \boldsymbol{o} |\!) = (\!| \boldsymbol{e} |\!)$ must hold.

Since $\boldsymbol{T}$ is a seminormed vector space, it is also a pseudometric space and we can induce a fully fledged metric space over the quotient space $\check{\boldsymbol{T}} = \boldsymbol{T}/\mathfrak{O}$.

For simplicity, we identify the coset of $\mathfrak{O}$ with respect to $\mathbf{O}$ with $\check{\mathbf{O}}$, and we take the induced metric on the normed vector space $\check{\boldsymbol{T}}$, to be

$$\mathrm{d}(\check{\boldsymbol{u}}, \check{\boldsymbol{v}}) = (\!| \check{\boldsymbol{u}} - \check{\boldsymbol{v}} |\!) \text{ for all } \check{\boldsymbol{u}}, \check{\boldsymbol{v}} \in \check{\boldsymbol{T}}.$$

We will continue to use $(\!| \check{\boldsymbol{u}} |\!)$ to denote the the induced absolute value of $\check{\boldsymbol{u}} \in \check{\boldsymbol{T}}$.

4.6 Element multiplication in $\check{T}$ and establishing the properties similar to those of a ring

In this section, we will define a multiplicative operator for trees. We want to establish these properties mainly so that it broadens the set of mathematical tools we have at our disposal to analyse sets of trees (clustering, classification, interpolation, extrapolation…). The element $(1, 1, \varnothing)$ acts as a multiplicative identity.

Definition 4.6.1. *We define the multiplication of two trees (or nodes) to be*

$$\check{\boldsymbol{u}} \cdot \check{\boldsymbol{v}} = \begin{cases} \check{\mathbf{O}} & \text{if } \check{\boldsymbol{u}} = \check{\mathbf{O}} \text{ or } \check{\boldsymbol{v}} = \check{\mathbf{O}} \\ (\, \check{\boldsymbol{u}}_{\nu}\, \check{\boldsymbol{v}}_{\nu},\ \check{\boldsymbol{u}}_{\mathrm{P}}\, \check{\boldsymbol{v}}_{\mathrm{P}},\ \check{\boldsymbol{u}}_{\mathrm{P}}\, \check{\boldsymbol{v}}_{C} \boxplus \check{\boldsymbol{v}}_{\mathrm{P}}\, \check{\boldsymbol{u}}_{C}) & \text{otherwise.} \end{cases}$$

where the notation $\check{\boldsymbol{u}}_{\mathrm{P}}\, \check{\boldsymbol{v}}_{C}$ or $\check{\boldsymbol{v}}_{C}\, \check{\boldsymbol{u}}_{\mathrm{P}}$ corresponds to the set obtained by multiplying each label in the root node of the set $\check{\boldsymbol{v}}_{C}$ by the label $\check{\boldsymbol{u}}_{\mathrm{P}}$. Clearly, if the set in the operation (on either side) is null, then the result is null.

Proposition 4.6.1. $\iota = (1, 1, \varnothing)$ *commutes with all other nodes, is the multiplicative identity, and it is unique,*

Proof. Let $\check{v}$ be some arbitrary tree, then

$$\begin{aligned}(\check{v}_\nu, \check{v}_\mathrm{P}, \check{v}_C) \cdot \iota &= (\check{v}_\nu 1, \check{v}_\mathrm{P} 1, \check{v}_C 1 \boxplus \varnothing \check{v}_\mathrm{P}) \\ &= (1\check{v}_\nu, 1\check{v}_\mathrm{P}, 1\check{v}_C \boxplus \check{v}_\nu \varnothing) \\ &= (1\check{v}_\nu, 1\check{v}_\mathrm{P}, \check{v}_\nu \varnothing \boxplus 1\check{v}_C) \\ &= (1\check{v}_\nu, 1\check{v}_\mathrm{P}, 1\check{v}_C) \\ &= \iota \cdot (\check{v}_\nu, \check{v}_\mathrm{P}, \check{v}_C) \\ &= (\check{v}_\nu, \check{v}_\mathrm{P}, \check{v}_C)\end{aligned}$$

□

We can see that, since both the weight and label are members of a field, the only possible value for both the weight and the label of the identity is one. This leaves us to consider our options for the set of children. Suppose we have an alternative identity, I, with a non-empty set of children; then the set of children in the product must be $\check{v}_C 1 \boxplus \check{\boldsymbol{I}}_C \check{v}_\mathrm{P}$. For this to be the identity, $\check{v}_C \boxplus \check{\boldsymbol{I}}_C \check{v}_\mathrm{P}$ must equal $\check{v}_C$. This means, however that $\check{\boldsymbol{I}}_C \check{v}_\mathrm{P}$ contributes only trees which are members of $\mathfrak{O}$, but this implies that $\check{\boldsymbol{I}}_\nu = 0$ which contradicts our observation that it must be 1.

Now we must prove that the necessary multiplicative properties so that we can be confident that arithmetic involving trees works in the "normal" way.[2]

Proposition 4.6.2. *Multiplication of trees in $\check{\boldsymbol{T}}$ is commutative.*

Proof. Suppose there is a number n such that multiplication is commutative for all trees $\check{u}$, $\check{v}$ such that $\mathrm{depth}(\check{u}), \mathrm{depth}(\check{v}) \leq n$.

Multiplication involving nodes with a depth of zero clearly commutes, so n may reasonably take the value 0.

In the case where both nodes are simple, it is evident that they must commute, since scalar multiplication commutes, and the multiplication of ring of multinomialsis commutative.

Suppose that one or both of the nodes has a depth of $n+1$. The children of the nodes all have depths of n or less, so the elements of the children of the product must be independent of the order of the operators in the multiplication, and both scalar multiplication and multiplication in the ring of multinomialscommute. Hence the multiplication of nodes with a depth of $n+1$ must commute. By induction, we can say that trees of arbitrary depth commute with this definition of multiplication in $\check{\boldsymbol{T}}$. □

Proposition 4.6.3. *Multiplication of trees in $\check{\boldsymbol{T}}$ is associative.*

[2]Ideally, we would have inverses for trees (and hence a multiplicative identity), but, like matrices, this may only be possible (if it is at all) for a comparatively small part of $\check{\boldsymbol{T}}$.

Proof. Let $\check{u}$, $\check{v}$, and $\check{w}$ be nodes in $\check{T}$. Then,

$$\begin{aligned}\check{u}\cdot(\check{v}\cdot\check{w}) &= (\check{u}_\nu, \check{u}_\mathsf{P}, \check{u}_C)\cdot(\check{v}_\nu\,\check{w}_\nu, \check{v}_\mathsf{P}\,\check{w}_\mathsf{P}, \check{w}_\mathsf{P}\,\check{v}_C \boxplus \check{v}_\mathsf{P}\,\check{w}_C)\\ &= (\check{u}_\nu\,\check{v}_\nu\,\check{w}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}\,\check{w}_\mathsf{P}, \check{u}_\mathsf{P}(\check{w}_\mathsf{P}\,\check{v}_C \boxplus \check{v}_\mathsf{P}\,\check{w}_C)\boxplus \check{v}_\mathsf{P}\,\check{w}_\mathsf{P}\,\check{u}_C)\end{aligned}$$

but tree addition is both commutative and associative, and multiplication in the ring of multinomialsalso commutes, so

$$\begin{aligned}&= (\check{u}_\nu\,\check{v}_\nu\,\check{w}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}\,\check{w}_\mathsf{P}, \check{u}_\mathsf{P}\,\check{w}_\mathsf{P}\,\check{v}_C \boxplus \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}\,\check{w}_C \boxplus \check{v}_\mathsf{P}\,\check{w}_\mathsf{P}\,\check{u}_C)\\ &= (\check{u}_\nu\,\check{v}_\nu\,\check{w}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}\,\check{w}_\mathsf{P}, \check{w}_\mathsf{P}(\check{u}_\mathsf{P}\,\check{v}_C \boxplus \check{v}_\mathsf{P}\,\check{u}_C)\boxplus \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}\,\check{w}_C)\\ &= (\check{u}_\nu\,\check{v}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{u}_\mathsf{P}\,\check{v}_C \boxplus \check{v}_\mathsf{P}\,\check{u}_C)\cdot(\check{w}_\nu, \check{w}_\mathsf{P}, \check{w}_C)\\ &= (\check{u}\cdot\check{v})\cdot\check{w}\end{aligned}$$

□

Proposition 4.6.4. *Tree-multiplication distributes over tree-addition in* $\check{T}$.

Proof. We want to show that for $\check{u}$, $\check{v}$, and $\check{w}\in\check{T}$, where nodes $\check{v}$ and $\check{w}$ are compatible, $\check{u}\cdot(\check{v}+\check{w}) = \check{u}\cdot\check{v}+\check{u}\cdot\check{w}$ is true.

Let us first consider multiplication of a sum by a node with a depth of one, $\check{u}$, over the the sum $\check{v}+\check{w}$,

$$\begin{aligned}\check{u}\cdot(\check{v}+\check{w}) &= (\check{u}_\nu, \check{u}_\mathsf{P}, \varnothing)\cdot(\check{v}_\nu, \check{v}_\mathsf{P}, \check{v}_C) + (\check{w}_\nu, \check{w}_\mathsf{P}, \check{w}_C)\\ &= (\check{u}_\nu, \check{u}_\mathsf{P}, \varnothing)\cdot(\check{v}_\nu+\check{w}_\nu, \check{v}_\mathsf{P}, \check{v}_C\boxplus\check{w}_C)\\ &= (\check{u}_\nu(\check{v}_\nu+\check{w}_\nu), \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{u}_\mathsf{P}(\check{v}_C\boxplus\check{w}_C)\boxplus(\check{v}_\mathsf{P}\varnothing))\\ &= (\check{u}_\nu(\check{v}_\nu+\check{w}_\nu), \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{u}_\mathsf{P}(\check{v}_C\boxplus\check{w}_C))\\ &= (\check{u}_\nu\,\check{v}_\nu+\check{u}_\nu\,\check{w}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{u}_\mathsf{P}\,\check{v}_C\boxplus\check{u}_\mathsf{P}\,\check{w}_C)\\ &= \check{u}\cdot\check{v}+\check{u}\cdot\check{w}\end{aligned}$$

Note that this is independent of the depths of nodes $\check{v}$ and $\check{w}$.

Suppose then that there is an integer n such that multiplication of nodes with a depth of n or less distributes over addition, and we consider the case where our factor, $\check{u}$, has a depth of $n+1$ or less. Then

$$\begin{aligned}\check{u}\cdot(\check{v}+\check{w}) &= (\check{u}_\nu, \check{u}_\mathsf{P}, \check{u}_C)\cdot\Big((\check{v}_\nu, \check{v}_\mathsf{P}, \check{v}_C) + (\check{w}_\nu, \check{w}_\mathsf{P}, \check{w}_C)\Big)\\ &= (\check{u}_\nu, \check{u}_\mathsf{P}, \check{u}_C)\cdot(\check{v}_\nu+\check{w}_\nu, \check{v}_\mathsf{P}, \check{v}_C\boxplus\check{w}_C)\\ &= (\check{u}_\nu(\check{v}_\nu+\check{w}_\nu), \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{v}_\mathsf{P}\,u_C\boxplus\check{u}_\mathsf{P}(\check{v}_C\boxplus\check{w}_C))\end{aligned}$$

but since $\check{u}_\mathsf{P}\in\mathbb{K}[A]$ and polynomial multiplication distributes over addition

$$\begin{aligned}&= (\check{u}_\nu\,\check{v}_\nu+\check{u}_\nu\,\check{w}_\nu, \check{u}_\mathsf{P}\,\check{v}_\mathsf{P}, \check{v}_\mathsf{P}\,u_C\boxplus(\check{u}_\mathsf{P}\,\check{v}_C\boxplus\check{u}_\mathsf{P}\,\check{w}_C))\\ &= (\check{u}_\nu, \check{u}_\mathsf{P}, \check{u}_C)\cdot(\check{v}_\nu, \check{v}_\mathsf{P}, \check{v}_C) + (\check{u}_\nu, \check{u}_\mathsf{P}, \check{u}_C)\cdot(\check{w}_\nu, \check{w}_\mathsf{P}, \check{w}_C)\\ &= \check{u}\cdot\check{v}+\check{u}\cdot\check{w}.\end{aligned}$$

□

4.7 Discussion

This set of metric spaces arose from attempts to capture the nuanced associations in survey questions like *"Thinking about the weather forecast, how would you rate the chances of your favourite sporting team in the coming match?"* and to be able to incorporate the sorts of conflicting data that respondents may provide into simulation models. Initially, the trees were no more than data structures with a rough and ready distance function, but as the work became more coherent, the underlying mathematical structure began to emerge, and the realisation that the trees might be useful for representing more than survey responses came about. The basic heuristic comparisons used in exploring the survey data were replaced with a better behaved metric based on the tree norm.

The loosely defined structure was defined and converted into a vector space so that we can construct model-spaces from a set of basis elements corresponding to submodels. Extending this structure to the assessment of configurations required a metric space.

In principle, we can treat each distinct symbol in a label as a length in an axis orthogonal to each of the other symbols. This interpretation makes a heuristic "distance" function between trees quite simple. This avenue has not yet been explored; the trees used in the example model of Chapters 3 and 5 largely makes use of the simple functions for cardinality, overlap and shadow.

In the example model developed later, the states of the model as a whole, subdomains of the model and the components within the model are represented by representative trees. There is also a set of trees which are identified by known-good configurations, and the mechanism which handles switching within the model uses the metric in its assessment. Like the model in Chapter 2, the approach is relatively simple, but the hope is that, having established the ring-like properties, more advanced clustering and discrimination techniques can be brought to bear.

In Chapter 3 the model developed is concerned with demonstrating the adaptive selection of models using abstracted representations of the model's components and of the model's configuration. These representations are in the form of trees in $\check{\boldsymbol{T}}$ with particular forms.

CHAPTER 5

Theory and an example implementation

Chapter 2 demonstrated that switching models can provide benefits in fidelity and efficiency, when compared with non-switching alternative models, and that maintaining state information across changes in representation are both feasible and beneficial. The example developed in Chapter 3 describes a model of a small trophic network to use in exploring how a general model might be constructed. Here, we describe a prototype framework, Remodel, with an accompanying model resembling the example described in Chapter 3 implemented in the framework.[1] In this chapter, submodels are models which are embedded in a multiprocessing system which, taken in aggregate, forms a more complex model which is based upon the Remodel framework. In this chapter, the term "submodel" will be used when we are explicitly dealing with subsidiary models. They may also be referred to a "models" from time to time, and the context should indicate which level of aggregation is implied. The term "framework" refers the machinery which supports a set of models or submodels to run and interact.

A guiding principle in the development of Remodel is that the construction of a submodel should conform – as much as is possible – to an equivalent model in a more conventional context: the body of an agent's code should look and behave much like the code of a similar agent in other models or frameworks. These goals have consequences in both the underlying framework and in the expression of the model in that framework. In this chapter, we will discuss the ways these consequences are addressed and other issues that have influenced the construction of the framework. The aim of this chapter is to discuss explicit solutions to problems that arise in complex model-swapping systems, and to reify the ideas that relate to using the states of the participating agents to determine an appropriate mix of representations for the simulated entities. It should also provide a pattern for researchers that wish to construct models or frameworks that support changes in the representation of simulated entities

[1] The framework and model this chapter refers to is released under the GPLv3 and available by running `git clone` **`http://github.com/snarkypenguin`**.

or systems.

