



Електротехнички факултет
Универзитет у Београду



Инжењерски оптимизациони алгоритми

др Драган Олћан, в. проф. (olcan@etf.rs)
Јована Петровић, асистент (jovanap@etf.rs)
Дарко Нинковић, асистент (darko@etf.rs)

Изборни предмет
Школска 2020/21. година

<http://mtt.etf.rs/si/ioa.htm>

Који су циљеви и исходи овог курса?

- Упознавање са основним класама оптимизационих алгоритама који се користе у инжењерству и ИТ струци
- Оспособљавање за практичну примену оптимизационих алгоритама при решавању инжењерских проблема
- Пројектовање се данас ради практично искључиво коришћењем рачунара
- Практично решавање инжењерских проблема по правилу захтева употребу оптимизације
- Познавање оптимизационих алгоритама је од изузетне важности [као и спознаја када и како их (не) користити]

Зашто курс о инжењерским оптимизационим алгоритмима?

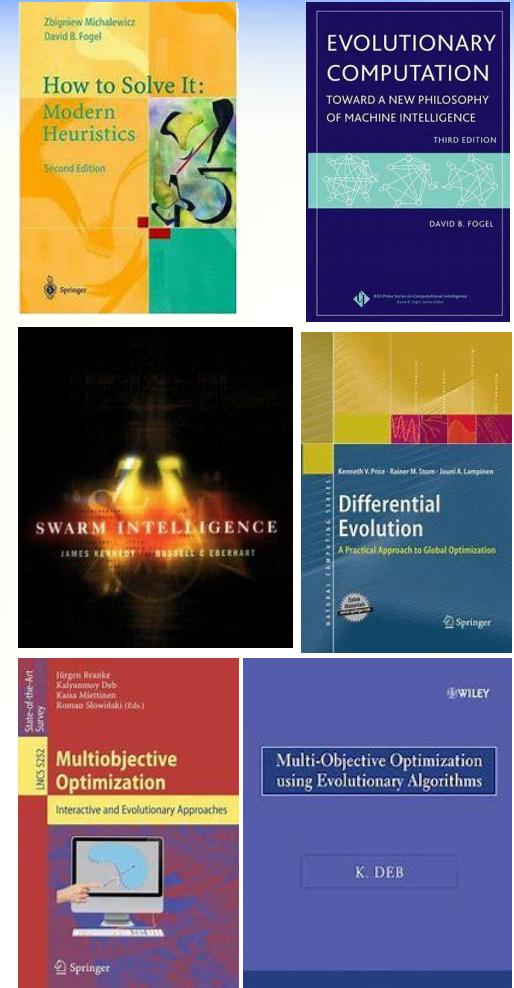
- Познавање модерних алгоритама за оптимизације је **неопходно** за све који се баве: **инжењерством, науком и бизнисом**
- Поред алгоритама, сагледаћемо **могућности за решавање** проблема данас
- Зашто једноставно не узмем рутину из неке од готових софтверских библиотека или постојећег софтвера?
 - Познавање и разумевање алгоритама је од изузетног значаја, чак и када се користе готова решења
 - Често је случај да постојећи софтвер није оптимално решење за наш проблем

Шта је градиво курса?

- **Увод.** Преглед појмова и представљање основне теорије решавања система нелинеарних једначина на које се своде оптимизациони алгоритми у инжењерству
- **Систематизација.** Поделе оптимизационих алгоритама
- **Оптимизациони алгоритми:**
 - систематско претраживање (енглески: systematic search)
 - случајно претраживање (енглески: random search)
 - градијентна метода (енглески: gradient method)
 - симплекс алгоритам (енглески: Nelder-Mead simplex)
 - Данцигов симплекс алгоритам (енглески: Dantzig simplex)
 - симулирано каљење (енглески: simulated annealing)
 - генетички алгоритам (енглески: genetic algorithm)
 - кретање јата (енглески: particle swarm optimization)
 - диференцијална еволуција (енглески: differential evolution)
- **Оптимизација са више критеријума.** Парето фронт и његово одређивање коришћењем оптимизационих алгоритама
- **Рад на рачунару.** Сагледавање особина и параметара оптимизационих алгоритама који су од значаја за практичну примену кроз програмирање и симулације на рачунару

Литература

- Литература из оптимизационих алгоритама је изузетно **обимна**
- Употреба оптимизационих алгоритама је изузетно распространена
- Избрани су они алгоритми (или класе алгоритама) који се данас најчешће употребљавају у инжењерској пракси
- Нагласак је на **алгоритмима широке намене** специјализовани алгоритми само као примери (нпр: quick-sort, Dijkstra's algorithm, Chess Master...)

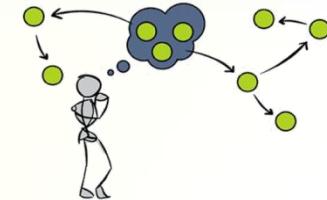


Како је организован курс?

- Предавања:
30 часова предавања + 30 часова вежби +15 часова лабораторије
- Литература
 - Материјал са предавања (PDF за сваку седмицу, линк најсајту предмета)
 - Z. Michalewicz, D.B. Fogel, *How to Solve It: Modern Heuristics*, Springer, 2004
 - Xin-She Yang, *Engineering Optimization An Introduction with Metaheuristic Applications*, University of Cambridge, Department of Engineering, Cambridge, United Kingdom, Wiley 2010
 - D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, 1989
- Оцењивање
 - **Предиспитне обавезе** (задаци на вежбама + бонуси) – одсеца се на 70 поена
 - **Задатак са вежби** се брани на вежбама у лабораторији
 - **Бонуси** до максимално 10 поена
 - **Испит** (задатак или задаци), највише 40 поена, одсеца се на 30 поена
 - **Коначна оцена** – укупан број поена се добија сабирањем поена добијених на основу предиспитних обавеза и испита. За полагање испита неопходно је освојити бар 51 поен. Оцене 6-10 су равномерно расподељене у опсегу од 51 до 100 поена
- Распоред (термини предавања и вежби):
предавања петак 12h-14h и вежбе 14h-16h @ MS Teams

Како се изводе предавања и вежбе?

- Предавања:
 - слайдови
 - табла & креда
 - рачунар за илустрације и показе
- Вежбе:
 - **Ви радите на рачунару**
 - програмирање: C/C++ и Python
битно је да програм (про)ради!
 - коришћење постојећег софтвера
- Наставници и сарадници задржавају право да, уколико посумњају или утврде да се студент не придржава правила "наставе на даљину", као и осталих норми понашања које важе током студирања на ЕТФу,
 - захтевају додатне детаље у вези са решењима задатака, у електронском облику,
 - захтевају да студент одговори на додатна питања, у електронском облику, или
 - не додељују поене на предиспитним обавезама током "наставе на даљину".



