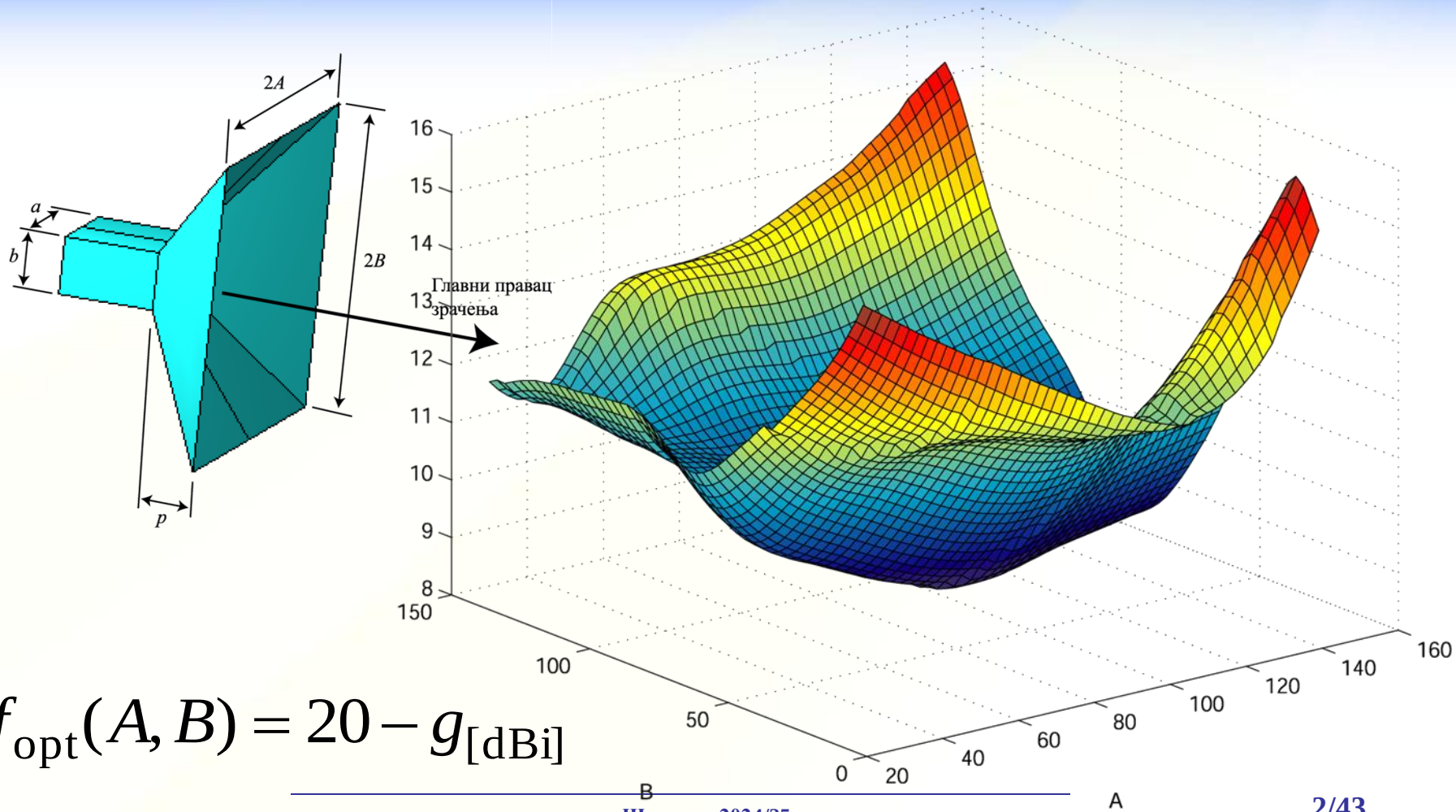


Оптимизациона функција

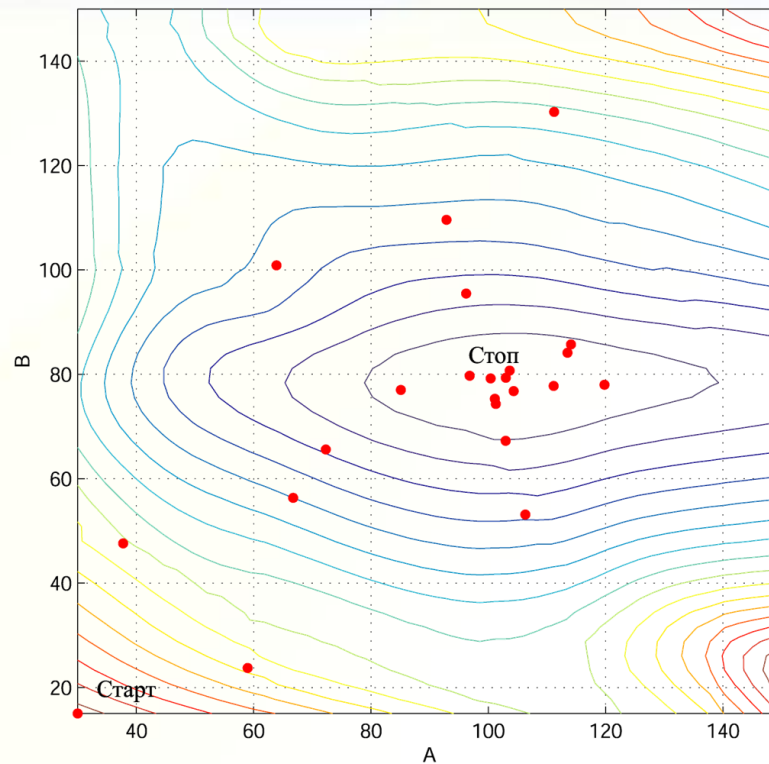
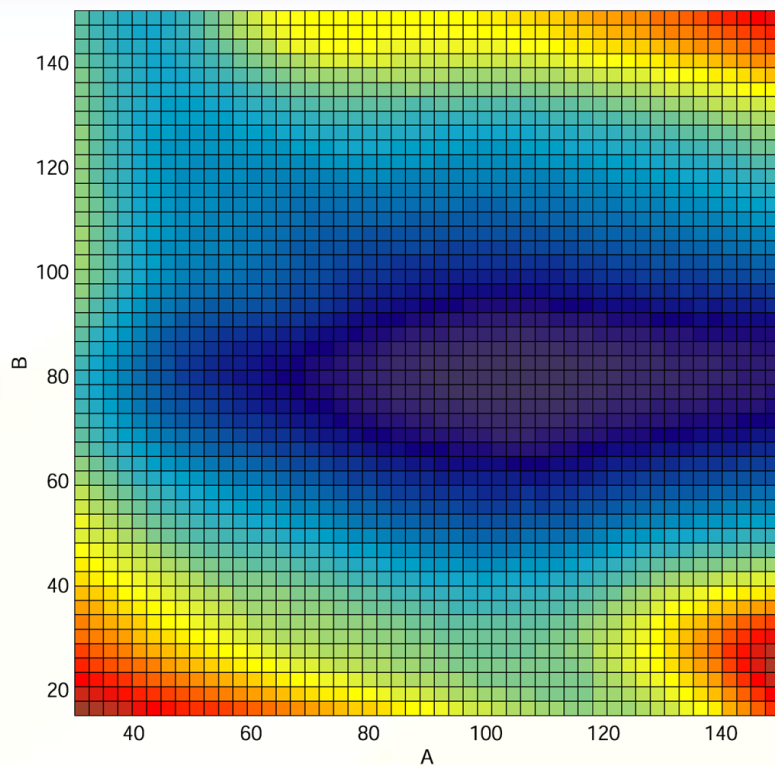
- Оптимизациону функцију, $f(\mathbf{x})$, дефинише онај ко решава проблем!
- Избор $f(\mathbf{x})$ може олакшати или отежати решавање
- Увек је потребно уградити СВЕ што знамо о проблему
- У неким ситуацијама постоји и шум
 - Нумеричка (ЕМ) анализа и грешка заокруживања
 - TSP и гужва у саобраћају
 - SAT и промењени бити услед грешака у преносу бита

Пример из праксе једне оптимизационе функције (NLP)

