RESEARCH REVIEW JOURNALS
Volume-04
Issue-05
May-2019

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com[UGC Listed Journal]

A Study of Reasoning System towards Artificial Intelligence in Mathematical Modeling

[1]Lippon Kumar Choudhary & [2]Dr. Shashank Swami

[1]Research Scholar, OPJS University, Churu, Rajasthan (India)
[2]Assistant Professor, OPJS University, Churu, Rajasthan (India)

ARTICLE DETAILS

Article History
Published Online: 25 May 2019

Keywords
Reasoning System, Artificial Intelligence, Mathematical Modeling, problem-solving techniques.

ABSTRACT

Artificial intelligence (AI) is as much a branch of computer science as are its other branches, which include numerical methods, language theory, programming systems, and hardware systems. While computational mechanics has benefited from, and closely interacted with, the latter branches of computer science, the interaction between computational mechanics and AI is still in its infancy. Artificial intelligence encompasses several distinct areas of research each with its own specific interests, research techniques, and terminology. These sub-areas include search technologies, knowledge representation, vision, natural language processing, robotics, machine learning, and others.A host of ideas and techniques from AI have the potential to impact the practice of mathematical modeling. In particular, knowledge-based systems and environments can provide representations and associated problem-solving methods that can be used to encode domain knowledge and domain-specific strategies for a variety of ill-structured problems in model generation and result interpretation. Advanced AI programming languages and methodologies can provide high-level mechanisms for implementing numerical models and solutions, resulting in cleaner, easier to write, and more adaptable computational mechanics codes. A variety of algorithms for heuristic search, planning, and geometric reasoning can provide effective and rigorous mechanisms for addressing problems such as shape description and transformation, and constraint-based model representation. Before discussing the applications of AI in mathematical modeling, we briefly review knowledge-based expert systems and problem-solving techniques.

1. Introduction

In information technology a reasoning system is a software system that generates conclusions from available knowledge using logical techniques such as deduction and induction. Reasoning systems play an important role in the implementation of artificial intelligence and knowledge-based systems.

By the everyday usage definition of the phrase, all computer systems are reasoning systems in that they all automate some type of logic or decision. In typical use in the Information Technology field however, the phrase is usually reserved for systems that perform more complex kinds of reasoning. For example, not for systems that do fairly straightforward types of reasoning such as calculating a sales tax or customer discount but making logical inferences about a medical diagnosis or mathematical theorem. Reasoning systems come in two modes: interactive and batch processing. Interactive systems interface with the user to ask clarifying questions or otherwise allow the user to guide the reasoning process. Batch systems take in all the available information at once and generate the best answer possible without user feedback or guidance.[1]

Reasoning systems have a wide field of application that includes scheduling, business rule processing, problem solving, complex event processing, intrusion detection, predictive analytics, robotics, computer vision, and natural language processing.

The first reasoning systems were theorem proverbs, systems that represent axioms and statements in First Order Logic and then use rules of logic such as modus ponens to infer new statements. Another early type of reasoning system were general problem solvers. These were systems such as the General Problem Solver designed by Newell and Simon. General problem solvers attempted to provide a generic planning engine that could represent and solve structured problems. They worked by decomposing problems into smaller more manageable sub-problems, solving each sub-problem and assembling the partial answers into one final answer. Another example general problem solver was the SOAR family of systems.

In practice these theorem provers and general problem solvers were seldom useful for practical applications and required specialized users with knowledge of logic to utilize. The first practical applications of automated reasoning were expert systems. Expert systems focused on much more well defined domains than general problem solving such as medical diagnosis or analyzing faults in an aircraft. Expert systems also focused on more limited implementations of logic. Rather than attempting to implement the full range of logical expressions they typically focused on modus-ponens implemented via IF-THEN rules. Focusing on a specific domain and allowing only a restricted subset of logic improved the performance of such systems so that they were practical for use in the real world and not merely as research demonstrations as most previous automated reasoning systems had been. The engine used for automated reasoning in expert systems was typically called inference engines. Those

used for more general logical inferencing are typically called theorem provers.[2]

With the rise in popularity of expert systems many new types of automated reasoning were applied to diverse problems in government and industry. Some such as case-based reasoning were off shoots of expert systems research. Others such as constraint satisfaction algorithms were also influenced by fields such as decision technology and linear programming. Also, a completely different approach, one not based on symbolic reasoning but on a connectionist model has also been extremely productive. This latter type of automated reasoning is especially well suited to pattern matching and signal detection types of problems such as text searching and face matching.

