Complexity Results for Serial Decomposability

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
email: byland@cis.ohio-state.edu

Abstract

Korf (1985) presents a method for learning macro-operators and shows that the method is applicable to serially decomposable problems. In this paper I analyze the computational complexity of serial decomposability. Assuming that operators take polynomial time, it is NP-complete to determine if an operator (or set of operators) is not serially decomposable, whether or not an ordering of state variables is given. In addition to serial decomposability of operators, a serially decomposable problem requires that the set of solvable states is closed under the operators. It is PSPACE-complete to determine if a given "finite state-variable problem" is serially decomposable. In fact, every solvable instance of a PSPACE problem can be converted to a serially decomposable problem. Furthermore, given a bound on the size of the input, every problem in PSPACE can be transformed to a problem that is nearly serially-decomposable, i.e., the problem is serially decomposable except for closure of solvable states or a unique goal state.

Introduction

Korf (1985) presents a method of learning macro-operators (hereafter called MPS for the Macro Problem Solver system that applied the method), defines a property of problems called serial decomposability, and demonstrates that serial decomposability is a sufficient condition for MPS's applicability.

However, several questions were left unanswered. How difficult is it to determine whether a problem is serially decomposable? How difficult is it to solve serially decomposable problems? Can MPS's exponential running time be improved? What kinds of problems can be transformed into serially decomposable problems?

Subsequent work has not addressed these questions in their full generality. Chalasani et al. (1991) analyze the "general permutation problem" and discuss how it is related to serial decomposability. In a permutation problem, the values of the state variables after applying an operator are a permutation of the previous values. Chalasani et al. show that this problem is in NP, but NP-completeness is open. Tadepalli (1991a, 1991b) shows how macro tables are polynomially PAC-learnable if the examples are generated using a macro table. In each case, a strong assumption (that operators are permutations or that a macro table generates the examples) is made. My results hold for serially decomposable problems in general.

Let a "finite state-variable problem" be a problem defined by a finite number of state variables, each of which has a finite number of values; a finite number of operators; and a unique goal state, i.e., an assignment of values to all state variables. A serially decomposable problem is a finite state-variable problem in which each operator is serially decomposable (the new value of a state variable depends only on its previous value and the previous values of the state variables that precede it), and in which the set of solvable states is closed under the operators (operator closure).

Assuming that operators take polynomial time, it is NP-complete to determine if an operator (or set of operators) is not serially decomposable, whether or not an ordering of state variables is given. It is PSPACE-complete to determine if a given finite state-variable problem is serially decomposable, primarily due to the difficulty of determining operator closure.[1] In fact, every solvable instance of a PSPACE problem can be transformed to a serially decomposable problem. Thus, MPS applies to all solvable instances of PSPACE problems, and MPS's exponential running time is indeed comparable to solving a single instance of a problem, assuming P≠PSPACE. Thus, any large improvement of MPS's running time is unlikely.

Furthermore, given a bound on the size of the input, every problem in PSPACE can be transformed to a problem that is "nearly serially-decomposable," i.e., serially decomposable except for either operator

[1] PSPACE is the class of problems solvable in polynomial space. PSPACE-complete is the set of hardest problems in PSPACE. All evidence to date suggests that PSPACE-complete is harder than NP-complete, which in turn is harder than P (polynomial time).

closure or a unique goal state. That is, the representation of any PSPACE problem with limited-size input can be changed to one in which the operators are serially decomposable, but some other property of serial decomposability is not satisfied. It should be noted that MPS is not guaranteed to work without operator closure or without a unique goal state. An open issue is determining the class of problems that can be efficiently transformed into "fully" serially-decomposable problems.

These results show that serial decomposability is a nontrivial property to detect and achieve. Of special interest is that detecting serial decomposability of operators is relatively easy in comparison to detecting operator closure. Also, except for special cases (Chalasani *et al.*, 1991), it is not clear how to transform a problem so that all properties of serially decomposability are attained. Another open question is the difficulty of determining whether an instance of a serially decomposable problem is solvable.

The above summarizes the technical results. The remainder of the paper recapitulates Korf's definition of serial decomposability, gives an example of how blocks-world operators can be transformed into serially decomposable operators, and demonstrates the new results.

