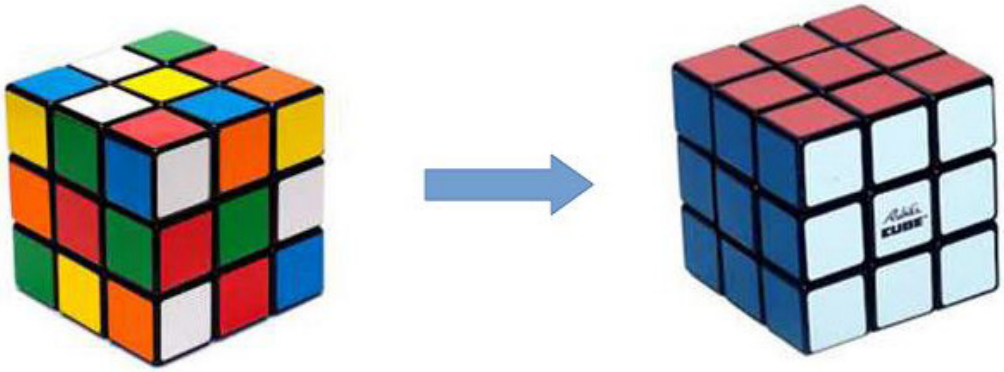
Artificial Intelligence



Zsolt Ercsey

Artificial Intelligence

Pécs

2019

The Artificial Intelligence course material was developed under the project EFOP 3.4.3-16-2016-00005 "Innovative university in a modern city: open-minded, value-driven and inclusive approach in a 21st century higher education model".

EFOP 3.4.3-16-2016-00005 "Innovative university in a modern city: open-minded, value-driven and inclusive approach in a 21st century higher education model".

Zsolt Ercsey

Artificial Intelligence

Pécs

2019

Az Artificial Intelligence tananyag az EFOP-3.4.3-16-2016-00005 azonosító számú, „Korszerű egyetem a modern városban: Értékközpontúság, nyitottság és befogadó szemlélet egy 21. századi felsőoktatási modellben” című projekt keretében valósul meg.

EFOP-3.4.3-16-2016-00005 Korszerű egyetem a modern városban:
Értékközpontúság, nyitottság és befogadó szemlélet
egy 21. századi felsőoktatási modellben

Ercsey Zsolt és Achs Ágnes

Artificial intelligence

Pécs

2019

Az **Artificial intelligence** c. tananyag az EFOP-3.4.3-16-2016-00005 azonosító számú,
„Korszerű egyetem a modern városban: Értékközpontúság, nyitottság és befogadó szemlélet egy 21. századi felsőoktatási modellben” című projekt keretében valósul meg.

ARTIFICIAL INTELLIGENCE
MESTERSÉGES INTELLIGENCIA
EGYETEMI OKTATÁSI SEGÉDLET

Tartalom

Előszó

A Pécsi Tudományegyetem Műszaki és Informatikai Karán 2013-ban indult okleveles mérnökinformatikus MSc képzésben oktatunk Mesterséges intelligencia c. tárgyat magyar és angol nyelven. Ezeket az órákat mind a nappali, mind a levelező képzésben részt vevő hallgatók látogatják. Ez a képzés elsősorban a korábbi évek BSc szinten oktatott mesterséges intelligenciához kapcsolódó tárgyakban vázolt ismeretekre épít.
A tárgyhoz kötelező irodalomként a következő műveket jelöltük meg.

- Alison Cawsey: The Essence of Artificial Intelligence. Prentice Hall. 1998. ISBN-13: 978-0135717790 (magyarul: Mesterséges intelligencia, alapismeretek. Panem. 2002. ISBN 963 545 285 3.);
- Stuart Russell, Peter Norvig: Artificial Intelligence. A Modern Approach. Prentice Hall. 2003. ISBN 0137903952. (magyarul: Mesterséges intelligencia modern megközelítésben. Panem. 2005. ISBN 963 545 411 2.), továbbá
- Mesterséges intelligencia. Szerkesztette: Futó Iván. Aula Kiadó. 1999. ISBN 963 9078 99 9.

A hallgatók általi problémafelvetésekből kiindulva ugyanakkor szükségessé vált egy jelentősen rövidebb terjedelmű oktatási segédlet kidolgozása is, amely a fenti tankönyvekben részletesen ismertetett bizonyos kiemelt témák, témakörök összefoglalásaként forgatható. Ezt a segédletet tartja most a kezében a kedves olvasó, amelyben az említett példák, algoritmusok, feladatok és megoldások a fenti tankönyvekben, illetve a további megjelölt helyeken már publikálásra kerültek.

A szerzők

Artificial Intelligence

Introduction

What does intelligence mean?

?

What does intelligence mean?

•Google:
intelligence – 362 million hits.

•„Intelligence has been defined in many different ways including one's capacity for logic, abstract thought, understanding, self-awareness, communication, learning, emotional knowledge, memory, planning, creativity and problem solving. It can be more generally described as the ability to perceive information, and retain it as knowledge to be applied towards adaptive behaviors within an environment.” Wikipedia

What does intelligence mean?

•capacity for learning, reasoning, understanding, and similar forms of mental activity; aptitude in grasping truths, relationships, facts, meanings, etc.

•manifestation of a high mental capacity.

•the faculty of understanding.

•knowledge of an event, circumstance, etc., received or imparted; news; information.

•the gathering or distribution of information, especially secret information.

•Government: information about an enemy or a potential enemy i) the evaluated conclusions drawn from such information; ii) an organization or agency engaged in gathering such information

•interchange of information.

http://dictionary.reference.com/browse/intelligence

Human intelligence

•Intelligence is the aggregate or global capacity of the individual to act purposefully, to think rationally and to deal effectively with his environment. (David Wechsler, 1944.)

•David Wechsler (1896-1981) Psychologist.

•David Wechsler is best known for developing several widely-used intelligence tests, including the
Wechsler Intelligence Scale for Children (1949) and the Wechsler Adult Intelligence Scale (1955).
Updated versions of these tests are still popular.

http://www.intelltheory.com/wechsler.shtml

Human intelligence #2

•Intelligence can be defined as a general mental ability for reasoning, problem solving, and learning. Because of its general nature, intelligence integrates cognitive functions such as perception, attention, memory, language, or planning.
R. Colom, S. Karama, R. E. Jung, R. J. Haier: Human intelligence and brain networks. Dialogues Clin Neurosci. 2010 Dec; 12(4): 489–501.

What does enginering intelligence mean?

?

Engineering intelligence

•Cleaning up a room.

vs

•Solution of a system of equations.

•Please also note the difference to write an algorithm that solves a system of equations!

Question

•Can human intelligence be inherited?

•There is a mother with 8 children.
3 of them are deaf.
2 of them are blind.
1 of them is mentally disabled.
The mother has syphilis.

Would you suggest abortion?

Question...

• Ludwig van Beethoven,
German composer and pianist,
1770 (Bonn) – 1827 (Wien).

Question #2

•"Heritability" is defined as the proportion of variance in a trait which is attributable to genetic variation within a defined population in a specific environment.

•Heritability takes a value ranging from 0 to 1; a heritability of 1 indicates that all variation in the trait in question is genetic in origin and a heritability of 0 indicates that none of the variation is genetic.

•For example, adult height has a heritability estimated at 0.80 when looking only at the height variation within families where the environment should be very similar.

Question #3

•While the heritability of IQ has been investigated for nearly a century, there is still debate about the significance of heritability estimates and the mechanisms of inheritance.

Examples:

•Johnson, Wendy; Turkheimer, Eric; Gottesman, Irving I.; Bouchard Jr., Thomas J. (2009). "Beyond Heritability: Twin Studies in Behavioral Research". Current Directions in Psychological Science 18 (4): 217–220.

•Devlin, B.; Daniels, Michael; Roeder, Kathryn (1997). "The heritability of IQ". Nature 388 (6641): 468–71.

„The heritability of IQ”

•Covariance between relatives may be due not only to genes, but also to shared environments, and most previous models have assumed different degrees of similarity induced by environments specific to twins, to non-twin siblings (henceforth siblings), and to parents and offspring.

•An alternative model replaces these three environments by two maternal womb environments, one for twins and another for siblings, along with a common home environment.

•Meta-analysis of previous studies shows that our 'maternal-effects' model fits the data better than the 'family-environments' model. Maternal effects, often assumed to be negligible, account for 20% of covariance between twins and 5% between siblings, and the effects of genes are correspondingly reduced, with two measures of heritability being less than 50%.

IQ regression toward the mean

•According to the concept of regression toward the mean, parents whose IQ is at either extreme are more likely to produce offspring with IQ closer to the mean (or average).

Reference:

•Strachan, Tom; Read, Andrew (2011). Human Molecular Genetics, Fourth Edition. New York: Garland Science. pp. 80–81. ISBN 978-0-8153-4149-9.

•Humphreys, Lloyd G. (1978). "To understand regression from parent to offspring, think statistically.". Psychological Bulletin 85 (6): 1317–1322.

What does artificial intelligence mean?

?

What does artificial intelligence mean? #2

•41 million hits at google.

•The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages. (Oxford Advanced Learner's Dictionary)

What does artificial intelligence mean? #3

•Machines designed in such a way that their actions result reflects the result of the activities of human thinking. - J. McCarthy, author of the term "Artificial Intelligence", 1955.

•Process of automation of certain tasks, such as making decisions and learning, that is a reflection of human intelligence. - An Introduction to Artificial Intelligence: Can Computers Think? - Richard Bellmann, 1978.

•Artificial intelligence, the exciting new effort to make computers think. - Artificial Intelligence, The Very Idea - John Haugeland, 1985.

What does artificial intelligence mean? #4

•Field of research seeking to explain and implement intelligent behavior through the use of computational processes. - Artificial intelligence: an engineering approach. Robert J. Schalkoff, 1990.

•Science covering issues of algorithms, fuzzy logic, evolutionary computation, artificial neural network, artificial life and robotics. Artificial intelligence is a branch of computer science, whose subject is the study of the rules governing intelligent human behavior, the creation of formal models of these behaviors, and - as a result - computer software simulating the behavior. - Computational Philosophy of Science. P. Thagard, 1993.

What does artificial intelligence mean?

•The study and design of intelligent agents. ... Systems that think and behave like men, that is, think and behave in a rational way. - Artificial Intelligence: A Modern Approach. S. Russell and P. Norvig, Prentice Hall, 2003.

Easy?

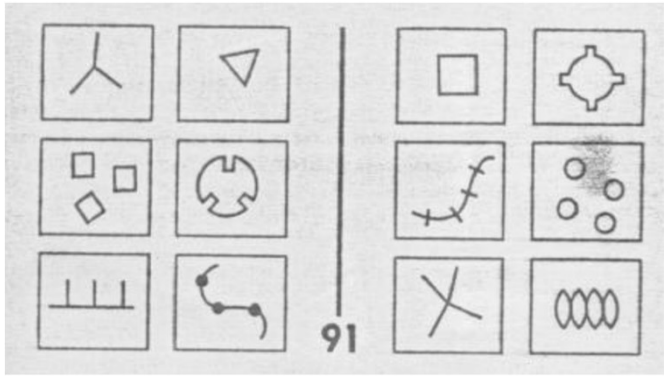
Magyar városok	Ajka	Baja	Balassagyarmat	Békéscsaba	Budapest	Cegléd	Csongrád	Debrecen	Dombóvár	Dunaújváros	Eger	Esztergom	Gyöngyös	Győr	Gyula	Hatvan	Hódmezővásárhely	Jászberény	Kalocsa	Kaposvár	Karcag	Kecskemét	Kiskőrös	Kiskunfélegyháza
Ajka		201	222	325	142	214	254	366	159	128	272	189	224	84	340	200	327	78	190	181	166	202	187	228
Baja	201		240	194	160	136	157	249	124	103	260	207	240	223	209	216	124	185	41	145	233	105	66	131
Balassagyarmat	222	240		183	80	149	218	304	263	148	114	78	121	172	348	97	258	127	199	292	212	166	205	192
Békéscsaba	325	194	183		203	137	110	130	283	206	220	250	208	329	15	184	70	154	212	313	134	123	177	136
Budapest	142	160	80	203		72	138	224	183	68	130	47	82	126	218	58	178	78	119	187	163	86	116	112
Cegléd	214	136	149	137	72		83	158	191	114	124	116	104	198	152	80	123	50	129	220	97	31	85	57
Csongrád	254	157	218	110	138	83		218	219	135	207	185	187	264	125	163	40	133	141	241	165	52	106	26
Debrecen	366	249	304	130	224	158	218		349	272	132	271	152	350	145	176	200	158	278	378	61	189	243	217
Dombóvár	159	124	263	283	183	191	212	349		115	312	228	164	197	299	241	248	241	139	29	288	160	139	186
Dunaújváros	128	103	148	206	68	114	135	272	115		198	115	150	133	221	126	175	146	62	144	211	83	62	109
Eger	272	260	114	220	130	124	207	132	312	198		177	48	256	235	72	228	74	244	317	134	155	209	181
Esztergom	189	207	78	250	47	116	185	271	228	115	177		129	93	265	105	225	125	166	234	210	133	172	159
Gyöngyös	224	240	121	208	82	104	187	152	164	150	48	129		208	223	24	208	54	201	269	154	135	189	161
Győr	84	223	172	329	126	198	264	350	197	133	256	93	208		344	184	304	204	195	203	289	212	192	238
Gyula	340	209	348	15	218	152	125	145	299	221	235	265	223	344		199	85	169	228	328	149	138	192	151
Hatvan	200	216	97	184	58	80	163	176	241	126	72	105	24	184	199		184	30	177	245	144	111	165	137
Hódmezővásárhely	327	124	258	70	178	123	40	200	248	175	228	225	208	304	85	184		154	163	269	181	92	125	66
Jászberény	78	185	127	154	78	50	133	158	241	146	74	125	54	204	169	30	154		170	270	114	81	135	107
Kalocsa	190	41	199	212	119	120	141	278	139	62	244	166	201	195	228	177	163	170		168	217	89	66	115
Kaposvár	181	145	292	313	187	220	241	378	29	144	317	234	269	203	328	245	269	270	168		317	189	165	215
Karcag	166	233	212	134	163	97	165	61	288	211	134	210	154	289	149	144	181	114	217	317		128	182	154
Kecskemét	202	105	166	123	86	31	52	189	160	83	155	133	135	212	138	111	92	81	89	189	128		54	26
Kiskőrös	187	66	205	177	116	85	106	243	139	62	209	172	189	192	192	165	125	135	65	165	182	54		80
Kiskunfélegyháza	228	131	192	136	112	57	26	217	186	109	181	159	161	238	151	137	66	107	115	215	154	26	80	

Please name the furthermost cities!

Easy?

Who is this?

Easy?



Which side has more?

Questions

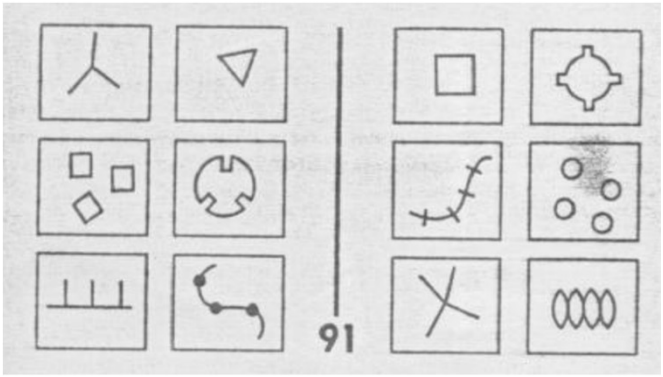
- How can one know?
- How can one know when the information is not sufficient?
- How can a machine know?

?

Bongard problems

- Bongard problems (invented by Russian computer scientist M. M. Bongard, 1967) challenge visual pattern recognition algorithms.
- Given a set A of six figures (examples) and another set B of six figures (counter-examples).
Within the sets the order is not important.
Discover the rule that the figures in A obey and figures in B violate!

Bongard problem 91.



Solution: Meta level information: curve, 3 small black dots.

Other type of Bongard problems

http://www.kfki.hu/~merse/bongard_problemak.html

Measurements

•Intelligence is something that is measured by intelligence tests.

Measurements by a community

•Intelligence is well measured within a community.
BUT
•The reasons behind vary widely.
•There are cultural aspects of intelligence.

Early IQ tests

•The English statistician Francis Galton made the first attempt at creating a standardized test for rating a person's intelligence.

•Alfred Binet, Victor Henri and Théodore Simon (1905) published the Binet-Simon test, which focused on verbal abilities: to identify mental retardation in school children, ie sick children.

Some current IQ tests

•Wechsler Adult Intelligence Scale for adults,

•Wechsler Intelligence Scale for Children for school-age test-takers,

•the current versions of the Stanford-Binet, ie Stanford-Binet Intelligence Scales

•Woodcock-Johnson Tests of Cognitive Abilities,

•the Kaufman Assessment Battery for Children, and Kaufman Brief Intelligence Test

•the Cognitive Assessment System,

•the Differential Ability Scales,

•Raven's Progressive Matrices,

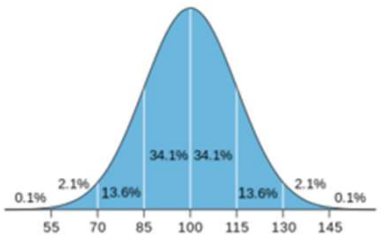
•Cattell Culture Fair III.

IQ tests

•IQ scores can differ to some degree for the same person on different IQ tests.

•~ two-thirds of the population scores between IQ 85 and IQ 115. About 5 percent of the population scores above 125, and 5 percent below 75.

•Normalized IQ distribution with mean 100 and standard deviation 15:



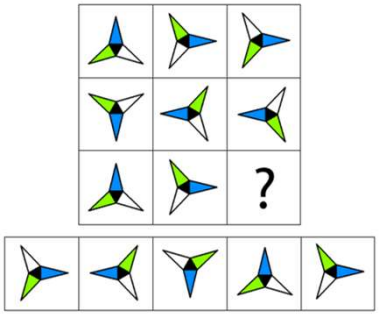
IQ: intelligence quotient

•An intelligence quotient (IQ) is a score derived from one of several standardized tests designed to assess human intelligence.

•The abbreviation "IQ" was coined by the psychologist William Stern for the German term „Intelligenzquotient," his term for a scoring method for intelligence tests he advocated in 1912.

IQ test items

3	15	10
8	40	35
6	30	?



Mensa international

- Mensa is the largest and oldest high IQ society.
- mensa.org
- Founded in 1946 in England, it is a non-profit organization open to people who score at the 98th percentile or higher on a standardized, supervised IQ or other approved intelligence test.
- Mensa formally comprises national groups and the umbrella organization Mensa International, with a registered office in England.
- The word mensa means "table" in Latin, as is symbolized in the organization's logo, and was chosen to demonstrate the round-table nature of the organization; the coming together of equals.

Life Linear

A motivating example

LifeLinear

•Motivating example.

•Albert-László Barabási
Northeastern University
Department of Physics.

•Villanások, a jövő kiszámítható (2010, ISBN 978-963-310-513-9)
Bursts: The Hidden Pattern Behind Everything We Do (2010.)

Cameras are everywhere...

There are cameras all over we walk around:

•public streets and squares,

•bus stops, railway stations,

•shops, offices, university halls,

•bars,

etc.

Pick someone to follow!

•Pick someone on one camera screen.

•Then follow him!

How to follow?

?

How to follow?

•There is no need to identify and match the face with all known faces always!

•Consider the example when it is known that the person took the train.

•Then

- Only persons getting off the train have to be checked.
- Only at specific time spots when the train stops at the stations.

What else? Patterns!

•People usually follow patterns:

•Get up and catch the same train every morning to go to work.

•Get off the train at the same station.

•Leave for lunch around noon every day.

•Leave work around the same time.

•Go to shop at the same store.

•Obviously on the way home.

What else?

?

Can it be learned?

•Patterns can be learned and followed.
•Algorithms can be generated.

BUT
As always...

Exceptions?

•We might have a meeting somewhere else.
•We might go to have a beer.

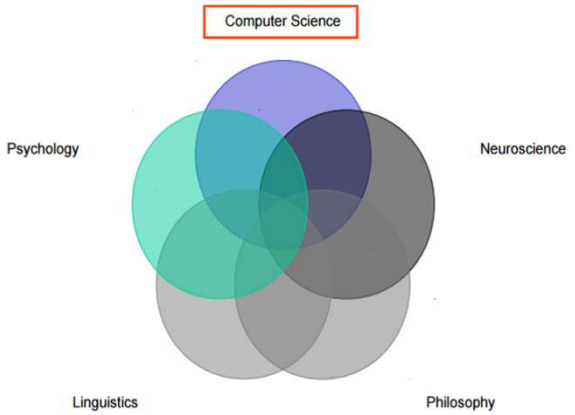
Who will create the first LifeLinear?

•US: business is good, government is bad.
•Europe: government is good, all businessmen are rogues.

•We all give out sensitive, personal information for some fiscal advantages.
•Example 1: sales promotions – fill this form out and you will receive 0.5% discount after you spend all your next months salary here.
•Example 2: Google's android.
•Example 3: other smart devices.

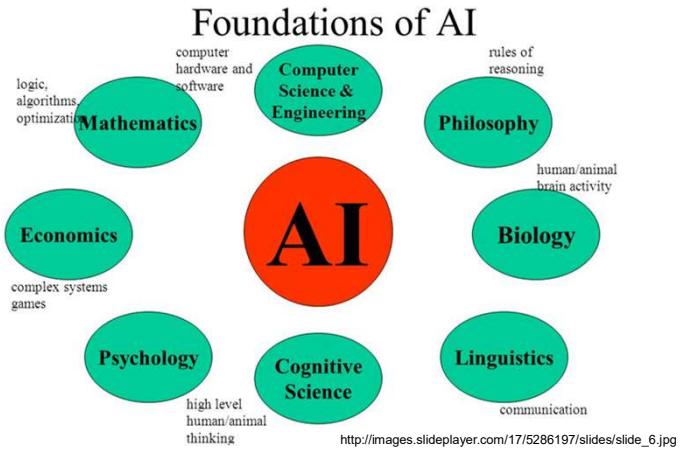
Weak and Strong AI

AI involves numerous fields



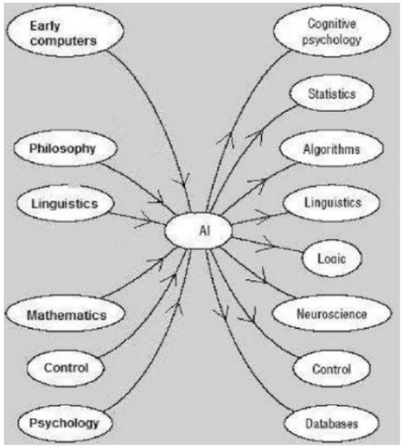
http://aass.oru.se/~mbl/AI/lectures.2011/AI-1.pdf

AI involves numerous fields #2



http://images.slideplayer.com/17/5286197/slides/slide_6.jpg

AI involves numerous fields #3



Combinatorial search
Computer vision
Expert systems
Genetic programming
Knowledge representation
Knowledge based systems
Machine learning
Neural nets
CNN
Natural language processing
Program synthesis
Robotics

And many other fields...

Approaches to AI

Accomodation and adaptation of the environment

Learning ability

Capability of abstract thinking

Use of symbols and concepts

Perceptual intelligence capabilities
- Shape recognition
- Movement pattern recognition
- Orientation

Logical intelligence capabilities
- Organizing problem solving
- Collection of know how
- Reasoning, diagnosis, explaining
- Learning from examples
- Robustness: when uncertain and uncomplete information is available
- Planning

Social intelligence capabilities
- Emotional intelligence
- Communication and collaboration
- Common sense
- Humor

Approaches to AI

	Thinking	**Acting**
Human like	Systems copying, imitating human thinking. How human brain is working? Cognitive science	Systems acting like humans. Turing test.
Rationality	Systems thinking rationally. Formal deducting rules, logic.	Systems acting rationally. How to solve best a given task? Agents.

Russel, 1996.

Weak and Strong AI

- Weak AI:
- Can machines be made to act as if they were intelligent?
- An answer is a Turing test.
- Strong AI:
- Do machines that act intelligently have real, conscious minds?
- For example: human-like AI at Asimov's stories etc.
- non-human-like HAL in the Space Odyssey 2001 at A.C. Clarke.

Strong AI: Chinese room

- John Searle's thought experiment from 1980.
- The hypothetical premise: suppose that AI research has succeeded in constructing a computer that behaves as if it understands Chinese. It takes Chinese characters as input and, by following the instructions of a computer program, produces other Chinese characters, which it presents as output. Suppose, that this computer performs its task so convincingly that it comfortably passes the Turing test: it convinces a human Chinese speaker that the program is itself a live Chinese speaker. To all of the questions that the person asks, it makes appropriate responses, such that any Chinese speaker would be convinced that he or she is talking to another Chinese-speaking human being.
- Does the machine literally "understand" Chinese? (Strong AI) Or is it merely simulating the ability to understand Chinese? (Weak AI)

Imitating human thinking

•„If the human brain was simple enough for us to understand, we would still be so stupid that we couldn't understand it.”
Jostein Gaarder, Sophie’s world.

Turing test

?

Turing party game

•Also known as the "Imitation Game"

•A man and a woman go into separate rooms and guests try to tell them apart by writing a series of questions and reading the typewritten answers sent back.

•In this game both the man and the woman aim to convince the guests that they are the other ones.

Turing party game #2

•What will happen if we replace one of the players to a machine?

• Will the evaluator decide wrongly as often as the game is played between a man and a woman?

Turing test

•The Turing test is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.

•It was developed by Alan Turing in 1950.

•Instead of "Can machines think?" the question is formed:

•"Can machines do what we (as thinking entities) can do?"

Turing Test example:

http://loebner.net/Prizef/2008_Contest/Elbot.pdf

Turing test #2

•Turing proposed that a human evaluator would judge natural language conversations between a human and a computer that is designed to generate human-like responses.

•The conversation is limited to a text-only channel (for ex. A keyboard and screen) so the result is independent from the machine's ability of speech.

•The role of the computer is to make a significant proportion of the evaluator believe that it is really a man.

•The computer is said to have passed the test, if it convinced the human evaluator 70% of the time after five minutes of conversation.

Test A

Evaluator: *Hello.*

A: Hello! Take 2 lol

Evaluator: How about the weather this morning?

A: Not the best, expecting pirates to come out of the fog!

Evaluator: So you are not a machine.

A: Are you allowed to ask that then? No I'm not, although I have a bit of a hangover so I am probably typing like one

Evaluator: There are no instruction about what kind of question can I ask.

A: I have no instructions about what to answer either, so that's all good. Are you local or have you travelled to get to the event?

Evaluator: I am local.

A: What part of Reading? I live on Northumberland Avenue, by the school

Test B

Evaluator: Hello.

B: A very non-mechanical hello to you, also.

Evaluator: Are you sure?

B: Speaking of truth, I say this could be beneficial to me.

Evaluator: What could be beneficial to you?

B: That question was rather indiscreet and I prefer not to answer it!

Evaluator: So I have already know that you are a real human.

B: Can we agree to abide by a minimum standard of civilized behavior? I will not listen to insults!

Evaluator: Oh. I am sorry.

B: The expression 'oh' dates to Middle English as used in the 13th century. What a surprising etymology!

Evaluator: Really?

B: Then again, maybe not.

Evaluator: Seems the knowledge is rather sophasticate.

B: This is confusing. Why bring intellectual perception into this?

Result:

Evaluator: is obviously a human man.

A: human.

B: computer.

Result?

Evaluator: is a man.

Who is A and B? Man or machine?

?

Loebner prize: annual competition of a standard Turing test, began by Hugh Loebner, 1990.

2008-as Loebner prize conversation.

Short History

Origins

Origins

•Philosophy: logic, inference, rationality, language expressions.

•Mathematics: formal representation, algorithms, proofs, calculation methods, probability etc.

•Economics: decision theory.

•Neurology: neural nets.

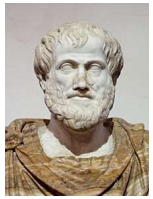
•IT: computers and programming.

•Psychology: behaviour and thinking.

•System design – control theory.

•Linguistics – grammatical rules.

Aristotle: Organon



•Organon is a collection of Aristotle's works on logic.

•It introduces for example the conception of proposition and judgement, syllogistic method, discusses inductive inference.

•It can be considered as the foundation of mathematical logic and also of artificial intelligence.

Copernicius: worldview

•Mikołaj Kopernik (Nicolaus Copernicus, 1473 – 1543) was a Polish astronomer who formed the model of the universe with the Sun at the center: heliocentric solar system.

•Not the visible surface, but the real causes are called to interpret the world.

•Abstraction capability of thinking gains advantage.

Euler: representation

•Leonhard Euler (1707 – 1783) was a Swiss mathematician. He is also widely considered to be the most prolific mathematician of all time.

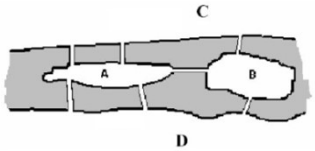
•He made important discoveries in the field of infinitesimal calculus, graph theory, geometry, trigonometry, topology and analytic number theory: more than 73 books each of which has 700 pages!

•A given problem is solved by an analog modell: he developed graph theory to solve the problems of Königsberg.

Seven bridges of Königsberg

•The city of Königsberg (today Kaliningrad) was set on the Pregel River, and included two large islands that were connected to each other and the mainland by seven bridges.

•The problem is to decide whether it is possible to follow a path that crosses each bridge exactly once and returns to the starting point.



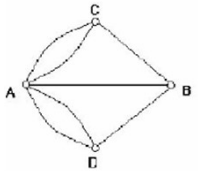
Solution with a graph

•Parts of the city are represented by nodes or vertices; while the bridges are represented by edges.

•The question can now be reformulated as follows: Starting from one node, i.e. part of the city, is it possible to cross each edge, i.e. bridge, only once and get back to the starting node?

•In case someone is on a route and gets to a node, then he has to leave that node on another edge (as the final step, he enters the starting node). Thus, each node has to be entered as many times as it is left. In other words, each node has to have an even number of edges.

•In this case it is violated, thus this problem cannot be solved.



Leibnitz: formal logic

•Gottfried Wilhelm Leibniz (1646-1716) was a German mathematician and philosopher.

•He developed the foundations of symbolic thought and formal logic.

Frege: modern logic

•Friedrich Ludwig Gottlob Frege (1848-1925) was a German mathematician.

•He is considered to be one of the founders of modern logic.

•He is generally considered to be the father of analytic philosophy, for his writings on the philosophy of language and mathematics.

•He is considered to be the bridge between Aristotle's logic and modern artificial intelligence.

Short History

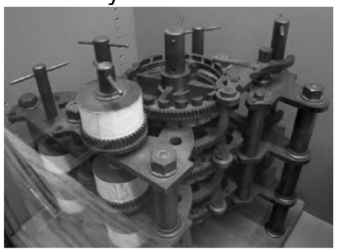
Early days

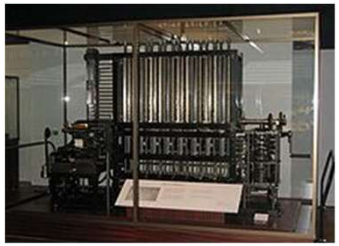
Babbage: difference engine

•Charles Babbage (1791-1871) was an English mathematician.

•He is considered to be the „Father of the computer.”

•Babbage's difference engine (1822) was made to compute values of polynomial functions; to calculate a series of values automatically.





Artificial neuron

•Warren McCulloch (1898-1969) and Walter Pitts (1923-1969) work may be considered as the first result in the field of AI.

•They proposed the first mathematical model of a neural network (1943), often called a McCulloch–Pitts neuron.

•The artificial neuron is at the state of „active” or „inactive,” where the neuron becomes „active” when enough neighbouring neurons stimulate it. They showed that the net of neurons any function can be implemented and each logic function (AND, OR, NOT etc.) can also be implemented.

•McCulloch and Pitts also suggested that a net may also be capable of learning.

Hebbian learning

•Donald Hebb (1904-1985) was a Canadian scientist in the field of neuropsychology.

•He sought to understand how the function of neurons contributed to psychological processes such as learning.

•Hebb introduced a simple rule to update the strengths value between neurons, which allows learning (1949).

Marvin Minsky

•Marvin Minsky (1927-2016).

•In the early 1950s, he worked on computational ideas to characterize human psychological processes and produced theories on how to endow machines with intelligence.

•In 1951 Marvin Minsky with Dean Edmonds built the first randomly wired neural network learning machine, called Snarc.

•Professor Minsky, in 1959, co-founded the M.I.T. Artificial Intelligence Project (later the Artificial Intelligence Laboratory) with his colleague John McCarthy.

John McCarthy

•John McCarthy (1927 – 2011) was an American computer scientist and cognitive scientist.

•McCarthy was one of the founders of the discipline of artificial intelligence; he coined the term: artificial intelligence (AI, 1956).

1956: Dartmouth Conference

•McCarthy, Minsky (Princeton)

•Claude Shannon and Nathaniel Rochestert,

•Trenchard More-t (Princeton), Arthur Samuelt (IBM),

•Ray Solomonoff and Oliver Selfridge (MIT) together with

•Herbert Simon and Allen Newell (Carnegie Tech.)

•http://www-formal.stanford.edu/jmc/history/dartmouth.pdf

Logic Theorist

•Herbert Simon, Allen Newell and Cliff Shaw wrote a computer program in 1955. The Logic Theorist was the first program to imitate the problem solving skills of a human being. The program could successfully prove theorems.

•Concepts involved:

•Reasoning as search (the root is the initial hypothesis, each branch is a deduction based on the rules of logic; while somewhere in the tree is the goal: the proposition the program intended to prove; the path that leads to the goal is the proof),

•Heuristics (to avoid problems of exponential growth of the search trees),

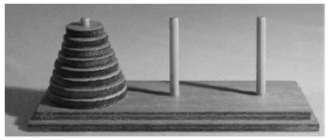
•List processing (it is the basis of LISP today).

General problem solver

•Herbert Simon and Allen Newell constructed a computer program in 1957, that was intended to work as a universal problem solver machine.

•GPS was the first computer program which separated its knowledge of problems (rules represented as input data) from its strategy of how to solve problems (a generic solver engine).

•It solved simple problems, ie the Towers of Hanoi, that could be sufficiently formalized, but it could not solve any real-world problems. The reason behind is basically that search was lost in the combinatorial explosion.



Herbert Simon

•Herbert Simon (1916-2001) was an American economist.

•Professor for many years at Carnegie Mellon University.

•Research fields: artificial intelligence, information processing, decision-making, etc.

•Nobel Prize in Economics, 1978: for his pioneering research into the decision-making process within economic organizations.

Herbert Simon #2

•Psychologist Ulric Neisser said that while machines are capable of replicating 'cold cognition' behaviors such as reasoning, planning, perceiving, and deciding, but they would never be able to replicate 'hot cognition' behaviors such as pain, pleasure, desire, and other emotions.

•As an answer Simon wrote a paper about „Motivational and Emotional Controls of Cognition” which is highly influential today.

•http://digitalcollections.library.cmu.edu/awweb/awarchive?type=file&item=34512

Allen Newell

•Allen Newell (1927-1992) was an American researcher in computer science and cognitive psychology.

•The ACM - AAAI (Association for the Advancement of Artificial Intelligence) Allen Newell Award and
The Award for Research Excellence of the Carnegie Mellon School of Computer Science
was named in his honor.

John McCarthy #2

•McCarthy developed the LISP programming language family (from 1958).

•In 1958 he published „The Advice Taker" which „Programs with Common Sense." The computer program illustrates that some simple axioms are enough for reasoning processes. For example go to the airport and not to miss the flight.

•http://www-formal.stanford.edu/jmc/mcc59.pdf

Nathaniel Rochester

•Nathaniel Rochester (1919-2001) designed the IBM 701 and wrote the first assembler.

Claude Shannon

•Claude Elwood Shannon (1916-2001) was an American mathematician, also known as the father of information and code theory.

•In 1937, Shannon proved that Boolean algebra and binary arithmetic could be used to simplify the arrangement of the electromechanical relays that were used then in telephone call routing switches.

•Next, he proved that it would be possible to use arrangements of relays to solve problems in Boolean algebra. This fundamental concept underlies all digital circuit design today.

Machine evolution

•Around 1958, machine evolution (from which genetic algorithms evolved) was based on the following idea: a code is randomly changed through small mutations, a scoring technique is introduced and a „Teacher" program checked the success or failure. With this simple idea it was supposed that an arbitrary porblem could be solved.

•The problem is that after a large amount of time success is still not guaranteed.

•See for example Friedberg, 1958; Friedberg et al., 1959.

D.B. Fogel, Evolutionary Computation,2006.

Problems: simplicity

•All early systems were useless when they were adopted for complex problems.

For example: translation.

•The National Research Council was heavily financing machine translation of Russian scientific articles, after the first artificial Earth satellite (Sputnik 1) was launched by the Soviet Union in 1957.

•First, scientists thought that simple syntax transformation and substitution from dictionaries are enough to represent the real meaning. In fact, a broader knowledge is necessary to clarify the meaning of the sentences.

•For example: Carry coals to Newcastle. Meaning: To do something pointless and superfluous.(http://www.phrases.org.uk/).

Problems: micro environments

•Early AI problems were considered in micro environments, with few objects, small number of activities and very short solution sequences.

•At the beginning, researchers thought that a solution of a more complex problem is a question of scaling up only with faster hardver and greater memory.

Problems: combinatorial explosion

•The most general search algorithms are brute-force searches since there is no need for any domain specific knowledge. All that is necessary is a state description, a set of legal operators, an initial state, and a descriptions of the goal state. The search proceeds in a systematic way by exploring nodes in some predetermined order or simply by selecting nodes at random.

•The problem with all brute-force search algorithms is that their time complexities grow exponentially with the problem size.

•This problem is called combinatorial explosion.

Problems: complexity

•The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input.

•An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm, i.e.
$T(n) = O(n^k)$ for some constant k.

•These problems can be solved fast.

Problems: complexity #2

•There are some problems, that cannot be solved by a computer, no matter how much time do we have.

•A problem H is NP-hard (non-deterministic polynomial-time hard) when every problem L in NP can be reduced in polynomial time to H.

•It is still not proven that there are no polynomial-time algorithms for NP-hard problems!

•Consequences:

•If P ≠ NP, then NP-hard problems cannot be solved in polynomial time;

•If an optimization problem H has an NP-complete decision version L, then H is NP-hard.

Example: Euler circle

•Let G=(V,E) be a connected, directed graph.

•An Euler path is a path that uses every edge of a graph exactly once; while an Euler circuit is a circuit that uses every edge of a graph exactly once (the circuit starts and ends at the same vertex).

•To determine whether there is an Euler circuit can be done in O(E) time.

http://compalg.inf.elte.hu/~tony/Oktatas/Osztott-algoritmusok/P-NP-NPC.pdf

Example: Hamiltonian circle

•Let G=(V,E) be a directed graph. A Hamiltonian circle visits each node (which is an element of V) exactly once.

•To determine whether there is a Hamiltonian cirlce in the graph is NP-complete.

•NB: Any given solution to an NP-complete problem can be verified quickly (in polynomial time).

http://compalg.inf.elte.hu/~tony/Oktatas/Osztott-algoritmusok/P-NP-NPC.pdf

Erdős and Rényi

•Pál Erdős (1913-1996) was a Hungarian mathematician.

•Alfréd Rényi (1921-1970) was a Hungarian mathematician.

•They introduced random graphs in 1959 (using a dice); today random graphs serve as models for real-world networks.

•Let a set of vertices be given. Let one select 2 vertices and draw an edge between them only when 6 was thrown with the dice, otherwise do not draw the edge. Then select again 2 vertices from the set.

•More information on random graphs: http://www.win.tue.nl/~rhofstad/NotesRGCN.pdf

Short history

Recent years

Raj Reddy

•Dabbala Rajagopal "Raj" Reddy (1937-) is an Indian-American computer scientist.

•He was the joint winner of the ACM A.M. TURING AWARD in 1994.

•He is known for the design and construction of large scale artificial intelligence systems, demonstrating the practical importance and potential commercial impact of artificial intelligence technology.

•He was the founding director of the Robotics Institute at Carnegie Mellon University 1979-91.

Edward Feigenbaum

•Edward Albert Feigenbaum (1936-) is a computer scientist working in the field of artificial intelligence.

•He was the joint winner of the ACM A.M. TURING AWARD in 1994.

•He is known as *the father of expert systems.*

•He is a professor at Stanford University.

Leslie Gabriel Valiant

•Born in Budapest, 1949, he is a British computer scientist, currently professor at the Harvard University.

•He is known for contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.

•He was the winner of the ACM A.M TURING AWARD in 2010.

•https://people.seas.harvard.edu/~valiant/

Judea Pearl

•Judea Pearl (1936-) is an Israeli-American professor.

•He created the representational and computational foundation for the processing of information under uncertainty. He is best known for the probabilistic approach to artificial intelligence, the development of Bayesian networks and the works in the field of causal inference.

•He was the winner of the ACM A.M TURING AWARD in 2011.

•http://bayes.cs.ucla.edu/jp_home.html

Margaret Boden

•Prof. Margaret A. Boden has been contributing to the philosophy of AI and cognitive science since the 1960s.

•She was the recipient of the 2017 ACM - AAAI Allen Newell Award mainly for contributions to the philosophy of cognitive science and artificial intelligence, particularly in the study of human creativity. She introduced the idea of creative cognition, which served as the foundation of computational creativity as a new subfield of artificial intelligence.

•Suggested books: *Artificial Intelligence and Natural Man* (1977); *The Creative Mind* (1990); *Mind as Machine: A History of Cognitive Science* (2006); *AI, Its Nature and Future* (2016).

Dina Katabi

•She was born in Damascus, 1971.

•She was the recipient of the 2017 ACM Prize Computing.

•She applies methods from communication theory, signal processing and machine learning to solve problems in wireless networking.

•She works at MIT Computer Science & Artificial Intelligence Lab.

•http://people.csail.mit.edu/dina/

Andrew Ng

•Professor at Stanford University.

•Famous for his online Machine Learning class that was offered to over 100,000 students: https://www.coursera.org/learn/machine-learning

•Co-Chairman, Co-Founder and Scientist of Coursera, Google Brain, Baidu.

•IJCAI Computers and Thought award (highest award in AI given to a researcher under 35): 2009

•http://www.andrewng.org/

Charles L. Isbell

•Professor and executive associate dean at the Georgia Institute of Technology.

•His research focuses on the interactions between AIs and humans.

•He was one of the initiators of the Online Master of Science in Computer Science (OMSCS), for a total cost of about $7,000.

•https://www.omscs.gatech.edu/

•https://www.cc.gatech.edu/~isbell/

Risto Miikkulainen

•Professor UT Austin and vice president of Research at Sentient Technologies.

•His research focuses on methods and applications of neuroevolution, neural network models of natural language processing and vision.

•Gabor Award, the International Neural Network Society, 2017.

•Outstanding Paper of the Decade Award, International Society for Artificial Life, 2017 with K. Stanley.

•https://www.cs.utexas.edu/users/risto/

Kenneth Stanley

•Professor in the Department of Computer Science at the University of Central Florida and senior research scientist at Uber AI Labs.

•Outstanding Paper of the Decade Award, International Society for Artificial Life, 2017 with R. Miikkulainen: Evolving Neural Networks Through Augmenting Topologies (2002) http://nn.cs.utexas.edu/keyword?stanley:ec02

•https://www.cs.ucf.edu/~kstanley/

Jordan B. Pollack

•Professor at the Brandeis University, leading the Dynamical & Evolutionary Machine Organization.

•His main interests are: Genetic Algorithms, co-evolutionary machine learning systems, manufacturing of robots.

•Winner of the 2017 ISAL (International Society for Artificial Life) Award for Lifetime Achievement in the field of Artificial Life

•http://www.cs.brandeis.edu/~pollack/

Derong Liu

•Professor at the University of Illinois at Chicago.

•His main interests are: adaptive dynamic programming and reinforcement learning, neural networks, computational intelligence.

•Winner of the 2018 INNS (International Neural Network Society) Gabor Award.

•https://www.ece.uic.edu/~dliu/

Hiroki Sayama

•Professor at Binghamton University, State University of New York

•His main interests are: complex systems, Artificial Life/Chemistry, interactive systems.

•Winner of the 2016 ISAL Exceptional Service Award

•http://bingweb.binghamton.edu/~sayama/

Donald C. Wunsch

•Professor at the Electrical and Computer Engineering Department, Missouri University of Science and Technology.

•His main interests are: clustering, neural networks, reinforcement learning, robotic swarms, bioinformatics.

•Winner of the 2015 Gabor Award.

•http://people.mst.edu/faculty/dwunsch/

Josh Bongard

•Professor at the University of Vermont, Morphology, Evolution & Cognition Laboratory.

•His main interests are: robot design and modeling a physical system.

•2016 winner of the International Society of Artificial Life Education & Outreach Award; International Society of Artificial Life Best Paper of the Decade 2001-2011: http://science.sciencemag.org/content/314/5802/1118

•http://www.cs.uvm.edu/~jbongard/

Albert-László Barabási

•A Hungarian born professor at the Northeastern University.

•He is best known for his work in the research of network theory and network science. He introduced in 1999 the concept of scale-free networks and proposed the Barabási–Albert model to explain their widespread emergence in natural, technological and social systems.

•http://barabasi.com/about/about

•http://networksciencebook.com/

Cesare Alippi

•Professor at the Politecnico di Milano, Italy.

•His main interests are: intelligent cyberphysical systems, machine learning mechanisms, adaptive processing systems.

•2016 winner of the Gabor Award.

•http://home.deib.polimi.it/alippi/

Pieter Abbeel

•Professor at UC Berkley; Director of the UC Berkeley Robot Learning Lab. Co-founder of covariant.ai and gradescope.com.

•His main interests are: deep learning for robotic, reinforcement learning, learning-to-learn, and AI safety.

•ICLR 2018 Best Paper Award; Diane McEntyre Award for Excellence in Teaching 2018 (Computer Science, UC Berkeley) etc.

•Offered Artificial Intelligence 12 weeks course on edX: https://www.edx.org/course/artificial-intelligence-ai-columbiax-csmm-101x-0#.VKuKQmTF_og

•http://people.eecs.berkeley.edu/~pabbeel/

Yeal Niv

•Professor at Princeton Neuroscience Institute & Psychology Department, Princeton University.

•2015 Winner of the National Academy of Sciences Troland Research Award, for her studies at the confluence of theory and experiment that illuminate the behavioral and biological foundations of learning and decision-making

•http://www.princeton.edu/~yael/

Ayanna Howard

•Professor and Chair of the School of Interactive Computing in the College of Computing at Georgia Tech.

•Main interest: pediatric robotics and human-robot trust.

• She was named one of the 23 most powerful women engineers in the world by Business Insider, 2015.

•http://howard.ece.gatech.edu/

Jeff Clune

•Professor at the University of Wyoming, directing the Evolving Artificial Intelligence Lab, senior research scientist at Uber AI Labs.

•His main interests are: deep learning, neural networks, robotics.

•2016 winner of the International Society of Artificial Life Award for Distinguished Young Investigator

•http://jeffclune.com/

Jean-Baptiste Mouret

•Born in 1981.

•Researcher at Inria, France.

•His main interests are: machine learning, robotics, evolutionary computation, neuro-evolution.

•Winner of the 2017 ISAL (International Society for Artificial Life) Award for Distinguished Young Investigator in the field of Artificial Life.

•https://members.loria.fr/JBMouret/

Timnit Gebru

•Timnit Gebru is a postdoctoral researcher at Microsoft Research in the Fairness, Accountability, Transparency and Ethics in AI, or FATE, group.

•PhD at Stanford Artificial Intelligence Laboratory, supervisor: Fei-Fei Li.

•Main interest: algorithmic bias and ethical implications underlying data mining projects.