2. Knowledge-Based Expert Systems

A good standard definition of knowledge-based expert systems (KBES) is the following: knowledge-based expert systems are interactive computer programs incorporating judgment, experience, rules of thumb, intuition, and other expertise to provide knowledgeable advice about a variety of tasks.

Many computer-aided engineering professionals initially react to this definition with boredom and impatience. After all, conventional computer programs for engineering applications have become increasingly interactive. They have always incorporated expertise in the form of limitations, assumptions, and approximations, as discussed above, and their output has long ago been accepted as advice, not as "the answer" to a problem.

There is a need, therefore, to add an operational definition to distinguish the new wave of KBES from conventional algorithmic programs that incorporate substantial amounts of heuristics about a particular application area. The distinction should not be based on implementation languages or on the absolute separation between domain-dependent knowledge and generic inference engine. The principal distinction lies in the use of knowledge. A traditional algorithmic application is organized into two parts: data and program. An expert system separates the program into an explicit knowledge base describing the problem-solving knowledge and a control program or inference engine that manipulates the knowledge base. The data portion or context describes the problem and the current state of the solution process. Such an approach is denoted as knowledge based.

Knowledge-based systems, as a distinctly separate AI research area, are about a decade old. This decade of research has seen many changes in the importance placed on various elements of the methodology. The most characteristic change is methodological; the focus has shifted from application areas and implementation tools to architectures and unifying principles underlying a variety of problem-solving tasks.

In the early days of knowledge-based systems, the presentation and analysis were at two levels: 1) the primitive representation mechanisms (rules, frames, etc.) and their associated primitive inferencing mechanisms (forward and backward chaining, inheritance, demon firing, etc.), and 2) the problem description. Unfortunately, it turned out that the former descriptions are too low level and do not describe the kind of problem that is being solved, while the latter descriptions are necessarily domain specific and often incomprehensible and uninteresting for people outside the specific area of expertise.

3. Problem Solving

Many problem-solving tasks can be formulated as a search in a state space. A state space consists of all the states of the domain and a set of operators that change one state into another. The states can best be thought of as nodes in a connected graph and the operators as edges. Certain nodes are designated as goal nodes, and a problem is said to be solved when a path from an initial state to a goal state has been found. State spaces can get very large, and various search methods to control the search efficiency are appropriate.

Search reduction. This technique involves showing that problem's answer cannot depend on searching a certain node. There are several reasons this could be true: (a) No solution can be in the subtree of this node. This technique has been called "constraint satisfaction" and involves noting that the conditions that can be attained in the subtree below a node are insufficient to produce some minimum requirement for a solution. (b) A solution in another path is superior to any possible solution in the subtree below this node. (c) The node has already been examined elsewhere in the search. This is the familiar dynamic programming technique in operations research.

Problem reduction. This technique involves transforming the problem space to make searching easier. Examples of problem reduction include: (a) planning with macro operators in an abstract space before getting down to the details of actual operators; (b) means-end analysis, which attempts to reason backward from a known goal; and (c) sub-goaling,

which breaks down difficult goals into simpler ones until easily solved ones are reached.

Adaptive search techniques. These techniques use evaluation functions to expand the "next best" node. Some algorithms (A*) will expand the node most likely to contain the optimal solution. Others (B*) will expand the node that is most likely to contribute the most information to the solution process.

Using domain knowledge. One way to control the search is to add additional information to nongoal nodes. This information could take the form of a distance from a hypothetical goal, operators that may be usefully applied to it, possible backtracking locations, similarity to other nodes that could be used to prune the search, or some general goodness in formation.

