Solving stochastic dynamic programming models without transition matrices

Paul L. Fackler

Department of Agricultural & Applied Economics and Department of Applied Ecology

North Carolina State University

Computational Sustainability Seminar

Nov. 3, 2017

Outline

Brief review of dynamic programming
- curses of dimensionality
- index vectors
- DP algorithms

Expected Value (EV) functions
- Staged models
- Models with deterministic post-action states
- Factored Models

Factored models & conditional independence
- Evaluation of EV functions
- Results for two spatial models:
  - dynamic reserve site selection
  - control of an invasive species on a spatial network

Models with transition functions and random noise

Wrap-up

Dynamic Programming Problems

Given state values S, action values A, reward function $R(S,A)$, state transition probability matrix $P(S^+|S,A)$ and discount factor δ, solve

$$V(S) = \max_{A(S)} \sum_{t=0}^{\infty} \delta^t E_t\left[R\left(S_t, A(S_t)\right)\right]$$

Equivalently solve Bellman's equation:

$$V(S) = \max_{A(S)} R(S, A(S)) + \delta \sum_{S^+} P(S^+|S, A(S))V(S^+)$$

Find the strategy $A(S)$ that maximizes:

the current reward R plus

the discount factor δ times

the expected future value $\sum_{S^+} P(S^+|S,A)V(S^+)$

Curses of dimensionality

Problem size grows exponentially with increases in the number of variables

Powell discusses 3 curses:
growth in the state space
growth in the action space
growth in the outcome space

In discrete models we represent
the size of the state space as n_s
the size of the state/action space as n_x

The state transition probability matrix is $n_s \times n_x$

Focus here on problems for which
vectors of size n_x can be stored and manipulated
but matrices of size $n_s \times n_x$ are problematic

Thus the focus in on moderately sized problems

By having techniques to solve moderately sized problems we can gain insight into the quality of heuristic or approximate methods that must be used for large problems

Index Vectors

Vectors composed of positive integers
Used for:

- extraction
- expansion
- shuffling

Let:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 2 & 0 \\ 2 & 1 \\ 3 & 0 \\ 3 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \\ 3 & 0 & 0 \\ 3 & 0 & 1 \\ 3 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

$I = [5 \quad 6 \quad 7 \quad 8]$ extracts the rows of B with the first column equal to 2: $B(I,1) = 2$

$I = [1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 5 \quad 5 \quad 6 \quad 6]$ expands A so $A(I,:) = B(:,[1\ 2])$

$I = [1 \quad 2 \quad 1 \quad 2 \quad 3 \quad 4 \quad 3 \quad 4 \quad 5 \quad 6 \quad 5 \quad 6]$ expands A so $A(I,:) = B(:,[1\ 3])$

Dynamic Programming with Index Vectors

Consider a DP model with 2 state variables each binary and 3 possible actions

S lists all possible states and matrix X lists all possible state/action combinations:

$$S = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \\ 3 & 0 & 0 \\ 3 & 0 & 1 \\ 3 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

Column 1 of X is the action and columns 2 and 3 are the 2 states

The expansion index vector that gives the states in each row of X is

$I_x = [1 \quad 2 \quad 3 \quad 4 \quad 1 \quad 2 \quad 3 \quad 4 \quad 1 \quad 2 \quad 3 \quad 4]$

This expands S so $S(I_x,:) = X(:,[2\ \ 3])$

Strategies as Index Vectors

A strategy can be specified as an extraction index vector with the ith element associated with state i:

$I^a = [1 \;\; 6 \;\; 7 \;\; 12\,]$ yields:

$$X(I^a,:) = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

i.e., a strategy that associates action 1 with state 1, action 2 with states 2 and 3 and action 3 with state 4

Strategy vectors select a single row of X for each state so $X(I^a, J^s) = S$ where J^s is an index of the columns of X associated with the state variables

Dynamic Programming Algorithms

Typically solved with function iteration or policy iteration

Both use a maximization step that, for a given value function vector V, solves:

$$\tilde{V}_i = \max_{j:\, I_x(j)=i} [R + \delta P^\top V]_j$$

with the associated strategy vector I^a:

$$I_i^a = \underset{j:\, I_x(j)=i}{\operatorname{argmax}} [R + \delta P^\top V]_j$$

This is followed by a value function update step

Function iteration updates V using:

$$V \leftarrow \tilde{V}$$

Policy iteration updates V by solving:

$$WV = (I - \delta P[:, I^a]^\top)V = R[:, I^a]$$

When the discount factor $\delta < 1$ the matrix $W = I - \delta P[:, I^a]^\top$ is row-wise diagonally dominant

Dynamic Programming with Expected Value (EV) functions

An EV function v transforms the future state vector into its expectation conditional on current states and actions (X):

$$v(V^+) = E[V^+|X]$$

An indexed evaluation transforms the future state vector into its expectation condition on the states and actions indexed by I^a

$$v(V^+, I^a) = E[V^+|X[I^a, :]]$$

The maximization step uses a full EV evaluation:

$$\max_{j:\, I_x(j)=i} R_j + \delta[v(V)]_j$$

Value function updates use an indexed evaluation

Function iteration:

$$V \leftarrow R[I^a] + \delta v(V, I^a)$$

Policy iteration (solve for V):

$$h(V) = V - \delta v(V, I^a) = R[I^a]$$

Note that policy iteration with EV functions

cannot be solved using direct methods (e.g., LU decomposition)

but can be solved efficiently using iterative Krylov methods

Advantages to using EV functions

The EV function v can often be evaluated far faster and use far less memory than using the transition matrix P

There are at least 3 situations in which EV functions are advantageous:

- Sparse staged transition matrices
- Deterministic actions
- Factored models with conditional independence

When the state transition occurs in 2 stages the transition matrix can be written as

$$P = P_2 P_1$$

where P_1 and P_2 are both sparse but their product is not

A deterministic action transforms the current state into a post-decision state
The transition matrix can be written as $P = \tilde{P} A$ where A has a single 1 in each column

In factored models individual state variables have their own transition matrices that are conditioned on a subset of the current states and actions

SPOMs with staged transitions

Stochastic Patch Occupancy Models (SPOMs):

N sites w/ each site either empty or occupied (0/1)

Individual site transition matrices for each stage are triangular:

$$E_i = \begin{bmatrix} 1 & e_i \\ 0 & 1 - e_i \end{bmatrix} \qquad C_i = \begin{bmatrix} 1 - c_i & 0 \\ c_i & 1 \end{bmatrix}$$

2^N possible state values

P has 4^N elements and is dense

If the transition is decomposed into extinction and colonization phases:

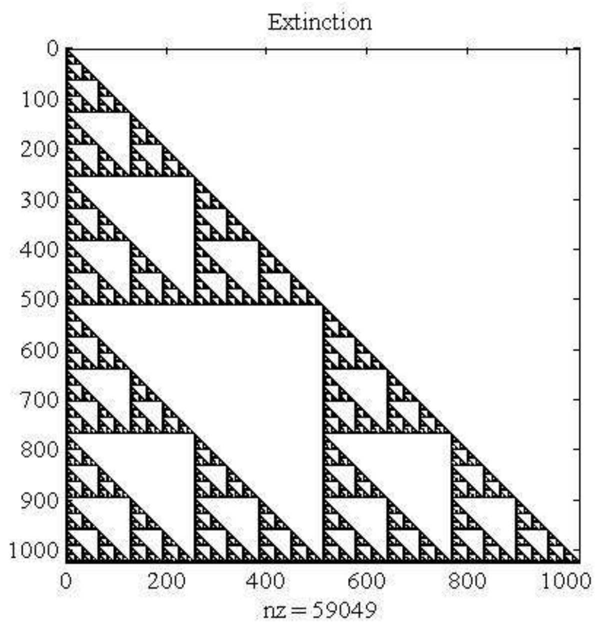
$P = EC$ or $P = CE$

E and C are sparse with each have 3^N non-zero elements in these matrices

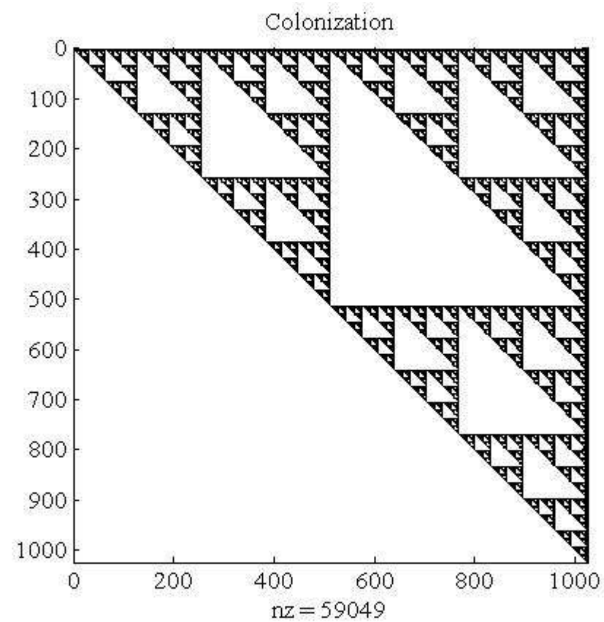
Sparsity patterns for extinction and colonization transition matrices

For $N = 10$

E



C



Typical computational times for SPOM model

Time required to do a basic matrix-vector and matrix-matrix multiply

	N						
	8	9	10	11	12	13	14
$E^\top(C^\top v)$	0.026	0.065	0.086	0.136	0.292	1.672	4.870
Pv	0.014	0.036	0.084	0.801	4.011	15.298	64.277
$P = CE$	0.008	0.008	0.046	0.154	0.724	3.499	19.332
density	0.100	0.075	0.056	0.042	0.032	0.024	0.018

Rows 1 & 2 display the time required for 1000 evaluations using
factored form $E^\top(C^\top v)$ and
full form $P^\top v$

Row 3 shows the setup time required to a form P

Row 4 shows the fraction of non-zero elements in E and C

These results are even more dramatic if each site can be classified into more than 2 categories.

Deterministic effect of actions and post-decision states

Post-decision state $\tilde{S}$ is a deterministic function of the state and action: $\tilde{S} = g_1(S, A)$

The future state depends stochastically on the post decision state: $P_2 = P(S^+ | \tilde{S})$

Example: fisheries models

state	current stock
action	harvest
post-harvest state	escapement

Future stock depends on escapement = current stock - harvest

In this case we require

$n_s \times n_s$ transition matrix P_2

n_x index vector $\mathcal{I}_1$ that defines the g mapping.

The expected value function can then be written as $EV(V) = [P_2^\top V](\mathcal{I}_1)$

This helps address curse of dimensionality in the action space

Factored models and conditional independence

Factored models can be expressed in terms of a set of variables, each with a transition matrix

When enough conditional independence exists use of the factored form leads to substantial computational efficiencies

Levels of conditional independence:

1) each future state has unique set of conditioning variables
2) conditioning variables involve overlapping sets of current states & actions
3) conditioning variables include overlapping sets of random variables
4) some future states are causally dependent on other future states

Examples

Level 1: dynamic reserve site selection
Level 2: network spatial model of invasive species control
Level 3: population dynamics with multiple age/stage classes

Level 1 Conditional Independence

If the conditioning sets for all the state variables are disjoint the transition matrix can be written as

$$P = P_1 \otimes P_2 \otimes \ldots \otimes P_d$$

The EV function is therefore

$$v(V) = (P_1^\top \otimes P_2^\top \otimes \ldots \otimes P_d^\top)V$$

This chained Kronecker product can be efficiently computed without forming P using a series of d matrix-matrix multiplies

A MATLAB implementation:

```
function y=chainkron(P,V);
    d = length(P);
    y = V;
    for i=1:d
        ni = size(P{i},1);
        y = reshape(y,numel(y)/ni,ni);
        y = P{i}'*y';
    end
    y = y(:);
```

Dynamic Reserve Site Selection Problem

Costello & Polasky (2004) "Dynamic Reserve Site Selection."

N sites
Each site in one of 3 categories: available, in the reserve or developed
If not acquired site i will move from available to developed with probability p_i.
One site can be acquired each period

State space represented by $3^N \times N$ matrix S

The action (acquisition) changes the state in a deterministic way so the model can be specified in terms of a post-acquisition transition matrix

P is a $3^N \times 3^N$ post-acquisition transition matrix which contains 4^N non-zero elements:

$$P = P_1 \otimes P_2 \otimes \ldots \otimes P_N$$

where the P_j are 3×3 individual site transition matrices

$$P_j = \begin{bmatrix} 1-p_j & 0 & 0 \\ 0 & 1 & 0 \\ p_j & 0 & 1 \end{bmatrix}.$$

The chained Kronecker product – vector multiplication can be implemented sequentially using $N3^{N-1}$ operations involving only the P_j

But it's complicated

The individual P_i are 3×3 and sparse with exactly 4 non-zeros

P is sparse with exactly 4^N non zero elements

The multiplication counts using

sparse P matrix: $4^N = \left(\frac{4}{3}\right)^N 3^{\mathrm{N}}$

N individual P_i: $N3^{N-1} = N\left(\frac{4}{3}\right)3^N$

$\left(\frac{4}{3}\right)^N < N\left(\frac{4}{3}\right)$ when $N \leq 8$ so using the full matrix has fewer operations for small N

It is possible that using more than 1 but less than N submatrices may be better yet

If we use a continuous approximation the multiplication count, using s submatrices, is

