

Introduction to SBSE

(1/2)

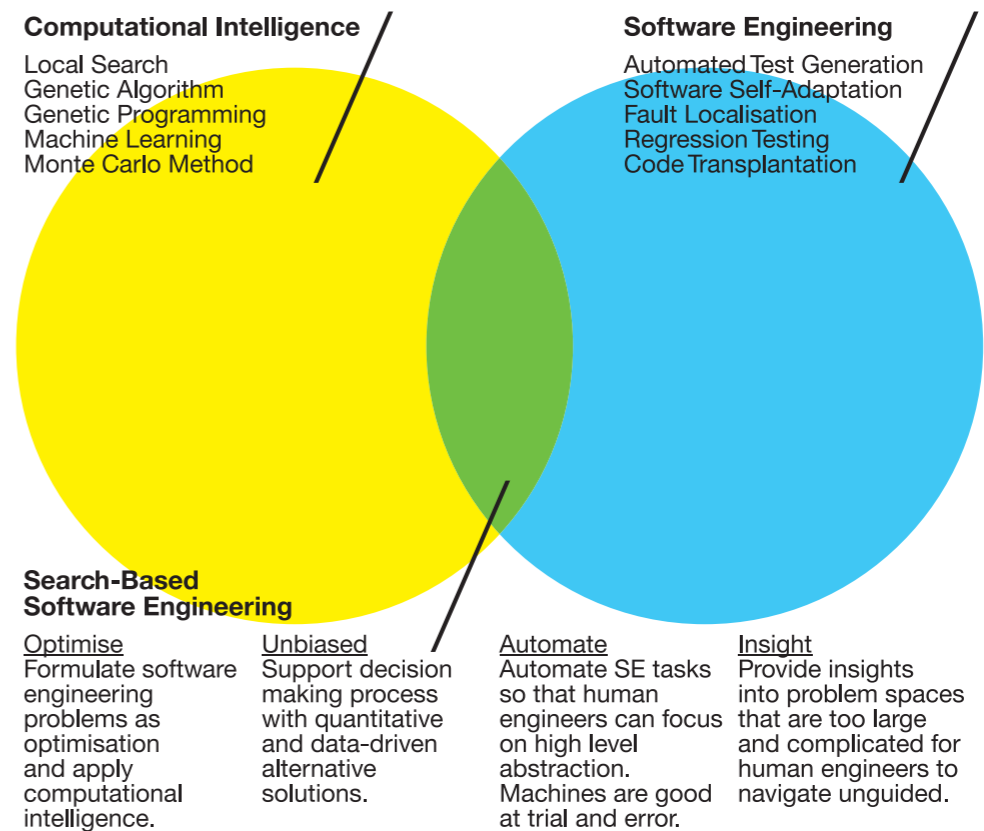
Shin Yoo

SEP592, Summer 2020, School of Computing, KAIST

Me

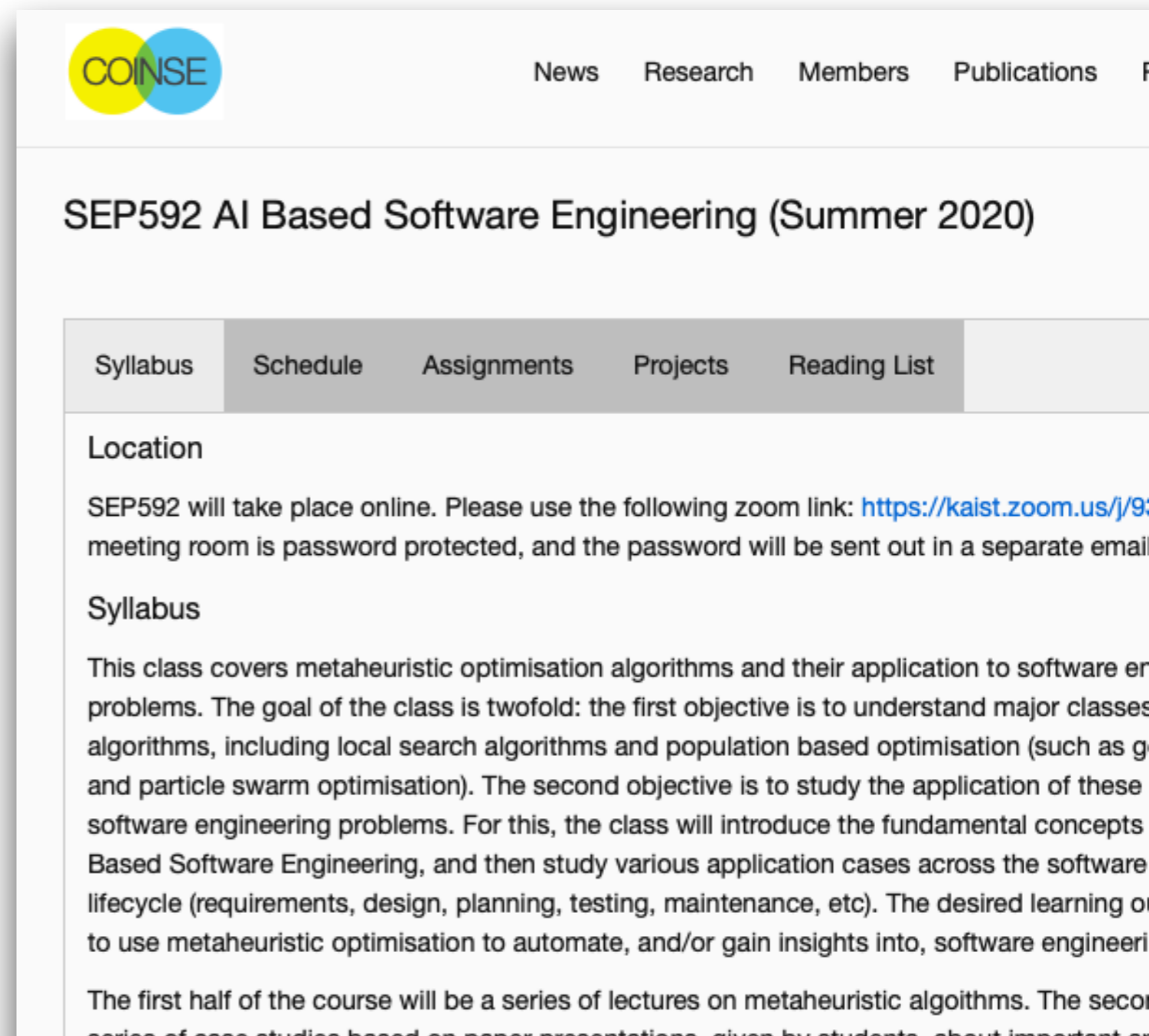
- Shin Yoo
 - PhD at King's College London, UK
 - Assistant Professor at University College London, UK
- COINSE (Computational Intelligence for Software Engineering) Lab
- Research interest: SBSE, regression testing, automated debugging, evolutionary computation, information theory, program analysis...
- shin.yoo@kaist.ac.kr