5.1 Formative design considerations

The general components of this example framework follow conventional patterns for agent-based models. The framework was influenced by previous work, Gray et al. [2006, 2014], which formed the core operational models for management strategy evaluation (MSE) dealing with anthropogenic effects on marine ecosystems. The principle behind MSE and adaptive management is that candidate strategies for managing the effects of human activity are explored in simulation.[2] Adaptive strategies actively monitor conditions in their domain[3] and change management strategies to suit the observed conditions and dynamics of the system. [Walters and Hilborn, 1976, Smith, 1993, Polacheck et al., 1999, Sainsbury et al., 2000, Keith et al., 2011] Each of these three major projects has provided the opportunity to explore the issues associated with modelling increasingly complex systems over increasingly large areas.

The focus of Gray et al. [2006] was broad and required a flexible approach to the problem of simulating the interactions of human activity and components of local ecosystems. In it, time was treated as a continuous variable, and a number of new kinds of simulated entities were required to reflect the activities influencing the system and the management of these activities. This required a more abstracted view of what constituted an "agent" in the system. Most species were able to be represented by more than one class of agent, and the range of representations included, for animals, individuals, super-individuals, and several forms representing populations; benthic communities could be represented by irregular cellular automata, or populations. This multiplicity of representations brought a great deal of flexibility in configuring model runs and case testing. It also used distinct representations for different life-stages of some species within the model. Gray et al. [2014] incorporated most of the conceptual development of the previous work, but took a novel approach to simulating whale-sharks: the whale-sharks are individually recognisable and were removed from the common domain as a part of their annual migration and reproduction cycle, maintained in a subsidiary model, and then reintroduced. Its implementation was awkward and constrained by the limitations of the software environment which had evolved to support the rest of the system.

The maintenance of state for superceded representations evolved from the basic strategy developed in Chapter 2. In the model of Chapter 2, the only interactions were between the simulated organisms and the environmental plume and there was no need to include mechanisms for agents to interact; this is obviously not usually the case.

[2] This is effectively impossible to do in the real world, since the application of a strategy has a high likelihood of changing the baseline for other candidates in the trial.

[3] The simulation must follow the same sampling protocols of actual environmental monitoring surveys.

The Remodel framework extends these concepts by allowing independent agents to assess the state of the system and suggest or apply changes in the representation mix. The assessement process within a model in the framework is explicitly considered part of the model and, as such, may be subject to change during a simulation. Remodel also has the machinery to support sets of state data across transitions in order to minimise the "edge effects" which can occur across transitions.

5.2 Principles

Large, complex models have problems which are not relevant to smaller exploratory models: communication between (instances of) submodels, the sequencing of events within the system, consistency in the treatment of temporal scales, spatial scales, and units of measurement, for example. A number of well known principles from the domain of computer science can be employed to reduce the angst associated with some of these issues, but they do not address the issues of *suitability* which can arise when cooperating submodels actually require features which their partners are unable to provide, or the basic dynamics within a representation is unsuited to either its own state or the state of other components that it depends on. It is the *monitor* agents that act to preserve a consistent mix of submodels.

5.2.1 Interactions

The interaction between two agents should be conducted by calls through the kernel which manages the execution of the simulation. Agents are implemented as *closures* in the framework: basically functions with hidden data. The kernel is told (by programming it) how to interact with an agent – to find out what it has or can do, to read or modify its data, and to act as an intermediary on the agent's behalf. In this way, making an agent "accessible" only requires a small set of code, and no other agents need prior knowledge of what an agent provides. [4] In the exemplar discussed below, the "kernel" interface of the framework is used to provide the accessor and mutator functions associated with every submodel.

This approach to interaction has a number of significant benefits: it reduces the total amount of code needed to include a new submodel, it makes the task of parallelisation or of rigging a model for distributed computation much easier, and it provides a well exercised mechanism for an agent to obtain or modify data in another instance of a submodel.

[4] This is similar to interprocess communication in Unix. One of the best books covering this is *Advanced Programming in the Unix environment* by W. Richard Stevens and Stephen A. Rago, published in the Addison Wesley Professional Computing Series; ISBN-13: 978-0321637734, ISBN-10: 0321637739.

5.2.2 Time

Modelled entities or processes may have quite different natural time scales, and the timescales appropriate to a cargo ship are unlikely to suit a speedboat. Even within a submodel, the time steps may vary according to what the submodel might be doing: a kitten may "sleep" twenty hours a day, eat for ten minutes and spend the rest of the time attacking things. Choosing to model *sleep* with the same the same time-step as *combat* will be slow, and may increase error.

A model with a number of these sorts of components – particularly when they cover a range of submodels – must deal with a system where "now" may be different for each of agents. This issue similar to those in multitasking and real-time operating systems, and a similar approach is taken: all entities are entered into a time ordered queue and each is run for some amount of time before being re-entered into the queue. When an entity ha a subjective notion of "now" which is significantly farther in the future that an agent which wishes to interact with it, we coerce them into synchrony. If they are *close enough*, a simpler alternative is to only agents to interact if they are "near enough" in time. Gray et al. [2014] and Gray et al. [2006] dealt with the need for synchrony by allowing agents to make requests through the kernel to become synchronous with other agents. In these studies, some submodels were treated as "always synchronous," such as bathymetry, or "nearly always synchronous" such as coral reefs. In these agents, changes could be accumulated with appropriate adjustments when interactions with other agents occurred.

Clearly it should be the case that the flow of time is monotonic for each participant in the model. We should be able to prioritise some agents so that they routinely run before others (such as the *monitor* agents) and within a time-slot and priority group we ought to be able to introduce a randomness in their insertion order. Introducing this randomness ensures that no individual agent or group of agents has a systematic advantage as a result of when it was first inserted into the queue. With that as the general goal, we can also entertain the idea of incorporating global checkpoints from which a simulation can be restarted non-deterministically.

5.2.3 Changing representation

On the surface, changing the representation of an instance of a submodel would seem to increase the complexity of the system as a whole. In some sense this is true, but such changes may also simplify the system by either reducing the number of agents in the simulation, or by making the computation simpler (even in the face of an increased number of agents).

Recall that a change in representation is triggered by the recognition of some condition indicating that the current situation is not optimal. It may be true that transitions may engender artifacts akin to the "ringing" associated with sharp transitions in continuous systems, the premise is that we have sufficient cause to change to a "better" representation, and that we may be able

to mitigate these effects. It seems likely that conditioning an individual-based model of an organism accurately enough to produce observed dynamics in large populations is much more difficult than a controlled change between population-level representations and individual-based representations when appropriate. Some individual characteristics may need special attention: the behaviour of an individual may be permanently altered by the conditions it encounters, or epigenetic changes in response to the environment may influence the development of an individual's progeny, so these data would need to be maintained in some way across representation changes and factored into the pertinent facets of that representation, such as vulnerability to predation or the nature of progeny.

5.2.4 Assessment and adaptation

As a model run progresses, data is generated and output which records a subset of the state of the system is usually written to a file or database. This process often occurs completely within a the body of a submodel and is independent of any other facet of the model. Similarly, submodels may monitor aspects of their state and adjust their operation to best fit the prevailing conditions. It is easy to see how an agent may decide to convert to a new representation, such as an adult lamprey or salmon converting to a breeding adult. This kind of change is relatively straightforward, but can be limited in its application: it occurs in a piecemeal way, happening – or not – on an individual-by-individual basis.

A more nuanced approach assesses sets of agents in order to determine whether a representation change is appropriate, such as basing the decision to convert a set of individual-based kangaroos in a given area to an equation-based model which uses both the state of the kangaroos, the state of the grass and the water content of the soil. This sort of assessment can be carried out by another agent within the system: a *monitor*. Monitors are constructed to select configurations of an ensemble of agents and to effect a change in the configuration of the system. Each monitor is acts both as an analogue of an objective function which is used for optimisation, and an agent of that optimisation. In this way, we can choose what we optimise

5.3 Framework and Example implementation

Remodel[5] is used to implement a model based on the the description of the example in Chapter Chapter 3. In the example, the domain consists of nine cells containing tree-like plants. These trees are a critical food source for a population of herbivores, and the trees rely on the young herbivores to eat the ripe fruit in order to make their seeds viable (perhaps the seeds have a particularly

[5]The printing styles associated with the code in the example implementation are noted in the appendix. Table A.1.

tough seedcase). A carnivore which preys solely on juvenile herbivores is introduced into a stable system. In Chapter 3, the introduction of the carnivore and a subsequent range-land fire initiated a sequence of events which, in turn, engendered changes in the configuration of the model as a whole. The implementation described here is not identical to the system described in the paper – some inadequacies of the model presented in the paper have been addressed (such as the inability of the plants to recolonise a cell after they have become extinct in it).

5.3.1 Scheme

The design decisions for the modelling framework was heavily influenced by experience with large ecosystem models which were primarily developed using C++ and C.

Scheme was designed with a minimalist approach, and is a useful medium for exploring fundamental ideas about language design and programming. Its small size and simple syntax make it relatively quick to learn. Scheme is a lexically scoped member of the LISP family. This means that variable scope in the code itself is structured like more like C, Pascal or Java, than some of the other members in the LISP family. As a dialect of LISP, Scheme makes extensive use of parentheses[6] to delimit statements, and this can be daunting to newcomers.

Remodel was implemented in the Scheme programming language because it possesses particular properties that are only now becoming part of the standard in members of the C family, such as anonymous functions, and first-class closures (described later).

A number of factors were important in the choice of implementation language, namely

- class-based multiple inheritance with polymorphism
- weak/dynamic typing
- linking with compiled code in other languages,
- able to support maintenance models
- potential for parallelising and distributed execution.

and

- interpreted operation if possible,

These constraints excluded a number of familiar languages, such as C++, C, Java, C#, Python and DD.

An interpreted language allowed a simple path to incorporating both functions[7] and numeric quantities with physical units in the parameterisation of modelled entities in the parameter files.

Scheme was designed as a tool for exploration Sussman and Steele [1998] and

[6] As well as brackets and braces in some dialects

[7] The mass-at-age functions for trees and animals are defined as parameters, rather than hard-coded in the body of the framework, for example

it remains an excellent tool for this purpose. As a language, it is most notorious for its `call-with-current-continuation` function (usually encountered as `call/cc`). Loosely speaking, this is a generalisation of flow control structures. Its use has been avoided in the code of Remodel, since the framework is intended to be a tool in the discussion of model-swapping methods, and the behaviour of injudicious use of `call/cc` is often hard to debug. Future work on the framework is likely to include implementing constraint solving routines that might employ `call/cc`, but these would be carefully managed.

Scheme is an actively developing language, and there are a number of very good interpreters and compilers for it; many of these implementations are also amenable to integration with other languages, either through directly linking binary objects or by interacting through a virtual machine (such as a JVM).

Interpreter/Compiler

Gambit-C[8] was used for the implementation language. It is a well regarded implementation of Scheme which runs on all of the major platforms (Linux, Unix, OSX, Android and Windows). Gambit incorporates both a mature Scheme interpreter and a compiler, and both can dynamically link programs with external, compiled code and libraries which may exist either as compiled C or Scheme code. The choice to use Gambit, rather than another version of Scheme, was heavily influenced by several points: it was the implementation used for the model in Chapter 2; its ability to link to compiled C code and its own compiled code can combine to produce programs which are noticeably faster than purely interpreted code; it supports very C-like infix notation (indeed, it can be programmed using syntax which is almost-native C syntax); and adding hand-crafted C or C++ code for particularly intensive routines is not difficult.

Porting Remodel to other versions of Scheme should be relatively simple; the only potentially awkward issue in using a different Scheme implementation would be the use of `d`efine-macro in the creation of the classes and methods in a model. These macros are used to automatically incorporate code which maintains data-structures that provide information essential to the communication between submodels, and the system as a whole. The macros are also able to detect and respond to some sorts of oversights or inconsistencies in the construction of methods or model-bodies. There are a number of approaches to constructing macros or macro-like syntax in Scheme; unlike many language families, many members of the LISP family may have their syntax extended, and there are a number of methods available to do this. Definitions of classes and methods use the most primitive mechanism of syntactic extension, `d`efine-macro, which is analogous to the `#define` found in C and C++.

Unfortunately, the use of `d`efine-macro complicates the process of locating syntax errors, since the interpreter is unable to track line numbers within a macro. [9] The only aspect associated with using these extensions which would

[8] http://gambitscheme.org

[9] The macros which wrap the methods called by an agent are implemented in such a way that

require broader changes with d efine-syntax is synthesising the context-specific identifiers (such as <animal>-model-body) which refer to the "body" of the different classes of agents; this is not insurmountable, but will be left for a subsequent version of the modelling framework.

First class closures and maintenance closures

There are alternatives to using first-class closures to maintain state-variables across representation changes, but these methods are not opaque, and it is more difficult to "protect" the data from accidental modification. A closure is essentially a function which is coupled with private state-data. The function may take arguments, operate on its state, and may return values, but no external code can directly modify its state. This feature means that an arbitrary representation of an entity typically knows nothing about the internal features of any maintenance closures it carries, and can only interact with them according to the services they provide.

A simple example of a closure might look like so:

```
;; The closure will be called like (@ 'tick dt)
(define @       ;; this line defines a variable or function
  (let ((TBT 0) ;; this reads "let the variable TBT take the value 0, and
          ;; the variable depuration-rate take
          ;; the value 0.0001
          ;; tributyltin
  (depuration-rate 0.0001))
 (lambda (action #!rest x) ;; "lambda" constructs a function
(cond ;; select the first true condition -- e.g.
 ;; (eqv? action 'reset), and execute the rest of the
 ;; contents of the parenthetic block, then exit
 ((eqv? action 'reset) (set! TBT 0))
 ((eqv? action 'value) TBT)
 ((and (eq? action 'tick) (= (length x) 1)
 (number? (car x)))
  (set! TBT
  (* TBT (- 1 (exp (* depuration-rate (cadr x)))))))
 (else 'ignored) ;; if none of the conditions are true
 ))))
```

This example maintains a contaminant level in the absence of contact with the contaminant, TBT. If the modelled entity enters a situation where TBT is present, a monitor class may decide that a representation change is necessary, and the closure would be queried for the correct value for the contaminant level to use in the construction of the new representation.

The variable is set to point to the function the lambda defines, and the state variable *TBT* is "global" to this function, rather like a static variable defined

the code they produce can be written to a file, and that file may take the place of the original block of code. In this way, the difficulties with respect to line number tracking can be managed, Such code could be used to generate a code-base for a model which was free of macros.

outside a function in a C source file. Closures are a coupling of a procedural element with data which is preserved across invocations of the procedure. An agent maintaining this closure would be able to fetch the value of TBT, cause the closure to run the depuration code for a time step (`dt`) or reset the contaminant level to zero, but nothing else. More complex closures would typically be able to provide a list of tags which specify external properties which are to be used in their update step.