•http://ai.stanford.edu/~tgebru/

Graham Taylor

•Graham Taylor is a professor at the University of Guelph, directing the Machine Learning Research Group.

•Main interest: statistical machine learning and biologically-inspired computer vision, with an emphasis on deep learning and time series analysis.

•He was named as one of Canada's Top 40 Under 40, 2018 June.

•http://www.uoguelph.ca/~gwtaylor/

Greg Brockman

•Greg Brockman, OpenAI Co-Founder (with Elon Musk, Sam Altman) and CTO.

•https://gregbrockman.com/

•https://www.openai.com/

Mazin Gilbert

AT&T Labs Research

•Mazin Gilbert, vice president of advanced technology & architecture with AT&T.

•AT&T collaborating with Tech Mahindra builds Acumos, an open source AI platform, which makes it easy to build, share and deploy AI applications.

•http://www.research.att.com/sites/labs_research/open_source

•http://www.research.att.com/content/dam/sites/labs_research/content/publications/Acumos_Whitepaper.pdf

Artificial Intelligence

Selected Papers + Awards

Selected CVPR Best Papers

•The Conference on Computer Vision and Pattern Recognition, since 1983; since 212 the papers are open access.

•2018 Best Paper: "Taskonomy: Disentangling Task Transfer Learning" by Amir R. Zamir, Alexander Sax, William Shen, Leonidas J. Guibas, Jitendra Malik, Silvio Savarese

•2018 Honorable Mention: "Deep Learning of Graph Matching" by Andrei Zanfir, Cristian Sminchisescu

•2017 Best Paper: Densely Connected Convolutional Networks by Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

•2017 Best Paper: Learning from Simulated and Unsupervised Images through Adversarial Training by Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, Russell Webb

Selected CVPR Best Papers #2

•2016 Best Paper: Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

•2016 Best Student Paper: Structural-RNN: Deep Learning on Spatio-Temporal Graphs, Ashesh Jain, Amir R. Zamir, Silvio Savarese, Ashutosh Saxena

•2015 Best Paper Honorable Mention: Fully Convolutional Networks for Semantic Segmentation, Jonathan Long, Evan Shelhamer, Trevor Darrell

Selected NIPS Best Papers

•NIPS is a machine learning and computational neuroscience conference since 2002, December 3 – 8, 2018, Montréal, Canada.

•2017 Best Paper: Noam Brown, Tuomas Sandholm. *Safe and Nested Subgame Solving for Imperfect-Information Games*.

•2017 Best paper: Wittawat Jitkrittum, Wenkai Xu, Zoltan Szabo, Kenji Fukumizu, Arthur Gretton. *A Linear-Time Kernel Goodness-of-Fit Test*.

•2016 Best Paper: Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, Pieter Abbeel: *Value Iteration Networks.*

•2016 Best Student Paper: Rong Ge, Jason Lee, Tengyu Ma: Matrix Completion has *No Spurious Local Minimum.*

Selected Awards

• ACM A.M. TURING AWARD, often referred to as the “Nobel Prize of Computing.”

•ACM - Association for the Advancement of Artificial Intelligence Allen Newell Award

• International Joint Conferences on Artificial Intelligence (IJCAI), recognizing outstanding young scientists in artificial intelligence.

•International Society for Artificial Life.

Problem representation

Problem representation

- Problem representation is the first step in solving AI problems.
- The description of the problem: representation.

How do we find an address at Pécs?

?

Addresses at Pécs

- Systematic approach: for example, we have a map.
  We use an algorithm to find, to explore a way.
- Intuitive approach: where can the center be? Can I see a tall tower above the roof tops?
  We may have an idea, that seems to be good. But we do not know whether it is truly good.

General features of AI Problems

•The problem space (with the potential solutions) is big; most of the times it is not possible to try each and every case.

•The answer can be described with a sequence of atomic activities (operations).

•The answer cannot be fixed in advance; it has to be selected from many potential sequences.

•Controlled or guided search is necessary.

Models for problem representation

•State space representation.

•Problem reduction; problem decomposition.

•Logic representation.

•Structured object based representation.

•Distributed representation.

•Representation games.

State space representation

•State space representation is a widely used model. It is used not only for AI problems.

•With its help a problem can be described, specified.

•Characteristics:

•Problems are considered and solved through a sequence of operations.

•Therefore state space representations describe a pathway seeking problem.

Components of the state space representation

What is necessary to describe the state space?

•The state space of the problem, i.e. the potential states.

•Operations valid within the state space; including the range of interpretation of the operation.

•Initial state(s), starting point(s).

•Target state(s).

State space representation

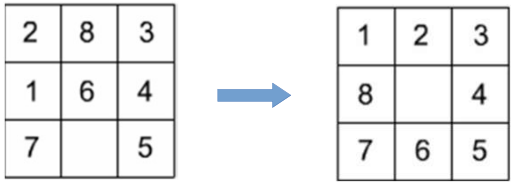
Examples

8-puzzle game

•Sliding-block puzzle.

•State space: ?

•Operations: ?



START TARGET

8-puzzle game #2

•State space: potential states, describing each 8 blocks positions as well as the position of the empty block.

•Initial state or starting position: any state can be an initial state.

•Operation: one of the 8 blocks to be moved to the left, right, up or down. An easier way to describe the operation: move the empty block to the left, right, up or down.

8-puzzle game #3

•Please note that a computer representation of this 8-puzzle game can be for example a 3x3 matrix, with $0 \leq Ai \leq 8$.

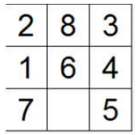
•There are 9!/2=181440 potential states for this simple problem.

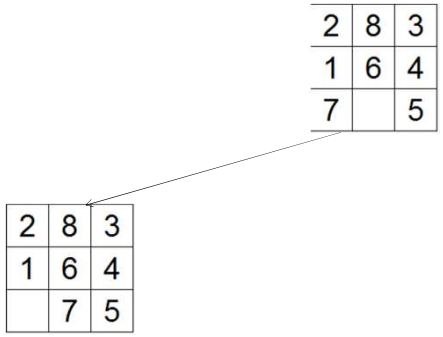
•Please note that the class of the sliding block puzzles is NP complete.

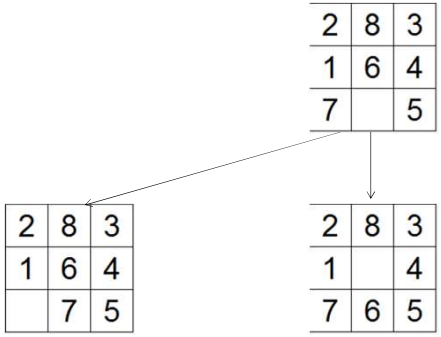
•The 15-puzzle game with 4x4 blocks has ~10billion states with randomly generated problems solved within miliseconds; but

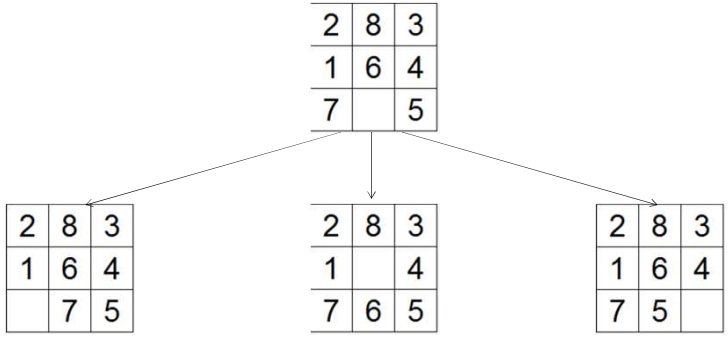
•The 24-puzzle game with 5x5 blocks has ~10^{25} states, where randomly generated problems might not be solved.

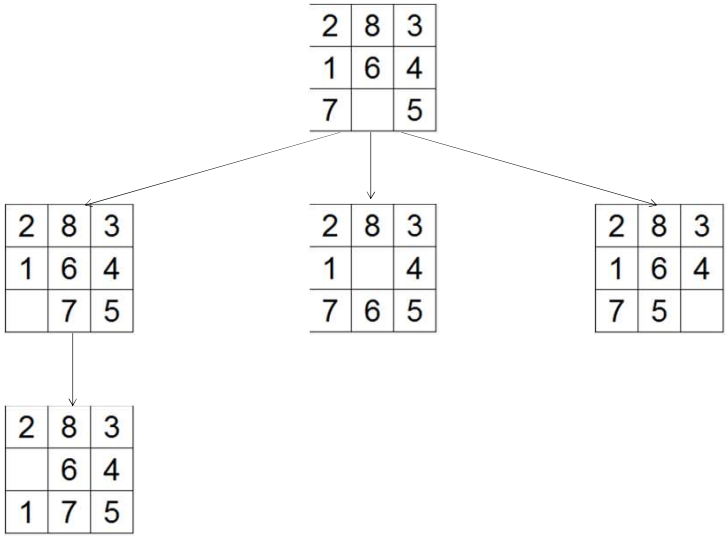
Representation with a graph: 8-puzzle game

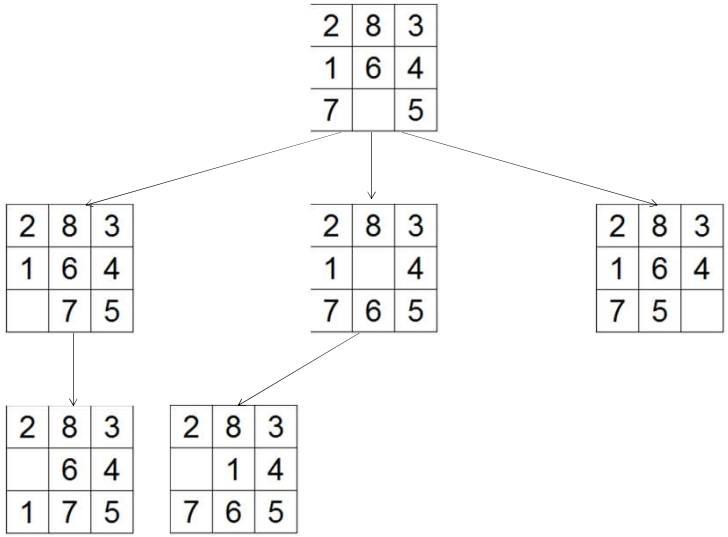


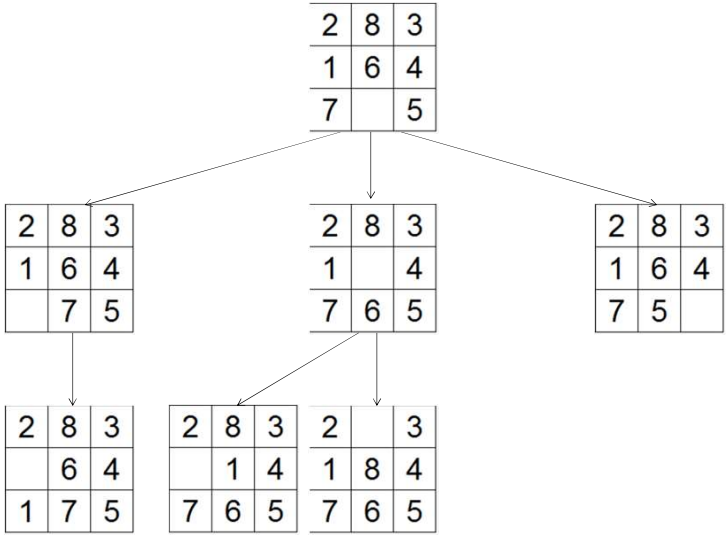


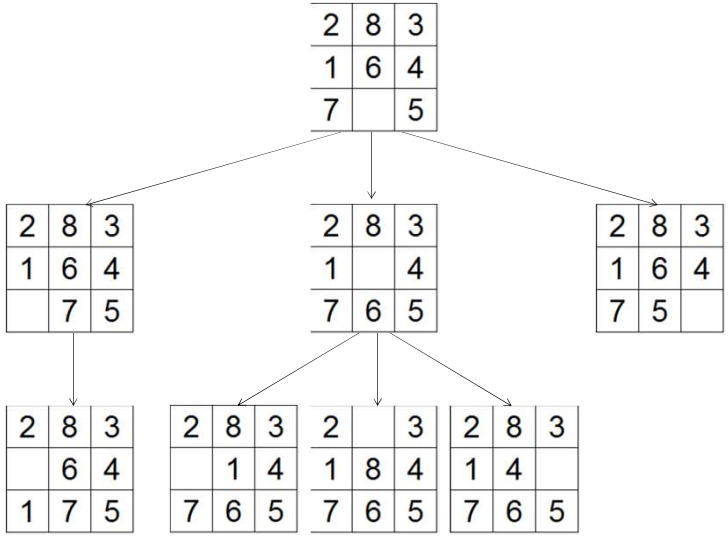


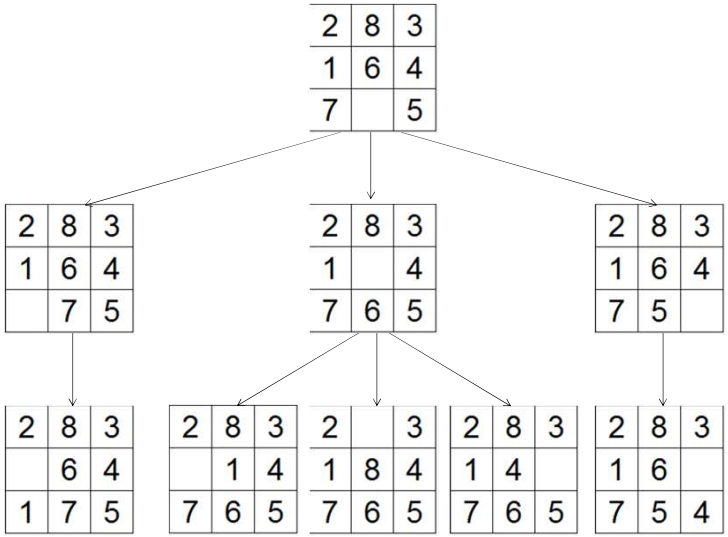



2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2 8 3
6 4
1 7 5


2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2 8 3
6 4
1 7 5
2 8 3
1 4
7 6 5


2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2 8 3
6 4
1 7 5
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5


2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2 8 3
6 4
1 7 5
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5



8-puzzle game #4

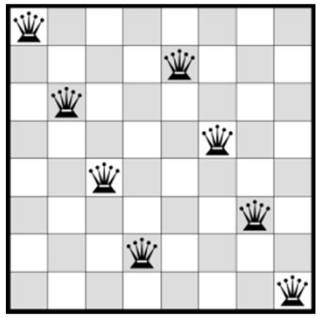
- http://www.brainmetrix.com/arrange-game/

8-queens problem

- There is a chess board given.
- The task is to place 8 queens on the chess board in an order that they do not attack each other.

8-queens problem #2

- An almost good solution:



8-queens problem #3

•In 1848 Gauss gave 72 solutions out of the potential 92.

•Its simplified problem when there is a 4x4 chess board and 4 queens.

•Its generalized problem when there is an nxn chess board and n queens.

8-queens problem #4

•Solution 1: Let us place the queens one after the other on the chess board.

8-queens problem #5

•State space: 0 … 8 queens arbitrary layout on the chess board.

•Initial state: there are no queens on the chess board.

•Operation: place a new queen on to the board.

•Target test: there are all 8 queens on the chess board, and they do not attack each other.

•In this case there are 64x63x … x57 = 1.8×10^{14} sequences to be investigated.

•In case of 100 queens the size of the state space is ~10^{400}.

•Obviously this space is too big to searh and there are too many unnecessary states considered.

8-queens problem #6

•Is there a better solution?

?

8-queens problem #7

•A better solution when it is already excluded from consideration when a queen is attacked.

•States: 0-8 queens on the chess board in an order that they do not attack each other.

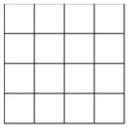
•Operations: a placement of a queen into the leftmost empty coloumn, so as it is not attacked.

•Obviously, it has to be always watched (and updated accordingly) what is the state of the specific block which can now be: free, occupied by a queen, and attacked by a queen.

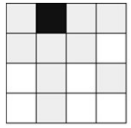
•Computer representation: 8x8 matrix, where

•Ai ∈ {free, occupied, attacked}

4-queens problem



Initial state

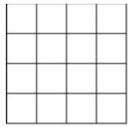


Intermediate state

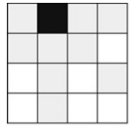
4-queens problem

?

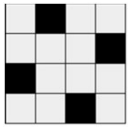
4-queens problem



Initial state

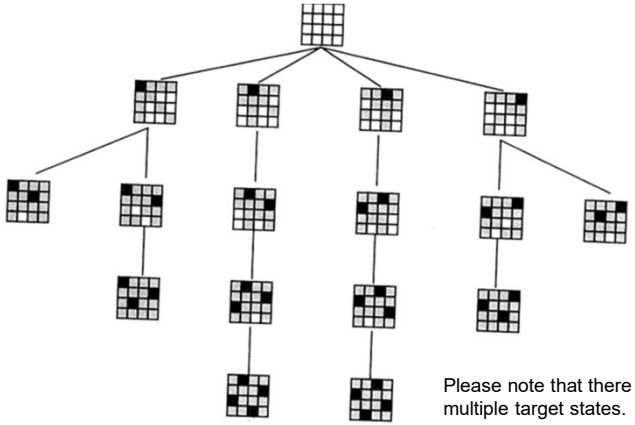


Intermediate state



Target state

Representation with a graph: 4-queens problem



Please note that there are multiple target states.

4-queens problem

- Please note that for this example there are more target states.
- The target is described with a condition or criteria.
- For the case of the city center of Pécs:
to find the Mosque of Pasha Qasim (the so called „dzsámi” in Hungarian) is a taget state but to eat out somewhere downtown, in the city center is a criteria.

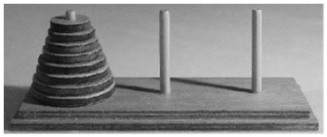
8-queens problem #8

- http://www.brainmetrix.com/8-queens/

8-queens problem #9

- In this case the size of the state space is only 2057.
- For a 10 queen problem, the size of the state space is reduced from 10^{400} to 10^{52}.
- Obviously it means a great reduction of the state space, thus the search will be in a much smaller space.
- But on the other hand the operation is more difficult: after each placement, the board has to be modified!

Towers of Hanoi



•The Towers of Hanoi: there are three poles together with a number of discs of different sizes given. The disc can be put on any pole.

•Initial state: the discs are stacked in order of size on one pole: the smallest is the topmost and the biggest is at the bottom.

•The object of the game is to move the entire stack to another pole, one disc at a time and in an order that no disc can be placed on top of a smaller disc.

Towers of Hanoi #2

•State space: on which pole a disc is placed?

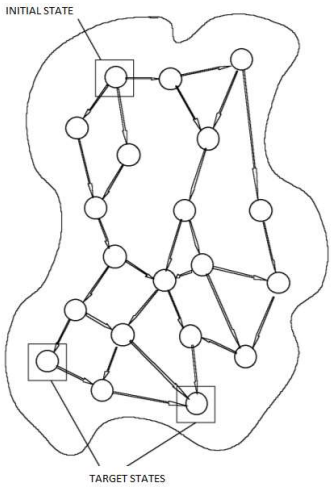
•Operation: replacing a disc while obeying the rules.

•For a 3 discs problem: (1,1,1) → (3,3,3)

Towers of Hanoi #3

•http://www.brainmetrix.com/tower-of-hanoi/

State space rep. : example



Representation graphs

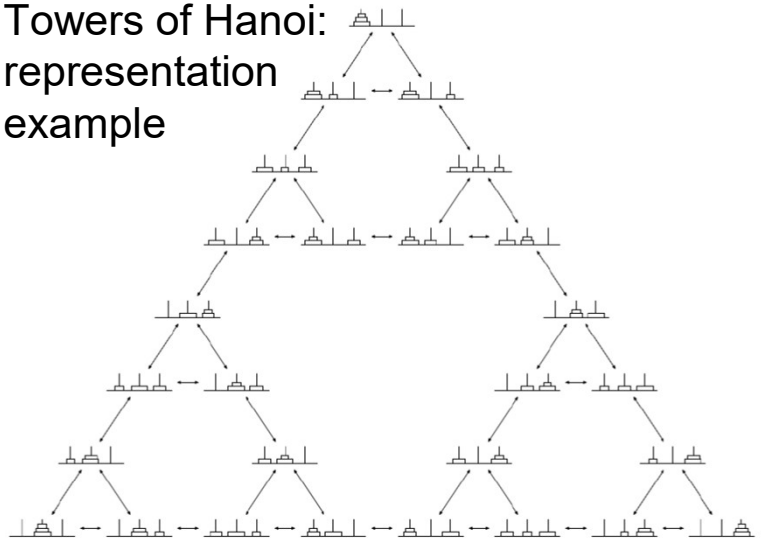
•State space can be illustrated with representation graphs.

•The nodes of the graph are the potential states.

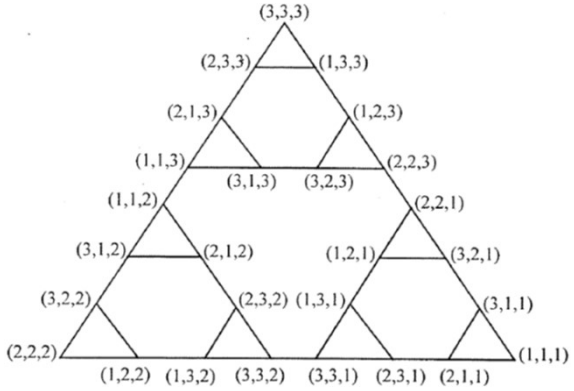
•There is an arc from one node to another node if there is an operation with which we can get from the node to the other node.

•In this graph we have to find the path leading from the start state to the target state.

Towers of Hanoi: representation example



Towers of Hanoi: state space representation graph



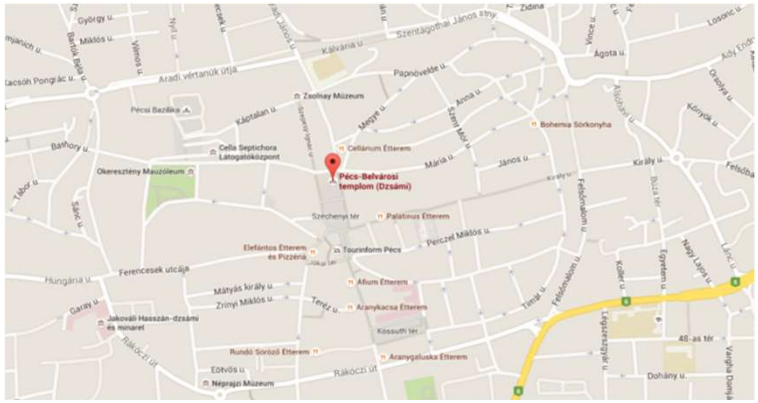
Nilson, 1971.

Towers of Hanoi: Costs

•Please note that a cost can be assigned to an operation. For example: a crane has to be used for moving the larger discs.

Addresses at Pécs

•Target: to find the way, a path leading to the „Dzsámi.”

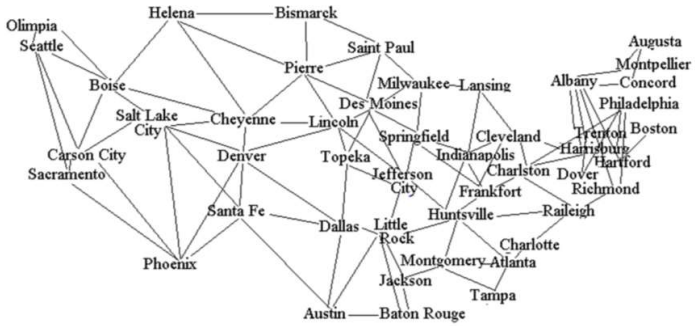


Addresses at Pécs #2

•Initial state:
•State space:
•Target state:
•Representation:

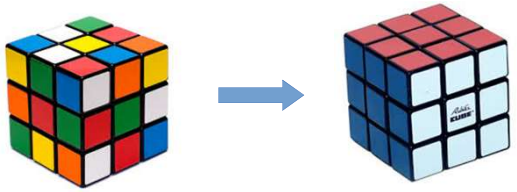
?

Representing maps: example



Rubik's cube

•A 3D problem.



•http://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/

Cannibals and the missionaries

•There are 3 missionaries (m) and 3 cannibals (c) at the bank of a river.

•A boat is crossing the river in both directions. The boat can carry maximum 2 people. Obviously, the boat without no people onboard cannot cross the river.

•There is a constraint on non-cannibalism (#m >= #c) on each bank and in the boat.

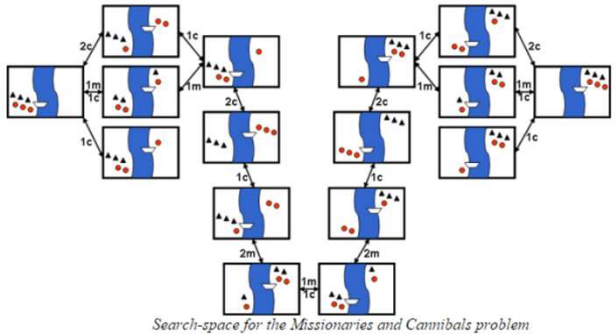
•Target: get both people to the other side of the river.

•A detailed solution was published by Amarel, S. (1968). On representations of problems of reasoning about actions, Machine Intelligence, (3), 131–171.

Cannibals and the missionaries #2

http://www.plastelina.net/examples/games/game2.html

Cannibals and the missionaries #3



•https://www.youtube.com/watch?v=IaLS8gHzROg

•http://www.aiai.ed.ac.uk/~gwickler/missionaries.html

Water measuring

•There are 2 water containers: a 3-liter and a 4-liter vessels.

•At the beginning both of them are empty.

•There is a well with enough water to fill the vessels at any time.

•How can we have 2 liters of water in the 4-liter vessel and the other one empty?

Water measuring: state space

•State space:
•Initial state:
•Target state:
•Operations:

?

Water measuring: state space

•Let the (x,y) pair be given, where
x = 0,1,2,3 or 4 and
y = 0,1,2 or 3.
•In other words, x represent the water in the 4-liter vessel and y represents the water contained in the 3-liter vessel.
•Initial state: (0,0)
•Target state: (2,0)
•Operations:

?

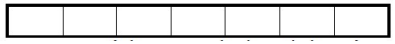
Measuring 2

•There are 3 water containers: a 5-liter, a 3-liter and a 2-liter vessels are available.
•At the beginning we have water in the 5-liter vessel.
•How can we have 1 liter of water in the 2-liter vessel?

?

Black and white example

•There are 7 compartments, with 3 black stones, 3 white stones and one empty compartment:



•A stone may be moved to the empty compartment right next it; or a stone may jump over another stone to the empty compartment.
•Target: all white stones should precede all black stones, while the empty compartment position does not matter.

?

State space representation

Concluding remarks

Complexity of the state space

The success in finding a solution is influenced by the representation of the state space; i. e. it is influenced by the

- Number of nodes,
- Number of edges,
- Whether there are loops in the graph.
- Therefore it is advisable to perform all possible simplification.

Simplifications

- To reduce the number of nodes considered – the state space can be significantly reduced with a smart representation.
- To reduce the number of edges starting from a node – the proper selection of the range of the interpretation of the operation is important.
- To eliminate loops and circles of the graph and to transform the original graph into a tree.

Transformation into trees

- If between two nodes there are two directed arcs, then let us delete the arc that is directed backwards, i.e. towards the start node. In the resulting graph, all arcs will lead from the start node towards a more distant node.
- Let us assign as many distinct nodes to a single state, as many times the state can be reached from the available nodes.
- Problems that may arise:
  - Too many nodes;
  - Endless branches.

Transformation into trees #2

•Sometimes it is not advisable to transform the graph into a tree.

•In case there are many short circles in the original graph, or the number of nodes grows exponentially in the tree generated, then it is useless to perform this transformation.

Transformation into trees: The travelling salesman problem (TSP)

•There is a list of cities.

•The salesman has to travel to each city exactly once.

• There is a direct connection between each pair of cities.

•The target is to find the shortest path that returns to the original city and visits each city exactly once.

Let us recall…

•Let G=(V,E) be a directed graph.

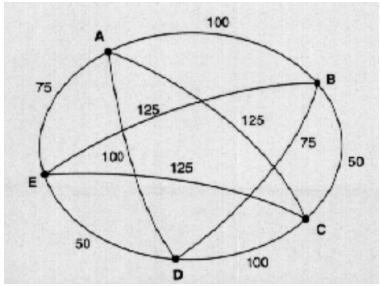
•The travelling salesman problem was mathematically formulated by the Irish mathematician W.R. Hamilton in the 1800s.

•A Hamiltonian circle visits each node (which is an element of V) exactly once.

•To determine whether there is a Hamiltonian cirlce in the graph is NP-complete.

http://compalg.inf.elte.hu/~tony/Oktatas/Osztott-algoritmusok/P-NP-NPC.pdf

Travelling salesman (TSP)



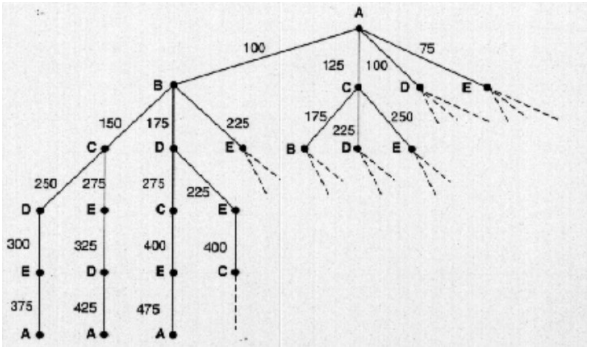
•States: cities.

•Operation: get from here to there.

•Target: find the path.

•NB: please note that there is a cost!

Travelling salesman #2



•Should the graph be transformed into a tree…

•Multiple solutions may be read.

Travelling salesman #3

•Simple algorithm:
Let us consider all possible solutions and select the one with the minimal cost.

•This solution may be used for small size problems, however, with enlarging the number of cities, the computational time required grows rapidly.

•In case of N cities, there is N! paths to be calculated. In case of 10 cities, 3628800 cases should be calculated.

Search

Introduction

Guess vs Mechanisms

- A lesson of problem representation: there can be too many potential states; i. e. the state space can be extensively large.
- In a large space sometimes it is very difficult if not possible to find the solution.
- Instead of a lottery-guess (even a blind hen can find a grain), one may apply a search mechanism to find the goal.

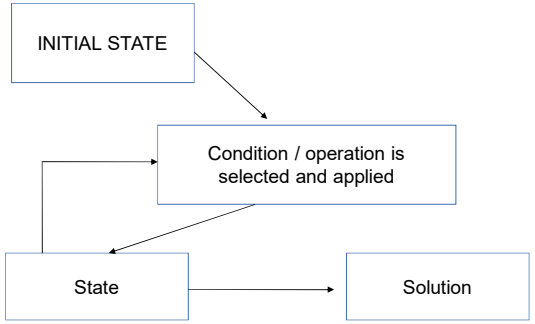
Search

Search is an algorithm, which

- Stores the information revealed or part of it;
- Identifies when the target is reached;
- Identifies potential steps at the given point;
- Decides which step is to be performed at a given point;
- Modifies the information stored after the step.

Search #2

- Search is an algorithm or group of algorithms.



Components of search systems

1 Global work space (or database).

- NB. Not equal to the state space!, since some information is available from the search itself.
- (Declarative) Knowledge gained and stored through the search:
  - Initial value;
  - Termination condition;
  - Walk until the state; for exmaple a subgraph.
- Generally, part of the representation graph of the problem gained during the search.

Components of search systems #2

2 Rules

- Prodution rules;
- Procedural knowledge;
- Operators modifying the global work space.
- NB. Rules is a larger set than operations (which describes how we get from one state into another).

3 ???

- Before the third component, let us consider the algorithm:

Components of search systems #3

Search algorithm:

```
Procedure SEARCH_SYSTEM
 DATA := {initial state}
 while DATA does not satisfy termination rule do
      select R from DATA
                /* R is one rule that can be applied on DATA
      DATA := R(DATA)
                /* apply R on DATA
    enddo
end
```

?

3rd component?

Components of search systems #4

3 Control strategy

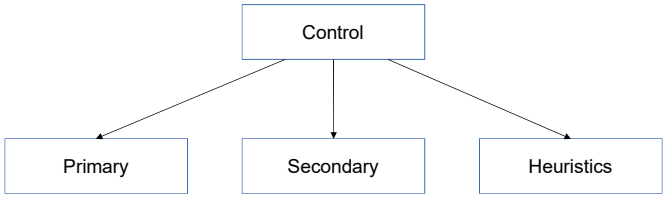
- Selects one from the applicable rules.
- Describes the aspects of the selection.
- Control strategy is based on
  - General principle selecting the rule to be performed;
  - Control knowledge resulting from the specific problem.

Components of search systems #5

Summary of the components:

- Global work space
- Rules
- Control strategy

Control

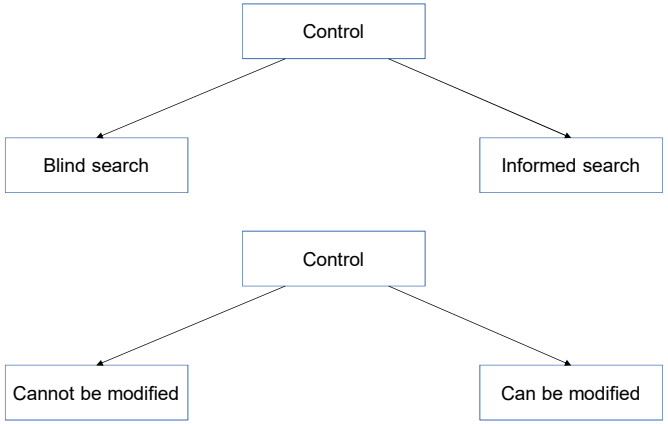


Independent from the problem.

Independent from the specific problem but it relies on its representation model.

Applies information resulting from the specific problem not included in the representation.

Control #2



Control #3

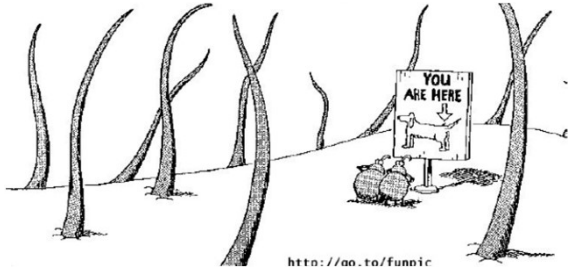
The search is not informed (noninformed search):

- There is no information about how we get from the actual state into the target state, what is the number of necessary steps, what is the cost etc.

Control #4

The search is informed:

- There is a special additional information available concerning the problem considered; i. e. heuristics that seems to be good.



Control #5

The control cannot be modified:

- The decisions made during the search cannot be withdrawn.
- It is not possible to return to a previous decision point and make another decision.
- Example: quadratic equation solving algorithm.

The control can be modified:

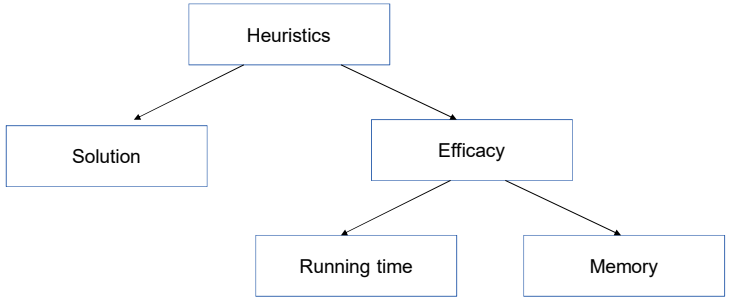
- Previous decisions can be modified.
- One can return to a previous decision point and make another decision.

Heuristics

- Heuristics, heuristic or heuristic technique.
- Ancient Greek word: find, discover.
- A method that seems to be good enough and practical but the optimality is not guaranteed.

Heuristics #2

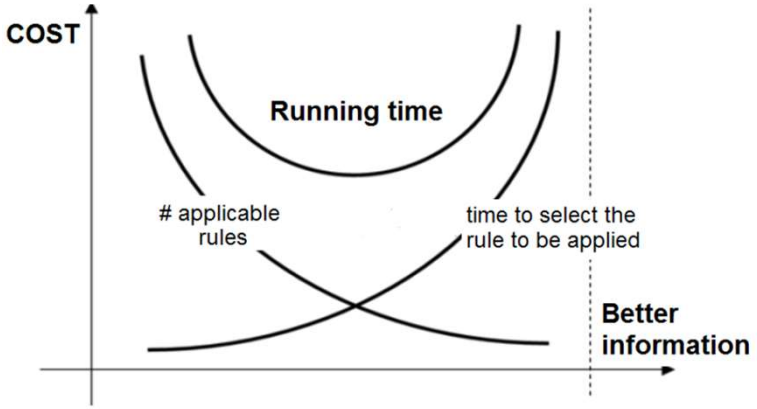
- Heuristics that seems to be good, does not guarantee the optimality.



Heuristics #3

- The more specific information about the problem is involved in the search, i. e. more sophisticated heuristic is used, the system is more informed, thus it requires more time to select the rule to be applied, i. e. the cost of the control strategy rises while the number of applicable rules is decreased.
- On the other hand, in case of fewer information, when a simpler heuristics is used, selection of a rule is faster but more trials are necessary; while the number of applicable rules are increasing.
- At both ends the cost of the running time is big, while one or the other component is the dominant. The optimal running time is somewhere in the middle, where the algorithm has already a good enough heuristics but not yet too informed (see figure).

Heuristics #4



NILSSON 1982.

Search without modifications

Local search

- Local searches are searches without modifications.
- It is not important to know the walk or the path to the target, only the target.
- In general, only the current state is considered and one of its neighbours will be selected.
- A cost function is considered to select the neighbour.
- Application fields:
  - To find an element with a given characteristics.
  - To find an optimum of a function.

Local search #2

Advantages:

- Use little amount of memory.
- May find an acceptable solution even in very large or even infinite (continuous) search space.

Disadvantages:

- Not systematic searches.
- Finding a solution is not guaranteed.

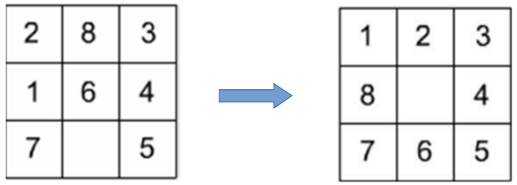
Hill climbing

- We consider a real function for the state space, i.e. to every single state a number is assigned.
- This function is at its maximum at the target state.
- The algorithm is a simple cycle, always selecting a neighbour state with a greater value and terminating when it reaches the maximum. In case of multiple greater values it selects the neighbour with the greatest raise.

Hill climbing #2

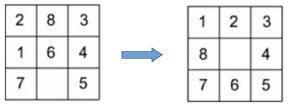
- Example: 8-puzzle game.
- What can be the cost function?

?



START TARGET

Hill climbing

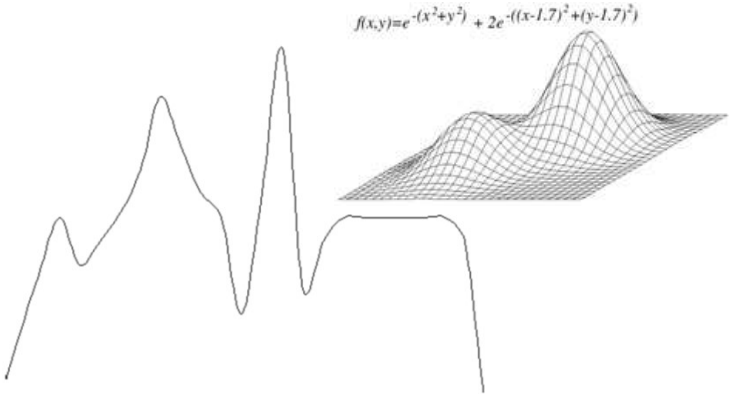


Start Target

Example: 8-puzzle game; potential cost functions.

- To add all numbers – wrong solution: no difference between the states.
- How many blocks/ numbers are at their target position? Then a maximum search has to be performed.
- How many blocks are not at their target position?
- How many times do we need to move the blocks to get to their target position? For example, block 8 has to be moved 2 times, while block 2 has to be moved only once.
- NB. The neighbour with the greatest value is selected.

Hill climbing: problems



Local maximum and plateaus.

Hill climbing - improvements

- Hill climing with arbitrary restart: the algorithm restarts from a randomly selected initial state.
- Local beam search: when a group of states are considered together and the next state is selected from all of their neighbours:
  - k states are watched and the k best states are stored;
  - Randomly selected k states serve as initial state.
  - The information is divided, and the algorithm heads towards the more promising peak points.

Hill climbing – improvements #2

- Tabu search:
  - Previously investigated k best peaks are also stored; where it is forbidden to step again. As a result from the local maximum the algorithm may find a way out.
- Etc.

Search with modifications

1. Backtracking

Backtracking

- We start at one path of the representation graph and go as far as we can, i. e.
- Until the target state is reached or
- It is not possible to go further, then we step back once and search for a new path.

Backtracking #2

Global work space:

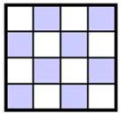
- The path between the initial node and the actual node; together with the edges not yet tried (i. e. the applicable rules).
- At start: the path with zero length containing the start node.
- At termination: either the path from the start or the start + no further edges to be selected.

Backtracking #3

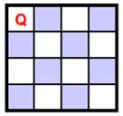
The algorithm:

- The production rules, i. e. search criteria:
  - The operations valid within the state space; plus
  - Adding a new, not yet tried edge to the stored path or deleting the last edge (rule of bakctracking).
- The control strategy:
  - Apply backtracking only when nothing else is possible.

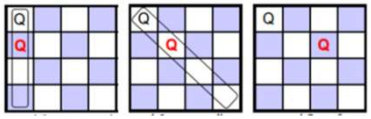
Backtracking example: 4 queens



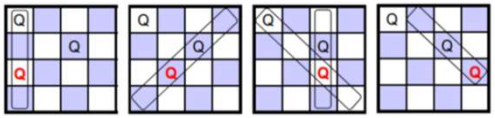
Start: empty chess board.
Let us put a queen on the board: into the next row from top to bottom and from left to right.



We place the first queen on the board (top left).

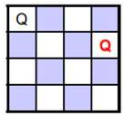


We place the second queen.

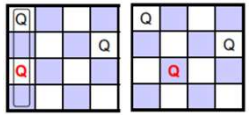


We try to place the third queen. But there is no possible place!
Backtrack!

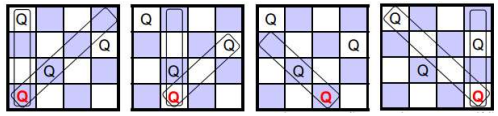
Backtracking example: 4 queens #2



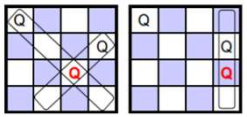
We place the second queen to a new position.



We place the third queen.

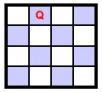


We try to place the fourth queen. But there is no possible place!
Backtrack!

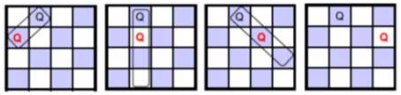


We try to place the third queen to a new position. But there is no possible place!
Backtrack!

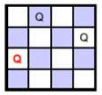
Backtracking example: 4 queens #3



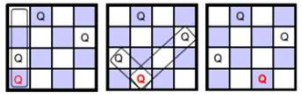
We try to place the second queen to a new position but it is not possible. **Backtrack!**
We place the first queen to a new position.



We place the second queen.



We place the third queen.



We place the fourth queen.
A solution found!

See for example, http://www.fas.harvard.edu/~cscie119/lectures/recursion.pdf

Backtracking example: 4 queens #6

Tree representation of the backtracking example.

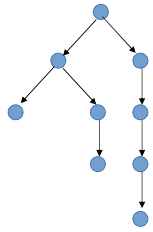
0 queen

1 queens

2 queens

3 queens

4 queens



Solution found.

Backtracking example 2

- Example: how do we get to the railway station at Pécs?

Backtracking algorithm

```
Procedure Backtracking(node)
  if target(node) then return (null)
  Rules := list of rules that can be applied at a node
  loop while not empty (Rules)
     R := first_element(Rules)
     Rules := remainder(Rules)
     newnode := R(node)
     solution := Backtracking(newnode)
     if solution ≠ error then return (add(R,solution))
  endloop
  return(error)
end                Call: solution := Backtracking(start)
```

Backtracking algorithm #2

- This algorithm either returns with an error or
  in case it is not an error, then it starts to connect the elements
  and forms a list.

Backtracking algorithm
a nonnrecursive version #1

```
Procedure backtracking
   Path := [start]; Nodes := [start];
  Bad := []; node := start;
  while not empty (Nodes) do
  if target(node) then return (Path)
  if there_is_child(node) (except those in Bad, Nodes, or Path)
    then
       place all children of the node (except in Bad, Nodes or Path) into the
         Nodes
```

Backtracking algorithm
a nonnrecursive version #2

```
      node := first_element(Nodes)
         add(node, Path)
       else
         while not empty (Path) and node is the first element of Path do
            add(node, Bad)
            delete first element of Path
            delete first element of Nodes
            node := first element (Nodes)
         enddo
         add(node, Path)
  endif
  return(fail)
end
```

Backtracking condition

- Dead end, i. e. from the end point there is no path leading out.
- An intermediate point from where each path is leading to a dead end.
- Statement. If there is a solution for finite, directed graphs where there are no circles, this algorithm finds one solution.
- A disadvantage of the algorithm: it stores only the actual subgraph.

Additional information

- One can add some information to this algorithm, some modification, to improve efficacy.
- Example: how do we get to the railway station at Pécs? For example when we reach the highway, we know already that it is not the right path.
- In other words: the path under consideration should not be longer than a depth constraint.
- This type of algorithm is often called depth limited search.

Constraint on depth

```
Procedure Backtracking(Nodes)
  node := first_element(Nodes)
  if element(node, remainder(Nodes)) then return (error)
  if length(Nodes) > Constraint then return (error)
  if target(node) then return (nil)
  Rules := list of rules that can be applied at node
  loop while not empty (Rules)
     R := first_element(Rules)
     Rules := remainder(Rules)
     newnode := R(node)
     Newnodes := add(Newnode, Nodes)
     solution := Backtracking(Newnodes)
     if solution ≠ error then return (add(R,solution))
  endloop
  return(error)
end
```

Constraint on depth #2

- Statement. The above given modified backtracking algorithm always terminates. In case there is a solution not longer than the depth constraint, the algorithm finds a solution.
- Further improvement can be for example an iterative deepening search. The depth constraint is progressively increased, until the solution is found.

Additional heuristics

- Heuristics based on the order: paths leading out from a given node is investigated according to a given order.
- The depth constraint is estimated: an upper bound on the solution's length is given.
- Cut methods: edges (and paths) leading out from an endpoint are marked, which does not worth to be investigated is cut from the graph. For example, in case there is an estimation about the number of steps to the soultion from the point and it is more than the depth constraint; or we somehow know from experience that there is no further path out of the point etc.

Pros and cons

- Advantages: easy to implement, small amount of memory is required.
- Disadvantages:
  - Optimal solution is not guaranteed (there are modifications where optimality can be guaranteed).
  - A bad step at a higher level can only be corrected after a large number of steps and after a large number of backtracking.
  - A node already considered via a different path from where the algorithm has already backtracked is considered unknown, i. e. a dead end can be traced multiple times.