4. Applications In Mathematical Modeling

Mathematical modeling is the activity devoted to the study of the simulation of physical phenomena by computational processes. The goal of the simulation is to predict the behavior of some artifact within its environment. Mathematical modeling subsumes a number of activities, as illustrated by Figure.

The following sections discuss the applications and potential impacts of AI technology on various mathematical modeling activities. The mathematical modeling activities presented include model generation, interpretation of numerical results, and development and control of numerical algorithms. Note that these activities are not independent, and this organization is used primarily to assist in the exposition of ideas.

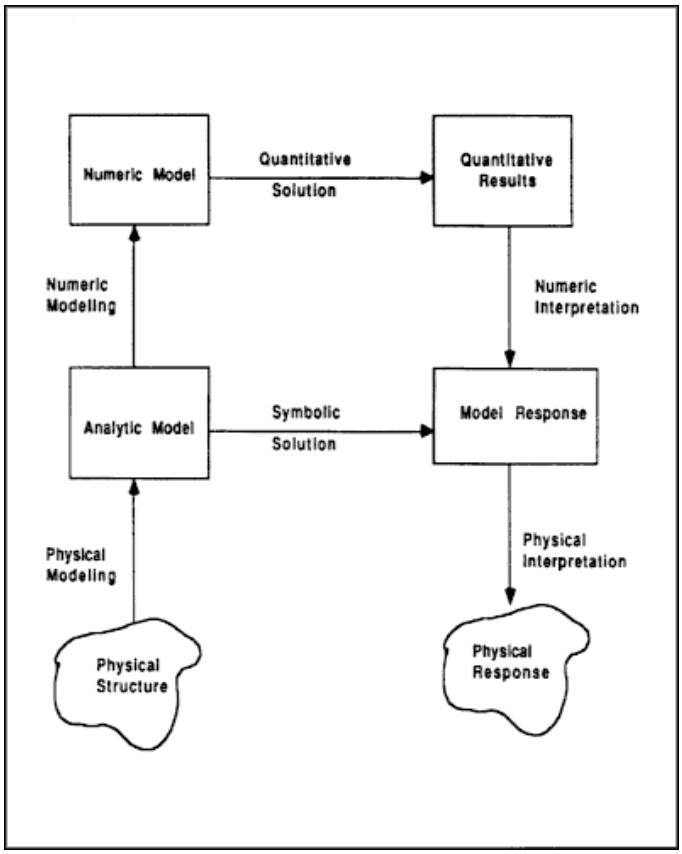

Figure Mathematical modeling process.

5. Model Generation

The term model generation is used to encompass all activities that result in the generation of models of physical systems suitable as input for a computational mechanics program. The generation of mathematical models from physical descriptions of systems is a problem of great practical importance. In all disciplines that use computational mechanics—aerospace, nuclear, marine, civil, and mechanics—there is a need to model an increasingly wider range of phenomena in all stages of system design from the earliest conceptual studies to the most detailed component performance evaluation. In addition, there is an urgent need for much closer integration of computational mechanics evaluations into computer-aided design (CAD) and to extend analyses to computer-aided manufacturing where there is a great interest in analyzing not just the finished components, structures, or systems but also the manufacturing processes themselves, such as casting, forging, or extrusion.

With the availability of literally hundreds of computational mechanics codes, including a large number of general-purpose finite element programs with a broad range of capabilities, model generation has become the primary activity of the analyst. However, in the current state of the art, the preparation of input data is a tedious, error-prone, and time-consuming process. Analysts are forced to interact with programs at a level much lower than the conceptual level at which they make decisions. Hence, there is a need for higher-level program interfaces that will free analysts from details, allowing them to generate models in terms of high-level behavioral descriptions, thereby increasing their productivity and improving the quality and reliability of their analyses.

Moreover, because of the very small number of experienced modelers who can confidently and reliably model physical problems and the increasing need for modeling expertise, it has also become increasingly important to capture and organize the modeling knowledge of experienced analysts and transfer it to the less experienced novice analysts. The methodology of AI and knowledge-based systems promises to provide an opportunity to respond to the needs identified above.