Предиспитне обавезе током епидемије*

- Задатак се задаје на крају предавања и важи до рока за предају (~ 6 дана)
- Током вежби решавате задатак и можете да питате асистента детаље око задатка (MS Teams канал)
- Решење задатка се састоји од
 - ASCII (TXT) фајла са јасно записаним (нумеричким) одговорима на питања из задатка
 - Кодом за решење задатка (C/C++ за Visual Studio 2017 или Python 3.7)
- Решење се шаље као **један ZIP фајл** (< 10 MB), кроз портал на сајту
- Решења се достављају најкасније до **четвртка** следеће седмице у **12:00**
- Освојени поени објављују се **четвртком до 21:00** на сајту предмета (структура поена за задатак биће изложена на одговарајућим вежбама)
- **Петком од 11:15 до 12:00** су показне лабораторијске вежбе где приказујемо званично решење задатка од прошле седмице (и евентуално решавамо примедбе на број освојених поена)

* У случају побољшања епидемиолошке ситуације, лабораторијске вежбе се одржавају у истом термину у Лабораторији 64б

<http://mtt.etf.rs/si/IOA/up.html>

The screenshot shows the homepage of the Faculty of Electrical Engineering (Katedra za opštu elektrotehniku) at mtt.etf.rs. The main content area displays the title "Inženjerski optimizacioni algoritmi" and a brief description of the course objectives. To the right, there is a sidebar titled "Obaveštenja" with a link to "Prezentacija predmeta". Below the main content, there is a form for submitting assignments. The form includes fields for "Indeks (godina/broj)_" and "ZIP sa rešenjem koji ćete poslati:", with "Choose File" and "Upload ZIP file" buttons. A large blue arrow points from the "Link za slanje rešenja zadatka" (link to submit assignment) at the bottom left towards the "Upload ZIP file" button. Another blue arrow points from the "Poeni osvojeni tokom tekuce školske godine." (points earned during the current school year) link at the bottom left towards the same button.

Inženjerski optimizacioni algoritmi

Cilj ovog predmeta je upoznavanje sa osnovnim klasama optimizacionih algoritama koji se koriste u inženjerstvu i IT struci. Naglasak je na sposobljavanju studenata za praktičnu primenu ovih algoritama za rešavanje optimizacionih problema. Osnovne klase algoritama koji se izučavaju u okviru predmeta su: potpuno nestražnjino, genetičko, genetički algoritam (engleski: genetic algorithm), optimizacija s popravljajućim operatorima, evolucijski algoritmi, evolutivni algoritmi. Razmatraju se i problemi određivanje rešenja u tom slučaju. Performanse algoritma su u pitanju.

[Karton predmeta.](#)

[Uputstvo za studente.](#)

[Fond časova:](#) 2+2+1

Literatura:

- [Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristic Methods for Optimization and Machine Learning](#)
- [Xin-She Yang, Engineering Optimization: An Introduction with MATLAB Examples](#)
- [D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning](#)

Termin za nastavu, školska 2020/21. godina

Studenti će moći da prate predavanja i vežbe korišćenjem **MS Teams platforme** u terminima prenosa (Po okončanju odabira izbornih predmeta, biće uzmotrena i mogućnost održavanja nastave u učionici). Naziv tima: **13S074IOA - Inženjerski optimizacioni algoritmi**. Link za pristup timu: [13S074IOA - Inženjerski optimizacioni algoritmi](#)

[Link za slanje rešenja zadatka](#)

[Poeni osvojeni tokom tekuce školske godine.](#)

Obaveštenja

[Prezentacija predmeta](#)

Inženjerski optimizacioni algoritmi:

Indeks (godina/broj)_: /

Odaberite ZIP sa rešenjem koji ćete poslati:

No file chosen

Pošaljite ZIP sa rešenjem:

Predmet	Naziv	Primenjeno u ak.	Приложено објеката														Svetski	Бројни	Однос																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
			1	2	3	4	5	6	7	8	9	10	11	12	13	14																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
1	1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	7010	7011	7012	7013	7014	7015	7016	7017	7018	7019	7020	7021	7022	7023	7024	7025	7026	7027	7028	7029	7030	7031	7032	7033	7034	7035	7036	7037	7038	7039	7040	7041	7042	7043	7044	7045	7046	7047	7048	7049	7050	7051	7052	7053	7054	7055	7056	7057	7058	7059	7060	7061	7062	7063	7064	7065	7066	7067	7068	7069	7070	7071	7072	7073	7074	7075	7076	7077	7078	7079	7080	7081	7082	7083	7084	7085	7086	7087	7088	7089	7090	7091	7092	7093	7094	7095	7096	7097	7098	7099	70100	70101	70102	70103	70104	70105	70106	70107	70108	70109	70110	70111	70112	70113	70114	70115	70116	70117	70118	70119	70120	70121	70122	70123	70124	70125	70126	70127	70128	70129	70130	70131	70132	70133	70134	70135	70136	70137	70138	70139	70140	70141	70142	70143	70144	70145	70146	70147	70148	70149	70150	70151	70152	70153	70154	70155	70156	70157	70158	70159	70160	70161	70162	70163	70164	70165	70166	70167	70168	70169	70170	70171	70172	70173	70174	70175	70176	70177	70178	70179	70180	70181	70182	70183	70184	70185	70186	70187	70188	70189	70190	70191	70192	70193	70194	70195	70196	70197	70198	70199	70200	70201	70202	70203	70204	70205	70206	70207	70208	70209	70210	70211	70212	70213	70214	70215	70216	70217	70218	70219	70220	70221	70222	70223	70224	70225	70226	70227	70228	70229	70230	70231	70232	70233	70234	70235	70236	70237	70238	70239	70240	70241	70242	70243	70244	70245	70246	70247	70248	70249	70250	70251	70252	70253	70254	70255	70256	70257	70258	70259	70260	70261	70262	70263	70264	70265	70266	70267	70268	70269	70270	70271	70272	70273	70274	70275	70276	70277	70278	70279	70280	70281	70282	70283	70284	70285	70286	70287	70288	70289	70290	70291	70292	70293	70294	70295	70296	70297	70298	70299	70300	70301	70302	70303	70304	70305	70306	70307	70308	70309	70310	70311	70312	70313	70314	70315	70316	70317	70318	70319	70320	70321	70322	70323	70324	70325	70326	70327	70328	70329	70330	70331	70332	70333	70334	70335	70336	70337	70338	70339	70340	70341	70342	70343	70344	70345	70346	70347	70348	70349	70350	70351	70352	70353	70354	70355	70356	70357	70358	70359	70360	70361	70362	70363	70364	70365	70366	70367	70368	70369	70370	70371	70372	70373	70374	70375	70376	70377	70378	70379	70380	70381	70382	70383	70384	70385	70386	70387	70388	70389	70390	70391	70392	70393	70394	70395	70396	70397	70398	70399	70400	70401	70402	70403	70404	70405	70406	70407	70408	70409	70410	70411	70412	70413	70414	70415	70416	70417	70418	70419	70420	70421	70422	70423	70424	70425	70426	70427	70428	70429	70430	70431	70432	70433	70434	70435	70436	70437	70438	70439	70440	70441	70442	70443	70444	70445	70446	70447	70448	70449	70450	70451	70452	70453	70454	70455	70456	70457	70458	70459	70460	70461	70462	70463	70464	70465	70466	70467	70468	70469	70470	70471	70472	70473	70474	70475	70476	70477	70478	70479	70480	70481	70482	70483	70484	70485	70486	70487	70488	70489	70490	70491	70492	70493	70494	70495	70496	70497	70498	70499	70500	70501	70502	70503	70504	70505	70506	70507	70508	70509	70510	70511	70512	70513	70514	70515	70516	70517	70518	70519	70520	70521	70522	70523	70524	70525	70526	70527	70528	70529	70530	70531	70532	70533	70534	70535	70536	70537	70538	70539	70540	70541	70542	70543	70544	70545	70546	70547	70548	70549	70550	70551	70552	70553	70554	70555	70556	70557	70558	70559	70560	70561	70562	70563	70564	70565	70566	70567	70568	70569	70570	70571	70572	70573	70574	70575	70576	70577