Serial Decomposability

The following definitions are slightly modified from Korf (1985).[2]

A *finite state-variable problem* is a tuple $\langle S, V, O, g \rangle$ where:

S is a set of n *state variables* $\{s_1, s_2, \ldots, s_n\}$;

V is a set of h *values*; each *state* $s \in V^n$ is an assignment of values to S;

O is a set of *operators*; each operator is a function from V^n to V^n; and

g is the *goal state*, which is an element of V^n.

A *serially decomposable problem* is a finite state-variable problem $\langle S, V, O, g \rangle$ where each operator is serially decomposable, and the set of solvable states is closed under the operators (operator closure). Both conditions are defined below.

[2] I avoid Korf's definition of "problem" because every such "problem" is finite, and so conflicts with the usage of "problem" in computational complexity theory. His "problem" is equivalent to my "finite state-variable problem" with the additional constraint of operator closure.

He also includes all solvable states in his definition of "problem," which could lead to a rather large problem description. I include "state variables" and their "values," instead.

Finally, he assumes that every instance of a serially decomposable problem has a solution. I modify this assumption to operator closure.

An operator o is *serially decomposable* if there exist functions f_i, $1 \leq i \leq n$ such that

$$o(s_1, s_2, s_3, \ldots, s_n) = (f_1(s_1), f_2(s_1, s_2), \ldots, f_n(s_1, s_2, s_3, \ldots, s_n))$$

That is, the new value of each s_i is only dependent on the previous values of s_1 through s_i.

An instance of a serially decomposable problem specifies an initial state s_{init}. A solution to such an instance is a finite sequence of operators $(o_1, o_2, \ldots, o_k)$ such that $o_k(\ldots(o_2(o_1(s_{init})))) = g$. The same operator can be used multiple times within a sequence. A state s is *solvable* if there is a solution when s is the initial state. A serially decomposable problem satisfies *operator closure* if the set of solvable states is closed under the operators, i.e., $o(s)$ is always a solvable state whenever s is.

Korf (1985) shows that a macro table exists for every serially decomposable problem, and describes the Macro Problem Solver (MPS) for learning macro tables. The definitions of macros, macro tables, and MPS are not essential to the results of this paper, though I should note that without operator closure or a unique goal state, MPS is not guaranteed to work. See Korf (1985) for more details.

Every serially decomposable problem is finite. With a finite number of state variables and a finite number of values, there are a finite number of states. So, to consider the asymptotic complexity of serial decomposability, it is necessary to consider the set of all serially decomposable problems. After an example of how the block-world problem relates to serial decomposability, I demonstrate the complexity of determining whether an operator is serially decomposable, the complexity of determining whether a finite state-variable problem is serially decomposable, and the transformation from PSPACE problems with limited size inputs to problems that are nearly serially-decomposable.

Blocks-World Example

This section provides an example of the nature of my results and the proofs underlying them. The blocks world problem is an example of a problem that does not have serially decomposable operators. This is because the dependencies from pre- to postconditions in operators have circularities. Any solvable blocks-world instance, however, can be transformed into a serially decomposable problem. The trick is assert inconsistency if an operator is applied when its preconditions are false, and permit the effects of an operator whether or not its normal preconditions are true. Unfortunately, it is difficult to transform a set of blocks-world instances into a single problem without giving up operator closure or a unique goal state.

Consider the Sussman anomaly. In this blocks-world instance, there are three blocks A, B, and C. Initially C is on A, A is on the table, and B is on the table. The goal is to have A on B and B on C. For every

block, let there be two variables to indicate whether the block is clear or on the table, respectively, e.g., *clear-A* and *A-on-table*. For every pair of blocks, let there be a variable to indicate whether they are on one another, e.g., *A-on-B*. Finally, include another variable *ok*. The initial state sets the block variables to 0 and 1 as appropriate. The goal is *A-on-B* = 1 and *B-on-C* = 1. As is well-known, the goal implies specific values for all other state variables.

Now consider the following operator to stack *A* from *B* to *C*:

if *A-on-B* = 1 ∧ *clear-A* = 1 ∧ *clear-C* = 1
  then *A-on-B* ← 0
    clear-B ← 1
    clear-C ← 0
    A-on-C ← 1

As given, this operator is not serially decomposable because both *A-on-B* and *clear-C* appear in the preconditions and are changed, i.e., they depend on each other. The complexity result for determining serial decomposability of operators depends on a nondeterministic way to find this kind of dependency.