COMPUTATIONAL INTELLIGENCE FOR SOFTWARE ENGINEERING LAB



Course Webpage

- Reading materials will be either linked or provided on the page
- <https://coinse.kaist.ac.kr/teaching/2021/sep592/>



COINSE

News Research Members Publications

SEP592 AI Based Software Engineering (Summer 2020)

Syllabus Schedule Assignments Projects Reading List

Location

SEP592 will take place online. Please use the following zoom link: <https://kaist.zoom.us/j/9311111111>; meeting room is password protected, and the password will be sent out in a separate email.

Syllabus

This class covers metaheuristic optimisation algorithms and their application to software engineering problems. The goal of the class is twofold: the first objective is to understand major classes of optimisation algorithms, including local search algorithms and population based optimisation (such as genetic algorithms and particle swarm optimisation). The second objective is to study the application of these algorithms to software engineering problems. For this, the class will introduce the fundamental concepts of AI Based Software Engineering, and then study various application cases across the software engineering lifecycle (requirements, design, planning, testing, maintenance, etc). The desired learning outcomes are to use metaheuristic optimisation to automate, and/or gain insights into, software engineering problems.

The first half of the course will be a series of lectures on metaheuristic algorithms. The second half will be a series of case studies based on paper presentations, given by students, about important

Course Evaluation

- **Assignments (20%)**
 - One essay
 - One individual coursework - solve a classical optimisation problems using algorithms learnt during the class. There will be an online leaderboard that shows who's winning :)
 - Quality of the solution
 - Performance of the solution
 - Quality of the implementation & SE practices

Course Evaluation

- **Project (40%)**: put the knowledge you obtained during the class to actual use.
 - Proposal presentation: outline your idea, get feedback.
 - Final report: submit your implementation and empirical evaluation.

Course Evaluation

- **Mid-term Exam (40%)** : written exam during the mid-term week.

Seminars

- We will read recent research papers in different subfields of software engineering, all using search-based techniques to some degree.
- Due to the large class size, I haven't worked out all the details yet. Please bear with me.

Textbook & reading material

- No textbook (we will read up to the bleeding edge)
- Lectures contain strongly recommend reading lists
- Supplementary books:
 - Introduction to Evolutionary Computing, *A. E. Eiben & J. E. Smith*, Springer
 - A Field Guide to Genetic Programming, *Ricardo Poli, William B. Langdon, & Nicholas McPhee* (freely available online at <http://www.gp-field-guide.org.uk>)

Communication

- We will use Slack for all class communication
- <https://bit.ly/slack-sep592-2021>
- If you are in the class, join: no excuse.

Online Presence

- Course Webpage: where you can download materials (slides, coursework descriptions, papers to read)
- KLMS: where you upload your courseworks and project deliverables

What is SBSE?

(wait, before that...)

What is SE?