Историја курса и примена у индустрији

WIPL-D Optimizer

WIPL-D Optimizer is a powerful multi-algorithm optimization tool that is being used by many successful professionals around the world. The tool calculates a single solution as well as multiple solutions for complex criteria optimizations. Thanks to its simple and intuitive graphical interface, you can quickly solve the problem at hand. It enables a high level of design automation for antenna, antenna system, scatterer, or a microwave

WIPL-D Optimizer is seamlessly integrated with [WIPL-D Pro](#) and [WIPL-D Microwave](#). When you create your project, you want to get optimum performance, you just start the Optimizer from within the design environment, select the parameters to be optimized and the optimization criteria, and let the tool do the rest. You can specify cost-function based on virtually all the EM simulation results that are calculated with WIPL-D 3D EM solver, as well as the simulation results from WIPL-D Microwave.

Various optimization algorithms are available. The set covers all major methods proven to work efficiently in practice. The available optimization algorithms are:

- Particle Swarm
- Genetic
- Simplex
- Random
- Systematic Search
- Simulated Annealing
- Gradient

Some of the special options that distinguish WIPL-D Optimizer from other commercially available optimization tools:

- Hybrid optimization (involving two consecutive optimization algorithms)
- Pareto fronts (set of the best compromises in a multi-criteria optimization)
- Estimation of local minima (keeping several solutions that are in some proximity of the best found solution)
- Optimization repetitions (multiple optimizations from random starting points - decreases the probability of finding a local optimum which is not in fact the global optimum)

Optimizer Method 1: Random
Max. Number of Iterations for Method 1: 100000
Optimizer Method 2: None
Max. Number of Iterations for Method 2: 300
Current Iteration of Method 1: 21
Last Iteration Cost = 1.17136e+002
Current Iteration of Method 2: 0
Total Solver Runs = 21

Symbol	Current Value	Lowest Value	Highest Value	The Best Value
reflector_din	2.00000e+000	None	None	2.00000e+000
frq_start	3.00000e+008	None	None	3.00000e+008
frq_stop	3.00000e+008	None	None	3.00000e+008
frq_step	0.00000e+000	None	None	0.00000e+000
<input checked="" type="checkbox"/> radius_1	5.64511e+002	2.00000e+002	5.00000e-001	1.64514e+001
<input checked="" type="checkbox"/> radius_2	1.13313e-001	2.00000e-002	5.00000e-001	6.59243e-002
<input checked="" type="checkbox"/> pitchang_1	5.96510e+000	5.00000e+001	1.50000e+001	3.52690e+000
<input checked="" type="checkbox"/> pitchang_2	2.06696e+000	5.00000e+001	1.50000e+001	3.42859e+000

Criterion | Weight
helix | 0

Run | Exit | Reset | About

Simple Example: Optimized horn antenna

The open end of a rectangular waveguide is a source of electromagnetic waves, but it also represents a discontinuity creation of unwanted higher modes. A horn is added at the open end to reduce the magnetic waves at this discontinuity and to diminish the higher modes.

Let us assume that the length of the horn is specified. In that case, we can only change the surface of the open end of the horn in order to achieve greater gain. But, how can we achieve the best possible gain? On one hand, increase of the horn's aperture leads to decrease of the antenna's gain due to changes in the distribution of phase at the opening. On the other hand, by increasing the surface of the opening, the physical surface of the antenna is being increased and so is its effective surface. This leads to the augmentation of gain.

Let's say that we want to achieve the gain along the axis of the waveguide greater than 20 dB, and we wonder how large the surface of the opening needs to be. We then set the optimization criteria and the range for height and width of the horn and then we run the WIPL-D Optimizer.

When the optimization procedure finishes, we get the optimum dimensions of the horn in just a few seconds.

Automated Design of a Waveguide Filter

In which you can see the possibility of optimization-based design of a microwave circuit illustrate an effective procedure for designing waveguide filters consisting of series of coupled initial guess. The coupling is performed by using arbitrary waveguide discontinuities (e.g.,

7-pole filter were -60 dB to 7.7 GHz and -0.3 dB between 7.8 and 8.4 GHz. The standard rectangular IEC-R84 (WR-112) e 28.5 mm width (A) and the dielectric is air and the metal is copper. The model is made using analytical models of rectangular H double step components in WIPL-D-Microwave Pro environment.

The obtained filter mostly satisfies the desired characteristic. Agreement in stopband was excellent (60 dB were obtained) and insertion loss was almost (up to 0.38 dB with losses included) in the range requested. WIPL-D software enables fast and practical waveguide filter design. Even when there is only a theoretical image of solution, WIPL-D Microwave and WIPL-D Optimizer can be used to find the filter that satisfies the given criteria.

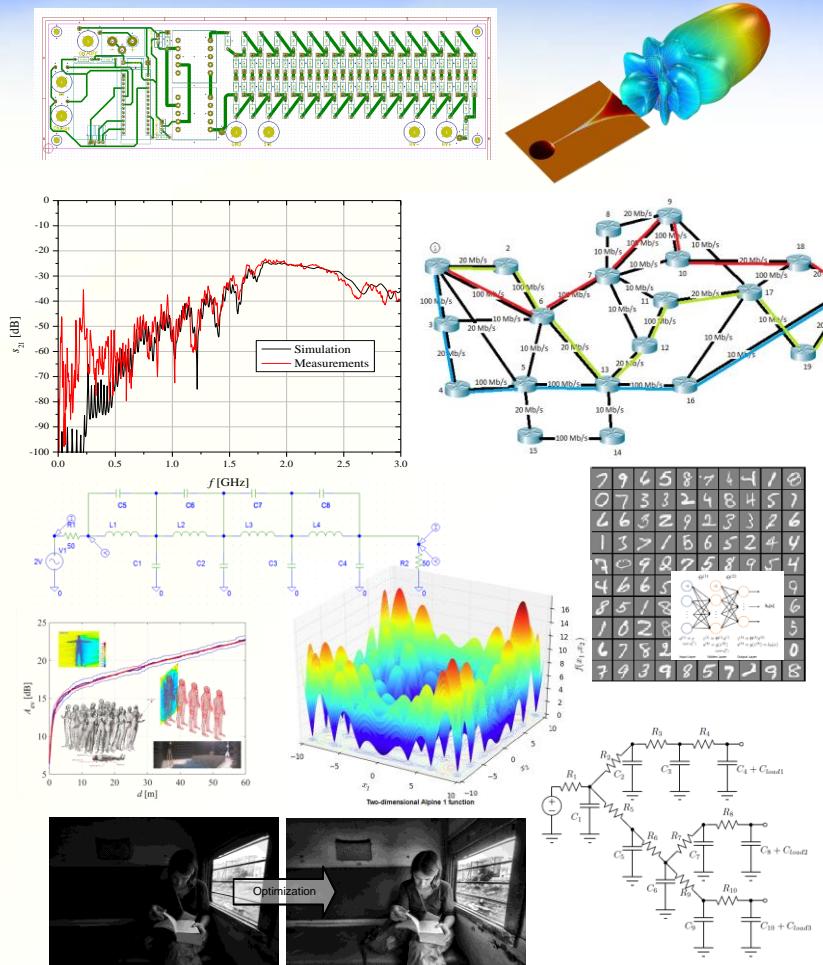
S21 parameter before and after optimization

Final S11 and S21 - optimized filter

Copyright © 2013 WIPL-D d.o.o. All rights reserved.

