RESEARCH STATEMENT

Ke Wang (kbwang@ucdavis.edu)

My research interests span Artificial Intelligence (AI), Deep Learning (DL) and Programming Languages (PL). My goal is to create methodologies, techniques and tools for improving the learning efficiency of students and increasing the programming productivity of developers. During the past several years, I have devised automated reasoning strategies to help K-12 students with STEM subjects [IJCAI'15a, IJCAI'15b, IJCAI'16], and developed deep neural architectures to tackle traditional program analysis tasks, such as semantic understanding and program repair [L@S'17, PLDI'18, ICLR'18].

Background and Current Work

Ever since I began my PhD studies, I set the goal for my research to be not only scientifically sound but also practically impactful. As the world is currently undergoing a nascent paradigm shift in education from traditional one-classroom-for-all to massively online and personalized learning, I have set my sight on K-12 education, especially the STEM subjects with which students struggle the most, such as Math and Geometry. In the past several decades, scientists have made remarkable progress in the field of automated reasoning. Research findings have been successfully applied to a wide range of domains such as logic, algebra, *etc.* However, those efforts have not focused on education and hence not well suited for learning purposes.

Specifically addressing the educational needs, I developed a new reasoning technique targeting K-12 education. My key insight is that problem solving typically involves the use and trained familiarity with a number of important problem solving strategies. Therefore, a powerful reasoning paradigm can be viewed as an iterative process of collecting and applying problem-solving strategies. A challenge then arises: How to distill problem-solving strategies in a given problem domain? At the level of each individual problem, directly treating a solution to the problem as a problem-solving strategy is overly specific and lacks generality. Rather, the key is to generalize the specific solutions by retaining their high-level, essential problem-solving strategies and abstracting away their low-level variations. An additional hypothesis, which we validated in our extensive, cross-disciplinary evaluations, is that the number of such problem-solving strategies is often small when appropriately represented. Its generality allowed me to apply the new reasoning methodology to several different settings, including geometry proof problems, Raven's Progressive Matrices (a standardized intelligence test) and Mathematical Word Problems. Educational Testing Service [ETS], a leading educational service provider, recently expressed strong interest in adopting my work to help design Raven's Progressive Matrices, which further highlights the need for and the practical utility of my work.

I am rooted in PL as a PhD student, thus program analysis has always been an important research interest of mine. As online courses have been designed and made available at an unprecedented rate, new challenging research problems have also emerged. In particular, one that has triggered much interest in the PL community is how to automate the feedback loop for online programming courses. A good solution to this problem will produce many benefits. For one, programming courses remain the most popular in the entry-level Computer Science curriculum. Although online courses have made the relevant materials more accessible, important innovations are needed to help provide students with comparable learning experience to that of traditional smaller classroom settings. Automatically generating feedback to students' programming submissions is one such essential utility. Existing approaches either provide inaccurate feedback or do not scale to the MOOC scale. I have proposed the "Search, Align and Repair" data-driven framework that combines PL and AI/DL, and introduced novel program representations to engineer a production-quality system. My system, SARFGEN, is the most powerful automated feedback generator to date, and has been deployed for production use in Microsoft [SARFGEN].

AI for K-12 education

With the same underlying reasoning framework, I am able to create technologies that automate several pedagogical tasks, including solution generation, interactive tutoring and problem generation.

Geometry Theorem Proving Geometry proof problems form one of the most challenging topics for K-12 students. The standard format of a geometry proof problem consists of a given geometry figure, a set of provided constraints and a goal to prove. Students are asked to write a step-by-step deduction using the standard Euclidean axioms. The task becomes more difficult when a proof requires auxiliary constructions — adding lines/arcs to the problem figure to help discover a proof — because finding appropriate constructions can be very challenging as one could add any geometric elements anywhere in the problem figure. I proposed the use of *six* template figures as the

problem-solving strategies to guide auxiliary construction [IJCAI'15a]. The approach is to identify additional lines to complete the partial template figures in the original problem figure. Applicable partial template figures are prioritized to favor those more complete ones to speed up the search process. My prototype, iGeoTutor, significantly outperforms all existing geometry solvers for Euclidean proofs, in terms of both capability and speed.

Interactive Geometry Proof Tutoring It is not always beneficial to directly give students the solution to a problem. Rather, a more helpful pedagogical principle is to help students gradually acquire problem-solving skills on their own. This is the exact goal of my research on interactive tutoring for geometry proof problems [AGPT]. Instead of presenting an auxiliary construction that ultimately leads to a solution, my tutoring system, AGPT, displays the underlying template figures used by iGeoTutor to search for auxiliary constructions, and teaches the student how to apply them to discover the constructions. AGPT guides students to explore the solution space on their own while engaging them by providing hints whenever necessary. To evaluate AGPT, I compared the learning effectiveness of three different groups of students, guided either by AGPT, iGeoTutor, or a human tutor. Study results show that students guided by AGPT perform as well as those guided by human tutors, and perform significantly better than those guided by iGeoTutor.