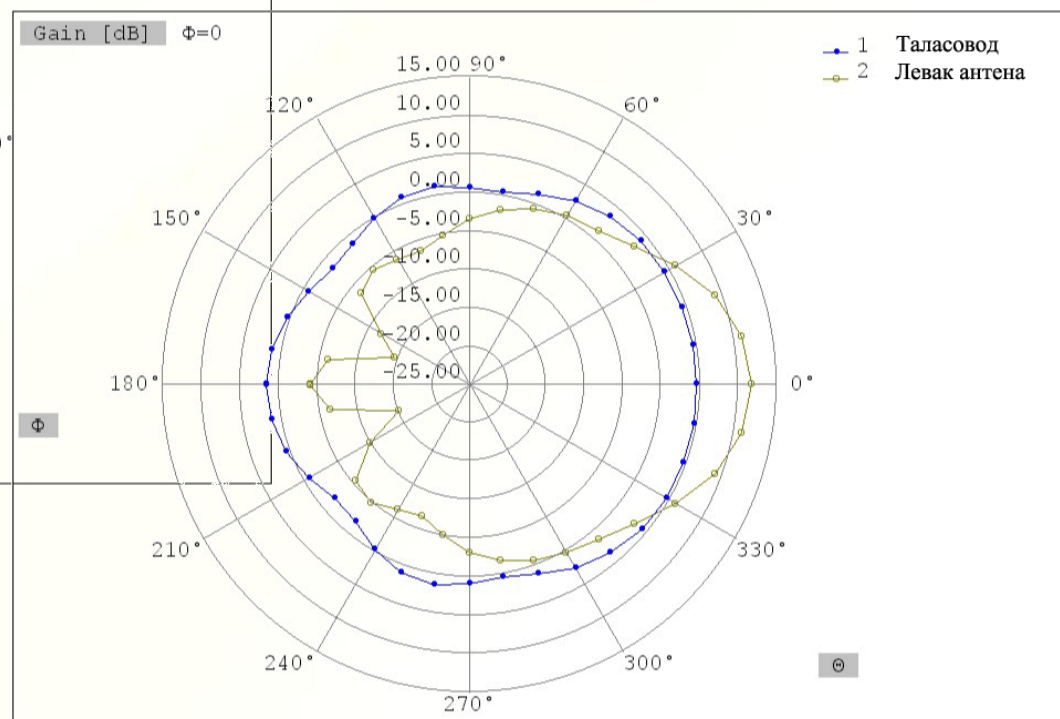
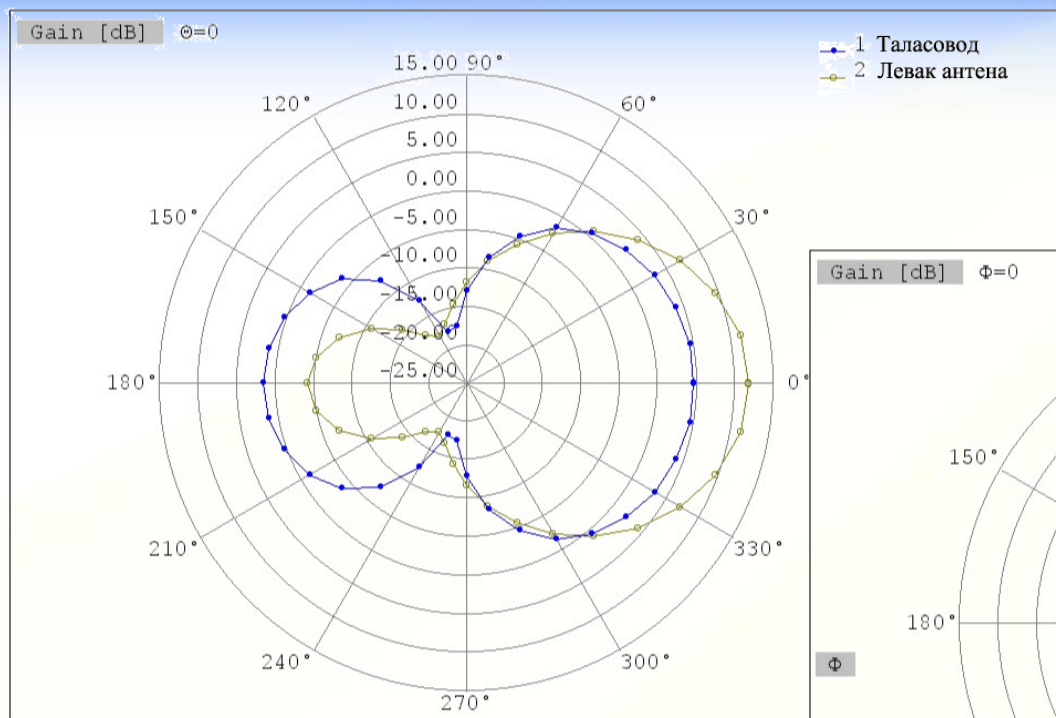


$$f_{\text{opt}}(A, B) = 20 - g[\text{dBi}]$$

Описна функција и оптимални отвор левак антене



Оптимална левак антена



Како проценити особине и перформансе опт. алгоритма?

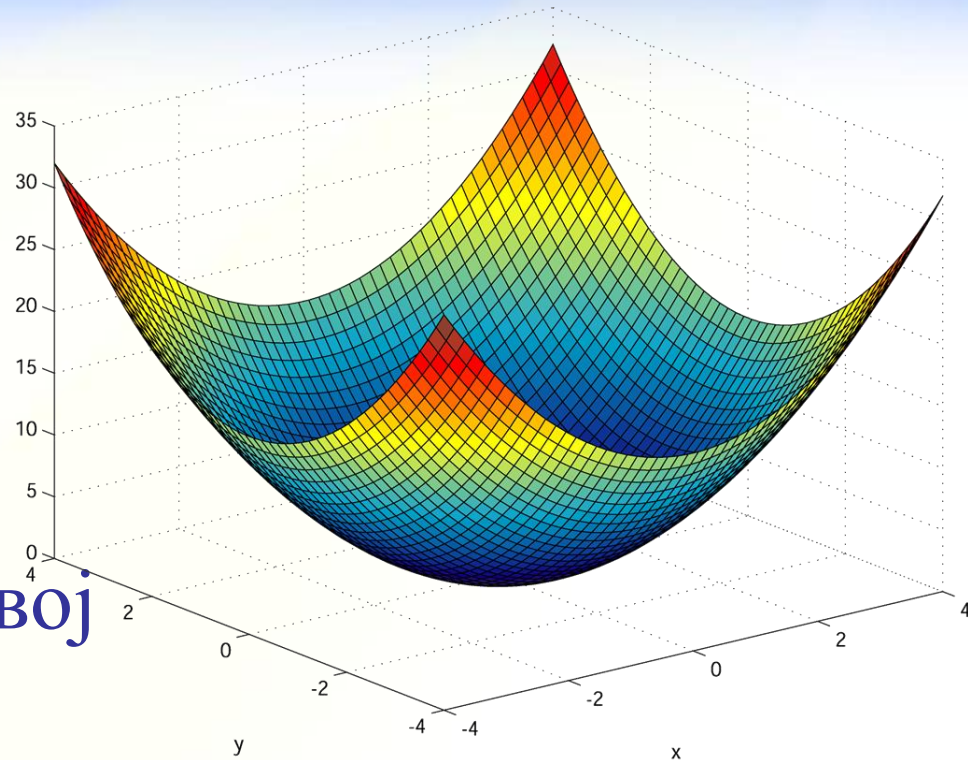
- Ситуација #1 (инжењерска пракса):
постављен је оптимизациони проблем и потребно је решити га
 - бирамо алгоритам на основу претходног искуства и информација које имамо о датом проблему
 - пробамо све доступне оптимизационе алгоритме и користимо оне који имају најбоље перформансе
- Ситуација #2 (истраживање / наука):
потребно је сагледати особине оптимизационог алгоритма
 - тестирамо алгоритме на познатим оптимизационим функцијама (проблемима) и поредимо перформансе оптимизационих алгоритма
- Потребне су нам познате оптимизационе функције (познати оптимизациони проблеми) са различитим особинама, када разматрамо алгоритме

Аналитичке (NLP) тест функције: сума квадрата

- Аналитички израз за D димензија

$$f(\mathbf{x}) = \sum_{i=1}^D x_i^2$$

- Једноставна: мин. по свакој променљивој
- Минимум познат: координатни почетак (или нуле полинома)