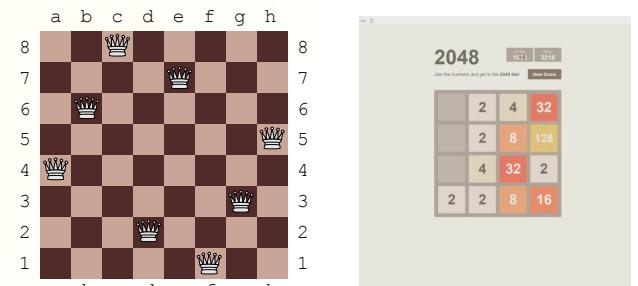
Примери оптимизације из електротехнике и рачунарства

- Пројектовати електрично коло са одговарајућим особинама (преносне функције филтра, појачање појачавача, израда елемената кола у различитим технологијама, Q-фактор калема итд.)
- Пројектовање антене за задати дијаграм зрачења
- Проналажење оптималног рутирања у мрежама
- Проналажење оптималних кодова за пренос и запис према различитим критеријумима
- Оптимално покривање подручја радио сигналом
- Оптимално искоришћење фреквенцијског спектра
- Минимизација површине чипа и максимизације његових рачунарских перформанси
- Минимизација времена извршавања програма
- Проналажење функције која оптимално фитује задати скуп података
- Минимизација потрошње електричне енергије уређаја
- Максимизација капацитета батерије
- ...



Други примери оптимизације

- Оптималан избор школе
- Оптималан избор изборног предмета на студијама
- Оптималан избор (будућег) радног места
- Оптимално коришћење свог и туђег времена
- Оптимална расподела економских ресурса
- Најбржа путања за задату полазну и крајњу тачку
- Избор оптималног потеза у игри
- Максимизација учинка радника
- Минимизација цене производње
- Оптималан избор тима за пројекат
- Оптимална расподела информација за постизање жељеног циља
- ...



Како је оптимизација повезана са другим научним дисциплинама?

- Инжењери користе знања математике, логике, економије, искуство и интуицију да пронађу решење проблема
- Математика: решавање система нелинеарних једначина на континуалном или дискретном домену
- Pattern recognition / machine learning / data mining / knowledge discovery су уско повезани са оптимизацијом
- Оптимизација се врши алгоритмима који се данас по правилу реализују програмираним
- Дарвин: еволуција ствара и чува особине које су боље за опстанак у условима у којима се одиграва природна селекција
- Економија је наука која се бави проучавањем како друштво управља ограниченим ресурсима



Шта је “проблем” или задатак?

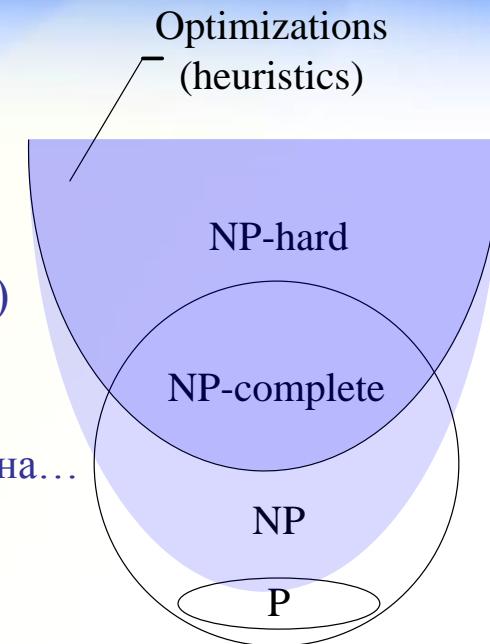
- Проблем (задатак) постоји онда када желимо нешто да решимо (постоји несклад/разлика између текућег и жељеног стања)
 - Задатак на испиту: имам поставку, а желим решење
 - Електротехника: направи електрични уређај или систем задатих карактеристика
 - Рачунарство: направи софтвер задатих карактеристика
- Реални проблеми се увек решавају ван јасно дефинисане области (на факултету, по правилу се решавају у оквиру области)
 - Велики и први корак ка решавању је знати одакле почети
- Решење је постизање жељеног циља у оквиру ограничења
- У пракси најчешће мора да се решава више проблема истовремено, а ти проблеми често могу имати опречне захтеве
- Постоје разни проблеми – ми ћемо се бавити рачунским

Рачунски проблем (енг: computational problem)

- Проблем који се решава помоћу рачуна (и рачунара)
- Проблем = скуп свих вредности улазних података + одговарајући резултат за сваку вредност улазних података
 - Скуп улазних података може бити коначан или бесконачан
- Подела:
 - Проблеми одлучивања: имају бинаран резултат {да, не} или {1, 0} или {true, false}
 - Проблеми оптимизације: имају генералан нумерички резултат {реалан број, низ бита, скуп бројева, пермутација елемената скупа итд. }
- Проблеми оптимизације могу се трансформисати у проблеме одлучивања формулацијом питања
 - Оптимизација: пронаћи најкраћу путању између задатих тачака
 - Одлука: да ли постоји путања краћа од x [m] између задатих тачака?
 - Последица: **теорија рачунске сложености** (енг: computational complexity theory) **важи и за оптимизационе проблеме!**

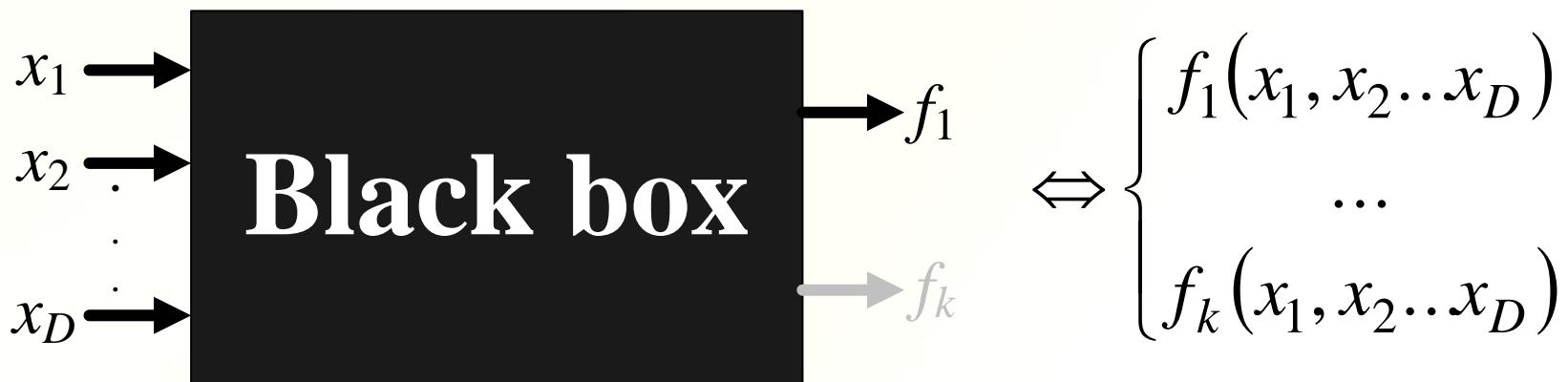
Сложеност алгоритама и поређење перформанси