Raven's Progressive Matrices Besides problem solving, my reasoning methodology can be leveraged to tackle automated problem generation, another important research problem in education. My first work in this direction considers Raven's Progressive Matrices (RPM) [IJCAI'15b]. A RPM problem is presented as a matrix filled with figures, except the last cell, which is left blank. The goal is to select the correct answer from a given set of choices to fill in the missing cell, so that the figures in all the cells follow some patterns.

Automating the construction of RPMs is very beneficial since traditional RPM problems have been manually designed and are quite limited in number. Thus, this work can provide abundant fresh problems both for practice and examination. The major technical challenge to be addressed is how to model the seemingly irregular and diverse variations in RPM problems. My solution is through generalizing the problem-solving strategies of RPMs first proposed in [CARPENTER]. My generalization considers three high-level variation patterns and is able to model any problem on the official RPM tests. Then I model figures as abstract objects and use a finite set of attributes to represent its properties. Governed by particular instantiations from this generalization, the properties among groups of figures can be set and an entire RPM can be assembled. My evaluation results show that the generated RPMs are indistinguishable from officially published RPMs for testing purposes and can also be easily customized with varied difficulty levels, therefore drastically expanding the scarce handcrafted RPM problem corpus.

Mathematical Word Problems This reasoning methodology has also been successfully applied to generate Mathematical Word Problems (MWPs) [IJCAI'16]. An MWP comprises several sentences to communicate a short narrative, which includes some numerical information and asks for some unknown quantities to be calculated. Automatically generating MWPs is a challenging task because a generated problem needs to both exhibit a well-founded mathematical structure and also an easily understood natural language story. My insight is to abstract a problem solving strategy as identifying and utilizing dimensional units from the natural language story to establish mathematical equations. Viewing this abstraction in reverse allows us to generate MWPs: synthesize a dimensionally consistent equation and compose the natural language story via a bottom-up traversal of the equation tree. I realized this approach into a working system for generating MWPs. According to an extensive evaluation, the generated problems are statistically indistinguishable from actual textbook problems.

AI for programming

While massive online courses have made education more accessible and affordable, it also introduces new challenges due to its massive scale. For example, traditional mechanism of manually giving feedback to online students is simply infeasible since the student-to-teacher ratio typically reaches hundreds to one or even in millions to one. Technologies that automate this feedback loop are necessary and need to provide prompt response. Existing approaches have limited applicability — require manual effort, provide imprecise feedback, cannot scale as the size of program increases, *etc.*

I introduce a novel conceptual framework — Search, Align and Repair — that effectively addresses the aforementioned challenges [L@S'17, PLDI'18]. The strategy is to use existing correct programs to repair incorrect programs: it searches for similar reference solutions, aligns each statement in the incorrect program with a corresponding statement in the reference solutions to identify discrepancies for suggesting changes, and pinpoints minimal fixes to patch the incorrect program. At a technical level, it needs to address three key challenges:

Search: Given an incorrect student program, how to efficiently identify a set of closely-related candidate programs among all correct solutions?

Align: How to efficiently and precisely align each selected program with the incorrect program to compute a correction set that consists of expression- or statement-level discrepancies?

Repair: How to quickly identify a minimal set of fixes out of an entire correction set?

For the "Search" challenge, I identify syntactically most similar correct programs *w.r.t.* the incorrect program. Traditional approaches of comparing program similarity via tree-edit distance on the abstract syntax trees do not scale in this setting as the search space of correct solutions is huge. Instead, I propose a new tree embedding scheme for programs using numerical embedding vectors with a new distance metric in the Euclidean space. The new program embedding vectors (called position-aware characteristic vectors) efficiently capture the structural information of the program ASTs. Because the numerical distance metric is easier and faster to compute, program embedding allows us to scale to a large number of programs.

For the "Align" challenge, I introduce a usage-based alpha-conversion to rename variables in a correct program using those from the incorrect program. In particular, I represent each variable with the new embedding vector based on its usage profile in the program, and compute the mapping between two sets of variables using the Euclidean distance metric. In the next phase, I split both programs into sequences of basic blocks and align each pair accordingly. Finally, I construct discrepancies by matching statements only from the aligned basic blocks.

For the "Repair" challenge, I dynamically minimize the set of corrections needed to repair the incorrect program from the large set of possible corrections generated by the alignment step. A naive enumerative search that traverses the whole set of discrepancies will likely run into performance issues. Instead, I incorporate deep learning models to prioritize the application of each correction in the set [ICLR'18]. The idea is to have deep learning models predict the type of errors that a given program may incur, and then rank the corrections based on their relevancy *w.r.t.* prediction results.

The accuracy heavily depends on the degree of understanding that deep learning models have on program semantics. Unlike images and text, a program has a deep and well-defined semantic meaning that can be difficult to capture by only considering its syntax[1]. Therefore existing syntax-based program embeddings are inadequate for our problem setting. I propose a novel semantic program embedding that is learned from the sequence of semantic states from program execution. My insight is that program states expressed as sequential tuples of live variable values not only capture program semantics more precisely, but also offer a more natural fit for *Recurrent Neural Networks* to model. Incorporating new deep neural architectures for training the semantic program embeddings, I improve the naive enumerative search by more than an order of magnitude. Beyond neural program repair, I believe that this dynamic program embeddings can be fruitfully utilized for many other neural program analysis tasks such as program summarization and synthesis.