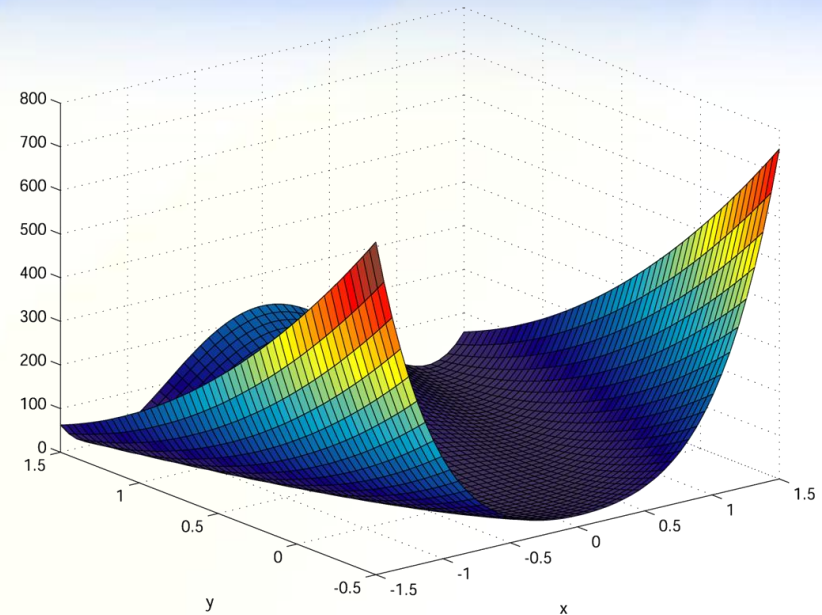


Розенброкова функција

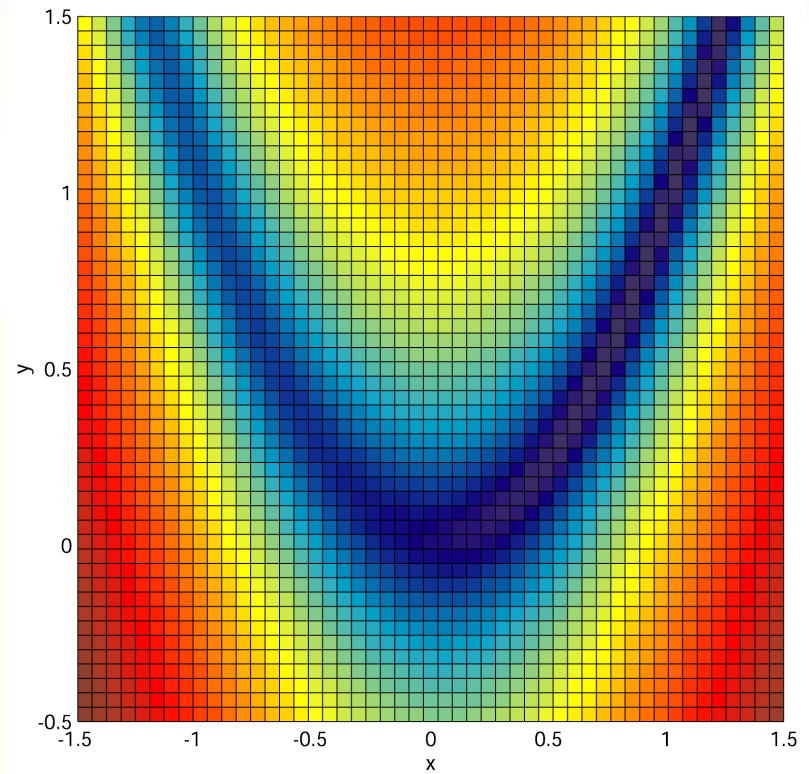
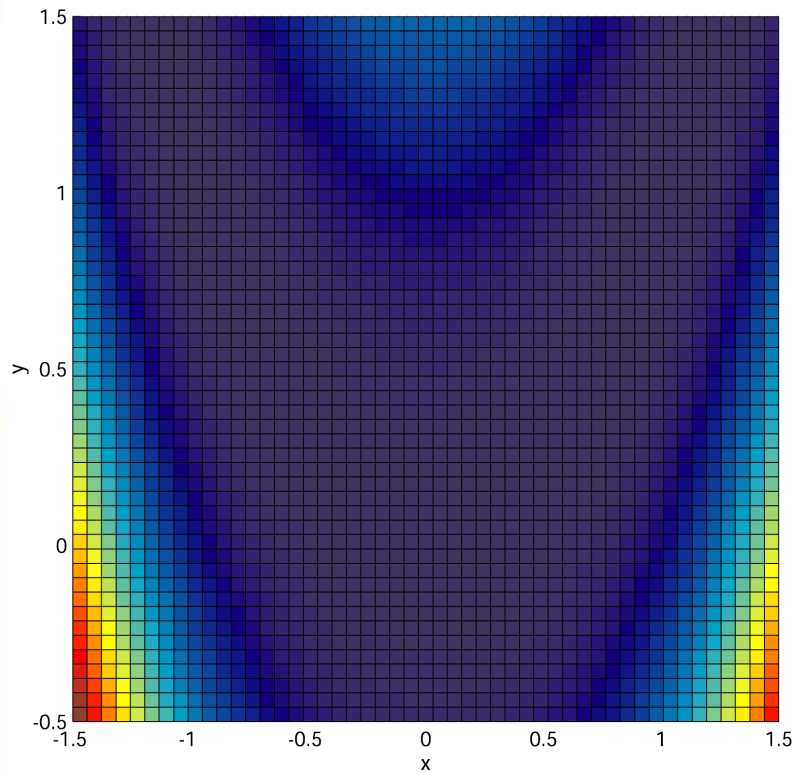
- Аналитички израз за D димензија

$$f(\mathbf{x}) = \sum_{i=1}^{D-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \right)$$

- Глобални минимум у тачки $(1, 1, \dots, 1)$
- Број локалних минимума за функцију са више од две димензије није познат
- Спрега између променљивих



Розенброкова функција: поглед одозго и лог-размера

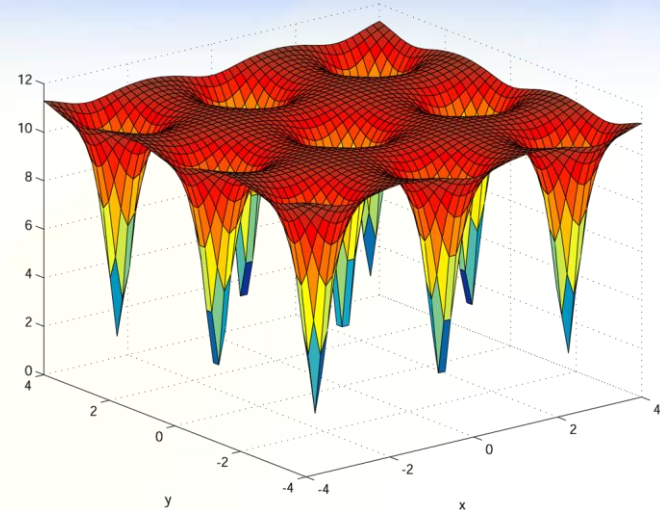


Шекелова функција (Fox-hole function)

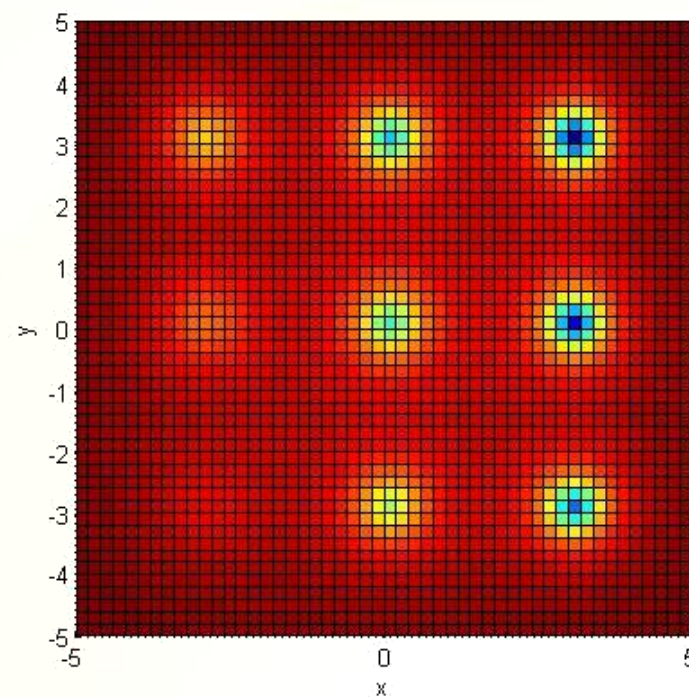
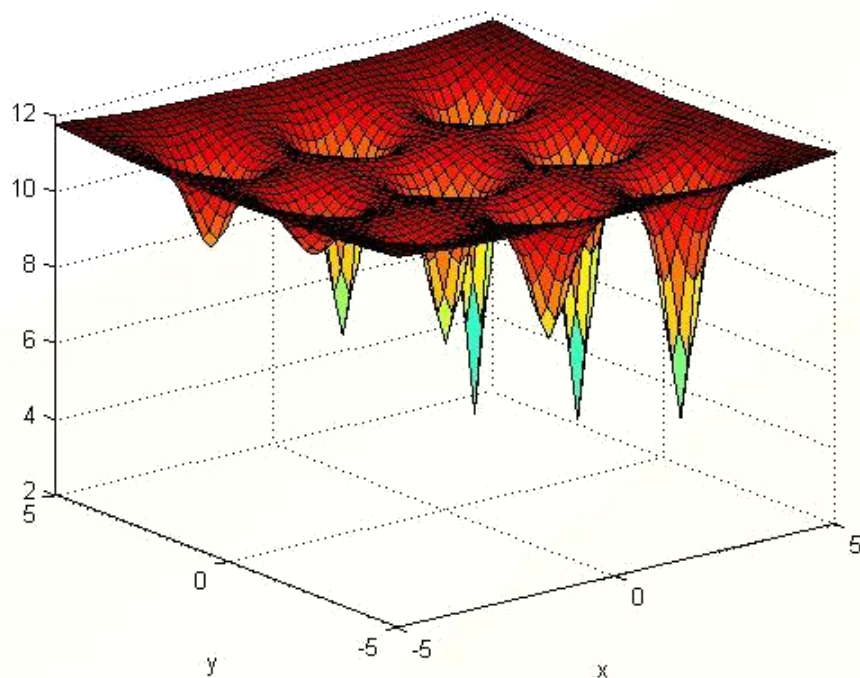
- Аналитички израз

$$f(\mathbf{x}) = K - \sum_{i=1}^M \left((\mathbf{x} - \mathbf{a}_i)^T (\mathbf{x} - \mathbf{a}_i) + c_i \right)^{-1}$$

- $\mathbf{x}$ је D -димензиони вектор
- $\mathbf{a}_i$ је вектор са координатама i -тог локалног минимума
- c_i је константа обрнуто пропорционална дубини i -тог локалног минимума
- M је укупан број минимума функције
- K је константа која контролише вредност функције између минимума