$$s\left(\frac{4}{3}\right)^{N/s}$$

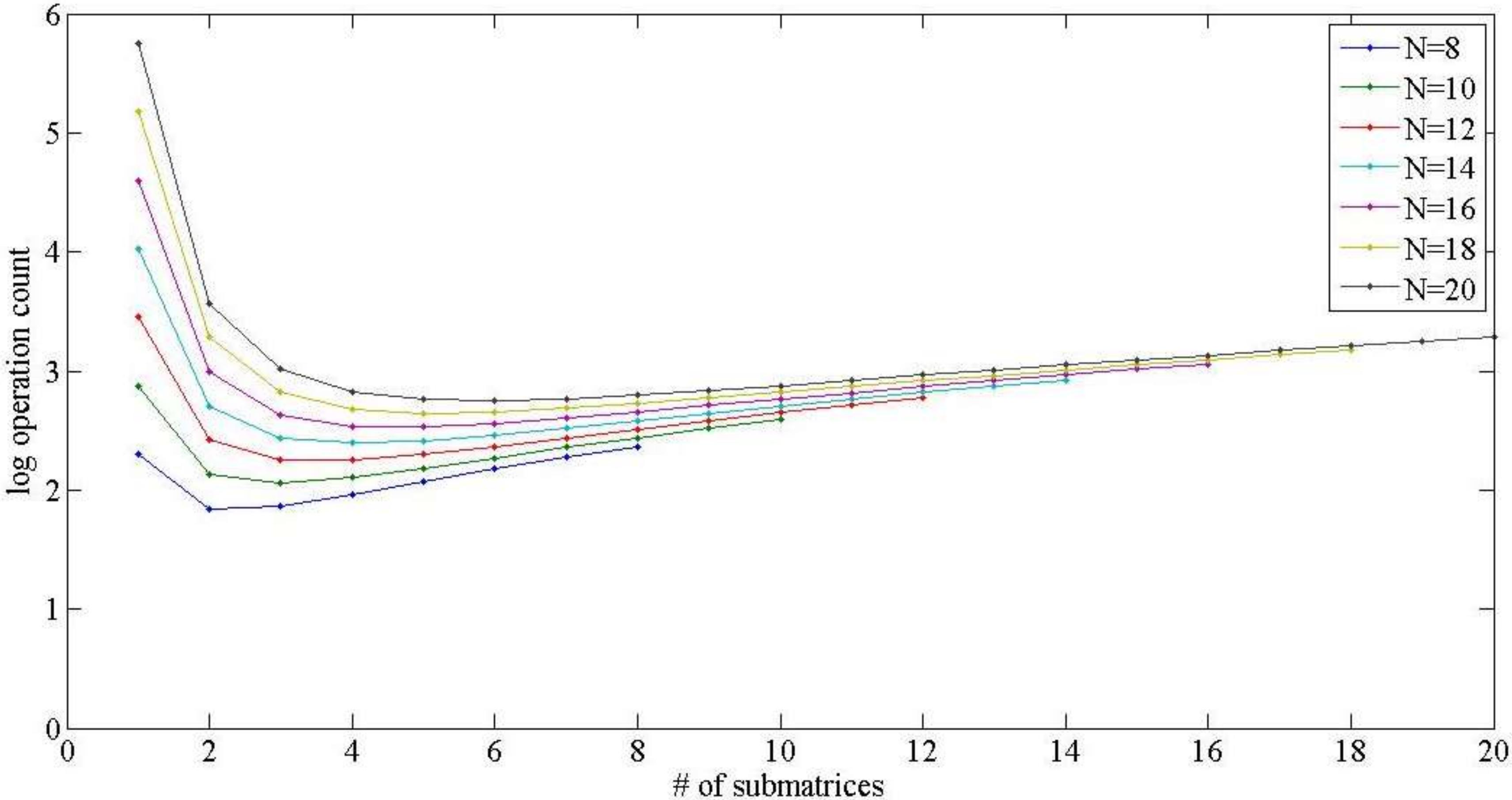
(this is exact when N/s is integer)

This expression is minimized when $s = \ln\left(\frac{4}{3}\right)N = 0.2877\ N$

In practice there seems to be a penalty for using more submatrices

But it's complicated (cont.)

Relative operation counts are shown below (log scale)



Use of a relatively small # of submatrices is indicated

In practice using 2 submatrices for $8 < N < 16$ and gradually increasing as N increases appears to be optimal

Level 2 Conditional Independence

Consider a model defined by a set of d state variables

The conditioning variables are organized into an $n_x \times d_x$ matrix X
each row represents a unique combination of states and actions

Each state has:
a CPT P_i representing the transition probability conditioned on a subset of X
an index vector q_i defining a set of conditioning (parent) variables, i.e., columns of X

Each CPT is processed sequentially using index vectors to match according to the conditioning variables

The basic approach requires an indexed multiplication of a 3-D array by a 2-D array:

$$y(h,k) \leftarrow y\left(h,:,I_i^y(k)\right) * P_i\left(:,I_i^p(k)\right)$$

where I_i^y and I_i^p are index vectors that match the 3rd dimension of y with the columns of P_i

No memory copying and shuffling of memory required

For each k there is a matrix vector multiply that is implemented with a call to dgemm

A function is produced that is called using
$v(V)$ for a full evaluation or
$v(V, I^a)$ for an indexed evaluation

Evaluating EV functions with index vectors

Set $y_0 = V$ and let y_i be the intermediate product after incorporating the first i CPTs

The I_i^p and I_i^y vectors have length m_i with $m_{i-1} \leq m_i \leq n_x$: $m_i = \prod_{j \in Q_i} n_j$ where $Q_i = \bigcup_{k=1}^{i} q_k$

In words, m_i is the size of the space of conditioning variables for the first i state variables

The total operation count is $\sum_{i=1}^{d} p_i m_i$ where $p_i = \prod_{j=i}^{d} n_j$

(p_i is the size of the space of the remaining unprocessed state variables)

This can be contrasted to the use of the full transition matrix, which uses $n_s n_x$ operations

Note that variable order matters and ideally we want the m_i to grow slowly

Using the I index vectors a full EV function evaluation is computed using the following algorithm:

set $y = v$
reshape y to be $\prod_{j=2}^{d} n_j \times n_1$
set $y \leftarrow y * p_1$
loop from $i = 2$ to $i = d$
  reshape y to be $\left(\prod_{j=i+1}^{d} n_j\right) \times n_i \times m_{i-1}$
  perform an indexed multiplication where $y(h,k) \leftarrow y\left(h,:,I_i^y(k)\right) * P_i\left(:,I_i^p(k)\right)$
return y

Indexed EV evaluations

The previous algorithm does a full EV evaluation
    returns $E[V(S^+)|X]$ for all state/action combinations

We also require an efficient way to compute $E[V(S^+)|X]$ for a specific strategy

Let the strategy be defined by the index vector I^a (with length n_s)

If the space of conditioning variables for states 1-i is smaller than space of state variables
    expand to match the common conditioning variables
    otherwise expand to match the strategy

Define J_i^p to be an index that expands the columns of P_i to match those of the full X matrix
    Each J_i^p is a vector of length n_x (equals the # of rows of X)

Thus use I_i^y and I_i^p indices while they are smaller than the I_a vector ($m_i < n_s$)

Then expand the intermediate factor y_i and switch to indexing with $J_i^p(I^a)$

An additional index vector J_{i-1}^y must be defined where i is the loop index when the change from I to J indexing occurs to expand y_i

The number of arithmetic operations is $\sum_{i=1}^d \prod_{j=i}^d n_j \min(m_i, n_s)$ (recall that $n_s = \prod_{j=1}^d n_i$)

Contrast this with an indexed operation using $P[:, I^a]$ which uses n_s^2 arithmetic operations

Indexed EV evaluations (cont.)

A full EV function evaluation could be computed using the following algorithm:

```
set y = v
reshape y to be ∏_{j=2}^{d} n_j × n_1
set y ← y * p_1
set useI = true
loop from i = 2 to i = d
    if m_i < n_s
        reshape y to be (∏_{j=i+1}^{d} n_j) × n_i × m_{i−1} and expand y(:,:,k) ← y(h,:,J_{i−1}^y(I^a(k)))
        set useI = false
    if useI=true
        reshape y to be (∏_{j=i+1}^{d} n_j) × n_i × m_{i−1}
        perform an indexed multiplication where y(h,k) ← y(h,:,I_i^y(k)) * P_i(:,I_i^p(k))
    otherwise
        perform an indexed multiplication where y(h,k) ← y(h,:,k) * P_i(:,J_i^p(I^a(k)))
return y
```

An example

Suppose there are 3 state variables and 1 action variables

The state variable sizes are all n and the action is n_a

With the action in the last column of X the parents vectors are given by

$$q_1 = [1 \;\; 4] \qquad q_2 = [1 \;\; 2 \;\; 4] \qquad q_3 = [2 \;\; 3 \;\; 4]$$