Future Research — AI/DL for Software Engineering

I am excited and passionate about leveraging deep learning models for solving traditional program analysis tasks. A critical challenge is program representation, *i.e.*, how programs should be represented to maximize deep learning architectures' capabilities. My recent work [ICLR'18] on dynamic program embeddings that I have described earlier starts this exciting journey, which can lead to powerful solutions to traditionally difficult program analysis and software engineering problems. Since Microsoft is one of the largest software company in the world, this line of research can have a significant impact inside the company such as helping Microsoft engineers to improve their productivity or building products to facilitate software developers in general (*e.g.*, providing more advanced IDE support in Visual Studio).

Applications of Dynamic Program Embedding

An immediate line of research is to explore applications of the new dynamic program embedding to a wide range of neural programming tasks, such as synthesis, summarization, and automatic program fixing. Because the dynamic program embedding can more accurately capture program semantics, it is possible to make all existing systems more effective. A technical challenge is how to incorporate the dynamic embedding into existing models in an efficient and elegant manner, as additional information may make models more difficult to generalize and result in worse performance.

[1] syntactically similar programs can exhibit vastly different run-time behavior.

Hybrid Program Representation

Another interesting direction is how to learn hybrid, fine-grained program representations that offer the best trade-offs. The idea is to exploit common programs' characteristics. For example, regarding elements that are more static in nature, such as declarations or assignment statements, syntax-level features (*i.e.* tokens and ASTs) may suffice. As for more dynamic elements like control statements, semantic features (*i.e.*, program dependency graphs or execution traces) may be needed. Therefore, the natural question is: Can we merge different levels of feature dimensions to build a hybrid program embedding that can be learned precisely and efficiently? Apart from combining program features, different techniques (*i.e.*, ngram or RNN modeling) for learning language models can also be considered. A possible idea is to pair each program feature with the most effective modeling technique such that the final hybrid program embedding can be trained with even higher precision.

Deep Neural Networks for Programs

Programs are fundamentally different from images and text. In particular, images and text have more lenient and robust syntax — small variations often lead to small semantic differences. In contrast, program syntax is more rigid and strict — small syntatic-level changes can lead to uncompilable code or code with drastically different semantics. The semantics of a program is also more difficult to interpret from its surface-level syntax as demonstrated in my work on dynamic program embedding [ICLR'18].

Those differences naturally prompt the following questions: (1) Would the programs be unfit for deep neural networks which were designed for images and texts; and (2) how to reinvent deep learning models that can exploit the unique characteristics of programs and do as well as the existing deep models do on images and texts or even better. The answers to those questions can be revolutionary in both DL and PL, either in the form of fundamental theoretic contributions, or in the form of extensive, convincing empirical analysis. Either can help lay out the foundation on which the second challenge can be suitably addressed. Indeed, such a research agenda should be a priority for PL researchers. Although the DL community has been producing ground-breaking results in the areas of computer vision and natural language processing, applying DL models to PL problems has not had nearly the same level of success. I am confident about what potential DL techniques can offer, and I am excited to be a pioneer who bridges the PL and DL gap.

References

[IJCAI'15a] Ke. Wang and Zhendong. Su, "Automated geometry theorem proving for human-readable proofs", *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[IJCAI'15b] Ke. Wang and Zhendong. Su "Automatic Generation of Ravens Progressive Matrices", *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[IJCAI'16] Ke. Wang and Zhendong. Su, "Dimensionally Guided Synthesis of Mathematical Word Problems", *Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016.

[AGPT] Ke. Wang and Zhendong. Su, "Interactive, Intelligent Tutoring for Auxiliary Constructions in Geometry Proofs", *arXiv preprint arXiv:1711.07154*, 2016.

[L@S'17] Ke. Wang, Benjamin. Lin, Bjorn. Rettig, Paul. Pardi and Rishabh. Singh, "Data-Driven Feedback Generator for Online Programming Courses", *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, 2017.

[ICLR'18] Ke. Wang, Rishabh. Singh and Zhendong. Su, "Dynamic Neural Program Embedding for Program Repair", *Sixth International Conference on Learning Representations*, 2018.

[PLDI'18] Ke. Wang, Rishabh. Singh and Zhendong. Su, "Data-Driven Feedback Generation for Introductory Programming Exercises", *Thirty-Ninth ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2018.

[ETS] Educational Testing Service. https://www.ets.org/.

[SARFGEN] AI for Education: Individualized Code Feedback for Thousands of Students. "AI for Education: Individualized Code Feedback for Thousands of Students", https://blogs.technet.microsoft.com/machinelearning/2017/10/25/ai-for-education-individualized-code-feedback-for-thousands-of-students/.

[CARPENTER] Patricia A. Carpenter, Marcel A. Just, and Peter. Shell, "What one intelligence test measures: a theoretical account of the processing in the Raven Progressive Matrices Test.", *Psychological review*, 1990.