Минимуми различитих дубина



G2 функција

- Максимизација функције

$$G2(\mathbf{x}) = \left| \frac{\sum_{i=1}^D \cos^4(x_i) - 2 \prod_{i=1}^D \cos^2(x_i)}{\sqrt{\sum_{i=1}^D i \cdot x_i^2}} \right|$$

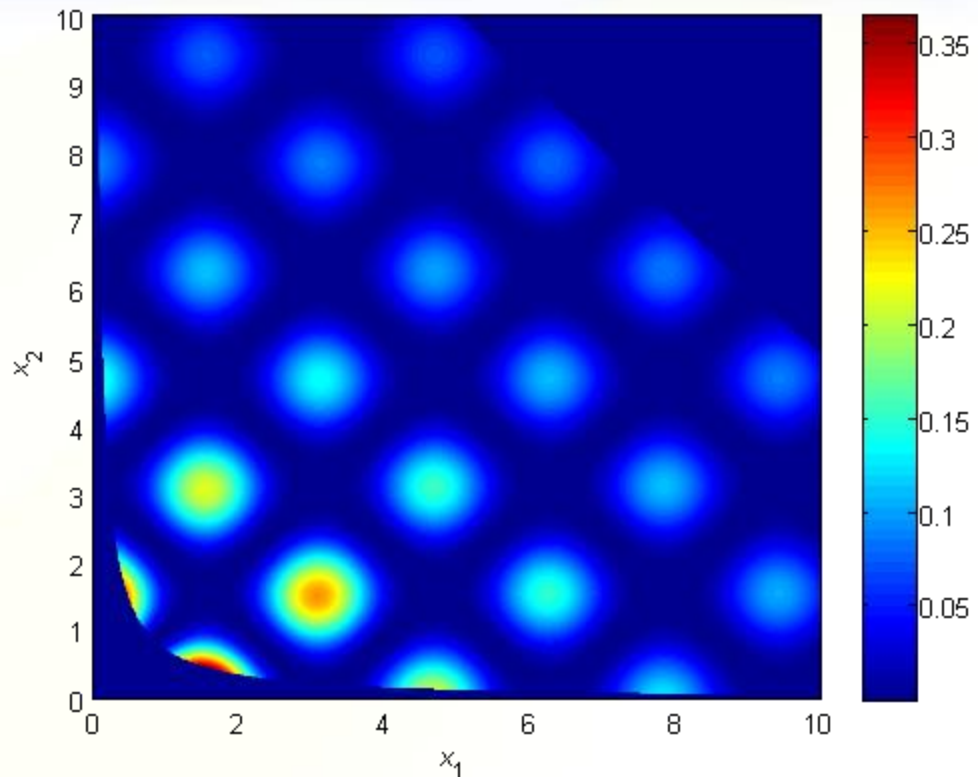
- Под условима:

$$\prod_{i=1}^D x_i \geq 0,75 \quad \sum_{i=1}^D x_i \leq 7,5D \quad 0 \leq x_i \leq 10, \quad i \in 1, 2, \dots, D$$

- Нула ако нису испуњени услови

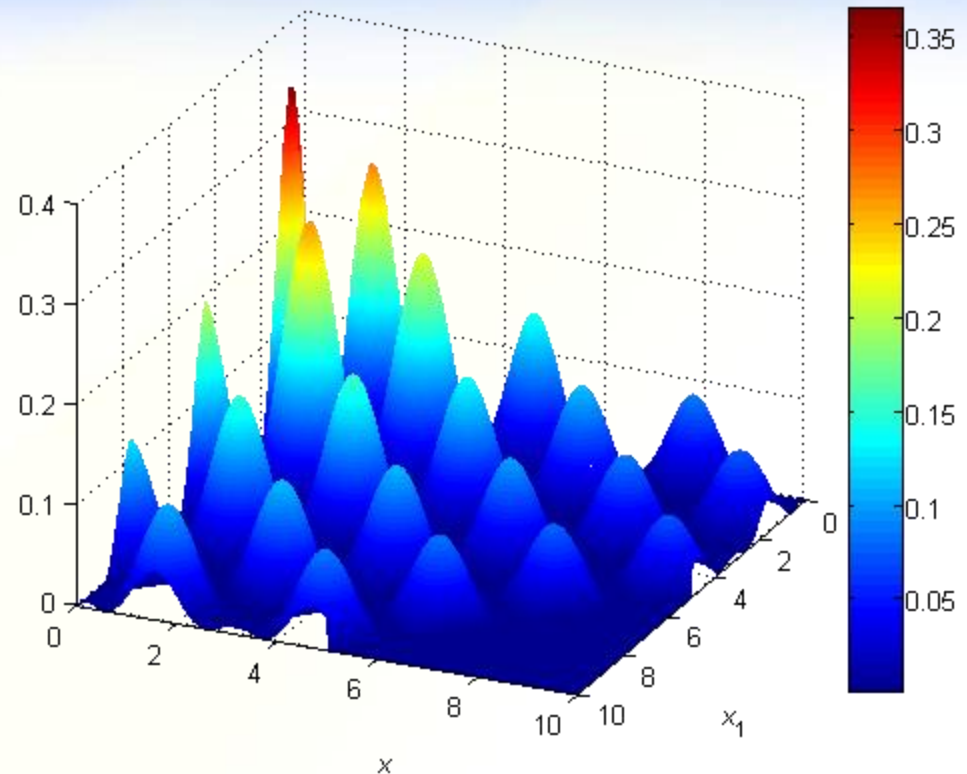
Функција G2 са 2 променљиве

- Глобални максимум непознат у општем случају
- Око координатног почетка



G2: изразито тешка функција за оптимизацију

- Изразито “брдовит” терен
- Врло незгодна функција за оптимизацију
- Нотација, друга од 11 тешких функција



Примери дискретних функција:

f_1 врло тешка, f_2 лакше решива

$$f_1(x_1, x_2, x_3, x_4) = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \qquad f_2(x_1, x_2, x_3, x_4) = \sum_{k=1}^4 x_k$$

x_1	x_2	x_3	x_4	f_1	f_2
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	2
0	1	0	0	0	1
0	1	0	1	0	2
0	1	1	0	0	2
0	1	1	1	0	3
1	0	0	0	0	1
1	0	0	1	0	2
1	0	1	0	0	2
1	0	1	1	0	3
1	1	0	0	0	2
1	1	0	1	0	3
1	1	1	0	0	3
1	1	1	1	1	4

Формирање оптимизационе функције и норме

- Оптимизациона функција је нека функција разлике величина које описују решења
- Примери:
 - разлика између перформанси жељеног и пројектованог електричног уређаја
 - разлика између мерених резултата и функције која их фитује
 - разлика између жељене и добијене цене производа
 - разлика између жељеног и оствареног управљања
 - итд.
- Да ли узети апсолутну вредност разлике, квадрат разлике или неки други степен?
- У оптимизационој функцији у пракси често се појављује **норма** разлике жељеног и оствареног

Норма: дефиниција

- За задати векторски простор V , норма је функција $f: V \rightarrow \mathbb{R}$, која задовољава следеће услове $V: \mathbf{x}, \mathbf{y}, \mathbb{R}: a$
 - $f(a\mathbf{x}) = |a| f(\mathbf{x})$
 - $f(\mathbf{y} + \mathbf{x}) \leq f(\mathbf{y}) + f(\mathbf{x})$
 - $f(\mathbf{x}) = 0 \iff \mathbf{x} = 0$
- Из претходних услова следи
 - $f(\mathbf{x}) \geq 0$

Норма: означавање и примери

- $\mathbf{x}$ је D димензиони вектор разлике између жељеног и пронађеног решења (оптимизација)
- Норма се најчешће означава као

$$\|\mathbf{x}\| := f(\mathbf{x})$$

- L_1 (taxicab, Manhattan norm)
растојање које такси прелази на мрежи

$$\|\mathbf{x}\|_1 := \sum_{k=1}^D |x_k|$$

Норма: примери и генерализација

- L_2 (Еуклидовска норма, растојање)

$$\|\mathbf{x}\|_2 := \sqrt{\sum_{k=1}^D |x_k|^2}$$

- Генерализација: L_p $\|\mathbf{x}\|_p := \left(\sum_{k=1}^D |x_k|^p \right)^{\frac{1}{p}}, p \geq 1$