6. Model Interpretation

This section describes the AI potential to assist in postanalysis operations. Postanalysis operations are generically referred to as interpretation, although they involve distinctly different types of processes, including model validation, response abstraction, response evaluation, and redesign suggestions.

7. Model Validation

Model validation determines if the mathematical model's numerical results accurately reflect the modeled system's real behavior. Knowledge-based techniques provide practical mechanisms to represent and characterize one important class of possible errors—idealization errors. Recently, a framework to validate idealized models has been proposed such that if the system model is methodically generated through the application of a set of assumptions any idealization error can be traced to one or more of those generative assumptions. Furthermore, each assumption encodes the conditions under which it is valid; hence, model validation involves checking the validity conditions of individual assumptions. There are many ways of verifying assumptions ranging in complexity from the evaluation of simple algebraic expressions to analysis of a model that is more detailed than the original one.

8. Abstraction Of Numerical Results

Response abstraction is the task of generating some abstract description of the raw numerical results obtained from the analysis program. This description is presented in terms of high-level aggregate quantities representing key response parameters or behavior patterns.

Response abstractions can be classified in two types. The first, functional response abstractions, depends on the role that various subsystems or components play outside the model proper, that is, the meaningful aggregate quantities that are generated depend on knowledge of characteristics beyond the geometric and material properties of the system. The ability to generate the function-dependent response abstraction depends on the ability to represent the functional information that underlies the object modeled.

The second type of response abstraction is function independent. One seeks to recognize patterns, regularities, and interesting aspects and generate qualitative descriptions (e.g., stress paths) of the numerical results, independent of the functional nature of the object being modeled. Techniques from computer vision and range data analysis can be used to generate these interpretations. Well-developed vision techniques such as aggregation, segmentation, clustering classification, and recognition can be applied to the task. One interesting use of response abstractions assists the user in

checking the "physical reasonableness" of numerical results by comparing the response abstractions between more refined models and simpler models.

9. Conformance Evaluation

Conformance evaluation is the task of verifying that the computed results satisfy design specifications and functional criteria such as stress levels, ductility requirements, or deflection limitations. Conformance evaluation is largely a diagnostic problem, and the well-developed techniques for diagnosis can be applied to the task. Conformance evaluation requires heuristics on what are the (1) possible failure modes, (2) applicable code and standard provisions, (3) response quantities (stresses, deflections, etc.) affected by the provisions, and 4) approximations of the provisions and responses necessary.

A major issue in developing expert systems for conformance evaluation is the representation of code and standard provisions in a form suitable for evaluation yet amenable to modification when standards change or an organization wishes to use its own interpretations. One technique suitable for this purpose is to represent standards provisions by networks of decision tables.

10. Integration In Design

Analysis is rarely, if ever, an end in itself. The overwhelming use of analysis is to guide and confirm design decisions. One important application of AI techniques is to provide redesign recommendations when the analyzed system's response is not satisfactory. One problem that has to be addressed is the nature of the knowledge base that can generate redesign recommendations. Should it be separate and independent, or should it use the same modeling knowledge responsible for generating and interpreting models? The second approach formulates redesign as a goal-oriented problem: Given some deficiencies uncovered by an analysis, what modifications to the design object are required so that a model whose response satisfies the design specifications can be generated?

Another problem that has to be addressed if computational methods are to be adequately incorporated in CAD is the general capability to provide analysis interpretations and design evaluations compatible with the progress of the design process from the initial conceptual sketch to a fully detailed description. Evaluations should occur at increasingly higher degrees of refinement throughout the design process. Early simple models can provide early feedback to the designer (or design system) on the global adequacy of the design, while evolved models, paralleling the evolving design, help to guide the designer in the detailed design stages. An important issue in developing general mechanisms for hierarchical modeling is how to generate and represent various kinds of geometric abstractions.