The EV function is performed in 3 steps with operation counts

i	y_i	P_i	# of operations
1	$n^2 \times n \times 1$	$n \times nn_a$	$n^4 n_a$
2	$n \times n \times nn_a$	$n \times n^2 n_a$	$n^4 n_a$
3	$1 \times n \times nn_a$	$n \times n^2 n_a$	$n^4 n_a$

The total operation count is $3n^4 n_a$

If the full transition matrix is used the operations count is $n^6 n_a$

An indexed EV evaluation

With function iteration most EV evaluations are indexed

Suppose that $n < n_a < n^2$

A strategy index has length $n_s = n^3$

The I_i indices have sizes nn_a, $n^2 n_a$ and $n^2 n_a$

Hence the crossover from I to J indexing would occur in step 2

i	y_i	P_i	# of operations
1	$n^2 \times n \times 1$	$n \times nn_a$	$n^4 n_a$
2	$n \times n \times nn_a$	$n \times n^2 n_a$	n^5
3	$1 \times n \times nn_a$	$n \times n^2 n_a$	n^4

The total operation count is $n^4(n_a + n + 1)$

If the full transition matrix is used by extracting the appropriate columns of P: $P[:, I^a]$ the operation requires n^6 operations

Combining CPTs

Thus far we've considered operating on each of the P_i in a sequence of d operations

It may be better to combine some of the CPTs in a preprocessing step

For example suppose that

$$q_1 = [1 \;\; 2 \;\; 4] \qquad q_2 = [1 \;\; 2 \;\; 4]$$

The first two steps with P_1 and P_2 have operation counts

i	y_i	P_i	# of operations
1	$n^2 \times n \times 1$	$n \times n^2 n_a$	$n^5 n_a$
2	$n \times n \times n n_a$	$n \times n^2 n_a$	$n^4 n_a$

If we combine P_1 and P_2 in a preprocessing step to form P_{12} the same operation has

i	y_i	P_{12}	# of operations
1	$n \times n^2 \times 1$	$n^2 \times n^2 n_a$	$n^5 n_a$

Thus we can do both operations in a single step with the same operation count as the previous first step

Optimal management of operations

A natural approach is to minimize arithmetic operations
    but this may not be fastest or most memory efficient

Efficiency is influenced by:
    the sequence that the CPTs are processed
    the preprocessing of CPTs into groups
    the algorithms performing the arithmetic operations

<u>Sequencing</u>

Optimal sequencing is a difficult problem to solve
    there do not appear to be any polynomial algorithms

The sequence problem might be addressed using
    heuristics (e.g., greedy algorithm)
    global optimization methods (e.g., genetic algorithm)

Graph theoretic and matrix reordering methods might be helpful (?)

<u>Grouping</u>

Given an ordering the minimal operations grouping can be found in polynomial time

<u>Arithmetic operations</u>

Use of high performance algorithms (e.g., dgemm) might improve performance even with higher arithmetic operation count

Use of smaller factors might improve overall memory access speeds

Memory shuffling should be avoided if possible

Optimal grouping

Optimal grouping of operations can be solved using an $O(d^3)$ dynamic programming algorithm

The problem is similar to the well-known matrix chain multiplication problem:

$$A_1 * A_2 * \ldots * A_d$$

Given a variable order the cost of incorporating a CPT that groups variables i through $j \geq i$ is

$$C_{ij} = p_i m_j$$

where

$p_i = \prod_{k=i}^{d} n_k$ and

m_j is the number of tuples of the parents of variables 1 through j.

For each (i, j) we can evaluate whether breaking the grouped variables into two further groups results in a less costly set of operations:

$$M_{ij} = \min\left(C_{ij}, \min_{k \in \{0,\ldots,j-i+1\}} M_{i,i+k} + M_{i+k+1,j}\right)$$

The minimal cost grouping is given by M_{1d}.

This is optimal for a full evaluation.

For an indexed evaluation use

$$C_{ij} = p_i \min(m_j, n_s)$$

Optimal management of operations (cont.)

Optimal combined sequencing of operations is related to the sum-product (tensor contraction) ordering problem

Given n multidimensional arrays F_i indexed by a set of indices given by q_i compute

$$\boldsymbol{G}(\boldsymbol{r}) = \sum_{\boldsymbol{k} \in \cup_{\boldsymbol{i}} \boldsymbol{q}_{\boldsymbol{i}} \setminus \boldsymbol{r}} \prod_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{d}} \boldsymbol{F}_{\boldsymbol{i}}\,(\boldsymbol{k} \in \boldsymbol{q}_{\boldsymbol{i}})$$

In words, we multiple together the arrays, matching along any common dimensions, and then sum out the dimensions that are not desired in the output

For an EV function

factors are the CPTs for the state variables along with the V vector,
output indices are the current states and actions
summed out variables are the future states and (possibly) additional noise terms

Creating EV functions

P : set of d_s transition probability matrices (CPTs)

X : d_x column matrix of state/action combinations

q : set of d_s index vectors indicating the columns of X associated with each state variable

EVcreate creates an EV function

It first performs variable reordering and optimal grouping if requested

It then groups variables if requested

It then sets of index vectors (I and J) used to guide operations

Finally it creates a function that implements the sequential incorporation of each of the P_i

Controlling invasive species on a spatial network

Chades et al. (2011) "General rules for managing and surveying networks of pests, diseases, and endangered species"

N sites with an $N \times N$ adjacency matrix C

Each site is either occupied or empty and either treated or not treated:

O/T, O/N, E/T, and E/N

A single site can be treated each period

Transition probability for site i depends on whether it is
occupied or empty (S_i)
treated or not treated (A_i)
if empty & not treated on the # of occupied/untreated neighbors: $q_i = \sum_{j=1}^{N} C_{ij} S_j \,(1 - A_j)$

The transition matrix for site i can be represented by a $2 \times (4 + K_i)$ matrix

$$P_i = \begin{bmatrix} p_{ot} & p_{on} & p_{et} & p_{en}^0 & p_{en}^1 & \dots & p_{en}^{K_i} \\ 1 - p_{ot} & 1 - p_{on} & 1 - p_{et} & 1 - p_{en}^0 & 1 - p_{en}^1 & \dots & 1 - p_{en}^{K_i} \end{bmatrix}$$

where p_{en}^{j} is the probability of occupancy if currently empty and untreated with j occupied/untreated neighbors (up to K_i)

State space has size 2^N and there are $N + 1$ possible actions (including doing nothing)