5.3.2 SCLOS– a Scheme implementation of CLOS

The class structure in the Remodel framework makes use of the Scheme implementation of CLOS that was written by Gregor Kiczales [Kiczales et al., 1993] while he was working at Xerox PARC in 1992 and 1993. Kiczales has been an instrumental researcher and proponent for the use of meta-object protocols (MOPs) as a tool for making computer programs clearer, more efficient, and more robust. The basic tenet is that generic methods or functions are used to manipulate objects, and that these generic objects inspect the nature of the data being passed to them and pass the processing to a specialised function of the same name which deals with the task most appropriately. While this is not an exact analogue to the problems we seek to address in this work, there is a substantial similarity and using SCLOS with its implicit MOP seemed a natural fit. SCLOS has been the basis for many of the significant object-oriented systems for Scheme. In the development of Remodel there have been some instances where the "wrong" methods seemed to have been called to act on an object, but these have inevitably been associated with a type error in the signature of an overloaded method (in the SCLOS idiom, that would be a generic method with more than one implementation), or a case where a generic method has been defined twice. These errors are now trapped by the framework by ensuring that there are no duplicate generic methods.

The basic SCLOS library has been slightly modified (the <***<object>***> class of SCLOS has been renamed to <***primitive-object***>) and additional support routines have been added in the file sclos+extn.scm. These additions include the recognition of a number of Scheme data-types, routines to examine the parents of an object, registers which recognise objects and classes which have been instantiated, extended support for the initialisation of slots in an SCLOS object (parameter files), and classification predicates.

5.3.3 Class structure

The submodels in the example are all derived from a basic <***agent***> class which provides each modelled entity the facilities for interacting with the kernel of the example implementation and mechanisms for the agents to obtain information about the other agents and a means of interaction.

The taxonomy of the classes is fairly conventional: the primitive *agent* is the superclass of almost all of the implementation's components (exceptions in-

clude classes for things like bounding regions), and it provides submodels with access to the infrastructure of the system.

At the lowest levels, the classes provide definitive features that characterise broad categories of entities, such as discrete locations, delineated areas, the ability to map between ordinate systems, or the ability to poll other agents at regular intervals.

Table 5.1: Fundamental classes in the Remodel framework framework-classes.scm

Classname	**role**
<***agent***>	The agent class provides the fundamental components for a model to run within the framework, such as essential state variables and common methods
<***projection***>	This class is derived from <***object***>; it adds methods and state variables to allow mapping between an agent's native coordinate system and the common coordinate system. It extensively used by the routines that output graphical data[10], – mapping each agent from its domain to the domain of the output page or image.
<***introspection-agent***>	The methods which allow <***monitor***> and <***log-introspection***> agents to operate is provided by this class – it forms the basis for agents which poll other agents
<***plottable***>	Children of this class can be plotted in map-like output. The attributes of this class include a "glyph" a location, an orientation, a preferred font, colour, glyph size (which may be scaled by the value of a slot) and an additional magnification which can be used to adjust the scale. This is the class that provides location and orientation to the animal and plant classes.
<***environment***>	An agent's basic spatial context is provided by this class's bounding box and a link to an explicit representation

The classes in Table 5.1 provide the basic mechanisms essential to agents of each branch. They provide a means of communication with the kernel of the framework, and mappings between the internal and external representations of agent's spatial domains.

Table 5.2: More fundamental classes – framework-classes.scm

Classname	**role**
<***object***>	This is the "primal" class for all of the different agent classes, and the more complex data structures like polygons
<***array***>	Arrays are used to construct a collection of lists which often act as the basis for a model representation between a fully defined individual-based representation and an analytic representation. Lacks the potential temporal sensitivity of an individual-based representation, but able to maintain many i-state variables (*sensu*Caswell and John [1992])
<***diffeq-system***>	Agents derived from this class run a Runge-Kutta4 algorithm for a system of differential equations. Class members don't maintain variables as such, they are passed "getters" and "setters" which obtain state variable values from other agents (such as *ecoservice* agents) and subsequently set new values in those agents.
<***proxy***>	Proxies represent some "element" (like a row in an <***array***> agent) when engaging in individual-to-individual interactions.
<***model-maintenance***>	Instances of this class aren't really agents in the usual sense. They provide the support for maintaining state variables for representations which are not currently active (such as in Chapter 2).

Introspection agents

The <***introspection***> class is geared to to supervising and extracting data from the set of running agents. The classes of agents (listed in Table 5.3) are derived directly or indirectly from <***introspection***> and the classes <***logger***> and <***monitor***> classes exist solely to poll the other agents within the system and, respectively, either extract information in an analogue of the sampling undertaken by researchers, or to monitor the states of agents, ensembles and the model as a whole in order to be able to effect changes in the constituents of the model ensemble when necessary. The loggers handle all of the generation of output files – the code which makes up the submodels only provides one or more ***log-data*** methods which the loggers use to assemble their output. The notion is that the agents do not need to know about what is being sampled, only how to make the data they can provide "presentable" and the loggers don't really need to know the details of a submodel's workings, only that it ought to obtain certain data from agents of particular types and generate an output file.

While this approach to data output engenders some extra overhead, it has a number of advantages which are not necessarily obvious. Consolidating the output into separate agents makes changes to the format of the generated output a simple matter of changing which output agent is used, or which format is required. The same mechanism, or one which is very similar, could be used as the basis for coupling distributed models.

Using <***logger***> agents rather than embedded output code also means that we can confine code associated with generating "output" (mainly the ***log-data*** methods) to a very small section of a few basic parent classes; thus instances of all the child classes can automatically be queried without the necessity of writing new output routines. The MOP of SCLOS is used to map calls from the introspection agent to the appropriate version of ***log-data***, and the chain of inheritance can be used to call the ***log-data*** methods of its parents to structure the output appropriately.

In many ways, this is a direct analogue of field sampling, and designing the mechanism in this way makes the direct simulation of field sampling quite straightforward.

Table 5.3: Introspection classes in Remodel– introspection-classes.scm

Classname	role
<***log-introspection***>	The fundamental mechanics that underpin the regular polling of other agents in the system
<***logger***>	provides the basic structure for generating model output
<***monitor***>	provides the machinery for assessing the state of the system and effecting changes in its configuration

The introspection classes are defined in the files introspection-classes.scm, monitor-classes.scm, and log-classes.scm. Of these, log-classes.scm is the most complex.

The monitor classes are able to maintain a history of the decisions they make and can, in principle, factor this history into the decisions they make subsequently. As yet, the <***monitor***> classes are still quite simple, acting to coerce changes in the constitution of the model ensemble according to fairly simple rules.

Agents with locations

The individual-based classes are all derived from a small ensemble of classes which endow them with the physical attributes we expect, such as location, and mass. All of the biotic agents in this branch of classes are able to maintain a history of their movement through the model's geographic domain and may also maintain a record of state variables. The agents may inspect these data and potentially use them in their own decision-making process; thus, some (as yet unimplemented) model of a animal may "recall" a recent encounter with a predator in a location and avoid the area.

Table 5.4: Basic individual-based representations – framework-classes.scm

Classname	role
<***tracked-agent***>	Agents derived from this class automatically maintain a time-stamped history of where they have been. This data can be extracted for analysis or plotting.
<***thing***>	This adds mass to a <***tracked-agent***>.
<***living-thing***>	This class adds age and mortality, also aware of the environmental context in which it lives.
<***plant***>	The general structure for plants is provide here. The more refined <***example-plant***> is derived from <***plant***>.
<***simple-animal***>	Instances of this class make use of a simple metabolism, simple modal behaviour
<***animal***>	extends the behaviour of simple animal by making the metabolism and modal behaviour more complex
<***example-animal***>	This class adds reproduction, density limits, and the ability to migrate when conditions warrant it. The classes used to represent the animals in Remodel example are derived from this class.

The classes for the simple organisms of Chapter 3 are defined in the files animal-classes.scm, plant-classes.scm, with corresponding "...-method.scm" files with the implementation of the methods.

Plants and animals are both implemented in classes derived from <***living-thing***>. An analytic representation for plants based on <***diffeq-system***> is also implemented.

There are essentially three implementations for modelling plants in the Remodel example. <***plant***> is the basic starting point for the <***example-plant***>, which implements a model substantially similar to that described in Section 3.3.4.1. The alternative representation for plants is the <***plant-array***> which is an array of state-vectors which apply update methods to each of the state variables which are similar to those used for the individual based version. <***plant-proxy***> is an interlocutor for a <***plant-array***>, though we have not employed it in this implementation.

The way we deal with animal classes is similar to that of the plant classes. Both the <***aherb***> and <***jherb***> can be represented by <***animal-array***> agents. We refrain from carrying the generalisation to include the <***carnivore***> class, since carnivores are unlikely to reach high population levels, and their participation in the system is more characteristically discrete.

Agents associated with areas

Most of the entities which are associated with an area rather than a specific location are members of the environmental classes defined in landscape-classes.scm. Exceptions include equation-based representations for populations which may be associated with particular <***environment***> agents.

Table 5.5: Non-spatial environments elements – landscape-classes.scm

Classname	**role**
<***ecoservice***>	Usually associated with a patch, this class provides data which can be inspected and modified by other agents. It is possible for the ecoservice to access data in other agents, and to have implicit growth or decay associated with its value. Instances are able to store the (time value) pairs in a history list.
<***population-system***>	This is a refined version of the <***ecoservice***> which primarily provides a simple "biomass" agent.

The classes in Table 5.5 are typically incorporated into environments associated with nominated areas, though they need not be. Both ecoservices and population systems can play the role of spatially agnostic agents which are pertinent for the whole model domain, such as external forcing conditions. In contrast the classes in Table 5.6 are explicitly associated with bounded regions currently represented by circles or arbitrary polygons, or sets of these bounded regions.

Table 5.6: Spatial environments – landscape-classes.scm

Classname	role
<***patch***>	The basic geographic region is implemented in <***patch***>. It has a bounded area, a list of ecoservices which are assumed to be uniformly available, a "notepad" which can be interrogated for state information, and it may also have "caretaker" routines defined in parameter files which get executed every time the patch gets a time step.
<***dynamic-patch***>	Patches in this class may have a system of differential equations which stand in the place of the simpler – possibly simplistic – dynamics possible using only the ecoservice mechanisms.
<***landscape***>	An instance of this class maintains a list of patches and a terrain function which provides an altitude ordinate to the locations in the geographic domain; in aquatic/-marine models, this would typically be used to provide the depth of water at a give point.

5.3.4 Parameterisation

Agents (instances of the above classes) are made by calling the ***create*** function. The principal tasks of ***create*** are to allocate storage, and to call the functions which initialise the state variables of an agent. ***create*** initialises its state variables using the data taken from the files in a nominated parameter directory. These parameter files contain Scheme code and they are loaded immediately after all of the code associated with implementing representations of submodels has been loaded, but before any model initialisation occurs.

The the names of the files in the parameter directory (in our case "./parameters/) directly associate the parameter sets to either a particular class, or with a particular taxon. When agents are created, the instantiation process first initialises all of the agent's state variables to the value <undefined>, which is used to identify uninitialised values. The next phase then applies any initialisations which may be found in the file associated with the <***agent***> class, namely ./parameters/<agent>. The process continues, from most general (the greatest grandparent class) to most specific, till all the relevant initialisations

found in the agent's class tree have been applied. The system then looks for a "taxon" file which which contains very specific initialisation data – we might have a generic class for oak trees – <***quercus***> – for example, which serves adequately for both the Q.robur and for the Q.petraea taxa, for the most part. The parameters which differentiate similar species are assigned after this with data taken from their taxon specific file, and these new data may overwrite more generic parameters. Finally any initialisation indications found in the ***(create ...)*** call used to instantiate the agent are applied. In this way, more specific parameters (like age, or metabolic rates) get applied last in the process.[11][12]

A typical parameter file might look like look like the '<example-plant>' file below:

```
'Parameters
(kdebug '(loading-parameters
          taxon-parameters "B.ex-longevity" "*.ex-*"))

(define B.ex-longevity (* 37 years))
(define B.ex-mass-max (* 150 kg))
(define B.ex-mass
  (rk4 (lambda (t y) ;; make the d.e.
(* 5e-7 (- 1 (/ y B.ex-mass-max)))) 0
B.ex-longevity (* 4 weeks) 0.1))
;; stepsize initial-val

(define <example-plant>-parameters
  (list (list 'cell '<uninitialized>)
  (list 'peak-mass 12)
  ;; this will seed trees in the
  ;; 4km x 4km square around
  ;; the origin
  (list 'location
  '($ (random-location
 (list -2000 -2000) (list 2000 2000))))
  (list 'mass
  '($ (nrnd 4 1 0.01 12)))
  (list 'height
  '($ (* kg (min 12 (+ 12 (nrnd 10))))))
  (list 'fruiting-mass (* 8 kg))
  (list 'fruiting-prob
  (/ 0.1 week)) ;; 10 percent per week
  (list 'fruiting-rate
  (* 7 (/ 1 kg))) ;; N / (kg week)
  (list 'mort-prob (/ 0.1 year))
```

[11] It doesn't matter if there are entries for things which are not state variables – any specification which isn't recognised is silently ignored.

[12] Within the code of Remodel, taxons are always represented by strings, and classes are always represented either directly by the class, or by a symbol which returns that class when it is evaluated by the interpreter.

```
  (list 'seeds-per-fruit '($ (+ 4 (urnd 8))))
  (list 'mass-at-age B.ex-mass)
  (list 'references
  '(J. Muppet Botany, S.Chef et al v2,1995))
  )
  )
(set! global-parameter-alist
(cons (cons <example-plant>
<example-plant>-parameters)
global-parameter-alist))
```

The 'Parameter line which is the first entry in the file is an indicator that this is a parameter file, and the example implementation uses this to determine that the file is a valid parameter file. The definition which follows is a list of lists where the first element of each of the second level lists must be a symbol. These symbols should (but are not required to) correspond to the state variables in the agent. Note that even though this file is associated with <***example-plant***>, it is not restricted to setting state variables defined in <***example-plant***> – it is *both reasonable and common* to set variables associated with parent classes, such as the plant's *location* in the example above.

The last three lines of the file add the definitions to the list of all the state variable defaults the system knows about – if they are missing or compromised, the definitions may not be accessible.

The parameter files from which the parameterisation is taken must be valid Scheme code, and as such can contain variable and function definitions. The example above includes the function definition for the mass-at-age function for the taxon "*B.exemplarii*". Naïvely we might assume that we would be able to perturb the starting mass of each individual tree by just incorporating a random number into the mass indicated in the parameter file: we can do so, but the parameter files are loaded early in the loading process, before any agents are actually instantiated. At the point we begin to create agents, the parameterisation data is static. In order to overcome this, the code which accesses parameters recognises that parameters of the form ($ ...) should be evaluated dynamically when they are accessed and passed back, so the line

```
(list 'height
   '($ (* kg (min 12 (+ 12 (nrnd 10))))),
```

will return a value greater than twelve, with a distribution of twelve added to the left half of a normally distributed variable with standard deviation of ten. Other examples can be found in the parameter files with carnivore, adult-herbivore, juvenile-herbivore and , B.exemplarii. It should be emphasised that the parameters obtained from the parameter files may be overridden in the ***create*** call, and by code within the model itself.