11. Numerical Model Formulation

The term formulation denotes the process of producing a computational mechanics capability—a set of numerical routines—from a representation of a physical phenomenon. It is well known that the development of a computational mechanics program is time consuming, expensive, and error prone. Processes that can help in the quick development of reliable numerical software can be of great practical benefit. Ideas from AI can contribute significantly to various aspects of the formulation process: performing symbolic computations, expressing subroutines in a form that makes them reusable, designing large systems with appropriate data abstractions, assisting in the synthesis of computational mechanics programs, and integrating heuristics and knowledge-based methods into numerical solutions. These aspects are examined in turn.

12. Symbolic Processing

One aspect of program development that is particularly time consuming and error prone is the transition from a continuum model involving operations of differentiation and integration to a computational model involving algebraic and matrix operations. A branch of AI deals with symbolic computations, culminating in symbolic computation programs such as MACSYMA and Mathematica. Programs in this class operate on symbolic expressions, including operations of differentiation and integration, producing resulting expressions in symbolic form. A particularly attractive practical feature of these programs is that the output expressions can be displayed in Fortran source code format.

The potential role of symbolic processing has been investigated by several researchers. Studies indicate that symbolic processing can significantly assist the generation of computational model components to be incorporated in the source code. Symbolic generation of source code for clement stiffness and load matrices can eliminate the tedious and error-prone operations involved in going from a differential algebraic representation to a discrete procedural representation. Additional run-time efficiency improvements are possible through functionally optimized code and the use of exact integrations. Finally, conceptual improvements are possible, such as symbolically condensing out energy terms that contribute to shear and membrane locking.

13. Reusable Subroutines

Numerical subroutines that perform function evaluations, domain and boundary integrations, linear and nonlinear equation solving, etc., abound in computational mechanics codes. However, the typical implementation of these subroutines bears little resemblance to our mathematical knowledge of the operations they perform. They are written as a sequence of concrete arithmetic operations that include many mysterious numerical constants and are tailored to specific machines. Because these routines do not exhibit the structure of the ideas from which they are formed, their structure is monolithic, handcrafted for the particular application, rather than constructed from a set of interchangeable parts that represent the decompositions corresponding to the elemental concepts that underly the routine. Often numerical routines are difficult to write and even more difficult to read.

The idea of expressing mathematical routines constructively is widely applicable. Even the simplest routines that are of ten thought of as "atomic"—such as sin(x)—can be constructed from their primitive constituent mathematical operations, that is, periodicity and symmetry of the sine function and a truncated Taylor expansion. Abelson et al.7 shows how Romberg's quadrature can be built by

combining a primitive trapezoidal integrator with an accelerator that speeds the convergence of a sequence by Richardson extrapolation. The idea is that instead of writing a subroutine that computes the value of a function, one writes code to construct the subroutine that computes a value.

Such a formulation separates the ideas into several independent pieces that can be used interchangeably to facilitate attacking new problems. The advantages are obvious. First, clever ideas need to be coded once in a context independent of the particular application, thus enhancing the reliability of the software. Second, the code is closer to the mathematical basis of the function and is expressed in terms of the vocabulary of numerical analysis. Third, the code is adaptable to various usages and precisions because the routine's accuracy is an integral part of the code rather than a comment that the programmer adds; just changing the number that specifies the accuracy will generate the single, double, and quadruple precision versions of a subroutine.

Writing subroutines in this style requires the support of a programming language that provides higher-order procedures, streams, and other powerful abstraction mechanisms available in functional languages. The run-time efficiency does not necessarily suffer. The extra work of manipulating the function's construction need be done only once. The actual function calls are not encumbered. Moreover, because functional programs have no side effects, they have no required order of execution. This makes it exceptionally easy to execute them in parallel.

14. Programming With Data Abstractions

The current generation of computational mechanics software is based on programming concepts and languages that are two or three decades old. As attention turns to the development of the next generation of software, it is important that the new tools, concepts, and languages that have emerged in the interim be properly evaluated and that the software be built using the best appropriate tools.