However, with an additional variable, this operator can be transformed into a serially decomposable operator. Let *ok* be the new state variable with *ok* = 1 in the initial and goal state. The transformed operator is:

if *A-on-B* = 0 ∨ *clear-A* = 0 ∨ *clear-C* = 0
  then *ok* ← 0
A-on-B ← 0
clear-B ← 1
clear-C ← 0
A-on-C ← 1

If the original preconditions are true, then this operator has the effect of the previous operator; otherwise *ok* = 0 and the effect is an inconsistent state representation. The ordering that makes this serially decomposable is that *ok* comes after all other variables.

If the other operators are similarly transformed, then the Sussman anomaly can be transformed into a serially decomposable problem. The unique goal state is the goal state of the Sussman anomaly. Operator closure can be achieved by including an operator that resets the other variables to the initial state, so that any state can reach the goal state. However, if this transformation is applied to an unsolvable block-world instance, then the reset operator ensures that solvable states can reach the unsolvable initial state, i.e., ensures that operator closure, and thus serial decomposability, is not satisfied. The complexity result for determining serial decomposability of a finite state-variable problem depends on this kind of transformation.

Can more than one blocks-world instance (i.e., with different initial states and goal states) fit into a single serially decomposable problem? No simple fix to the above formulation appears to work. If a set of state variables are used for remembering the initial state and encoding the goal state, a unique goal state can be achieved only if these variables are "erased." However, it is not possible to be sure that the goal state is reached before erasing these variables—that would violate serial decomposability. For example, the variables encoding the goal state would be used to set another state variable indicating that the goal state has been achieved, so the goal state variables must be ordered before the goal-state-achieved variable. However, serial decomposability means that erasing the goal state variables cannot depend on the value of goal-state-achieved variable. The problem transformation result meets a similar fate. It is possible to ensure serially decomposable operators and either operator closure or a unique goal state, but not both.

Serial Decomposability of Operators

Let SD-OPERATOR-GIVEN be the problem of determining whether a given operator of a given finite state-variable problem is serially decomposable for a given ordering of state variables. Let SD-OPERATOR-ANY be the problem of whether a given operator is serially decomposable for any ordering of state variables. I assume that applying an operator takes polynomial time.

Theorem 1 *SD-OPERATOR-GIVEN is coNP-complete.*

Proof: First, I show that SD-OPERATOR-GIVEN is in coNP. To demonstrate that an operator is not serially decomposable, nondeterministically choose two states in which state variables s_1 up to s_i have the same values, but after applying the operator, the two new states have different values for s_1 to s_i, implying that some variable in s_1 to s_i depends on some variable in s_{i+1} to s_n. If no such two states exists, then it must the case that no s_1 to s_i depends on s_{i+1} to s_n.

To show that this problem is coNP-hard, I reduce from SAT to SD-OPERATOR-GIVEN so that a yes (no) answer for the SD-OPERATOR-GIVEN instance implies a no (yes) answer for the SAT instance.

Let F be a boolean formula. Let $\{u_1, \ldots, u_m\}$ be the m boolean variables used in F. Create an SD-OPERATOR-GIVEN instance as follows Let $S = \{s_0, s_0', s_1, \ldots, s_m\}$ be $m+2$ state variables in order with $V = \{0, 1\}$ as possible values. Now define an operator o as:

if $u_i = s_i, 1 \leq i \leq m$, satisfies F
  then exchange values of s_0 and s_0'

If F is not satisfiable, then o is simply the identity function, which is serially decomposable. Otherwise, the new values for s_0 and s_0' depend on each other and on whether the values of the other variables satisfy F; hence, o is not serially decomposable for this ordering of state variables. Thus, SD-OPERATOR-GIVEN is coNP-hard. Because it is also in coNP, it follows that SD-OPERATOR-GIVEN is coNP-complete. □

Note the above reduction works for total decomposability as well. Therefore, it follows that:

Corollary 2 *It is coNP-complete to determine whether a given operator of a given finite state-variable problem is totally decomposable.*

Theorem 3 *SD-OPERATOR-ANY is coNP-complete.*

Proof: The reduction in Theorem 1 shows that SD-OPERATOR-ANY is coNP-hard because if the formula F in the proof is satisfiable, then s_0 and s'_0 depend on each other, and so no ordering of the state variables can satisfy serial decomposability.