- Тјурингова машина
- Детерминистички и недетерминистички приступи
- Временска сложеност и просторна сложеност:
рачунарски ресурси потребни за решавање проблема
= време (рачунске операције) + простор (меморијске локације)
- Претпоставимо да алгоритам обрађује N улазних елемената
- Сложеност алгоритма
 - линеарна, квадратна, логаритамска, полиномска, експоненцијална...
- Означавање
 - $O(N)$, $O(N^2)$, $O(\log N)$, $O(N^{\text{const.}})$, $O(2^N)$ итд.
- Теорија: алгоритми се пореде према сложености
- Питање од милион долара: $P \neq NP$ или $P = NP$?
- Пракса: $t = k_t \cdot O(f_t(N))$ и $m = k_m \cdot O(f_m(N))$
да ли проблем може да се реши на задатом хардверу у задатом времену?
- За стохастичке оптимизационе алгоритме тражићемо и
најбрже проналажње (средњег) најбољег решења



Black-box оптимизације (Модел црне кутије)

- У пракси решавање се своди на проблеме за које не постоји предзнање
 - све што зnamо о проблему потребно је искористити за решавање
- За задате улазне податке (побуду) $\mathbf{x} = (x_1, x_2, \dots, x_D)$ добијају се резултати (одзиви) $\mathbf{f} = (f_1, f_2, \dots, f_k)$
- Одзиви нису познати унапред за све могуће побуде (ако су сви одзиви познати, најбољи одзив је решење проблема)
- Унутрашња организација непозната (због природе, ограничења...)



Резултати црне кутије:

“The Blind Men and the Elephant”

- John Godfrey Saxe (1816-1887) -

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind.

The *First* approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the **Elephant**
Is very like a WALL!"

The *Second*, feeling of the tusk,
Cried, "Ho, what have we here,
So very round and smooth and sharp?
To me 'tis mighty clear
This wonder of an **Elephant**
Is very like a SPEAR!"

The *Third* approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "**the Elephant**
Is very like a SNAKE!"

The *Fourth* reached out an eager hand,
And felt about the knee
"What most this wondrous beast is like
Is mighty plain," quoth he:
""Tis clear enough the **Elephant**
Is very like a TREE!"

The *Fifth*, who chanced to touch the ear,
Said: "E'en the blindest man
Can tell what this resembles most;
Deny the fact who can,
This marvel of an **Elephant**
Is very like a FAN!"

The *Sixth* no sooner had begun
About the beast to grope,
Than seizing on the swinging tail
That fell within his scope,
"I see," quoth he, "**the Elephant**
Is very like a ROPE!"

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

Формализација

- Формално, сваки одзив је функција више променљивих (побуда)
- Одзив, или нека функција одзива, је мера колико је решење добро или лоше
- При оптимизацији (решавању проблема), за сваки скуп улазних података које желимо да разматрамо, потребно је израчунавати одзив(е)
- Подела модела (поставки) према броју одзива:
 - са једним одзивом (један критеријум оптимизације)
 - са више одзива (више критеријума оптимизације)
- Први део курса: оптимизације са једним критеријумом
- Последња трећина курса: вишекритеријумске оптимизације

Оптимизација и функције више променљивих

- Побуде \leftrightarrow оптимизационе променљиве
- Одзив \leftrightarrow оптимизациона функција
(нумеричка мера квалитета решења)
 - Оптимизациона функција је функција са једном или више променљивих $f(x_1, x_2..x_D)$
 - Променљиве и резултат оптимизационе функције могу бити из скупа дискретних или континуалних бројева

Терминологија

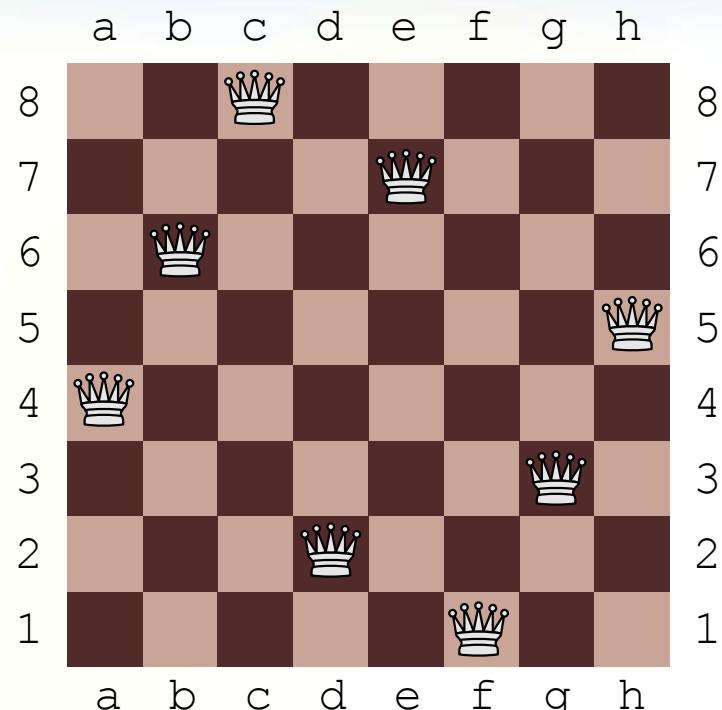
- **Оптимизациона функција** ≡
функција грешке ≡ cost function ≡ evaluation function ≡
нумеричка мера квалитета разматраног решења
- Једно израчунавање оптимизационе функције ≡ **итерација**
- **Оптимизационе променљиве** ≡ улазни подаци за проблем
- **Оптимизациони простор (S)** је
скуп свих могућих вредности оптимизационих променљивих
(енг: optimization space, search space, set of candidate solutions...)
- **Простор (смислених) решења (F)** је
скуп свих решења које има смисла разматрати
(енг: feasible solutions,...)
$$F \subseteq S$$
- **Број димензија оптимизационог простора (D)**
је број оптимизационих променљивих
- Свака тачка у оптимизационом простору
представља један избор вредности улазних података

Колико је велики оптимизациони простор?

- Величина оптимизационог простора је од изузетне важности
- Избор оптимизационог алгоритма зависи од:
 - величине оптимизационог простора
 - домена оптимизационих променљивих
- Величина простора зависи од усвојеног записа!
 - Бијективни записи имају исту величину оптимизационог простора
 - Најбољи запис је онај за који важи $F = S$
- Оптимизациони простор може бити ограничен или неограничен
(constrained vs. unconstrained)

Пример записа и величине опт. простора: N -краљица

- Проблем: поставити N -краљица на “шаховску таблу” димензија $N \times N$ тако да се краљице не нападају
- Нека су r_k и c_k позиције краљице k :
 $1 \leq k \leq N$ (редни број)
 $1 \leq r_k \leq 8$ (ред)
 $1 \leq c_k \leq 8$ (колона)
- Краљица:
 $f1 \rightarrow (r_k, c_k) = (1, 6)$
- Колико је велики оптимизациони простор?