Science: intellectual and practical activity encompassing the systemic study of the structure and behaviour of the physical and natural world

Mathematics: abstract science of number, quantity, and space, either as abstract concepts or as applied to other disciplines

Engineering: branch of science and technology concerned with the design, building, and use of engines, machines, and structures



Software as science

- Precise computation and algorithm
- Study of truth
- Simplicity of theory



Software as engineering

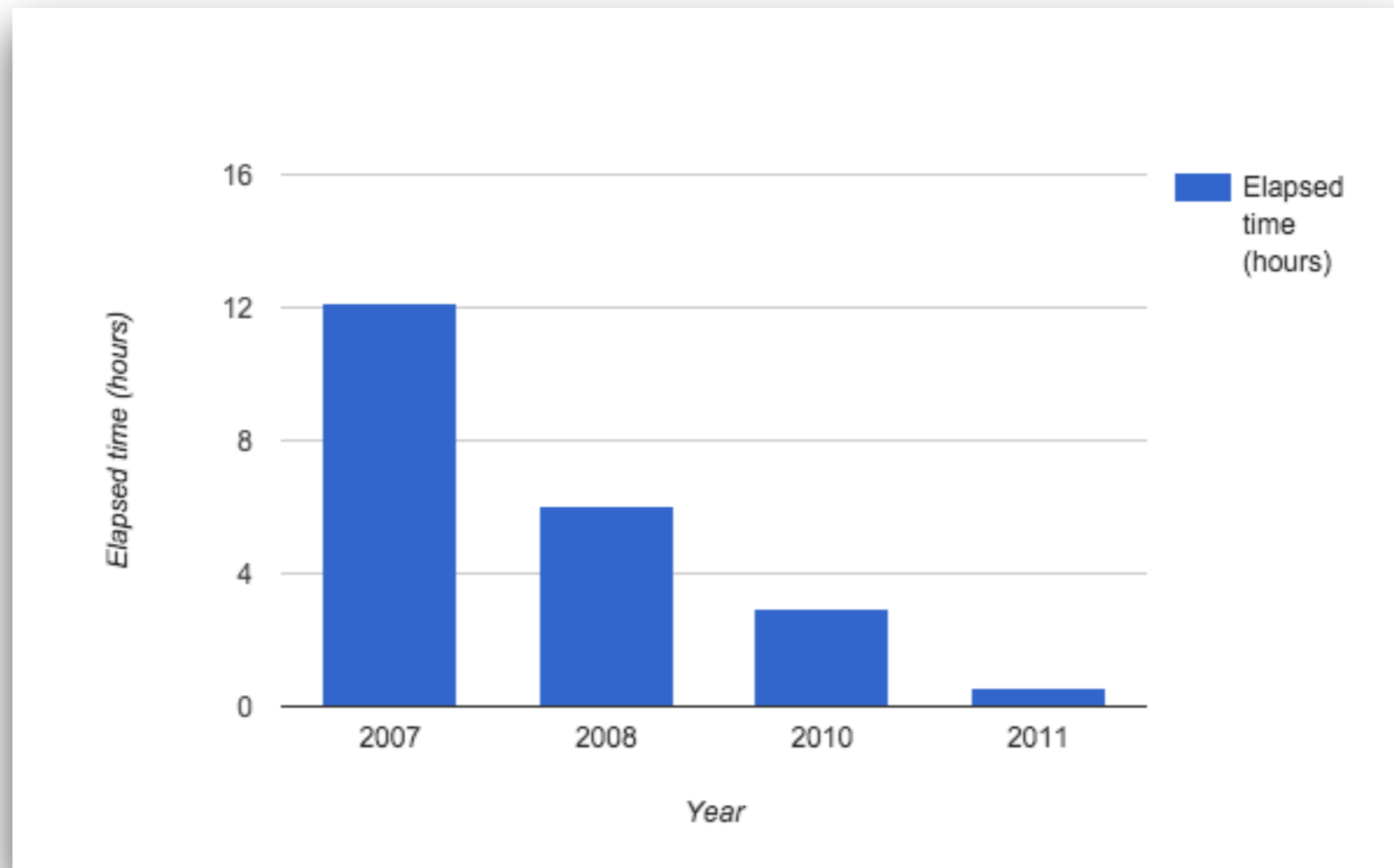
- Consideration of adequate cost
- Study of utility
- Scalability of theory

Science of sorting

```
1 def mergesort(x):
2     """ Function to sort an array using merge sort algorithm """
3     if len(x) == 0 or len(x) == 1:
4         return x
5     else:
6         middle = len(x)/2
7         a = mergesort(x[:middle])
8         b = mergesort(x[middle:])
9         return merge(a,b)
```

#precise #theory # $O(n \log n)$ #provable

Engineering of sorting



1PB = 1,000,000,000,000,000 Byte (10^{15})

#1PB #onlyatGoogle #maynotbecorrect #notenoughstorage

Science

- Is the theory correct?

Engineering

- Is the **implementation** correct?
- If it is not, how do we find out?
- How can we commit fewer mistakes while implementing?
- How can N members collaborate effectively and efficiently?
- How can we reduce the development cost?
- How can we maintain quality when team members change?