It remains to show that SD-OPERATOR-ANY is in coNP. To demonstrate that s_i cannot be ordered before s_j, choose two states in which all state variables have the same values except for s_j, but after applying the operator, the two new states have different values for s_i. If two such states exist, then it must be the case that s_i depends on s_j, i.e, s_j must be ordered before s_i. If no such pair of states exists, then s_i must be a function of the other variables, i.e., s_i can be ordered before s_j.

To demonstrate that no ordering exists, choose two states for each $\langle i, j \rangle$, $i \neq j$, and based on the pairs that cannot be ordered, show that no partial ordering exists. This requires $O(n^3)$ nondeterministic choices—$2n$ choices for each of $O(n^2)$ pairs of state variables—and so SD-OPERATOR-ANY is in coNP. Because it is also coNP-hard, it follows that SD-OPERATOR-ANY is coNP-complete. □

The above results easily generalize to determining serial decomposability of a set of operators. For Theorem 1, this just involves a nondeterministic choice of an operator. For Theorem 3, a choice of operator must be made for each ordered pair of state variables. The interesting consequence is that detecting serial decomposability of a set of operators for any ordering of state variables is not much harder than detecting serial decomposability of a single operator for a given ordering. The difference is only a polynomial factor rather than a more dramatic change in complexity class.

Serial Decomposability of Problems

Let SDSAT be the problem of determining whether a given finite state-variable problem is serially decomposable. That is, an instance of SDSAT is a tuple $\langle S, V, O, g \rangle$ with definitions as given above. I again assume that applying an operator takes polynomial time.

Theorem 4 *SDSAT is PSPACE-complete.*

Proof: To show this, first I shall show that SDSAT is in PSPACE, and then I shall show that the set of Turing machines whose space is polynomially bounded can be reduced to SDSAT.

Lemma 5 *SDSAT is in PSPACE.*

Proof: Theorems 1 and 3 show that determining serial decomposability of an operator is in coNP.

Determining operator closure is in PSPACE if determining whether a state is solvable is in PSPACE. To show that an instance of SDSAT does not have operator closure, find two states s and s' such that s' and the goal state g can be reached from s, but g cannot be reached from s'. This would show that a unsolvable state s' can be reached from a solvable state s.

Determining whether a state s is solvable is in PSPACE. If there are n variables and h values, and there is a solution, then the length of the smallest solution path must be less than h^n. Any solution of length h^n or larger must have "loops," i.e., there must be some state that it visits twice. Such loops can be removed, resulting in a solution of length less than h^n. Hence, no more than h^n nondeterministic choices are required. Each operator only requires polynomial time, which implies polynomial space. Once an operator has been applied, its space can be reused for the next operator. Thus, determining whether a state is solvable is in NPSPACE. Because NPSPACE = PSPACE, this problem is also in PSPACE.

Because each property of serial decomposability can be determined in PSPACE, it follows that SDSAT is in PSPACE. □

Lemma 6 *SDSAT is PSPACE-hard.*

Proof: Let M be a Turing machine with input $x_1 x_2 \ldots x_n$ and whose space is bounded by a polynomial $p(n)$. Without loss of generality, I assume that M is a one-way Turing machine with a single tape. Because NPSPACE=PSPACE, it does not matter whether the machine is deterministic or not.

Create state variables with possible values $V = \{0, 1\}$ as follows:

- $s_{i,x}$ equals 1 if the ith tape square contains symbol x, for $0 \leq i < p(n)$ and $x \in \Sigma$, where Σ is the tape alphabet of M
- s_q equals 1 if M is in state q, for $q \in Q$, where Q is the set of states of M
- s_i equals 1 if the tape head is at the ith tape square
- $s*$ equals 1 if M accepts the input
- $s\#$ equals 1 if the other variables encode a coherent configuration of M

Create operators as follows. Let o_{init} set the state variables so they correspond to the initial configuration of M. For $1 \leq i \leq n$, $s_{i,x_i} = 1$. For $i = 0$ and $n < i < p(n)$, $s_{i,b} = 1$, where $b \in \Sigma$ is the blank symbol. $s_{q_0} = 1$, where q_0 is the start state. Also, $s_1 = 1$ and $s\# = 1$. All other state variables are set to 0, including $s*$.