Величина оптимизационог простора зависи од записа!

- Запис #1: 64 могућа места $(1,1), (1,2), \dots (8,8)$ на која можемо да ставимо 8 краљица
 $\mathbf{x} = \{K_1, K_2, \dots, K_8\}, 1 \leq K_n \leq 64$
- Запис #2: свака краљица у свом реду
 $\mathbf{x} = \{K_1, K_2, \dots, K_8\}, 1 \leq K_n \leq 8$
K#1: $(1, K_1), \dots, K#8: (8, K_8)$
- Запис #3: краљице не могу бити у истом реду и у истој колони
 $\mathbf{x} = \{3, 6, 2, 4, 1, 5, 7, 8\}$
K#1: $(1, 3), K#2: (2, 6), \dots, K#8: (8, 8)$

$$|F| = \binom{64}{8} = 4\,426\,165\,368$$

$$|F| = 8^8 = 16\,777\,216$$

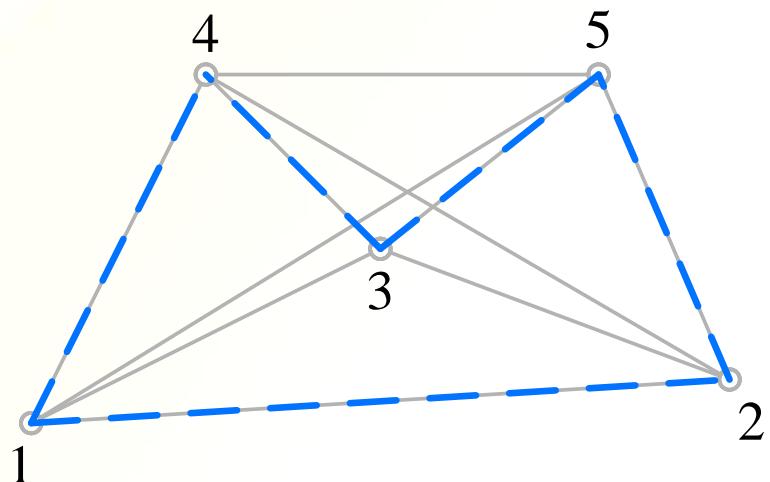
$$|F| = 8! = 40\,320$$

Основни типови оптимизационих проблема (TSP, SAT, NLP)

- Проблем трговачког путника
(енглески: traveling salesman problem, TSP)
- Булова алгебра (дискретна стања)
(енглески: Boolean satisfiability, SAT)
- Нелинеарни проблеми
(енглески: Nonlinear programming, NLP)
- Домени основних типова проблема
(дискретан или континуалан)

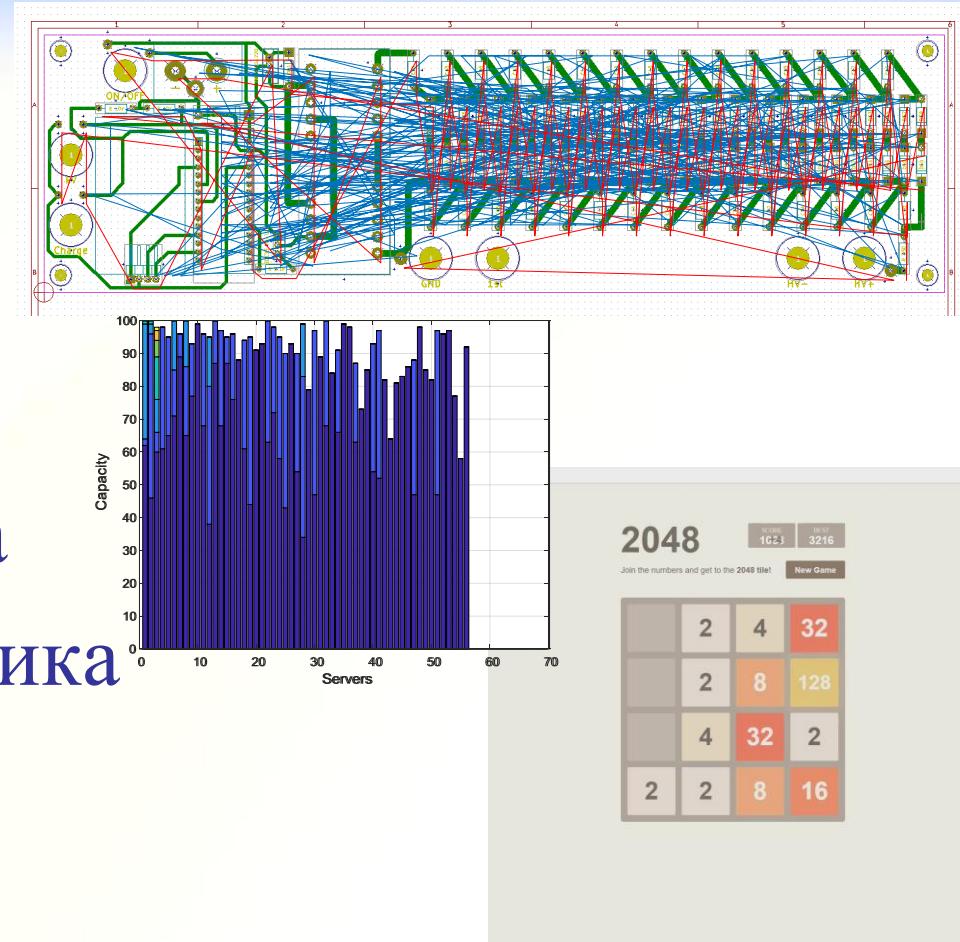
TSP: поставка

- За задати скуп D градова и позната растојања између сваког паре градова, пронаћи најкраћи пут који пролази кроз сваки град и завршава се у полазном граду
- Број могућих путања D !
 - $D = 15$, број путања $\approx 1,3 \cdot 10^{12}$
 - $D = 150$, број путања $\approx 57,1 \cdot 10^{261}$
- Приказана путања
5-3-4-1-2(-5)
- Подела:
 - Симетричан: $dist(p,q) = dist(q,p)$
 - Асиметричан: $dist(p,q) \neq dist(q,p)$
 - Метрички ($d_{AB} \leq d_{AC} + d_{CB}$) и они који то нису
 - Еуклидски: симетричан + метрички



TSP: инжењерски проблеми

- Минимална путања алата за бушење на штампаним плочама
- Рутирање возила
- Путање у графовима
- Планирање и логистика
- ...



