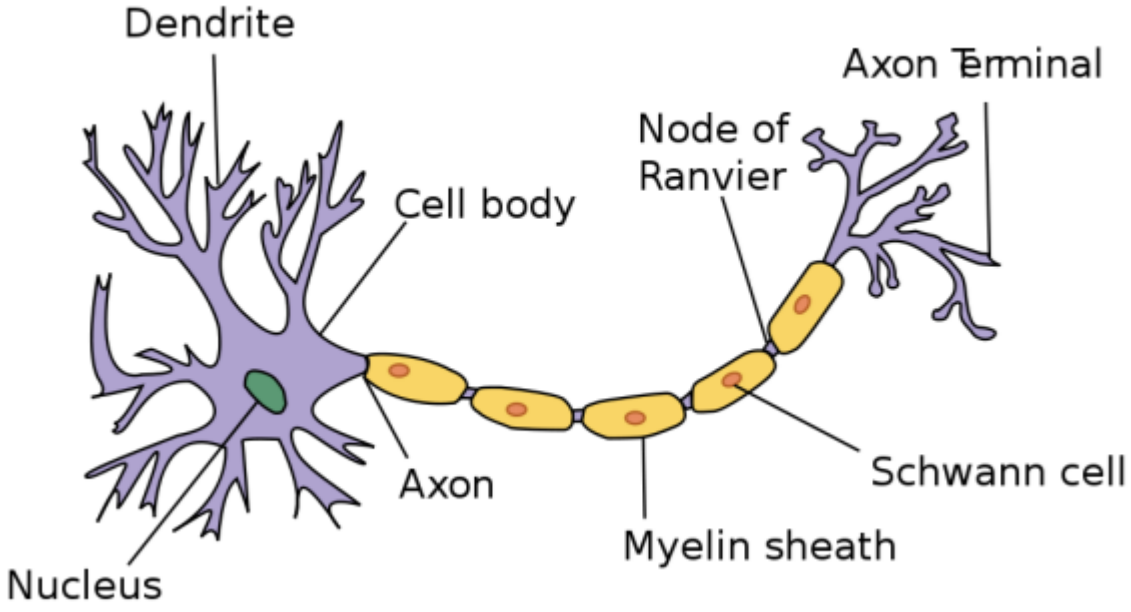
# Računarska inteligencija

## Uvod u računarsku inteligenciju

**Inteligencija** se definiše kao mogućnost razumevanja i ostvarivanja koristi od razumevanja. Uključuje kreativnost, veštine, svesnost, emocije i intuiciju. **Računarska inteligencija** (**Computational Inteligence** - **CI**) je podgrana veštačke inteligencije (Artificial Intelligence - AI) koja izučava mehanizme inteligentnog ponašanja u složenim i promenljivim okruženjima. To su mehanizmi koji mogu da uče, da se prilagođavaju, uopštavaju i slično. Srodan termin je **Soft Computing** koji podrazumeva upotrebu višestrukih CI paradigmi i verovatnosnih metoda. **Paradigme računarske inteligencije** su:

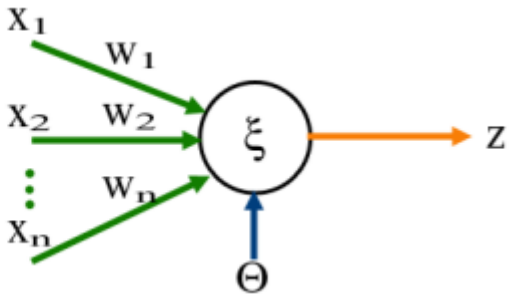
- **veštačke neuronske mreže** (**Artificial Neural Networks** - **ANN**)
- **evolutivna izračunavanja** (**Evolutionary Computation** - **EC**)
- **inteligencija grupa** (**Swarm Intelligence** - **SI**)
- **veštački imuni sistem** (**Artificial Immune System** - **AIS**)
- **rasplinuti** (**fazi**) **sistemi** (**Fuzzy Systems** - **FS**)