Software Crisis

- As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
— *Edsger Dijkstra*, The Humble Programmer, Communications of the ACM, 1972



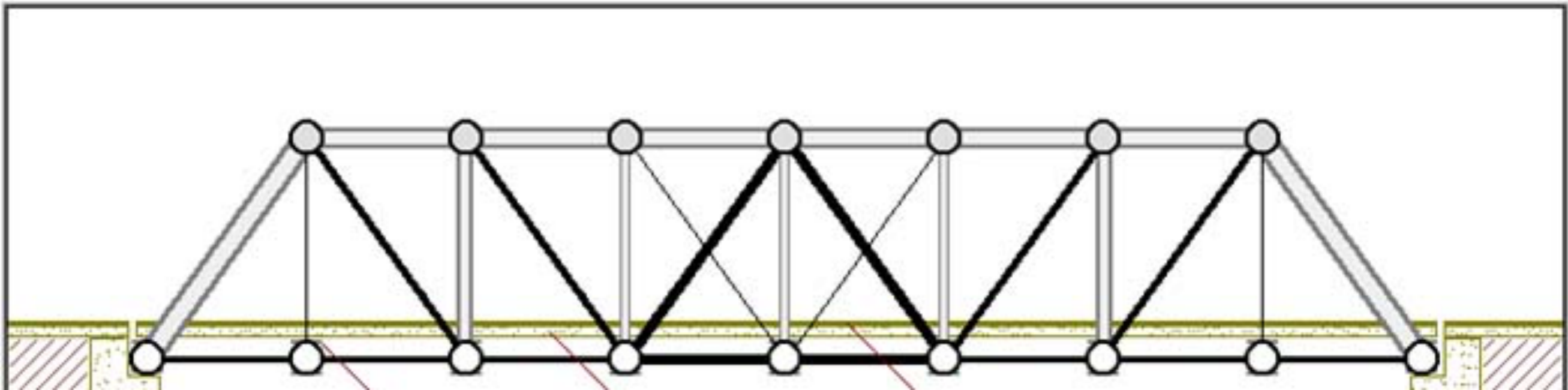
NATO conference 1968

- ...concluded that software engineering should use the philosophies and paradigms of established engineering disciplines, to solve the problem of software crisis.

But what do “established engineering principles” do to improve quality?

- Theory
- Simulation
- Optimisation

Let's say we want to build a steel bridge.



Theory

- When does a steel beam break?
- Stress: force per unit area
- Tensile strength: the maximum stress a material can resist

Stress is the force per unit area.

$$\text{stress} = \frac{\text{force}}{\text{area}}$$

What does stress mean?

Stress allows us to get a fair comparison of the effects of a force on different samples of a material. A tensile force will stretch and, possibly, break the sample. However, the force needed to break a sample will vary - depending on the cross sectional area of the sample. If the cross sectional area is bigger, the breaking force will be bigger. However, the breaking *stress* will always be the same because the stress is the force per unit area.

In picture 4.2, sample A breaks with a force of **60 kN**. Sample B breaks with a force of **120 kN** because it has twice the area (you can think of it as being two pieces of sample A next to each other, with **each one** needing a force of 60 kN to break it). Although the force is bigger for sample B, the stress is the **same** for both samples – it is 60 kN cm².

So breaking **stress** is a more useful measurement than breaking **force** because it is constant for a given material. It allows us to fairly compare the strengths of different materials.

For example:

The force needed to break a piece of steel wire with a cross sectional area of $2 \times 10^{-6} \text{ m}^2$ is 2400 N.

a) What is its breaking stress?

b) What force would be needed to break a steel bar with a cross section of $5 \times 10^{-4} \text{ m}^2$?

a) The breaking stress = breaking force ÷ area
= $2400 \div 2 \times 10^{-6}$
= 1,200,000,000 N m⁻²
= 1.2 GPa.

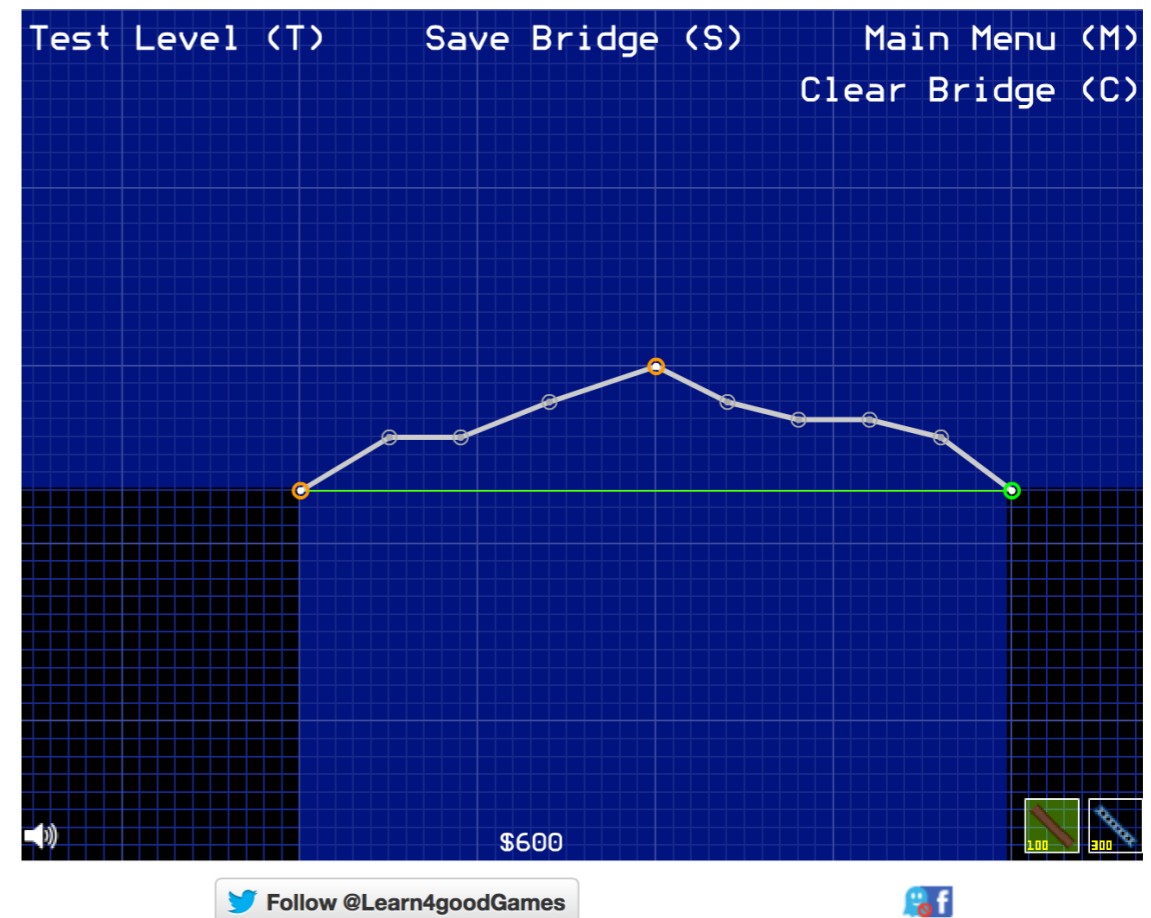
b) To break the steel bar, the force needed = breaking stress x area
= $1.2 \times 10^9 \times 5 \times 10^{-4}$
= 600,000 N