- Maximum norm $\|\mathbf{x}\|_\infty := \max\{|x_1|, |x_2|, \dots, |x_D|\}$

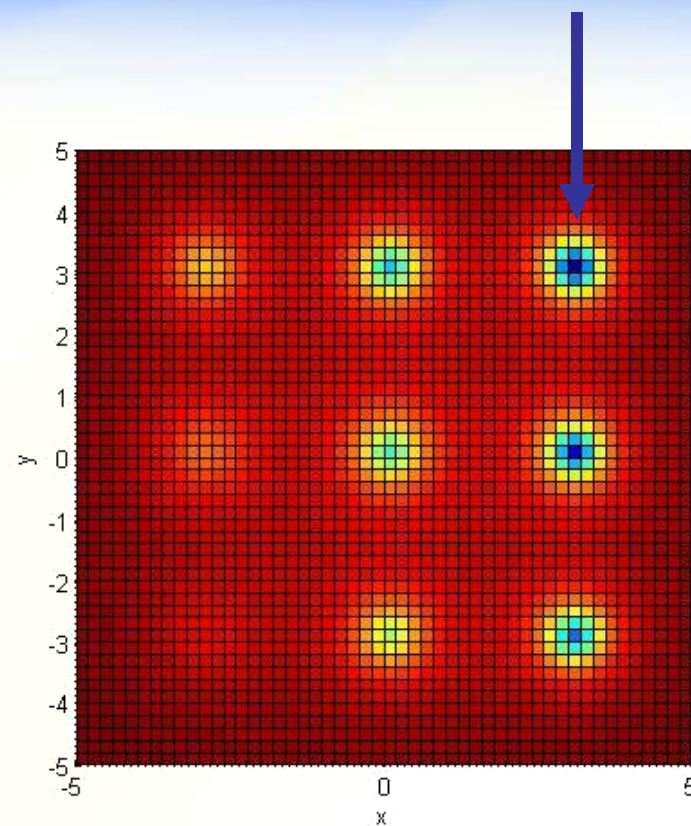
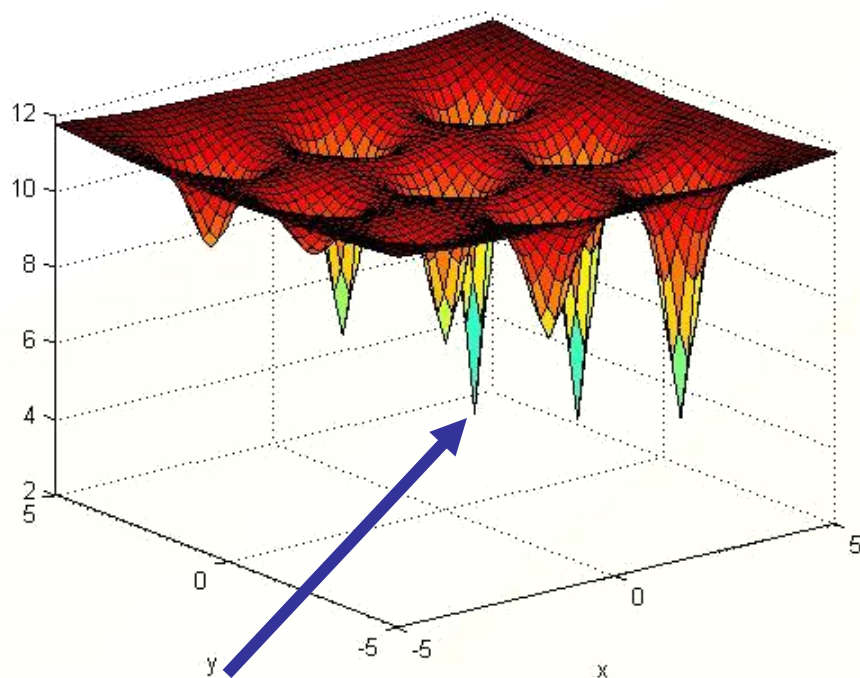
Дефиниција оптимизационог проблема са једним критеријумом

- Пронаћи минимум функције f
 $\text{minimize}\{ f(\mathbf{x}) \}$
 $\mathbf{x} = (x_1, x_2, \dots, x_D)$
 $x_k \in R \quad k = 1, 2, \dots, D$
- Под условима
 $\phi_k(\mathbf{x}) = 0 \quad k = 1, 2, \dots, E$
 $\psi_k(\mathbf{x}) \leq 0 \quad k = 1, 2, \dots, L$
- Минимизација и максимизација: подразумевамо минимизацију, уколико се не нагласи другачије
- Еквиваленција: $\text{minimize}\{ f(\mathbf{x}) \} = \text{maximize}\{ -f(\mathbf{x}) \}$

Глобални минимум: формална дефиниција

- Оптимизациони простор S
- Домен (смислених) решења F , $F \subseteq S$
- Оптимизациона функција $f(\mathbf{x})$
- Глобални минимум, $\mathbf{x}_g$, опт. проблема је
$$\forall \mathbf{y} \in F, \quad f(\mathbf{x}_g) \leq f(\mathbf{y})$$
- Глобални минимум је најбоље могуће решење
- Да ли постоји само један глобални минимум?
 - Може постојати један или више глобалних минимума

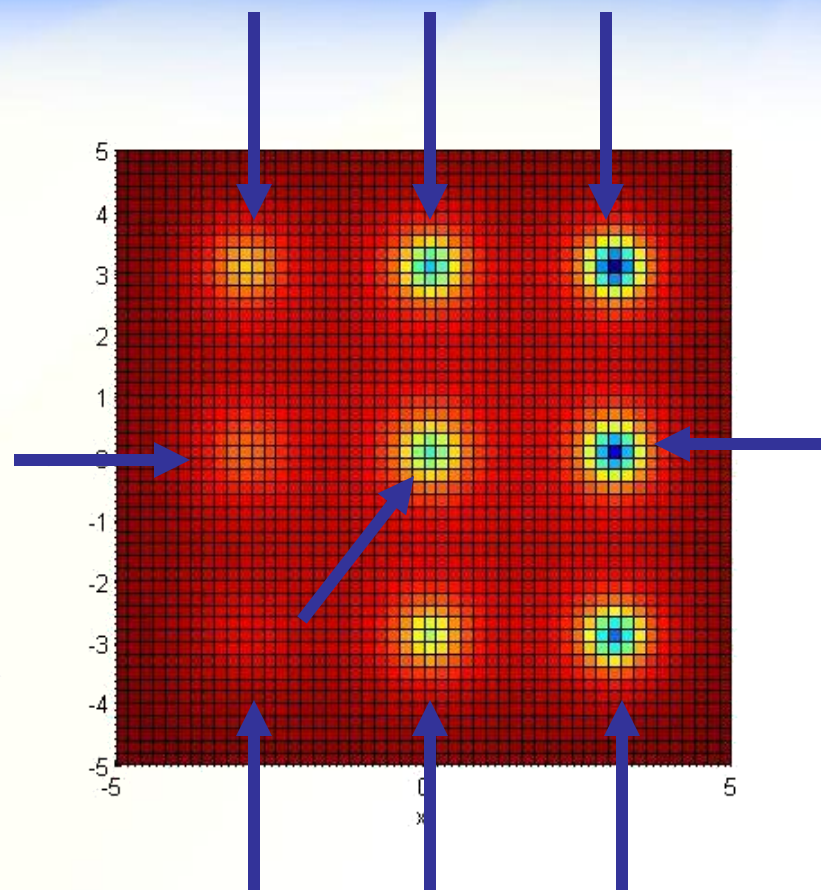
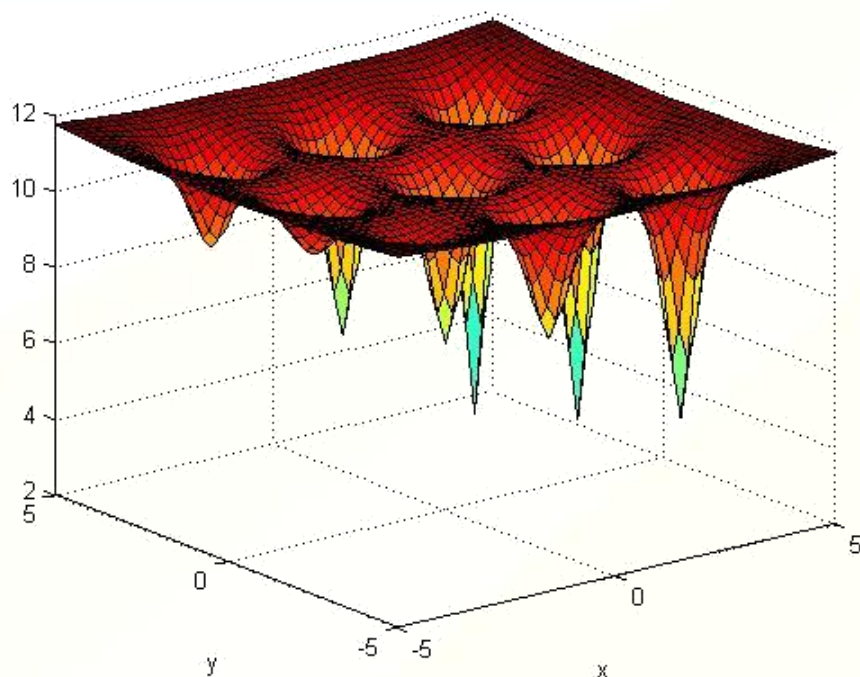
Пример глобалног минимума



Локално решење: формална дефиниција

- Епсилон околина решења $\mathbf{x}$ је $N(\mathbf{x}) = \{\mathbf{y} \in S, \text{dist}(\mathbf{x}, \mathbf{y}) \leq \varepsilon\}$
- За NLP класу проблема:
Еуклидовско растојање
$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^D (x_k - y_k)^2}$$
- За SAT класу проблема:
Хамингово растојање
$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^D x_k \oplus y_k$$
- За TSP класу проблема:
број различитих прелазака
$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D-1} \begin{cases} 1, & x_k \leftrightarrow x_{k+1} \in \mathbf{y} \\ 0, & x_k \leftrightarrow x_{k+1} \notin \mathbf{y} \end{cases}$$
- $\text{dist}(\mathbf{x}, \mathbf{y})$ може да се дефинише на много начина!
- Локални минимум $\mathbf{x}$, у околини $N(\mathbf{x})$:
$$f(\mathbf{x}) \leq f(\mathbf{y}), \forall \mathbf{y} \in N(\mathbf{x})$$

Примери локалних минимума (зависи од околине!)