### Veštačke neuronske mreže

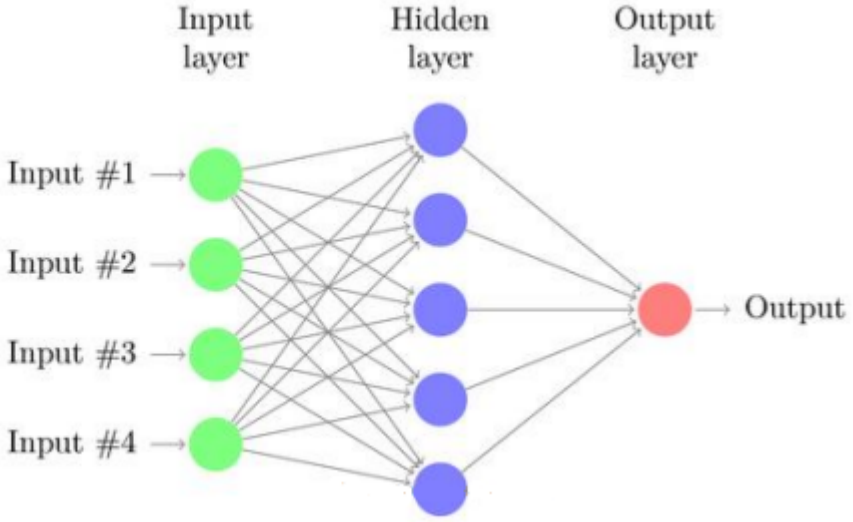



Ljudski mozak ima mogućnost prepoznavanje oblika, percepciju, motoriku i slično. U tome je mnogo efikasniji od bilo kog računara. Sadrži između 10 i 500 milijardi neurona i 50 triliona sinapsi. Pitanje je da li će ikada biti moguće modelovati ljudski mozak u potpunosti. Današanje veštačke neuronske mreže omogućavaju rešavanje pojedinačnih zadataka, dok

mozak može da rešava mnogo zadataka istovremeno. **Biološki neuroni** su međusobno povezani vezama koje se otvaruju između aksona jednog neurona i dendrita drugog neurona. Ta veza naziva se **sinapsa**. Signali se kreću od dendrita, preko tela ćelije do aksona. Slanje signala dešava se povremeno - u trenutku kada ćelija "ispali" signal. Neuron može da suzbije ili pojača jačinu signala. **Veštački neuron** (**Artificial Neuron** - **AN**) je model biološkog neurona. AN prima signal od okruženja ili od drugog AN, a nakon toga ga prenosi svim povezanim AN. Ispaljivanje signala i njegova jačina su kontrolisani formulom.




**Veštačka neuronska mreža** (**Artificial Neural Network** - **ANN**) je mreža slojevito poređanih AN. Obično sadrži ulazni, izlazni i nula ili više središnjih slojeva neurona.




ANN mogu biti:

- **jednoslojne**
- **višeslojne sa propagacijom unapred**
- **temporalne i rekurentne** - predviđanja iz prethodnih koraka utiču na trenutni korak
- **samoorganizujuće**
- **kombinovane**

ANN primenjuju se u medicinskoj dijagnostici (npr. prepoznavanje tumora), prepoznavanju zvuka, procesiranju slika, predviđanju u analizi vremenskih serija, kontroli robota, klasifikaciji i kompresiji podataka i tako dalje.

## Evolutivna izračunavanja

**Evolutivna izračunavanja (Evolutionary Computation - EC)** imitiraju proces prirodne evolucije, pri čemu je glavni koncept **preživljavanje najprilagođenijih**. Slabije prilagođene jedinke izumiru. Pod izumiranjem se ne misli na ontološki završetak života već na ostavljanje

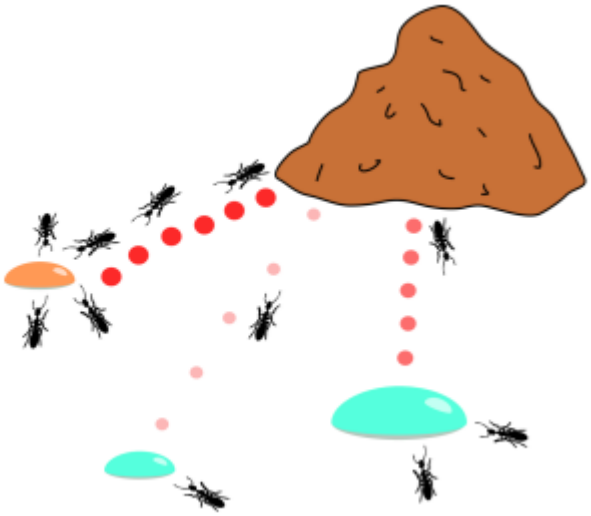
potomstva, odnosno prolongiranje života gena. Jedinka populacije obično se naziva **hromozom**. Karakteristike hromozoma nazivaju se **geni**, a vrednost gena naziva se **alel**. EC algoritmi iterativno, kroz generacije simuliraju evoluciju. Postoje različite realizacije EC:

- **genetski algoritmi**
- **genetsko programiranje**
- **evolutivno programiranje**
- **evolutivne strategije**
- **diferencijalna evolucija**
- **kulturalna evolucija**
- **koevolucija**

EC koriste se za rešavanje teških problema i u diskretnoj i u kontinualnoj optimizaciji. Koriste se i za klasifikaciju podataka, klaster analizu, aproksimaciju vremenskih serija i tako dalje.

## Inteligencija grupa

**Inteligencija grupa (Swarm Intelligence - SI)** zasnovana je na socijalnom ponašanju nekih organizama kao što su mravi, pčele i ptice. Najpoznatija optimizaciona tehnika je **Particle Swarm Optimization (PSO)** modelovana po uzoru na ponašanje jata ptica. Svaka jedinka u jatu predstavlja kandidat rešenje posmatranog problema i predstavljena je svojom pozicijom. Jedinke koje u prostoru rešenja zauzimaju bolju poziciju privlače druge jedinke, odnosno jedinke koje vide da je neka druga bolja žele da se ugledaju na nju, ali i dalje zadržavaju neki nivo individualnosti. Slično važi i za ostale tehnike. **Optimizacija kolonijom mrava** imitira mravlju koloniju gde se mravi u početku dosta nasumično kreću i "istražuju" prostor. Mravi koji slučajno nalete na hranu će je odneti u koloniju, pri čemu stalno luče feromon koji će privlačiti druge mrave i navoditi ih u pravcu hrane.



# Optimizacija

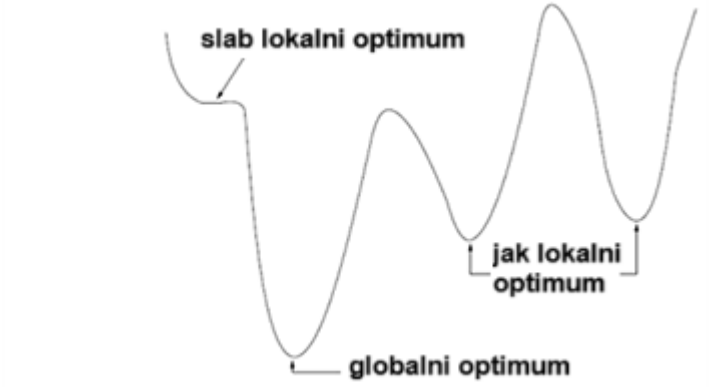
Optimizacioni algoritmi pripadaju grupi **algoritama pretrage**. Cilj je pronaći rešenje problema takvo da je:

- neka ciljna funkcija (**funkcija cilja**) maksimalna/minimalna.
- neki **skup ograničenja** zadovoljen. Ovaj uslov je opcion, odnosno nekada nećemo imati ograničenja.

Optimizacija je sama po sebi teška, pogotovo kod rešavanja NP teških problema gde se veruje da su algoritmi u najboljem slučaju eksponencijalne složenosti. Još neki od izazova optimizacije su:

- Rešenje može biti predstavljeno kao kombinacija vrednosti iz različitih domena.
- Ograničenja, a i sama funkcija cilja, mogu biti nelinearna.
- Karakteristike problema mogu varirati tokom vremena.
- Funkcija cilja može biti u konfliktu sa ograničenjima.

Neka je $S$ **domen** posmatranog problema. **Funkcija cilja** je funkcija $f : S \to R$ koju želimo da minimizujemo ili maksimizujemo. Ona govori o kvalitetu rešenja. Važi da je minimum funkcije $f$ isto što i maksimum funkcije $-f$, pa po konvenciji uvek možemo govoriti o minimizaciji. Skup $x$ predstavlja **nezavisne promenljive** koje utiču na vrednost funkcije $f$ i za date vrednosti promenljivih funkcija ima vrednost $f(x)$. Skup ograničenja najčešće postavlja zavisnosti između nezavisnih promenljivih, ali može i da ograničava nezavisne promenljive na neki interval ili skup vrednosti. Ukoliko ne postoji funkcija cilja, već samo ograničenja koja treba zadovoljiti, onda se radi o **programiranju ograničenja (Constraint programming)**, a ne o optimizaciji. Klasični primeri su problem $n$ dama i SAT problem.