There are therefore $(N + 1)2^N$ state/action combinations

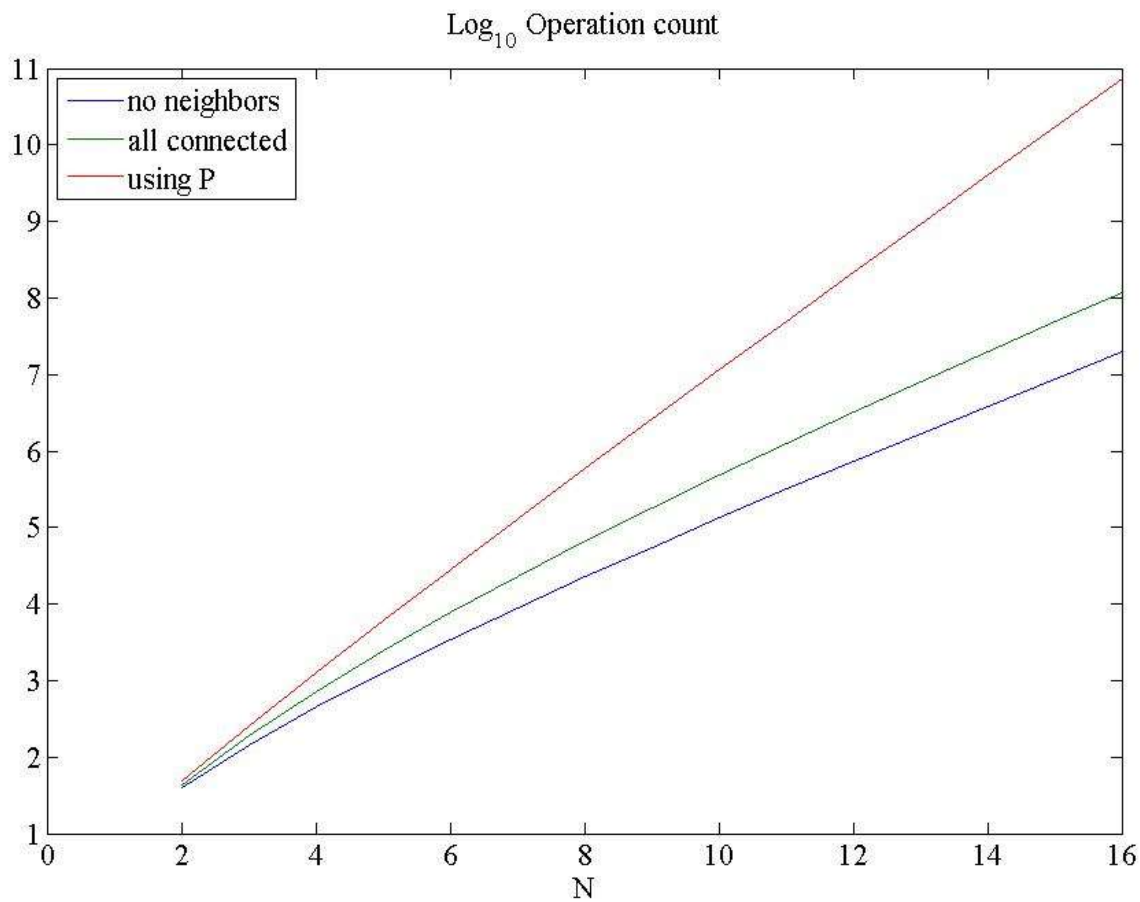
EV versus Transition Matrix

The operation count depends on the density of the network

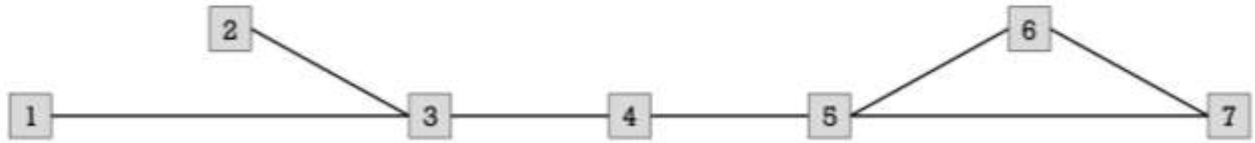
Range from all isolated to all connected

Operation count increases as network becomes more connected

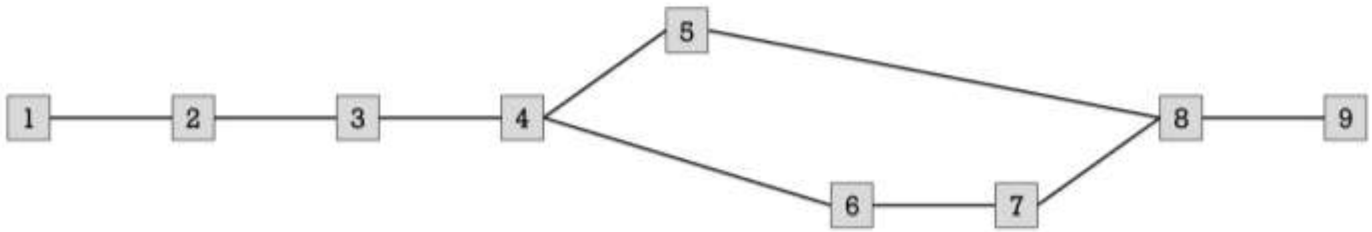
Even a fully connected network requires significantly less operations than using P