Tensile strength

A tensile test is used to find out what happens when a material such as steel is stretched. A steel bar is placed in a device that pulls one end away from the other fixed end. The **tensile strength** is the maximum stress that the bar can withstand before breaking.

Simulation

- Given the physical laws as the foundation, it is possible to build simulations.



Optimisation

- ...and with simulation, optimisation follows naturally.
- It is, simply, trial and error, which is only possible because we have the simulation environment.

Creating Models of Truss Structures with Optimization

Jeffrey Smith
Carnegie Mellon University

Jessica Hodgins
Carnegie Mellon University

Irving Oppenheim
Carnegie Mellon University

Andrew Witkin
Pixar Animation Studios

Abstract

We present a method for designing truss structures, a common and complex category of buildings, using non-linear optimization. Truss structures are ubiquitous in the industrialized world, appearing as bridges, towers, roof supports and building exoskeletons, yet are complex enough that modeling them by hand is time consuming and tedious. We represent trusses as a set of rigid bars connected by pin joints, which may change location during optimization. By including the location of the joints as well as the strength of individual beams in our design variables, we can simultaneously optimize the geometry and the mass of structures. We present the details of our technique together with examples illustrating its use, including comparisons with real structures.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; G.1.6 [Numerical Analysis]: Optimization—Nonlinear programming; G.1.6 [Numerical Analysis]: Optimization—Constrained optimization

Keywords: Physically based modeling, truss structures, constrained optimization, nonlinear optimization

1 Introduction

A recurring challenge in the field of computer graphics is the creation of realistic models of complex man-made structures. The standard solution to this problem is to build these models by hand, but this approach is time consuming and, where reference images are not available, can be difficult to reconcile with a demand for visual realism. Our paper presents a method, based on practices in the field of structural engineering, to quickly create novel and physically realistic truss structures such as bridges and towers, using simple optimization techniques and a minimum of user effort.

“Truss structures” is a broad category of man-made structures, including bridges (Figure 1), water towers, cranes, roof support trusses (Figure 10), building exoskeletons (Figure 2), and temporary construction frameworks. Trusses derive their utility and distinctive look from their simple construction: rod elements (beams)

{jeffrey|jkh}@cs.cmu.edu, ijo@andrew.cmu.edu, aw@pixar.com

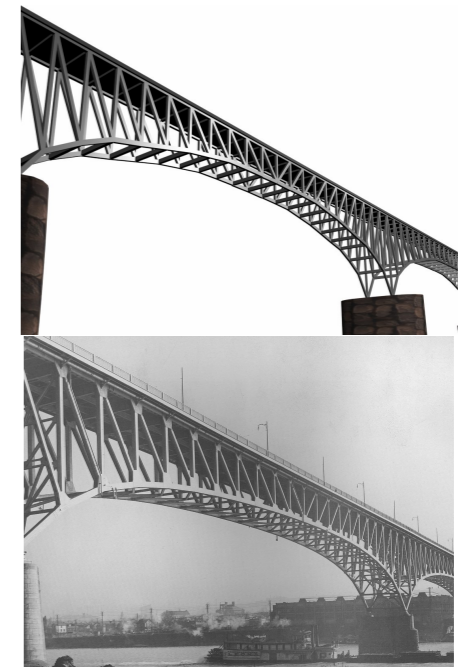


Figure 1: A cantilever bridge generated by our software, compared with the Homestead bridge in Pittsburgh, Pennsylvania.

which exert only axial forces, connected concentrically with welded or bolted joints.

These utilitarian structures are ubiquitous in the industrialized world and can be extremely complex and thus difficult to model. For example, the Eiffel Tower, perhaps the most famous truss structure in the world, contains over 15,000 girders connected at over 30,000 points [Harriss 1975] and even simpler structures, such as railroad bridges, routinely contain hundreds of members of varying lengths. Consequently, modeling of these structures by hand can be difficult and tedious, and an automated method of generating them is desirable.