In general, units are specified in the parameter files, and should also be specified when indicating particular initialisations for agents. The (extensible) list of known units is found in the file units.scm.

5.3.5 Model initialisation

The first stage in a model run is the creation and initialisation of the components of the model. A model is constructed by a (comparatively) short Scheme program which uses the infrastructure to create and configure the agents which comprise the model ensemble. These agents are added to a list called the *runqueue*, which corresponds to the execution queue in a simple multitasking operating system. The agents are created by a call to a routine in sclos+extn.scm that looks like

```
(create <landscape> taxon 'name "domain" ...)
```

where the ellipsis indicates additional state variable assignments which supercede any previously set values from the parameter files. Agents which have an associated region or location would usually require some additional code to add agents to encompassing entities (such as including trees in the landscape) this would often be a part of the code which creates the agents[13], or would be effected in the class-method ***initialise-instance*** which is called as a part of the process that prepares an agent to be run; generally, initialisation of an agent would be accomplished using both of these approaches.

5.3.6 Methods, model bodies and closures

Model methods are essentially functions that are explicitly associated with the class signature of the arguments passed to them. For example, there are several versions of the model method ***dump***, and each is made distinct by the class of its arguments, particularly the first argument, and each of the methods may be invoked only by members of the appropriate class or of its descendants. This ability to have several functions with the same name is made possible by the use of instances of <***generic-methods***> which maintain an internal list of actual functions and information about the types of the arguments they expect. When a call to a method is made, it is the generic-method that receives it, and it examines the arguments passed in order to determine which of the model-methods should handle the call. The call is then forwarded to the appropriate actual implementation. This means that <***shark***> and <***seal***> may both have an ***eat*** method – moreover, <***shark***> may have two ***eat*** methods: one which recognises when the item being eaten is an <***animal***>, and one where it is merely an <***object***>, which would usually be inedible. This multiple-dispatch means that no explicit test is required in the model code. In this case it would be worth writing an <***object***> version which causes the agent to spit out the offending item. If there was only a version of the ***eat*** method which implemented eating <***fish***>, for <***shark***> agents, they will only attempt to eat fish, and presenting them with anything else will raise an error. This ability to catch and recover from unexpected arguments makes the extension of classes much less vulnerable to undetected error conditions.

[13] See the file specific-model.scm, or spawning code in animal-methods.scm for an illustration of this.

There are two golden rules to model-methods

Only define each generic method once: Redeclaring a generic-method or defineing a method (define ***hunt*** (***generic-method***)) destroys any previously declared ***hunt*** methods, generic or otherwise! To this end, all declarations of generic methods must be made in the framework-declarations.scm file, which should contain the declarations of the all generic methods used by the model, apart from a very special few declared in sclos+extn.scm. Related to this is the problem of defining a method before you have a generic-method declared – the macros in framework should catch this.

Don't construct methods that match the same argument lists: Generally, SCLOS is very good at getting the right method for the job, but there are situations where it can be confusing for it. If you have a situation where there are several possible desired code paths for essentially similar arguments, it is clearer to have an explicit selection made within the body of a single method, and that way is much easier to debug.

Model bodies are special methods which are only called by either the kernel or by the subsidiary execution lists embedded in agents. There is no provision for an agent to directly call model bodies. Each agent's model-body is run by the kernel at each of the agent's time steps; there is no object-body since objects are (by design) not able to be run by the kernel.

5.4 Execution and Control flow

After the initial cohort of agents have been instantiated and introduced into the runqueue (or in the *subsidiary-runqueue*'s of other agents), control is passed to the kernel which then begins to run each agent in turn for an appropriate time step. The queue is ordered first by the *subjective-times* of the agents, then their relative precedence, and finally an optional *jiggle* value which can be added to ensure that there are no systematic preferences within a time step.

The most important routines in maintaining an orderly flow of control from one agent to the next are

run-agent which manages the interactions between the agent and the *runqueue* and constructs the procedure that the agent uses to communicate with the kernel.

run calculates the upper limit on the amount of time the agent will run, then runs the body of the model (declared in a model-body block), next it runs any nested agents and finally attends to the maintenance of any data from superceded representations.

The sorted queue of agents is maintained by the system; the agent at the head of this list is removed from the list and execution is passed from a call to ***run-agent*** to ***run*** which then passes control to the agent's model-body. The call

chain is split in this way to separate interactions with the run queue and the management of the activities of agents. The motivation for this separation is that provides an avenue for changing the nature of the system used to run the submodels, say from a single-threaded execution list to a number of parallel lists.

The model-body of an agent takes control with an indicated maximum amount of time over which it might run. Agents need not run their whole time step – events may occur which cause them to truncate their turn, returning control to the kernel with an indication of the amount of time they actually used. When the kernel receives control from an agent, it examines the data passed to it by the agent and acts accordingly: terminated agents may be silently dropped from the queue, for example, and other agents may be reinserted in an appropriate place in the queue for the their next time step, or agents representing organisms which have just reproduced may introduce new agents into the runqueue.

Control also passes from agents to the kernel when the agent makes queries looking for prey or to interact with other agents: a carnivore might want to know whether there are any prey animals within a certain radius, for example. Here, a request would go to the kernel for a list of nearby prey. The kernel would then examine the list of agents meeting the requirements (type, spatial location, temporal contiguity) and pass the list – and control – back to the hunter. This is a common paradigm in both agent-based modelling and in computer-operating systems design, where access to information or resources are obtained from a delegated "authority".

The maintenance of state data from a superceded representation is rather different. When an agent is given data to maintain, it processes them at the end of each of its time steps. Each is asked what set of data their maintenance closure needs in the update step; this data is obtained and duly passed to the maintenance closure to use in its update processing. When the closure has finished, control returns to the agent, and ultimately passes back to the kernel.

5.5 Interaction with the kernel and other agents

5.5.1 Calls to the kernel

The first step in an interaction between agents is often to find other agents which fit particular criteria: a predator looks for nearby prey or a travelling salesman might look for accommodation within their budget, for example. Many of these searches are conducted either by using one of the ***locate*** calls provided by the kernel, or by taking advantage of previously cached information maintained by the searcher. The searches can be restricted to particular spatial regions, particular sets of taxa (as determined by the agent's *taxon*), particular classes, or the possession of particular state variables (such as *location* or *available-rooms*). The nature of the interactions may range from merely ascertaining the presence of another agent, to predation, or extracting state in-

formation (such as mass, location or the price of a room). A typical call to the kernel would look like

```
(set! target (kernel 'locate (apply *provides-*? (my 'requires))))
(set! vtargets (kernel 'providers? (my 'requires)))
```

Both *targets* and *vtargets* should have the same values. There are a number of predicates defined in slos+extn.scm which are similar to ***provides-*?** which select on class, taxon, location, possession of particular state variables, the roles an agent may play, or combinations there-of.

5.5.2 Spatial queries

Agents that regularly interact with other agents which are close to them in the spatial domain will often cache references to their important neighbours to reduce the overhead of calls to the kernel routines, since direct communication with target agents has a much lower overhead. Direct contact does require agents to check to see whether the agent being queried is active or has been replaced by another representation, since both state and representation changes can occur without the agent knowing. Most state changes of an agent will be from alive to dead or terminated: the first two indicate that the agent is still available for interaction, the third indicates that it is no longer a participant in the system. Should the object of an agent's interest become terminated, the agent can then query the terminated agent to see if it has been replaced by another representation; if so, it may be able to obtain a reference to the new instance directly from the superceded agent, or query the kernel for the reference.

Scheme only reclaims objects (in this case instances of a submodel) when all references to the object have vanished; this means that as long as an agent maintains a reference to the entities it is interested in, those entities will persist, and can be used to obtain a reference to instances which have replaced it.

Agents representing animals or plants will typically already know about their "domain" – the landscape they inhabit. Landscapes are usually cast as bounded geographic regions associated with environmental data such as water availability, topography or contaminant levels. In contrast, an animal would typically *not* keep track of other animals. The agents in Remodel only cache a limited amount of information in this regard.

The basic calls which return the locations associated with agents look like

```
(let* ((prey (kernel 'locate
 (apply *provides-*? (my 'food-list))))
 (cover  (look-for self (my 'prey-hides)) )
 ;; another way of querying
 ;; plants that can hide my prey
 (preylocations
   (map (lambda (x)
```

```
    (query x self 'location
    (my 'search-radius)))
    (append prey cover)))
 )
 ;; body of processing code using prey, cover and
 ;; preylocations
 ...
)
```

In this example, the first of the three clauses obtains a list of candidate prey with a call to the kernel[14] The next searches for potential hiding places with the ***look-for**** call. This call does not enforce synchrony[15], and the last actually extracts targets' locations for subsequent use.

Once an agent is aware of another agent, it will typically interact using either the ***query*** call or calls to the methods appropriate to the target. [16]

5.6 Introspection agents: loggers and monitors

Introspection classes are primarily structured to periodically pass through a list of running agents. Instances of these classes have the ability to glean data from the agents they query, cause the agents to change their behaviour, and even remove or replace agents. The classes with these special properties are all derived from the general <***introspection***> class. The most important role they play is in the generation of the model output, but they are also responsible for the ability of the implementation to adaptively change the representations used in is simulation of a system.

5.6.1 Generating output files – loggers

The <***log***> agents do their work by first generating a list of agents to be polled, and then calling a routine to "emit a page", which really just means that it should emit the data which is appropriate for this iteration. This task has three phases: first it calls the method ***page-preamble*** which prepares state variables for processing the appropriate sort of assessment (agent-level, taxon-level niche-level, or configuration-wide), then it iterates through the list of targets provided by the kernel apply the ***log-data*** method, and finally it calls ***page-epilogue*** to finish generating the output. It should be noted that *only* <***log***> has an ***emit-page*** method – this method works for all loggers, since they are derived from <***log***> and the class-specific specific details are devolved to the children.

[14] In this case locate returns all possible candidates irrespective of location, but insists that the targets are contemporaneous. locate*, in contrast, does not enforce temporal consistency. Both are able to filter the results based on location, class or the role targets play in the simulation.

[15] The alternative ***look-for*** does enforce synchrony.

[16] The MOP paradigm makes it possible to use substantially the same code for many agents of many classes, but calls through ***query*** may be easier to convert for parallelism later.

The method ***log-data*** is used because the child classes often have significant differences in their state variables, their internal data structures and output formats. Dividing the processing in this way means that data can be accumulated in an orderly way and either written or cached until it is suitable for writing. Each of the classes that can be logged must either use the <***log***> version of default ***log-data*** (associated with one of their parents classes) or supply its own. Some of the leaf classes produce some of their output in a ***log-data*** method associated with their class, but chain back to the parents ***log-data*** methods to generate the rest of their output.

As mentioned before, the decoupling of the subsidiary models and the generation of their output is not without some overhead, but the net gain – at least in this example implementation – has been worthwhile. Not only is starting the logging of data after the model's "spin-up" simple, but the loggers can be configured to replicate different field studies without requiring modifications to the underlying model.

5.6.2 Changing representations

Monitors are very similar to the logging agents and use essentially the same infrastructure for agent selection and processing. The process of assessment and the management of the representations of the systems being modelled is reasonably simple to describe: periodically the monitoring agent will interrogate the subset of the model for which it is responsible; each time it does so, it generates a tree or set of trees which encode the state of the model and the system's assessment of their merit with respect to the location of the model (or components) in the state space. These periodic assessments are essentially "maps" of the current configuration of the model into the metric-space of the trees of Chapter 4. We use this mapping to calculate the closest candidate configuration from a set of configurations we believe (or know) to represent the system well in a given part of its state space. Because the trees are elements of a ring, we can, in some cases, interpolate between configurations, arriving at possibly advantageous configurations which have not been explicitly specified.

There is an overhead to using monitors to adjust the configuration: like loggers, they must run periodically and poll agents. While a branch to an alternative processing path within a submodel is undoubtedly a lower cost, the tests attendant to both approaches will be substantially similar and, by shifting the business of changing representation to an appropriate monitor, all representations for an entity automatically become capable of making that change without altering their code. We also gain the benefit of greater control over how frequently tests and changes of representation occur without muddling the code which comprises the various representations.

The processing loop for a <***monitor***> agent has a number of steps:

— Each of the agents will be queried for its status and its own assessment of its state in the context of the other elements it interacts with. This may

be fairly trivial for most agents, particularly those which change slowly, or are not significantly influenced by the activities of other agents. In the majority of cases, the agents will return a list with a self-assessment of their current levels of fidelity and cost.

— The monitor synthesises a number of representative state-trees which describe the status of the configuration of agents by taxon and niche, and it generates a state-tree which describes the current configuration. Other factors dealing with significant conditions which pertain to the simulation – perhaps relating to the geographic distributions, population numbers or density, the abundance of resources, or the presence of significant conditions such as fire, may be incorporated into the configuration trees and the candidate configuration trees.

— The monitor then compares the current state against the *known-good* and *known-bad* configurations, and records these values. Each of the generated status trees is then compared with appropriate *good* and *bad* trees. The results of the comparisons are used to select the best candidate configurations and representations for consideration. The trees that represent the model and its constituents are constructed so that we can add trees that represent agents or groupings of agents to construct a tree that represents a model or component. Each of these synthetic trees is ranked and, if there is enough benefit, the desirability of a change is indicated.

The assessment trees are then used to direct the conversion of groups of agents to other representations. Sometimes this will entail the collapsing of a number of individual-based agents into a super-individual, an equation-based representations or possibly a hybrid of some sort. Other times, a more aggregated entity may be converted to a more discrete form. The strategies for conversion depend on the nature of the representations in question, and this is probably the most challenging part of constructing a modelling system of this kind – in order to maximise its utility, the conversions should be carefully considered. One of the benefits of this type of examination is that the sensitivities of the model must be explicitly analysed and dealt with.

Developing a body of configurations which function well is likely to be, at least initially, a process of intuitive selection and trial and error. Initially, we begin with coarse knowledge regarding the utility of particular representations under given conditions. <***loggers***> can collect and record information about the agents within the simulation, and their performance. This data can be used as the basis for the selection and testing of new configurations, and we can classify them into varying degrees of "good" and "bad" configurations.

This process depends on being able to discriminate between configurations, and we do this by mapping configurations into state-trees which are constructed in a way which encodes coarse spatial information (the relationship between *patches*, for example), the numbers of entities represented by different classes, and their own performance measures.

The replacement of an agent by another is accomplished by a routine using the ***create*** function as used in the specific-model.scm file. There is an issue associ-

ated with the risk of exhausting the available memory, however: if any memory reference in an active memory allocation refers to a scheme object then the object is protected from garbage-collection. This means that update-closures which are created with a built-in reference to the creating agent protect that agent from having its memory returned to the pool.[17]

Maintaining state across representations

When an entity is being prepared to change its representation, its current representation may create an encapsulation of its critical state variables in a closure that is bound to a function which is able to access these variables, and to maintain their values. The closure may ultimately pass these values back to another agent which will take over the role played by the model maintenance function.