Pri optimizaciji razlikujemo nekoliko tipova optimuma:

- **globalni optimum** - najbolje rešenje na čitavom dopustivom skupu rešenja $S$. Moguće je da postoji više globalnih optimuma.
- **jak lokalni optimum** - najbolje rešenje u nekoj okolini $N \subseteq S$. Predstavlja najveću prepreku za optimizacione algoritme.

- **slab lokalni optimum** - jedno od najboljih rešenja u okolini $N \subseteq S$. Većina optimizacionih algoritama može lako da savlada ovaj tip lokalnog optimuma.

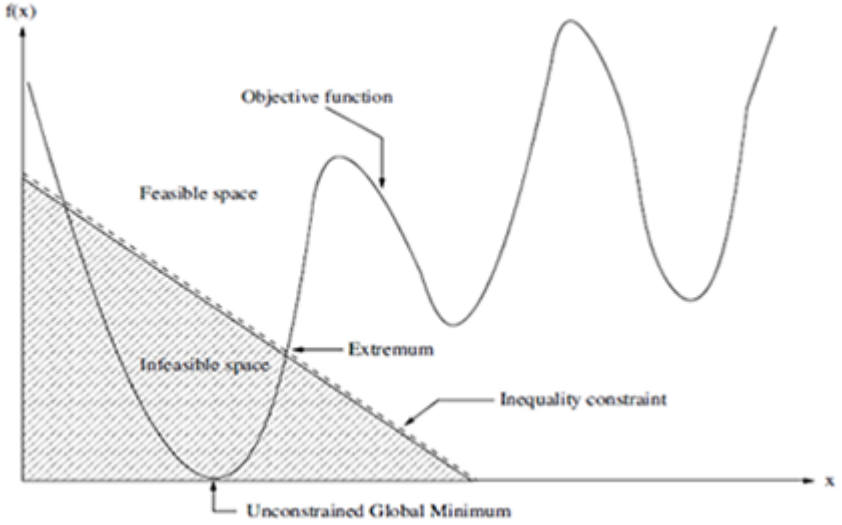
Optimizacione metode traže optimum u prostoru **dopustivih rešenja** - rešenja koja ispunjavaju ograničenja problema. Metode se prema fokusu pretrage dele na:

- **lokalne** - nisu sposobne da nađu globalne optimume. Jedna velika grupa lokalnih metoda su **gradijentne metode** koje na neki način prate gradijent. Lokalne metode mogu naći i globalni optimum ako imaju sreće, odnosno ako su početni uslovi dobri.
- **globalne** - imaju mehanizme koji im omogućavaju da izbegnu lokalne optimume, na primer mehanizam razmrdavanja.

Optimizacione metode se prema pristupu pretrage dele na:

- **stohastičke** - koriste elemente nasumičnosti u procesu pretrage i ne garantuju uvek isto rešenje za iste ulazne parametre. Globalne metode su češće stohastičke jer im to omogućava bolju pretragu prostora rešenja i pronalaženje globalnog optimuma.
- **determinističke** - koriste precizno definisana pravila i algoritme za pretragu, što znači da će za iste ulazne parametre uvek dati iste izlazne rezultate. Lokalne metode su češće determinističke.

**Optimizacija bez ograničenja** podrazumeva da se funkcija cilja minimuzuje na celom domenu $S$. **Optimizacija sa ograničenjima** podrazumeva minimizaciju funkcije cilja uz domenska ograničenja promenljivih, kao i ograničenja zasnovana na jednakostima i/ili nejednakostima. Forma funkcija ograničenja u opštem slučaju nije linearna.




Nedopustiva rešenja mogu se obraditi na razne načine:

- **odbacivanje**
- **dodeljivanje penala** - na primer negativni faktor u slučaju maksimizacije ili pozitivan faktor u slučaju minimizacije. Omogućava veću fleksibilnost jer možda je nedopustivo rešenje jako blizu optimuma pa je korisno iskoristiti ga.

- **svođenje na rešenje bez ograničenja**, a potom konvertovanje u rešenje koje poštuje ograničenja
- **održavanje dopustivosti** dizajnom metoda - krećemo se po prostoru rešenja tako da nikada ne dolazimo u nedopustiva rešenja.
- **uređivanje nedopustivih rešenja** prema stepenu nedopustivosti
- **popravljanje nedopustivih rešenja** - na primer krećemo se do najbližeg dopustivog rešenja.