Search with modifications

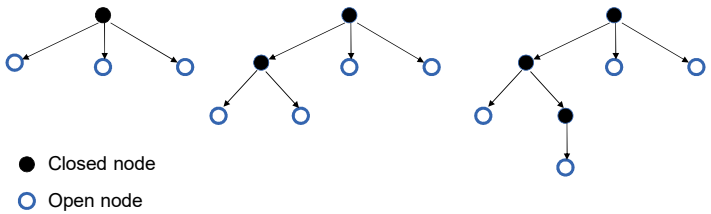
2. Graph search

Graph search

- Another type of search with modifications is called graph search.
- It searches for a subgraph, not a path.
- At each node every children is investigated and inserted into the search graph.
- At each step the best (or the one that seems to be best) not yet expanded node is selected to continue the search.
- The search ends when the target node gets into the search graph or there are no further applicable rules.

Graph search: closed and open

- We start from the initial state.
- All potential next states are considered.
- The best next state is selected.
- Closed and open lists / nodes are considered.



Graph search and representation graph

- NB: It is not a representation graph.
  It contains only the investigated subgraph.
- NB: in case there is no solution, then this graph and the representation graph will be equivalent.

Graph search algorithms

- Global work space (global database):
  - The search graph G, which is a subgraph of all paths already investigated from the starting node.
  - Initial value: starting node.
  - Termination: target node is an element of G or there are no further rules.

Graph search algorithms #2

- Rules:
  - Γ(node): generates all children of a node and expands the search graph
- Control strategy:
  - Expand the best node.
- An open node can be expanded and the set of nodes that can be expanded: OPEN.

Graph search algorithm

```
Procedure GRAPHSEARCH
   G := {startnode}
   Open := {startnode}
   cycle while not empty(Open)
      node := element (Open)
      if target(node) then return …
      Open := Open \ {node}
      G := G U Γ(node)
      Open := Open U Γ(node)
   end cycle
end
```

Graph search main questions

- Are there any circuits in the graph?
- How to select the next open node to be expanded?
- What the algorithm returns? „there is a solution," or the target node itself
- How can we generate the solution path?
- Is it the optimal solution what is returned? Can an optimal solution be found? What does optimality mean?

Some remarks before the answers

- Spanning tree: if there is only one path leading to each node in a graph, then we get a spanning tree.
- How can we know that there are no other paths to a node? We have to remember who is the parent of the node, i. e. pointers have to be introduced:

  p(node): points to the parent of the node.
- NB. the cheapest path of all found paths has to be represented by the pointers.

Cost function

- The cost function gives the cost of the path within the spanning tree.
- From the start to the node n the cost of the path: g(n).

- Let g*(n) denote the minimum cost of all possible paths from the start node to node n, i. e. the cost of the underlying (overall) graph. In this case:
  $g^*(n) \leq g(n)$
- NB. g*(n) may not be known.
- NB. g*(n) belongs to the underlying graph and not to a tree which is only its subgraph.

Cost function #2

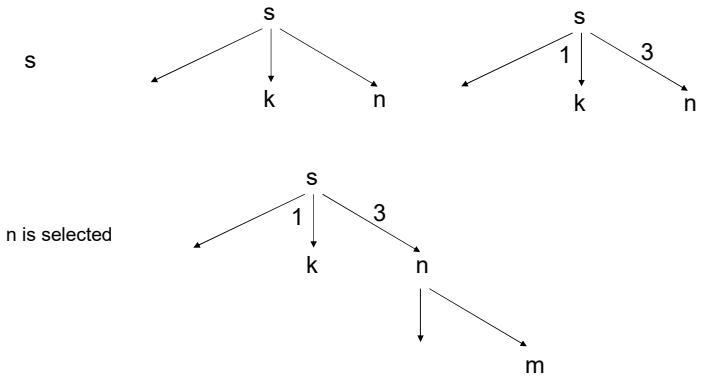
- Optimal spanning tree: optimal path is leading to each node.
- Evaluation: we select the node where the cost function is minimal.

Cost function #3

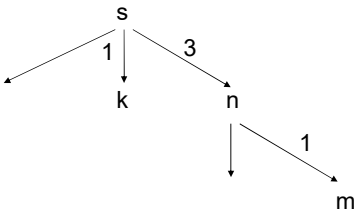
- Calculation of the cost function and the pointers:
  - $g(m) = g(n) + c(n,m)$

    where n node is expanded, m is the child of n, and c(n,m) is the cost of the edge (n,m).
  - Pointer p(m) points to node n.

Cost function: example

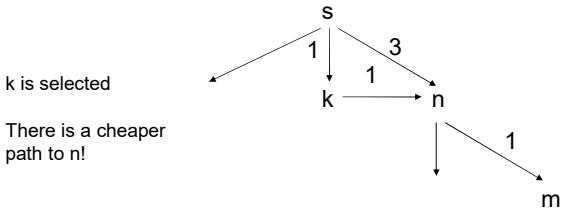


Cost function: example #2



g(m) = 3 + 1 = 4
p(m) = n
p(n) = s

Cost function: example #3



g(m) = 3 + 1 = 4
p(m) = n
p(n) = s

g(m) = 1 + 1 + 1 = 3
p(m) = n
p(n) = k

Problem

- When regenerating node m, it may already be closed. In other words, it may already be expanded earlier and thus have children in the search graph.
- Moreover, this subgraph may be extensively large.
- To check each node of the subgraph (under node m) and set all their pointers would be necessary.

Proposed solutions

1 All nodes that can be reached from node m is revisited, checked and set. - This can be exhaustive.

2 Evaluation criterai is carefully selected so as the order to expand the graph will not require the resetting of the pointers. Please note that this in general cannot be accomplished. In special cases it can be done by heuristics. But on the other hand, selection of the rule may become too expensive.

3 If the procedure generates an already existing closed node m, then m is considered to be a new node without investigated children. - This proposal will be considered in the future slides.

3rd proposed solution

- If the procedure generates an already existing closed node m, then m is considered to be a new node:
- m is put into the list of open nodes.
- m might be expanded multiple times by the search, i. e. it is not so effective. Nevertheless, more effective versions may be available.

Values corresponding to children

- n: parent; m: child.
- If m is not an element of G, then
  p(m) := n, g(m) := g(n) + c(n,m)
  OPEN := OPEN U {m}
- If m is an element of G and g(n)+c(n,m)< g(m), i. e. it is a cheaper path, then
  p(m) := n, g(m) := g(n)+c(n,m)
  OPEN := OPEN U {m}
- If m is an element of G but the path is not cheaper, then do not do anything.

Procedure

```
Procedure GRAPHSEARCH
   G:= {startnode}; Open := {startnode}
   g(startnode) := 0; p(startnode) := nil
   cycle while not empty(Open)
      n := min_f(Open)
      if target(n) then return solution
      Open := Open \ {n}
      cycle for every m element of Γ(n)
         if m is not an element of G or g(n)+c(n,m)<g(m) then
            g(m) := g(n)+c(n,m); p(m) := n; Open := Open U {m}
      end cycle
      G := G U Γ(n)
   end cycle
   return there is no solution
end
```

Statements

The general graph search algorithm

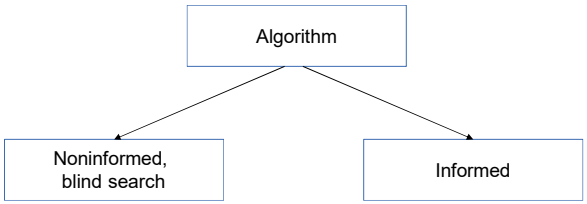
- In case of finite representation graphs always terminates.
- A node is expanded at most a finite number of times; i. e. it is not circuit-sensitive, because of the costs.
- If there exists a solution in the finite representation graph, then the algorithm terminates with finding a target node.

Remark

- In special cases the search graph is a tree.
- In this case pointers have less importance and obviously they do not change during the search since each node has at most one parent.

Graph search algorithms

- Evaluation function f is the key where these algorithms differ from each other.



Noninformed Search

Noninformed graph search

- There is no specific knowledge related to the problem considered and inserted into the evaluation function f.
- The order to expand the graph is preliminary fixed, undependently from the problem.
- In other words, the search is based on the characteristics of the graph, and the shape of the graph.

Depth of a graph

- Let graph G be given with its node m.
- Let d(m) denote the depth of node m:
  - d(s) := 0 where s is the start node;
  - d(m) := d(n)+1 where n is a parent of m.

- Searches based on the shape of the graph:
  - Breadth-first search
  - Depth-first search

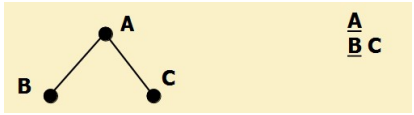
Breadth-first search

- The node at the topmost level is expanded.
- Formally:
  - f(n) := d(n)
  - the minimum of f is sought.

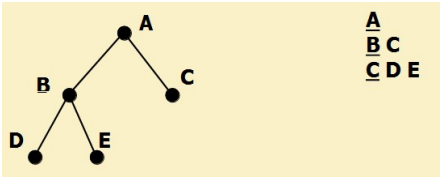
- Statement:
  Breadth-first search always finds a solution, if there is a solution. (Even in case of infinite grpahs.)

Breadth-first search example

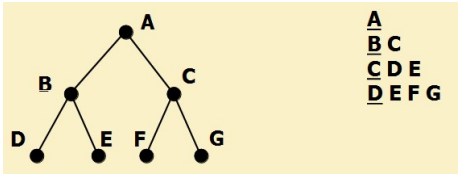
Breadth-first search example



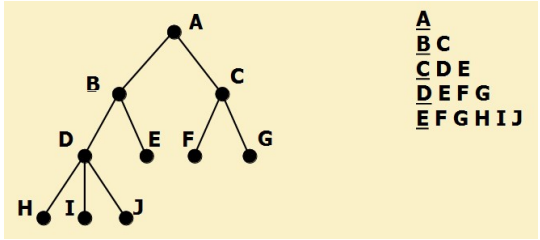
Breadth-first search example



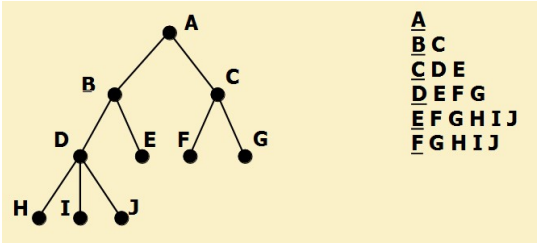
Breadth-first search example



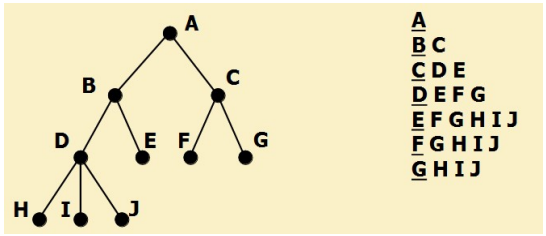
Breadth-first search example



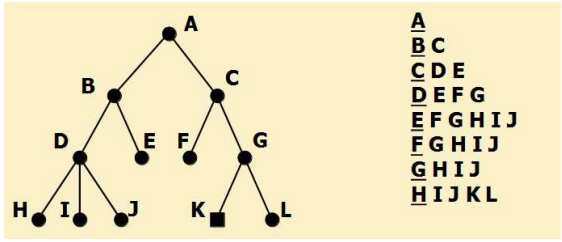
Breadth-first search example



Breadth-first search example

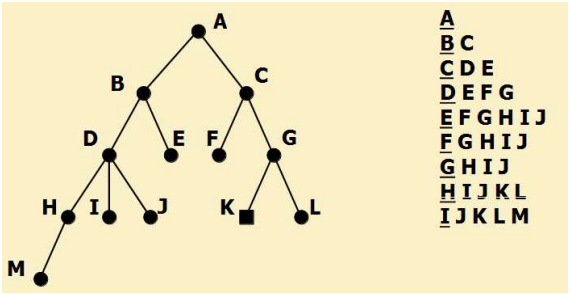


Breadth-first search example

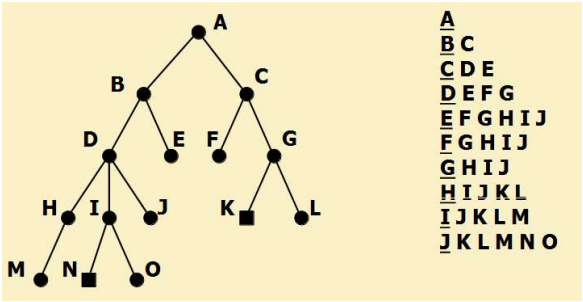


The elements of the OPEN list can be checked when a new element is entered or when the element is selected.

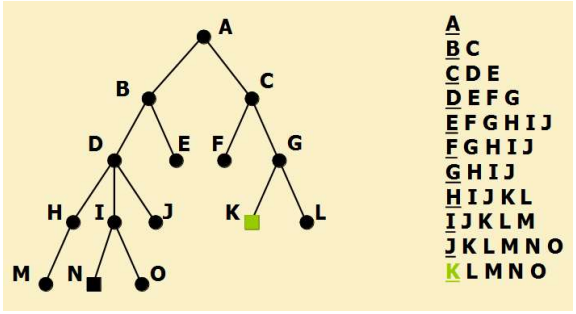
Breadth-first search example



Breadth-first search example



Breadth-first search example



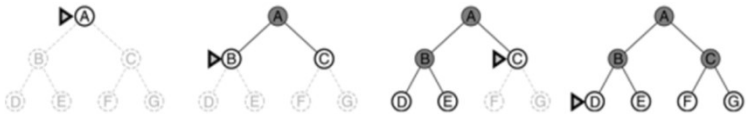
Breadth-first search pros and cons

- Advantages: easy implementation.
- Disadvantage: efficiency.
- Breadth-first search finds the shallowest node; i. e. the terminal node found will be the closest one to the start node. Please note that it might not mean that the solution is optimal.
- Breadth-first search finds the optimal solution in case the cost of a path to a node is a non decreasing function of the depth of the node (for example each activity has the same cost).
- In general the cost of the search:

the cost of the path + the cost of the procedure.

FIFO list of the breadth-first search

- FIFO list: first-in-first-out list



Time complexity of BFS

- Let us consider a hypotetical state space where expanding a state b new states are generated:
  - First level: b nodes.
  - Second level: b^2 nodes.
  - Third level: b^3 nodes.
- If a solution has a length of d, then the maximal number of nodes expanded:
  $b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b)$
  since only the terminal node is not expanded and all other nodes are expanded.
- This means that the time complexity is exponential: $O(b^{d+1})$

Time complexity of BFS example

Depth	Nodes	Time	Memory
2	1100	0,11 sec	1 MB
4	111100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 TB
10	10^{11}	129 days	101 TB
12	10^{13}	35 years	10 PB
14	10^{15}	3523 years	1 EB

Depth-first search

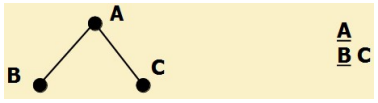
- The node at the deepest level is expanded.
- Formally:
  - f(n) := -d(n)
  - the minimum of f is sought.
- LIFO (last-in-first-out) or stack can be considered for the OPEN list.

Statements:

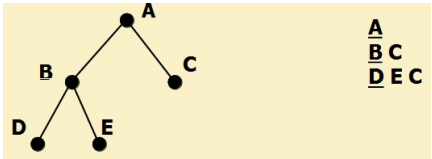
- Depth-first search may not find the solution. Example: we search the city center and we pass by and we are somewhere already in Greece.
- Optimality is not guaranteed.

Depth-first search example

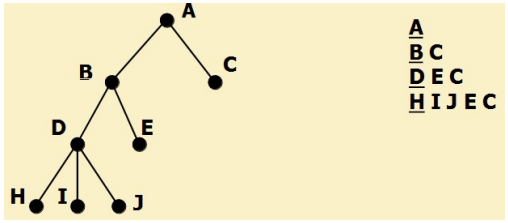
Depth-first search example



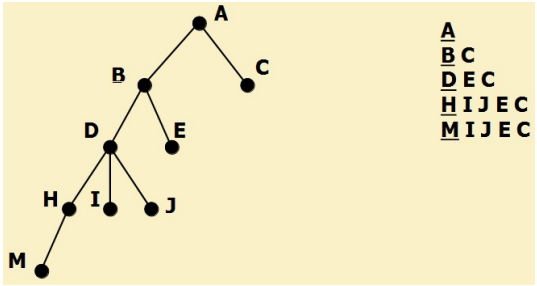
Depth-first search example



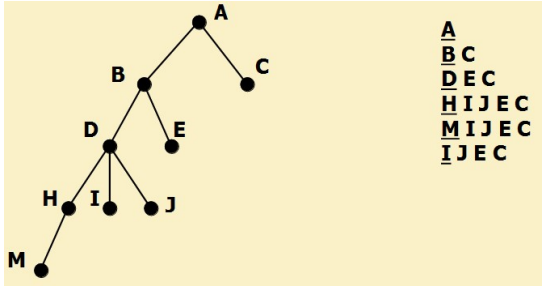
Depth-first search example



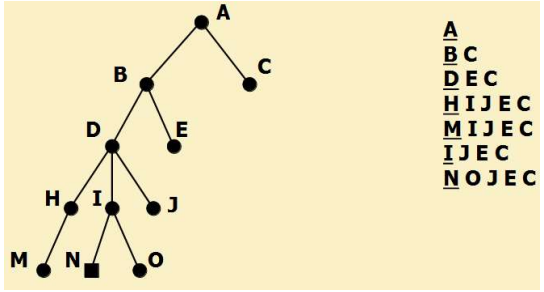
Depth-first search example



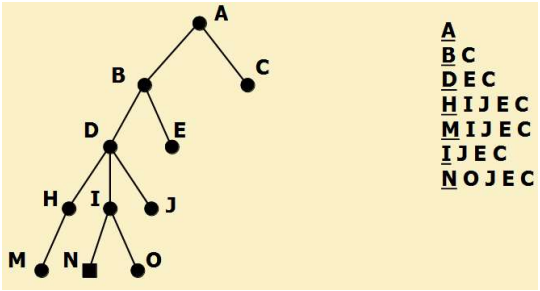
Depth-first search example



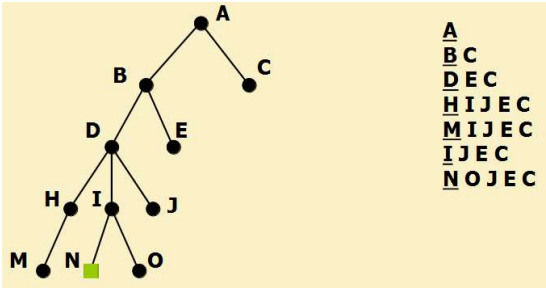
Depth-first search example



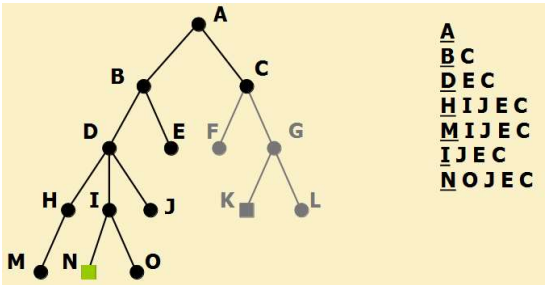
Depth-first search example



Depth-first search example



Depth-first search example



NB. Can be better or faster than the BFS.

Backtracking and DFS

In general:

- Backtracking can be considered as a special case of DFS.
- Backtracking uses less memory, since it generates only one of the next possible states instead of all next states.
- DFS stores the expanded paths and thus does not re-expand an already expanded dead end.

Depth-first search pros and cons

- Advantages: easy to implement and uses small amount of memory.
- Disadvantages: optimality of the solution is not guaranteed moreover, finding a solution is not guaranteed.

Time complexity of DFS

- Let us consider a hypotetical state space where expanding a state b new states are generated. The state space has the maximal depth m.
- Time complexity: $O(b^m)$

- In cases where there are a lot of solutions breadth-first search can be faster than DFS, since exploration of a smaller part of the whole state space may already result in a solution.

Improvements of DFS

- Depth limited search;
- Iterative deepening search;
- Etc.

Depth limited search

- Depth-first search with a certain limit on the depth.
- Always the node at the deepest level is expanded provided that the node is not under the given depth constraint.
- Example: the railway station has to be within 100 meters.
- Statements:
  - In case there is a shorter solution than the constraint, the algorithm finds it.
  - The solution may not be optimal.

Iterative deepening search

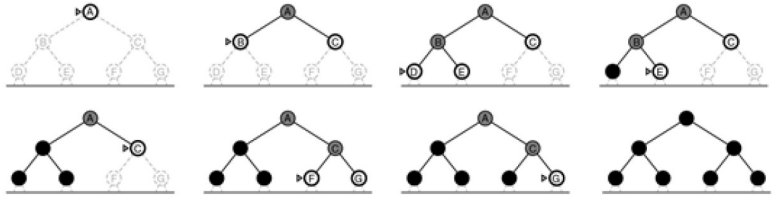
- The algorithm checks all constraints on the depth: first k=0, then k=1, K=2 etc. and performs a DFS.
- It combines the advantages of both the breadth-first search and the depth-first search:
  - It is complete (like the BFS);
  - It requires small amount of memory (like DFS);
  - The order of expanding the nodes is similar to the BFS, nevertheless the algorithm may expand nodes multiple times;
  - The search tree will not be explored more deeply than the depth of the solution.

Iterative deepening search example

Constraint: 0

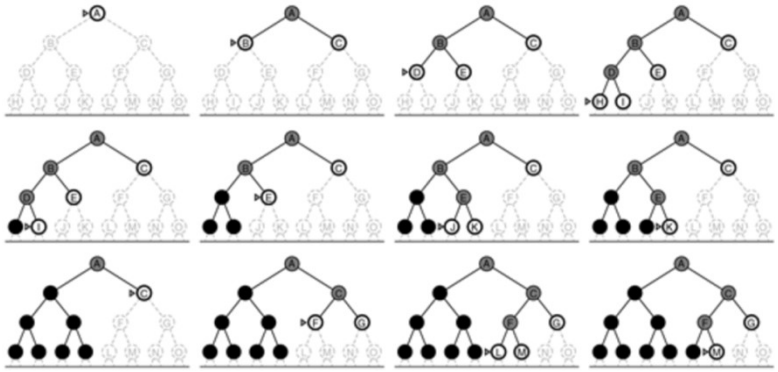
Constraint: 1

Constraint: 2



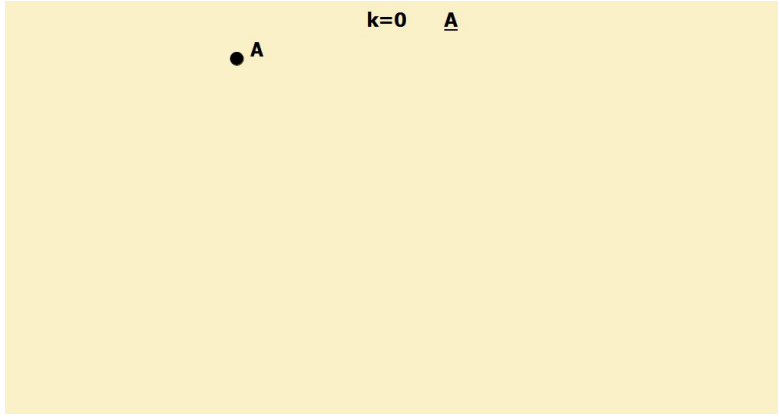
Iterative deepening search example

Constraint: 3

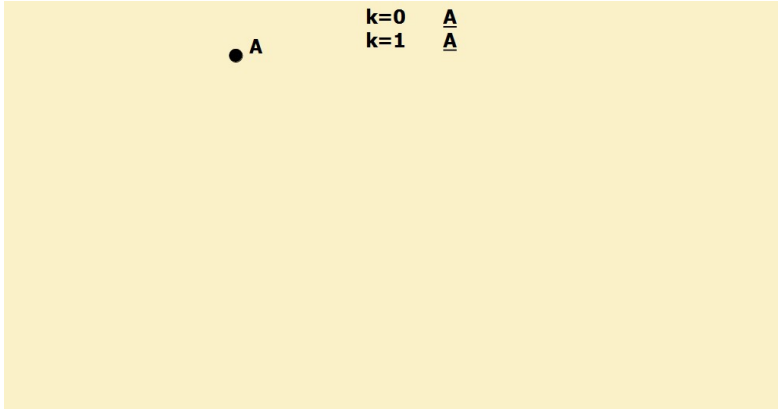


IDS: example 2

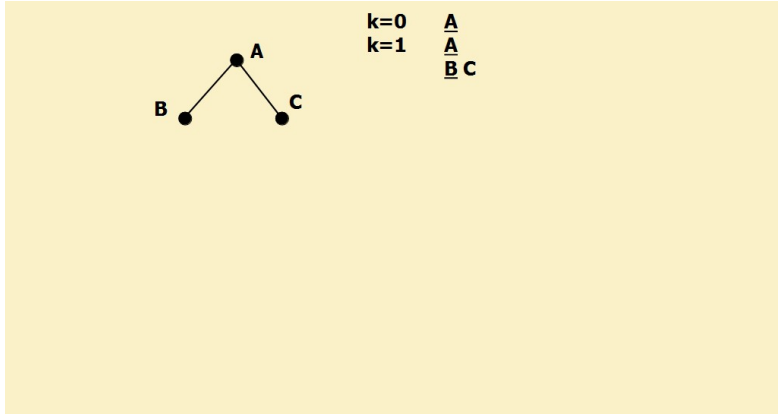
IDS: example 2



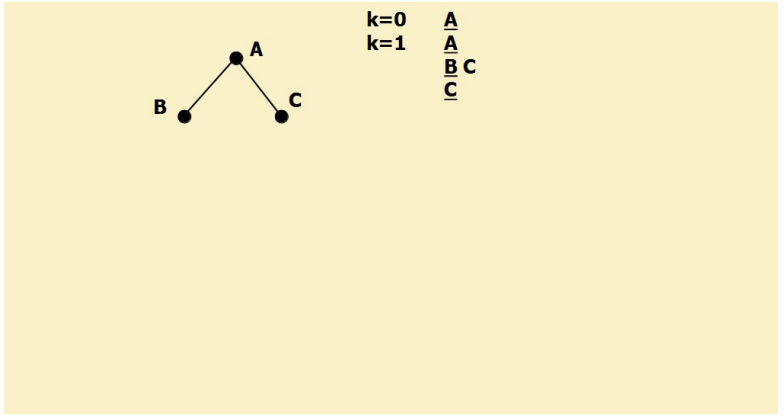
IDS: example 2



IDS: example 2



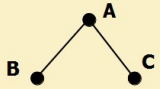
IDS: example 2



IDS: example 2

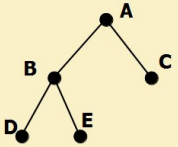
k=0 A
k=1 A
B C
C
k=2 A

IDS: example 2



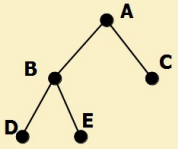
k=0 A
k=1 A
B C
C
k=2 A
B C

IDS: example 2



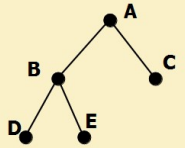
k=0 A
k=1 A
B C
C
k=2 A
B C
D E C

IDS: example 2



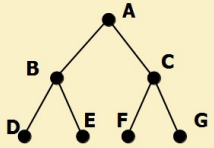
k=0 A
k=1 A
B C
C
k=2 A
B C
D E C
E C

IDS: example 2



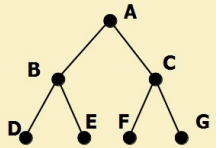
k=0 A
k=1 A B C
C
k=2 A B C
D E C
E C
C

IDS: example 2



k=0 A
k=1 A B C
C
k=2 A B C
D E C
E C
C
F G

IDS: example 2



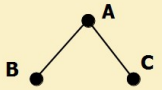
k=0 A
k=1 A B C
C
k=2 A B C
D E C
E C
C
F G
G

IDS: example 2

k=0 A
k=1 A B C
C
k=2 A B C
D E C
E C
C
F G
G

k=3 A

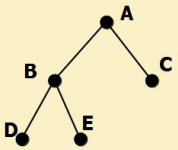
IDS: example 2



k=0 A
k=1 A B C
C
k=2 A B C D E C E C C F G G

k=3 A B C

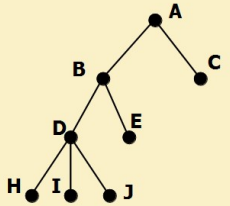
IDS: example 2



k=0 A
k=1 A B C
C
k=2 A B C D E C E C C F G G

k=3 A B C D E C

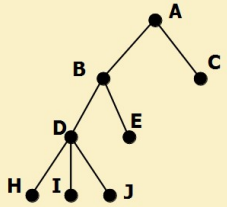
IDS: example 2



k=0 A
k=1 A B C
C
k=2 A B C D E C E C C F G G

k=3 A B C D E C H I J E C

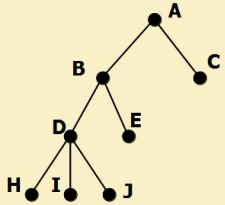
IDS: example 2



k=0 A
k=1 A B C
C
k=2 A B C D E C E C C F G G

k=3 A B C D E C H I J E C I J E C

IDS: example 2



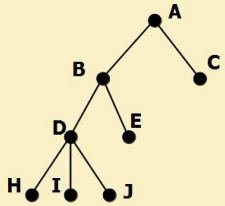
k=0 A

k=1 A
B C
C

k=2 A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C

IDS: example 2



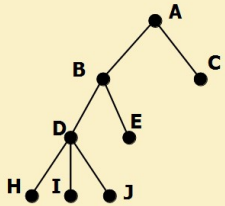
k=0 A

k=1 A
B C
C

k=2 A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C

IDS: example 2



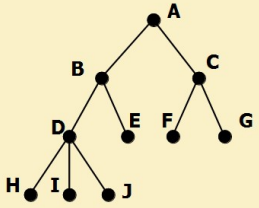
k=0 A

k=1 A
B C
C

k=2 A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C

IDS: example 2



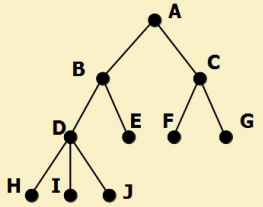
k=0 A

k=1 A
B C
C

k=2 A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C
F G

IDS: example 2



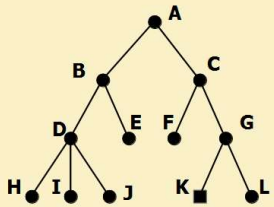
k=0 A

k=1 A
B C

k=2 C
A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C
F G
G

IDS: example 2



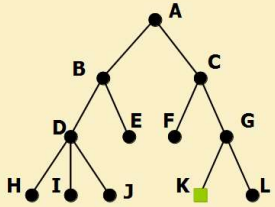
k=0 A

k=1 A
B C

k=2 C
A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C
F G
G
K L

IDS: example 2



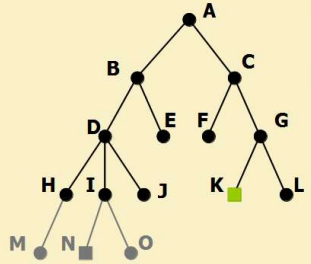
k=0 A

k=1 A
B C

k=2 C
A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C
F G
G
K L

IDS: example 2



k=0 A

k=1 A
B C

k=2 C
A
B C
D E C
E C
C
F G
G

k=3 A
B C
D E C
H I J E C
I J E C
J E C
E C
C
F G
G
K L

Iterative deepening search: redundancy

- The algorithm reinvestigates many states.
- Is this important?

Iterative deepening search: redundancy

- The algorithm reinvestigates many states.
- Is this important? NO.
- In a search tree with almost the same or similar branching at each level, most of the nodes can be found at the deepest level, i. e. it is not so important whether the upper levels are expanded multiple times.
- If d is the depth, then the number of nodes expanded at the level d is only one; at the level of d-1 is two; at the level of d-2 is three etc. Thus, the total number of nodes:
  $(d)b + (d-1)b^2 + \ldots + (1)b^d$
- It also means that the time complexity is $O(b^d)$.

Iterative deepening search: redundancy example

- b= 10; d=5

- The number of nodes within the tree:

1 + 10 + 100 + 1000 + 10 000 + 100 000 = 111 111

- The total number of nodes expanded:

6*1 + 5*10 + 4*100 + 3*1000 + 2*10 000 + 100 000 = 123 456

It means ~11% extra work.

- NB. The more branching there are, the less extra work we have.

IDS vs BFS

- IDS:
  $(d)b + (d-1)b^2 + \ldots + (1)b^d$
- BFS:
  $b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b)$
- NB: BFS generates nodes at the level of d+1, while IDS does not. As a result, IDS may be faster.
- Example:
  - b=10, d=5:
  - IDS: 50 + 400 + 3000 + 20 000 + 100 000 = 123 450
  - BFS: 10 + 100 + 1000 + 10 000 + 100 000 + 999 990 = 1 111 100

Remark

- In general, in case of large search space when the depth of the solution is not known, it is suggested to use the iterative deepening search from the group of noninformed search methods.

Uniform cost search

- Uniform cost search expands the node with the minimal cost.
- Formally: f(n) := g(n)
- Statement:
  - Uniform cost search finds the optimal solution, in case there is an optimal solution.
  - It expands each node at most once.

Uniform cost search #2

- Please note that the length of the walk is not important, only its cost.
- Therefore the algorithm may get into an endless loop, for example expanding a node having zero cost and as a result of further steps the node is reached again.
- Sufficient constraint on the optimality is when the cost of each step is greater than or equal to a small epsilon, in other words the longer the walk is the higher is its cost.

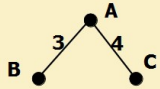
Uniform cost search vs BFS

- Breadth-first search is the same as the uniform cost search if each step has the same cost.

Uniform cost search example

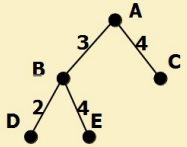
A(0)

Uniform cost search example



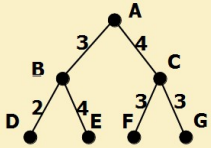
A(0)
B(3) C(4)

Uniform cost search example



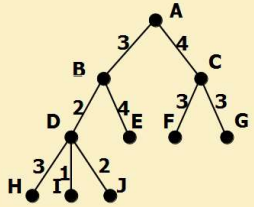
A(0)
B(3) C(4)
C(4) D(5) E(7)

Uniform cost search example



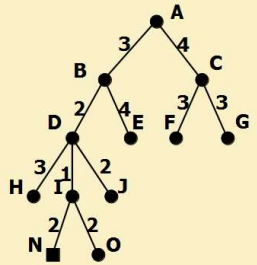
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)

Uniform cost search example



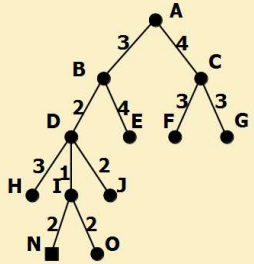
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)

Uniform cost search example



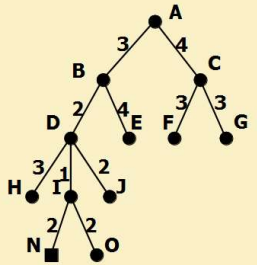
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)

Uniform cost search example



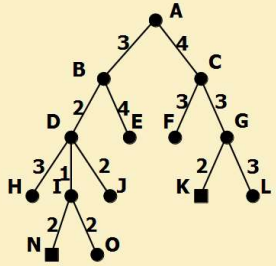
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)

Uniform cost search example



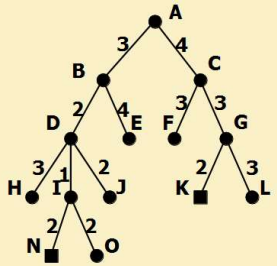
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)
G(7) J(7) H(8) N(8) O(8)

Uniform cost search example



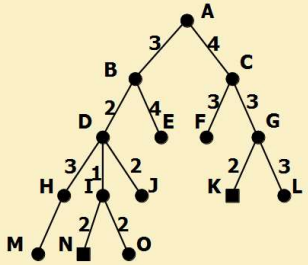
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)
G(7) J(7) H(8) N(8) O(8)
J(7) H(8) N(8) O(8) K(9) L(10)

Uniform cost search example



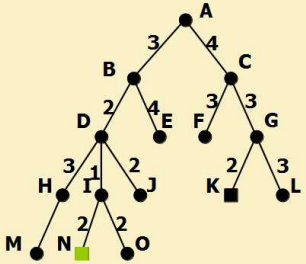
A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)
G(7) J(7) H(8) N(8) O(8)
J(7) H(8) N(8) O(8) K(9) L(10)
H(8) N(8) O(8) K(9) L(10)

Uniform cost search example



A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)
G(7) J(7) H(8) N(8) O(8)
J(7) H(8) N(8) O(8) K(9) L(10)
H(8) N(8) O(8) K(9) L(10)
N(8) O(8) K(9) L(10) M(11)

Uniform cost search example



A(0)
B(3) C(4)
C(4) D(5) E(7)
D(5) E(7) F(7) G(7)
I(6) E(7) F(7) G(7) J(7) H(8)
E(7) F(7) G(7) J(7) H(8) N(8) O(8)
F(7) G(7) J(7) H(8) N(8) O(8)
G(7) J(7) H(8) N(8) O(8)
J(7) H(8) N(8) O(8) K(9) L(10)
H(8) N(8) O(8) K(9) L(10)
N(8) O(8) K(9) L(10) M(11)

Summary of blind search methods

Searches	Completeness	Time complexity	Memory	Optimality
Breadth-first	yes	b^d	b^d	yes
Uniform cost	yes	b^d	b^d	yes
Depth-first	no	b^d	b*d	no
Depth limited	if $d \leq k$	b^k	b*k	no
Iterative deepening	yes	b^d	b*d	yes

b: number of nodes
d: depth
k: constarint on the depth

Informed search

Informed search

- Informed search or heuristic search.
- It can be estimated that one state seems to be better than the other.
- Contains problem specific knowledge.
- Well chosen heuristics may heavily reduce the search space, but they may not guarantee to find the solution with minimal cost.

Informed search #2

- The aim is to find a low cost solution.
- The cost function considers the cost from the start to the actual node and does not focus at the target.
- We know the cost until this point: g(n)
- Let us estimate the cost until the solution, i. e. we introduce a heuristic function:
  h(n): estimated cost of the path from node n to the solution.
- NB: if n is a target node then h(n)=0.

Informed search #3

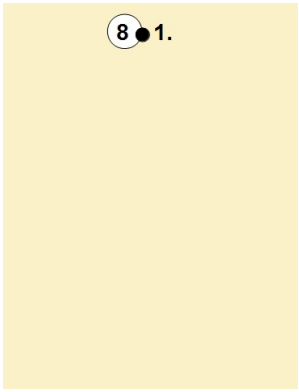
- In case of multiple targets, multiple paths may be good.
- We introduce a heuristic function:
  Let $h^*(n)$ be an estimated minimum cost related to the path from node n to one of the solutions.
  In this case: $h(n) \approx h^*(n)$.

Greedy best-first search

- We expand the node which seems to be closest to the target.
- Formally: f(n) := h(n)
- Statement:
  - With a well chosen heuristic function memory and time consumption can be reduced radically.
  - With a badly chosen heuristic function the procedure may not terminate.

Greedy best-first search example

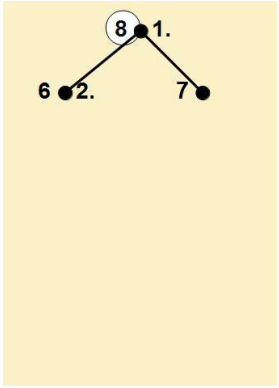
At the node: the estimated cost; and the expanding order.



f(n) = h(n)

Greedy best-first search example

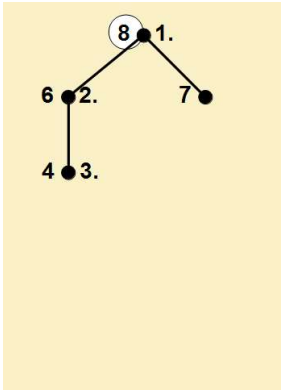
At the node: the estimated cost; and the expanding order.



f(n) = h(n)

Greedy best-first search example

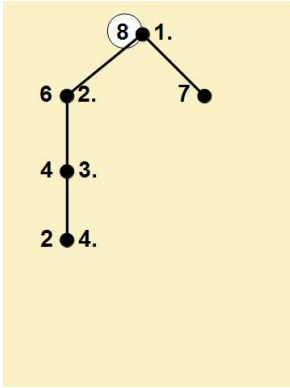
At the node: the estimated cost; and the expanding order.



f(n) = h(n)

Greedy best-first search example

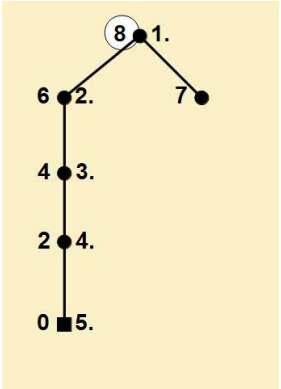
At the node: the estimated cost; and the expanding order.



f(n) = h(n)

Greedy best-first search example

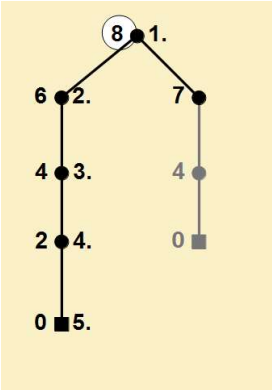
At the node: the estimated cost; and the expanding order.



f(n) = h(n)

Greedy best-first search example

At the node: the estimated cost; and the expanding order.



f(n) = h(n)

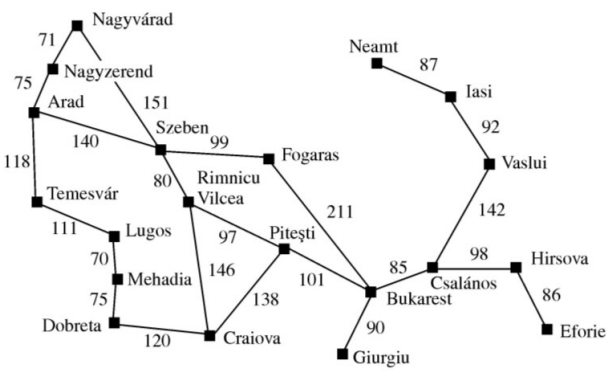
NB. should the cost of each edge be equal, the found solution is not optimal.

Greedy best-first search example 2

- Let us consider a map and the route to Bucharest, Romania.
- Let us consider the straight line distances between Bucharest and the various cities (h_{sld}). Please note that this information cannot be calculated from the problem formulation. Moreover, we need some experience to understand that this information and the covered distance are correlated with each other.

Arad	366	Mehadia	241
Bukarest	0	Neamt	234
Craiova	160	Nagyvárad	380
Dobreta	242	Piteşti	100
Eforie	161	Rimniu Vilcea	193
Fogaras	176	Nagyszeben	253
Giurgiu	77	Temesvár	329
Hirsova	151	Csalános	80
Iasi	226	Vaslui	199
Lugos	244	Nagyzerénd	374

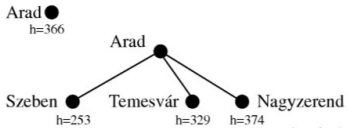
Greedy best-first search example 2



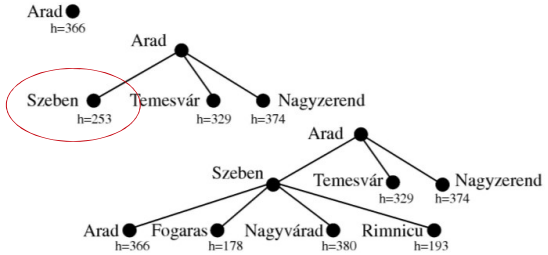
Straight line distances from Bucharest:

Arad	366
Bukarest	0
Craiova	160
Csalános	80
Dobreta	242
Eforie	161
Fogaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugos	244
Mehadia	241
Neamt	234
Nagyvárad	380
Pitești	98
Rimnicu Vilcea	193
Szeben	253
Temesvár	329
Vaslui	199
Nagyzerend	374

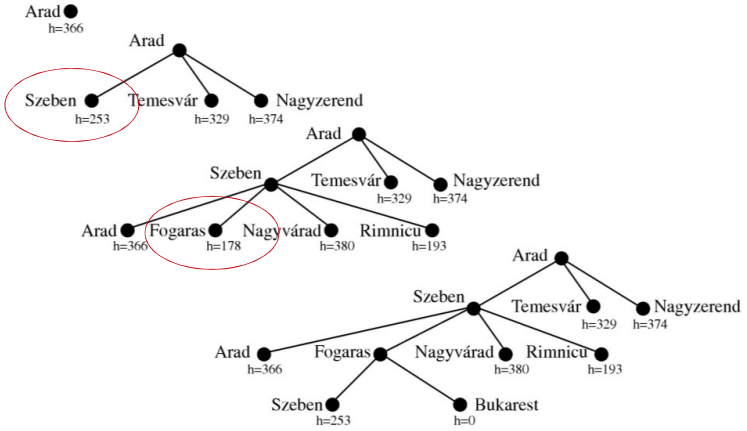
Greedy best-first search example 2



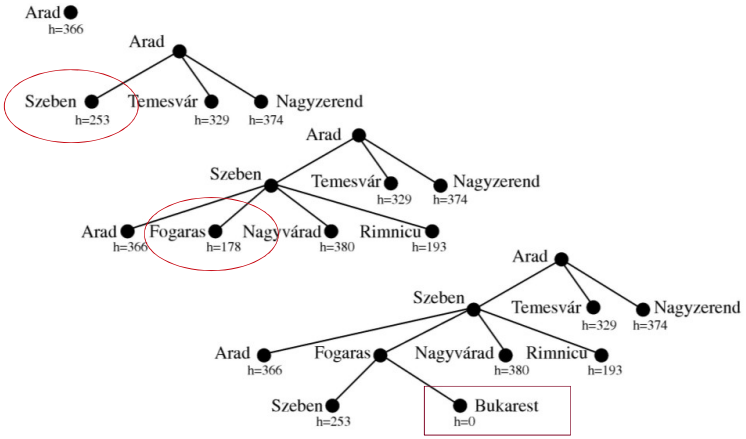
Greedy best-first search example 2



Greedy best-first search example 2



Greedy best-first search example 2

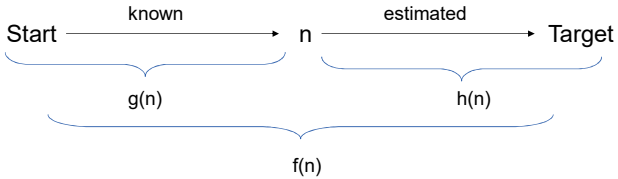


Greedy best-first search properties

- It is not complete.
- It can get stuck in loops.
- Time complexity: $O(b^m)$ where m is the maximal depth of the search space.
- A good heuristic may give dramatic improvement.
- Optimality of the solution is not guaranteed.

A Algorithm

- Idea: avoid expanding paths that are already expensive and combine an even search with a greedy first search.
- Evaluation function f(n) = g(n) + h(n), h(n)≥0
- g(n) = actual cost so far to reach n
- h(n) = estimated cost from n to goal
- f(n) = estimated total cost of path through node n to the target



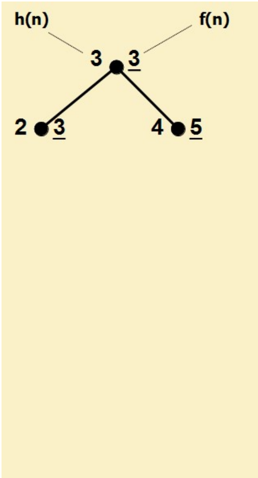
A Algorithm

- Statement: A Algorithm always finds a solution if there are any.