1.1 Background

Very little has been published in the graphics literature on the problem of the automatic generation of man-made structures. While significant and successful work has been done in recre-

	Theory	Simulation	Optimisation	Product
Bridge Building	Abstract	Computation	Computation	Real World
Software Engineering	Best Practices	?	?	Computation

Theory of software engineering is (somewhat) lacking.

Simulation is done via **modelling**.

Optimisation?

We are probably the only engineering principle, in which the materials for product, simulation, and optimisation are **the same**.

Search-Based Software Engineering

- A large movement(?) that seeks to apply various optimisation techniques to software engineering problems (**NOT** search engines or code search)
- Still relatively young (by academic standards)

Why optimisation?

- **Automate** SE tasks (either fully, or at least until human engineers can attend to the issue)
- Gain **insights** into complicated problem domain that are either too large or too complicated for humans to understand
- **Unbiased** decision support that is data-driven

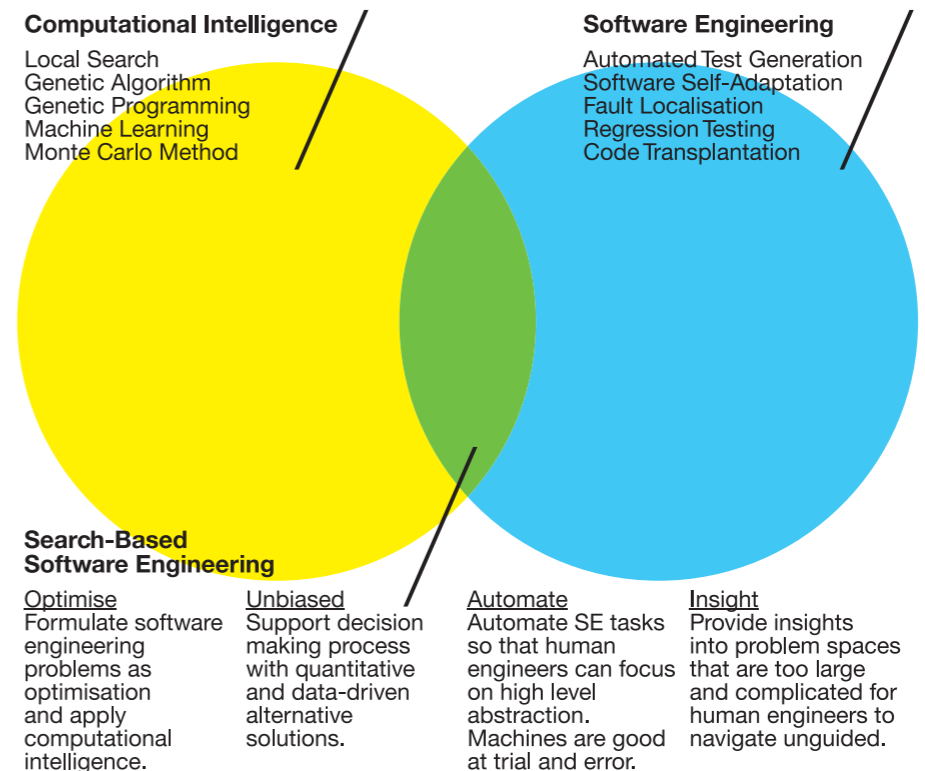
Our Toolbox

- Classical (exact) optimisation would be desirable but often cannot cope with the scale
- A heavy focus on stochastic optimisation, with a heavy emphasis on evolutionary computation and other nature-inspired algorithms (mostly due to historical reasons).
- But we are open to applications of any machine intelligence: neural nets and reinforcement learning are gaining much attention these days naturally.