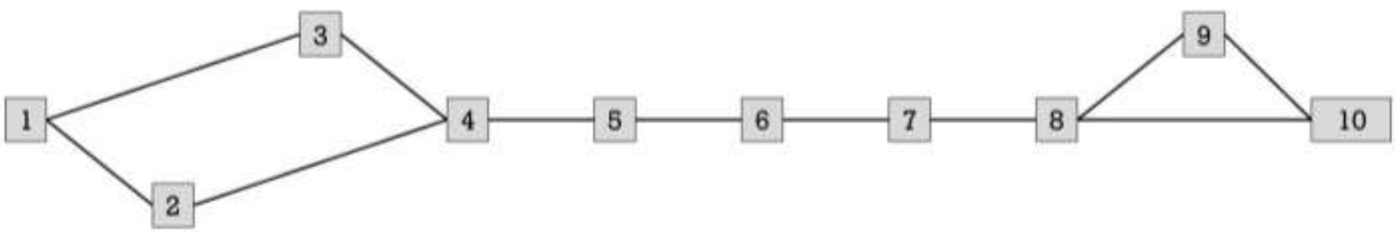
Network 1



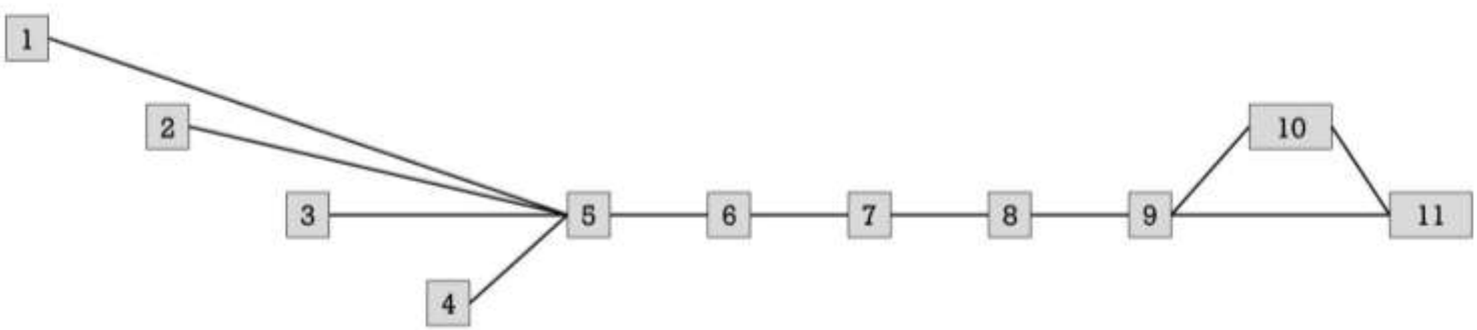
Network 2



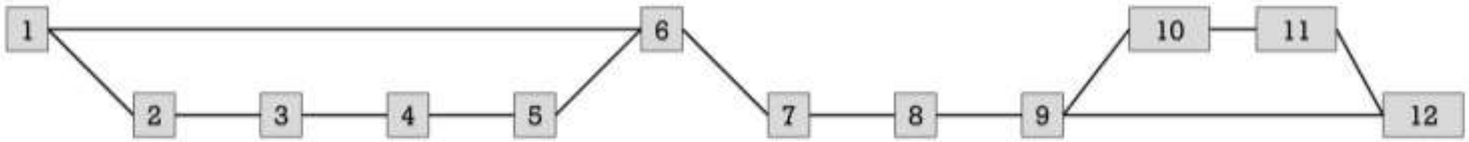
Network 3



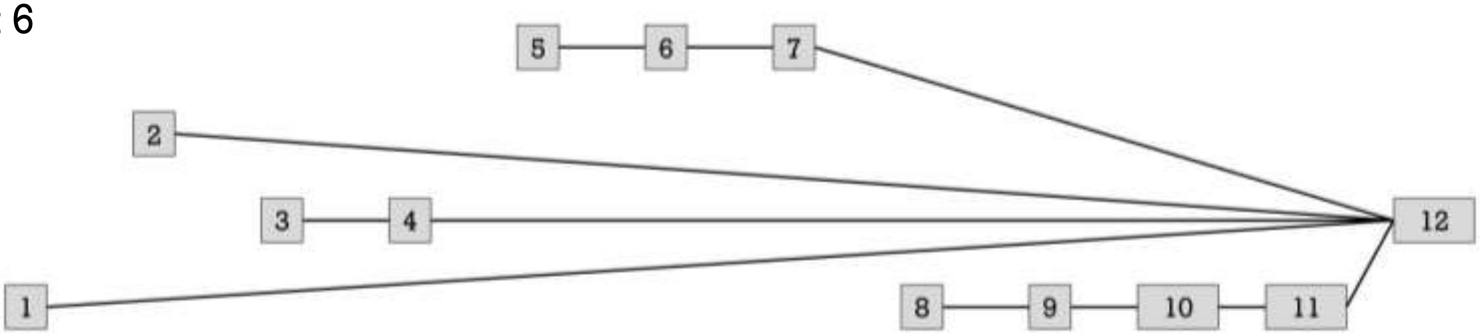
Network 4



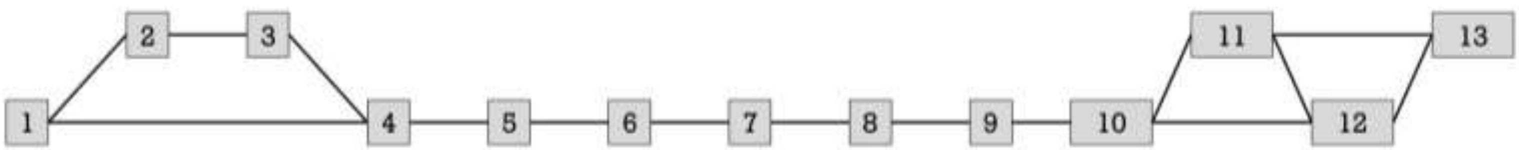
Network 5



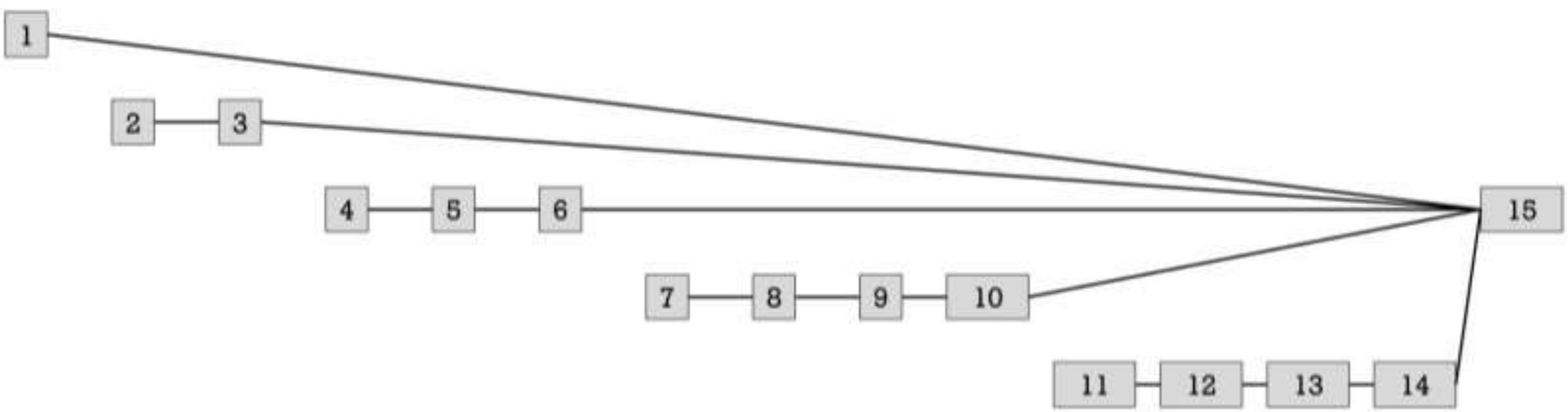
Network 6



Network 7



Network 8



Timing Results for Invasive Species Networks

Results with P, d sequenced EV and optimally grouped EV

		5 full evaluations			25 indexed evaluations		
network	N	P	EV	EV*	P	EV	EV*
1	7	0.0003	0.0092	0.0048	0.0162	0.0196	0.0084
2	9	0.0021	0.0130	0.0122	0.0069	0.0522	0.0356
3	10	0.0199	0.0287	0.0260	0.0666	0.1052	0.0732
4	11	0.0826	0.0627	0.0556	0.2933	0.2244	0.1478
5	12	0.3422	0.1406	0.1214	1.3913	0.5324	0.3304
6	12	0.3532	0.1699	0.1399	1.3858	0.6204	0.3477
7	13	1.8283	0.3049	0.2521	6.2127	1.1681	0.7290
8	15	NA	2.0905	1.2996	NA	7.0212	3.8731

Results with handpicked groupings with many, few and 2 factors

		5 full evaluations				25 indexed evaluations			
network	N	EV*	EV m	EV f	EV 2	EV*	EV m	EV f	EV 2
1	7	0.0048	0.0028	0.0056	0.0051	0.0084	0.0100	0.0076	0.0067
2	9	0.0122	0.0127	0.0124	0.0120	0.0356	0.0485	0.0356	0.0333
3	10	0.0260	0.0153	0.0356	0.0250	0.0732	0.0510	0.0447	0.0684
4	11	0.0556	0.0340	0.0390	0.0681	0.1478	0.1135	0.0946	0.1596
5	12	0.1214	0.0688	0.0618	0.1255	0.3304	0.2435	0.1783	0.3318
6	12	0.1399	0.1139	0.1201	0.1274	0.3477	0.3285	0.3280	0.3469
7	13	0.2521	0.1514	0.1468	0.3312	0.7290	0.4863	0.4441	0.8862
8	15	1.2996	1.2410	1.2149	2.0978	3.8731	3.8074	3.4394	5.4797

EV functions with transition functions

In a dynamic system the transition law can be written as

$$S^{+} = g(X, e)$$

where:

X represents the current state & action variables of the system
e represents a set of random noise terms with specified distributions

In factored form we have $S_i^{+} = g_i(X_i, e_i)$ where X_i and e_i are subsets of X and e

One approach to solving this sort of model is to discretize S, X and e and compute Conditional Probability Tables P_i for each S_i^{+}

MDPSolve implements this approach using linear interpolation weights as probabilities

The main issue that arises here is that when $e_i \cap e_j \neq \emptyset$ the CPTs are functions of the noise terms and are not conditional on X alone

The algorithms for merging CPTs in the EV functions would need to be modified to only sum out a noise variables once all future states conditional on that variables have been processed

Alternatively one could group the variables with common noise terms and compute the single CPT for the group