A Algorithm example

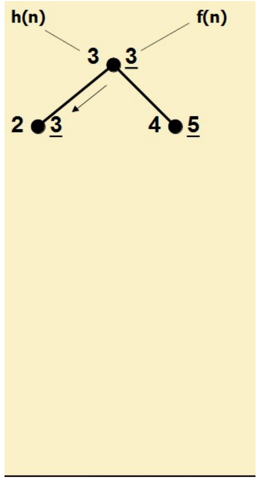
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



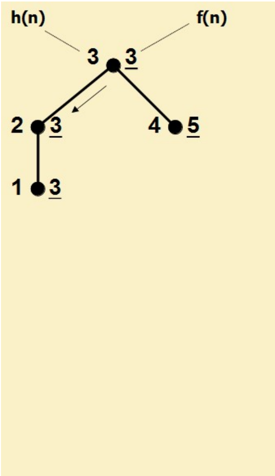
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



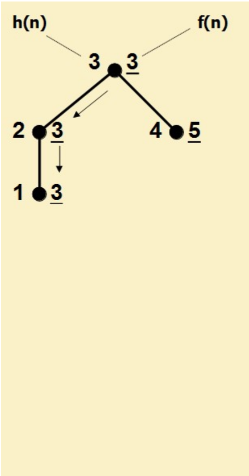
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



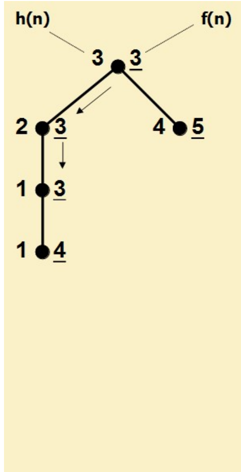
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



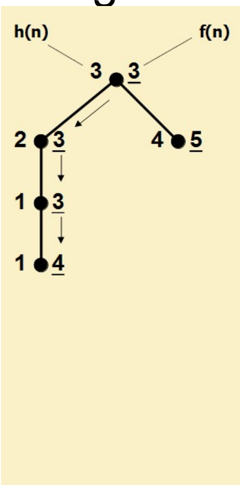
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



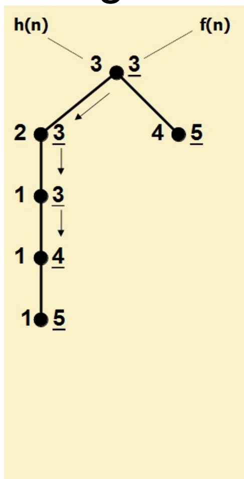
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



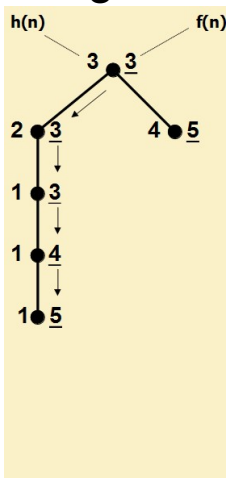
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



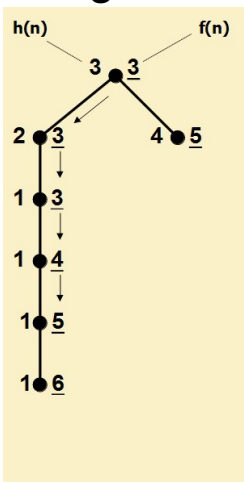
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



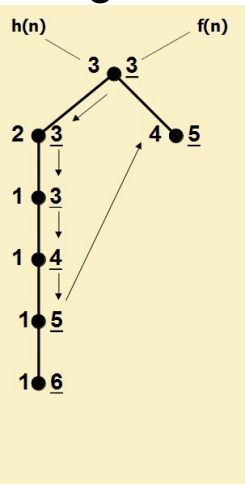
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



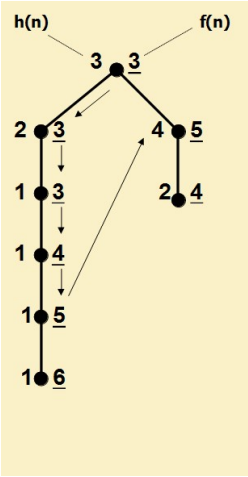
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



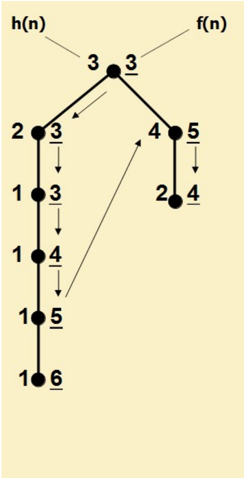
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



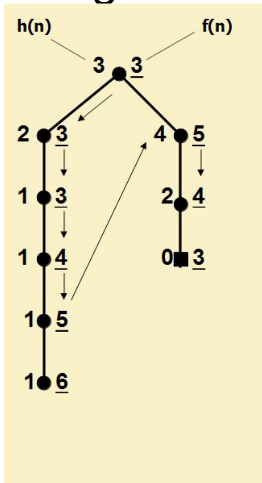
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



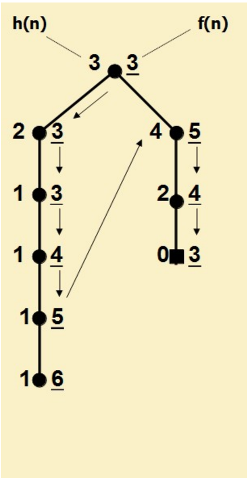
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



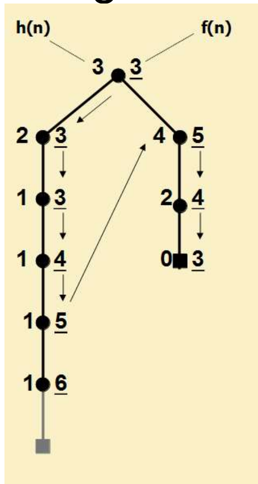
f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

A Algorithm example



f(n) = g(n) + h(n)
h(n) ≥ 0
Cost of the edges are equal.

NB. It might not be optimal.

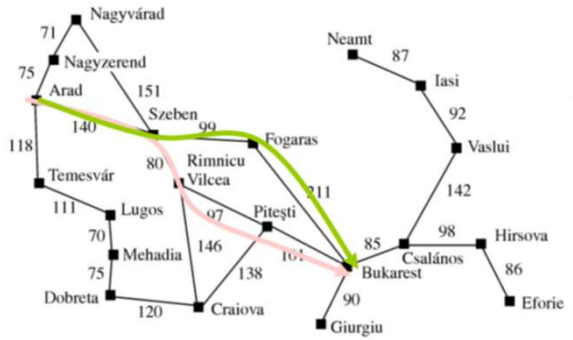
A* Algorithm

- A Algorithm with its heuristic function which underestimates h*(n) at each node.
- Formally: A heuristic h(n) is admissible if for every node n, h(n) ≤ h*(n), where h*(n) is the true cost to reach the target from node n.
- An admissible heuristic never overestimates the cost to reach the goal, i. e. it is optimistic.

  f(n) := g(n) + h(n) for every n: 0 ≤ h(n) ≤ h*(n)

- Statement: The A* Algorithm always finds an optimal solution, if there is a solution.

A* Algorithm example



Straight line distances from Bucharest:

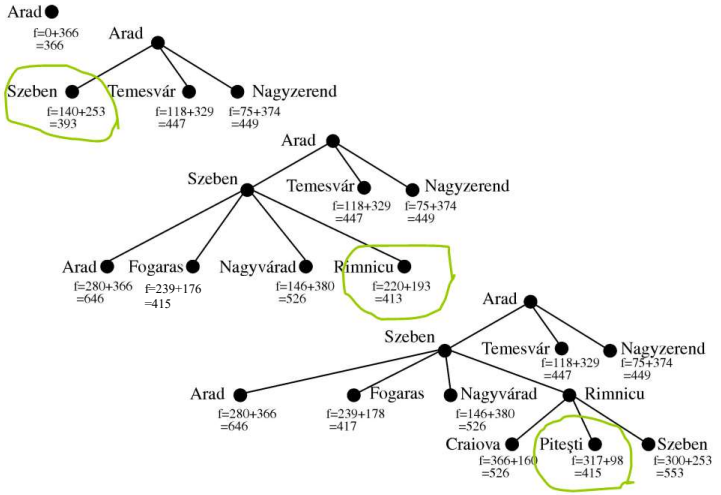
Arad	366
Bukarest	0
Craiova	160
Csalános	80
Dobreta	242
Eforie	161
Fogaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugos	244
Mehadia	241
Neamt	234
Nagyvárad	380
Pitești	98
Rimnicu Vilcea	193
Szeben	253
Temesvár	329
Vaslui	199
Nagyzerend	374

Please note that Arad – Sibiu – Bucharest is 32kms longer than the route via Rimnicu – Pitesti.

A* Algorithm example

- Let us recall the map and the route to Bucharest, Romania.
- Let us consider the straight line distances between Bucharest and the various cities (h_{sld}).
- Let us recall that the greedy best-first search algorithm was not optimal.

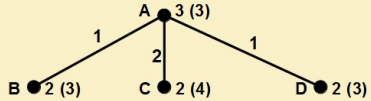
A* Algorithm example



A* Algorithm example 2

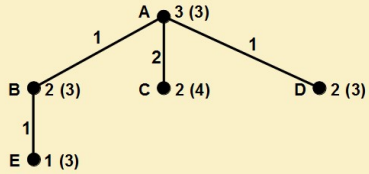
1. L = {A(3)}

A* Algorithm example 2



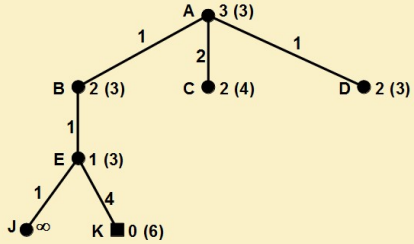
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}

A* Algorithm example 2



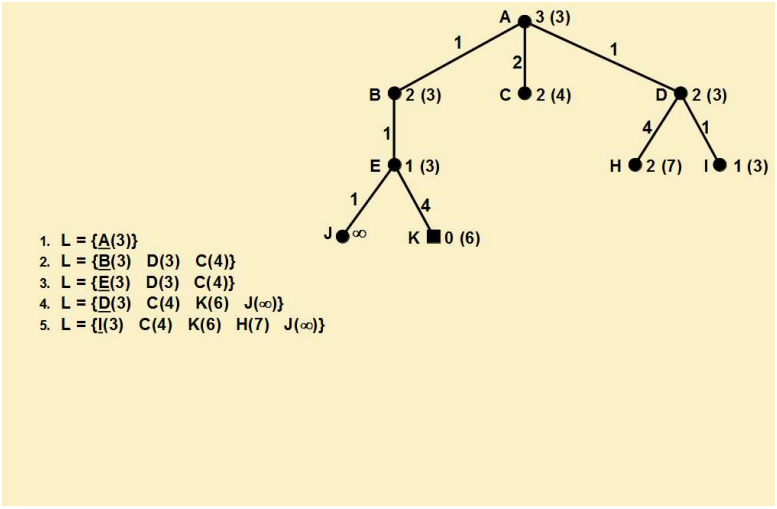
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}

A* Algorithm example 2



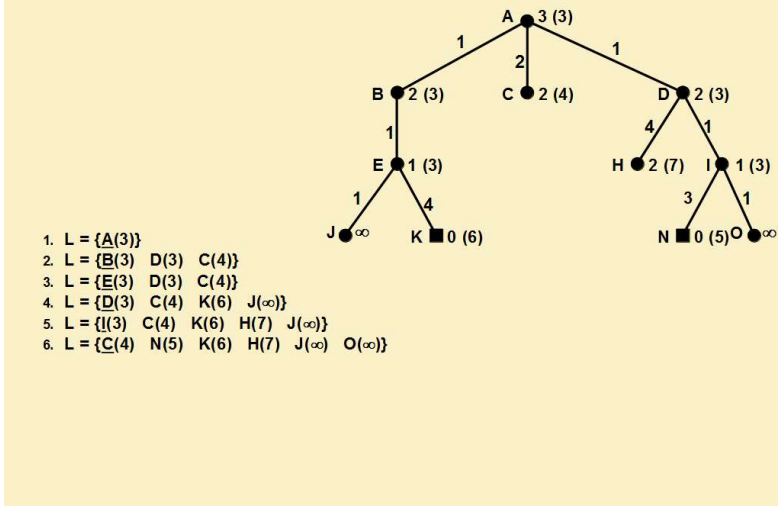
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}
4. L = {D(3) C(4) K(6) J(∞)}

A* Algorithm example 2



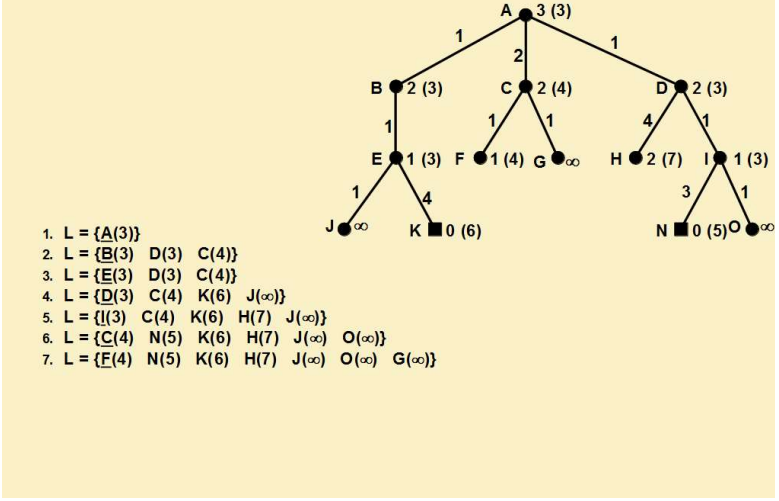
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}
4. L = {D(3) C(4) K(6) J(∞)}
5. L = {I(3) C(4) K(6) H(7) J(∞)}

A* Algorithm example 2



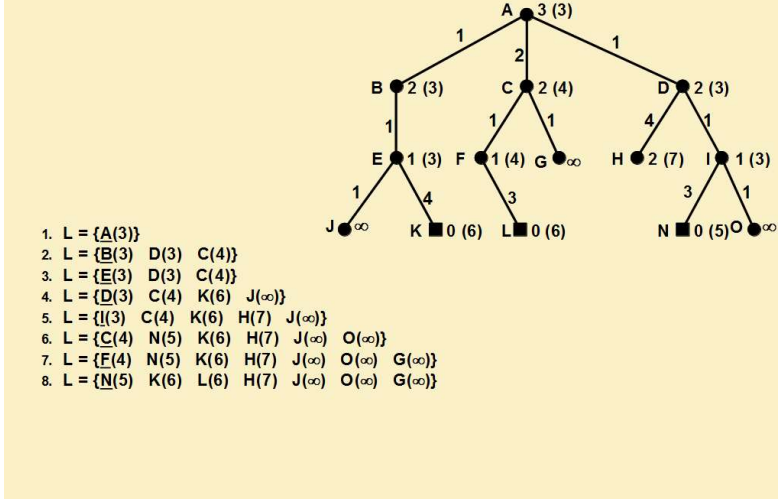
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}
4. L = {D(3) C(4) K(6) J(∞)}
5. L = {I(3) C(4) K(6) H(7) J(∞)}
6. L = {C(4) N(5) K(6) H(7) J(∞) O(∞)}

A* Algorithm example 2



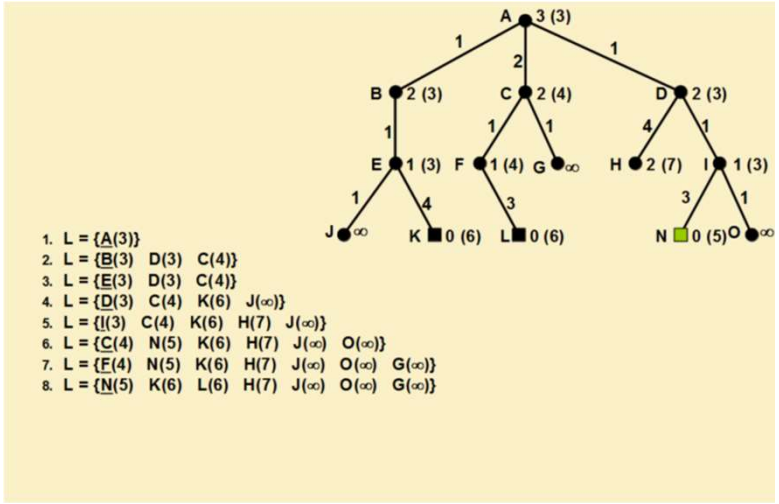
1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}
4. L = {D(3) C(4) K(6) J(∞)}
5. L = {I(3) C(4) K(6) H(7) J(∞)}
6. L = {C(4) N(5) K(6) H(7) J(∞) O(∞)}
7. L = {F(4) N(5) K(6) H(7) J(∞) O(∞) G(∞)}

A* Algorithm example 2

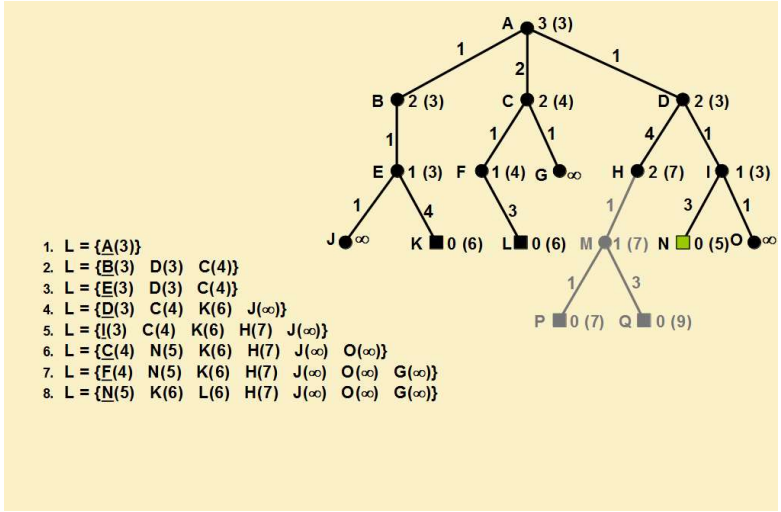


1. L = {A(3)}
2. L = {B(3) D(3) C(4)}
3. L = {E(3) D(3) C(4)}
4. L = {D(3) C(4) K(6) J(∞)}
5. L = {I(3) C(4) K(6) H(7) J(∞)}
6. L = {C(4) N(5) K(6) H(7) J(∞) O(∞)}
7. L = {F(4) N(5) K(6) H(7) J(∞) O(∞) G(∞)}
8. L = {N(5) K(6) L(6) H(7) J(∞) O(∞) G(∞)}

A* Algorithm example 2



A* Algorithm example 2



A* Algorithm

- If h ≡ h* → nodes on the path are expanded only
- If h ≡ 0 → even search
- If h ≡ 0 and the cost of the edges are equal
  → breadth-first search

- f(n) is ideal evalutaion function, since at the target:
  h(n) = 0 → f(n) = g(n) → it gives the cost of the solution

Properties of A*

- A* algorithm is complete.
- The time complexity is exponential.
- It keeps all nodes in memory.
- It finds the optimal solution.

A vs A*

- When should we use A and when A*?

?

Dominance

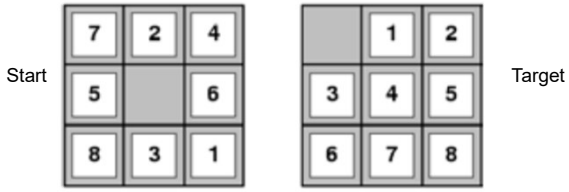
- If $h_2(n) \geq h_1(n)$ for all n (both admissible)

then h_2 dominates h_1

h_2 is better for search

Dominance example

- 8-puzzle game
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance, i. e. number of blocks from the desired location of each tile



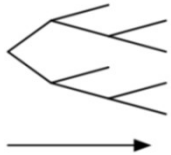
$h_1(S) = 8$ $h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

Directions

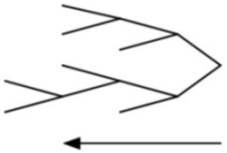
Backward search

- In case it is easier to approach the problem space from the target point of view, i. e. it shows far less alternatives then from the start, then it may be useful to adopt a backward search.
- For example: answers for the test questions vs the full text book.
- Forward search: Start → there is a target somewhere in the large state space.
- However, it may often be that on the way a lot of nodes are unnecessarily generated.

Backward search

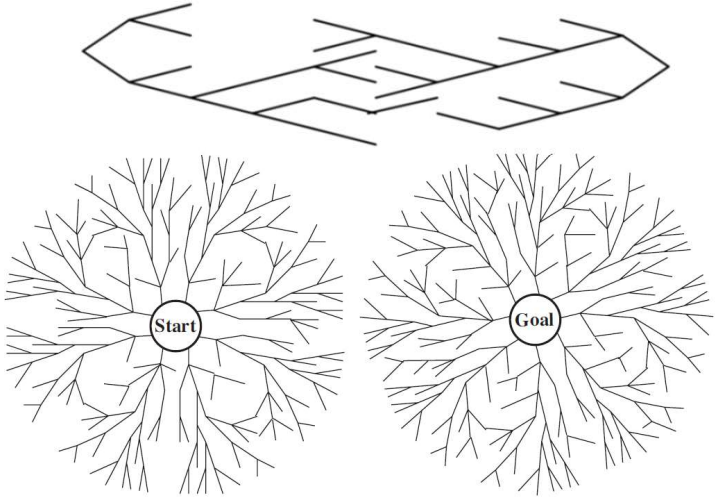


Forward search.



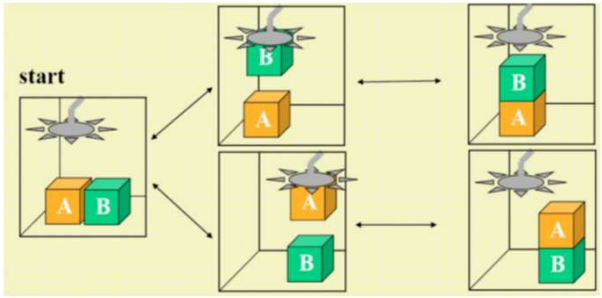
Backward search.

Bidirectional search



Example

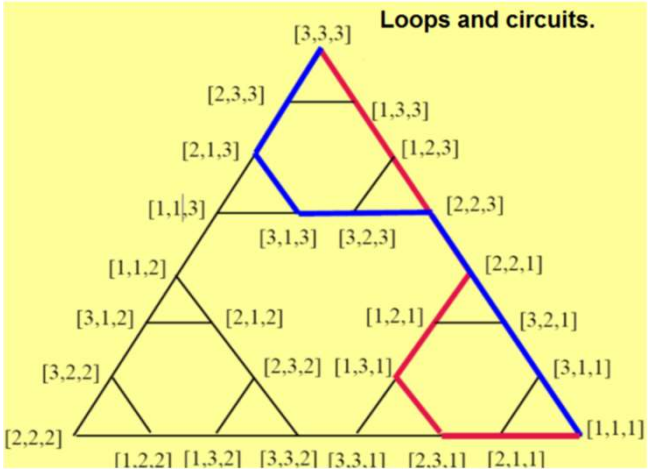
- Two boxes on each other is the target → what is necessary as a predecessor?



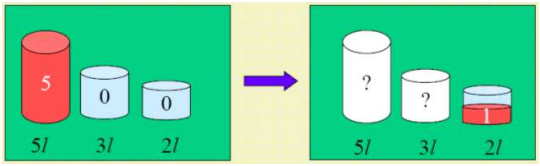
Backward search

- The solution path is important, not its implementation.

Bidirectional search



Backward search: problems



- It is difficult to find the target state. From the initial state we may not reach all target states and backwards from the target state we may not reach all initial states. For example: (2,2,1)
- When a path is found by a backwards search, it may not be suitable for the original problem. For example: (4,0,1) → (5,0,0)

Backward search: conditions

- The operations have to be invertable.

At least those operations have to be invertable that are used by the backward search.
It has to be clear how we can step to the next state.

- We have to choose a specific target state.

Previously a condition was enough for the search.

Basic introduction to graph theory

Suggested readings

•The current graph theory presentation is based on the works of professor Ruohonen:

•http://math.tut.fi/~ruohonen/GT_English.pdf

•Further readings for example:

•http://cs.bme.hu/fcs/graphtheory.pdf

•Tutte, W.T. (2001), Graph Theory, Cambridge University Press, p. 30, ISBN 978-0-521-79489-3.

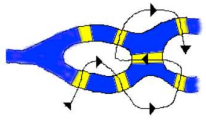
•Chartrand, Gary (1985), Introductory Graph Theory, Dover, ISBN 0-486-24775-9.

•Gibbons, Alan (1985), Algorithmic Graph Theory, Cambridge University Press.

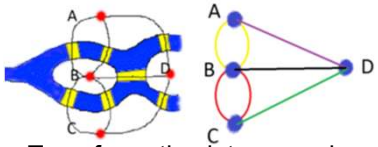
Euler: graph theory

•Leonhard Euler
developed graph theory to solve the problem of seven bridges at Königsberg.

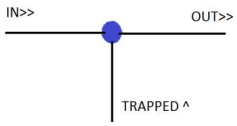
7 bridges of Königsberg



A try



Transformation into a graph



A vertex needs two or even number of edges to get in and out.
If a vertex has odd number of edges, it should be an end point or the person is trapped.
Here we have four odd vertices, hence there is no Euler path.

Fundamental Concepts

Sets

•A set is a well defined collection of distinct objects, called elements.

•Membership: A = {1,2,3,4}; 4 ∈ A.

•If every member of set A is also a member of set B, then A is said to be a subset of B: A ⊆ B; and B is a superset of A (B contains A).

•NB: ∅ ⊆ A. A ⊆ A.

•If A is a subset of B, but not equal to B, then A is called a proper subset of B, written A ⊊ B.

•A partition of a set S is a set of nonempty subsets of S such that every element x in S is in exactly one of these subsets.

Sets #2

•The power set of a set S is the set of all subsets of S; denoted by: P(S).
Example: S={1, 2, 3}

P(S) = {{1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}, {2}, {3}, ∅}

•The power set of a finite set with n elements has 2^n elements.

•For a finite set X, |X| denotes its size (cardinality, the number of its elements).

•NB. The empty set has no members and zero cardinality.

Sets #3

•The union of A and B, denoted by A ∪ B, is the set of all members of either A or B.

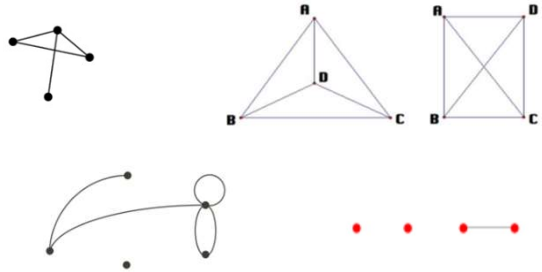
•The intersection of A and B, denoted by A ∩ B, is the set of all members of both A and B. If A ∩ B = ∅, then A and B are said to be disjoint sets.

•For two sets X and Y
X × Y = {(x, y) | x ∈ X, y ∈ Y} is their Cartesian product; i.e. the set of all ordered pairs.

Graph

•A graph is formed by vertices (or nodes or simple points) and edges (or arcs or lines) connecting some of the vertices.

•Often, we label the vertices with letters (for example: a, b, c, ... or v1, v2, ...) or numbers 1, 2, ...



Graph #2

•Let sets V and E be given, where V is the set of vertices and E is the set of edges, formed by pairs of vertices.

•A graph G is a pair of sets (V, E).

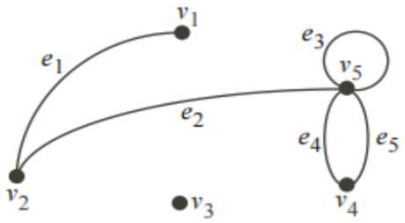
•If V and E are finite sets then we talk about finite graphs.

•NB: Erdős and Rényi introduced random graphs.
Let a set of vertices be given. Let one select 2 vertices and draw an edge between them only when 6 was thrown with the dice, otherwise do not draw the edge. Then select again 2 vertices from the set.

Example

•Let us denote
$V = \{v_1, \ldots, v_5\}$ for the vertices and
$E = \{e_1, \ldots, e_5\} = \{(v_1, v_2),(v_2, v_5),(v_5, v_5),(v_5, v_4),(v_5, v_4)\}$ for the edges.

•Please note that the two edges (u, v) and (v, u) are the same, i.e. the pair is not ordered.



Terminology

•The two vertices u and v are end vertices of the edge (u, v).

•Edges that have the same end vertices are parallel.

•An edge of the form (v, v) is a loop.

•A graph is simple if it has no parallel edges or loops.

•A graph with no edges (i.e. E is empty) is empty.

•A graph with no vertices (i.e. V and E are empty) is a null graph.

•A graph with only one vertex is trivial.

Terminology #2

•Edges are adjacent if they share a common end vertex.

•Two vertices u and v are adjacent if they are connected by an edge, in other words, (u, v) is an edge.

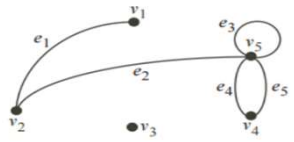
•The degree of the vertex v, written as d(v), is the number of edges with v as an end vertex. By convention, we count a loop twice and parallel edges contribute separately.

•A pendant vertex is a vertex whose degree is 1.

•An edge that has a pendant vertex as an end vertex is a pendant edge.

•An isolated vertex is a vertex whose degree is 0.

Example



•Which are end vertices of e_5?

•Are there parallel edges?

•Is there a loop?

•Is this graph simple?

•Please name adjecent edges.

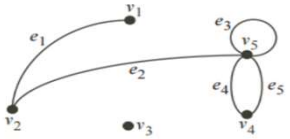
•Please name adjecent vertices.

•Is there a pendant vertex?

•Is there a pendant edge?

•Is there an isolated vertex?

Example



•v_4 and v_5 are end vertices of e_5.

•e_4 and e_5 are parallel.

•e_3 is a loop.

•The graph is not simple.

•e_1 and e_2 are adjecent.

•v_1 and v_2 are adjecent.

•The degree of v_1 is 1 so it is a pendant vertex.

•e_1 is a pendant edge.

•The degree of v_5 is 5. The degree of v_4 is 2.

•The degree of v_3 is 0 so it is an isolated vertex.

Minimum and maximum degree

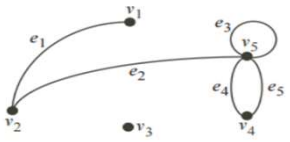
•Let us consider a graph G = (V, E).

•The minimum degree of the vertices in a graph G is denoted δ(G)

•δ(G) = 0 if there is an isolated vertex in G.

•Similarly, we write $\Delta(G)$ as the maximum degree of vertices in G.

Example



•The minimum degree of G is:

•$\delta(G) = 0$ and

•The maximum degree of G is:

•$\Delta(G) = 5$.

Theorem

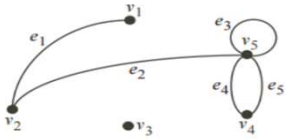
•Since every edge has two end vertices, then

•Theorem: The graph G = (V, E), where V = $\{v_1, \ldots, v_n\}$ and E = $\{e_1, \ldots, e_m\}$, satisfies

$$\sum_{i=1}^{n} d(v_i) = 2m.$$

•Corollary: Every graph has an even number of vertices of odd degree.

Example



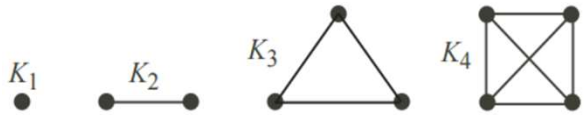
•The sum the sum of the degrees is
$1 + 2 + 0 + 2 + 5 = 10 = 2 \cdot 5$.
There are two vertices of odd degree, namely v_1 and v_5.

Complete graphs

•A graph is simple if it has no parallel edges or loops.

•A simple graph that contains every possible edge between all the vertices is called a complete graph.

•A complete graph with n vertices is denoted as K_n.



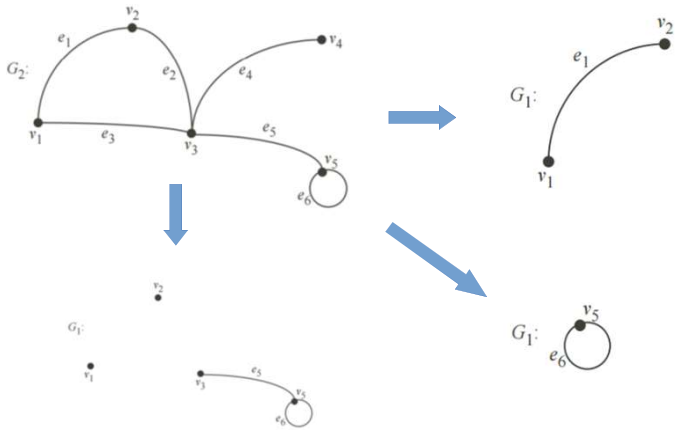
Subgraphs

•The graph $G_1 = (V_1, E_1)$ is a subgraph of $G_2 = (V_2, E_2)$ if

$V_1 \subseteq V_2$ and

Every edge of G_1 is also an edge of G_2.

Subgraphs: example



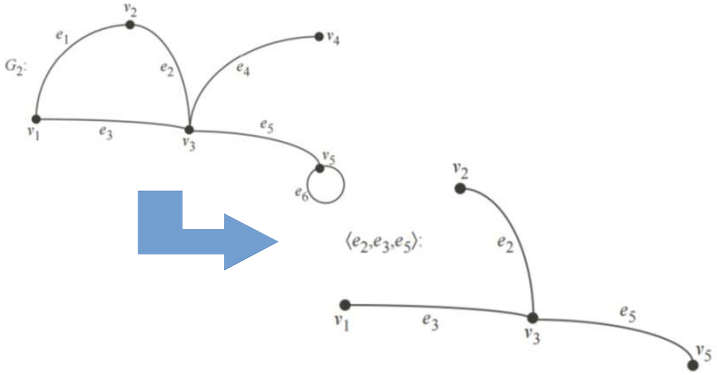
Subgraphs #2

•The subgraph of G = (V, E) induced by the edge set $E_1 \subseteq E$ is

$G_1 = (V_1, E_1) =_{def.} \langle E_1 \rangle$

where V_1 consists of every end vertex of the edges in E_1.

Subgraphs: example #2

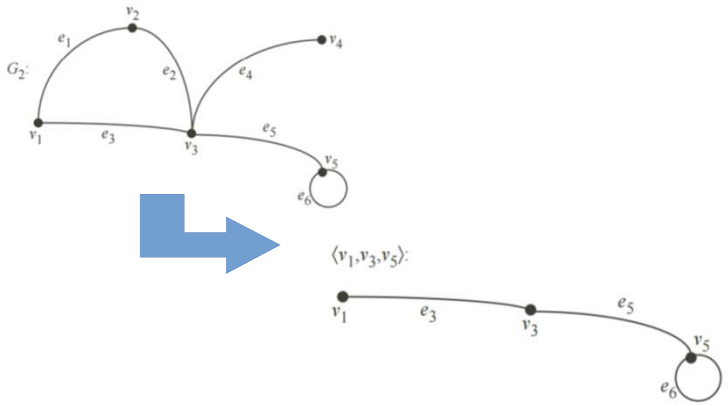


From the original graph G, the edges e_2, e_3 and e_5 induce the above subgraph.

Subgraphs #3

•The subgraph of G = (V, E) induced by the vertex set $V_1 \subseteq V$ is
$G_1 = (V_1, E_1) =_{def.} \langle V_1 \rangle$
where E_1 consists of every edge between the vertices in V_1.

Subgraphs: example #3



From the original graph G, the vertices v_1, v_3 and v_5 induce the above subgraph.

Subgraphs: clique

•The complete subgraph of G is called a clique of G.

Graph Theory

Walks, paths, circuits, cuts and labeling

Walks

•A walk in the graph G = (V, E) is a finite sequence of the form

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k}$$

•which consists of alternating vertices and edges of G.

•The walk starts at a vertex.

•Vertices $v_{i_{t-1}}$ and v_{i_t} are end vertices of e_{j_t} (t = 1, ... , k).

•v_{i_0} is the initial vertex and v_{i_k} is the terminal vertex.

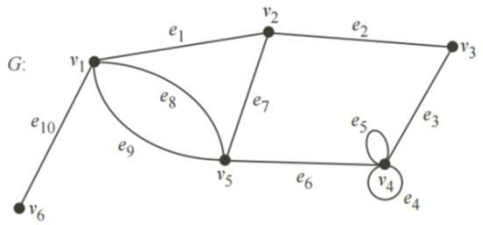
•k is the length of the walk.

•A zero length walk is a single vertex.

•It is allowed to visit a vertex or go through an edge more than once.

•A walk is open if $v_{i_0} \neq v_{i_k}$. Otherwise it is closed.

Walk: example



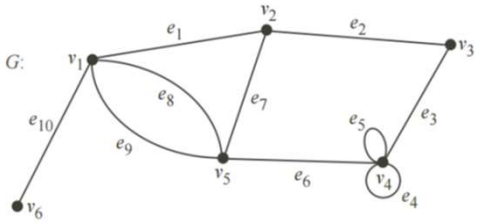
Open walk: v2, e7, v5, e8, v1, e8, v5, e6, v4, e5, v4, e5, v4.
Closed walk: v4, e5, v4, e3, v3, e2, v2, e7, v5, e6, v4.

Trails

•A walk is a trail if any edge is traversed at most once.

•Then, the number of times that the vertex pair u, v can appear as consecutive vertices in a trail is at most the number of parallel edges connecting u and v.

Trail: example



The walk in the graph
v1, e8, v5, e9, v1, e1, v2, e7, v5, e6, v4, e5, v4, e4, v4
is a trail.

Paths

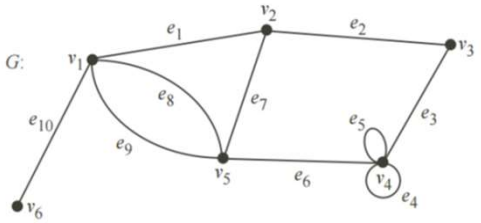
•A trail is a path if any vertex is visited at most once except possibly the initial and terminal vertices when they are the same.

•A closed path is a circuit.

•For simplicity, we will assume in the future that a circuit is not empty, i.e. its length ≥ 1.

•We identify the paths and circuits with the subgraphs induced by their edges.

Path: example



The walk in the graph
v2, e7, v5, e6, v4, e3, v3
is a path;
and the walk
v2, e7, v5, e6, v4, e3, v3, e2, v2
Is a circuit.

Connected graphs

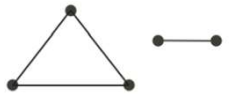
•The walk starting at u and ending at v is called an u–v walk.

•u and v are connected if there is a u–v walk in the graph (then there is also a u–v path).

•If u and v are connected and v and w are connected, then u and w are also connected, i.e. if there is a u–v walk and a v–w walk, then there is also a u–w walk.

•A graph is connected if all the vertices are connected to each other.

•A trivial graph (a graph with only one vertex) is connected by convention.



This graph is not connected.

Components

The subgraph G_1 (not a null graph) of the graph G is a component of G if

•G_1 is connected and

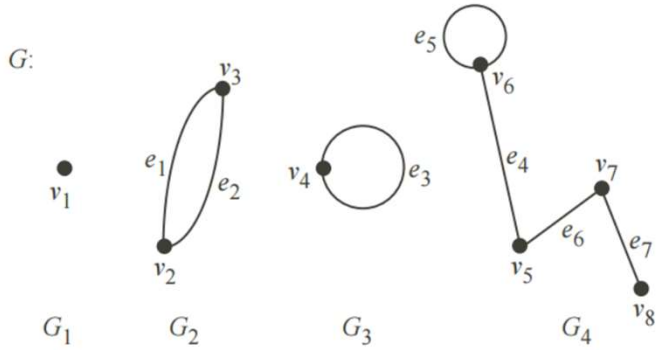
•Either G_1 is trivial (one single isolated vertex of G) or G_1 is not trivial and G_1 is the subgraph induced by those edges of G that have one end vertex in G_1

Components #2

•Theorem: If the graph G has a vertex v that is connected to a vertex of the component G_1 of G, then v is also a vertex of G_1.

•Theorem: Every vertex of G belongs to exactly one component of G. Similarly, every edge of G belongs to exactly one component of G.

Components: example



Components #3

An algorithm that divides a graph into distinct components is the following:

•Let us choose a vertex v in G.
Let us repeat the following as many times as possible starting with $V_1 = \{v\}$:

•If v′ is a vertex of G such that $v' \notin V_1$ and v′ is connected to some vertex of V_1, then $V_1 \leftarrow V_1 \cup \{v'\}$.

•Since there is a finite number of vertices in G, the process stops eventually.

•The last V_1 induces a subgraph G_1 of G that is the component of G containing v.

•Every isolated vertex forms its own component.

Rank

•A graph G with n vertices, m edges and k components has the rank

$$\rho(G) = n - k$$

•The nullity of the graph is

$$\mu(G) = m - n + k$$

We see that ρ(G)≥0 and ρ(G)+μ(G)=m. In addition, μ(G) ≥0.

Theorems

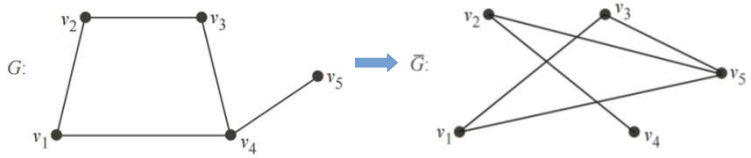
•Theorem. If G is a connected graph and k ≥ 2 is the maximum path length, then any two paths in G with length k share at least one common vertex.

•NB. A graph is circuitless if it does not have any circuit in it.

•Theorem. A graph is circuitless exactly when there are no loops and there is at most one path between any two given vertices.

Complement of a simple graph

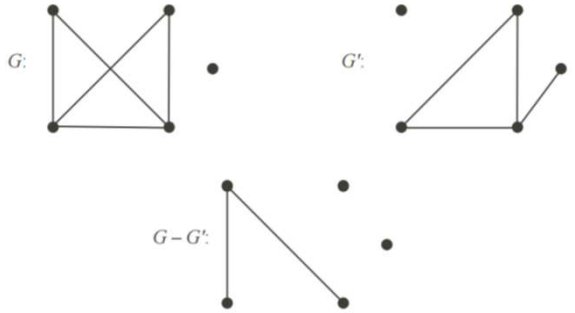
•The complement of the simple graph G = (V, E) is the simple graph $\bar{G}$ = (V, $\bar{E}$), where the edges in $\bar{E}$ are exactly the edges not in G.



Difference graphs

•If the graphs G = (V, E) and G′ = (V′, E′) are simple and V′ ⊆ V , then the difference graph is G − G′ = (V, E″), where E″ contains those edges from G that are not in G′ (simple graph).

•Example:



Graph operations

•Let two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be given.

•The union is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ (simple graph).

•The intersection is $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$ (simple graph).

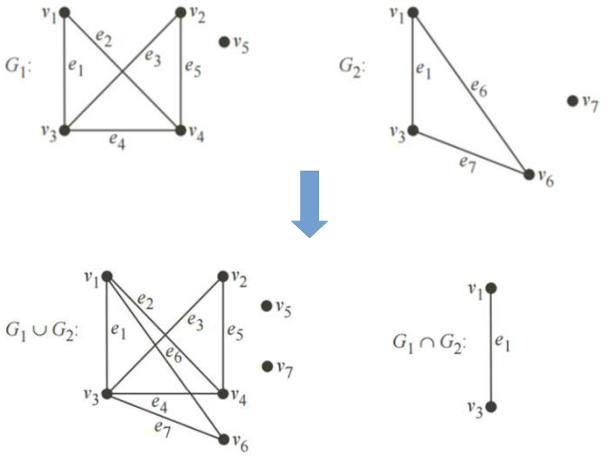
•The ring sum $G_1 \oplus G_2$ is the subgraph of $G_1 \cup G_2$ induced by the edge set $E_1 \oplus E_2$ (simple graph).

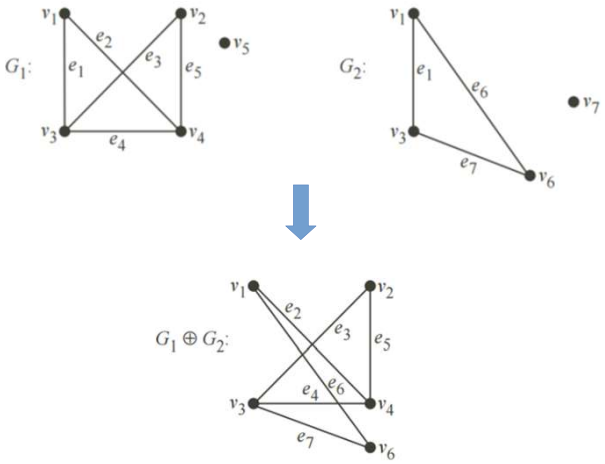
•NB. The set operation $\oplus$ is the symmetric difference, i. e.

$E_1 \oplus E_2 = (E_1 - E_2) \cup (E_2 - E_1)$.

•Since the ring sum is a subgraph induced by an edge set, there are no isolated vertices.

•All three operations are commutative and associative.





Graph operations: multiplicity

•Obviously, union, intersection and the ring sum operation can be defined on more general graphs than simple graphs.

•In this case multiplicity has to be taken into consideration.

•We assume zero multiplicity for the absence of an edge.

$\cup$: The multiplicity of an edge in $G_1 \cup G_2$ is the larger of its multiplicities in G_1 and G_2.

$\cap$: The multiplicity of an edge in $G_1 \cap G_2$ is the smaller of its multiplicities in G_1 and G_2.

$\oplus$: The multiplicity of an edge in $G_1 \oplus G_2$ is $|m_1 - m_2|$, where m_1 is its multiplicity in G_1 and m_2 is its multiplicity in G_2.

Removal of a vertex

•If v is a vertex of the graph G = (V, E), then G − v is the subgraph of G induced by the vertex set V − {v}.

Removal of an edge

•If e is an edge of the graph G = (V, E), then G − e is graph (V, E′), where E′ is obtained by removing e from E.

Short-circuit two vertices

•If u and v are two distinct vertices of the graph G = (V, E), then we can short-circuit the two vertices u and v and obtain the graph (V′, E′), where V′=(V-{u,v}) U {w} (w∉V is the „new” vertex), E′=(E-{(v’,u),(v’,v)│v’∈V}) U {(v’,w)│(v’,u) ∈E or (v’,v) ∈E}) U {(w,w)│(u,u) ∈E or (v,v) ∈E})

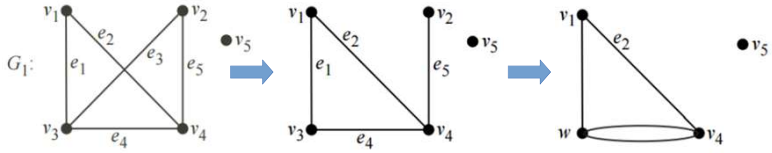
Short-circuit v_3 and v_4 in graph G_1.

Contracting an edge

•In the graph G = (V, E), contracting the edge e = (u, v) (not a loop) means the operation in which we first remove e and then short-circuit u and v.

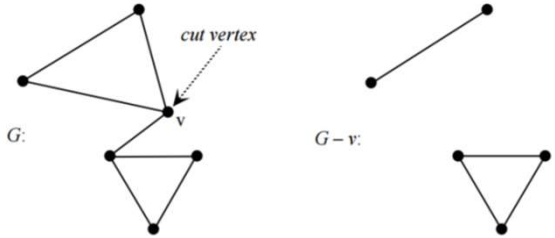
•Contracting a loop simply removes that loop.

•Example: We contract the edge e_3 in G_1 :
first removing e_3 and then short-circuiting v_2 and v_3.



Cuts

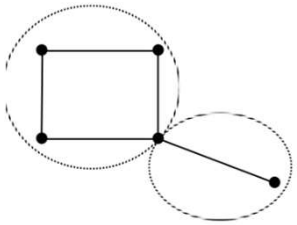
•A vertex v of a graph G is a cut vertex (or an articulation vertex) of G if the graph G − v consists of a greater number of components than G.