Postoje još neki vidovi optimizacionih problema:

- **problemi sa višestrukim optimumom** - cilj je pronaći sva rešenja koja su optimalna ili dovoljno blizu optimuma. Na primer, detekcija otiska na ekranu.
- **višeciljna optimizacija** - problem ima više funkcija cilja. Na primer, želimo da dođemo od tačke A do tačke B tako da istovremeno utrošimo i najmanje vremena i najmanje goriva.
- **dinamička optimizacija** - funkcija cilja i ograničenja mogu se menjati tokom vremena. Na primer, autopilot aviona ili automobila.

Prema tipu domena nad kojim se vrši, optimizacija se deli na:

- **kombinatornu (diskretnu)** - dopustivi skup vrednosti promenljivih je najviše prebrojiv, odnosno konačan ili prebrojivo beskonačan. Specijalni slučaj su problemi **binarne optimizacije** gde promenljive uzimaju vrednosti 0 ili 1.
- **globalnu (kontinualnu)** - dopustivi skup vrednosti promenljivih je neprebrojiv, iz skupa realnih brojeva.

Primer kombinatorne optimizacije je **problem trgovačkog putnika (Travelling Salesmen Problem - TSP)**: Neka je zadat skup $C$ od $m$ gradova i funkcija udaljenosti $d(c_i,\ c_j) \in N$ za svaki par gradova. Pronaći permutaciju $p : [1..m] \to [1..m]$ takvu da je ukupna suma udaljenosti grana koje se prolaze obilaskom minimalna. Funkcija udaljenosti može biti proizvoljna, na primer rastojanje između gradova ili vreme potrebno da se stigne iz jednog grada u drugi. Moguće je rešiti TSP na više načina:

- **Brute-force** - totalna enumeracija. Veličina dopustivog skupa je $m!$, pa pomoću Stirlingove formule uočavamo da bi rešenje bilo eksponencijalne složenosti.
- **Dinamičko programiranje (tehnika memoizacije)** - može se ostvariti malo poboljšanje, ali asimptotski rešenje ostaje eksponencijalne složenosti.
- **P-aproksimativni algoritam** - sa stepenom sigurnosti P moguće je matematički dokazati da se posmatrani problem svodi na neki drugi. Za TSP je utvrđeno da postoji 2-aproksimativni algoritam baziran na minimalnom razapinjućem stablu. To znači da je rešenje dobijeno ovim algoritmom najviše 2 puta gore od optimalnog, što i nije baš dobar faktor aproksimacije. Obično imaju asimptotski manju složenost od prva dva pristupa.

Prva dva pristupa su neadekvatna kada dimenzija problema naraste, ali prednost je što daju optimalna rešenja ako se završe. Aproksimativni pristup je efikasan (polinomijalan) i imamo garanciju da rešenje ne može da bude više od P puta lošije, ali mana je što dobijeno P nekada nije zadovoljavajuće. **Metaheuristike** su pristupi koji će nam omogućiti da vreme izvršavanja bude razumno, ali i da kvalitet rešenja bude "očekivano" blizak optimalnom. Mana u odnosu na aproksimativni pristup je što nemamo nikakve garancije koliko će rešenje zapravo biti dobro. Metaheuristike pripadaju široj grupi algoritama pretrage.

Opšta forma algoritma **lokalne pretrage** je:

- Započni od polaznog rešenja $x(0) \in S,\ t = 0$
- Dok nije zadovoljen kriterijum završetka ponavljaj:
  - Izračunaj vrednost $f(x(t))$
  - Izračunaj pravac i smer pretrage $q(t)$
  - Izračunaj dužinu koraka pretrage $\eta(t)$
  - Pređi u naredno rešenje $x(t+1) = x(t) + \eta(t)q(t)$
  - $t = t + 1$
- Vrati $x(t)$ kao konačno rešenje

**Dužina koraka pretrage** predstavlja stepen poverenja u izračunati pravac i smer pretrage. On se povećava tokom vremena jer želimo da budemo sve sigurniji. **Kriterijum završetka** može biti fiksan broj iteracija, stabilizacija pomeranja i slično. **Pravac i smer pretrage** u kontinualnoj optimizaciji se može prirodno računati preko izvoda. U diskretnom slučaju mogu se koristiti metode koje će malo izmeniti rešenja i imitirati izvod. Na primer, ako rešenja predstavljamo kao niz bitova, invertovanje nasumičnog bita može predstavljati funkciju koja imitira izvod. Metaheuristike koriste principe nasumičnosti i determinizma kako bi izbegle problem lokalne pretrage, a to je zaglavljivanje u lokalnom minimumu. Dakle, metaheuristika svodi se na dva koraka:

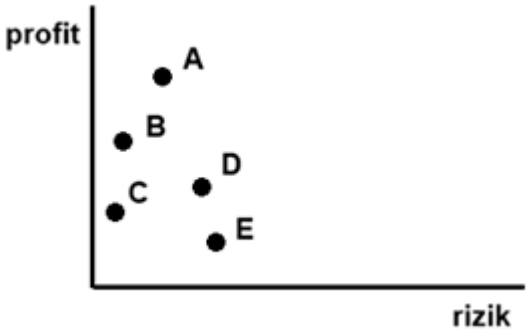
- **diverzifikacija** - slučajno se pomeramo na neko rešenje, odnosno istražujemo prostor rešenja.
- **intenzifikacija** - u okolini tog rešenja dolazimo do lokalnog optimuma lokalnom pretragom.

Metaheuristike se dele na:

- **S metaheuristike** - rade samo sa jednim rešenjem. Primer je **simulirano kaljenje** gde u svakom koraku sa nekom verovatnoćom prihvatamo i lošije rešenje. Ta verovatnoća opada vremenom jer želimo da diverzifikacija bude mala pri kraju pretrage.
- **P metaheuristike** - rade sa čitavom populacijom. Primer su genetski algoritmi koji simuliraju evoluciju.

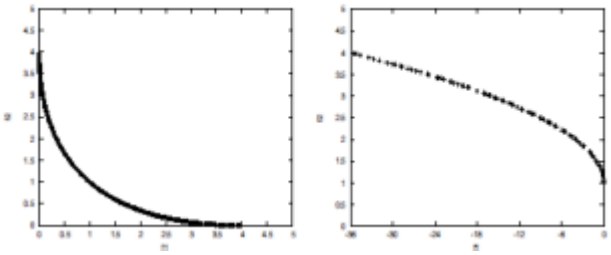
U praksi je često potrebno zadovoljiti više funkcija cilja, odnosno izvršiti **višeciljnu optimizaciju**. Na primer, potrebno je pronaći akciju sa maksimalnim prihodom i minimalnim

rizikom ili je potrebno pronaći putanju sa minimalnim utroškom goriva i minimalnim utroškom vremena. Često poboljšanje jedne funkcije cilja izaziva pogoršanje druge, na primer poboljšanje strukturalne stabilnosti mosta izaziva povećanje troškova izgradnje. U tom slučaju potrebno je na neki način napraviti balans, tj. kompromis. Trivijalan način rešavanja jeste pravljenje ponderisanih proseka odnosno agregacija. Za svaku funkciju cilja određuje se težina koja predstavlja njen značaj, pa se na ovaj način problem svodi na jednociljnu optimizaciju. Ovaj pristup je efikasan, ali ne možemo uvek znati da li su težine koje smo zadali zaista dobre. Postoje i drugi načini rešavanja. Posmatrajmo primer traženja akcija sa najvećim profitom i najmanjim rizikom. Možemo prikazati zavisnost rizika i profita u svakoj tački:




Posmatrajmo tačke A i B. Tačka A ima veći profit, ali i veći rizik od tačke B. Odnosno, profit je bolji kod tačke A, a rizik je bolji kod tačke B. Sa druge strane, ako posmatramo tačke A i D, vidimo da tačka A ima i veći profit i manji rizik od tačke D. Kažemo da rešenje $x$ **dominira** nad rešenjem $y$ ako nijedna vrednost funkcije cilja od rešenja $y$ nije bolja od odgovarajuće vrednosti funkcije cilja rešenja $x$. Rešenje $x$ je **Pareto-optimalno** ako ne postoji nijedno drugo rešenje koje dominira nad njim. Dakle, rešenja D i E su dominirana rešenja, a rešenja A, B i C su Pareto-optimalna jer međusobno nijedno ne dominira nad ostalima.