Creating EV functions from transition functions

g : set of d_s transition functions

X : d_x column matrix of state/action combinations

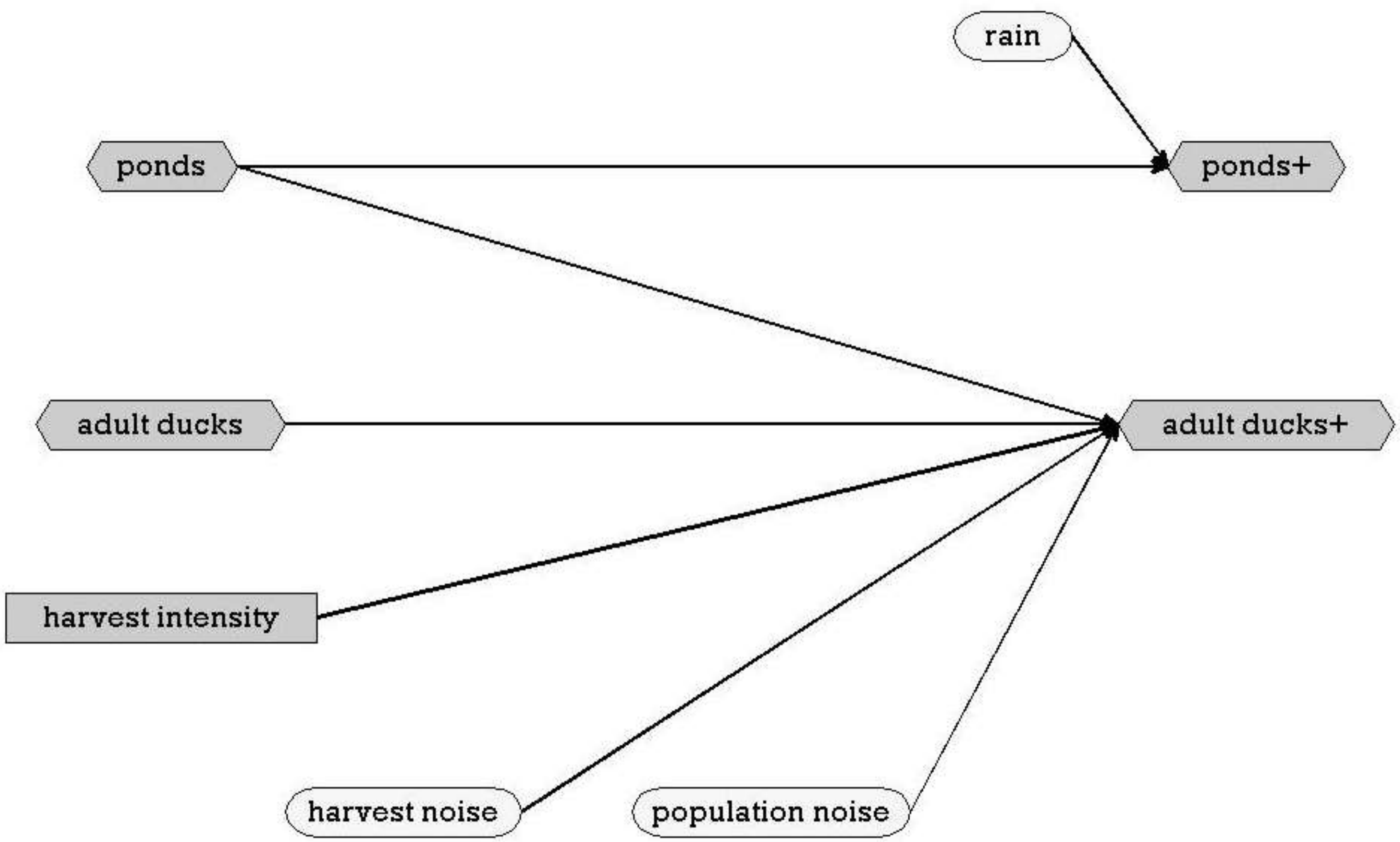
e : d_e element set of random variables

q : set of d_s index vectors indicating the columns of X and elements of e associated with each state variable

g2EV converts transition functions to transition matrices which are then be passed to EVcreate to create an EV function

Currently g2EV requires that variables have no common random noise terms in their conditioning sets

Mallard Duck Model



Central flyway mallard duck model used to set harvest levels by USFWS

State variables are associated with disjoint sets of noise variables

Mallard Duck Model: timing results

Using:

151 values of ponds

351 values for adult ducks

	matrix sizes	10 full evaluations	25 indexed evaluations
P	53001×212004	2.605	3.200
EV	151×151 & 351×212004	1.130	0.864

Using:

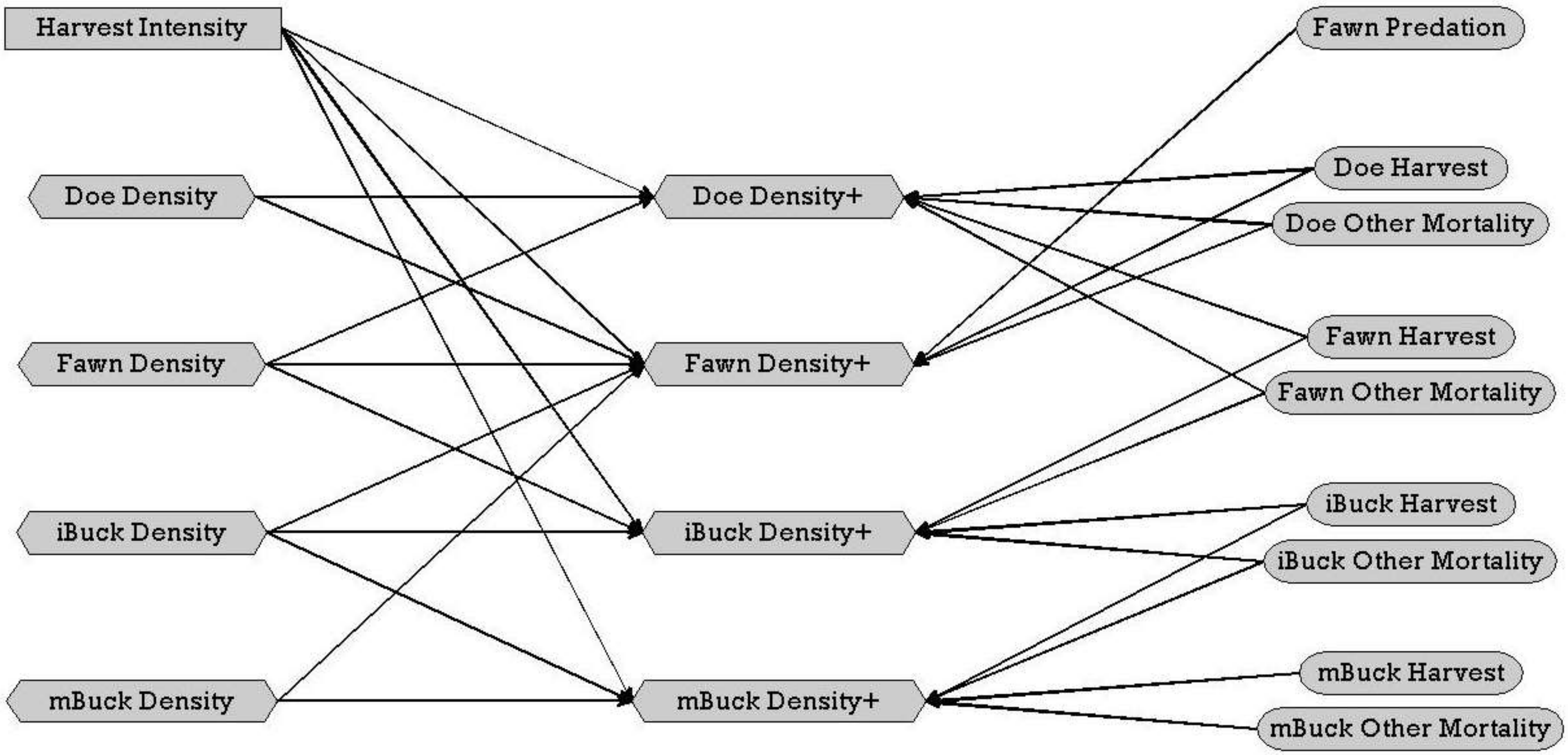
311 values of ponds

711 values for adult ducks

	matrix sizes	10 full evaluations	25 indexed evaluations
P	221121×884484	11.602	13.959
EV	311×311 & 711×884484	5.763	4.140