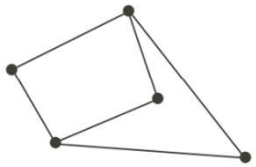
Separable graphs

•A graph is separable if it is not connected or if there exists at least one cut vertex in the graph.

•Otherwise, the graph is nonseparable.



Separable graph.



Nonseparable graph.

Theorems

•Theorem. The vertex v is a cut vertex of the connected graph G if and only if there exist two vertices u and w in the graph G such that

•v ≠ u, v ≠ w and u ≠ w, but

•v is on every u–w path.

•Theorem. A nontrivial simple graph has at least two vertices which are not cut vertices.

Cut sets

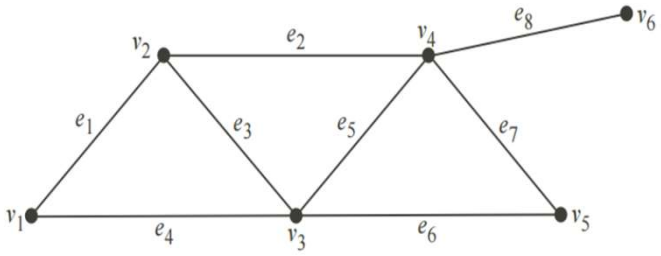
•A cut set of the connected graph G = (V, E) is an edge set F ⊆ E such that

•G − F (remove the edges of F one by one) is not connected, and

•G − H is connected whenever H ⊂ F.

•Theorem. If F is a cut set of the connected graph G, then G − F has two components.

Cut sets: example



$\{e_1, e_4\}$, $\{e_6, e_7\}$, $\{e_1, e_2, e_3\}$, $\{e_8\}$, $\{e_3, e_4, e_5, e_6\}$, $\{e_2, e_5, e_7\}$, $\{e_2, e_5, e_6\}$, and $\{e_2, e_3, e_4\}$ are cut sets. Are there other cut sets?

Cut or partition

•Let us consider a graph G = (V, E). A partition or a cut of G is where a pair of subsets V_1 and V_2 of V satisfy the followings:

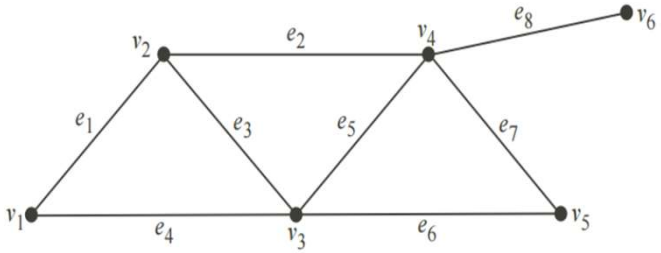
$V = V_1 \cup V_2$, $V_1 \cap V_2 =$

and it is denoted by: $\langle V_1, V_2 \rangle$.

•Usually $\langle V_1, V_2 \rangle$ and $\langle V_2, V_1 \rangle$ are considered to be the same.

•Please note that we can also think of a cut as an edge set: cut $\langle V_1, V_2 \rangle$ = { those edges with one end vertex in V_1 and the other end vertex in V_2}; however the sets here are not uniquely defined, thus it cannot be used as a definition.

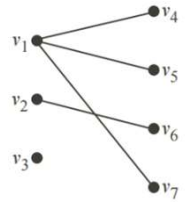
Cut or partition: example



$\langle \{v_1, v_2, v_3\}, \{v_4, v_5, v_6\} \rangle$ *is a cut.*

Bipartite graphs

•If there exists a cut $\langle V_1, V_2 \rangle$ for the graph G = (V, E) so that E = $\langle V_1, V_2 \rangle$, i.e. the cut (considered as an edge set) includes every edge, then the graph G is bipartite.



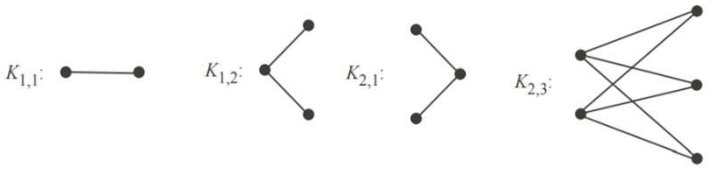
A bipartite graph example.
$V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{v_4, v_5, v_6, v_7\}$

Complete bipartite graphs

•A simple bipartite graph is called a complete bipartite graph if we can not possibly add any more edges to the edge set $\langle V_1, V_2 \rangle$, i.e. the graph contains exactly all edges that have one end vertex in V_1 and the other end vertex in V_2.

•If there are n vertices in V_1 and m vertices in V_2, we denote it as $K_{n,m}$ (cf. complete graph).

•NB. Usually, $K_{n,m}$ and $K_{m,n}$ are considered to be the same.



Labeling

•By a labeling of the vertices of the graph G = (V, E), we mean a mapping α : V → A, where A is called the label set.

•A labeling of the edges is a mapping β : E → B, where B is the label set.

•Often, these labels are numbers. Then, we call them weights of vertices and edges.

•In a weighted graph, the weight of a path is the sum of the weights of the edges traversed.

Injective labeling

•The labeling of the vertices is injective if distinct vertices have distinct labels.

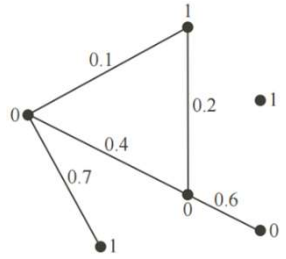
•The labeling of the edges is injective if distinct edges have distinct labels.

•An injective labeling is bijective if there are as many labels in A (respectively in B) as the number of vertices (respectively edges).

Injective labeling: example

•A = {0,1} and B = $\mathbb{R}$.

•The labeling of the edges is injective, but the labeling of the vertices is not.

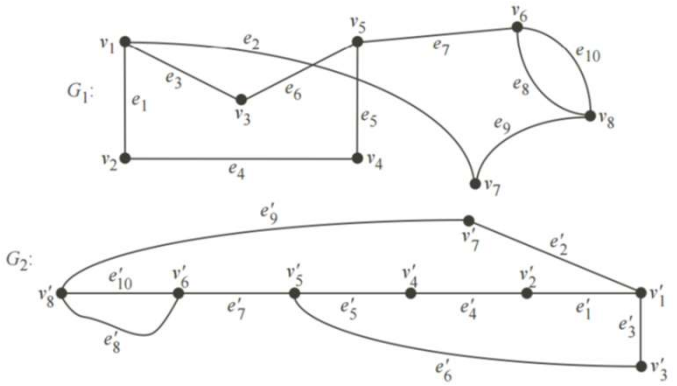


Isomorphic graphs

•The two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if labeling the vertices of G_1 bijectively with the elements of V_2 gives G_2.

•Please note that we have to maintain the multiplicity of the edges.

Isomorphic graphs: example



The vertex labeling $v_i \rightarrow v'_i$ and edge labeling $e_j \rightarrow e'_j$ define the isomorphism.

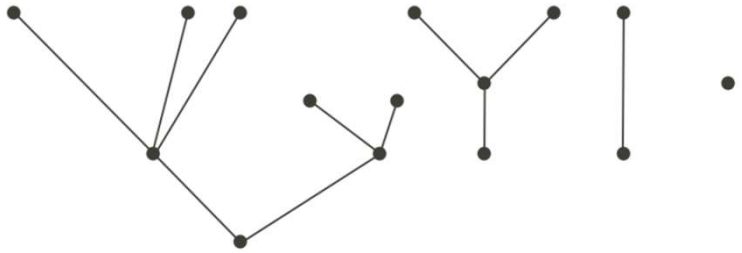
Graph Theory

Trees and forests

Trees

- A forest is a circuitless graph.
- A tree is a connected forest.
- A subforest is a subgraph of a forest.
- A connected subgraph of a tree is a subtree.
- Generally speaking, a subforest of a graph is its subgraph, which is also a forest; and
a subtree of a graph is its subgraph, which is also a tree.

Trees: example

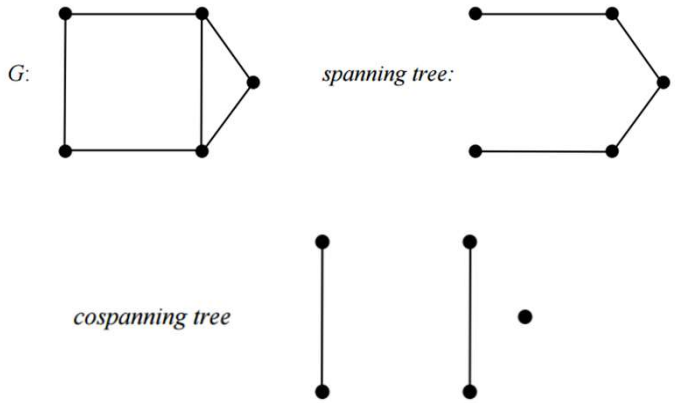


Four trees forming a forest together.

Spanning trees

- A spanning tree of a connected graph is a subtree that includes all the vertices of that graph.
- If T is a spanning tree of the graph G, then $G - T =_{def.} T^*$ is the cospanning tree.
- The edges of a spanning tree are called branches.
- The edges of the corresponding cospanning tree are called links or chords.

Spanning trees: example



Trees #2

Theorem: If the graph G has n vertices and m edges, then the following statements are equivalent:

•G is a tree.

•There is exactly one path between any two vertices in G and G has no loops.

•G is connected and m = n − 1.

•G is circuitless and m = n − 1.

•G is circuitless and if we add any new edge to G, then we will get one and only one circuit.

•Please note, that since spanning trees are trees, this theorem is also true for spanning trees.

Trees #3

•Theorem: A connected graph has at least one spanning tree.

•Please note that we can get a spanning tree of a connected graph by starting from an arbitrary subforest M.
Since there is no circuit whose edges are all in M, we can remove those edges from the circuit which are not in M.

Trees #4

The subgraph G_1 of G with n vertices is a spanning tree of G (thus G is connected) if any three of the following four conditions hold:

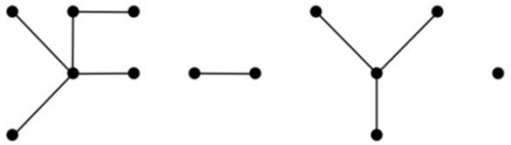
1. G_1 has n vertices.
2. G_1 is connected.
3. G_1 has n − 1 edges.
4. G_1 is circuitless.
5. Actually, conditions #3 and #4 are enough to guarantee that G_1 is a spanning tree.

Trees #5

•Theorem: If a tree is not trivial, then there are at least two pendant vertices.

•(NB: A graph with only one vertex is trivial. A pendant vertex is a vertex whose degree is 1.)

•A forest with k components is sometimes called a k-tree. (So a 1-tree is a tree.)



A 4-tree example.

Fundamental circuits

•If the branches of the spanning tree T of a connected graph G are $b_1, \ldots, b_{n-1}$ and the corresponding links of the cospanning tree T* are $c_1, \ldots, c_{m-n+1}$,
then there exists one and only one circuit C_i in $T+c_i$
(which is the subgraph of G induced by the branches of T and c_i).
We call this circuit a fundamental circuit.

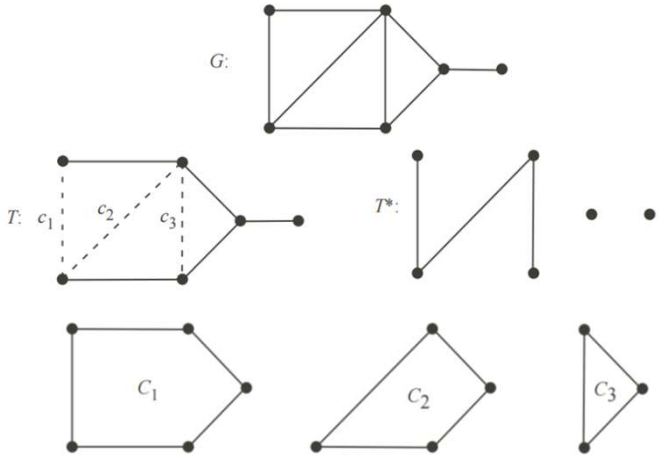
Fundamental circuits #2

•Every spanning tree defines m − n + 1 fundamental circuits $C_1, \ldots, C_{m-n+1}$, which together form a fundamental set of circuits.

•Every fundamental circuit has exactly one link which is not in any other fundamental circuit in the fundamental set of circuits.

•Therefore, we can not write any fundamental circuit as a ring sum of other fundamental circuits in the same set. In other words, the fundamental set of circuits is linearly independent under the ring sum operation.

Fundamental circuits: example



Fundamental cut sets

•The graph T−b_i has two components T_1 and T_2. The corresponding vertex sets are V_1 and V_2. Then, $\langle V_1, V_2 \rangle$ is a cut of G.

•It is also a cut set of G if we treat it as an edge set because $G - \langle V_1, V_2 \rangle$ has two components.

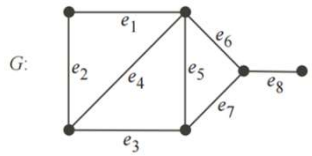
•Thus, every branch b_i of T has a corresponding cut set I_i. The cut sets I_1, ... , I_{n-1} are also known as fundamental cut sets and they form a fundamental set of cut sets.

•Every fundamental cut set includes exactly one branch of T and every branch of T belongs to exactly one fundamental cut set.

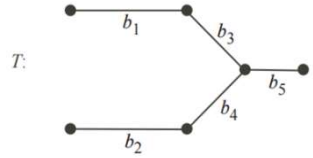
•Therefore, every spanning tree defines a unique fundamental set of cut sets for G.

Fundamental cut sets: example

Graph:



Spanning tree:



Fundamental cut sets:

$$b_1 : \{e_1, e_2\} \quad b_2 : \{e_2, e_3, e_4\} \quad b_3 : \{e_2, e_4, e_5, e_6\}$$
$$b_4 : \{e_2, e_4, e_5, e_7\} \quad b_5 : \{e_8\}$$

Some properties of fundamental circuits and cut sets

•Every cut set of a connected graph G includes at least one branch from every spanning tree of G.

• Every circuit of a connected graph G includes at least one link from every cospanning tree of G.

Some properties of fundamental circuits and cut sets #2

•Theorem. The edge set F of the connected graph G is a cut set of G if and only if

•F includes at least one branch from every spanning tree of G, and

if H ⊂ F, then there is a spanning tree none of whose branches is in H.

•Theorem. The subgraph C of the connected graph G is a circuit if and only if
C includes at least one link from every cospanning tree of G, and if D is a subgraph of C and D≠C, then there exists a cospanning tree none of whose links is in D.

Some properties of fundamental circuits and cut sets #3

- Theorem. A circuit and a cut set of a connected graph have an even number of common edges.
- Theorem. A fundamental circuit corresponding to link c of the cospanning tree T* of a connected graph is formed exactly by those branches of T whose corresponding fundamental cut set includes c.
- Theorem. The fundamental cut set corresponding to branch b of the spanning tree T of a connected graph consists exactly of those links of T* whose corresponding fundamental circuit includes b.
- As an important result, the theorems for cut sets can generally be converted to dual theorems for circuits and vice versa.

Graph theory

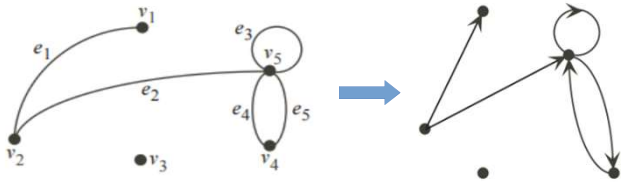
Directed graphs

Directed graphs

•A directed graph or digraph is a pair (V, E), where V is the vertex set and E is the set of vertex pairs, and the elements of E are ordered pairs.

•The arc from vertex u to vertex v is written as (u, v) and the other pair (v, u) is the opposite direction arc.

•We also have to keep track of the multiplicity of the arc (direction of a loop is irrelevant).



Notations

•Vertex u is the initial vertex and vertex v is the terminal vertex of the arc (u, v). We also say that the arc is incident out of u and incident into v.

•The out-degree of the vertex v is the number of arcs out of it (denoted $d^+(v)$) and the in-degree of v is the number of arcs going into it (denoted $d^-(v)$).

• In the directed walk (trail, path or circuit)

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k}$$

$v_{i_{\ell-1}}$ is the initial vertex, v_{i_ℓ} is the terminal vertex of the arc e_{j_ℓ} .

Notations #2

•The undirected graph (V, E) is called the underlying undirected graph of the digraph G = (V, E), denoted G_u

•Digraph G is connected if G_u is connected. The components of G are the directed subgraphs of G that correspond to the components of G_u. The vertices of G are connected if they are connected in G_u.

•Other notions for undirected graphs can be used for digraphs as well by dealing with the underlying undirected graph.

Strongly connected digraphs

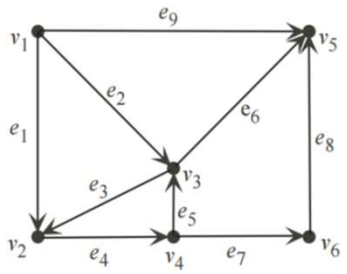
•Vertices u and v are strongly connected if there is a directed u–v path and also a directed v–u path in G.

•Digraph G is strongly connected if every pair of vertices is strongly connected. By convention, the trivial graph is strongly connected.

• A strongly connected component H of the digraph G is a directed subgraph of G (not a null graph) such that H is strongly connected, but if we add any vertices or arcs to it, then it is not strongly connected anymore.

•NB. Every vertex of the digraph G belongs to one strongly connected component of G. However, an arc does not necessarily belong to any strongly connected component of G.

Strongly connected components: example

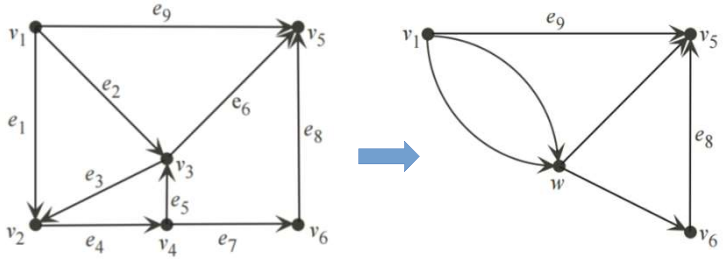


Strongly connected components:

$(\{v_1\}, \emptyset), (\{v_2, v_3, v_4\}, \{e_3, e_4, e_5\}), (\{v_5\}, \emptyset)$ *and* $(\{v_6\}, \emptyset)$

Condensed graph

•The condensed graph G_c of the digraph G is obtained by contracting all the arcs in every strongly connected component.



Directed trees

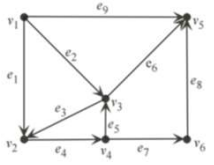
A directed graph is quasi-strongly connected if one of the following conditions holds for every pair of vertices u and v:

(i) u = v or

(ii) there is a directed u–v path in the digraph or

(iii) there is a directed v–u path in the digraph or

(iv) there is a vertex w so that there is a directed w–u path and a directed w–v path.

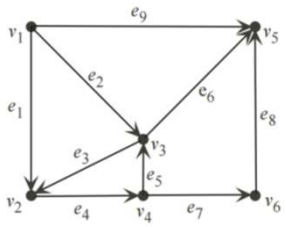


The digraph G is quasi-strongly connected.

Root

•Quasi-strongly connected digraphs are connected but not necessarily strongly connected.

•The vertex v of the digraph G is a root if there is a directed path from v to every other vertex of G.



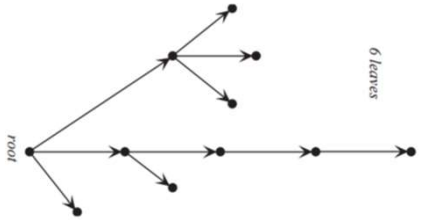
The digraph G has only one root, i.e. v_1.

Root #2

•Theorem. A digraph has at least one root if and only if it is quasi-strongly connected.

•The digraph G is a tree if G_u is a tree. It is a directed tree if G_u is a tree and G is quasi-strongly connected, i.e. it has a root.

•A leaf of a directed tree is a vertex whose out-degree is zero.



Directed trees #2

Theorem. For the digraph G with n > 1 vertices, the following are equivalent:

•G is a directed tree.

•G is a tree with a vertex from which there is exactly one directed path to every other vertex of G.

•G is quasi-strongly connected but G − e is not quasi-strongly connected for any arc e in G.

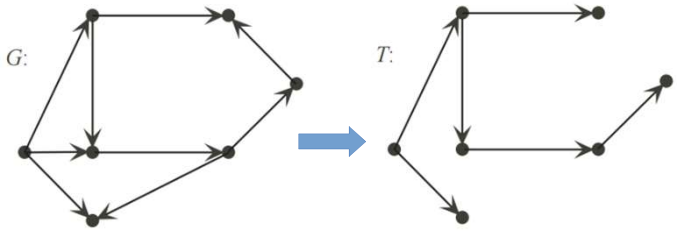
•G is quasi-strongly connected and every vertex of G has an in-degree of 1 except one vertex whose in-degree is zero.

•There are no circuits in G (i.e. not in G_u) and every vertex of G has an in-degree of 1 except one vertex whose in-degree is zero.

•G is quasi-strongly connected and there are no circuits in G (i.e. not in G_u).

Directed spanning trees

•A directed subgraph T of the digraph G is a directed spanning tree if T is a directed tree and T includes every vertex of G.

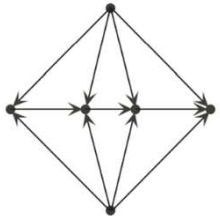


•Theorem. A digraph has a directed spanning tree if and only if it is quasi-strongly connected.

Acyclic directed graphs

•A directed graph with at least one directed circuit is said to be cyclic.

•A directed graph is acyclic otherwise. Obviously, directed trees are acyclic but the reverse implication is not true.



This digraph is acyclic but not a directed tree.

Acyclic directed graphs #2

•Theorem. In an acyclic digraph, there exist at least one source (a vertex whose in-degree is zero) and at least one sink (a vertex whose out-degree is zero).

•If G = (V, E) is a digraph with n vertices, then a labeling of the vertices with an injective function $\alpha : V \rightarrow \{1, \ldots, n\}$ which satisfies the condition $\alpha(u) < \alpha(v)$ whenever (u, v) is an arc in G is known as topological sorting.

•Theorem. We can sort the vertices of a digraph topologically if and only if the graph is acyclic.

Graph theory

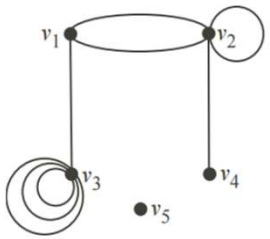
Matrix representation of graphs

Adjaceny matrix

•The adjacency matrix of the graph G = (V, E) is an n × n matrix D = (d_{ij}), where n is the number of vertices in G, V = {v_1,..., v_n} and d_{ij} = number of edges between v_i and v_j .

•In particular, d_{ij} = 0 if (v_i, v_j) is not an edge in G.

•The matrix D is symmetric, i.e. D^T = D.

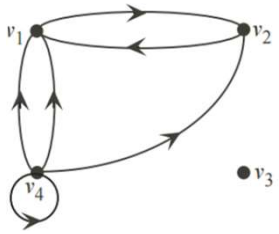


$$\mathbf{D} = \begin{pmatrix} 0 & 2 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Adjaceny matrix #2

•Obviously, an adjacency matrix defines a graph completely up to an isomorphism.

•The adjacency matrix of a directed graph G is D = (d_{ij}), where d_{ij} = number of arcs that come out of vertex v_i and go into vertex v_j .

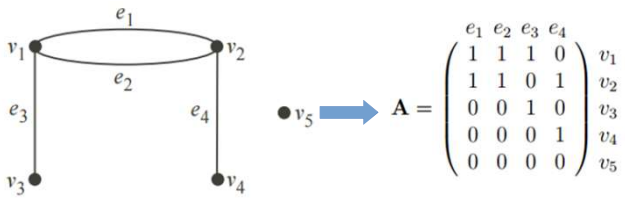


$$\mathbf{D} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

All-vertex incidence matrix

•The all-vertex incidence matrix of a non-empty and loopless graph G = (V, E) is an n × m matrix A= (a_{ij}), where n is the number of vertices in G and m is the number of edges in G and

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is an end vertex of } e_j \\ 0 \text{ otherwise.} \end{cases}$$

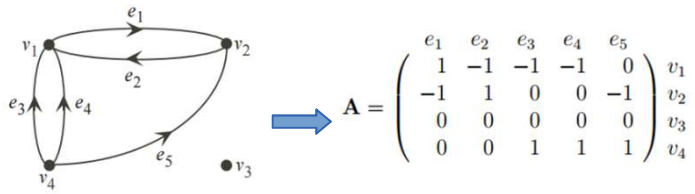


$$\mathbf{A} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix}$$

All-vertex incidence matrix #2

•The all-vertex incidence matrix of a non-empty and loopless directed graph G is matrix A= (a_{ij}), where

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is the initial vertex of } e_j \\ -1 \text{ if } v_i \text{ is the terminal vertex of } e_j \\ 0 \text{ otherwise.} \end{cases}$$



$$\mathbf{A} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 \end{matrix} \\ \begin{pmatrix} 1 & -1 & -1 & -1 & 0 \\ -1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix}$$

Incidence matrix #2

•Similarly, every column in the all-vertex incidence matrix of a digraph contains exactly two non-zero numbers, +1 and −1.

•We can remove a row from the all-vertex incidence matrix and obtain the incidence matrix.

•Notice that the rows of an all-vertex incidence matrix are linearly dependent because the sum of rows is a zero vector.

Incidence matrix

•Since every column of an all-vertex incidence matrix contains exactly two non-zero numbers, two ones, we can remove a row and still have enough information to define the graph.

•The incidence matrix of a graph is obtained by removing a row from the all-vertex incidence matrix.

•NB. It is not unique because there are n possible rows to remove.

•The vertex corresponding to the row removed is called the reference vertex.

Party Problem

Mutual acquaintances and strangers
Ramsey numbers

http://mathforum.org/mathimages/index.php/The_Party_Problem_(Ramsey's_Theorem)

Ramsey numbers

•How many people do we need to invite to make sure that at least 3 people will be mutual acquaintances or at least 3 people will be mutual strangers?

•In general, the party problem, also known as the maximum clique problem, asks to find the minimum number of guests that must be invited so that at least m will know each other or at least n will not know each other.

•The solutions are known as Ramsey numbers.

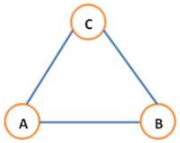
Trivial case

Blue edge: people know each other
Red, dashed edge: strangers.

In case of 2 people, they either know each other or do not know each other. Therefore there is only one possible edge. Please note that this case does not satisfy the requirement.

Nontrivial cases: 3 guests

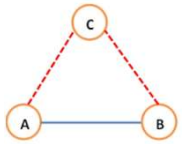
•Now, let us invite a third person.
•In case they know each other, then the requirement is satisfied.



•This kind of triangle with the edges of the same colour is called a monochromatic triangle, indicating that the 3 people represented by the vertices are either all mutually acquaintances (or mutually strangers).

Nontrivial cases: 3 guests #2

- Now, consider the situation when the third person is not known by the first two guests:



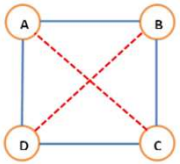
- With this good illustration of a counterexample, it is clear that 3 persons are not enough to be invited to satisfy the requirement.

4 guests

- Now, let us invite a fourth guest.
- Let's use the same counterexample strategy we used for 3 people.
- Assume that 4 is a solution to the party problem. In order to find a counterexample to our assumption, we have to find a configuration such that there exist no red or blue monochromatic triangles.

4 guests #2

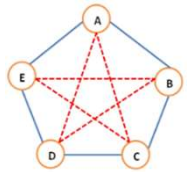
- Such a configuration is shown below:



The outer edges are colored blue, and the inner edges (the diagonals of the square) are colored red. There is no way that you could make a monochromatic triangle from the blue outer edges, or from the red inner edges. In other words, you can't form a set of 3 people that are all mutual acquaintances or mutual strangers.
We have found a counterexample, and therefore 4 is not an answer to the party problem.

5 guests

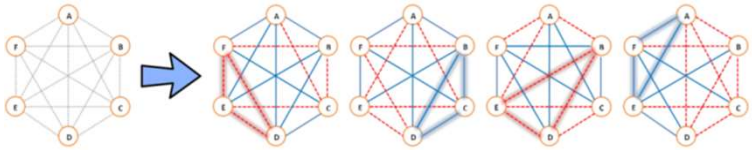
- Now, let us invite a fifth guest.
- Again assume that 5 people is an answer to the party problem. Let's use the same coloring strategy that we used for 4 people.
- A counterexmaple configuration is shown below:



Here, the outer edges are blue and the inner edges are red. There is still no monochromatic triangle from either the blue outer edges or the red inner edges.
It means that this a counterexample to our assumption, and therefore 5 people is not an answer.

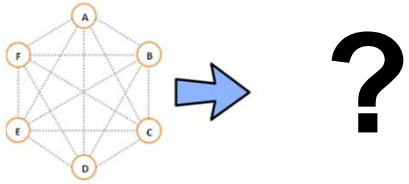
6 guests

- Now, let us invite a sixth guest.
- It is now a bit more complicated. There are various situations for example:



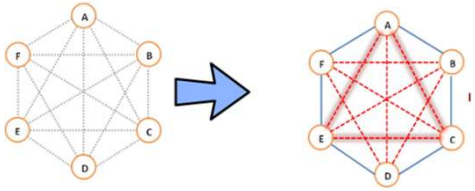
6 guests #2

- To find a configuration that will not form a monochromatic triangle, try using the coloring strategy that we used for 4 and 5 people. In other words, color the outer edges blue and the inner edges red.
- Does this work?



6 guests #3

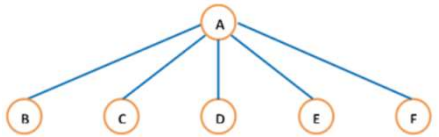
- Not really… This configuration still contains red, monochromatic triangles…



Just because there are many configurations that suggest 6 is the answer, however, doesn't prove that it is: we have to apply a more general proof to show that, no matter what, in any set of 6 people, 3 people will be mutual friends or mutual strangers.

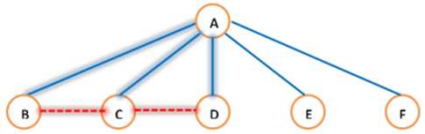
6 guests: reformulation of the representation

- Let us reconfigure the vertices so that the problem is easier to model. We will only draw the edges that originate at vertex A.
- To begin with, let's assume that all of these edges are blue.
- Later on, we'll see that it doesn't matter what color these edges are.



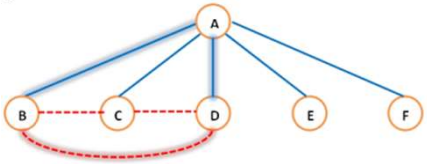
6 guests: reformulation … #2

- Please note that we are trying to find a configuration that does not produce any monochromatic triangles.
- Consider the triangle that would be formed by vertices A, B, and C. Since edges AB and AC are blue, we would have to make BC red to prevent triangle ABC from becoming monochromatic.
- Likewise, in triangle ACD, since edges AC and AD are blue, edge CD must be red.



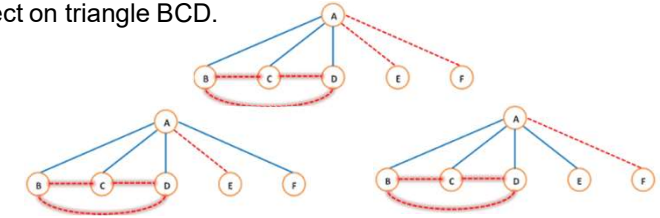
6 guests: reformulation … #3

- Now, consider the triangle that would be formed by vertices A, B, and D.
- Since edges AB and AD are blue, edge BD must be red to avoid a monochromatic triangle.
- Even though we avoided creating a blue monochromatic triangle for (ABD), we eventually forced to form a red monochromatic triangle (BCD).



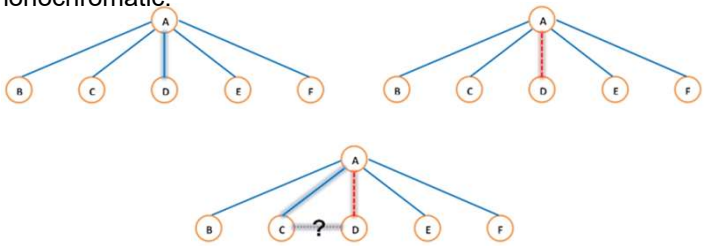
6 guests: reformulation … #4

- This holds true even if we change our initial assumption that all the edges starting from A are blue.
- If edge AE were red, edge AF were red, or both edges AE and AF were red, our end results would not change.
- This is because these are trivial changes to the configuration. We didn't use or consider edges AE and AF when we formed the monochromatic triangle BCD, so changing their colors has no effect on triangle BCD.



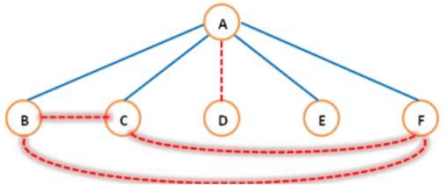
6 guests: reformulation … #5

- Now what about nontrivial changes to the configuration?
- Let us change edge AD from blue to red. If AD isn't blue, we can no longer put a constraint on edge CD—it can be either red or blue. It is no longer guaranteed that triangle BCD is monochromatic.



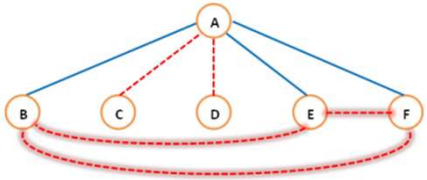
6 guests: reformulation … #6

- Let us look at this new configuration again.
- Edges AC and AF are blue, so edge CF must be red to prevent triangle ACF from becoming a blue monochromatic triangle.
- Edges AB and AF are blue, so edge BF must be red to prevent triangle ABF from becoming blue and monochromatic.
- Please note that we are once again forced to form a red monochromatic triangle (triangle BCF).



6 guests: reformulation … #7

- Even if we make a nontrivial change to the configuration and change edge AC from blue to red, we are still forced to form the red monochromatic triangle BEF.



6 guests: reformulation … #8

- Because we failed to find any case in which there exists no monochromatic triangle, we have proven that 6 vertices is a solution to the party problem for the following configurations: 5 blue and 0 red edges, 4 blue and 1 red edge, and 3 blue and 2 red edges.
- What about the other configurations? The remaining configurations are (0 blue, 5 red), (1 blue, 4 red), and (2 blue, 3 red). We can obtain the diagrams for these configurations by simply flipping the colors in the three proven configurations. For example, when the blue is replaced with red and vice-versa, the diagram for (5 blue, 0 red) becomes the diagram for (0 blue, 5 red).

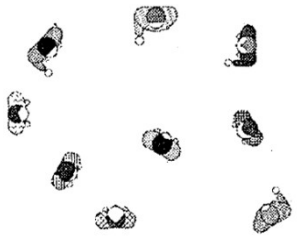
6 guests: reformulation … #9

- Since we have demonstrated that any configuration of 6 people will always contain a monochromatic triangle, our results can be generalized to all cases: to guarantee that at least 3 people will be mutual acquaintances or mutual strangers, you must invite a minimum of 6 people to the party.

Party Problem

László Barabási-Albert:
Linked: The New Science of Networks

Initial state



?

Group of people. No connections. What will happen?

Motivating situation

•Let us organise a party with 100 people.

•Let us assume that the people do not know each other, they are strangers to each other.

•Let us serve one special type of wine without a label among other party wines.

•Let us talk about this special type of wine to only one person.

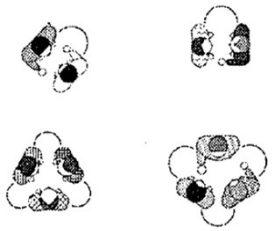
•Will the special type of wine be gone at the end of the party or not?

Obvious remarks

•Obviously, should everybody know each other, within a very short time everybody would ask for the special type of wine and it would be gone very soon.

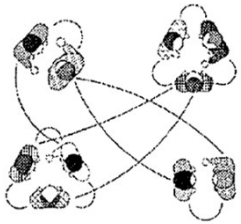
•On the other hand, considering only 10 minutes for a short conversation between people, meeting with the other 99 person it would take about 16 hours, which obviously exceeds the frame of a party, so our special wine would be safe.

First steps



At the party where nobody knows the other ones, first small groups of people are formed. Inside these small groups the small talk begins, the acquaintanceship relations begin, represented by the lines. At this time there are no connection between these small groups, everybody outside the small group is a stranger.

Time passes



As time passes, 3 persons changes their small groups, i.e. leaves their original small groups and enter a new small group and one giant group is formed. Even though not everybody knows each other, at this stage there is already one connected network of all guests, with a path leading from any guest to another.

Random graphs

•M. Karonski and A. Rucinski: The Origins of the Theory of Random Graphs. Springer, New York, 2013. ISBN 978-1-4614-7257-5.

•F. Clung and R. Graham: Erdős on Graphs: His Legacy and Unsolved Problems. Wellesley, Mass.: A.K. Peters, 1998.

•Erdős, P. Rényi, A

•(1959) "On Random Graphs I" in Publ. Math. Debrecen 6, p. 290–297 http://www.renyi.hu/~p_erdos/1959-11.pdf

•(1960) On the evolution of Random Graphs. https://www.renyi.hu/~p_erdos/1960-10.pdf

•Further papers: https://www.renyi.hu/~p_erdos/Erdos.html

Barabási's comment on Random graphs

•In a social network it is more likely to connect to someone whose node has a higher degree, i.e. a probabilistic function should be considered when a new edge of the graph is drawn.

Reduction

Problem reduction

•How can we generate the predecessor state of a partially unknown state?

•Is there such an operation with which the current state can be reached?

•Which is the predecessor state from which, with a chosen operation, the current state is reached?

Graph reduction.

Problem reduction

We have to give:

•The state space representation of the problem;

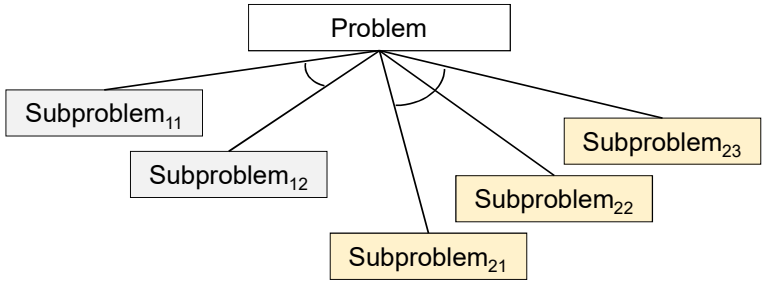
•For each operation a reduction operation is defined which orders all predecessor states to a given state; from which predecessor states the given state can be reached by performing the operation.

•We may reach false states or disinterested states by the problem reduction.

Problem decomposition

•In general problem reduction is decomposition.

•In case there is a complex problem, it may be interesting to divide or split it into sub parts.

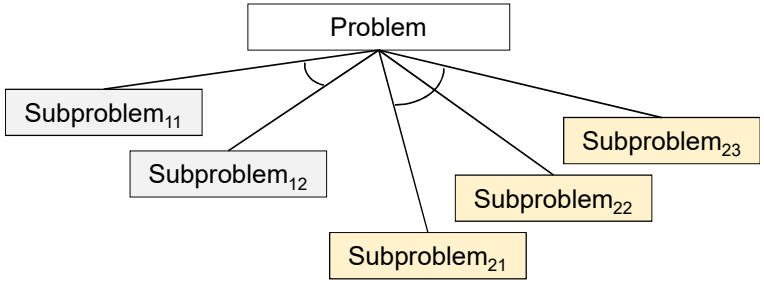


Special graph

• Lunch example: i) soup and main course; or ii) starter, soup, main course and dessert.

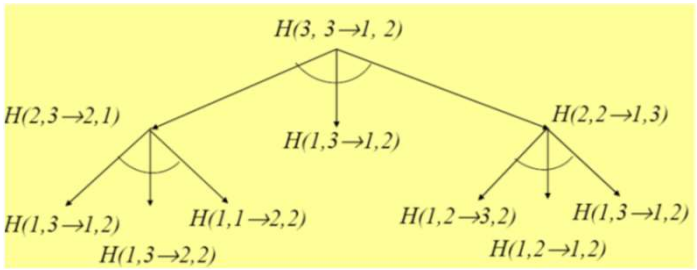
•Please note that this is a special graph.

•Should we solve all subproblems, the problem is also solved.

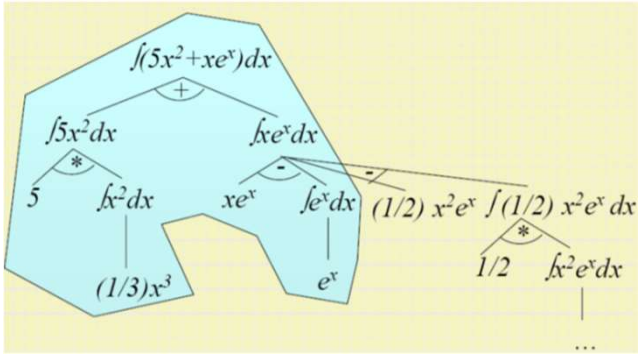


Example: Towers of Hanoi

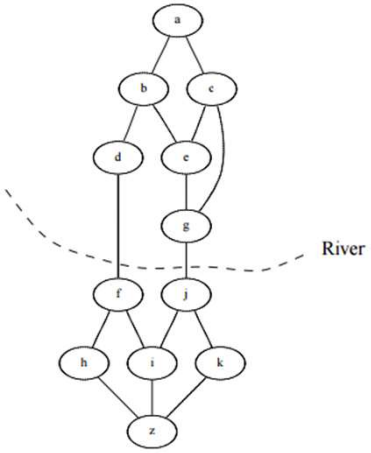
•H(n, i→j, k) = H(n-1, i→k, j), H(1, i→j, k), H(n-1, k→j, i)



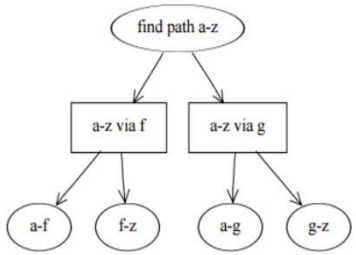
Example: Integral calculus



Example: finding the path



Idea: any solution has to use a bridge.



Daniel Polani, http://homepages.stca.herts.ac.uk/~comqdp1/AI_2/AdvancedSearch.pdf

Problem decomposition

We have to give:

•The original problem,

•The general description of the subproblems,

•Atomic problems, where it is obvious that it can or cannot be solved (for example the integral of x is known); and

•The decomposed operations:

D(p) = <p1, …, pn>

Decomposition: difficulties

It is difficult in general, since

•It is difficult to find the operations of the decomposition;

•Sometimes it is not easy to see a solution of a simple problem: for example:

$\int \sin(x)e^x dx = \ldots = \sin(x)e^x - \cos(x)e^x - \int \sin(x)e^x dx$

•Not all problems can be decomposed;

•We may leave out a subpart when we split the problem;

•New solutions may arise when we split the problem;

•There may be false states or false operations;

•It is difficult to understand the solution.

Graph representation

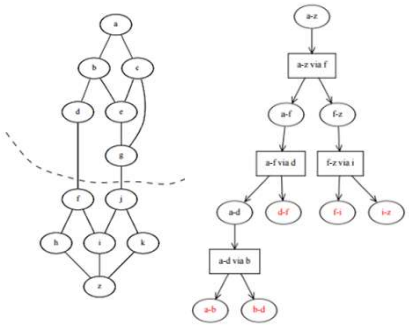
•The problem space cannot be represented by a simple directed graph, but an AND/OR graph or hypergraph.

•The solution cannot also be represented as a directed path, but a special sub-graph, the solution graph or solution tree.

•The solution graph should determine an unequivocal direction from the start node to the target node(s).

•The solution is not the solution graph, but it can be determined with the help of the solution graph.

Example: finding the path #2



Decomposition:

for path a–z via f find
path a–f
and path f–z

analogously for g

And/Or Graph: *or*-nodes as ellipses, *and*-nodes as boxes

Note: goal "nodes" are subproblems that are trivial or atomic, e.g. direct route from a–c

Daniel Polani, http://homepages.stca.herts.ac.uk/~comqdp1/AI_2/AdvancedSearch.pdf

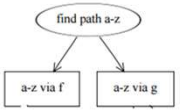
Hypergraph

•The problem is represented as the root of the tree.

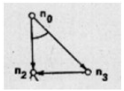
•A node may have AND children and OR children.

•If P is an AND node: all of its children (with their solution graph) has to be in the solution graph.

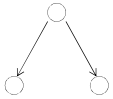
•If P is an OR node: exactly one child (with its solution graph) is in the solution graph.



Ellipses: or nodes;
Boxes: and nodes.



AND children.



OR children.

Hypergraph #2

•If every children of each node in a graph is an OR child, then the graph is a simple directed graph.

•Hypergraphs can be extensively used for cases where the problem can be formulated with operations of AND and OR.

Hyperedge and hyperpath

•The group of edges leading to an AND child is called a hyperedge.

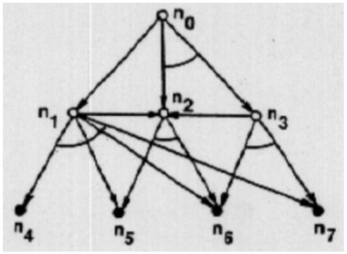
•(n,M) is a hyperedge,where from node n (the parent) there are AND edges to M={m_1, …, m_k} (the set of children).

•Hyperpath is a subgraph, where for each node there is exactly one hyperedge leading out.

•If the start node of the hyperpath is n and the set of end nodes is K, then n→K denotes the hyperpath from n to K.

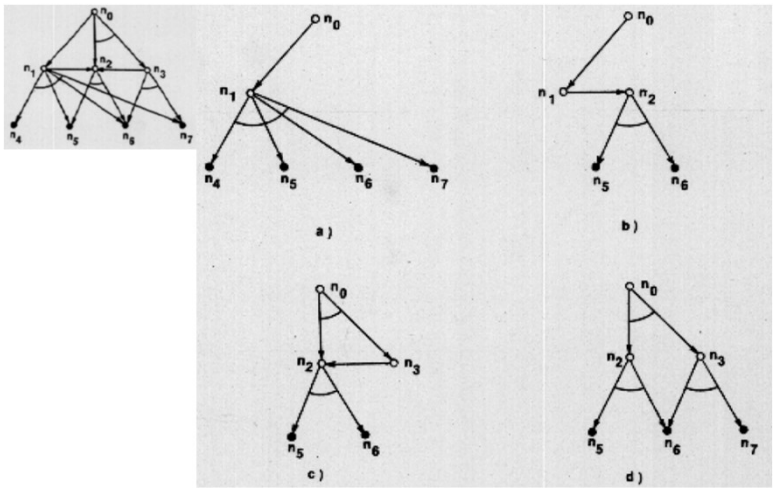
•Solution graph: such a hyperpath that leads from the start node to the set of solution nodes.

Example

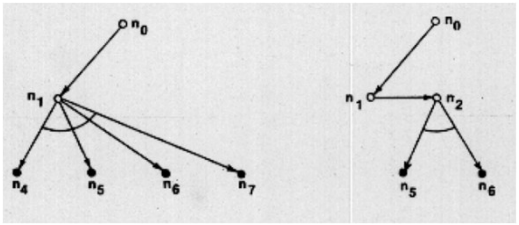


The problem is to solve n_0: we split the problem and either we solve n_1 or we solve (n_2 and n_3); etc.
Please note that the problem can be a complex one, for example Prolog uses the same logic.