Skup svih Pareto-optimalnih rešenja naziva se **Pareto-optimalni skup**. **Pareto-optimalna površ** predstavlja površ koju formiraju funkcije cilja kada se primene nad Pareto-optimalnim skupom rešenja. Problem se dalje rešava korišćenjem algoritama pretrage koji efikasno pretražuju Pareto-optimalnu površ. To su na primer populacione strategije koje iterativno poboljšavaju skup dobrih rešenja upotrebom svojstva dominacije. Primer Pareto-optimalnih površi:



## Klase složenosti izračunavanja

**Algoritam** za rešavanje nekog problema je konačan spisak pravila čijim praćenjem dolazimo do rešenja bilo kojeg partikularnog problema iz zadate klase. Praćenje pravila traje konačno mnogo koraka. **Problemi odlučivanja** su problemi čije se rešenje sastoji u potvrđivanju ili

opovrgavanju neke osobine. Različiti optimizacioni problemi mogu se svesti na problem odlučivanja, Na primer, možemo se pitati da li za konkretan TSP problem postoji rešenje sa troškom manjim od C, a onda na osnovu odgovora možemo menjati granicu i tako iterativno tražiti najbolje rešenje. Problem pripada **klasi polinomsko rešivih problema (klasi P)** ako postoji algoritam $A$ za rešavanje tog problema i polinom $p(n)$ takav da $A$ završava izvršavanje za ne više od $p(n)$ koraka za svaku instancu tog problema, pri čemu je $n$ dimenzija problema. Polinomijalne algoritme smatramo "dobrim" algoritmima. Problem odlučivanja pripada klasi P. Algoritme čije vreme ne možemo da ograničimo polinomom podrazumevano ograničavamo eksponencijalnom funkcijom $c^n,\ c > 1$. Problem pripada **klasi eksponencijalno rešivih problema (klasi EXPTIME)** ako je za njega dokazano da ne može biti rešen algoritmom bržim od eksponencijalnog. To su na primer problemi evaluacije poteza u uopštenom šahu i igri GO ili problem pronalaženja skupa svih razapinjućih stabala u potpunom grafu, gde je dokazano da je broj razapinjućih stabala $n^{n-2}$. Problem pripada **klasi nedeterminističih polinomskih problema (klasi NP)** ako se za njega ne zna da li postoji polinomski algoritam za njegovo rešavanje, ali se za neko konkretno ponuđeno rešenje problema odlučivanja može utvrditi da li je odgovor potvrdan u polinomijalnom broju koraka. Za TSP se za bilo koju datu permutaciju gradova i dati trošak može dati potvrdan ili odričan odgovor u polinomskom vremenu.

Neka su data dva problema odlučivanja $A_1$ i $A_2$. Pretpostavimo da se za $A_1$ može konstruisati polinomski algoritam u kojem se kao jedan od koraka pojavljuje algoritam za rešavanje problema $A_2$. Ako je pritom algoritam za rešavanje problema $A_2$ polinomski onda i za $A_1$ postoji polinomski algoritam. Kažemo da se problem $A_1$ **redukuje** na problem $A_2$ ako se za svaki specijalan slučaj $X$ problema $A_1$ može u polinomskom vremenu pronaći specijalan slučaj $Y$ problema $A_2$, takav da je za problem $X$ odgovor potvrdan akko je i za $Y$ odgovor potvrdan. Jasno je da je problem $A_2$ teži. Problem je **NP potpun (kompletan)** ako za svođenje bilo kog NP problema na posmatrani problem postoji polinomski algoritam. To znači da ako se za bilo koji NP potpun problem pronađe polinomski algoritam time se dokazuje postojanje polinomskog algoritma za svaki NP problem. Odnosno, pokazalo bi se da važi $P = NP$. U kontekstu optimizacije spominju se **NP teški problemi** - problemi čije su odlučive varijante NP potpuni problemi. Da bi se za neki novi problem pokazalo da je NP težak (kompletan) dovoljno je redukovati neki od postojećih NP teških (kompletnih) problema na njega, jer to znači da je novi problem težak bar koliko i taj koji je redukovan na njega. Postoje dva opšta pristupa za rešavanje problema:

- **egzaktno rešavanje** - postupci koji dovode do garantovano optimalnog rešenja ako se završe. Polinomski problemi se rešavaju egzaktno.
- **približno (aproksimativno) rešavanje** - postupci koji čak i kada završe ne garantuju optimalnost. Ovde se ubrajaju i metaheuristike i algoritmi sa garancijom kvaliteta. NP teški problemi se rešavaju približno.