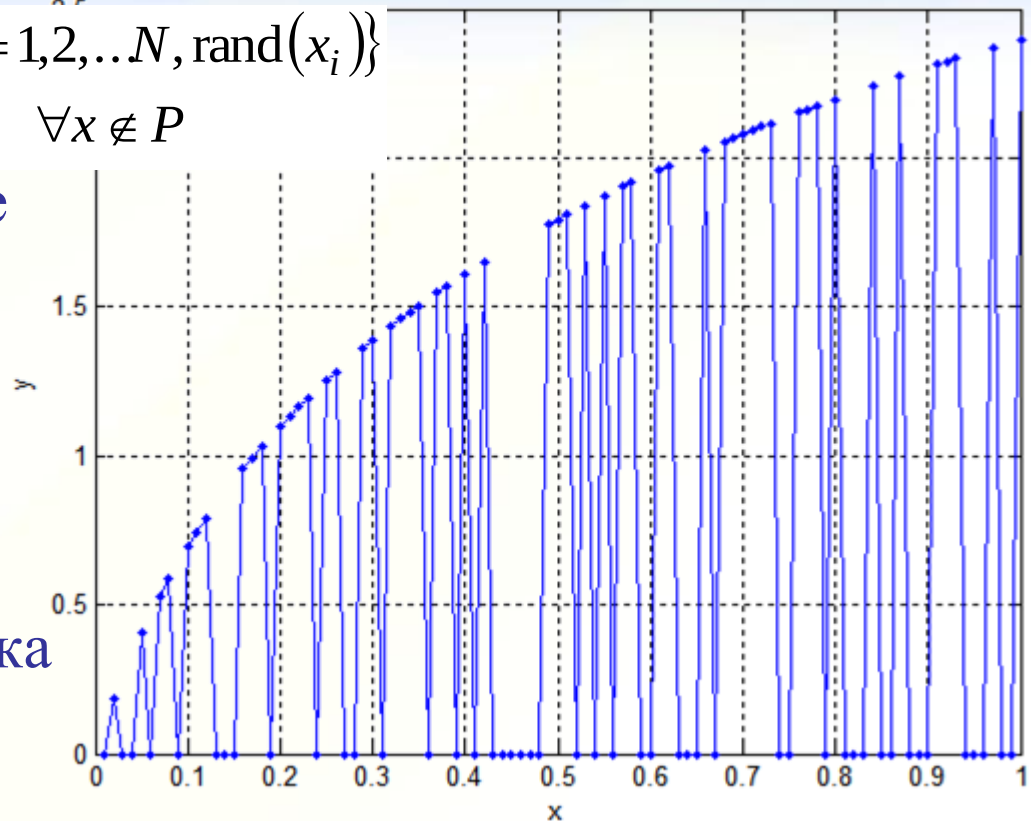
NFL теорема

- NO-FREE-LUNCH theorem
 - Wolpert D.H.; Macready W.G. "No free lunch theorems for optimization " *Evolutionary Computation IEEE Transactions on* vol.1 no.1 pp. 67-82 Apr 1997
 - уколико су све оптимизационе функције једнако вероватне
сви опт. алгоритми су једнако (не)ефикасни
- Зашто бисмо онда разматрали различите оптимизационе алгоритме?

Пример функције уз NFL теорему

$$f(x) = \begin{cases} 0, & P = \{i = 1, 2, \dots, N, \text{rand}(x_i)\} \\ \ln(1 + 10x), & \forall x \notin P \end{cases}$$

- Пример мало вероватне оптимизационе функције
- Овакав проблем није добро дефинисан
- Пример: оптимизација линка за пренос података који је недоступан у тренуцима који су случајно генерисани

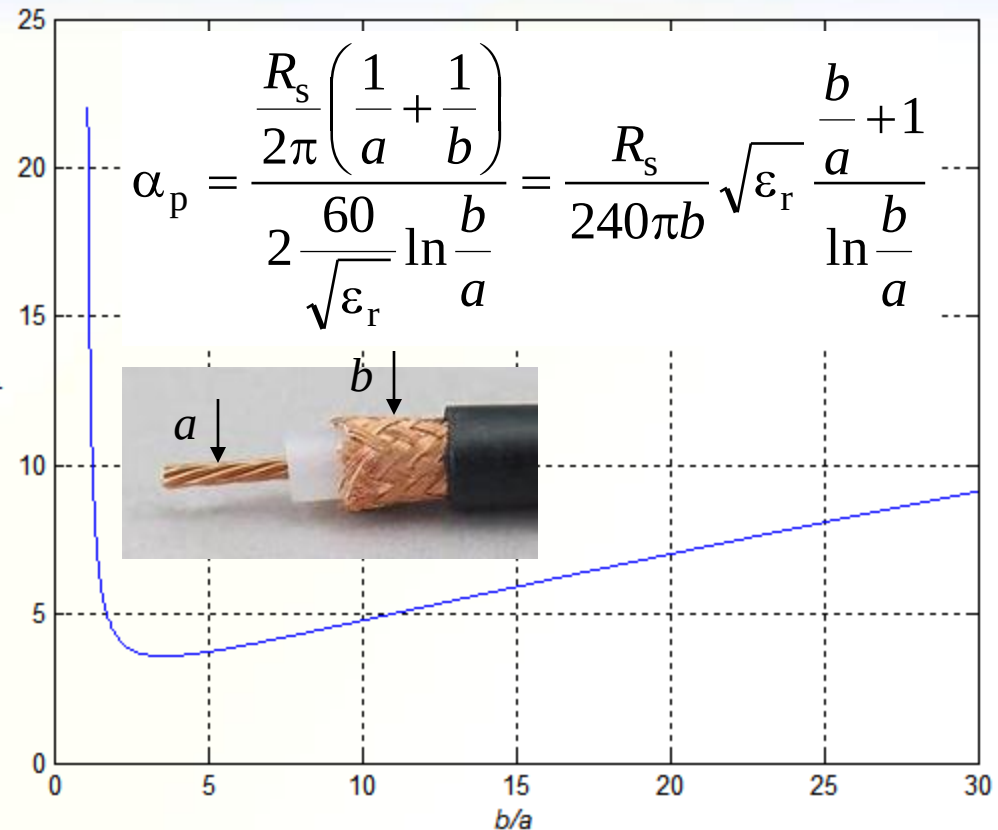


Критички осврт на NFL теорему (познати докази када НЕ важи)

- D.W. Corne J.D. Knowles “Some multiobjective optimizers are better than the others ” *Proc. of IEEE CEC 2003* pp. 2506-2512.
 - For multiobjective optimization NFL does not hold
- C. Igel M. Toussaint “A No-Free-Lunch theorem for non-uniform distributions of target functions ” *J. Math. Model. Algorithms* 3(4) pp. 313-322 (2004)
 - Classes of functions relevant in practice are not likely to satisfy the NFL scenario
- A. Auger O. Teytaud “Continuous lunches are free plus the design of optimal optimization algorithms ” *Algorithmica* 57 2010 pp. 121-146.
 - For continuous domains NFL does not hold

Пример инжењерског проблема: минимизација слабљења кабла

- Коаксијални кабл
 - полупречници a и b
 - површинска отпорност R_s
 - задато b (габарит)
- Пронаћи b/a тако да је α_p минимално
- Непрекидна и диференцијабилна опт. функција
- Постоји јасна узрочно-последична зависност побуде (b/a) и одзива (α_p)