Designs for finite element systems based on object-oriented concepts have begun to emerge in the literature. As these designs show, object-oriented programming, an offshoot of AI research, can have a major impact on computational mechanics software development. It is possible to raise the level of abstraction present in large-scale scientific programs (i.e., allowing finite element programmers to deal directly with concepts such as elements and nodes) by identifying and separating levels of concern. Programs designed in this manner allow developers to reason about program fragments in terms of abstract behavior instead of implementation. These program fragments are referred to as objects or data abstractions, their abstract quality being derived from precise specifications of their behavior that are separate and independent of implementation and internal representation details.

15. Model Synthesis Assistance

While the bulk of today's computational mechanics production work is done by means of large comprehensive programs, there is a great deal of exploratory work requiring the development of "one-shot" ad hoc custom-built programs. Developers of such ad hoc programs may have access to subroutine libraries for common modules or "building blocks" but not much else. These developers frequently have to reimplement major segments of complete programs in order to "exercise" the few custom components of their intended program.

One potential application of AI methodology is an expert system to assist in synthesizing computational programs tailored to particular problems, on the fly. The system would require some specifications of the program goal; the constraints (e.g., language, hardware environment, performance); and the description of custom components (e.g., a new equation solver, a new element, a new constitutive equation for a standard element) as input. The system's knowledge base would contain descriptions of program components with their attributes (language, environment, limitations, interface descriptions, etc.) and knowledge about combining program components, which might include the knowledge to write interface programs between incompatible program segments. The expert system would have to use both backward and forward chaining components—the former to break down the goal into the program structure and the latter to select program components to "drive" the low-level custom components.

16. Monitoring Numerical Solutions

Combining numerical techniques with ideas from symbolic computation and with methods incorporating knowledge of the underlying physical phenomena can lead to a new category of intelligent computational tools for use in analysis. Systems that have knowledge of the numerical processes embedded within them and that can reason about the application of these processes can control the invocation and evolution of numerical solutions. They can "see what not to compute" (Abelson 1989) and take advantage of known characteristics of the problem and structure of the solution to suggest data representations and appropriate solution algorithms.

The coupling of symbolic (knowledge-based) and numerical computing is particularly appropriate in situations where the application of pure numerical approaches does not provide the capabilities needed for a particular application. For example, numerical function minimization methods can be coupled with constraint-based reasoning methods from AI technology to successfully attack large nonlinear problem spaces where numerical optimization methods are too weak to find global minima. To derive a solution to the problem, domain-specific knowledge about problem solving in terms of symbolic constraints guides the application of techniques such as problem decomposition, constraint propagation, relaxation, and refinement.

17. Conclusion:

In this study we mainly focus on using a combined reasoning approach in solving a type of problems that cannot be solved by any of the aforementioned standalone systems. We refer to this type as investigation problem which models to some extent a generic situation which might arise in, say, medical diagnosis or the solving of a crime. That is, there are a number of possible diagnoses/suspects (candidates), and the problem is to use the facts of the case to rank them in terms of their likelihood of being the cause of the illness/guilty of the crime. Such ranking often leads to further medical tests/police enquiries focusing on the most likely candidates, which will

bring to light further information about the current case. We use the term dynamic investigation problems to describe a series of such problems to be solved. Solving each problem entails using the facts of the case, coupled with prior knowledge about the domain to narrow down the candidates to just one. However, when there is no upright solution due to lack of some essential information, additional relevant information can often be found in related past cases thereby irregularities can be observed and utilized. Hence, dynamic investigation problems are hybrid machine-learning/constraint solving problems, and as such are more realistic and of interest to the wider AI community.