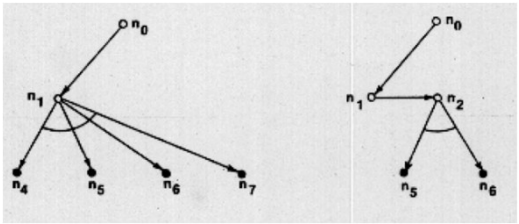
Example #2



Example: cost?

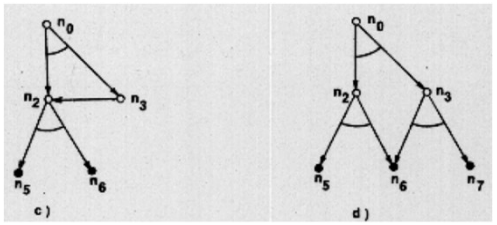


Example: cost?



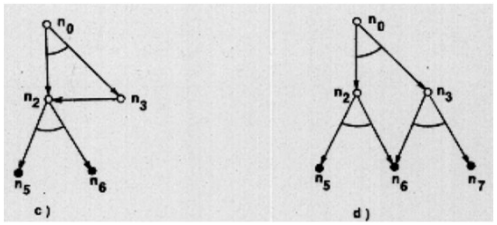
The cost: 5 4

Example: cost?



The cost:

Example: cost?



The cost: 7 6

Hypergraph: cost

•For the cost of the hypergraph, we have to consider the cost of the edges and the cost of the nodes as well.

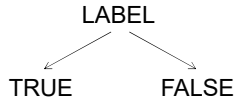
Hypergraph: search

•The search algorithms introduced for ordinary graphs cannot be applied without modifications.

•The main reason behind is the fact that we are looking for a sub graph, i. e. a solution graph in this case and not only a path from the start node to a terminal node.

Hypergraph: labeling

•The leaves of the graph are primitive/ atomic problems; the corresponding decision can always be done. They represent either a target state or they are unsolvable; either true or false, etc.



Labeling procedure

•An arbitrary node is considered to be solved, if
  – It represents a primitive problem;
  – It has at least one OR child which is solveed and has a label;
  – All of its AND children are solved and has a label.
•A node is considered to be unsolvable, if
  – It has no children but it is not a primitive problem;
  – All of its OR children are unsolvable;
  – It has at least one unsolvable AND child.
•The question is the label of the root.

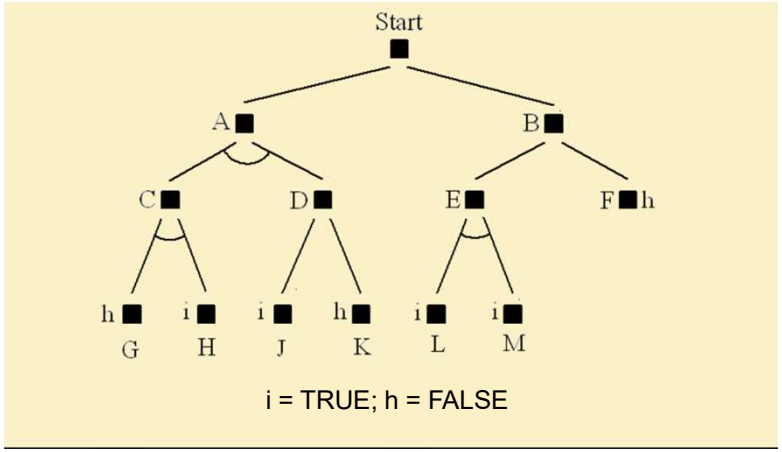
Labeling procedure #2

•We start from the leaves and label all nodes.
•Please note that from the leaves backwards, up to the root we can label each node!
•If we have a finite graph, all nodes, including the root, receives a label.
•The label of the root denotes whether the original problem is solvable or not.

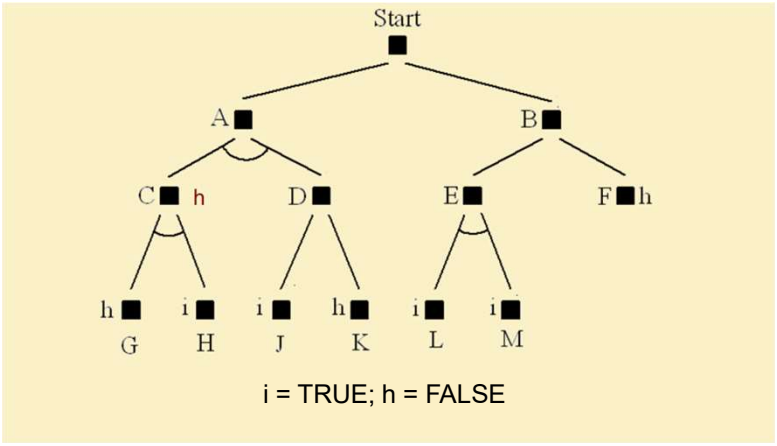
Labeling procedure #3

•The solution graph shows for the original problem which subproblems should be solved but obviously does not solve the problem itself.

Labeling example

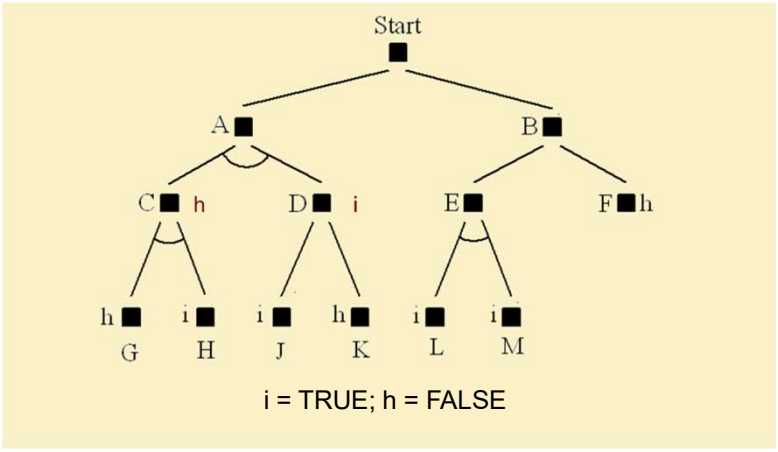


Labeling example



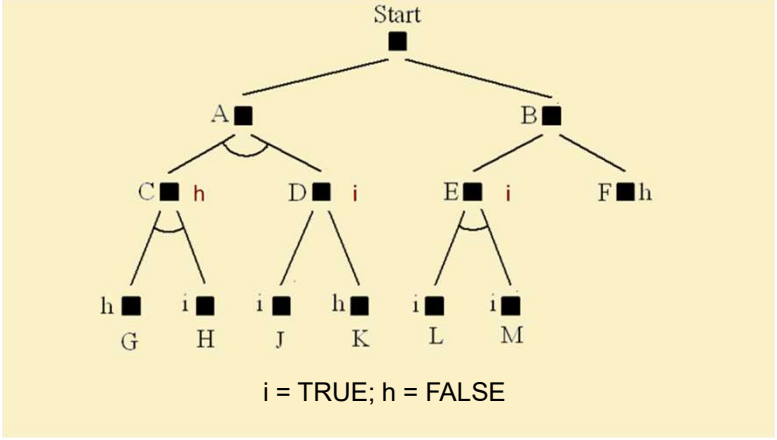
i = TRUE; h = FALSE

Labeling example



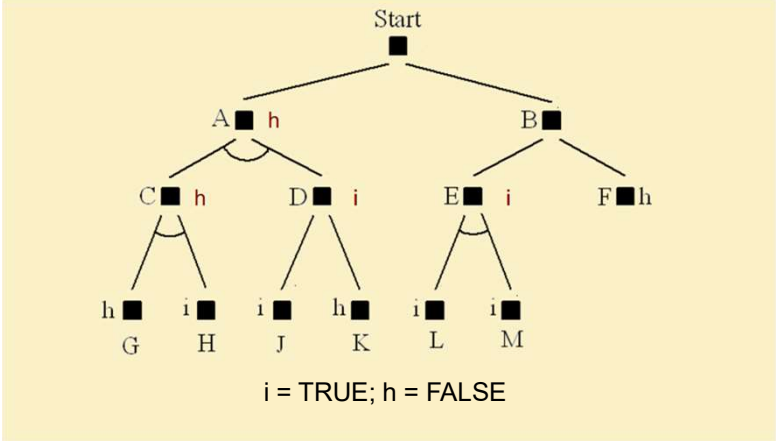
i = TRUE; h = FALSE

Labeling example



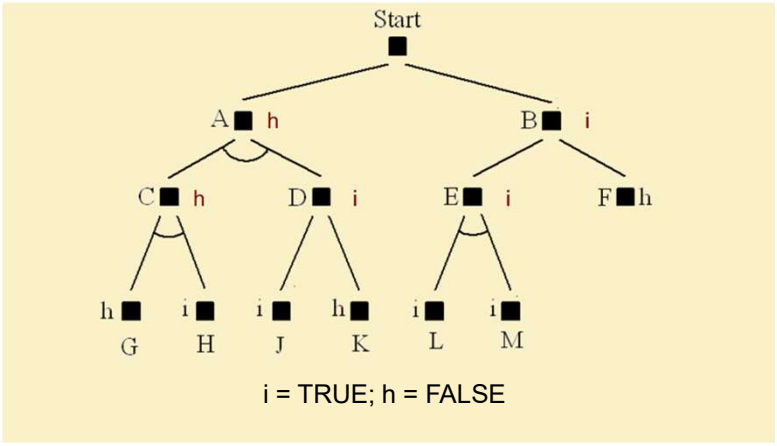
i = TRUE; h = FALSE

Labeling example



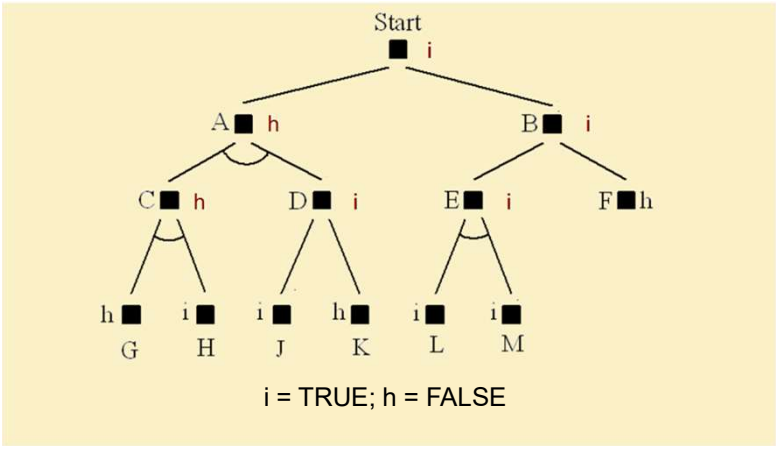
i = TRUE; h = FALSE

Labeling example



i = TRUE; h = FALSE

Labeling example



i = TRUE; h = FALSE

Games

Motivation

•1957, Herbert Simon: computers will beat the chess masters within 10 years.

•László Mérő: Észjárások. ISBN 9638453303.

•First it was published at the end of the 80s. He said that there will be no chess program to beat a chess master.

•Then came Deep Blue from IBM. Strong hardware, fast computing capacity, lots of possibilities were tried by the algorithms.

•So he published a revision of the book stating now, ok, there is Deep Blue but there will be no go programs to beat a go master.

•Then in 2016, there was AlphaGo.

Games

Why are games important?

•The aim is to develop a strategy, how we can reach our goal, while we meet obstacles, i. e. somebody else counteracts, tries to prevent us from winning.

•There are two players (with the same level of intelligence).

•There are rules and constraints of the game.

•The interest of the players are usually differ, thus it can be considered as a conflict model.

Games #2

•There are two players, who take turns alternatively, one after the other.

•Perfect information: Both players know their own and the opponent's all choices for the next move together with the consequences of the move. NB. in a card game we do not see the opponent's cards.

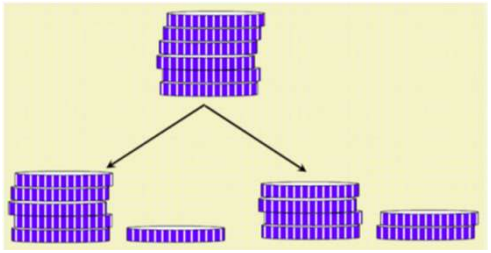
•Finite: Both players have a finite number of choices in each case; the game ends within a finite number of steps. (NB: we have a finite graph.)

•Zero-sum: One wins as much as the other looses. Simplest version: one wins, the other loses, may be tie.

Grundy's game

•Grundy mama illustrates the game problem for us.

•There is a pile of coins or beer caps. The two players take turn splitting a single pile into two piles of different sizes. The game ends when only such piles remain, which cannot be split unequally. The person who made the last move wins.



NB. There are no other choices.

State space representation

•State: current state + name of the next player.

•Initial state: initial state + player who starts.

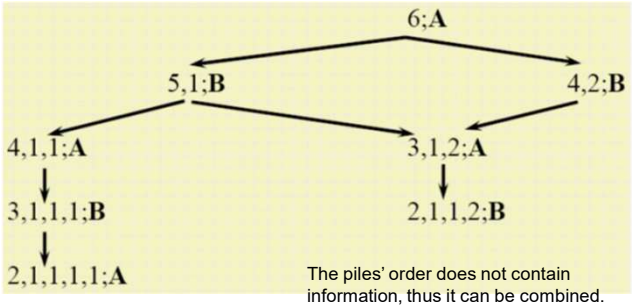
•Operation: the rules that can be applied at the current state. NB. it is finite.

•Series of operations: an actual game, always finite.

•Control: cannot be modified – alternately searches for the players; it cannot go back; the space is large.

•Target is to win (at least not to lose).

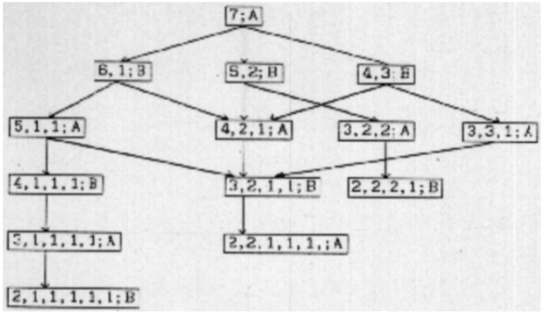
Grundy mama's game graph



The piles' order does not contain information, thus it can be combined.

This is a conventional graph, since there are 2 paths for one state.
At each level, the same player takes turn.
It has to be transformed first into a tree.

Grundy mama's game graph #2



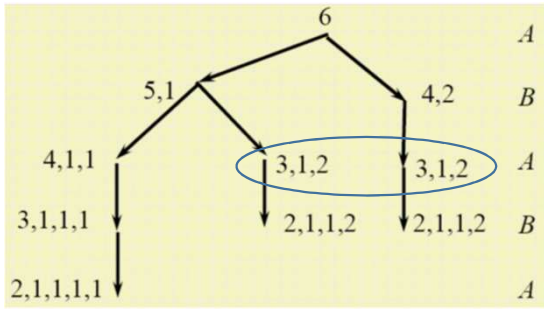
With different number of coins, the game graph is different.

Game graph

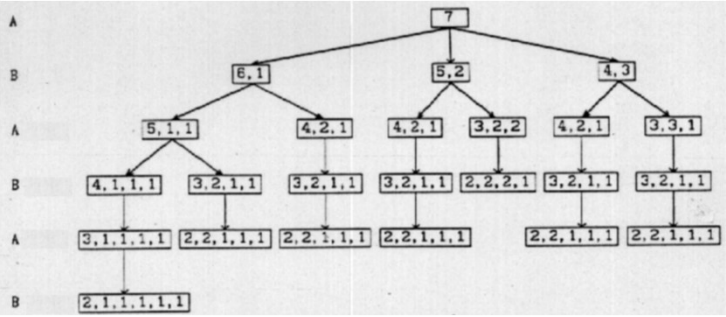
•Node: one stand + the name of the next player.
•Edge: a step from one stand to the next.
•Root: initial stand + player who starts.
•Leaves: terminal stands + winner or loser or tie.
•Path: an actual game.

Game graph – game tree

•Obvious task is to transform the game graph into a tree.
•Let us consider a state as many times as it is necessary.



Game tree example



Game tree

•Node: one stand. Please note that one stand can be represented by mutual nodes.
•Level: denoting the next player.
•Edge: a step from one stand to the next level.
•Root: initial stand + player who starts.
•Leaves: terminal stands + winner or loser or tie.
•Path: an actual game

Winning strategy

•The aim is to determine the winning strategy. Obviously it will mean different things for the different players.

•What can be considered to be the winning strategy? If there is always minimum one step that for an arbitrary step of the other player he can get to a winning terminal stand, i. e. he can win no matter the other plays (or at least he can avoid to lose).

•Obviously, this target set differs for the players.

•Key questions:

•Is there a winning strategy?

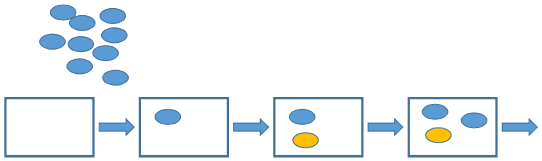
•Can both players have a winning strategy at the same time?

Remark: a game with stones

- There is a given number of stones and a box.
- There are two players.
- The player takes one stone and puts it into the box at a turn.
- The game ends when all stones are within the box.
- The winner is who takes the last turn.
- Who may have a winning strategy?

A game with stones #2

- Should we know the number of stones and who starts, all further steps are determined.
- In other words, we know who will win, it is determined at the initial state.
- There is nothing like a winning strategy here…



Another game with stones

- There is a given number of stones and a box.
- There are two players. Player A starts.
- At a turn the player takes one stone and puts it into the box, or takes two stones and puts them into the box.
- The game ends when all stones are within the box.
- The winner is who takes the last turn.
- Who may have a winning strategy?

Another game with stones #2

- Should the number of stones be not multiples of 3, then let A put as many stones into the box to achive that the number of the remaining stones be a multiple of 3. In this case he will win.
- Should the number of stones be multiple of 3, then A may only bluff and hope that the other player does not have a strategy, otherwise who has the second turn will win.
- Why? Should player B have 3 or the multiple of 3 stones, the player A can always achive that the number of remaining stones be a multiple of 3: B puts in only 1 stone, then A puts in 2 stones; B puts in 2 stones, then A puts in 1 stone only at the turn.

# stones	1	2	3	4	5	6	7
winner	A	A	B	A	A	B	A

once again… winning strategy

•The aim is to determine the winning strategy. Obviously it will mean different things for the different players.

•What can be considered to be the winning strategy? If there is always minimum one step that for an arbitrary step of the other player he can get to a winning terminal stand, i. e. he can win no matter the other plays (or at least he can avoid to lose).

•Obviously, this target set differs for the players.

•Key questions:

•Is there a winning strategy?

•Can both players have a winning strategy at the same time?

Winning strategy #2

•The aim is to determine the winning strategy; different for the different players.

•Let us consider the game tree from one of the players point of view:

•The player can decide about his next step arbitrarily; i. e. these next steps are in OR connection to each other; while

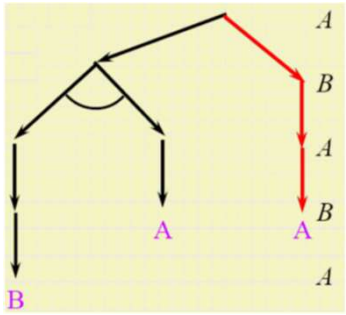
•All of the opponent's steps have to be considered, since it is independent from the players decision; i. e. those steps are in AND connection to each other.

•AND / OR GRAPH
(HYPERGRAPH)

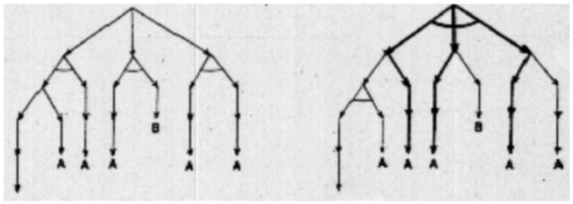
Grundy's game
(from A point of view)

•Is there a hyper path with each edge that A is the winner?

•NB. There is a winning strategy for A!



Grundy's game #2

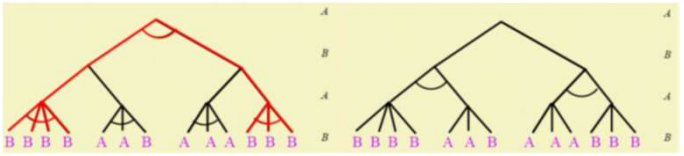


A point of view B point of view

7 coins, A starts.
First level: AND vs OR; since the point of view is different.
In this case B has a winning strategy.

In general

•Who has a winning strategy?
•Is there always a winning strategy?



•Only one of the players can have a winning strategy

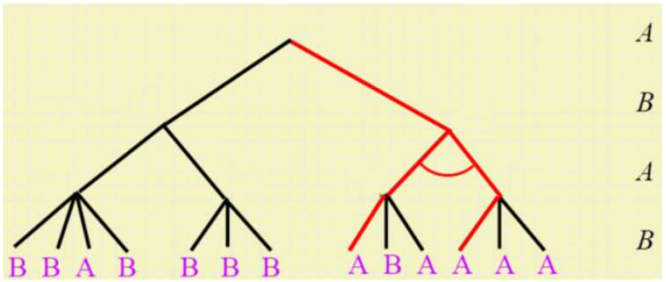
Theorem

•In a perfect information, 2 player game there is always a winning strategy for one of the players, if tie is not allowed.
•Proof: labeling procedure.

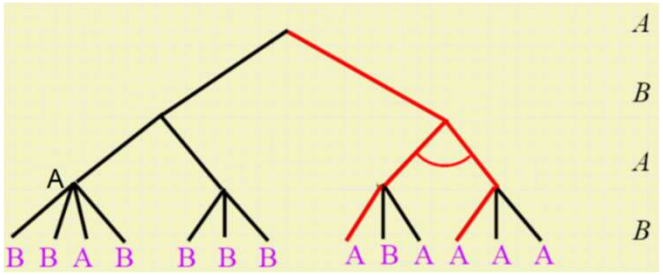
Labeling the winning strategy

•Let us construct the whole game graph.
•If we know the whole game graph, each of the leaves can be labelled: winner or loser.
•Upway, each node can receive a label:
if the player is among the children and it is his turn then the node receives the player's label, otherwise the other's.
•At each level, it is only considered who takes the turn.
•In case the graph is finite, then all nodes will receive a label and thus we will know who can have a winning strategy plus we will know the winning strategy!

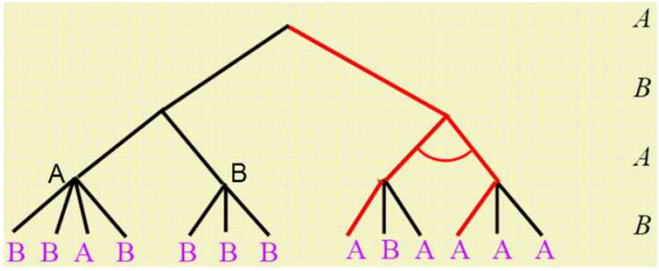
Labeling example



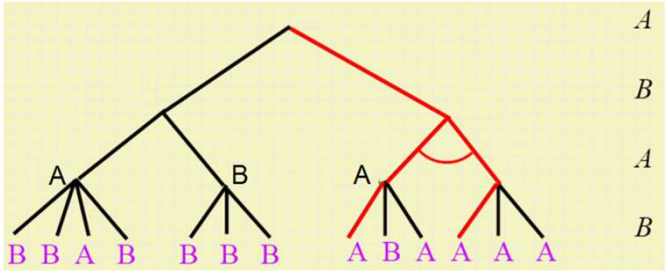
Labeling example



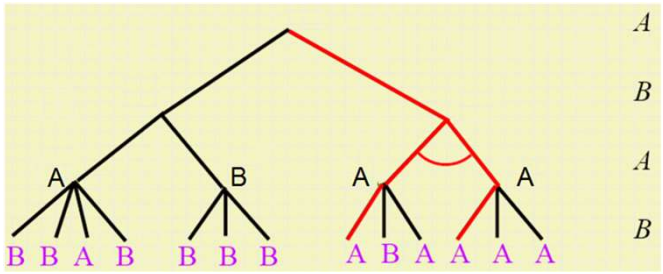
Labeling example



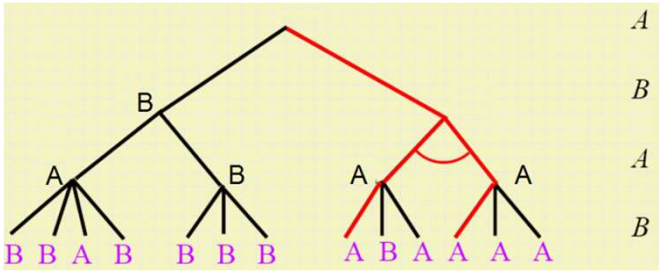
Labeling example



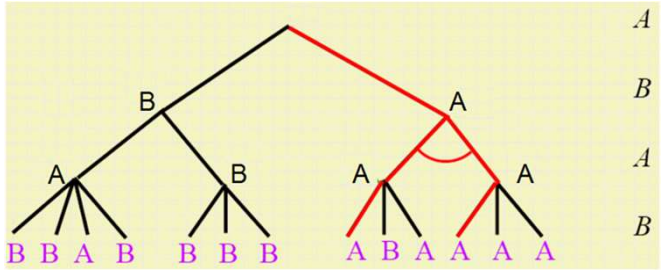
Labeling example



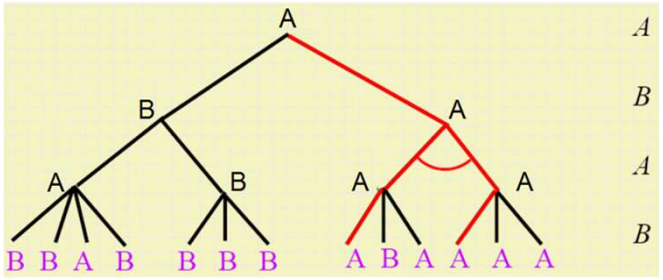
Labeling example



Labeling example



Labeling example



Theorem

•In a perfect information, 2 player game there is always a winning or minimum a not loosing strategy for one of the players, if tie is allowed.

Other games

Ex.1. There are 10 coins on the table. Two players take away 1, 2 or 3 coins at a turn. The winner is who takes the last coin.

Ex.2. There are 10 coins and 7 coins on the table. Two players take away coins alternately, any number of coins can be taken away but only from one of the piles. The winner is who takes the last coin.

•Who may have a winning strategy?

•NB: these are maths examples for students of age 10-14. Help: http://www.typotex.hu/upload/books/254/Hany_eves.pdf

Chess

•Chess is a perfect information, 2 player game, where tie is allowed.

•A usual chess game has about 45 turns → there are 90 levels of the game tree.

•The usual number of legal steps at a stand is ~35.

•The tree has 35^{90} leaves to be evaluated.

Chess reduction

•Obviously not all of the 35 possible steps are evaluated at each time.

•According to statistical investigations a master player considers 1.76 steps to be „good."

•The reduced tree has $1.25*10^{22}$ leaves to be evaluated.

•Since the formation of the Earth it passed 4.6 billion years, i. e. approximately 10^{18} seconds.

Is it good for nothing?

•Besides its theoretical importance, considering the chess example, even with the reduction, all these efforts seems meaningless.

•In practice, problems with small game trees can be searched only.

•However, the search can be accelerated with heuristics or engineering knowledge.

Deep Blue

•Herbert Simon in 1957 said that machines will be better than humans.

•IBM Deep Blue was the first to beat a chess master Garri Kaszparov 3,5:2,5 (Goodman and Keene, 1997).

Deep Blue #2

•Parallel, RS/6000 SP Thin P2SC-based system with 30 nodes.

•Each node has a 120 MHz P2SC microprocessor, enhanced with 480 special purpose VLSI chess chips.

•Its chess playing program was written in C and ran under the AIX operating system.

•It was capable of evaluating 200 million positions per second.

•In June 1997, Deep Blue was the 259th most powerful supercomputer according to the TOP500 list, achieving 11.38 GFLOPS on the High-Performance LINPACK benchmark.

Deep Blue #3

•The system derived its playing strength mainly from brute force computing power.

•Average: 126 million nodes/ second;
peak: 330 million nodes/ second.

•30 billion stands, 14 levels depth per turns.
Typically search to a depth of between six and eight moves.

•Basic algorithm: alfa-beta search + further expansion of the more interesting nodes up to a depth max. of 40.

•Evaluation function had 8000 properties.

Deep Blue #4

•Used the chess opening book with more than 4000 stands plus description of 700 000 games as a database, in addition to a large database of endgames.

•As a result of the improvements of the algorithms since 1992 many chess world chanpionships were won by programs.

AlphaGo

•More possibilities than in a chess game: „Go, a game that is exponentially more complex than chess.”

•More intuitive game.

•DeepMind accelerated the progress of computer Go using two complimentary forms of machine learning techniques that allow machines to learn certain tasks by analyzing vast amounts of digital data and, in essence, practicing these tasks on their own.

•AlphaGo was initially trained to mimic human play by attempting to match the moves of expert players from recorded historical games, using a database of around 30 million moves.

Lee Sedol and AlphaGo, 2016

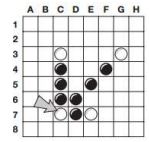
•Match score: 1:4.

•AlphaGo's algorithm uses a Monte Carlo tree search and

•Deep neural networks for learning.

•https://deepmind.com/alpha-go.html

Reversi / Othello



•Place black coloured disks on the board, while the opponent places white ones.

•Place a disk anywhere on the board horizontally, vertically, or diagonally, but it has to flip min. one of the opponent's disks.

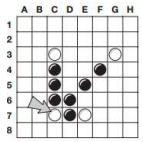
•After the placement, all surrounded disks have to be flipped over.

•Please note that there may be states when it is not possible to place a disc for one of the players; then it is the opponent’s turn.

•The game ends when there are no more legal moves.

•Target is to have more pieces on the board than the opponents.

Reversi / Othello



•The search space is smaller than that of the chess.

•Logistello is one of today's strongest Othello programs: https://skatgame.net/mburo/log.html

Type of games

	Deeterministic	Chance
Perfect information	Chess Go Othello	Backgammon Monopoly
Imperfec information		Bridge Poker Scrabble

Algorithms at the back

•Two player games evaluate the game trees only partially; i.e. full evaluation of the game tree is not considered, therefore the certain winning strategy is also not considered.

•Instead we are looking for a „strong" or „good enough" solution.

•Partial evaluation may usually mean a constraint on the depths of the search or constraint on time or memory. In this case the leaves of the search tree are not leaves of the game tree, i.e. not terminal states of the game.

•An evaluation function has to be considered for the terminal leaves of the search tree. This evaluation function gives the „goodness" of the current state at the specific node.

Partial Evaluation Algorithms

Partial evaluation algorithms

- Minimax algorithm.
- (M-N) average algorithm.
- Alpha-beta cut algorithm.
- Further developments.

Minimax algorithm

- Method for the selection of the best first step for the player in turn.
- Either to minimize the maximum value of a number of decision variables; or in some cases to minimize the possible loss for a worst case (maximum loss) scenario.

Steps:

- Generation of the game tree with the specified depth.
- Determination of the values at the terminal leaves.
- Propagation of the maximal and minimal values towards the top.
- Deceision based on the values.

Minimax principle

•An optimality principle for a two-person, zero-sum game, expressing the tendency of each player to obtain the largest sure pay-off.

•The minimax principle holds in such a game $\Gamma = \langle A, B, H \rangle$ if the equality holds: $v = \max_{a \in A} \min_{b \in B} H(a,b) = \min_{b \in B} \max_{a \in A} H(a,b)$

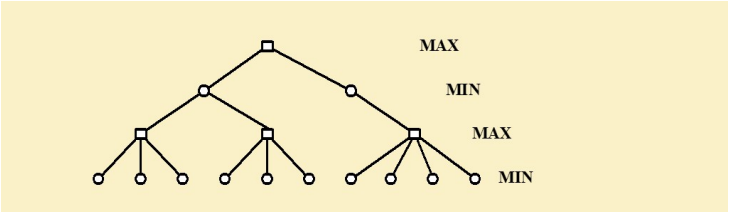
that is, if there are a value of the game equal to *v*, and optimal strategies for both players.

•The minimax principle guarantees to player I (II) a gain (loss) of not less (not more) than the value of the game.

•NB: The fact that the minimax principle holds if mixed strategies are allowed is called the minimax theorem; it is due to J. von Neumann.

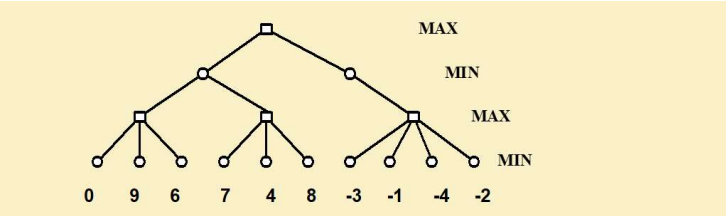
https://www.encyclopediaofmath.org/index.php/Minimax_principle

Minimax algorithm example

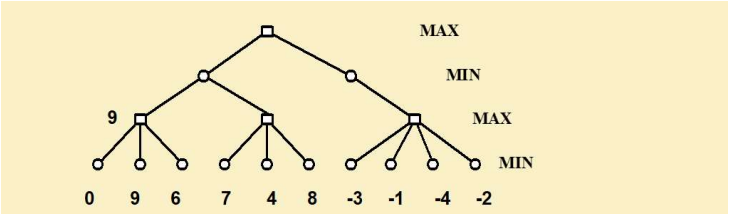


NB. Here we have MIN and MAX instead of A and B. The aim is to have the best step selected for the player.

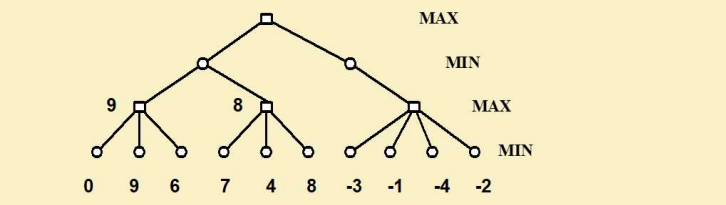
Minimax algorithm example



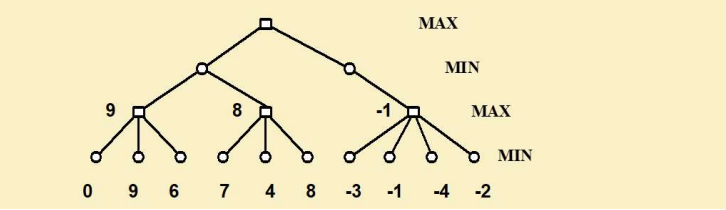
Minimax algorithm example



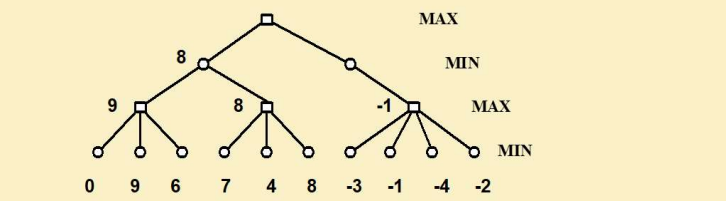
Minimax algorithm example



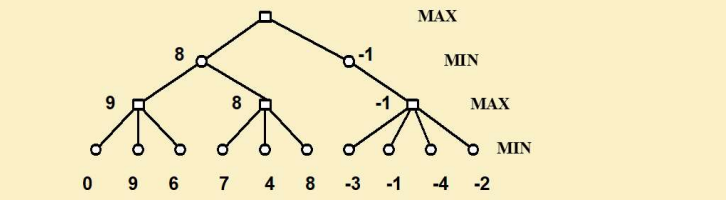
Minimax algorithm example



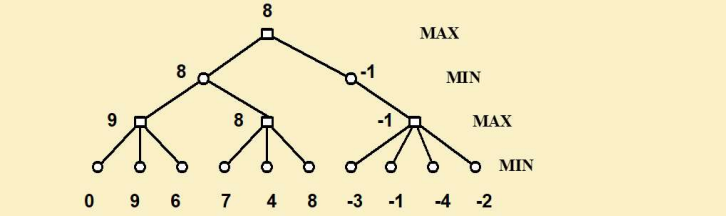
Minimax algorithm example



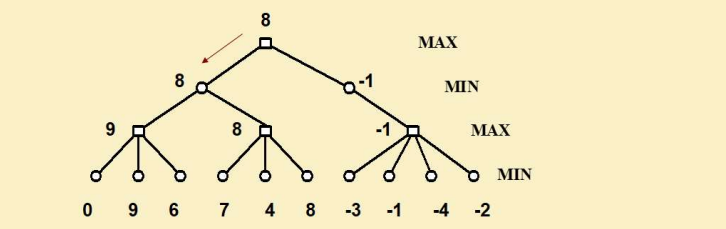
Minimax algorithm example



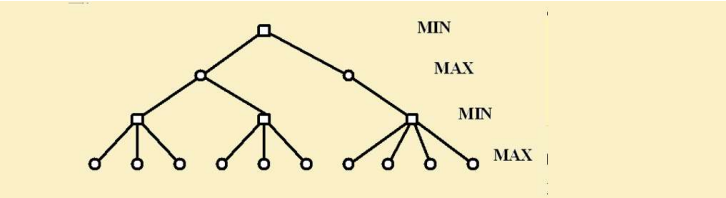
Minimax algorithm example



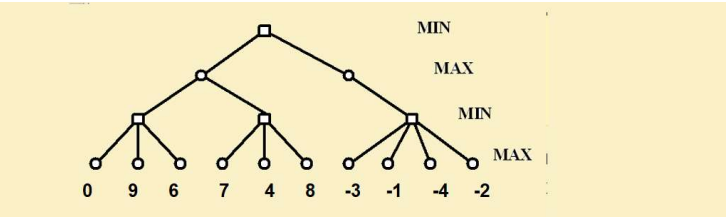
Minimax algorithm example



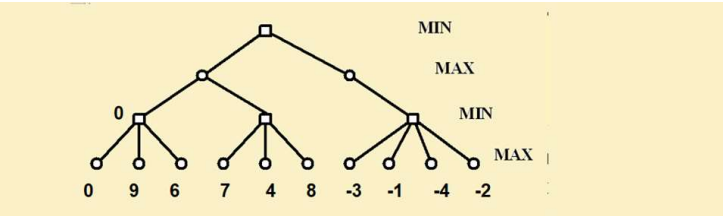
Minimax algorithm example 2



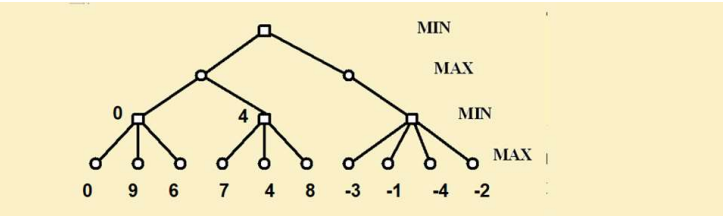
Minimax algorithm example 2



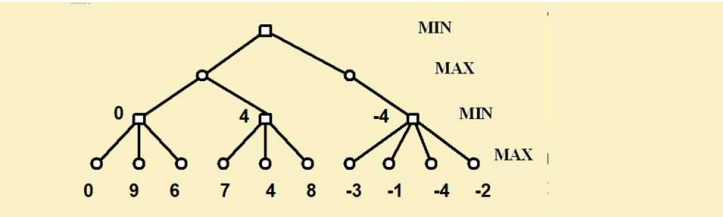
Minimax algorithm example 2



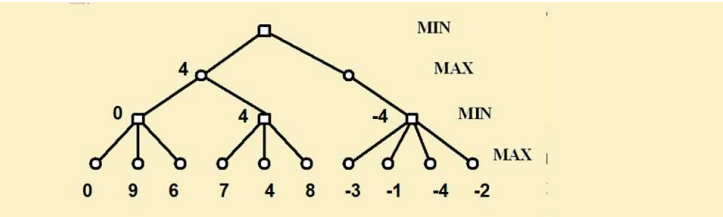
Minimax algorithm example 2



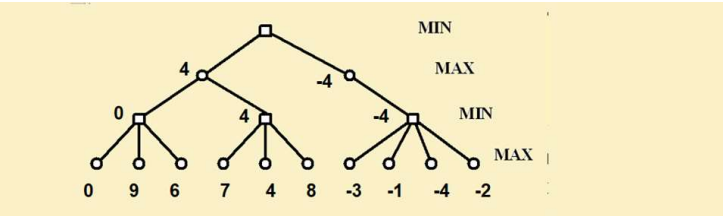
Minimax algorithm example 2



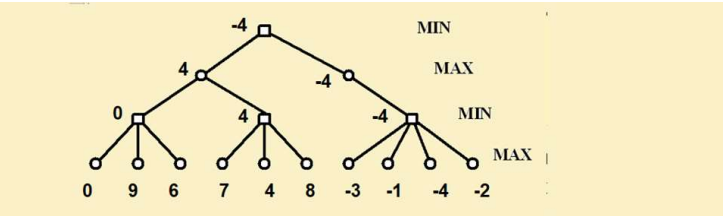
Minimax algorithm example 2



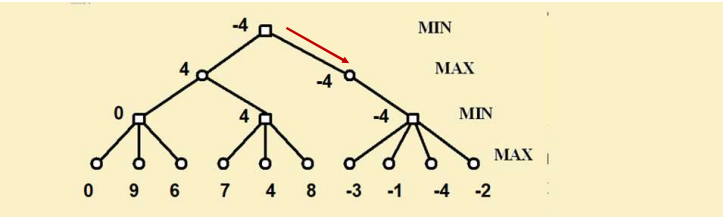
Minimax algorithm example 2



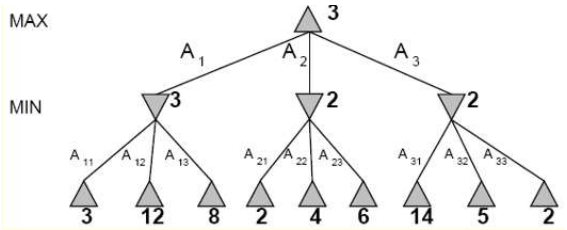
Minimax algorithm example 2



Minimax algorithm example 2



Minimax algorithm example 3



Minimax algorithm

function MINIMAX-DECISION($state, game$) **returns** *an action*

  $action, state \leftarrow$ the a, s **in** SUCCESSORS(s_{state})
    such that MINIMAX-VALUE($s, game$) is maximized
  return *action*

---

function MINIMAX-VALUE($s_{state}, game$) **returns** *a utility value*

  if TERMINAL-TEST($state$) **then**
    return UTILITY(s_{state})
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS($state$)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(s_{state})

(M-N) average algorithm

•What happens in case the other player decides in another way, i.e. not as we expected?

•To smooth the possible errors of the evaluation function we may average, or use a mean. In other words, not only one state is considered but some of the states.

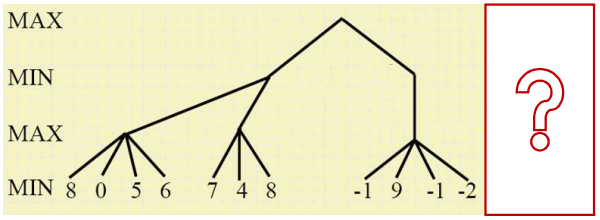
For example

•At Max levels: maximal average of M.

•At Min levels: minimal average of N.

(M-N) average algorithm example

Let M=2 and N=2.



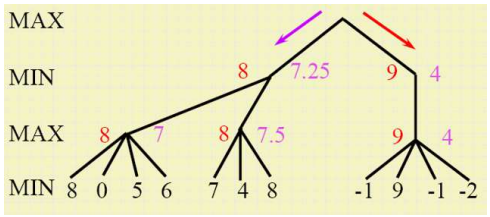
Red denotes: minimax
Purple denotes: (M-N) average

•At Max levels: maximal average of M.

•At Min levels: minimal average of N.

(M-N) average algorithm example

Let M=2 and N=2.



Red denotes: minimax
Purple denotes: (M-N) average

•At Max levels: maximal average of M.

•At Min levels: minimal average of N.

(M-N) algorithm

•May result in a different path than previously.

•May smooth some errors.

•However, there is no guarantee.

Evaluation functions

•These functions may contain information about the game, i.e. heuristics.

•Example in chess: the probability of winning seems to be in strong correlation with the number of chessmen and with their strength.

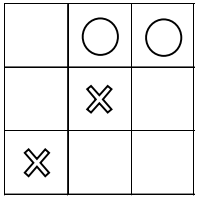
•Chessmen points:

queen: 9 = w_1,
rook: 5 = w_2,
knight and bishop: 3 = w_3 and 3 = w_4
pawn: 1 = w_5.

•f_1 (s) = number of white queens – number of black queens.

•Evaluation function example: w_1f_1 (s) + w_2f_2(s) + …

Example



Max: ⨯

Min: ◯

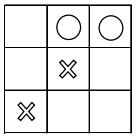
It is x's turn.

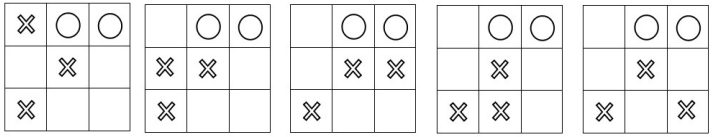
A) In 2 depths B) In 1 depth only.

Evaluation function:

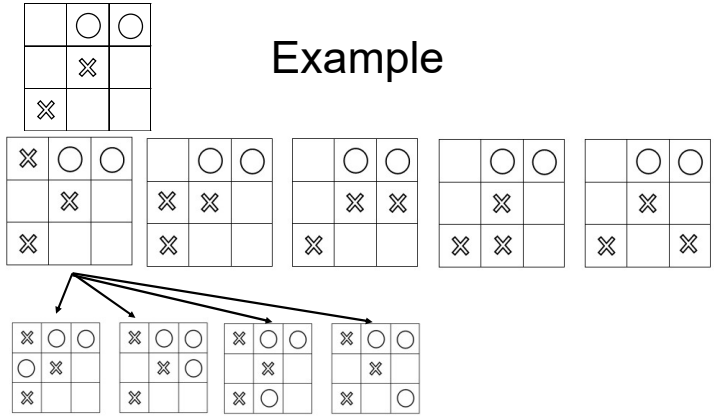
- Number of open states for this player minus the open states for the other player.
- In case of winning: ∞

Example



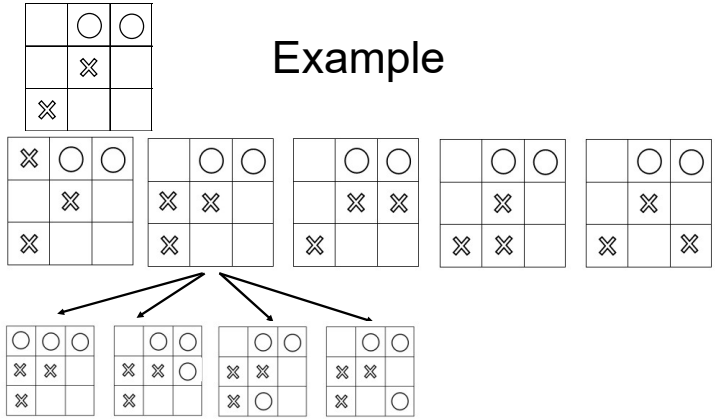


Example



We have open:
2:1 3:1 3:1 2:1

Example



We have open:
Oh, no! O will win! Our value is -∞! Do we have to evaluate the other states?

Remarks

•It is important to know how deep will we search and evaluate.
NB: on the first level there are states with equal winning chance.

•Sometimes there are states that are not important to evaluate.