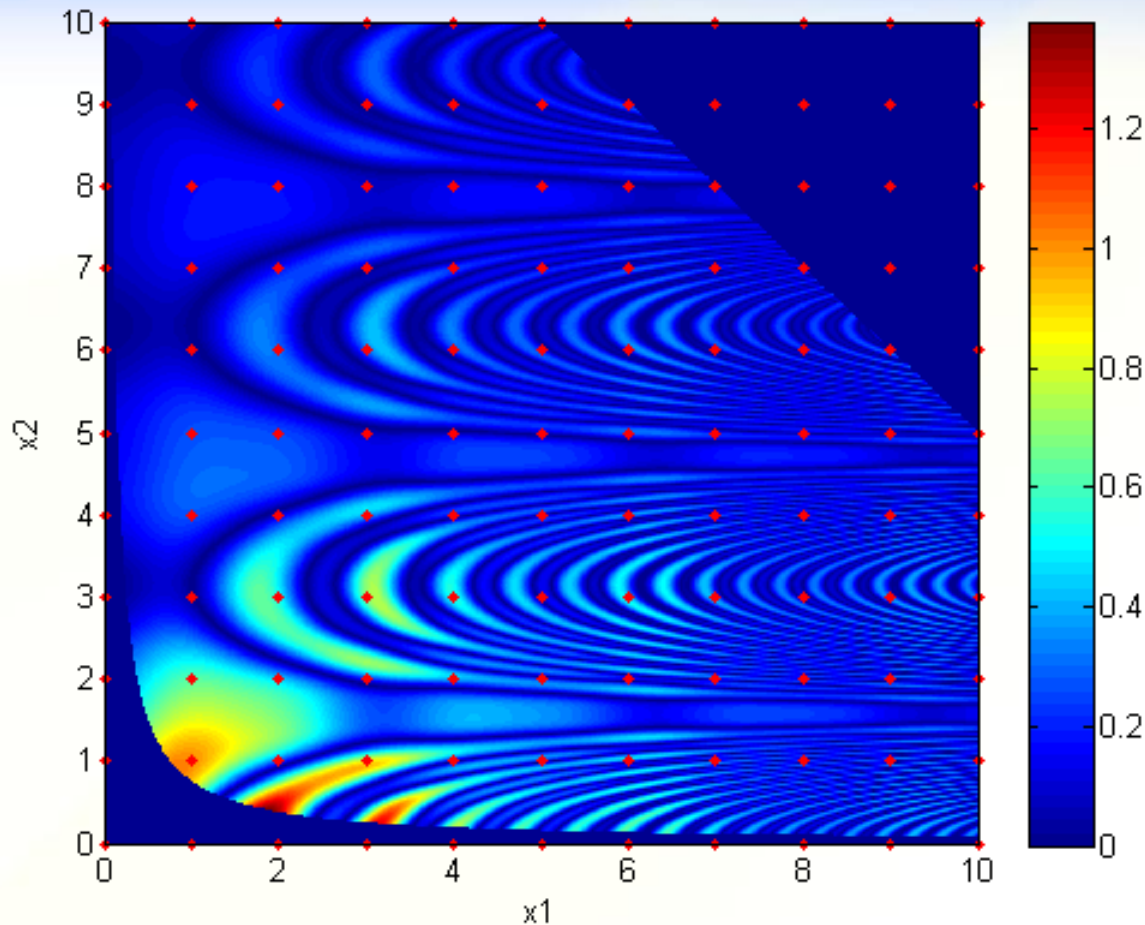
Инжењерска пракса: неки алгоритми су БОЉИ

- Пракса:
 - оптимизациона функција је смислено формулисана
 - оптимизациони проблем је једнозначан
 - поновљивост везе улаз-излаз “црне кутије”
- Инжењерски проблем подразумева **узрочно-последичну везу**
 - Неке оптимизационе функције су много вероватније од других
- У инжењерској пракси:
неки оптимизациони алгоритми су знатно бољи од других алгоритама
 - У зависности од проблема који се решава

Систематско претраживање: континуалне променљиве

- За сваки оптимизациони параметар задаје се број корака (или дужина)
- Вишедимензионална мрежа се формира у оптимизационом простору
- У сваком чвору мреже израчунава се оптимизациона функција
- За D -димензионални оптимизациони простор укупан број итерација је $k_1 * k_2 * \dots * k_D$ где је k_i број корака по димензији i
- Сложеност алгоритма је $O(N^D)$
 N број корака по једној димензији
 D број димензија оптимизационог простора

Систематско претраживање једне континуалне функције



Систематско претраживање: дискретне променљиве

- Проверити СВА дискретна стања у оптимизационом простору
- Примери:
 - испитати сва дискретна стања прекидача
 - испитати све варијације са понављањем
 - испитати све комбинације без понављања
 - испитати све пермутације без понављања
 - ...
- Истинитосне таблице при испитивању таутологије у логици

Варијације са понављањем (Систематско претраживање SAT)

- На колико начина се може распоредити n елемената на k места?
- Број начина распоређивања је n^k
- Пун назив: варијација са понављањем k -те класе скупа X_n (Енг: n -tuples)
- Формирање као број са k цифара а цифре су $1\dots n$:

1111	1112	...	111 n
1121	1122	...	112 n
...			
$nnn1$	$nnn2$	...	$nnnn$

```
void variations_with_repetition(int n, int k)
{
    int q;
    int *P = new int [k];
    for(int i=0; i<k ;i++)
        P[i]=0;

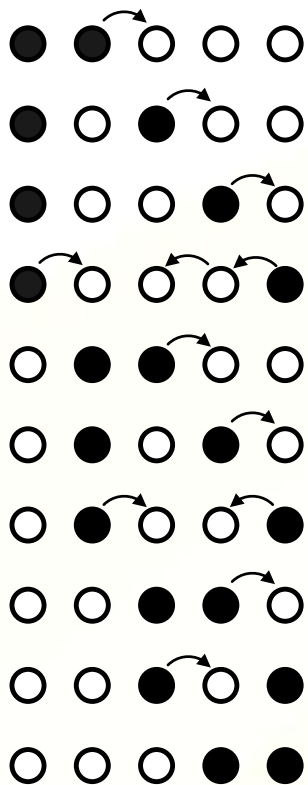
    do
    {
        for(int i=0; i<k; i++)
            printf("%5d ",P[i]+1);
        printf("\n");

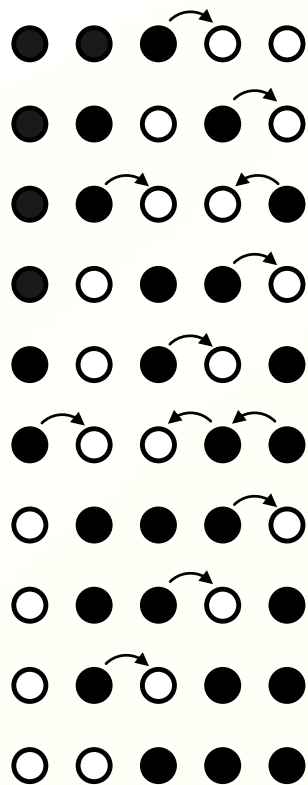
        q = k-1;
        while (q >= 0)
        {
            P[q]++;
            if (P[q] == n)
            {
                P[q] = 0;
                q--;
            }
            else
                break;
        }
    } while (q >= 0);
    delete [] P;
}
```


Комбинације без понављања

- Подскуп од k елемената бирамо из скупа од n елемената (редослед у подскупу од k елемената није битан)
- Број могућих избора је $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, $0 \leq k \leq n$, $0! = 1$
- Пун назив: комбинација k -те класе скупа X_n (енг: k -combinations)
- Основа за имплементацију:
 - Поставити k елемената на прва слободна места са леве стране
 - Померити крајњи десни удесно за по једно место док може
 - Када више нема слободних места десно померити први слободни елемент са десне стране за једно место десно, а остале десно од њега ставити иза њега редом
 - Поновити процедуру док сви елементи не буду десно

Илустрација и C/C++ имплементација

$$\begin{pmatrix} 5 \\ 2 \end{pmatrix}$$


$$\begin{pmatrix} 5 \\ 3 \end{pmatrix}$$


```

1  #include <stdio.h>
2
3  void combinations_without_repetition(int n, int k)
4  {
5      int i,j;
6      bool b;
7      int *P = new int[k];
8
9      for (i = 0; i < k; i++)
10         P[i] = i+1;
11
12     do
13     {
14         for (i = 0; i < k; i++)
15             printf("%3d ", P[i]);
16         printf("\n");
17
18         b = false;
19         for (i = k-1; i >= 0; i--)
20         {
21             if (P[i] < n - k + 1 + i)
22             {
23                 P[i]++;
24                 for (j = i + 1; j < k; j++)
25                     P[j] = P[j - 1] + 1;
26
27                 b = true;
28                 break;
29             }
30         }
31     } while (b);
32
33     if(P!=NULL) delete[] P;
34 }
    
```

Пермутације без понављања (Систематско претраживање TSP)

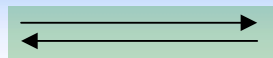
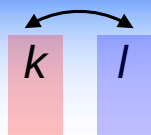
- На колико начина је могуће распоредити n елемената на n места (без понављања елемената)?
- Број различитих распореда је $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$
- Пун назив: пермутације скупа X_n (енг: permutation of set)
- Алгоритам изабран по критеријуму **најбржег извршавања** и **исписивања пермутација у лексикографском поретку**

1	2	3	4	3	1	2	4
1	2	4	3	3	1	4	2
1	3	2	4	3	2	1	4
1	3	4	2	3	2	4	1
1	4	2	3	3	4	1	2
1	4	3	2	3	4	2	1
2	1	3	4	4	1	2	3
2	1	4	3	4	1	3	2
2	3	1	4	4	2	1	3
2	3	4	1	4	2	3	1
2	4	1	3	4	3	1	2
2	4	3	1	4	3	2	1

Алгоритам: основна идеја

- Полазимо од пермутације
 $P = \{P[0] \ P[1] \ \dots P[n - 1]\}$
- Пронаћи највеће k ($0 \leq k < n - 1$)
тако да је $P[k] < P[k+1]$
- Пронаћи највеће l ($l > k$ и $l \leq n - 1$)
тако да је $P[l] > P[k]$
- Заменити вредности $P[l]$ и $P[k]$
- Обрнути редослед елемената
 $P[k+1] \ P[k+2] \ \dots P[n - 1]$

Алгоритам: илустрација



1	2	3	<i>k</i> 4	<i>l</i> 5	→	1	2	3	5	4	→	1	2	3	5	4
1	2	<i>k</i> 3	5	<i>l</i> 4	→	1	2	4	5	3	→	1	2	4	3	5
1	2	4	<i>k</i> 3	<i>l</i> 5	→	1	2	4	5	3	→	1	2	4	5	3
1	2	<i>k</i> 4	<i>l</i> 5	3	→	1	2	5	4	3	→	1	2	5	3	4
1	2	5	<i>k</i> 3	<i>l</i> 4	→	1	2	5	4	3	→	1	2	5	4	3
1	<i>k</i> 2	5	4	<i>l</i> 3	→	1	3	5	4	2	→	1	3	2	4	5
1	3	2	<i>k</i> 4	<i>l</i> 5	→	1	3	2	5	4	→	1	3	2	5	4