A typical closure definition might look like

```
(update-closure <plant>-B.exemplarii:14
  '(age mass taxon location)
  (list ;; initialise with these values
    (slot-ref self 'age)
    (slot-ref self 'mass)
    (slot-ref self 'taxon)
    (slot-ref self 'location))
  (list ;;
    (lambda (t dt a m t l) (set! age (+ age dt)))
      (lambda (t dt a m t l) (set! mass (mass-at-age (+ age dt))))
      #f
      #f))
```

In this example the state-variables *age* and *mass* are updated in the body. Closures are (at least in this model) unable to access anything other than the arguments passed to them by the model responsible for maintaining them. The ***update-closure*** block is implemented as a macro that is turned into a functional closure with internal state-variables. Calls to a maintenance closure passes a list of expected values to the closure in the expected order, and the current values for time and the length of the time step (*t* and *dt*) are always passed in by the system. The maintenance closure can be passed a single symbolic value – quit – which indicates that it should terminate and return the current value of the variables it maintains.

[17] This sort of issue is easy to recognise – the computer rapidly becomes sluggish or almost completely unresponsive because it is spending all its time in swap space.

5.6.3 Comparison of states

The first step in being able to quantitatively compare configurations of models is to construct a means for encoding the state in a way that supports some sort of qualitative or quantitative comparison. The trees described briefly in Chapter 3 and more comprehensively in Chapter 4 are elements of a metric space. Since the set of indeterminate variables in the labels of these trees is quite arbitrary, we can identify any particular attributes of interest in a family of representations with indeterminates, and we can also extend this to denoting the various representations for the modelled entities.

The framework maps the configuration of the model to a tree in the metric space defined in Chapter 4 as a means of allowing us to calculate how close one configuration is to another. The example tree (below) consists of only a few agents, but it illustrates the type of structure and information used in determining the relative merits of different configurations. Only three taxa are represented in this tree – *fruit*, *seeds* and *B.exemplarii*, and only the plants are represented by more than one type of agent. The tree includes in "leafmost" nodes the identifying information that ties specific agents to particular branches of the tree. In this way, we can apply a trim operation (Def. 4.2.5) and calculate the distance between representations without the details of specific members of the aggregate data interfering in the result.

In this example, the numbers in the "weight" position, after the colon, refer either to the aggregate values (in the case of equation-based ecoservices), the numbers of distinct agents, or the serial numbers of the specific agents in leafmost nodes. A number of useful indicators can be constructed from the simple operations defined at the beginning of Chapter 4.

```
(taxon : 0 {
  (fruit : 2792746.800173795
    {(fruit + eqn-based + <ecoservice> : 625055.9572307098
      {(gridcell-1,2:fruit : 21  {})})
     (fruit + eqn-based + <ecoservice> : 132324.34236008758
      {(gridcell-1,1:fruit : 19  {})})
    (fruit + eqn-based + <ecoservice> : 362001.18511443434
     {(gridcell-1,0:fruit : 17  {})})
      (fruit + eqn-based + <ecoservice> : 696015.3639367699
    {(gridcell-0,2:fruit : 15  {})})
     (fruit + eqn-based + <ecoservice> : 462154.55908067356
    {(gridcell-0,1:fruit : 13  {})})
     (fruit + eqn-based + <ecoservice> : 515195.3924511202
    {(gridcell-0,0:fruit : 11  {})})
     }
  )
  (seeds : 4950004.701824954
    {(seeds + eqn-based + <ecoservice> : 825000.4339876496
      {(gridcell-1,2:seeds : 20  {})})
       (seeds + eqn-based + <ecoservice> : 825000.6790376918
```

```
      {(gridcell-1,1:seeds : 18  {})})
       (seeds + eqn-based + <ecoservice> : 824999.6759855568
      {(gridcell-1,0:seeds : 16  {})})
       (seeds + eqn-based + <ecoservice> : 825001.7606296955
      {(gridcell-0,2:seeds : 14  {})})
       (seeds + eqn-based + <ecoservice> : 825000.96499662
      {(gridcell-0,1:seeds : 12  {})})
       (seeds + eqn-based + <ecoservice> : 825001.1871877399
      {(gridcell-0,0:seeds : 10  {})}
       )}
  )
  (B.exemplarii : 6
    {(super-individual + B.exemplarii + <plant-array> : 3
      {(B.exemplarii_1 : 6  {})})
     (individual + B.exemplarii + <example-plant> : 1
      {(B.exemplarii_4 : 9  {})})
     (individual + B.exemplarii + <example-plant> : 1
      {(B.exemplarii_3 : 8  {})})
     (individual + B.exemplarii + <example-plant> : 1
      {(B.exemplarii_2 : 7  {})})
    }
  )}
)
```

Analogous trees can be constructed which can be used in a similar way to match spatial patterns. We do this by identifying the neighbours of a region with symbolic variables in the labels of nodes; a very simple example using grid cells might look like so:

```
(taxon : 0 {
  (gridcell-1,1 : 1044
   {(gridcell-2,1 : 37 {})}
   {(gridcell-2,2 : 637 {})}
   {(gridcell-1,2 : 937 {})}
  )

  (gridcell-2,1 : 37
   {(gridcell-1,1 : 1044 {})}
   {(gridcell-2,2 : 637 {})}
   {(gridcell-1,2 : 937 {})}
  )

  (gridcell-1,2 : 1044
   {(gridcell-2,1 : 37 {})}
   {(gridcell-2,2 : 637 {})}
   {(gridcell-1,1 : 1044 {})}
  )
```

```
  (gridcell-2,2 : 637
   {(gridcell-2,1 : 37 {})}
   {(gridcell-1,1 : 1044 {})}
   {(gridcell-1,2 : 937 {})}
  )
 }
)
```

Like the status trees in the first example, these trees can be compared to reference patterns to identify spatial distributions which require the intervention of a monitor. An example of this type of situation might occur in a model of suburban housing: if large numbers of residents are abandoning one particular region, there may be effects on the surrounding regions – increase in civil disorder, for example – which are not accounted for by the representations for those areas.

Monitors can be constructed to maintain a list of status trees which track changes in the ensemble through time. At the moment, this would be a computationally expensive process, but as the implementation of the underlying mathematical machinery improves, the notion of using this kind of higher order analysis of the dynamics in the system becomes more tenable.

5.7 Future work

5.7.1 Distributed models

The top-level <***introspection***> class may ultimately form the basis for connecting distributed models, a promising library for this purpose is ∅MQ ("zeroMQ" in Corp. [2012]). No work has yet been done to directly support distributed execution, the approach taken to address these issues has been structured to make this type of connection feasible and as straightforward as possible.

5.7.2 Cross-representation interactions

A model which couples elements which are discrete entities (*e.g.* cattle) with others that have some type of continuous representation (*e.g.* a vegetated area) must deal carefully with the way the interactions are played out. Although interactions that arise are not limited to predation, predation will be used as an example that illustrates a way of playing out an interaction between different representations.

Models at the "individual-based" end of the spectrum can be very sensitive to fine changes in parameterisations or gradients within the model. In some contexts, this is exactly what we wish for, but in others it can make tuning the model difficult and there may be underlying structural factors that have

undue influence in the model's trajectory.[18] Also, processes like predation are computationally expensive when large numbers of prey are present. It is possible for us to model predation between individual-based and population-based representations, but to do so requires an approach quite unlike the approaches of conventional individual-based models, where the pursuit, capture and consumption may be explicitly simulated, or the approaches of conventional age-class structured population models. The work outlined in this section provides a reasonably efficient example of how couplings between individual-based and analytic components may be accomplished.

In this example, we will consider a set of individual-based predators preying on a size structured population. Here, there is a limitation on the size of prey which is available imposed by the gape of the predator. This type of constraint is not all that uncommon in species which are lone predators.

We can map population histograms (where x-axis corresponds to the 'size' of a fish) onto piece-wise linear functions with the property that the partial integrals of the linear functions correspond to the partial sums of histograms at each boundary in the histogram. If populations are functions, then we can evaluate the consequences on the population.

Let us consider the following "known" attributes in a system

Table 5.7: Symbols

$m_a(l)$	the member distribution with respect to size of each agent of interest
$G_{i,j}(l,w)$	the gape filter for predator i with respect to prey j
$T(a)$	returns the taxon or type number of agent a
M_a	the total number of members for agent a
$M_i^*(w)$	the sum of all the distributions of agents with a type i
$\bar{M}_i = \int_0^\infty M_i^*(w)dw$	the total number of entities of type i

Calculating mortality

Let

$$I_{i,j}(l,w) = M_i^*(l)G_{i,j}(l,w)$$

and

$$J_{i,j}(l,w) = M_j^*(w)G_{i,j}(l,w).$$

$I_{i,j}$ is the raw distribution of pressure of predator i onto prey j, and $J_{i,j}$ is the raw distribution of the vulnerability of prey j to the predator i.

[18] In an unpublished model I wrote in the very early 1990's, simulated fish lined up neatly on boundaries between the pixels of the digital elevation map that made up the seabed – they maximised their access to prey by straddling the different domains.

If the constants

$$k_{i,j} = \int_0^\infty \int_0^\infty I_{i,j}(l,w)dldw$$

and

$$h_{i,j} = \int_0^\infty \int_0^\infty J_{i,j}(l,w)dldw$$

are non-zero, they can be used to scale $I_{i,j}$ and $J_{i,j}$ so that they form kernel functions, and we get

$$K_{i,j}(w) = \frac{1}{k_{i,j}} \int_0^\infty I_{i,j}(l,w)dl$$

which is the normalised predatory pressure with respect to size, and the normalised vulnerability

$$H_{i,j}(w) = \frac{1}{h_{i,j}} \int_0^\infty J_{i,j}(l,w)dl.$$

Values of zero in $k_{i,j}$ and $h_{i,j}$ indicate that no predation is possible – usually because M^* has collapsed. In this case we take either (or both) $K_{i,j}(w)$ and $H_{i,j}(w)$ to be zero.

We calculate

$$v_{i,j} = \int_0^\infty K_{i,j}(w)H_{i,j}(w)dw$$

which has a value in the range $[0,1]$. If $v_{i,j}$ is non-zero we can construct the normalised interaction

$$V_{i,j}(w) = \frac{1}{v_{i,j}} K_{i,j}(w)H_{i,j}(w)$$

which indicates the proportion by size of type j subjected to predation from type i at the given length w. Again a zero value for $v_{i,j}$ indicates that no interaction ("diner" or "dinner") are possible.

The function

$$e_{i,j}(w) = M_j^*(w)V_{i,j}(w)$$

can be used to give us the number

$$E_{i,j} = \int_0^\infty M_j^*(w)V_{i,j}(w)dw$$

which is the exposure of prey population j to the predators in population i. The converse,

$$C_{i,j} = \int_0^\infty M_i^*(w)V_{i,j}(w)dw,$$

is the potential for predation of the predator type i on an "average" prey of type j.

We can then use a predation relationship of some sort to get the raw number of "kills" based on the exposure averaged over the potential volume (or area) of contact per unit of time, which we call $\Omega_{i,j}$, where $\Omega_{i,j} = \bar{M}_i\mathfrak{F}(E_{i,j}/A_j, p_{i,j})\Delta t$

where $\mathfrak{F}$ is the predation relationship, and $p_{i,j}$ is the parameterisation for the species, and Δt is the time step, and A_j is the area/volume we divide by to get a density.

We can sum over a predator type

$$\Omega_j^* = \sum_i \Omega_{i,j}$$

to give us the total possible consumption of prey type j.

Alternatively, we can calculate a consumption-by-size distribution and define the function $\omega_{i,j}(w)$, the raw number of kills for a length w on a consumption-by-size basis, by

$$\omega_{i,j}(w) = \bar{M}_i \mathfrak{F}(e_{i,j}(w)/A_j, p_{i,j})\Delta t.$$

Thus the impact on the prey population (at least those of length w) is

$$\omega_j^*(w) = \sum_i \omega_{i,j}(w)$$

and for the whole population it is

$$\int_0^\infty \omega_j^*(w)dw$$

(or something like that).

Alternatively, we can express things more in the way that it is calculated in the Atlantis model [Fulton, 2011] with

$$Z_i(w) = \frac{g_i M_i^*(w)}{g_i/c_i + \sum_j a_{i,j} e_{i,j}(w)}$$

where $Z_i(w)$ reflects the aggregate clearance rate of a predator of type i if we take c_i to be the "clearance rate" which incorporates the volume it sweeps and a proportion of prey captured and we take g_i to be the predator's growth rate.

So $\int_0^\infty Z_i(w)e_{i,j}(w)\Delta t dw$ is the amount of prey of type j consumed by the predators of type i over the interval Δt.

It should be pointed out that the number of types is fairly small compared to the number of agents, and this shouldn't be too onerous a calculation (at least compared to playing it all out individually).

Apportioning mortality

Mortality can be calculated either by apportioning it to each agent according to the proportion of the global population it represents (and within it, apportioning the mortality to ages in an analogous fashion), or we can apportion mortality to each age in each agent according to how much of the population it represents.

For the agent-by-agent update we have

$$\delta m_a(w) = \Omega^*_{T(a)} \frac{m_a(w)}{M^*_{T(a)}(w)}$$

and for the age-by-age update we have

$$\delta m_a(w) = \omega^*_{T(a)}(w) \frac{m_a(w)}{M^*_{T(a)}(w)}.$$

The new distribution

$$n_a(w) = m_a(w) - \delta m_a(w).$$

In these steps, places where there are no members of size w should be dealt with carefully in the division, and at all points $M^*_{T(a)}(w) \geqslant m_a(w)$.

5.8 Observations

Conveniences

A number of features of the model have proved to be – unexpectedly – very useful during the construction of the example described in this chapter. Particularly noteworthy items are the parameterisation mechanism, output generation, the instrumentation to collect the elapsed time within routines, and the code for debugging output.

Parameter files are small snippets of Scheme code which adhere to a particular form. In these files, data are associated with symbols which get used to determine what state variables are initialised to. There is a mechanism which supports the delayed evaluation of the value to initialise a state variable allows the modeller to perturb values (such as mass) in the instantiation of the agents. Similarly, we are able to move functions that are traditionally coded as *part of the model* into the parameter space – thus, we could easily write one basic Galapagos finch model and parameterise the many different behaviours and food preferences.

All of the output in the model arise from one of two basic mechanisms: debugging/warning messages emitted by the ***kdebug*** routines which were briefly discussed in Section 5.3.4 or output generated by a <***logger***> . There have been a number of benefits from not incorporating the output within the submodels. The most significant benefit is that it makes the task of producing "exotic" output, such as postscript or animations, much more straightforward, and the code to produce the output is much less likely to interfere with the model itself, since the output routines are quite distinct from the submodels they may query. With this approach, an output only needs to know is how to obtain data from an agent, and how to generate appropriate output (see Figures A.1, A.2 and A.3 in the appendix). In this way, the ***log-data*** routines are able to be applied to a number of quite different models, and their output is consistent irrespective of the underlying model generating the data.

The debugging messages provided by the ***kdebug*** call and its kin can be constructed to emit messages according to arbitrary flags which can consist of symbols, can contain wildcards, class-names, or parameter taxa. The same mechanism can conditionally execute more complex debugging or validation code, and virtually vanishes when the no messages are required. This has proved very useful both in debugging code, and in the parameterisation of models.