References

1. Donath, Judith, and boyd, danah. (2014). An alternative view of privacy on Facebook. Information, 2 (1), 140-165. Special issue on “Trust and privacy in our networked world“, edited by Dieter M. Arnold and Herman T. Tavani (journal article)
2. Ellison, N. B. ,Vitak, J., Steinfield, C., Gray, R., and Lampe, C.. (2011). The political economy of privacy on Facebook. Television & New Media, -. (journal article)
3. Fang, L., LeFevre, K., (2010.). Information Revelation and Privacy in Online Social Networks. Proceedings of WPES'05. (pp. 71-80). Alexandria, VA: Association of Computing Machinery (conference paper)
4. Felt, Adrienne, and Evans, David. (2008). NOYB: Privacy in Online Social Networks. Proceedings of the first workshop on Online social networks. (conference paper)
5. Fogel, J., and Nehmad, E.. (2008). Security Issues and Recommendations for Online Social Networks. (techreport)
6. Fuchs, Christian . (2016). The Fourth Amendment and privacy issues on the 'new' internet: Facebook.com and MySpace.com. Southern Illinois University Law Journal, 31. (journal article)
7. Fuchs, Christian. (2011). Privacy: Is There An App For That?. Proceedings of the Symposium on Usable Privacy and Security, July 2011. Pittsburgh, PA..ACM. A survey of Facebook app users via Facebook Platform (conference paper)
8. Grimmelmann, James. (2008). Facebook: Threats to Privacy. 6.805/STS085 (unpublished)
9. Gross, Ralph, and Acquisti, Alessandro. (2015). Characterizing Privacy in Online Social Networks.Proceedings of the first workshop on Online social networks. (conference paper)
10. Guha, Saikat, Tang, Kevin, and Francis, Paul. (2008). Publicly Private and Privately Public: Social Networking on YouTube. JCMC, 13 (1). [Special Issue of JCMC on Social Network Sites, Eds.: danahboyd and Nicole Ellison.] (journal article)
11. Hobgen, G..(2007). Teens, Privacy and Online Social Networks: How teens manage their online identities and personal information in the age of MySpace. (techreport)
12. Hodge, Matthew J.. (2016). The Taste for Privacy: An Analysis of College Student Privacy Settings in an Online Social Network. Journal of ComputerMediated Communication, 14(1), 79-100. (journal article)
13. Jennifer King, AiriLampinen, and Alex Smolen. (2011) Understanding privacy behaviors of millennials within social networking sites, Proceedings of the Association for Information Science and Technology, Volume 48, Issue 1, Pages 1–10
14. Jones, Harvey, Soltren, Jose Hiram. (2015). Friends Only: Examining a PrivacyEnhancing Behavior in Facebook.CHI 2010.Atlanta, GA (conference paper)
15. Krishnamurthy, Balachander, and Wills, Craig E..(2008). Privacy in interaction: Exploring disclosure and social capital in Facebook. Proceedings of the 6th annual International Conference on Weblogs and Social Media. (pp. 3195-3204). (conference paper)
16. Lange, Patricia. (2007). Determining personality traits & privacy concerns from Facebook activity. Black Hat Briefings.Abu Dhabi December
17. Lenhart, A., and Madden, M.. (2007). Can You See Me Now? Audience and Disclosure Management in Online Social Network Sites. Bulletin of Science and Technology Studies. (journal article)
18. Lewis, K., Kaufman, J., and Christakis, N.. (2008). The privacy paradox on social network sites revisited: The role of individual characteristics and group norms. Cyberpsychology: Journal of Psychosocial Research on Cyberspace, 3(2), article 1. (journal article)
19. OCLC. (2007). Students' Facebook 'friends': public and private spheres.Journal of Youth Studies, 12(6), 615-627. (journal article)
20. Pamela Read, Lupita S-O'Brien, Jaqueline Woolcott, Chirag Shah. (2011). How risky are social networking sites? a comparison of places online where youth sexual solicitation and harassment occurs. Pediatrics, 121 (2), E350-E357. (journal article)
21. Solove, D.J.. (2007). Online Social Networking: An Australian Perspective. International Journal of Emerging Technologies and Society, 7(1), 39- 57. (journal article)
22. Stutzman, F., and Kramer-Duffield, J.. (2010) ,"A review of Security and Privacy Issues in Social Networking", International Journal of Computer Science and Information Technologies, Vol. 2 (6) , 2011, 2784-2787