A state transition of M is specified by (q, x, y, q', d), i.e., if q is the current state and x is the current symbol, then change x to y, make q' the current state, and move in direction d (−1 or +1). For each transition t and tape square i, create an operator $o_{t,i}$ that performs the following code:

if $s_q = 0 \vee s_i = 0 \vee s_{i,x} = 0$
  then $s\# \leftarrow 0$
$s_q \leftarrow 0$
$s_{i,x} \leftarrow 0$
$s_{i,y} \leftarrow 1$
$s_{q'} \leftarrow 1$
$s_i \leftarrow 0$
$s_{i+d} \leftarrow 1$

That is, if Turing machine M is not in state q with tape head at square i containing symbol x, then applying this operator results in a incoherent configuration and so 0 is assigned to $s\#$. o_{init} needs to be applied again to achieve $s\# = 1$. In any case, set other state variables as if M were at state q at tape square i containing symbol x. Special operators can be encoded for the ends of the tape. Note that all the $o_{t,i}$ operators are serially decomposable if $s\#$ is ordered after all other state variables (except for $s*$ because of an operator described below).

The reader might be uncomfortable with the large number of operators (number of tape squares times number of transitions) that are created. This can be avoided if the s_i variables are allowed to range from 0 to $p(n) - 1$, and modifications are made as follows. o_{init} sets $s_i = i$, $0 \leq i < p(n)$, and only one o_t operator is created for each transition. o_t looks for the s_i equal to 1 to determine which $s_{i,x}$, $s_{i,y}$, and s_{i+d} variables to change. To move the tape head, o_t either adds to or subtracts 1 from all the s_i variables (modulo $p(n)$) depending on the direction. This modification requires that all the s_i variables be ordered first.

Finally, for M's accepting states, create an operator o_{final} that sets all state variables to 0 except that $s*$ is set to 1 if $s\# = 1$ and if an accepting configuration has been reached. The goal state is then $s* = 1$ and all other variables equal to 0.

All operators are serially decomposable and there is a unique goal state. Suppose that M halts in an accepting configuration. Then the goal state can be reached from any other state by applying o_{init}, followed by operators corresponding to the transitions of M, followed by o_{final}. Thus operator closure is satisfied, and the finite state-variable problem is serially decomposable.

Suppose M does not halt in an accepting configuration. Then the goal state cannot be reached after o_{init} is applied. Because the goal state is a solvable state, and applying o_{init} leads to an unsolvable state, operator closure is not satisfied, and the finite state-variable problem is not serially decomposable.

By inspection, this reduction takes polynomial time. Thus, an algorithm for SDSAT could be used to solve any problem in PSPACE with polynomial additional time. It follows that SDSAT is PSPACE-hard. □.

Because SDSAT is in PSPACE and is PSPACE-hard, SDSAT is PSPACE-complete. □

It is interesting to note that if the PSPACE Turing machine instance is satisfiable, then the SDSAT instance is not only serially decomposable, but also every state is solvable.

Problem Transformation

The above proof transforms a solvable instance of a PSPACE Turing machine to a serially decomposable problem, but different solvable instances would be transformed to different serially decomposable problems. Of course, transforming a single infinite problem into a finite serially decomposable problem is out of the question, but it might be possible to transform finite subsets of an infinite problem to a serially decomposable problem.

There is an unsatisfying answer based on boolean circuit complexity. It is possible to encode a limited-size version of any polynomial-time problem into a polynomial-size boolean circuit. Such circuits have no circularities, and so, can easily be transformed into a serially decomposable problem. This might be considered uninteresting because only one operator is required to propagate values from one level of the circuit to the next.

In the remainder of this section, I show how a PSPACE Turing machine, given a limit on the size of the input, can be transformed in polynomial time to a problem that is nearly serially-decomposable, i.e., serially decomposable except for either operator closure or a unique goal state. It is an open question whether or not a transformation to "fully" serially decomposable problems is possible, but I shall suggest some reasons why such a transformation probably does not exist.