A small amount of effort has been devoted to improving the efficiency of the modelling framework. The primary aim of Remodel is currently not so much the implementation of production models, but as a tool for exploration. That said, its early versions ran slowly enough, that instrumentation to time procedure-level and block level code was added. Serendipitously, this code neatly meets anticipated needs to assess the execution speed for the purposes of adjusting configurations.

Using this, computationally greedy code can be located by using the alternative syntaxes define%, model-method%, and model-body% which extend define, model-method, and model-body by adding instrumentation which allows the modeller to determine how much time was actually spent in the functions, methods or model-bodies, with a call to `(timing-report)'`. This make the task of finding the bottlenecks in the code much simpler. Access to the data for assessment can be accessed by calling the function ***elapsed-time*** with the symbol specified as the tag in a ***timing-block***, and these functions are defined in the file timer.scm.

—

The code for the example described is under continuing development. The submodels which comprise it are deliberately simple in order to maximise the ability to track the dynamics arising from model switching. The models are comprehensive enough that comparisons between "unimodal" runs can be made and the parameterisations for the different representations can be made as compatible as possible. The model in Chapter 2 was, in many ways, an ideal platform for initial exploration . . . except that exploring much further than its ambit proved unfruitful. The essential limitation posed by the lack of a bidirectional interaction suggested that a model capable of predator-prey interactions was needed.

Early implementation decisions were based on experience with the models in Lyne et al. [1994a], Gray et al. [2006] and Gray et al. [2014], along with the general notions formulated in the development of Chapter 2 and Chapter 3. The class structure is readily extended, and the ability to use temporary instances of other agents as intermediaries – such as may be found in <***plant-array***>:***log-data,*** where a temporary plant is used to sequentially hold the essential data to pass to the tree plotting routine – makes the process of constructing new representations much easier than expected.

Current efforts are focused on making the framework more robust and simpler to use, particularly with respect to maintenance closures. While constructing

closures to maintain variables is straight-forward in Scheme, a means of implementation that did not rely on a reasonable fluency in the language would be better. Similarly, additional support routines for crafting <***monitor***> agents would help. Reference trees must currently be constructed by hand, and this is a somewhat tedious process. n

CHAPTER 6

Conclusion

There is a trend for ecosystem models to become more detailed and to cover larger domains, which can be viewed as the natural consequence of a growing desire for realism; in fact, this realism is often essential (Fulton et al., 2003; Rose et al., 2010). It is no great leap to suspect that the situation is similar in other fields of research. There are many other reasons why our expectations of models are growing, not least of which is our desire to understand and be able to predict the dynamics of physical and ecological systems we depend on. These expectations raise serious issues for the researchers involved with modelling: as the scope of a models domain increases, the computational requirements of the model typically increase more rapidly adding new species or doubling the area may increase the number of interactions by orders of magnitude, and the number of connections between components in models can grow much faster than simply a multiple of the number of components.

There are several major consequences of this growth in model size and complexity, the most obvious is that it takes longer to bring a new model to the point of usefulness. Not only does it take longer to assemble the model, but it also take a great deal more effort to test the model due to the increase in the number of possible connections between components. Stand-alone models may have radically different approaches to the treatment of the physical environment and coupling these can create regions where the usual methods do not work well. The demand for realism and the application of models in important processes, such as deciding how to manage industry in sensitive habitats, drives a process of model extension which is often accompanied by hard reporting dates. The combination of these factors puts run-time at a premium, since at least in ensemble models many simulations need to run with time left for analysis and writing. Clearly more memory, faster hardware or more efficient algorithms may help extend the limits on what a model may do, but it isnt always the case that a more resolved model will improve the correspondence between model results and the trajectory of a real system [**?**]Fulton et al., 2003; Sperber and Palmer, 1996).

The situation is not inherently intractable; a number of modelling approaches are available to us and they all bring particular strengths which we can exploit.

The work in Chapter 2 used two quite different approaches to the same problem, and each had advantages over the other. It also demonstrated that there was no substantive reason to avoid using both representations concurrently to maximise our advantage.

This work attempts to construct a theoretical context to aid the development and exploration of models composed of many interacting subsystems. The metric-space developed in Chapter 4 can be used in constructing an encoding which can

- represent a models state,
- be used to assess that state relative to known landmark needs,
- provide a vector space in which to search for appropriate alternatives,

should the need arise.

The formal structure which casts each component in the model as a distinct entity which communicates with other participants through well-defined and consistent interfaces is conventional practice. In the context of ecosystem modelling, this approach makes the inclusion or coupling of new systems such as new trophic levels or land forms much more straightforward.

Where the formal framework described differs most significantly is the inclusion of both self-assessment and supervisory assessment by the <monitor> agents. While the monitoring agents play a special role, in that they may force changes in the representation of other agents, they may also be subject to the assessment-and-replacement paradigm which applies to agents which simulate the system a model seeks to simulate. This meta-structure means that not only is it possible to automatically change the composition of a running model in terms of how its component parts are represented, but we can adjust our assessment and configuration mechanisms in the same way.

Structures that allow a flexible representation for components can help to reduce the overall complexity of a model as a whole by making each of its active components simpler.

Models like Gray et al. [2006] and Gray et al. [2014], have handled major changes in behaviour or niche by examining either their own state, their immediate neighbours or other external state data. This approach comes at a cost, however: as more and more exceptions occur, problems begin to arise:

- more time is expended on tests to see which path to take
- the number of possible trajectories through the code-path increase dramatically, making adequate testing tedious or difficult;
- there is a risk that complex sets exceptions which ought to be exclusive fail to be adequately guarded;

and

– we eventually reach a point where it becomes difficult for a modeller that

has not been involved in the evolution of the model to understand the dynamics of the system.

Branching is a plausible solution, but it doesnt offer compelling advantages over a representation change.

Experience suggests that as model size increases, the modellers ability to maintain consistency across the the code which handles the exceptions decreases. Failing to maintain the exceptions while the mainline code evolves presents silent risks to the integrity of the model as a whole. Adding insult to injury, the clauses which test which path should be taken may be executed frequently with no effect other than to slow the model.

The dividend that changing representation pays is a dramatic increase in the flexibility of the model: all the representations of an entity are automatically able to cede their place to a more appropriate representation, should the need arise, because the changes in representation are effected by another agent - a <monitor> - which is responsible for the assessment and change. The specific advantage provided by model switching over branching is that we not only avoid unnecessary condition testing, but we can devolve the whole process of assessment and switching onto a monitoring agent which can test only when needed.

The multiple inheritance provided by TinyCLOS makes it possible to construct variants of a model which may possess largely identical code-bases but exhibit different behaviours[1]. Each of these representations can typically proceed as though no other representation exists.

The model used as an example in Chapter 2 provided a simple demonstration that a model can modify the nature of the representations of processes, such as whether to represent a population analytically or as a group of migrating animals, in order to maintain the fidelity of model outcomes while improving computational efficiency. The explicit assumption in Chapter 2 was that an individual-based representation for the organisms would be taken to be the gold-standard for modelling contaminant uptake. Given this, the example runs showed that the adaptive version of the model was roughly four times as efficient in terms of computation cost. There was also a noticable difference between the results using the analytic representation for the population and the individually resolved representation; in the absence of a corresponding physical experiment, it would be somewhat presumptuous to press the claim that the individual-based and mutating models were more consistent with the real dynamics of a physical analogue of the system, however tempting that claim may be. In it, the change from one representation to another was simple, and the maintenance of the state variable which was to be preserved across transitions required no additional datain most cases a prohibitively constraining assumption, but even in such a simple model, the benefit of using the technique was clear.

[1] This property would also be true if Remodel had been written in C++, or any other language with multiple inheritance.

In ecosystem modelling, there are many circumstances where a group of individuals can be treated as homogeneous on the surface, it seems unlikely that one would ever need to discriminate between blowflies or sardines in a school, for example. Unfortunately, it is easy imagine scenarios where the uniqueness of an individual does become important. Contact with behaviour modifying parasites can make an organism less mobile and thus vulnerable to predation by the hosts that the parasite breeds in, thus providing a positive feedback in the system. In such cases, the individual characteristics, such as resistance to parasites, resistance to contaminants, the ability to store fat and the animals life-history, have an influence on what happens to the population as a whole[2]. Chapter 2 demonstrates that we can efficiently represent a system using sub-models with vastly different assumptions about individuality without necessarily compromising the quality of our results.

The model examined in Chapter 2 was, in a sense, a minimal construction: at slightly more than a thousand executable lines, it was sufficient to accomplish its intent to demonstrate that a strategy using adaptive representations can be significantly advantageous, and that state could be efficiently maintained across representations but the example cannot take the argument much further than that. In order to explore the implementation issues more fully and to be able to make informed judgements on where the technique is most applicable, a more general adaptive hybrid model with the capacity to deal with a broader range of situations was necessary. The description of a model in Chapter 3 described a baseline scenario that this more general model should be able to implement. The model has dynamics which create interdependencies amongst the biotic elements, there is a spatial heterogeneity, and the potential for migration within the region. Chapter 3 extended the notions discussed in Chapter 2 to include the possibility of dynamically determined (rather than hard-coded) strategies for changes in the representation of model components. This end has not yet attainable, but the example implementation, Remodel, has been constructed with this endpoint in mind, and development along these lines is continuing. Remodel is still under active development many of the desired outcomes have been achieved, but moving it from a pedagogic tool to a basis for real model development is the dominant long term objective. This modelling approach has application in a number of situations where conventional models may fail to perform adequately, whether in terms of process-fidelity, computational cost, or flexibility:

> Systems with dynamics which oscillate between levels where the influence of individuals have a significant effect on the outcome, and levels at which the mixing assumption holds;
>
> Real-time situations where execution speed and fidelity must be traded against each other, such as in virtual environments, online commerce, and computer games;
>
> Simulating complex systems with numerous tipping points;

[2]The evolution of species might be constructive evidence that individual variability is an important feature in long-term projections.

Autonomous control systems that operate in complex or unpredictable environments.

The ability to assess a models configuration and performance in arbitrary (measurable) terms, such as fidelity, computational efficiency and run-time, gives modellers the scope to construct systems which readily adapt to a wide range of use-cases: adjusting the sets of known-good and known-bad configurations could allow us to use the same basic model for long-term forecasting and for time-critical predictions which may be needed for emergency response decisions.

The conceptual strategy for constructing models described in this work addresses the problems which arise when dealing with complex systems whose components exhibit dynamics which are sensitive to the state of the system. Models can be constructed so that, as a simulation progresses, the representation of their component parts can adapt to the most appropriate form for the state of the model. In practice, it is possible to choose a strategy which optimises for any appropriate measure speed, fidelity, or the cost of real-world management, for example.

6.1 Into the future

This approach to modelling seems likely to be able to map readily into parallelised or distributed systems. Since it inherently encapsulates the data associated with submodels and provides mechanisms which control the communication between agents, a parallelising kernel should be reasonably straightforward. As a second stage, distributed processing is also likely to be a tractable problem; serialising the data structures is simple, and the whole system is geared to permit the dynamic introduction and removal of agents.

`call-with-current-continuation` is, like wizards, subtle but quick to anger. Also like wizards, it can be very useful when it comes to solving knotty problems. A class (or set of classes) implementing backtracking is planned, and it is likely to use `call/cc`. Backtracking is a method of attacking constraint solving problems, such as arise when hunting "optimal" model mixes, or parameter sets that cause a model to track a given set of data most accurately.

While these are the applications that spring to mind, a search for optimal solutions to constrained problems arises in many situations, and the ability to apply the technique both within a submodel and in managing the framework has great appeal.

Data assimilation and ensemble modelling are significant techniques which im prove our ability to construct realistic representations of systems which interest us. The use of a model-swapping approach in could be an attractive addition in these contexts - container models which implicitly apply techniques from these paradigms to subsidiary representations of a system could be readily implemented within Remodel. Moreover, a model-swapping approach could readily be employed at a number of levels, namely

– as the implementation paradigm for member models,
– as a mechanism for adjusting a mix of the models which constitute the ensemble, or

– at a macro assessment level, integrating the results of ensembles.

How the approach discussed in the preceding chapter might be best used in these types of models requires more investigation. Collaboration with scientists and modellers which are fluent in their use is likely to bring new perspectives on the development of Remodel, and possibly the methods in use in the domains dominated by these assimilation and ensemble techniques.

The central theme in this thesis has been the use of dynamic assessment and the reconfiguration of models to best manage the fidelity and efficiency of the model. The assessments and decisions of a model-run can be recorded like any other data generated by the model, and these data can inform the development of more comprehensive or nuanced decision strategies or models. The data may also provide insight into how the system changes its character through time at a macroscopic level.

It incorporating a <log> based checkpointing facility in Remodel would be fairly simple to implement. Such a facility could support a model configuration sensitivity analysis akin to the parameter sensitivity analyses we are familiar with. The data derived from this could be fed directly into subsequent assessments within the run in principle, should a monitor detect that a model has run off the rails, the whole model could be restored from a previous checkpoint, the decision logs could be examined and an alternative choice made at a point that precedes the models loss of fidelity. This sort of feature may be attractive in the implementation of submodels using data assimilation.

Early ecological models were necessarily quite simple in order to be able to run on the hardware available at the time. Moores Law has seemed to hold since it was first articulated in 1965, but it seems inevitable that, like organisms in ecosystems, the software will expand to make comprehensive use of its environment. This effect has been widely observed, and personal experience suggests that the domain of ecosystem modelling is not immune: Though hardly a broad survey, the scope of the studies Lyne et al. [1994a], Gray et al. [2006] and Gray et al. [2014] illustrate this growth. These models were based on the InVitro framework and the code bases for these models doubled in size roughly every three and a half years.

While quantum processors may also help with some aspects of simulation (such as path resolution or prey selection) to ameliorate this, it seems unlikely that they will provide a conceptual magic carpet that carries us immediately to our goal; as suggested above, the problem any new model addresses will grow to become just barely tractable. If this is true, no real technological advance or methodological insight (such as changing representations!) is really going to do much more that shift the boundary: modellers will still sit impatiently waiting for the model run to finish, wishing they had more memory, more qubits, and a faster CPU.

The goal of this work has been to develop a conceptual basis for models which are able to use their own state to direct the adaptation of their representation to optimise some aspect of their simulation. The evidence of the utility of this approach was established in the simple model of Chapter 2. Making models with the properties discussed in the preceding chapters is not a trivial undertaking, but with appropriate support it is only marginally more complicated than a conventional approach.

APPENDIX A

Typesetting conventions

Table A.1: Printing styles

Example	role
scheme-syntax	indicates a syntactic component of Scheme, or a macro declared in framework
symbolic-value	indicates a Scheme symbol
" *character string*"	indicates a string of characters
variable-name	indicates a Scheme variable
method-name	indicates an SCLOS method
agent-state-variable	indicates the name of an attribute of an agent
<***class-name***>	indicates the name of an SCLOS class

Keep in mind that in Scheme, a symbol may be bound to something (like a C pointer) such as a symbol, a string, a number, a list or a function. Methods are similar to functions (which are accessed through a *variable*), but they are not the same thing. A method *requires* an agent it knows how to handle as its first argument. State variables are usually only accessible by asking the agent for the value: other agents, and even the kernel, cannot delve into an agent's innards without help from the agent.