TSP класа проблема и поделе

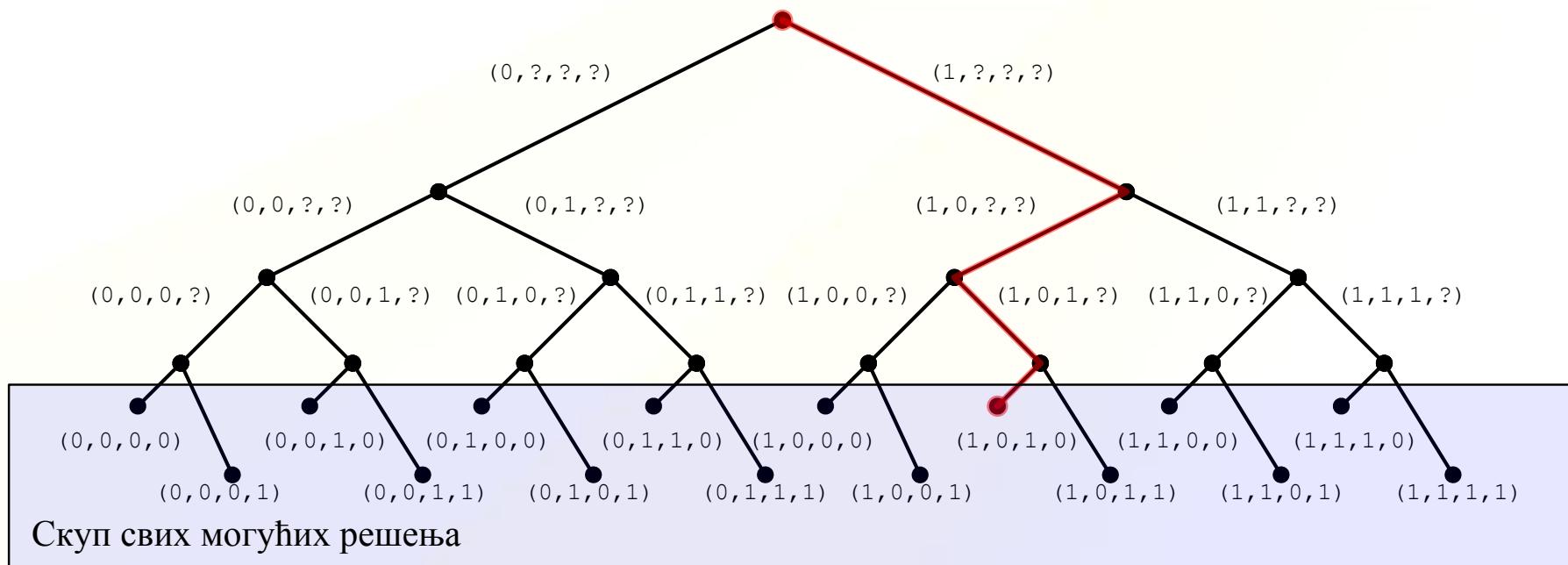
- Сваки проблем који може да се сведе на проверу **свих пермутација** називаћемо **проблем TSP класе**
- Енглески: TSP-encoded problems
- Поделе TSP:
 - Симетричан (енг: symmetric TSP, STSP)
 - Асиметричан (енг: asymmetric TSP, ATSP)
 - Најкраћи Хамилтонов пут (енг: shortest Hamiltonian path, SHP)
 - Хамилтонова контура (енг: Hamiltonian cycle)
 - Обилазак са условима (енг: Sequential ordering problem, SOP)
 - Рутирање возила ограниченог капацитета (енг: Capacitated vehicle routing problem, CVRP)
 - ...
- Посебне поткласе могу имати једноставнија решења
- Генерализација: job shop problem

SAT: поставка

- За задати логички израз $F(x_1, x_2, \dots, x_D)$ пронаћи вредности променљивих (x_1, x_2, \dots, x_D) , $x_k \in \{\text{true (1), false (0)}\}$, $k = 1, 2, \dots, D$ тако да је $F(x_1, x_2, \dots, x_D) = \text{true}$
$$F(\mathbf{x}) = x_1 \wedge ((x_2 \wedge x_1) \vee (x_2 \wedge x_3 \wedge \neg x_4))$$
- Пример $\mathbf{x}_{\text{opt}} = (x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$
$$F(\mathbf{x}_{\text{opt}}) = \text{true}$$
- Број различитих \mathbf{x} је 2^D
 - $D = 15$, број могућих решења 32768
 - $D = 150$, број могућих решења $\approx 1,4 \cdot 10^{45}$
- Улазни подаци могу да се запишу као низ бита 100110...

SAT: приказ помоћу графова (бинарно стабло графа)

$$F(\mathbf{x}) = x_1 \wedge ((x_2 \wedge \bar{x}_1) \vee (\bar{x}_2 \wedge x_3 \wedge \bar{x}_4))$$



SAT класа проблема и поделе

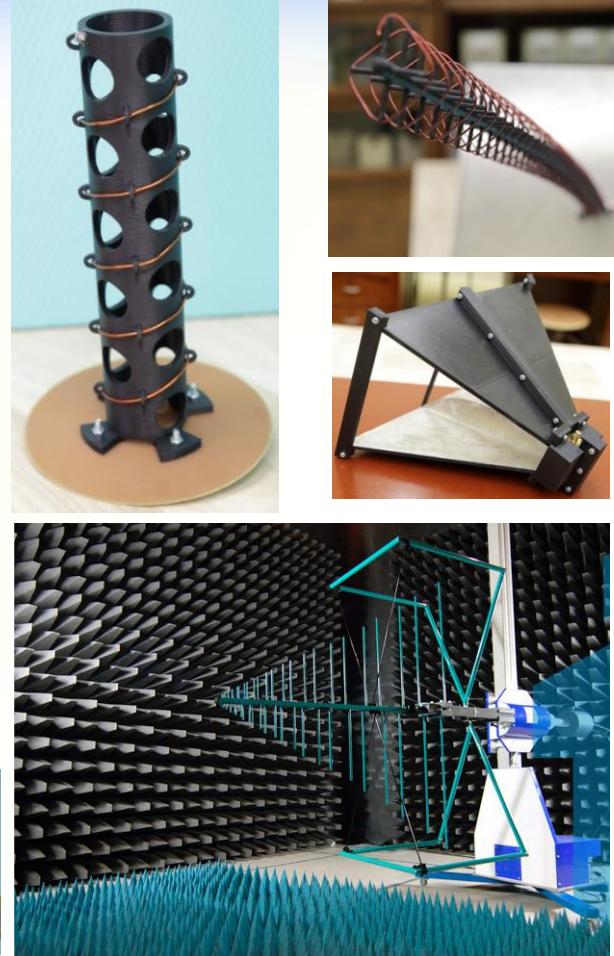
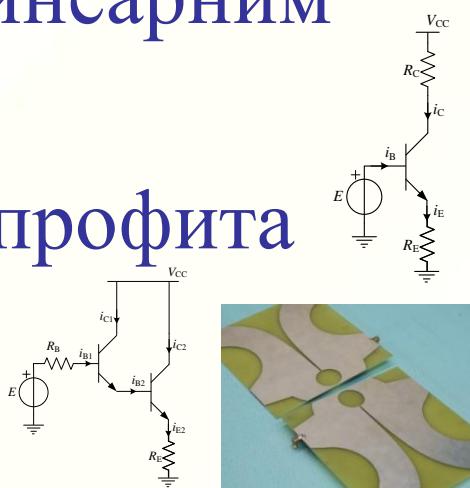
- Сваки проблем који се своди на проверу свих могућих вредности секвенце бита називаћемо проблем SAT класе
- Поделе (Karp's 21 NP-complete problems)
 - проблем ранца (knapsack)
 - подела посла (job-sequencing)
 - directed/undirected Hamiltonian cycle
 - ...
- Примери: свака манипулација битима може да претвори у проблем SAT класе
 - Откључавање ZIP архиве
 - Пробијање RSA кодова
 - ...