Our Stance

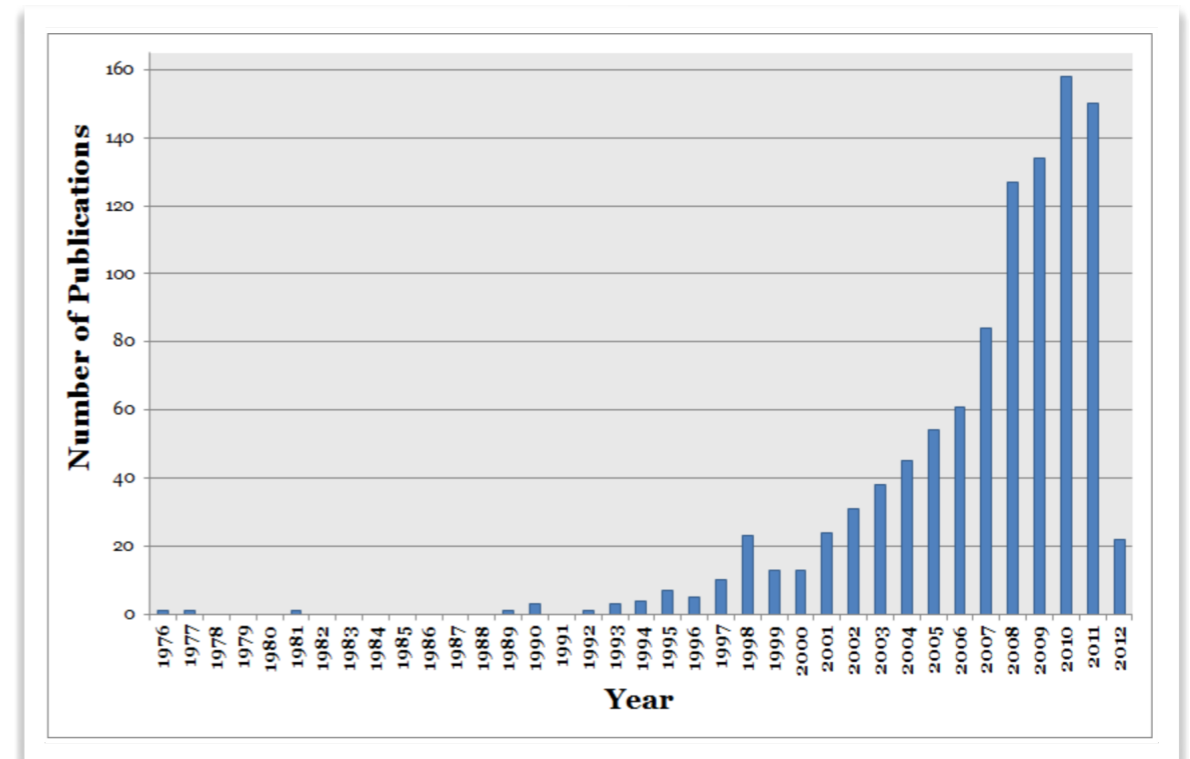
- We stand at the intersection of computational intelligence and software engineering.
- Pragmatic application has stimulated theoretical results in computational intelligence, and vice versa.
- Course: about 40% on optimisation techniques, 60% on applications on SE

COMPUTATIONAL INTELLIGENCE FOR SOFTWARE ENGINEERING LAB



Up and Coming

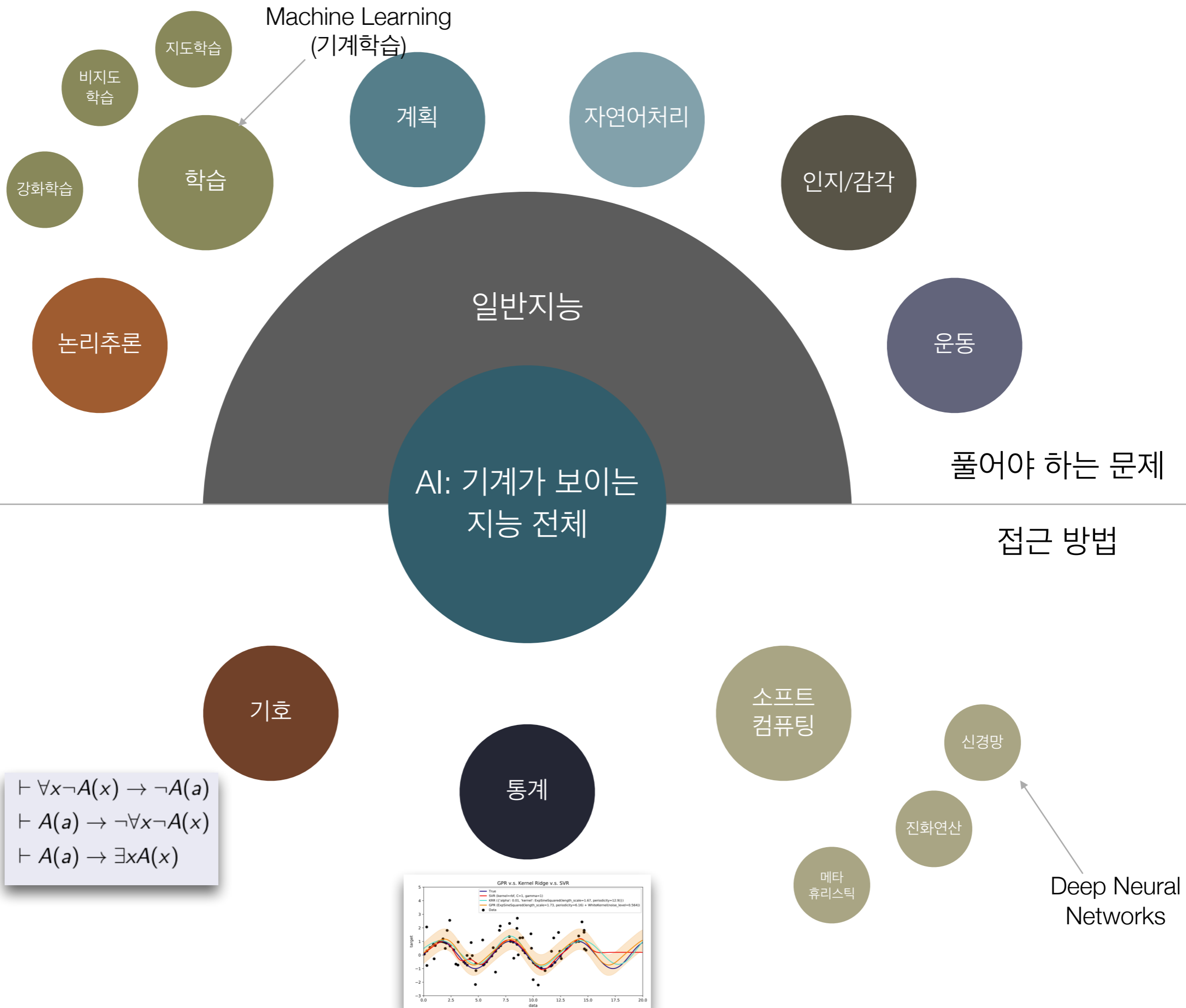
- Meta-heuristic and computational intelligence techniques are found increasingly frequently in SE papers.
- Two major conferences in software engineering - ICSE and FSE - now tend to have whole sessions dedicated to SBSE.
- Evolutionary computation conferences have tracks dedicated to SE.
- Dedicated international conference (SSBSE) and many other workshops.