For a Turing machine M, let M_n denote M with size of input limited by n.

Theorem 7 *For any PSPACE Turing machine M and positive integer n, it takes time polynomial in n to reduce M_n to a problem that is serially decomposable except for either operator closure or a unique goal state.*

Proof: M_n uses at most polynomial space $p(n)$. Transform M for an arbitrary input $x_1 \ldots x_n$ of size n into a finite state-variable problem as in Lemma 6 with the following modifications.

Add $s'_{i,x}$ variables, $1 \leq i \leq n$, such that $s'_{i,x_i} = 1$ and 0 otherwise.

Modify the o_{init} operator so it copies the $s'_{i,x}$ variables to the $s_{i,x}$ variables.

Modify the o_{final} operator so it changes the $s'_{i,x}$ variables to zero.

Different inputs for M_n can now be accommodated by initializing the $s'_{i,x}$ variables to different values. The operators remain serially decomposable as long as the $s'_{i,x}$ variables are ordered before the $s_{i,x}$ variables. However, operator closure is not satisfied. The o_{final} operator irrevocably erases the $s'_{i,x}$ variables, and the Turing machine can no longer be initialized to any proper configuration. Thus, this finite state-variable

problem is serially decomposable except for operator closure.

Suppose that the finite state-variable problem is changed as follows: any state in which $s* = 1$ is a goal state, and o_{final} does not change any of the $s'_{i,x}$ variables. Now the problem has operator closure because no operator erases the initial sate; however, there is no longer a unique goal state. Thus, this problem is serially decomposable except for a unique goal state. □

It is very unlikely that the above transformation results in a "natural" representation. However, it suggests that many PSPACE problems are likely have a "natural" counterpart with serially decomposable operators, which might take some effort to find.

There is a good argument against the existence of a transformation from PSPACE or even NP to "fully" serially decomposable problems.

Operator closure suggests that either operators never make a decision that requires backtracking, or that it is always possible to restore the initial state. NP-hard problems appear to require backtracking, so that leaves restoring the initial state as the only possibility, which implies that the state variables always remember the initial state. But a unique goal state means that there is no memory of any specific state when the goal is reached. In addition, serial decomposability of operators means that information can only "flow" in one direction—from s_1 to s_n. The variables used to restore the initial state must be before all other variables, including any that indicate whether a solution has been found. As a result, erasing the initial state cannot depend on whether a solution has been found. So restoring the initial state is not a viable possibility, either. Somehow, applying an operator never loses the information that the instance is solvable.

This is the opposite what seems necessary for NP-hard problems. For example, consider the problem of satisfying a boolean formula. To transform this problem to a serially decomposable problem, there must be operators that erase the original formula in order to reach a common goal state from any formula. However, if other operators decide on assignments to variables, and if incorrect assignments have been made, then erasing the original formula results in an unsolvable state.

Conclusion

I have shown that it is hard (coNP-complete) to determine serially decomposability of operators, and even harder (PSPACE-complete) to determine serially decomposability of problems. Thus, it is nontrivial to test for serial decomposability, especially to test for operator closure. I have also shown that an efficient transformation from PSPACE Turing machines to nearly serially-decomposable problems exist, given a limit on the size of the input. That is, it is easy to achieve serial decomposability of operators, albeit without operator closure or a unique goal state. A problem instance can be transformed into a serially decomposable problem, but it is not clear when a set of problem instances can be transformed into a single serially decomposable problem. Another major question that is left open is: If a problem is known to be serially decomposable, how difficult is it to determine whether a given instance is solvable?

Acknowledgments

The reviewers made valuable suggestions. This research has been supported in part by DARPA/AFOSR contract F49620-89-C-0110.

References

Chalasani, P.; Etzioni, O.; and Mount, J. 1991. Integrating efficient model-learning and problem-solving algorithms in permutation environments. In *Proc. Second Int. Conf. on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts. 89–98.

Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Tadepalli, P. 1991a. A formalization of explanation-based macro-operator learning. In *Proc. Twelfth Int. Joint Conf. on Artificial Intelligence*, Sydney. 616–622.

Tadepalli, P. 1991b. Learning with inscrutable theories. In *Proc. Eighth Int. Workshop on Machine Learning*, Evanston, Illinois. 544–548.