APPENDIX B

Snapshot of Remodel run

The following three figures show the trees in a simulation of fifty days. The diameter of each tree is proportional to its mass.

Note that in the first figure, there are a number of relatively large trees, and no small trees outside the four clusters. At day 25, the larger trees have been heavily grazed and there are a number of small trees distributed between the clusters. By day fifty, most of the trees are significantly smaller and there are a many more small trees distributed around the region.

The track of a single herbivore has been included as a half grey trace on each of the maps (visible in the upper right hand cluster).

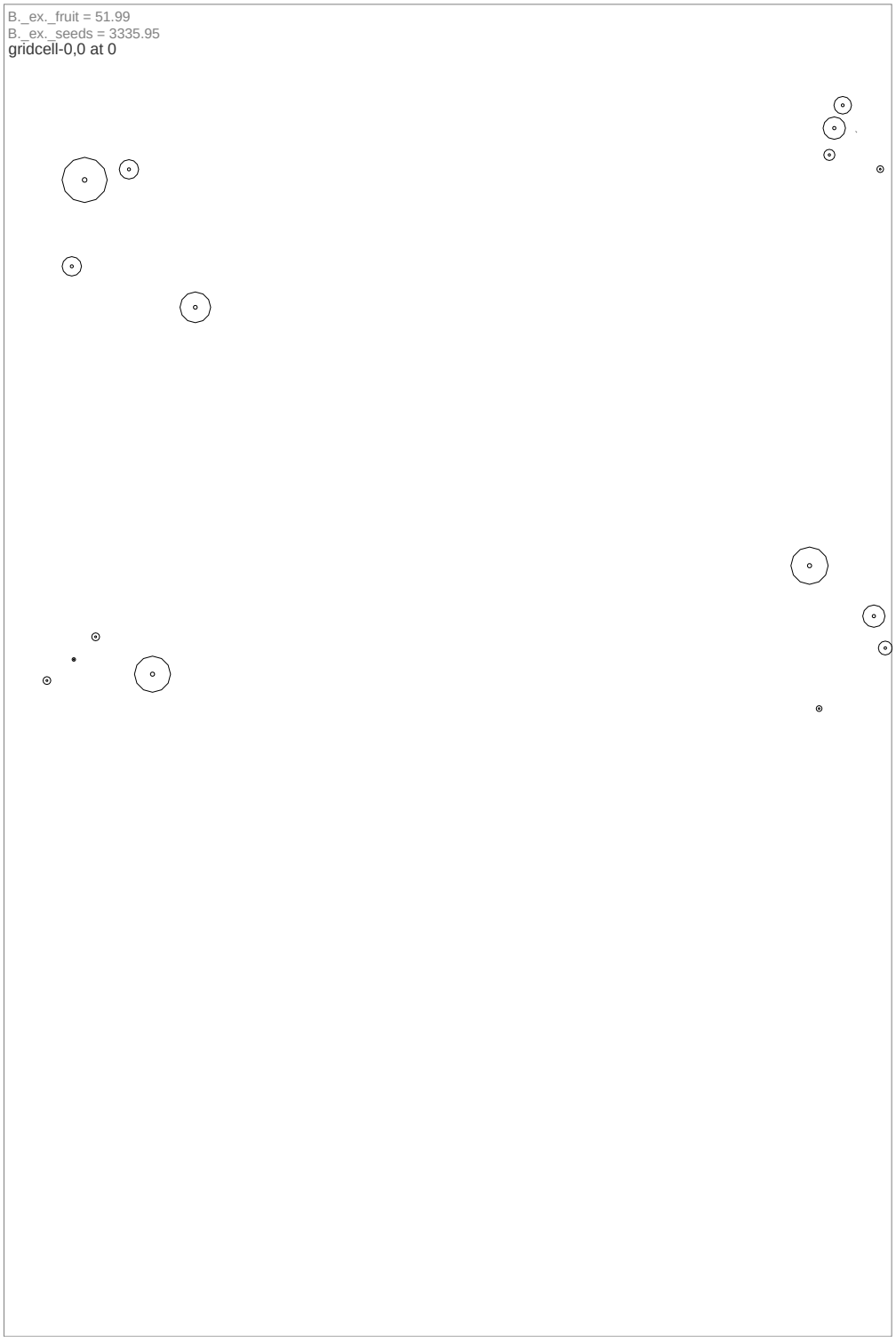

Figure B.1: Snapshots of tree-cover locations at day 0. The herbivore is not visible, because it hasn't moved yet.

Figure B.2: Snapshots of tree-cover locations at day 25. The herbivore is grazing on the foliage of the trees.

Figure B.3: Snapshots of tree-cover locations at day 50. The herbivore is still grazing on the foliage of the trees.

APPENDIX C

Supplementary Material

We can cast the dynamics of the system as a system of differential equations for the changes in distribution of biomass amongst the populations. These equations are solved at each step by a fourth order Runge-Kutta scheme that calculates the biomass distribution for each species.

So we denote our types by

- `P` – *plants*, from germination onwards,
- `F` – *fruit*, which are available for consumption,
- `S` – viable *seeds*, which have passed through a herbivore,
- `H` – *herbivores*, which eat the plants and fruit, and
- `C` – *carnivores*, which eat the juvenile herbivores,

and we take the notation $N_{\mathtt{A}}(t)$ to denote the total biomass of things of type `A` at the time t, $N_{\mathtt{A}}(t,x)$ indicates the total biomass of the members of type `A` which have a size x, and $\hat{N}_{\mathtt{A}}(t,x)$ denotes the proportion of the population that has a size x at t. Thus we have

$$N_{\mathtt{A}}(t) = \int_0^\infty N_{\mathtt{A}}(t,x)\,dx$$

and

$$\hat{N}_{\mathtt{A}}(t,x) = \frac{N_{\mathtt{A}}(t,x)}{N_{\mathtt{A}}}(t).$$

Clearly for positive values of $N_{\mathtt{A}}(t)$, $\hat{N}_{\mathtt{A}}(t,x) \in [0,1]$, for non-positive values of $N_{\mathtt{A}}(t)$ we will take $\hat{N}_{\mathtt{A}}(t,0) = 1$.

So for plants, we also make the following assumptions:

- Plants produce fruit continuously (without seasonal variation), and their mortality is uniformly distributed with respect to size and distributed evenly through the year,

- plants grow slowly enough that it is possible for all the adult herbivores to die before a plant fruits — this means that a near complete collapse of the plant population can trigger a collapse in the herbivore population
- fruit has a probability of spoiling, and the associated seeds will spoil
- seeds that have passed through the juveniles will germinate

The change in plant biomass for a given size due solely to growth is

$$dN_{\mathrm{P}}(t)|_{\mathrm{growth}}(t,x) = \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right) \int_0^\infty \Gamma_{\mathrm{P}}(x,y) N_{\mathrm{P}}(t,y)\, dy \tag{C.1}$$

where $\Gamma_{\mathrm{P}}(a,b)$ is the probability that a plant of size b will grow to size a. The germination of seeds contributes

$$dN_{\mathrm{P}}|_{\mathrm{germination}}(t,x) = \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right) N_{\mathrm{S}}(t)\Gamma_{\mathrm{S}}(x) \tag{C.2}$$

where $\Gamma_{\mathrm{S}}(x)$ is the probability of a seed growing to size x. We will also make germination success dependent on the density of plants relative to the carrying capacity, K_{P} — for the sake of the argument, germination might only occur in full sunlight.

The change in biomass in plants due to predation by herbivores is

$$dN_{\mathrm{P}}|_{\mathrm{predation}}(t,x) = -\int_0^\infty N_{\mathrm{P}}(t,x) N_{\mathrm{H}}(t,y) E_{\mathrm{HP}}(y,x)\, dy \tag{C.3}$$

where $E_{\mathrm{HP}}(y,x)$ is the probability that a plant of size x will be eaten by a herbivore of size y.

The mortality is given by

$$dN_{\mathrm{P}}|_{\mathrm{mortality}}(t,x) = -\Omega_{\mathrm{P}} N_{\mathrm{P}}(t,x) \tag{C.4}$$

Thus, the collective change in biomass can be written

$$\begin{aligned} dN_{\mathrm{P}}(t,x) =& \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right)\left[\int_0^\infty N_{\mathrm{S}}(t)\Gamma_{\mathrm{S}}(y)\, dy + \int_0^\infty \Gamma_{\mathrm{P}}(x,y) N_{\mathrm{P}}(t,y)\, dy\right] \\ & - N_{\mathrm{P}}(t,x)\left[\Omega_{\mathrm{P}} + \int_0^\infty N_{\mathrm{H}}(t,y) E_{\mathrm{HP}}(x,y)\, dy\right]. \end{aligned}$$

We'll take the production of fruit to be related to the surface area of a minimal bounding volume of a plant, so we include a quadratic term in x with the assumption that volume is proportional to mass, so

$$dN_{\mathrm{F}}|_{\mathrm{production}}(t) = \int_0^\infty \rho_{\mathrm{P}} f_{\mathrm{P}} x^{\frac{2}{3}} N_{\mathrm{P}}(t,x)\, dx, \tag{C.5}$$

where $f_{\mathrm{P}} x^{\frac{2}{3}}$ is the ("instantaneous") likelihood that a plant with a mass of x will fruit and we assume that ρ_{P}, the mean size of fruit, to be similar across all

sizes of plant. This is clearly a little silly, since newly germinated plants are unlikely to produce fruit that may be larger than they are, but in a spirit of blindly hacking through the undergrowth, we'll forge ahead.

A decrease in the population of fruit may be the result of "spoilage" and by the predation of the (juvenile) herbivores, so

$$dN_{\mathrm{F}}|_{\text{consumption}}(t) = -N_{\mathrm{F}}(t)\left(\kappa_{\mathrm{F}} + \int_0^\infty N_{\mathrm{H}}(t,y)E_{\mathrm{HF}}(y)\,dy\right) \tag{C.6}$$

where $\kappa_{\mathrm{F}}N_{\mathrm{F}}$ is a "spoilage" term that is constant across the population. Putting the equations together, we get

$$dN_{\mathrm{F}}(t) = \int_0^\infty \rho_{\mathrm{P}} f_{\mathrm{P}} x^{\frac{2}{3}} N_{\mathrm{P}}(t,x)\,dx - N_{\mathrm{F}}(t)\left(\kappa_{\mathrm{F}} + \int_0^\infty N_{\mathrm{H}}(t,y)E_{\mathrm{HF}}(y)\,dy\right) \tag{C.7}$$

Seeds are fairly simple,

$$dN_{\mathrm{S}}(t) = N_{\mathrm{F}}(t)\int_0^\infty N_{\mathrm{H}}(t,y)E_{\mathrm{HF}}(y)\,dy - \kappa_{\mathrm{S}} - \left(1 - \frac{N_{\mathrm{P}}(t)}{K_{\mathrm{P}}}\right)N_{\mathrm{S}}\int_0^\infty (t)\Gamma_{\mathrm{S}}(x)\,dx \tag{C.8}$$

The any change in the biomass of herbivores is determined by birth, growth, natural mortality, starvation and the effect of predation; the only real difference between adults and juveniles is their size and the functions E_{HF} and E_{HP}. Thus we get the following component equations:

$$dN_{\mathrm{H}}|_{birth}(t,x) = N_{\mathrm{H}}(t,x)f_{\mathrm{H}}(x) \tag{C.9}$$

$$\begin{aligned} dN_{\mathrm{H}}|_{growth}(t,x) = N_{\mathrm{H}}(t,x)\Big(&\Gamma_{\mathrm{H}}(x)\int_0^\infty N_{\mathrm{P}}(t,w)E_{\mathrm{HP}}(x,w)\,dw + \min\left(0, \frac{N_{\mathrm{F}} - \omega_{\mathrm{HF}}(x)N_{\mathrm{H}}}{N_{\mathrm{F}}}\right) \\ &+ \Gamma_{\mathrm{H}}(x)N_{\mathrm{F}}(t)E_{\mathrm{HF}}(x) - M_{\mathrm{H}}\Big) \end{aligned} \tag{C.10}$$

$$dN_{\mathrm{H}}|_{mortaltiy}(t,x) = -N_{\mathrm{H}}(t,x)\Big(\Omega_{\mathrm{H}}(x) + \int_0^\infty N_{\mathrm{C}}(y)E_{\mathrm{CH}}(y,x)\,dy\Big) \tag{C.11}$$

where $\Omega_{\mathrm{H}}(x)$ is the probability of natural mortality for a herbivore, and f_{H} is the expected reproductive contribution from a herbivore with a size of x, $\Gamma_{\mathrm{H}}(x)$ is the growth coefficient for herbivores of size x and M_H is a maintenance rate per unit of biomass. Combining these components, we get the the equation

$$\begin{aligned} dN_{\mathrm{H}}(t,x) = N_{\mathrm{H}}(t,x)\Bigg[&f_{\mathrm{H}}(x) + \Gamma_{\mathrm{H}}(x)\int_0^\infty N_{\mathrm{P}}(t,w)E_{\mathrm{HP}}(x,w)\,dw + \min\left(0, \frac{N_{\mathrm{P}} - \omega_{\mathrm{HP}}(x)N_{\mathrm{H}}}{N_{\mathrm{P}}}\right) \\ &+ \min\left(0, \frac{N_{\mathrm{F}} - \omega_{\mathrm{HF}}(x)N_{\mathrm{H}}}{N_{\mathrm{F}}}\right) + \Gamma_{\mathrm{H}}(x)N_{\mathrm{F}}(t)E_{\mathrm{HF}}(x) - \left(M_{\mathrm{H}} + \int_0^\infty N_{\mathrm{C}}(y)E_{\mathrm{CH}}(y,x)\,dy\right)\Bigg] \end{aligned} \tag{C.12}$$

The population of carnivores is largely similar to that of herbivores with the exception that they prey on the herbivores and do not suffer from predation themselves. growth, birth, and their ability to catch juvenile herbivores.

$$dN_{\mathtt{C}}(t,x) = N_{\mathtt{C}}(t,x)\Bigg[f_{\mathtt{C}}(x) + \Gamma_{\mathtt{C}}(x)\int_0^\infty N_{\mathtt{H}}(t,w)E_{\mathtt{CH}}(x,w)\,dw + \min\Bigg(0, \int_0^\infty \frac{N_{\mathtt{H}}(y) - \omega_{\mathtt{CH}}(x)N_{\mathtt{H}}(y)}{N_{\mathtt{H}}(y)}\,dy\Bigg) - \big(M_{\mathtt{C}} + \Omega_{\mathtt{C}}(x)\big)\Bigg] \quad \text{(C.13)}$$

where $\omega_{\mathtt{AB}}(x) \geq 1$ indicates how much an animal with a mass, x, needs to eat relative to the mass of the prey[1]. The predation functions $\omega_{\mathtt{HP}}, \omega_{\mathtt{HF}}, E_{\mathtt{CH}}, E_{\mathtt{HP}}$ and $E_{\mathtt{HF}}$ are constructed so that the preying species observes the conditions on the system: $E_{\mathtt{CH}}(x,y)$, for example, is constructed so that for small x the carnivores do not prey on herbivores of any size, and for larger x they prey only upon herbivores with small values of y. Similarly, $E_{\mathtt{HF}}(x)$ is only non-zero for small values of x, and $E_{\mathtt{HP}}$ is non-zero where $E_{\mathtt{HF}}$ is zero. The implication of this is that $\int_0^\infty \omega_{\mathtt{HP}}(x)\omega_{\mathtt{HF}}(x)\,dx = 0$ and $\int_0^\infty E_{\mathtt{HP}}(x)E_{\mathtt{HF}}(x)\,dx = 0$.