Publication growth up to 2012

Another Important Angle

- SEP592/CS454 is named AI-based Software Engineering, Initially, this meant that we use AI based techniques to solve SE problems (AI for SE).
- Increasingly, we need to solve SE problems in AI (SE for AI). Most importantly, there is an urgent need to **test machine learning modules in larger systems**.
- Very early stage, but we are going to look into this.



Machine Learning
(기계학습)

지도학습

비지도
학습

강화학습

학습

계획

자연어처리

인지/감각

운동

논리추론

일반지능

AI: 기계가 보이는
지능 전체

풀어야 하는 문제

접근 방법

기호

통계

소프트
컴퓨팅

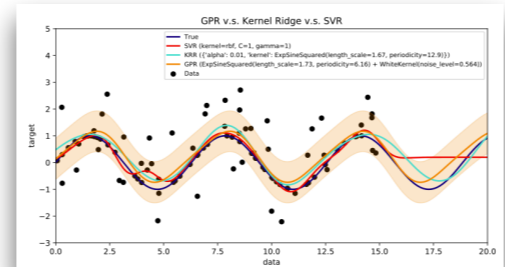
신경망

진화연산

메타
휴리스틱

Deep Neural
Networks

$\vdash \forall x \neg A(x) \rightarrow \neg A(a)$
 $\vdash A(a) \rightarrow \neg \forall x \neg A(x)$
 $\vdash A(a) \rightarrow \exists x A(x)$



주어진 “정답” 예제, 즉 지도(supervision)를 통해 학습

- 분류 (classification): A, B, C에 해당하는 예제를 많이 본 다음, 새로운 입력이 A/B/C 중 어떤 것인지 예측
- 회귀 (regression): $y = f(x_1, \dots, x_n)$ 에 해당하는 예제를 많이 본 다음, 새로운 입력 (x'_1, \dots, x'_n) 이 주어졌을 때 y' 값을 예측

지도학습

주어진 학습 자료에 내재한 공통점을 기반으로 학습

- 클러스터링 (clustering): 여러 개의 입력이 주어졌을 경우, 입력들을 특정 기준에서 서로 비슷한 것들끼리 모음
- 이상탐지 (anomaly detection): 정상적 상태의 특징을 학습한 뒤, 이를 벗어나는 비정상 상태가 됐을 경우 감지 및 경보

비지도학습

학습

환경으로부터 보상이 주어질 때, 보상을 최대화하도록 학습

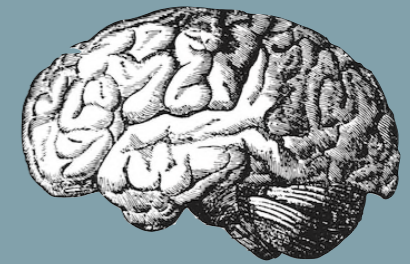
- 몬테카를로 트리 서치(Monte Carlo Tree Search): 게임을 할 경우, 상대의 수를 모르는 상황에서 확률적으로 가장 유리한 수를 탐색

강화학습

소프트 컴퓨팅

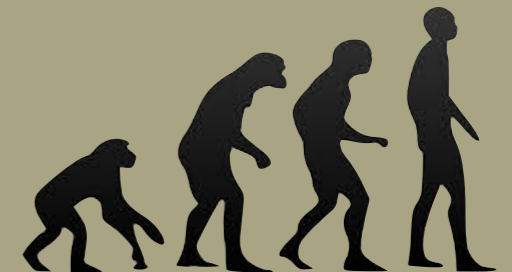
신경망

연결주의(connectionism), 즉 두뇌의 구조인 뉴런의 연결을 통해 지능을 구현할 수 있다는 입장에 기반.



진화연산

다윈의 진화론을 계산을 통해 모방, 원하는 답이 자연 선택을 통해 “진화”할 수 있다는 입장에 기반.



메타
휴리스틱

최적의 답(optimal solution)을 계산하는 것이 불가능할 경우 쓸만한 답을 구하는 문제풀이법을, 문제 유형에 관계없이 적용할 수 있도록 개발.



Coursework 1: Getting to know the subject

- Choose a paper from the reading list. Write an essay containing the following points:
 - What is the software engineering problem authors are trying to solve?
 - What are the technical contributions of the paper?
 - Why do you like this work, or are interested in this work?
- Please submit a PDF file containing your essay. Submission should be made via KLMS. It is due on 23:59, 14th July. Late submission is not allowed for this coursework.