Имплементација у С/С++

```
1 #include <stdio.h>
2 int next_permutation(const int N, int *P)
3 {
4     int s;
5     int* first=&P[0];
6     int* last=&P[N-1];
7     int* k=last-1;
8     int* l=last;
9     // find largest k so that P[k]<P[k+1]
10    while(k>first)
11    {
12        if(*k<*(k+1))
13        {
14            break;
15        }
16        k--;
17    }
18    // if no P[k]<P[k+1], P is the last permutation in lexicographic order
19    if(*k>*(k+1))
20    {
21        return 0;
22    }
23    // find largest l so that P[k]<P[l]
24    while(l>k)
25    {
26        if(*l>*k)
27        {
28            break;
29        }
30        l--;
31    }
32    // swap P[l] and P[k]
33    s = *k;
34    *k = *l;
35    *l = s;
36    // reverse the remaining P[k+1]...P[N-1]
37    first=k+1;
38    while(first<last)
39    {
40        s=*first;
41        *first=*last;
42        *last=s;
43
44        first++;
45        last--;
46    }
47    return 1;
48 }
49 }
```

```
51 void main(void)
52 {
53     int N=5;
54     int* P=new int [N];
55
56     //initialize the first permutation
57     for(int i=0; i<N; i++)
58         P[i]=i+1;
59
60     do
61     {
62         for(int i=0; i<N; i++)
63             printf("%2d ", P[i]);
64         printf("\n");
65     }while(next_permutation(N, P));
66
67     delete [] P;
68 }
```


Колико је брз алгоритам?

- Колико највише пермутација може да се изврши за:
 - 1 min, а колико за
 - 60 min?
- Колико времена је потребно за $N = 20$?
($20! \approx 2,4 \cdot 10^{18}$)
- Да ли постоји бржи алгоритам?

Друге имплементације

- **C/C++**

```
#include <algorithm>
std::next_permutation(...)
```

- **Python**

```
1 from itertools import permutations
2 perm = permutations([1, 2, 3, 4])
3 for i in list(perm):
4     print(i)
```

- **MATLAB**

```
perms([1 2 3 4])
```

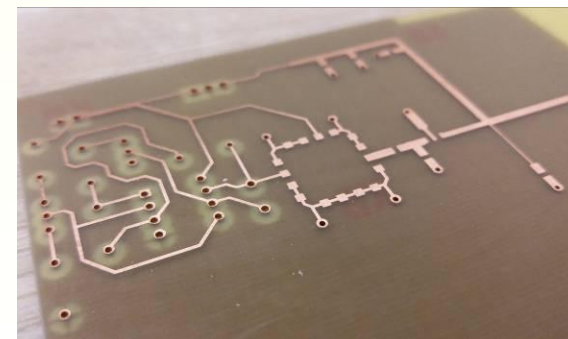
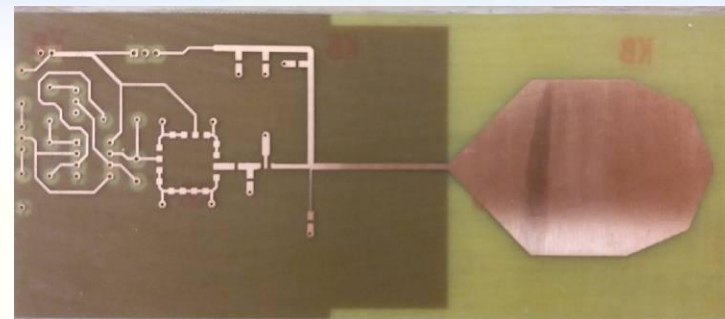
- **Wolfram Mathematica**

```
Permutations[{a b c}]
```

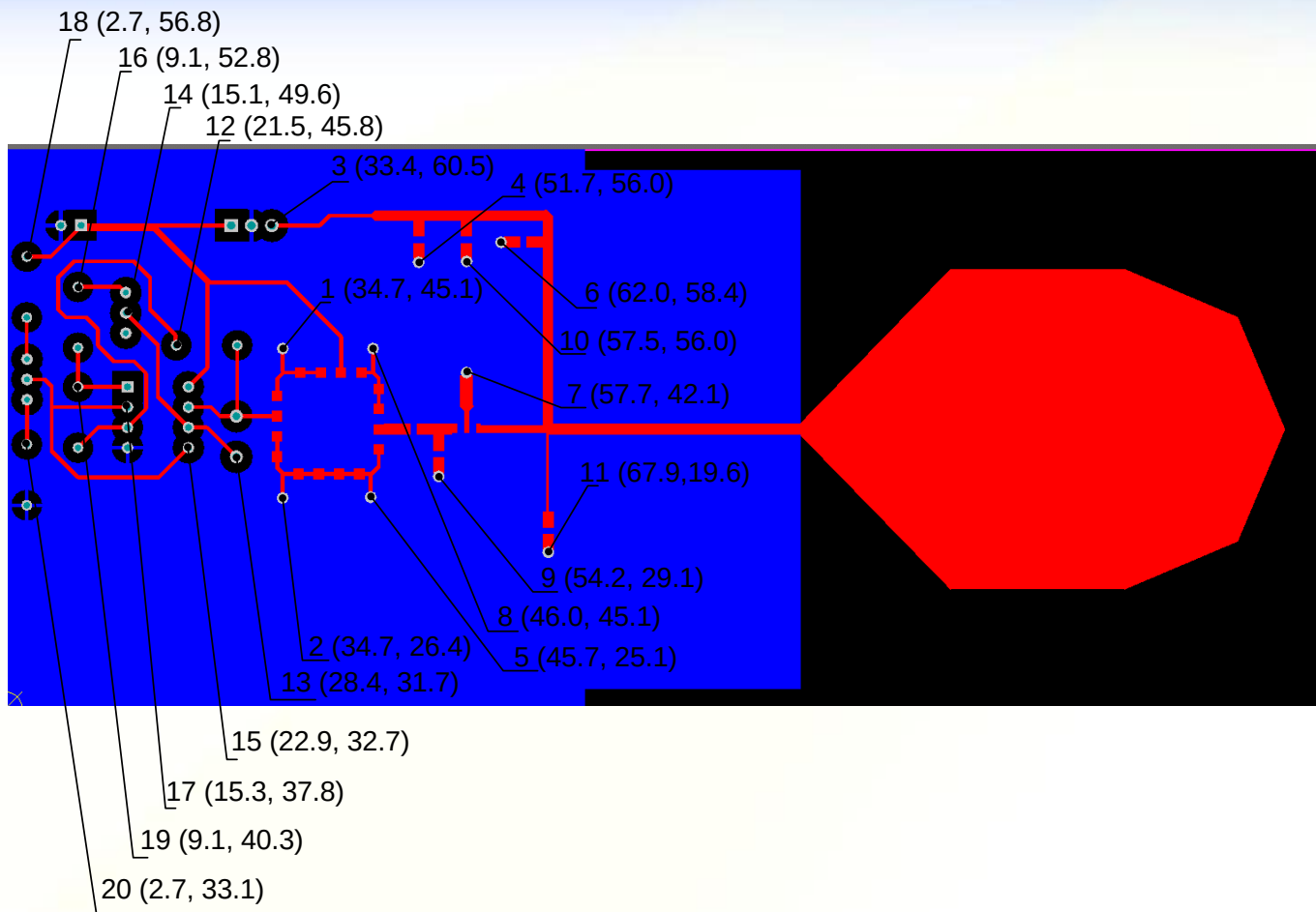
Задатак за вежбе

(Оптимално бушење плочице)

- Штампана плочица се израђује аутоматски
- На њој постоји више рупа истог пречника које је потребно избушити
- Аутоматска бушилица има коначну брзину позиционирања
- Израђује се велики број идентичних плочица
- Уштеда времена израде једне плочице је од великог практичног интереса (множи се бројем плочица у серији)
- Потребно је минимизирати пут који пређе бушилица (и минимизирати време израде)
- Бушилица се не враћа на место полазне рупе када заврши један циклус бушења (у пракси враћа се истом путањом уназад)
- Координате рупа на слици су дате у милиметрима



Координате тачка у [mm] на штампаној плочици



Задатак за вежбе (деталји и израда)

- Пронаћи дужину најкраћег пута и записати редослед обилазака рупа за (а) првих 8 и (б) првих 12 рупа
- Задатак се своди на TSP проблем са
 - 8 “градова” ($8! = 40\,320$ путања)
 - 12 “градова” ($12! = 479\,001\,600$ путања)
- Растојање између два рупе је L_2 норма
- Бушилица може да крене из било које рупе (од првих 8 (а) или 12 (б))
- У код за формирање свих пермутација додати
 - рачунање дужине путање
 - поређење са претходно нађеном најкраћом путањом и
 - памћење (испис) најкраће пронађене путање

**** Lin-Kernighan heuristics [LKH] је једна од најбољих хеуристика за TSP**