[1] This relationship is likely to only hold in very specific circumstances.

BIBLIOGRAPHY

AnyLogic. Release: Anylogic, ver. 4, December 2001.

MP Bailey and WG Kemple. The scientific method of choosing model fidelity. In *Proceedings of the 24th conference on Winter simulation*, pages 791–797. ACM, 1992.

M Berdoy, JP Webster, and DW Macdonald. Fatal attraction in rats infected with toxoplasma gondii. *Proceedings of the Royal Society of London B: Biological Sciences*, 267 (1452):1591–1594, 2000.

P Bhatanacharoen, D Greatbatch, and T Clark. The tipping point of the tipping point-metaphor: Agency and process for waves of change. 2011.

GV Bobashev, DM Goedecke, F Yu, and JM Epstein. A hybrid epidemic model: combining the advantages of agent-based and equation-based approaches. In *Simulation Conference, 2007 Winter*, pages 1532–1537. IEEE, 2007.

F Boschetti, C Richert, I Walker, J Price, and L Dutra. Assessing attitudes and cognitive styles of stakeholders in environmental projects involving computer modelling. *Ecological Modelling*, 247:98–111, 2012.

DB Botkin, JF Janak, and JR Wallis. Rationale, limitations, and assumptions of a northeastern forest growth simulator. *IBM Journal of Research and Devevelopment*, 16:101–116, 1972a.

DB Botkin, JF Janak, and JR Wallis. Some ecological consequences of a computer model of forest growth. *The Journal of Ecology*, 60:849–872, 1972b.

JB Calhoun. Death squared: the explosive growth and demise of a mouse population. *Proceedings of the Royal Society of Medicine*, 66(1 Pt 2):80, 1973.

H Caswell and AM John. From the individual to the population in demographic models. In D. L. DeAngelis and L. J. Gross, editors, *Individual-based models and approaches in ecology*, pages 36–61. Springer, 1992.

WJ Chivers. *Generalised, parsimonious, individual-based computer models of ecological systems*. University of Newcastle, 2009.

N Collier, T Howe, and M North. Onward and upward: the transition to repast 2.0. In *First Annual North American Association for Computational Social and Organizational Science Conference, Pittsburgh, PA*, 2003.

iMatix Corp. ∅MQ: The Intelligent Transport Layer, November 2012. URL `http://www.zeromq.org`.

SJ de Almeida, R Poley M Ferreira, ÁE Eiras, RP Obermayr, and M Geier. Multi-agent modeling and simulation of an aedes aegypti mosquito population. *Environmental Modelling & Software*, 25(12):1490–1507, 2010.

DL DeAngelis. Model for the movement and distribution of fish in a body of water. Technical report, Oak Ridge National Lab., Tenn.(USA), 1978.

DL DeAngelis and LJ Gross, editors. *Individual-Based Models and Approaches in Ecology*. Springer, isbn-13: 978-0412031618, isbn-10: 0412031612 edition, 1992.

DL DeAngelis, LJ Gross, MJ Huston, WF Wolff, DM Fleming, EJ Comiskey, and SM Sylvester. Landscape modelling for everglades ecosystem restoration. *Ecosystems*, 1:64–75, 1998.

T DelSole and J Shukla. Model fidelity versus skill in seasonal forecasting. *Journal of Climate*, 23(18):4794–4806, 2010.

AP Dobson. The population biology of parasite-induced changes in host behavior. *Quarterly Review of Biology*, pages 139–165, 1988.

BD Elderd, J Dushoff, and G Dwyer. Host-pathogen interactions, insect outbreaks, and natural selection for disease resistance. *The American Naturalist*, 172(6):829–842, 2008.

S Farolfi, J-P Müller, and B Bonté. An iterative construction of multi-agent models to represent water supply and demand dynamics at the catchment level. *Environmental modelling & software*, 25(10):1130–1148, 2010.

Food and Agriculture Organisation of the United Nations (FAO). Best practices in ecosystem modelling: Modelling ecosystem interactions for informing an ecosystem approach to fisheries. fisheries management, 2008.

EA Fulton. Approaches to end-to-end ecosystem models. *Journal of Marine Systems*, 81(1):171–183, 2010.

EA Fulton, 2011. Personal communication.

EA Fulton, ADM Smith, and CR Johnson. Biogeochemical marine ecosystem models. i: Igbem – a model of marine bay ecosystems. *Ecological Modelling*, 174:267–307, 2004a.

EA Fulton, ADM Smith, and CR Johnson. Effects of spatial resolution on the performance and interpretation of marine ecosystem models. *Ecological Modelling*, 176: 27–42, 2004b.

EA Fulton, R Gray, M Sporcic, R Scott, and M Hepburn. Challenges of crossing scales and drivers in modelling marine systems. *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation, July*, pages 2108–2114, 2009.

EA Fulton, R Gray, M Sporcic, R Scott, LR Little, M Hepburn, B Gorton, B Hatfield, M Fuller, T Jones, W De la Mare, F Boschetti, K Chapman, P Dzidic, G Syme, J Dambacher, and D McDonald. Ningaloo collaboration cluster: Adaptive futures for ningaloo. Technical Report 5.3, Ningaloo Collaboration Cluster, Hobart, Tasmania, October 2011a. URL `http://www.ningaloo.org.au/www/en/NingalooResearchProgram/Background.htm`.

EA Fulton, JS Link, IC Kaplan, M Savina-Rolland, P Johnson, C Ainsworth, P Horne, R Gorton, RJ Gamble, ADM Smith, et al. Lessons in modelling and management of marine ecosystems: the atlantis experience. *Fish and Fisheries*, 12(2):171–188, 2011b.

R Gray and S Wotherspoon. Increasing model efficiency by dynamically changing model representations. *Environ. Model. Softw.*, 30:115–122, April 2012. ISSN 1364-8152. doi: 10.1016/j.envsoft.2011.08.012.

R Gray, EA Fulton, LR Little, and R Scott. *Ecosystem model specification within an agent based framework. North West Shelf Joint Environmental Management Study Technical Report*. Number 16 in CSIRO-CMAR NWSJEMS Technical Reports. CSIRO, Hobart, Tasmania, Hobart, Tasmania, 2006. ISBN 1 921061 80 4 (pbk), ISBN 1 921061 82 0 (pdf).

R Gray, EA Fulton, and R Little. Human-ecosystem interaction in large ensemble-models. In A Smajgl and O Barreteau, editors, *Empirical Agent-Based Modelling - Challenges and Solutions*, pages 53–83. Springer New York, 2014. ISBN 9781-461-4613-3-3. doi: 10.1007/978-1-4614-6134-0_4.

V Grimm and SF Railsback. *Individual-based Modeling and Ecology: (Princeton Series in Theoretical and Computational Biology)*. Princeton University Press, July 2005. ISBN 0-691-09666-X. URL `http://www.worldcat.org/isbn/069109666X`.

V Grimm, U Berger, F Bastiansen, S Eliassen, V Ginot, J Giske, J Goss-Custard, T Grand, SK Heinz, et al. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2):115–126, 2006. ISSN 0304-3800. doi: DOI:10.1016/j.ecolmodel.2006.04.023. URL `http://www.sciencedirect.com/science/article/B6VBS-4K606T7-3/2/1dad6192bec683f32fce6dee9d665b51`.

CJ Harvey, SP Cox, TE Essington, S Hansson, and JF Kitchell. An ecosystem model of food web and fisheries interactions in the baltic sea. *ICES Journal of Marine Science*, 60:939–950, 2003.

X Hu and DH Edwards. Behaviorsim: A simulation environment to study animal behavioral choice mechanisms. In *Proceedings of the 2005 DEVS Integrative M&S Symposium, Spring Simulation Multiconference, San Diego CA*. Citeseer, 2005.

M Huston, D DeAngelis, and W Post. New computer models unify ecological theory. *BioScience*, 38(10):682–691, 1988.

DA Keith, TG Martin, E McDonald-Madden, and C Walters. Uncertainty and adaptive management for biodiversity conservation. *Biological Conservation*, 144(4):1175–1178, 2011. ISSN 0006-3207. doi: 10.1016/j.biocon.2010.11.022. URL `http://www.sciencedirect.com/science/article/pii/S0006320710004933`. <ce:title>Adaptive management for biodiversity conservation in an uncertain world</ce:title>.

G Kiczales, JM Ashley, L Rodriguez, Amin Vahdat, and Daniel G Bobrow. Metaobject protocols: Why we want them and what else they can do. *Object-Oriented Programming: The CLOS Perspective*, pages 101–118, 1993.

KD Lafferty, AP Dobson, and AM Kuris. Parasites dominate food web links. *Proceedings of the National Academy of Sciences*, 103(30):11211–11216, 2006.

LR Little, EA Fulton, R Gray, D Hayes, R Scott, AD McDonald, and K Sainsbury. Management strategy evaluation results and discussion for the north west shelf. Final Report 14, CSIRO Australia, Hobart, Tasmania, 2006. ISBN 1 921061 74 X (pbk), ISBN 1 921061 76 6 (pdf).

S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 SwarmFest Workshop*, 2004.

V. Lyne, R. Gray, K. Sainsbury, and R. Scott. Integrated biophysical model investigations. Final report, CSIRO Australia, Division of Fisheries, Hobart, Tasmania, 1994a.

V Lyne, R Gray, and R Scott. North west shelf joint environmental management study: Invitro inputs-statistical extrapolation of currents. Final report, CSIRO Australia, Division of Fisheries, Hobart, Tasmania, 1994b.

Thomas Robert Malthus. *An Essay on the Principle of Population*. Library of Economics and Liberty, Internet, 16 feb 2010 edition, 1798. URL `http://www.econlib.org/library/Malthus/malPop.html`. First edition, originally published by J. Johnson, London.

N Minar, R Burkhart, C Langton, and M Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations, 1996.

L Monte. A methodological approach to develop *contaminant migration-population effects* models. *Ecological Modelling*, 220:3280–3290, 2009.

T Polacheck, NL Klaer, C Millar, and AL Preece. An initial evaluation of management strategies for the southern bluefin tuna fishery. *ICES Journal of Marine Science: Journal du Conseil*, 56(6):811–826, 1999. doi: 10.1006/jmsc.1999.0554. URL `http://icesjms.oxfordjournals.org/content/56/6/811.abstract`.

R Poulin. Meta-analysis of parasite-induced behavioural changes. *Animal Behaviour*, 48(1):137–146, 1994.

KA Rose, JI Allen, Y Artioli, M Barange, J Blackford, et al. End-to-end models for the analysis of marine ecosystems: Challenges, issues, and next steps. *Marine and Coastal Fisheries: Dynamics, Management, and Ecosystem Science*, 2:115–130, 2010.

KJ Sainsbury, AE Punt, and ADM Smith. Design of operational management strategies for achieving fishery ecosystem objectives. *ICES Journal of Marine Science: Journal du Conseil*, 57(3):731–741, 2000. doi: 10.1006/jmsc.2000.0737. URL `http://icesjms.oxfordjournals.org/content/57/3/731.abstract`.

M Scheffer, JM Baveco, DL DeAngelis, and KA Rose. Super-individuals: a simple solution for modelling large populations on an individual basis. *Ecological Modelling*, 80:161–170, 1995.

AD Smith. Management strategy evaluation - the light on the hill. In D. A. Hancock, editor, *Australian Society for Fish Biology Workshop Proceedings*, volume 97 of *Population Dynamics for Fisheries Management*, pages 249–253. Australian Society for Fish Biology, August 1993. URL `http://asfb.org.au/pdf/1993/1993-06-04.pdf`.

GJ Sussman and GL Steele. The first report on scheme revisited. *Higher-Order and Symbolic Computation*, 11(4):399–404, 1998.

GE Swan, R Cuthbert, M Quevedo, RE Green, DJ Pain, P Bartels, AA Cunningham, N Duncan, AA Meharg, JL Oaks, et al. Toxicity of diclofenac to gyps vultures. *Biology letters*, 2(2):279–282, 2006.

JC Thiele and V Grimm. Netlogo meets r: Linking agent-based models with a toolbox for their analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

WH van Dobben. The food of the cormorant in the netherlands. *Ardea*, 40(1-2):1–63, 1952.

E Vincenot, F Giannino, M Rietkerk, K Moriya, and S Mazzoleni. Theoretical considerations on the combined use of system dynamics and individual-based modeling in ecology. *Ecological Modelling*, 222(1):210–218, 2011. ISSN 0304-3800. doi: DOI: 10.1016/j.ecolmodel.2010.09.029. URL `http://www.sciencedirect.com/science/article/B6VBS-518MX70-2/2/199628e93c44d13863a48b3299472a32`.

G Wallentin and C Neuwirth. Dynamic hybrid modelling: Switching between ab and sd designs of a predator-prey model. *Ecological Modelling*, 345:165–175, 2017.

C Walters, J Korman, LE Stevens, and B Gold. Ecosystem modeling for evaluation of adaptive management policies in the grand canyon. *Conservation Ecology*, 4(2):1, 2000.

CJ Walters and R Hilborn. Adaptive control of fishing systems. *Journal of the Fisheries Research Board of Canada*, 33(1):145–159, 1976. doi: 10.1139/f76-017.

CJ Walters and SJD Martell. *Fisheries ecology and management*. Princeton University Press, 2004.

RMP Ward and CJ Krebs. Behavioural responses of lynx to declining snowshoe hare abundance. *Canadian Journal of Zoology*, 63(12):2817–2824, 1985.

U Wilensky. Netlogo, 1999. URL `http://ccl.northwestern.edu/netlogo/`.

WF Wolff. An individual-oriented model of a wading bird nesting colony. *Ecological Modelling*, 72(1-2):75–114, 1994. ISSN 0304-3800. doi: DOI:10.1016/0304-3800(94)90146-5. URL `http://www.sciencedirect.com/science/article/B6VBS-48YNSFT-2X/2/09ef35a0a8e95b3261adfc540cd87a23`.

M Xu, J-W Bao, TT Warner, and DJ Stensrud. Effect of time step size in mm5 simulations of a mesoscale convective system. *Monthly weather review*, 129(3):502–516, 2001.

WS Yip and TE Marlin. The effect of model fidelity on real-time optimization performance. *Computers & chemical engineering*, 28(1):267–280, 2004.

SM Zala and DJ Penn. Abnormal behaviours induced by chemical pollution: a review of the evidence and new challenges. *Animal Behaviour*, 68(4):649–664, 2004.

S Zhang, Q Zhu, and AK Roy-Chowdhury. Adaptive algorithm and platform selection for visual detection and tracking. *CoRR*, abs/1605.06597, 2016. URL `http://arxiv.org/abs/1605.06597`.

W Zhang and DR Montgomery. Digital elevation model grid size, landscape representation, and hydrologic simulations. *Water resources research*, 30(4):1019–1028, 1994.