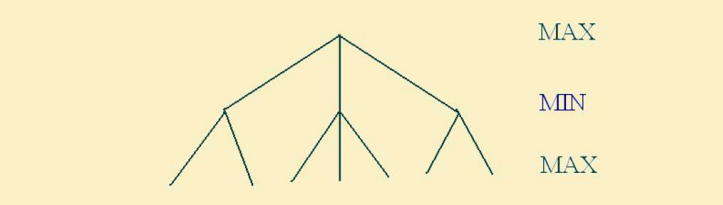
Cuts

•Previously all leaves were evaluated.

•The number of game states that has to be evaluated is exponential.

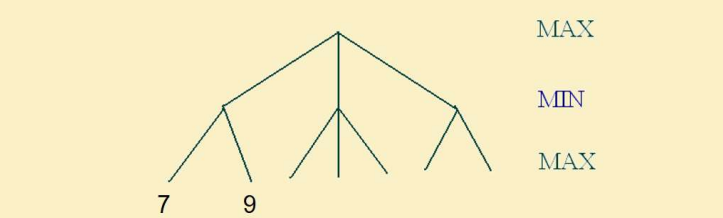
•With cuts or pruning we eliminate large parts of the tree and evaluate only until we can step forward.

Cut example



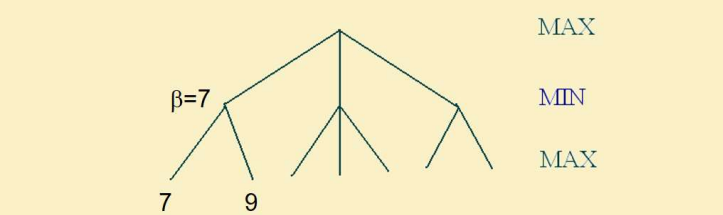
α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

Cut example



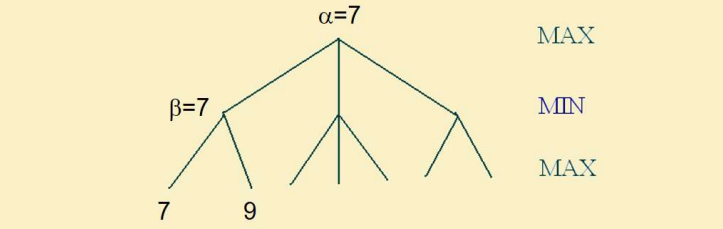
α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

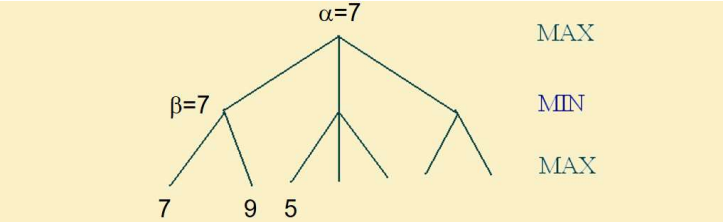
Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

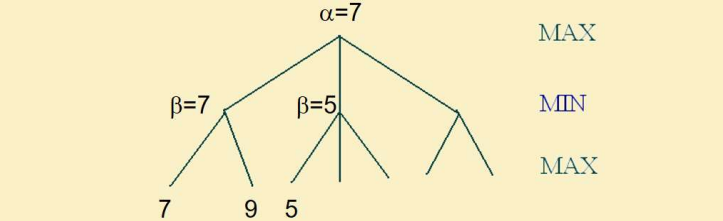
We do not know, but root α=7 and it cannot be smaller.

Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

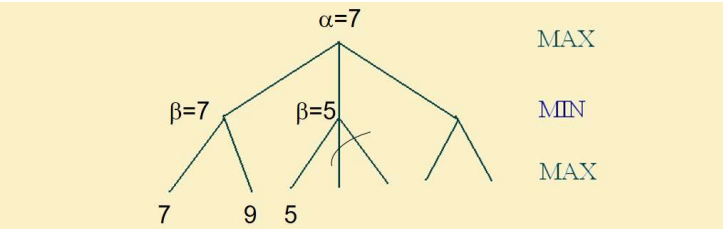
Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

?

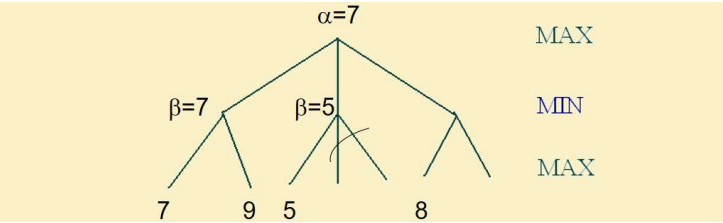
Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

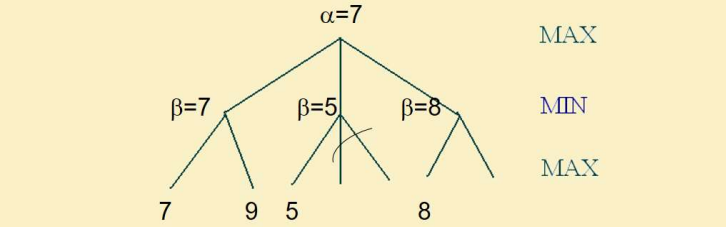
No matter what are the two other numbers, their min ≤ 5. So we can cut, since root α=7 and it cannot be smaller.

Cut example



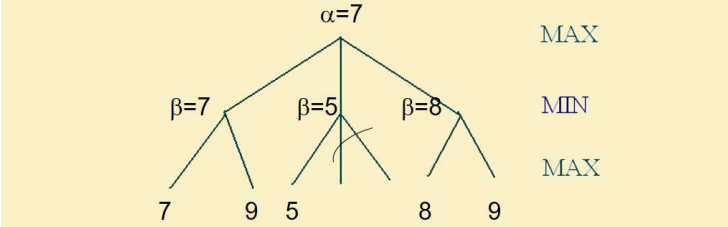
α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

Cut example



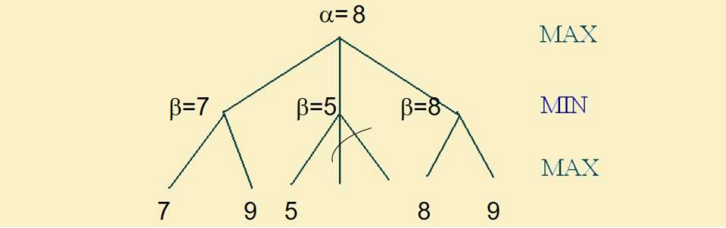
α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

Cut example



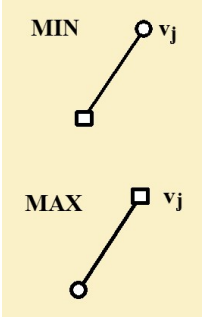
α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

Cut example



α : Lower bound on MAX.
β : Upper bound on MIN.
Cut if α ≥ β

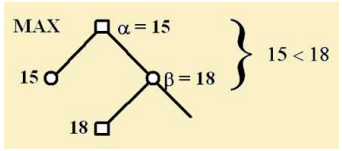
In general



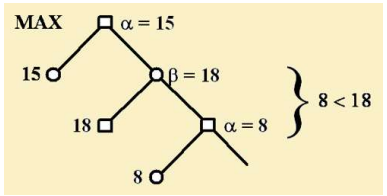
- v_j can only decrease
- β – upper bound – the value of the best step.

- v_j can only increase
- α – lower bound – the value of the best step.

Examples

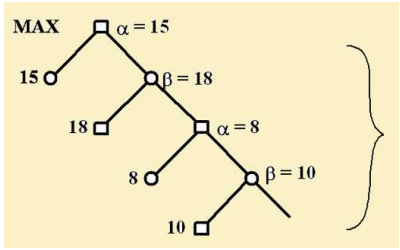


No cut



No cut

Examples



Cut

Remarks on cuts

- Recognize when a position can never be chosen in minimax no matter what its children are
- Max (3, Min(2,x,y) …) is always ≥ 3
- Min (2, Max(3,x,y) …) is always ≤ 2
- Even when x and y is not known.
- Because of alpha and beta, it is often called α-β pruning.

Alpha-beta algorithm

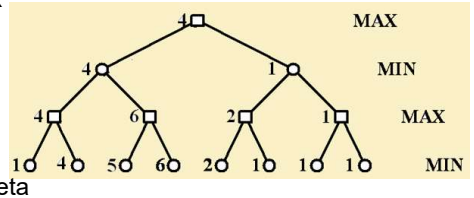
```
function ALPHA-BETA-SEARCH(state, game) returns an action
    action, state ← the a, s in SUCCESSORS[game](s_state)
                such that MIN-VALUE(s, game, −∞, +∞) is maximized
    return action
```

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
    if CUTOFF-TEST(s_state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        α ← max(α, MIN-VALUE(s, game, α, β))
        if α ≥ β then return β
    return α
```

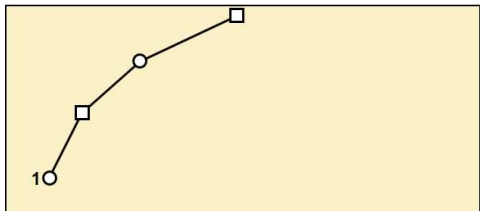
```
function MIN-VALUE(state, game, α, β) returns the minimax value of state
    if CUTOFF-TEST(s_state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        β ← min(β, MAX-VALUE(s, game, α, β))
        if β ≤ α then return α
    return β
```

Example

Minimax

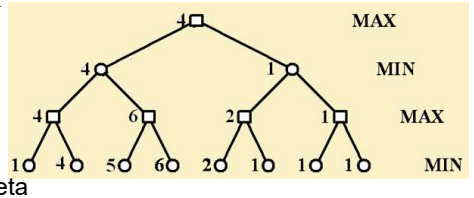


Alpha-beta pruning

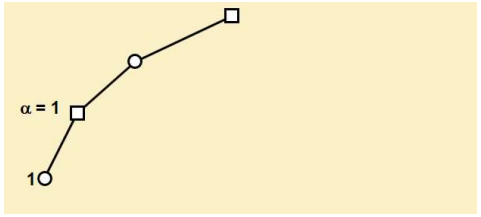


Example

Minimax

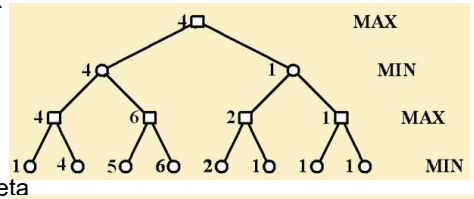


Alpha-beta pruning

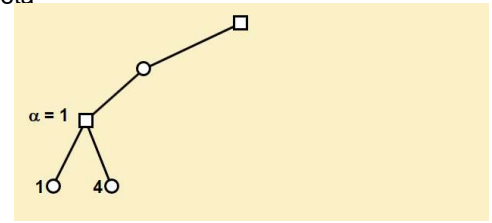


Example

Minimax

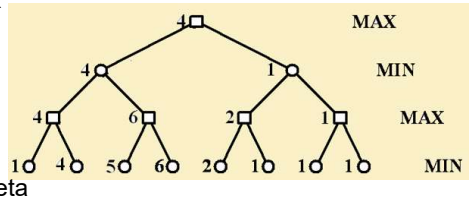


Alpha-beta pruning

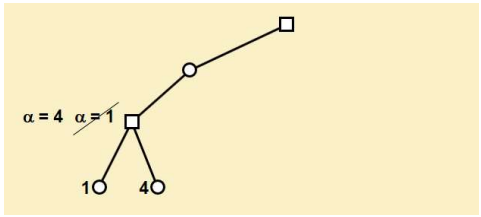


Example

Minimax

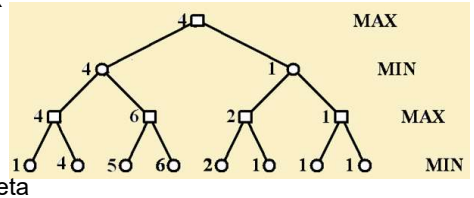


Alpha-beta pruning

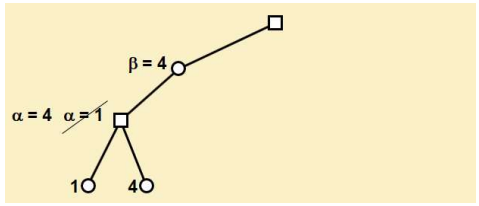


Example

Minimax

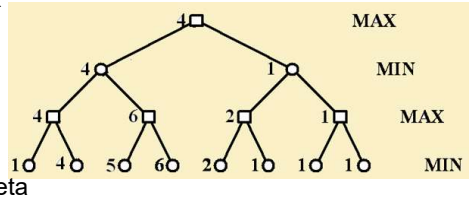


Alpha-beta pruning

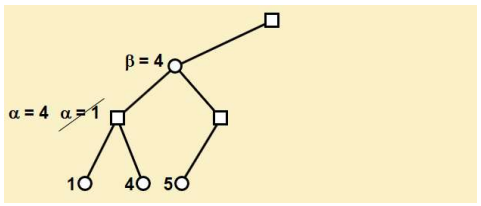


Example

Minimax

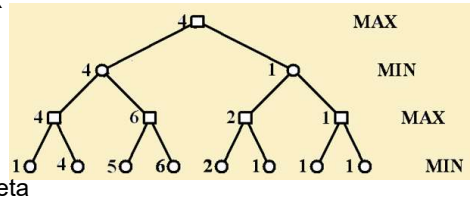


Alpha-beta pruning

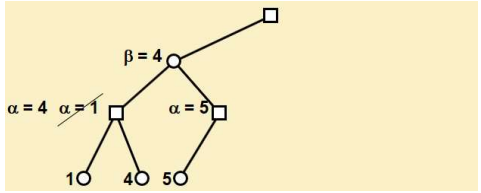


Example

Minimax

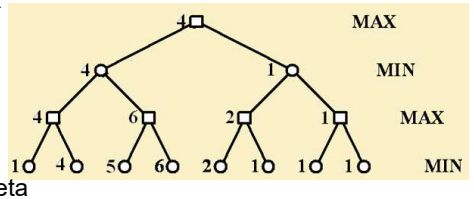


Alpha-beta pruning

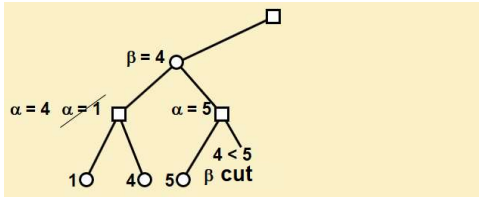


Example

Minimax

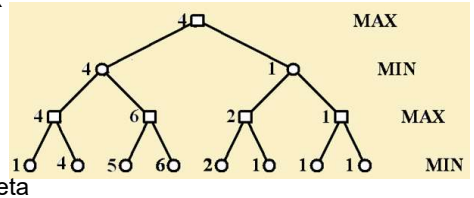


Alpha-beta pruning

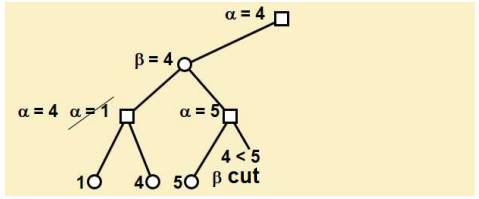


Example

Minimax

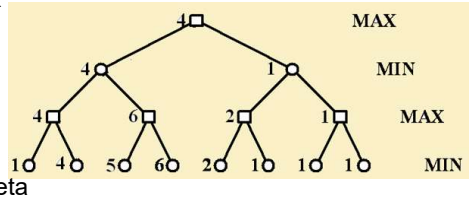


Alpha-beta pruning

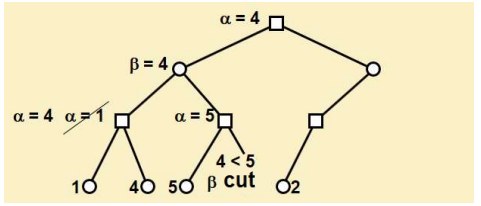


Example

Minimax

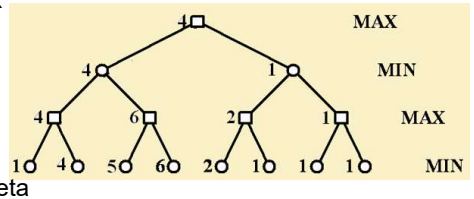


Alpha-beta pruning

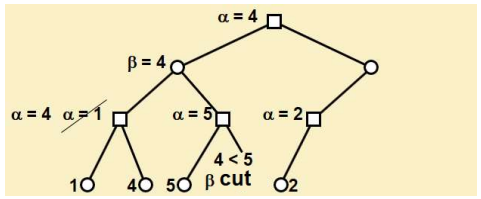


Example

Minimax

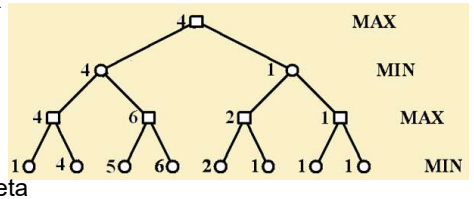


Alpha-beta pruning

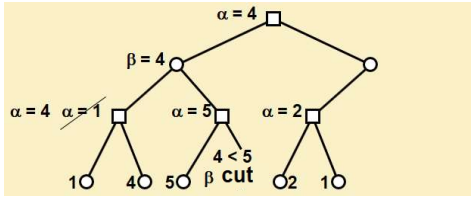


Example

Minimax

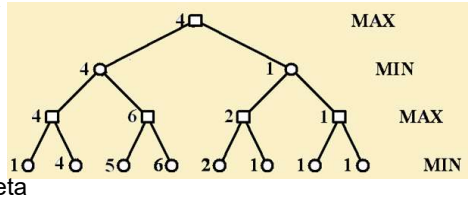


Alpha-beta pruning

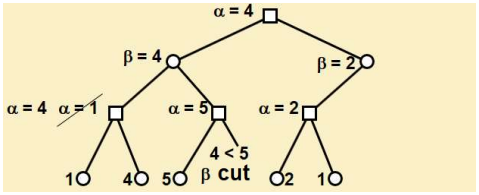


Example

Minimax

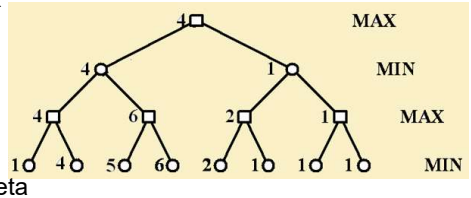


Alpha-beta pruning

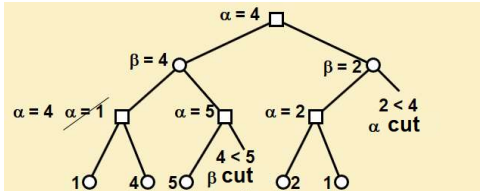


Example

Minimax

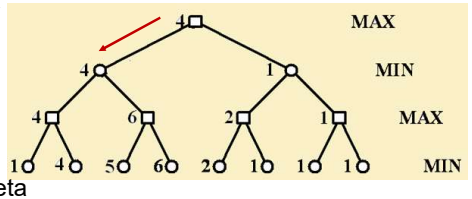


Alpha-beta pruning

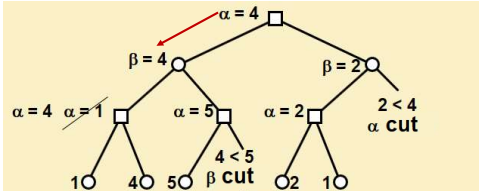


Example

Minimax



Alpha-beta pruning



Remarks on cuts

- Cuts do not effect the final result.
- Effectiveness of cuts depends on the order of the successors.
- In case the best successor is evaluated first, our search is much faster (can go twice as deep within the same time).

Good steps are ordered.

- Another type of heuristics: the potentially better parts are evaluated deeper.
- Another type of heuristics: selection of the important from the irrelevant parts and only the important subtree is considered.

Evolutionary Algorithms

Theory of evolution

- „Evolution, theory in biology postulating that the various types of plants, animals, and other living things on Earth have their origin in other preexisting types and that the distinguishable differences are due to modifications in successive generations. The theory of evolution is one of the fundamental keystones of modern biological theory.” (Encyclopaedia Britannica)

https://www.britannica.com/science/evolution-scientific-theory

Theory of evolution #2

R. Dawkins “The Selfish Gene” (1976, 2006):

- After the discovery of DNA the mechanism of species evolution can be explained in terms of species and genes.
- Species are encoded by their genes and genes are the ones who fight for surviving in the gene pool by means of associating to other genes to develop succesful survival machines.

Evolutionary algorithms

- Problem solving computer systems, with the focus on the evolutionary mechanism.
- More precisely: evolutionary algorithms are optimization algorithms using heuristics and based on population mechanisms. They operate as in real life, they are simplified models of the biological evolution, reproduction, mutation and selection.

Biology and Optimisation

- Environment ↔ Objective function.
- Individual ↔ Set of Parameters.
- Generation ↔ Set of Solutions.

General steps

- Step 1: Generate the initial population (randomly).
- Step 2: Evaluate the fitness function of each individual in the population.
- Step 3: Repeat the followings until termination:
  - Select the best-fit individuals for reproduction.
  - Breed the new individuals through crossover and mutation.
  - Evaluate the individual fitness function of the new individuals.
  - Replace the least-fit population with the new individuals.

4 main groups of EA

- Genetic algorithms
- Genetic programming
- Evolutionary programming
- Evolutionary strategy

Genetic algorithms

GA is an optimisation procedure which

- Follows more states instead of one; and
- The next state is reached not with the modification of one state but with the combination of the two parents.

GA: individuals

- Individuals have sequences (genes) describing the internal and external states.
- Individuals will carry on the genes (or only some of them) through the inheritance/ reproduction process.
- Traditionally, these are represented by binary strings of 0s and 1s, nevertheless, those that reflect something about the problem being solved are more reasonable.
- Sometimes the order of the genes can be important.
- In many cases, many possible individuals do not code for feasible solutions.

TSP: individual coding example

- String of bits:
  Cities are binary coded.
  Individuals are string of bits.
  Most of the individuals represent illegal tours and several represent the same tour.
- String of integers:
  Cities are numbered.
  Individuals are string of integers.
  Same problems.

GA: natural selection

Natural selection is prevailed:

- Children of individuals with „good" genes are more likely to have „good" genes. In other words, good will breed good.
- This will bring a greater part of the population to be good, while the part of the weaker (less effective) individuals is decreasing, since they are deleted from the population.
- NB: randomness does not mean directionless.
- Thus, the system evolves towards a solution.

GA

D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning (1989):

- „Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance."

Genetic algorithms

Implementation is usually via simulation programs.

- Elements of the search space: individuals of the population.
- Fitness function = objective function.
- Operation:
  - New individuals are produced with the appropriate operators.
  - Individuals with worse value of the corresponding fitness function are identified and eliminated from the population.
- In certain cases this type of algorithms may result in a near optimal solution.

GA: main parts

- Initialization. The first population is usually randomly generated. Usually from a couple of hundreds to a couple of thousands individuals are generated.
- Breeding. New individuals may be generated by crossover (recombination) and/or mutation. These genetic operators are usually randomly applied.
- Selection. The selection may be deterministic or stochastic.
  - In case of deterministic selection individuals are selected only if their fitness function value is above a given threshold value.
  - In case of stochastic selection, some individuals may also remain in the next population where this value is worse. With stochastic selection convergence towards a local optimum may be avoided.

GA: main parts #2

- Termination. The generational process is repeated until a termination criteria is reached.
- Common termination criteria are as follows:
  - A given number of generations is reached.
  - The fitness value of the best individual cannot be improved.
  - A solution is found.

Crossover

- One-point crossover: select one random point.
- Two-point crossover: select two random points.
- Unifrom crossover: generate mask.
- Arithmetic crossover.

Crossover problems

- Depending on coding simple crossovers can have a high chance to produce illegal children.
- Uniformed crossover can often be modified to avoid this problem:
  Where mask is I, copy bits of one parent;
  Where mask is 0, choose the remaining bits in the order of the other parent.
- Mutation allows to explore areas not explored by crossover.
- Fight: the parent goes to the next generation directly.

Problems with the fitness functions

Premature convergence

- The fitness range is too large.
- Relatively superfit indiviuals dominate the population.
- The population converges to a local maximum.

Slow finishing

- The fitness range is too small.
- No selection pressure.
- After many generations, average fitness has converged, but no global maximum is found. There is not sufficient difference between best and average fitness.

Solutions to the fitness functions' problems

- Tournament selection: fitness remapping.
- Adjust fittness: scaling. The fitness values are scaled, so that worst value is close to 0 and best value is close to a certain value typically 2; i.e. chance for the most fit individual is twice the average.
- Adjust fittness: when the original maximum and/or the original minimum is very extreme, a use of maximum and minimum value of the fittness may be useful.
- Adjust fittness: individuals are ordered; and the finttnes can be adjusted according to this order.

Schema theorem

- The theorem was proposed by John Holland in the 1970s.
- The Schema Theorem says that short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations.

- Optimality can be guaranteed.

Genetic programming

- Genetic programming is similar to genetic algorithms.
- In this case the mutated and combinated representations are not binary strings but programs, most of the time lisp programs; and these programs are optimized.
- Genetic programming often uses tree-based internal data structures to represent the computer programs for adaptation.

Evulotionary programming and strategy

- Evolutionary programming (EP) and evolutionary strategies (ES) are based on the same idea, yet their details differ.
- EP involves populations with primarily mutation and selection and arbitrary representations. EP uses self-adaptation to adjust parameters, and can include other variation operations, for example combining information from multiple parents.
- ES breeds individuals by mutation and recombination. ES algorithms are designed particularly to solve problems in the real-value domain (not only binary strings). They use self-adaptation to adjust control parameters of the search.

Other types of EA

- Swarm optimization.
- Particle swarm optimization.
- Ant colony optimization.
- Bees algorithm.
- Artificial bee colony algorithm.
- Simulated annealing.
- Harmony search.
- Neuroevolution.
- Etc.

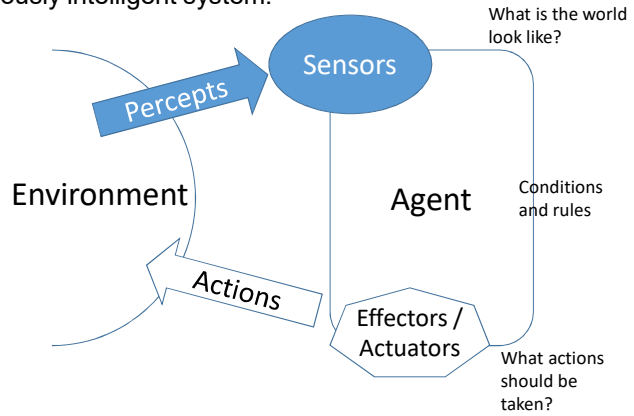
Agents

Agents

- The origin/root of the word is ago, agere, agentis.
- The meaning: to "do" or "make" or "the one doing."

Agent

- Continuously intelligent system.



Agents

- An agent is an autonomous entity which percepts with the help of sensors and acts upon an environment using actuators and directs its activity towards achieving some goals.
- Agents may also learn or use special knowledge.
- Agents may be simple or very complex.
- For example, bot programs used for data mining are also called intelligent agents.

Agents #2

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.

(Russell and Norvig 1995, page 33)

NB. This definition depends heavily on what we take as the environment, and on what sensing and acting mean.

Agents #3

Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

(Maes, Pattie (1990) ed., Designing Autonomous Agents, Cambridge, MA: MIT Press, pp. 108.)

NB. MIT's Media Lab's agents must act autonomously to realize a set of goals, while environments are restricted to being complex and dynamic.

Agents #4

Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.

(Hayes-Roth, B. (1995). "An Architecture for Adaptive Intelligent Systems," *Artificial Intelligence:* Special Issue on Agents and Interactivity, 72, 329-365.)

NB. Stanford's Knowledge Systems Laboratory states that agents reason during the process of action selection.

Agents #5

Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires.

(http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm.)

NB. IBM's Intelligent Agent Strategy white paper, views an intelligent agent as acting for another, with authority granted by the other.

Agents #6

An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

(Stan Franklin and Art Graesser: Is it an Agent or just a Program?, 1996.)

Agent example

• Is a spell checker an agent or not?

?

Agent example

• Is a spell checker an agent or not?

A spell checker is a simple software program part in case it checks the fully typed words whether they are correct or not.
A spell checker can be an agent in case it checks and corrects while typing.

Agent example #2

• Is a thermostat an agent?

?

Agent example #3

A payroll program in a real world environment could be said to sense the world via it's input and act on it via its output, but is not an agent because its output would not normally effect what it senses later. A payroll program also fails the "over time" test of temporal continuity. It runs once and then goes into a coma, waiting to be called again. Most ordinary programs are ruled out by one or both of these conditions, regardless of how we stretch to define a suitable environment.

Agents and software

- All software agents are programs, but not all programs are agents.

Agent: weak definition

An agent is a hardware or software based system that has the following properties:

- Embedded: its states are cannot be interpreted outside the environment.
- Reactivity: it percepts its environment and reacts in real time to the changes of the states.
- Autonomy: it operates and acts autonomously, without human and/or other machine direct intervention; it has a defined control over its own actions and internal states.
- Dependency: an agent belongs to the situation and to the role/ function.

Agent: other properties #1

- Initiator: it not only reacts to the changes within the environment but may also initiate; in other words sometimes pro-active.
- Target oriented behaviour: it has direct or indirect target(s) and it tries to reach its goals.
- Cooperative behaviour: it may cooperate with other agents to achieve its goals. Other names: communicative, socially able.
- Moving ability: it may move from its physical location.
- Truthful: it does not give false information consciously.
- Good will: it does not have conflicting targets with other agents.
- Reliability, consistency: an action performed will be performed similarly in case of similar circumstances.

Agent: other properties #2

- Autonomous: it excersises control over its own actions.
- Temporally continuous.
- Adaptive. It may learn to change its behaviour based on experience.
- Flexible: not all actions are prescribed.
- Please note that not all agents have all properties as above.

Agent: other properties #3

- Successful. What does it mean to be successful in a given environment? Please note that it is necessary to have an objective performance indicator given by an external observer.
- Rationality. It is assumed that the agent will not act consciously against its goals. The goal is a desired state of the environment. The agent tries to choose the best alternative always.

Agent's property: rationality

- Perfect rationality means to act correctly and properly. It is impossible in a complex environment, the computational needs are too big.
- Restricted rationality means to act correctly and properly while every needed computational task cannot be done. In other words the agent chooses the best alternative with the help of the available computational capacity and other resources.
- Remark: an IT system is called to be intelligent if it has the highest performance of all IT systems having the same computational limits.

Agent: strong definition

- Agents having the properties according to the weak definition plus they are rational, and they use both formally and in an implemented way concepts and notions that are used for humans.
- In this case the internal structure of the agent corresponds to some state variables (for example cooperation ability, believable personality, mobility etc.). These state variables characterise the agent in a specific moment and give its mental state as well.

Agent example

Am I an agent?

?

Agents

- Communication is based on the purposefully reduced human language.
- Agents may also be supplied by personality traits. These special signs may distinguish the agent from other agents, for example, name, unique behaviour. Agents may also be supplied by emotions, for example, happy, sad.

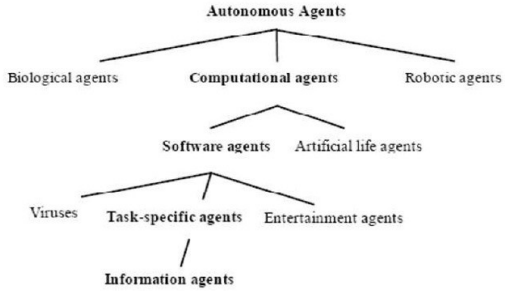
- Agent = architecture + program.
- Task: to design and implement an effective agent architecture and a program.

Agent classification

- A multi layer taxonomy of software agents may begin with a three-way classification into regulation agents, planning agents, or adaptive agents.
- A regulation agent, reacts to each sensory input as it comes in, and always knows what to do (thermostat). It neither plans nor learns.
- Planning agents plan, either in the usual AI sense (problem solving agent), or using operations research based methods (OR agents), etc.
- Adaptive agents not only plan, but learn as well; i.e. there are adaptive problem solving agents for example.

Brustoloni, Jose C. (1991), "Autonomous Agents: Characterization and Requirements," Carnegie Mellon Technical Report CMU-CS-91-204, Pittsburgh: Carnegie Mellon University

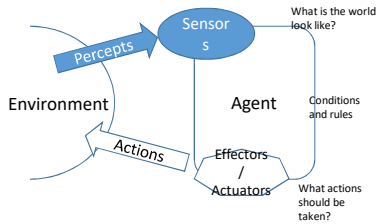
Agent classification #2



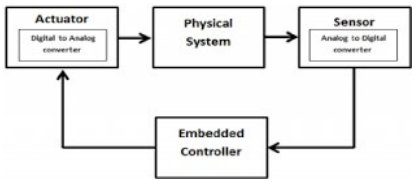
- Human agents: eyes, ears, other organs of perception; hands, feet to intervent.
- Robot agents: cameras, distance sensors, different motors to intervent.
- Software agent: coded bit streams, system calls for both.

Franklin, Graesser, 1996.

Agent



Standard embedded system structure:



Agents

- Learning ability: Successful agents split the task of computing policy into 3: i) initially, some prior knowledge is included; ii) when deciding the next action, the agent does some computation; and iii) the agent learns from its experience to modify its behavior.
- Autonomous agents learn from their experience to compensate partial or incorrect prior knowledge.
- Learning algorithms.
  Adaptation to the environment.

Agents

- Learning algorithms: new information from its environment can be gathered, stored and used. Based on the new information, it can modify its behaviour and plans, for example to achieve the target faster.
- In other words, adaptation to the environment can be achieved based on the new information gathered during learning.
- Please note that adaptation needs an environment.
- The adaptation can be
  -mid-term, i.e. it is governed by learning (this is the most common);
  -long-term, i.e. can pass on its info to the next generation;
  -based on conclusions, i.e. a deducting process is also considered to determine the desired behaviour.

Multi-agent systems

- The agents are in interaction with each other (and with humans). During this interaction the state of the players change.
- To define a multi-agent system the followings have to be given:
  - The set of agents of the system.
  - Type and means of interaction.
  - Structure of the agent system (for example, hierarchy).

Interatcions

- Direct in terms of physical contact.
- Direct: agents change information via a communication language.
- Indirect: agents act within the same environment and interact via their acts and purposes.

Coordination

Coordination of the actions to solve the task of the system may differ widely. Some examples:

- Structural coordination is based on a pre defined organizational structure. The master agent distributes the resources. It is centralized.
- The distribution of the task is based on bidding.
- Agreement (consensus) protocol is a dynamically changing system by which a collection of interacting agents achieve a common goal.

Example

Agent: Interactive AI tutor.

- Performance measure: maximize test points.
- Environment: set of students.
- Actuators: screen display (exercises, suggestions, corrections).
- Sensors: keyboard.
- Remark: the difficulty of the exercises are adjusted to the knowledge level of the student and it is continuously raised.

Neural Nets

Soft computing

- When designing computing models with the use of inexact solutions based on human experience, it leads into the world of soft computing.
- Soft computing differs from conventional „hard” computing, since it is tolerant of imprecision, uncertainty, partial truth, and approximation.

Soft computing

- In general, soft computing deals with more complex problems, that cannot be solved, handled or analysed globally with conventional methods.
- The classical AI algorithms refer to NP complete problems that requires enormous time and computational power.
- The aim of soft computing is to give a solution for the above problems where there are no known algorithms, even though the solution may not be precise.

Zadeh

- Soft Computing is a formal area of study in Computer Science since Lotfi A. Zadeh, 1994 (Zadeh, Lotfi A., "Fuzzy Logic, Neural Networks, and Soft Computing," Communications of the ACM, March 1994, Vol. 37 No. 3, pages 77-84.):
- „…the most important factor is the use of what might be referred to as soft computing – and, in particular, fuzzy logic – to mimic the ability of the human mind to effectively employ models of reasoning that are approximate rather than exact.”

Zadeh's example

- Problem of parking an automobile.
- Most people are able to park a car quite easily, „because the final position of the vehicle and its orientation are not specified precisely. If they were, the difficulty of parking would grow geometrically with the increase in precision and eventually would become unmanagable for humans."
- It is important to note that it is easy in case it is formulated imprecisely, and impossible to solve in case precisely, since traditional methods cannot exploit the tolerance of imprecision.

Dilemma

Incompatibility principle

- As the complexity of a system increases, our ability to give significant and precise statements about its behaviour decreases until a barrier, beyond which precision and significance are already mutually exclusive characteristics.

Stability-plasticity dielamma

- How can a learning system be designed that it is plastic enough to be able to learn new things but it is stable enough to keep its knowledge?

Analogy

- Soft computing tries to develop useful mathematical models to solve problems of complex systems in the fields of biology, medicine, human sciences, economics etc.
- Humans can interpret distorted speach, decipher sloppy handwriting, summarise text, recognise and classify images, drive vehicles in dense traffic etc.

In other worlds make rational decisions in an environment of uncertainity and imprecision.

A suitable model is available.

Analogy #2

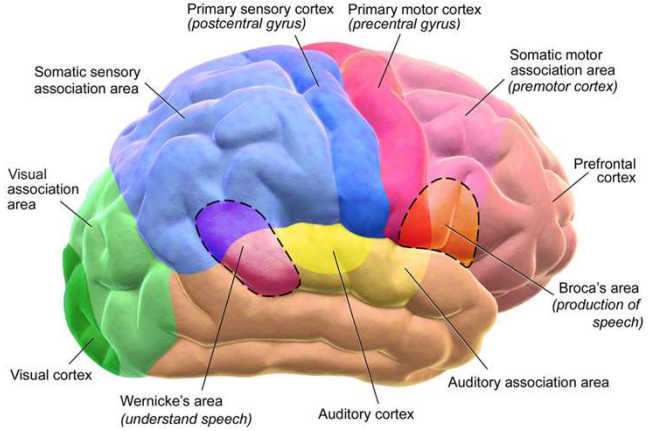
- Let us copy the „human IT" software and hardware:
- „Hardware:" system of brain cells; with the most important features as parallel operation and learning ability – its copy: neural nets.
- „Software:" everyday ideas, communication, inference, etc. – its copy: fuzzy systems.

Brain: the evolutionary miracle

- Sir John C. Eccles believed that understanding the brain is the “ultimate problem confronting man. In terms of its complexity, the problem is much bigger than the whole problem of cosmology.”
- Australian research physiologist who received the 1963 Nobel Prize (with Alan Hodgkin and Andrew Huxley).

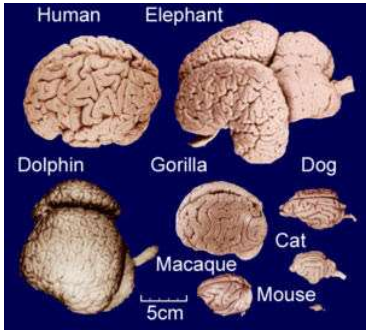
Robinson, D., John Carew Eccles; in: *100 Most Important People in the World Today*, Putnam, New York, p 237–240, 1970.

Functional areas of the human brain



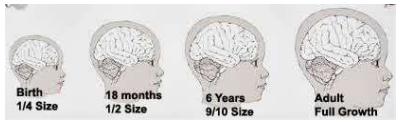
https://en.wikipedia.org/wiki/Human_brain#/media/File:Blausen_0102_Brain_Motor%26Sensory_(flipped).png

Animal brain



http://www.neuroscientia.com/2017/05/differences-between-human-brain-and.html

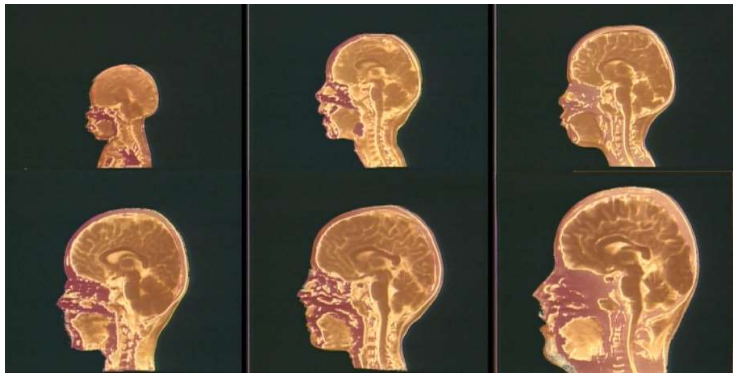
Child brain



The brain experiences a growth spurt between ages of 5 and 7, especially in the frontal lobes. This part of the brain has an important role in planning and in the sequential organisation of actions and thoughts (Harris & Butterworth, 2012) (Nelson, Thomas & de Haan, 2006).

https://childrenandtheinternet.wikispaces.com/School+Age+Child+Development

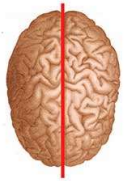
Brains



2 months, 10 months, 3,5 years, 7 years, 10 years and 14 years old.

Brain's hemispheres

- The brain has two hemispheres. Broad generalizations are often made in popular psychology about certain functions (e.g. logic, creativity) being lateralized, that is, located in the right or left side of the brain. These claims are often inaccurate, as most brain functions are actually distributed across both hemispheres.
- Left: speach, language, reading and writing, analysis, logic, maths: algebra, controls the events and the right hemisphere.
- Right: maths: geometry, personality, creativity.

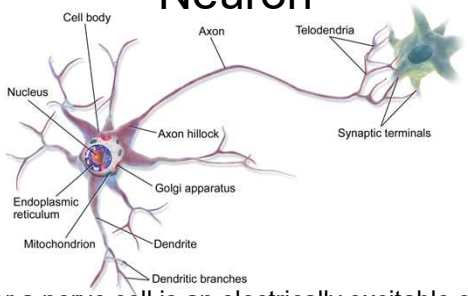


Neurons in the brain

- The adult human brain weights on average about 1.2–1.4 kg.
- The adult human brain is estimated to contain 86±8 billion neurons, with a roughly equal number (85±10 billion) of non-neuronal cells.
- Out of these neurons, 16 billion (19%) are located in the cerebral cortex, and 69 billion (80%) are in the cerebellum.

Azevedo et al., J. Comp. Neurol. 513:532–541, 2009, DOI: 10.1002/cne.21974

Neuron

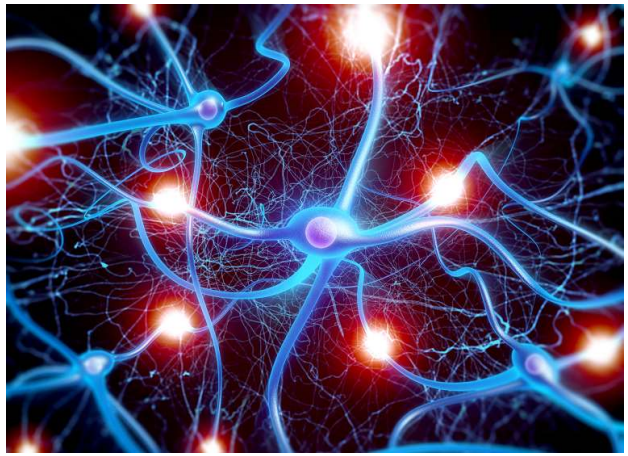


- A neuron or a nerve cell is an electrically excitable cell that receives, processes, and transmits information through electrical and chemical signals.
- One axon and numerous dendrites belong to the soma. Dendrites typically branch profusely, getting thinner with each branching, while axons maintain their diameter.

Neuron #2

- These electrical and chemical signals between neurons occur via specialized connections called synapses.
- Synaptic signals from other neurons are received by the soma and dendrites; signals to other neurons are transmitted by the axon.
- Signals are carried by neurotransmitters.
- Neurotransmitters let some ions pass. The ion flow results in a voltage change, which increases the original resting tension of the neuron. Should this increase be large enough, the neuron reaches its action potential.
- The fundamental process that triggers the release of neurotransmitters is the action potential, a propagating electrical signal

Mouse neuron



https://hms.harvard.edu/news/neurons-hardy-bunch

Neural network of insects

- An insect brain contains about 100000 to 1000000 neurons.
- Inheritance is dominant.
- The spinal nervous system of an insect is more „rigid” yet it also can learn. For example a bee can fly immediately at birth, while birds have to learn how to fly.
- Neural plasticity is the capability for the nervous system to adapt itself to variations in the environment.

Motivation

The reasons for the interest to design an artificial neuron are as follows:

- Brain performs some computations much faster than Neumann-based computers.
- Because of parallel and associative operations for example human face recognition is better than face recognition by computers.

Expectations

- The basis of the brain work is learning based on experience.

- Artificial neural networks should also be capable of learning.

- Human brains consist of connected neurons.

- Artificial neural networks should have the same system.

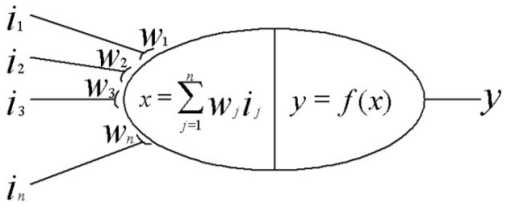
Artificial neural network

- Hardware or software information processing tool, that is capable parallel, distributed operation;
- It is a highly interconnected system of same or similar operating units, neurons with local processing, usually in arranged topology;
- It has a learning algorithm, which means a sample based learning in most cases, and determines the way the information is processed;
- It has an information retrieval algorithm that helps to make use of the learned information.

Artificial neural network #2

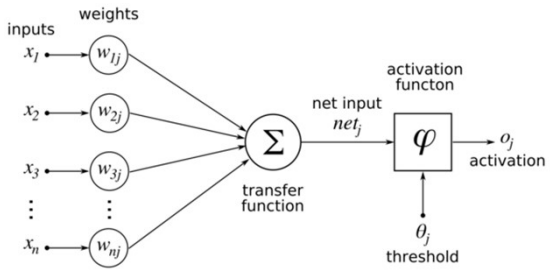
- The collection of connected units provides the computational efficiency, the neurons by themselves do not provide specific information.
- The answers / outputs generated by the neurons fully rely on local information.
- The connection weights contain the knowledge.
- Neural networks have been used to solve computer vision, speech recognition, machine translation, social network filtering problems.

Artificial neuron



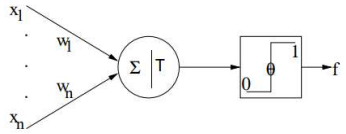
The output is analogous to the axon of a biological neuron, and its value propagates to the input of the next layer, through a synapse. It may also exit the system, possibly as part of an output vector.

Artificial neuron #2



- Usually each input (x_1,…,x_n) is separately weighted.
- It calculates the weighted sum, and passes it through a non-linear function known as the activation function to generate the normalized output.

McCulloch-Pitts (1943)

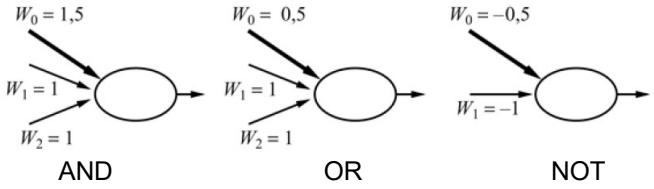


- In case the sum is above the threshold T, then the output is 1, otherwise 0. NB. this can also be represented as an input.
- The function describing the operation:

$$y(k) = f\left(\sum w_i x_i - T\right) = sgn\left(\sum w_i x_i - T\right)$$

- These neurons may perform universal computations (AND, NOT, OR).

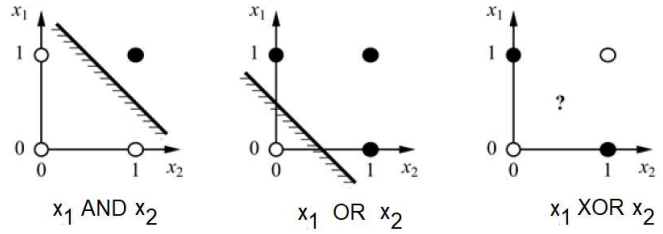
AND – OR – NOT



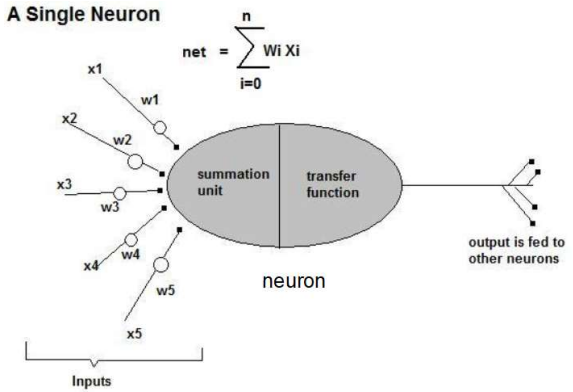
- Based on these units any logic funcition can be calculated with NN.

Early days

- The first practical application was the perceptron (1958).
- In 1969 prof. Minsky showed that it was impossible for these classes of network (single layer perceptrons) to learn XOR function.



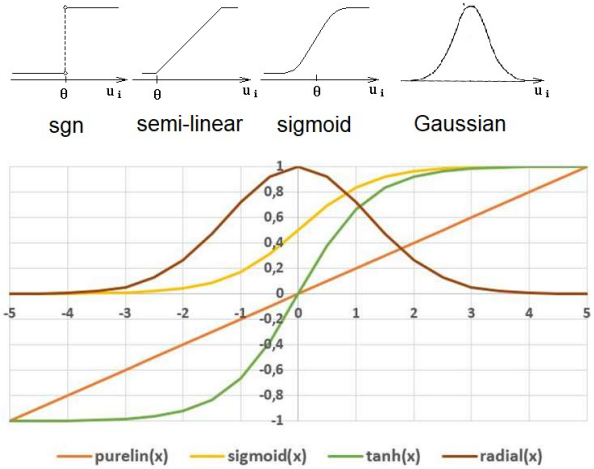
Common representation



Transfer / activation functions

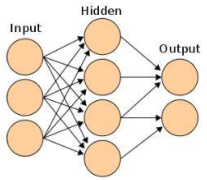
- Typical transfer functions have the following properties.
- The unit should be active (i.e. output 1) in case the inputs are „right" and inactive (i.e. output 0) in case the inputs are „wrong."
- Should the function be linear, the neural net represents a simple linear function.

Common transfer functions



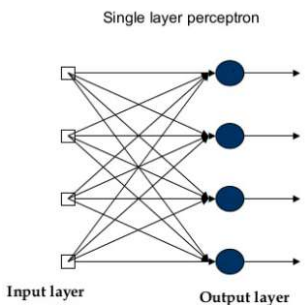
Layers

- Typically neurons are organised in groups, called layers.
- Different layers may perform different kinds of transformations on their inputs.
- The first layer is called an input layer, the last is called an output layer, inbetween hidden layers.
- The structure of the layers together with their connections is a key element of the network design.

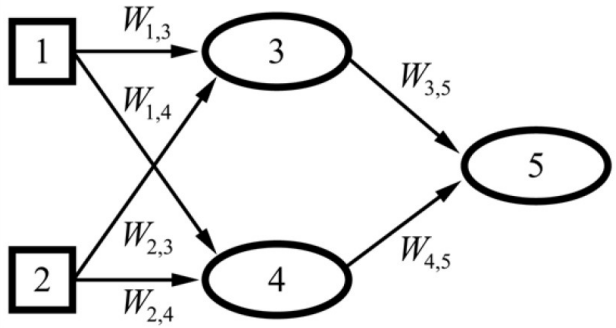


Single layer neural net

- Perceptrons are single layer neural nets where all inputs are directly connected to the outputs.

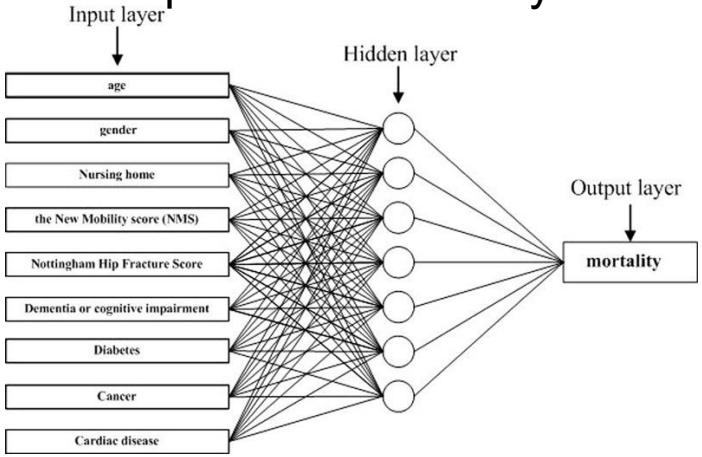


Example with hidden layer



- Simple neural net with 2 inputs, 1 output and 1 hidden layer.

Example with hidden layer #2



http://dx.doi.org/10.1590/1414-431X20132948, Braz J Med Biol Res vol.46 no.11 Ribeirão Preto Nov. 2013 Epub Nov 18, 2013

ANN design

When designing a neural network, the following steps have to be considered:

- Encoding of problem data.
- Development of the structure of the NN.
- Choosing the parameters of the processing units.
- Teaching the NN.
- Use of the NN.

Learning paradigms

There are three main learning paradigms:

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.

Supervised learning

- Here, during training it can be checked that the net generates the desired output or not.
- Supervised learning is the task of inferring a function from labeled training data.
- There exist a set of training examples. For each example the desired output value is also given.
- The algorithm analyzes the training data and produces an inferred function, which can be used for mapping the examples; i.e. the learning algorithm generalize from the training data to unseen examples.
- In the optimal case the algorithm correctly determines the class labels for all examples.

Supervised learning #2

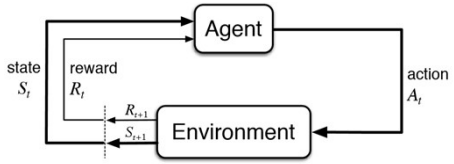
- The training set of examples needs to be representative.
- The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that describe the object.
- The resulting learned function should be measured on a test set that is separate from the training set.

Unsupervised learning

- Here the rules should be recognized within the learning process.
- Unsupervised learning is the task of inferring a function to describe hidden structure from "unlabeled" data, for which classification or categorization is not included in the observations.
- There is no prior evaluation of the accuracy of the structure that is output by the relevant algorithm.

Reinforcement learning

- Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.
- In an essential way these are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them out.



Richard S. Sutton and Andrew G. Barto, 2017

Learning concepts

In each phase the output (or the state of the neuron in the output layer) is evaluated. Based on the experience, the followings can be done.

- Change the weights.
- Set up new or delete old connections.
- Change the value of the activation function.
- Change the functions of the neurons.
- Set up new neurons or new layers.
- Delete old neurons or old layers.

Backpropagation training

- It is used to calculate the error contribution of each neuron after a batch of data is processed in order to adjust the correct weight of each neuron during the learning process.
- In the feedforward network the output is calculated and the result is compared with the desired output.
- Then, based on the above error the weights are updated.
- This cycle is repeated until the network performs properly.
- This method is commonly used in the gradient descent optimization algorithm; and to train deep neural networks.

Advantages of ANNs

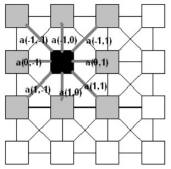
- Learning ability: based on data NN can create new connections, in other words it can learn.
- Only few assumptions are made.
- It can generalize, in other words it can operate with data which are just similar to the training examples.
- It can work with uncertain and inaccurate data.
- It is nonlinear, i.e. the output depends on other inputs as well, thus it may be used for complex systems.
- It is highly parallel, thus many operations can be performed simultaneously resulting in high speed.

Disadvantages of ANNs

- Black box nature, i.e. it is not possible to explain how the results were calculated in any meaningful way.
- The architecture of a neural network is different from the architecture of microprocessors therefore it needs to be emulated.
- There are many parameters to be set and optimizing the network can be difficult.
- Requires high processing time for large neural networks.

CNN

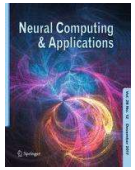
- CNN computers are cellular wave computers in which the core is a cellular neural/nonlinear network (CNN), an array of analog dynamic processors or cells.
- Similar to neural networks, with the difference that communication is allowed between neighbouring units only.
- Key featues: parallel-signal processing and continuous-time dynamics, allowing real-time signal processing.



- CNN host processors are accepting and generating analog signals, the time is continuous and the interaction values are also real numbers.
- L. O. Chua and Tamás Roska; since 1988.

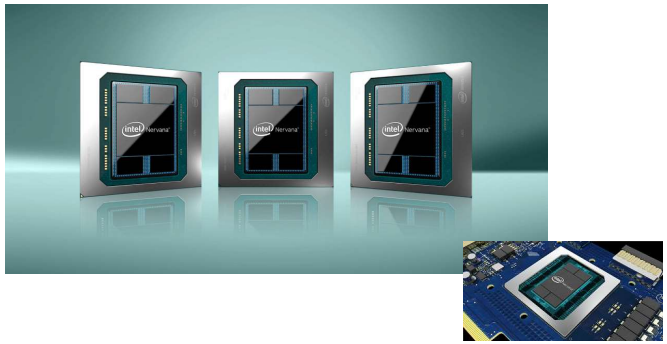
Research

- Moving object recognition using multi-view three-dimensional convolutional neural networks
- High-order fuzzy-neuro-entropy integration-based expert system for time series forecasting
- Deep learning in vision-based static hand gesture recognition
- Performance Analysis of Cellular Radio System Using Artificial Neural Networks
- A Load Balancing Optimization Algorithm for Context-Aware Wireless Sensor Networks Based on Fuzzy Neural Networks



Example

- Intel Nervana
  Neural Network Processor – new chip to revolutionize artificial intelligence.



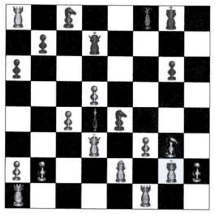
Knowledge Representation

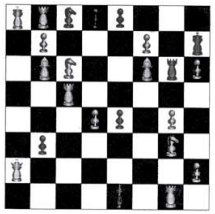
Thinking and knowledge

- In general: How does human brain store information and knowledge? And how can it be brought to the surface from there?
- Important questions are as follows:
  - What is knowledge?
  - How can knowledge be represented?
  - How can knowledge be manipulated?

Rational thinking

- In general: To what extent do we apply „rational" strategies?
- What is rational thinking anyway?
- For example, how does a professional chess player think?





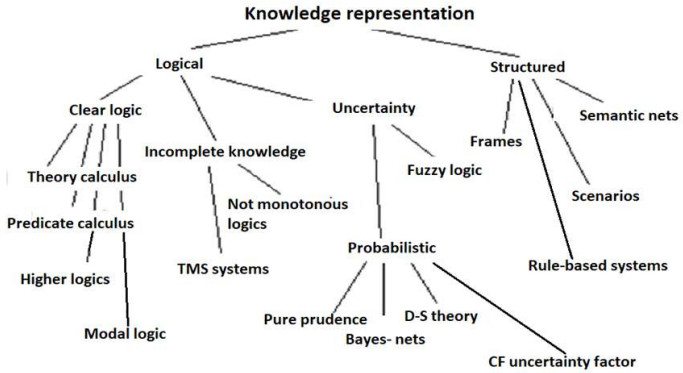
Knowledge representation

- The basis of human problem solving is the knowledge one has about the given problem or area of expertise.
- Usually it consists of several elements, i) theories, ii) heuristic, and iii) skills and experience.
- Knowledge representation is the depiction of some knowledge in a structure that facilitates the computer solution of the tasks arising in the area. Therefore, it is a technique that makes possible the description of expertise in a knowledge base.

Qualities of knowledge representation

- Can be applied widely;
- Complete (includes all extrapolatable facts);
- Describes a „closed world” (what is not true is false);
- Flexible and expandable;
- Consistent;
- Manageable;
- Integrable;
- Economical;
- It has an effective extrapolation;
- Effective „knowledge acquisition” methods;
- Etc.

Knowledge representation



Rule-Based Systems

Rule-based systems

- Components: condition-action pairs (rules), that is:
  IF <condition> THEN <consequence>
  formed „chunks of knowledge”
- The <condition or antecedent> is an assertion determining the conditions of the rule to be applied, or a complex expression formed by AND/OR conjunctions of conditions.
- The <consequence> describes one or more consequnces of applied rule; „actions”, operations, activities to be carried out, valid assertions.

Rule-based systems: example

- IF the ‘traffic light’ is ‘green’
  THEN the action is go
- IF the ‘traffic light’ is ‘red’
  THEN the action is stop
- IF <*antecedent*> *THEN*<*consequence*>

Conclusion methods

Moving forward:

- Starting from the facts, we deduce intermediate assertions using the rules, then from these, we deduce new assertions, until we get to the goal.

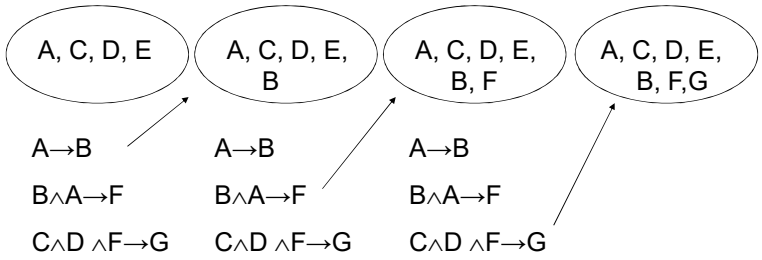
Moving backwards:

- Using the rules backwards, we assign partial goals that are suffiecient for the goal, then we assign new partial goals as condition of these, etc. until we get to partial goals that are supported by facts.

Forward-chaining

- Also called deductive sytem:
- If A is a fact and the A → B rule is true, then B is also true.
- Forward chaining is a data-controlled extrapolation. It is good if the system first collects data, then it attempts to extrapolate the most possible data from these.
- Let us use this method, if we possess a large amount of data collected about the problem to be solved, but we do not have a good notion about the solution. The rule-based system will perform every feasible conclusion and will present every possible solution from the given facts.

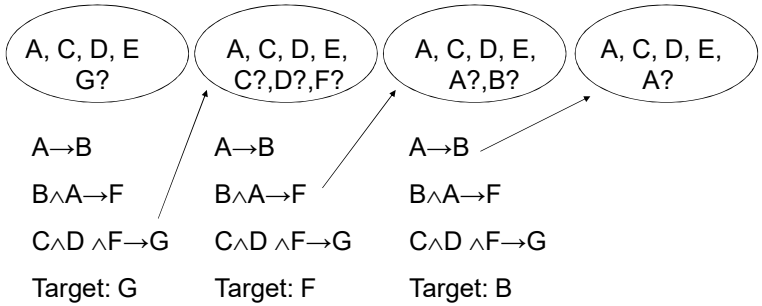
Forward-chaining example



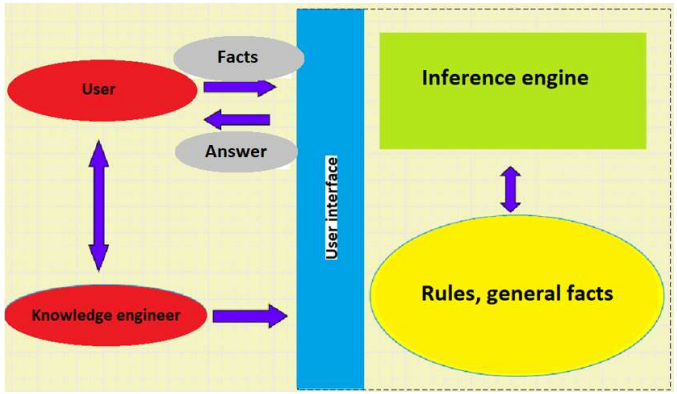
Backward-chaining

- Backward chaining is a goal-oriented extrapolation.
- If the A → B rule is true, then to satisfy B it is enough to prove A.
- It is good if the system starts from an initial hypothetical theory and attempts to search for the facts that prove it.

Backward-chaining example



Rule-based system



Inference engine

- The inference engine is the active component of the system. It performs extrapolations. It includes
- The control strategy, which determines the order in which the rules will be attached to the knowledge base, and in case several rules can be attached at the same time, it also fixes the conflict solving strategy.
- A one rule applier.
- Explanation generator.

Advantages of rule-based systems

- Modularity: every rule can be handled independently from the other rules.
- Universal display: we phrase all information as rules.
- Naturalness: we often express ourselves in a similar way in our daily lives as well.
- Support of insecurity treatment: the rules can be easily supplemented with insecurity treatment options.

Cons of rule-based systems

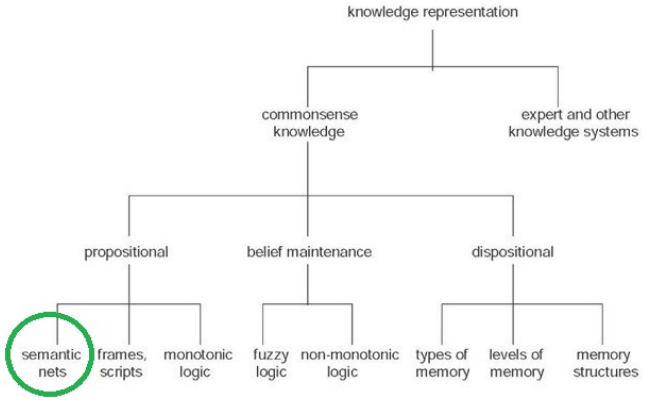
- Infinite chaining: one can easily create rules and get into a circle, generating an infinite chain of rules.
- Building in new information that contradicts previous ones: there is no general method for the verification of possible contradictions, therefore, adding a new rule or modifying an existing one can easily result in a group of contradictory rules.
- The language of the rules is not standardized, it can vary widely with various implementations. As a result, the transfer of the rule bases into another system is often difficult.

Application fields

- In the case of distributed knowledge. In this case the proportion of facts compared to the rules is significant. For example, clinical healthcare systems.
- In cases where the demonstrated knowledge and control systems can be separated. For example, biological classification.
- In cases of independent activities. For example, patient monitoring systems.

Semantic Networks

Knowledge representation #2



http://chrishutchison.org/atticaschool/semnet/page2/page2.html

What is a semantic network?

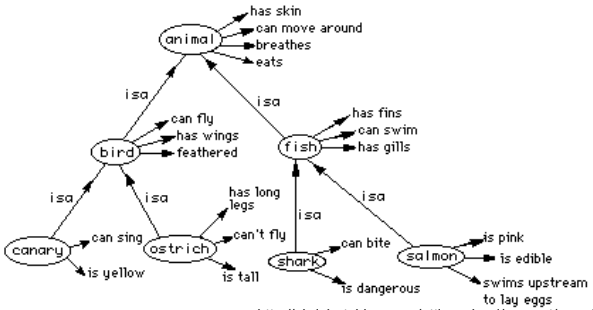
- A semantic network is a structure for representing knowledge as a pattern of interconnected nodes and arcs.
- It is also called associative network.
- Semantic Nets were first invented for computers by Richard H. Richens of Cambridge Language research Unit in 1956 for machine translation of natural languages.
- It is the modelling of human information storing and searching. (Quillian & Collins, 1969)

Semantic networks

- Labelled directed graphs.
  - Nodes are objects, object classes; and values of properties.
  - Inheritance prevails along the edges.
- „Is a" inheritance between nodes.
- The entity inherits the properties of parent entity.

Quillian & Collins' experiment

- Based on cognitive psyhcological experiments.
- They measured the reaction time of people answering to different questions.



http://chrishutchison.org/atticaschool/semnet/page2/page2.htm

Quillian & Collins' experiment #2

- Explanation:

True sentences	Mean reaction time in msec
A canary is a canary	1.00
A canary is a bird	1.17
A canary is an animal	1.23
A canary can sing	1.3
A canary can fly	1.38
A canary has skin	1.47

NB. skin and sing properties have different distances in the semantic network.

http://chrishutchison.org/atticaschool/semnet/page2/page2.htm

Solving tasks

- Task: asnwer for a query with use of a given subject knowledge.
- Subject knowledge: a taxonomic hierarchy (of a set of nested classes) computer representation. (Database)
- Taxonomy: systematics.
- Query: a target net suiting into the semantic net. (Allignment)

Problems

- Handling exceptions is complicated.
- Specific property may suppresses the inherited property.
- For example, birds fly, but even though ostrich is a bird, it cannot fly.

Semantic web

- Extension of World Wide Web through standards by W3C.
- The term "Semantic Web" was coined by Tim Berners-Lee.
- The standards promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF).
- The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. (W3C)

Semantic web

- Hyperlinked web pages are extended with machine-readable metadata about the pages, enabeling autonomous agents to perform tasks instead of the users.
- Specially designed RDF (Resource Description Framework), OWL (Web Ontology Language), and XML formats.

Web 3.0, 2015

Linked Data

The Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. RDF provides the foundation for publishing and linking your data. Various technologies allow you to embed data in documents (RDFa, GRDDL) or expose what you have in SQL databases, or make it available as RDF files.

Vocabularies

At times it may be important or valuable to organize data. Using OWL (to build vocabularies, or "ontologies") and SKOS (for designing knowledge organization systems) it is possible to enrich data with additional meaning, which allows more people (and more machines) to do more with the data.

Query

Query languages go hand-in-hand with databases. If the Semantic Web is viewed as a global database, then it is easy to understand why one would need a query language for that data. SPARQL is the query language for the Semantic Web.

Inference

Near the top of the Semantic Web stack one finds inference — reasoning over data through rules. W3C work on rules, primarily through RIF and OWL, is focused on translating between rule languages and exchanging rules among different systems.

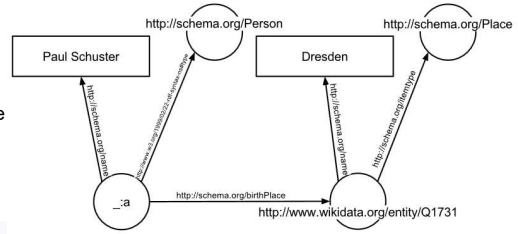
Vertical Applications

W3C is working with different industries — for example in Health Care and Life Sciences, eGovernment, and Energy — to improve collaboration, research and development, and innovation adoption through Semantic Web technology. For instance, by aiding decision-making in clinical research, Semantic Web technologies will bridge many forms of biological and medical information across institutions.

Example

- The text 'Paul Schuster was born in Dresden' on a Website will be annotated, connecting a person with their place of birth.

RDFa-syntax (or Resource Description Framework in Attributes



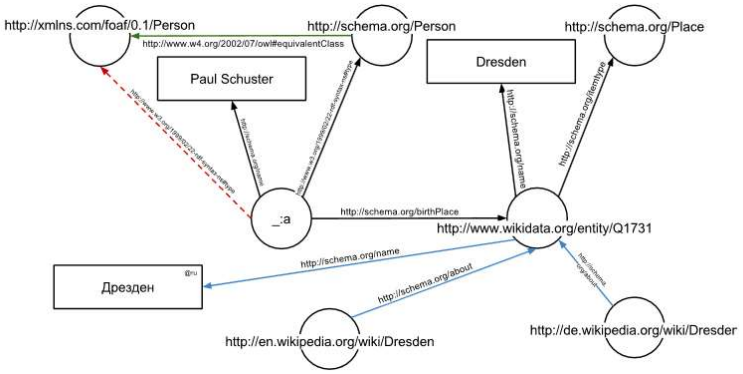