Full evaluations are about 2 times and indexed evaluations about 3-4 times as fast

Alabama Deer Model



Left hand variables: current states & actions
Middle variables: future states
Right hand variables: noise variables

Here the noise terms do not separate

EV function may be no better than full transition matrix

Operation count analysis

Notice that fawn predation and the doe and mature buck noise terms affect only 1 variable and can be incorporated into the CPTs

If we use the processing sequence mBuck, iBuck, Doe, Fawn and suppose that there are n_p values for each of the states, n_a actions and n_e values of the noise terms

The operation count for sequential processing will be

i	variable	y_i	P_i	# of operations
1	mBuck	$S_1^+ S_2^+ S_3^+ S_4^+$	$S_1^+ S_1 S_2 A e_1 e_2$	$n_p^6 n_a n_e^2$
2	iBuck	$S_2^+ S_3^+ S_4^+ S_1 S_2 A e_1 e_2$	$S_2^+ S_1 S_2 S_4 A e_1 e_2 e_3 e_4$	$n_p^6 n_a n_e^4$
3	Doe	$S_3^+ S_4^+ S_1 S_2 S_4 A$	$S_3^+ S_1 S_2 S_3 S_4 A e_3 e_4$	$n_p^6 n_a n_e^2$
4	Fawn	$S_4^+ S_1 S_2 S_3 S_4 A$	$S_4^+ S_1 S_2 S_3 S_4 A$	$n_p^5 n_a$

Contrast with $n_p^8 n_a$ operations with the full P matrix

The key operation here is number 2 with $n_p^6 n_a n_e^4$ operations. Typically $n_e \ll n_p$ so it is possible that this is less than $n_p^8 n_a$

Two changes might help:

combine the noise terms for each category

use a staged transition with an extra juvenile category

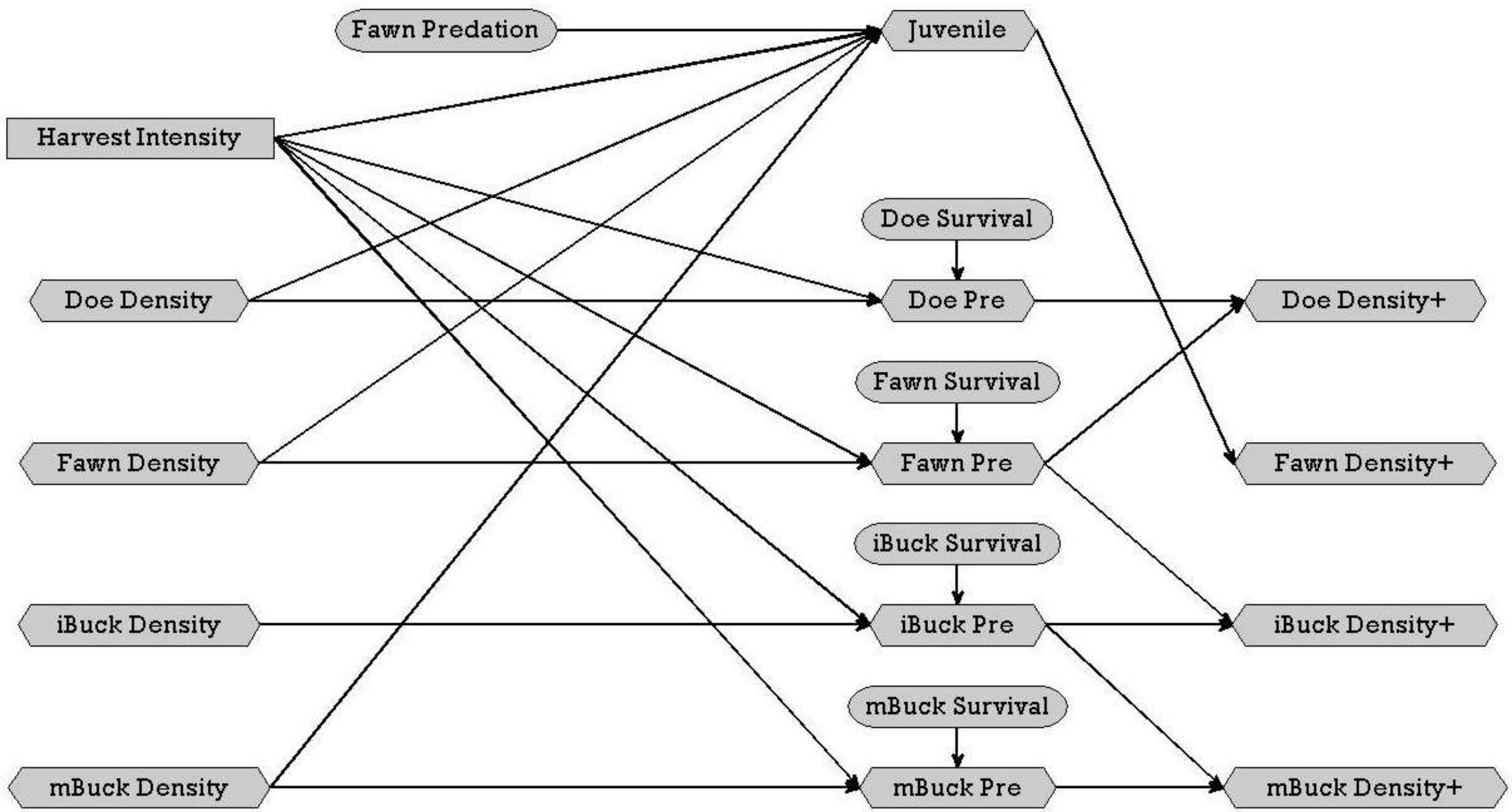
w/ stage 2 representing category change

An alternative approach

Define post-harvest categories for each age/stage class

Introduce new intermediate juvenile class

Use a 2-stage approach



Wrap Up

EV functions can replace the use of transition probability matrices

They use less memory and can be evaluated faster (sometimes by orders of magnitude)

Procedures to create EV functions will be incorporated into the next release of MDPSolve (or can be obtained from GitHub)

EV functions are especially advantageous in exploiting conditional independence in factored models

In factored models EV functions are evaluated in a sequence of indexed multiplication operations

Sequence of operations and groupings of operations in a preprocessing step matter

Optimal organization of operations is a difficult problem though some headway has been made

To Do

Extend the indexed multiplication approach to allow noise terms to be factored out during evaluation

Explore the optimal ordering (sequencing/grouping) issue more deeply

Perhaps use penalties on number of submatrices to encourage shorter sequences