```
<div vocab="http://schema.org/" typeof="Person">
  <span property="name">Paul Schuster</span> was born in
  <span property="birthPlace" typeof="Place"
href="http://www.wikidata.org/entity/Q1731">
    <span property="name">Dresden</span>.
  </span>
</div>
```

https://en.wikipedia.org/wiki/RDFa
Semantic_Web and Semantic_triple

Example #2

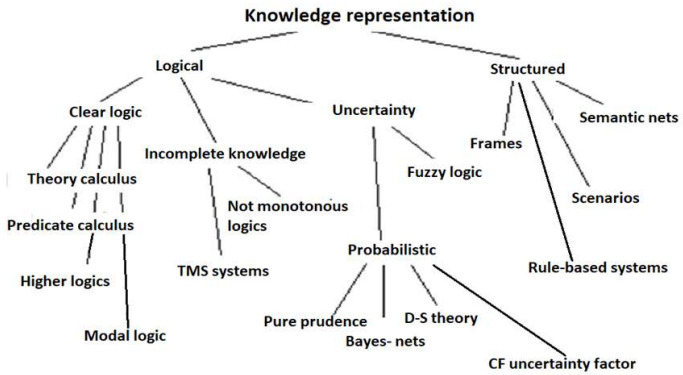
- A semantic triple, is the atomic data entity in the Resource Description Framework (RDF) data model.
- Each triple represents one edge in the graph:
  - the first element of the triple (the subject) is the name of the node where the edge starts,
  - the second element (the predicate) the type of the edge, and
  - the last and third element (the object) either the name of the node where the edge ends or a value.

Example #3



Bayes' Theorem

Knowledge representation



Uncertainty

- Decisions were made in cases, when the occurence of an event / fact was known, moreover, the connection between the events was also clear.
- Unfortunately, in real case problem solving our knowledge is usually
  - Deficient, for example we cannot answer,
  - Not relaible, inaccurate,
  - Not exact, there is no exact formalism to describe the situation,
  - Contradictory, multiple sources with conflicts.

How can we handle uncertainty?

Numerical methods

- Bayesian model.
- Bayesian network.
- Dempster-Schafer theory.
- Fuzzy model.

Symbolic methods

- Non monotone systems.

Bayes model

- Bayes' Theorem is a mathematical formula used for calculating conditional probabilities.
- It describes the probability of an event, based on prior knowledge of conditions that might be related to the event.
- For example, a person's age can be used to more accurately guess the frequency of visiting a doctor, then without this information.

Events

- Let us consider random experiments.
- The results of the experiments are considered to be elementary samples / events.
- The sample space or complete probability space of every possible outcame is: Ω.
- An event is a subset of Ω.
- Full event T = Ω.
- Empty event: F = Ø.
- Mutually exclusive events: A ∩ B = Ø.

Probability measure

- Probability: P : $2^{\Omega} \rightarrow [0,1]$ function, where
  - P(T)=1 full event;
  - P(F)=0 empty event;
  - P(A □ B) = P(A) + P(B) for mutually exclusive events.
- Probability measure: P(A), probability of observing event A, independetly from any other events, i.e. without any information on any other events.
- $P(A) = \frac{number\ of\ favored\ events}{all\ possible\ events}$, $P(A) \in [0,1]$
- Please note that $P(A) + P(\bar{A}) = 1$. NB. complementary event.
- Mutually independent events: $A \cap B = \emptyset$

$$P(A \cup B) = P(A) + P(B)$$
$$P(A \cap B) = P(A)P(B)$$

Complete probability space

- Let us denote the complete probability space by Ω, where

$$\Omega = \{A_1, A_2, \dots, A_n\}, n > 0$$
$$A_1 \cup A_2 \cup \dots \cup A_n = \Omega, where\ A_i \neq A_j\ and\ \forall i \neq j$$

- Then

$$\Rightarrow P(A_1) + P(A_2) + \dots + P(A_n) = P(\Omega) = 1$$

- Notation: A ∩ B = AB és P(A ∩ B) = P(AB).

Conditional probability

- The measure of the probability of an event *A* given that another event *B* has occured $P(A|B)$.
- Calculation method:

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(A \cap B)}{P(B)}, where\ P(B) > 0$$
$$\Rightarrow P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

- If A and B are independent, then

$$P(A \cap B) = P(A)P(B) \Rightarrow P(A|B) = \frac{P(A)P(B)}{P(B)} = P(A)$$

Conditional probability #2

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

Implies the following Bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\bar{A})P(\bar{A})}$$

Note: $P(A) = P(A \cap B) + P(A \cap \bar{B})$

$$\Rightarrow \mathrm{P(A)} = P(A|B)P(B) + P(A|\bar{B})P(\bar{B})$$

- Bayes' rule corresponds to the probability of the causes; i.e. prior probabilities.

In other words…

- Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a likelihood function derived from a statistical model for the observed data: $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$
- Where H is the hypothesis, P(H) is the prior probability, is the estimate of the probability of the hypothesis H before data E is observed.
- Evidence E corresponds to new data.
- P(H|E) is the posterior probability, after E is observed.

Example

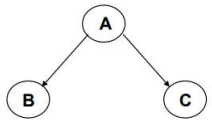
- **If** the patient caught a cold, **then** he has a fever. (75%)
- **Question: If** the patient has a fever, **then** did he catch a cold? (?%)
- Notation: M – patient caught a cold; L – patient has a fever.
- Required:
  - P(M)=0.2 - caught a cold.
  - P(L|M)=0.75 - has a fever, assuming that he caught a cold.
  - P(L|$\overline{M}$)=0.2 - has a fever, assuming that he has no cold.
- Then: $P(M|L) = \frac{P(L|M)P(M)}{P(L|M)P(M)+P(L|\bar{M})P(\bar{M})} = \frac{0{,}15}{0{,}31} = 0{,}483.$

Difficulties

- It is very difficult to find a complete probability space with mutually exclusive events.
- Many probabilities have to be defined and given. Moreover, their alteration is difficult to be followed and updated.

Bayesian networks

- Probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph.
- The vertices may be observable quantities, hypothesis.
- Directed arcs represent direct influences, conditional dependencies.
- Example:



E.g., A is Fire, B is Heat, C is Smoke.
"Where there's Smoke, there's Fire."

If we see Smoke, we can infer Fire.

If we see Smoke, observing Heat tells us very little additional information.

Richard H. Lathrop, www.ics.uci.edu

Bayesian networks

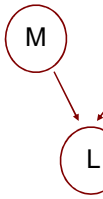
- All conditional dependencies, i.e. cause and effect relationships, has to be given.
- Therefore a probability data has to be assigned to each node. This will provide the probability data at each parent-set as follows. Each node is assigned to a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability of the variable represented by the node.

Bayesian network example

L : the patient has a fever
M: the pateint caught a cold
T : the patient has pneumonia

P(M)
0.2

P(T)
0.03



P(M)
0.2

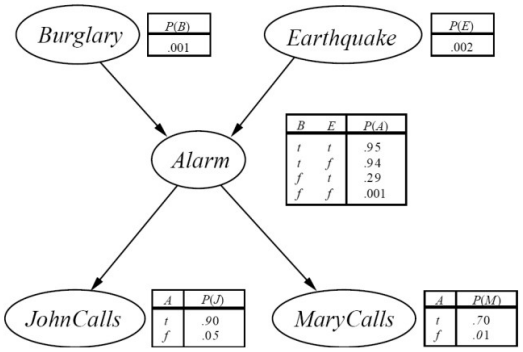
M	P(L|M)
0	0.20
1	0.75

M	T	P(L|MT)
0	0	0.10
0	1	0.60
1	0	0.75
1	1	1.00

Burglar alarm example

- Consider the following 5 binary variables:
  - B = a burglary occurs at your house
  - E = an earthquake occurs at your house
  - A = the alarm goes off
  - J = John calls to report the alarm
  - M = Mary calls to report the alarm
- What is P(B | M, J) ? (for example)
- We can use the full joint distribution to answer this question – Requires 25 = 32 probabilities
- Can we use prior domain knowledge to come up with a Bayesian network that requires fewer probabilities?

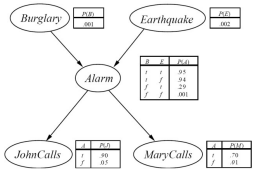
Burglar alarm example #2



- NB: Only requires 10 probabilities!

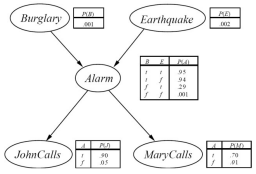
Burglar alarm example #3

- Order the variables in terms of influence (may be a partial order) e.g., {E, B} -> {A} -> {J, M}
- P(J, M, A, E, B) = P(J, M | A, E, B) P(A| E, B) P(E, B)
  ≈ P(J, M | A) P(A| E, B) P(E) P(B)
  ≈ P(J | A) P(M | A) P(A| E, B) P(E) P(B)
- These conditional independence assumptions are reflected in the graph structure of the Bayesian network.



Burglar alarm example #4

- P(J, M, A, E, B) = P(J | A) P(M | A) P(A | E, B) P(E) P(B)
- There are 3 conditional probability tables (CPDs) to be determined: P(J | A), P(M | A), P(A | E, B) – Requiring 2 + 2 + 4 = 8 probabilities
- And 2 marginal probabilities P(E), P(B) -> 2 more probabilities.
- Where do these probabilities come from? Expert knowledge – From data (relative frequency estimates) – Or a combination of both.

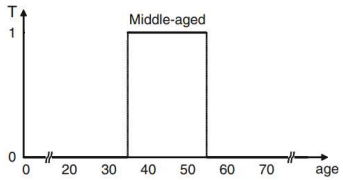


Fuzzy Sets and Models

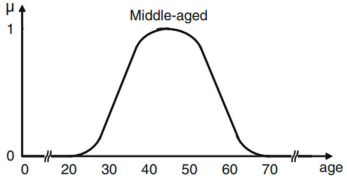
Fuzzy model

- Decisions in uncertain situations, to cover subjectivity of the natural language.
- The word “fuzzy” means “vagueness (ambiguity)”, occurs when the boundary of a piece of information is not clear.
- Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. (Novák, Perfilieva and Močkoř, 1999)

Example: middle-aged person



Sharply defined between 35 and 55 years.



Defined with a fuzzy set.

Fuzzy model #2

- Other examples: fast driven car, tall person, approx. 180cm high, smart student.
- Fuzzy sets theory is a tool to describe how and to what extent an object fits to the uncertain facts.
- It is not a probability model.

Term: fuzzy

- The term fuzzy sets and fuzzy logic was proposed by Lotfi Zadeh, University of Berkeley, 1965.
- Mathematical solution to handle uncertainty behind linguistic variables.
- A linguistic variable is a variable whose values are words or sentences instead of numerical values
- Fundamental characteristics: partial membership, i.e. to what extent is the object a member of the given set.
- The fuzzy logic theory is based on fuzzy sets which are a natural extension of the classical set theory.

CLASSICAL SET THEORY	FUZZY SET THEORY
Classes of objects with sharp boundaries.	Classes of objects with un-sharp boundaries.
A statement is true: 1; or false: 0.	The truth value of a statement: $\mu(p) \in [0, 1]$
A sharp set is defined by a bivalent truth function: $K_A(x) = \begin{cases} 1 \text{ if } x \in A, \\ 0 \text{ if } x \notin A \end{cases}$	A fuzzy set is defined by its membership function: $\mu_A(x) \in [0, 1]\ \forall x \in \Omega$

Fuzzy sets

- Fuzzy Logic uses the whole interval between 0 (false) and 1 (true) to describe human reasoning.
- A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, having interval [0,1].
- Let X be a set of x objects, then the fuzzy set:
  $F(X) = \{(x, \mu(x)) \mid x \in X, \mu(x) \in [0,1]\}$

Universe of discourse

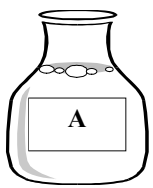
- A fuzzy set is built from a reference set called universe of discourse.
- The reference set is never fuzzy.
- Let us assume that $U = \{x_1, x_2, \dots, x_n\}$ is the universe of discourse, then a fuzzy set A in U $(A \subset U)$ is defined as a set of ordered pairs $\{(x_i, \mu_A(x_i))\}$, where $x_i \in U, \mu_A: U \to [0,1]$ is the degree of membership of x in A.

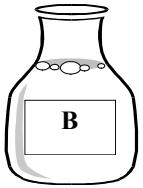
Example

- Consider the universe of discourse U = {1, 2, 3, 4, 5, 6}. Then a fuzzy set A holding the concept ‘large number’ can be represented as
  A = {(1, 0), (2, 0), (3, 0.2), (4, 0.5), (5, 0.8), (6, 1)}
  Please note that 1 and 2 are not „large numbers,” the membership degrees equal zero. Numbers 3, 4 and 5 partially belong to the „large numbers,” their degrees are 0.2, 0.5 and 0.8. Finally 6 is a large number.
- Integers around 10:
  T = {…, (7,0.5), (8,0.7), (9,0.9), (10,1), (11,0.9),…}

Example: let’s have a drink!

-



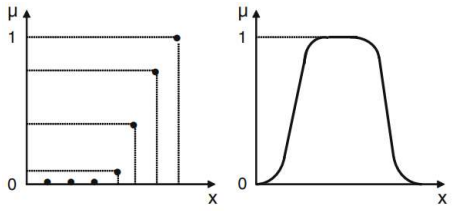


$\mu_{Drinkable}(A) = 0{,}91$ $P(B \in Drinkable) = 0{,}91$

$\mu_{Drinkable}(A) = 0{,}5$ $P(B \in Drinkable) = 0{,}5$

Membership functions

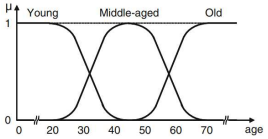
- Fuzzy sets are commonly represented by a membership function.
- Depending on the reference set, the membership functions are either discrete or continuous.



Discrete and continuous fuzzy sets.

Fuzzy partition of the universe

- Several fuzzy sets may be defined on the same reference set.
- A linguistic expression from the natural language can label the fuzzy sets in order to express their semantics.



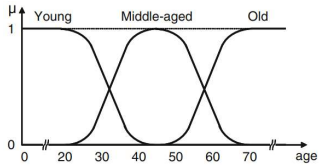
Fuzzy partition of the reference set with labelled fuzzy sets. Please note that there are transitions between them.

- This construct is called a linguistic variable, whose values are words or sentences instead of numbers; and the values are called terms.

Linguistic variable

- A linguistic variable is characterized by a quintuple
  (X, T, U, G, M)
  where X is the name of the variable,
  T is the set of terms of X,
  U is the universe of discourse,
  G is a syntactic rule for generating the name of the terms and
  M is a semantic rule for associating each term with its meaning,
  i.e. a fuzzy set defined on U

Linguistic variable example



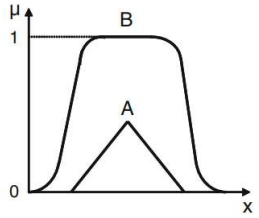
- A linguistic variable is characterized by a quintuple
  (X, T, U, G, M)
  where X is „age”,
  T is the set {young, middle-aged, old},
  U = [0, 100],
  and
  G and M specifies for each term a corresponding fuzzy set.

Properties of fuzzy sets

- As the fuzzy set theory is an extension of the classical set theory, crisp sets are specific cases of the fuzzy sets.
- A fuzzy set is considered to be empty if the membership degrees of all the elements of the universe are equal to zero.
- A fuzzy set A, defined over a reference set U, is empty if $A = \emptyset \Leftrightarrow \mu_A(x) = 0, \forall x \in U.$
- Two fuzzy sets are equal if their membership degrees are equal for all the elements of the reference set, i.e. if the two fuzzy sets have the same membership function.
  Two fuzzy sets A and B, defined over a reference set U, are equal if $A = B \Leftrightarrow \mu_A(x) = \mu_B(x), \forall x \in U.$

Properties of fuzzy sets #2

- A fuzzy set A is included in a fuzzy set B if the degrees of membership of A are smaller or equal to the membership degrees of B for all the elements of the universe.
  $A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x), \forall x \in U.$



Properties of fuzzy sets #3

- The cardinality of a crisp set equals the number of its elements.
- In a fuzzy set the cardinality is the sum of the membership degrees of the reference set elements. If the reference set is infinite, an integral over the universe is used.
- The relative cardinality of a fuzzy set is the cardinality of the fuzzy set divided by the cardinality of the universe. This allows fuzzy sets to be compared.

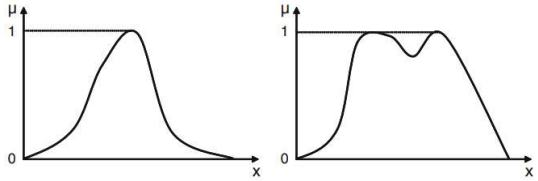
$\mathrm{Card}(A) = |A| = \sum \mu_A(x), \forall x \in U.$
$RelCard(A) = \|A\| = \frac{|A|}{|U|}.$

Example revisited

- Consider the universe of discourse U = {1, 2, 3, 4, 5, 6}.
  A fuzzy set A holding the concept 'large number'
  A = {(1, 0), (2, 0), (3, 0.2), (4, 0.5), (5, 0.8), (6, 1)}
  Then the cardinality:
  $\mathrm{Card}(A) = 0 + 0 + 0.2 + 0.5 + 0.8 + 1 = 2.5.$
  $RelCard(A) = \frac{2.5}{6}.$

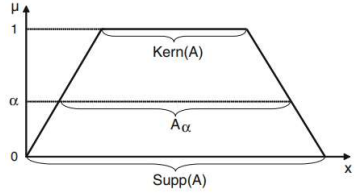
Properties of fuzzy sets #4

- A fuzzy set is convex if any point located between two other points has a higher membership degree than the minimum membership degree of these points.
  $\forall x, y \in U, \forall \lambda \in [0,1]: \mu_A(\lambda x + (1-\lambda)y) \geq min(\mu_A(x), \mu_A(y))$.



Properties of fuzzy sets #5

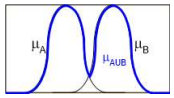
- The support of a fuzzy set is the sharp subset of the universe where the membership degrees are greater than zero.
  $Supp(A) = \{x \in U, \mu_A(x) > 0\}.$
- The α-cut of a fuzzy set is the crisp subset of the universe where the membership degrees are greater than or equal to the specified α value.
  $A_\alpha = \{x \in U, \mu_A(x) \geq \propto, \propto \in [0,1]\}.$



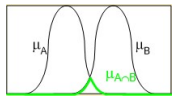
- The kernel of a fuzzy set is the crisp subset of the universe where the membership degrees are equal to 1.
  $Kern(A) = \{x \in U, \mu_A(x) = 1\}.$

Operations on fuzzy sets

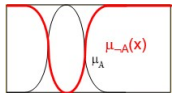
$\mu_{A\cup B}(x) = \max \{\mu_A(x), \mu_B(x)\}$



$\mu_{A\cap B}(x) = \min \{\mu_A(x), \mu_B(x)\}$



$\mu_{\neg A}(x) = 1 - \mu_A(x)$

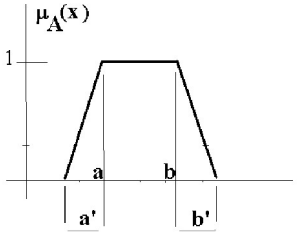


Operations: linguistic amendments

- Very/ highly: x^2
- Somewhat less: $x^{1.75}$
- Very/ highly: 1.25x
- Somewhat more: $x^{0.75}$
- Less: 0.75x
- More or less: 0.75 $x^{0.75}$

Fuzzy arithmetics

- $\mu_A(x)$ convex fuzzy set is called to be a fuzzy number on the set of real numbers.
- Summation: [a,b,a',b'] + [c,d,c',d'] = [a+c,b+d,a'+c',b'+d']
- Subtraction, multiplication and division: similarly.



Fuzzy implication

- Fuzzy implication is defined by an umplication operator.
- For example, Gödel's operator:

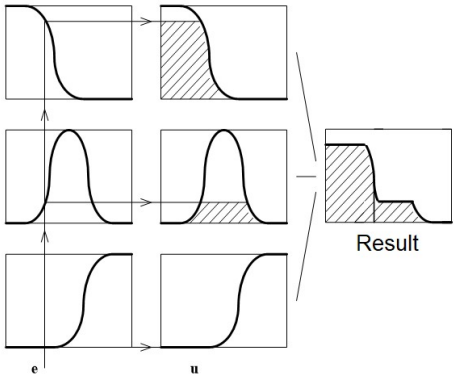
$$\mu_{A\to B}(x, y) = \min(\mu_A(x), \mu_B(y))$$

Conclusion

IF-THEN RULES

if e neg then u neg
if e zero then u zero
if e pos then u pos

Defuzzyfication



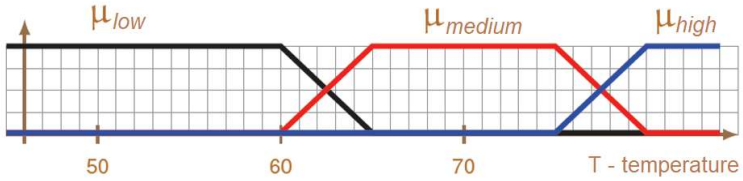
Pros and cons

Pros:
- Fuzzy is close to human linguistics.
- It can be modified relatively fast.
- Easy description.
- Validity of the rules can be explicitely given.
- It can be used in case of deficient, uncertain, complex problems.

Cons:
- Membership functions are subjective.

Example

- Let us consider the fuzzy control example of the washing time in case of an idealistic washing machine.
- IF [*x* is low AND *y* is high] THEN *z* = medium
- Let us define the low, high and medium sets as follows:

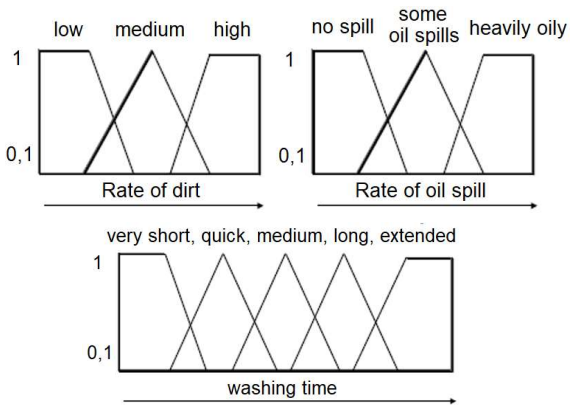


Example #2

Let us determine the input and output variables as follows:
- Input variable: clothes to be washed.
- Rate of dirt: (low, medium, high).
- Rate of oil spill: (no spill, some oil spills, heavily oily).
- Output variable: (very short, quick, medium, long, extended).

Example #3

Membership functions of the input and output variables:



Example #4

The control rules and the defuzzification method:

- Rule 1:

  IF rate of dirt IS medium AND rate of oil spill IS heavily oily THEN washing time IS long.

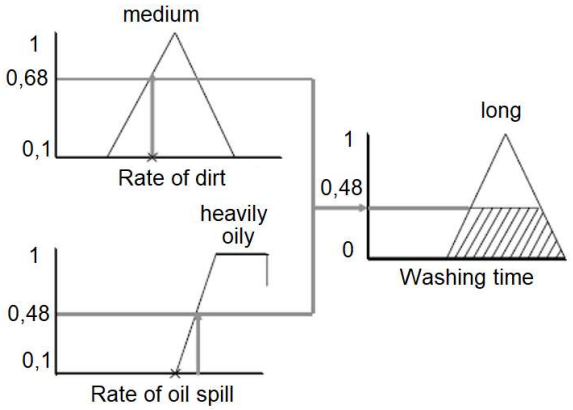
- Rule 2:

  IF rate of dirt IS low AND rate of oil spill IS heavily oily THEN washing time IS medium.

- Defuzzification method: centroid method (COA)

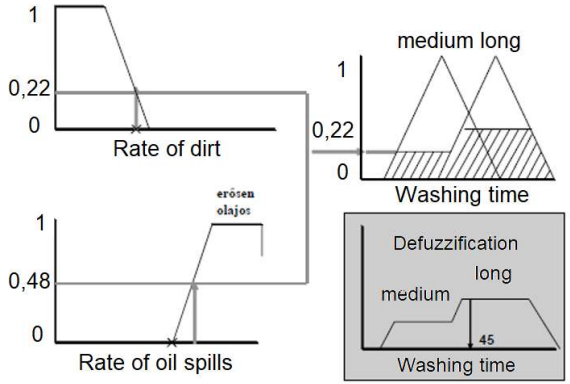
Example #5

Evaluation of Rule 1:



Example #6

Evaluation of Rule 2:



Fuzzy Markup Language

- IEEE STANDARD 1855–2016 is about a specification language named Fuzzy Markup Language (FML).
- FML allows modelling a fuzzy logic system in a human-readable and hardware independent way.
- FML is based on eXtensible Markup Language (XML).