NLP проблеми

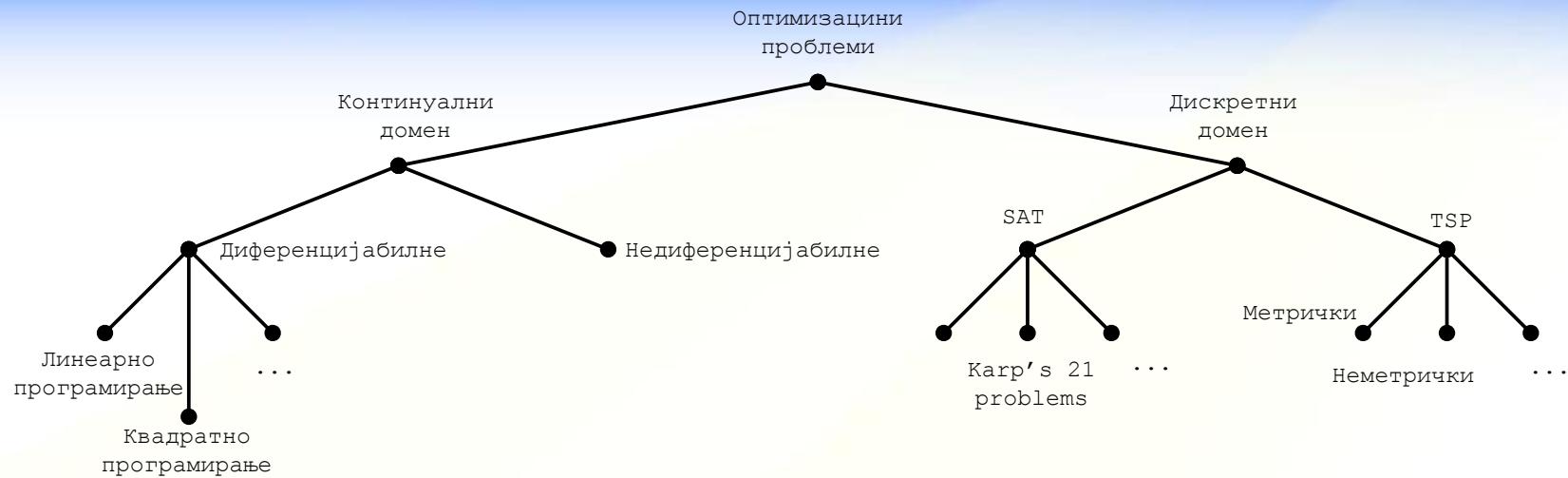
- Сваки оптимизациони проблем са **континуалним променљивима** спада у NLP класу проблема
- Број могућих решења је (теоријски) **бесконачан!**
- Посебни случајеви
 - диференцијабилне и недиференцијабилне f
 - линеарно програмирање
 - квадратно програмирање
 - конвексни/конкавни проблеми
 - са ограниченим или неограниченом доменом
 - ...

NLP инжењерски примери

- Пројектовање ЕМ уређаја
- Решавање нелинеарних кола
- Фитовање резултата мерења
- Управљање нелинеарним системима
- Максимизација профита
- ...



Класификација оптимизационих проблема



- Подела није јединствена
- Пермутације су специјалан случај низова бита: $TSP \subset SAT$?
- Све што радимо на рачунару је са коначном тачношћу (64-бита): $NLP \subset SAT$?
- Класификација је према **запису улазних података (побуда)** и природи оптимизационог проблема

Колико је брз мој рачунар?

- Колико времена могу да потрошим за оптимизацију?
- Колико пута могу да израчунам опт. функцију $f(\mathbf{x})$?
- Желим процену брзине: максималан број позива (оптимизационе) функције
- Једноставан програм: прототип оптимизације

```
#include <stdio.h>
#include <time.h>

double F(double a, double b)
{
    return a + b;
}

int main(void)
{
    double x,eval;
    double max = 7e9;

    time_t t1, t2;
    time(&t1);

    for (x=0; x < max; x=x+1.0 )
        eval = F(x,x);

    time(&t2);

    printf("%5.5e\n", (t2 - t1)/max);
    return 0;
}
```

Колико времена је потребно да решимо проблеме?

- Претпоставимо да је свака провера 1ns
- TSP 100 градова:
 $100! \cdot 1\text{ns} \approx 3 \cdot 10^{141}$ година!
- SAT 100 бита:
 $2^{100} \cdot 1\text{ns} \approx 4 \cdot 10^{13}$ година!
- NLP: **x** има ∞ могућности
→ потребно је теоријски бесконачно много времена!



Решавање реалних оптимизационих проблема

- Број могућих решења је изузетно велики те је потребно изузетно много рачунарских ресурса
- Постоји много **ограничења** тако да је тешко наћи било какво решење (а не оптимално)
 - Посебно ограничење: човек који решава проблем није адекватно припремљен
- Често је доволно наћи **задовољавајуће решење**
 - Трагање за најбољим (оптималним) решењем је можда занимљиво али најчешће неприхватљиво
- Тада се примењују
оптимизациони алгоритми и хеуристике

Шта су хеуристике?

- ХЕУРИСТИКА (енг: heuristic)
грчки корен речи “Εύρισκω“ – пронаћи или открити
- IEEE: All engineering is heuristic
- Технике решавања проблема засноване на искуству, учењу и откривању које доводе до решења (не мора нужно бити оптимално али је довољно добро)
- Када год је немогуће или непрактично потпуно претраживање простора, користе се хеуристике (интуиција, стереотипи, здрав разум, енг: rule-of-thumb, educated guess, ...)
- **State-of-the-art** проблеми се увек решавају хеуристички

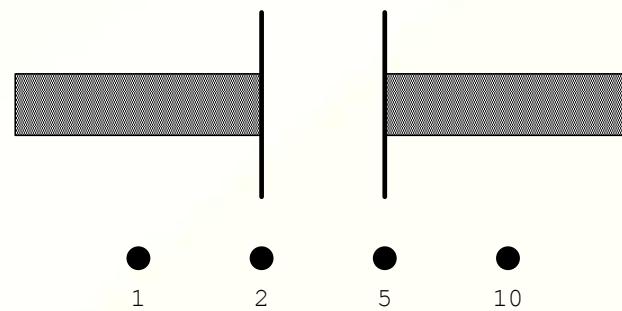
Шта је заједничко за све приступе решавању “проблема” (задатака)?

- Структура сваког алгоритма (приступа) за решавање “проблема” има три основна дела
 1. запис могућих решења,
 2. циљ који је потребно постићи и
 3. оптимизациону функцију (нумеричку меру појединачних решења)
- Запис (енг: representation)
 - TSP: једна пермутација, нпр. $\mathbf{x} = (2,1,3,6,4,5,7)$
 - SAT: један низ бита, нпр. $\mathbf{x} = 1001100101$
 - NLP: један вектор реалних бројева, нпр. $\mathbf{x} = (1.5, -3.2, 4.76, 17.2)$
- Запис не мора да буде “природан”: TSP као бинарни низ, реални бројеви као низ бита, итд.
 - Генерално на запис се може применити било која трансформација (пресликавање) и добити други запис.
 - Бијективни записи проблема имају исту сложеност решавања!
- Циљ и оптимизациона функција НИСУ исто!
- Запис и опт. функцију формулише онај ко решава проблем

Пример:

4 човека и трошни мост

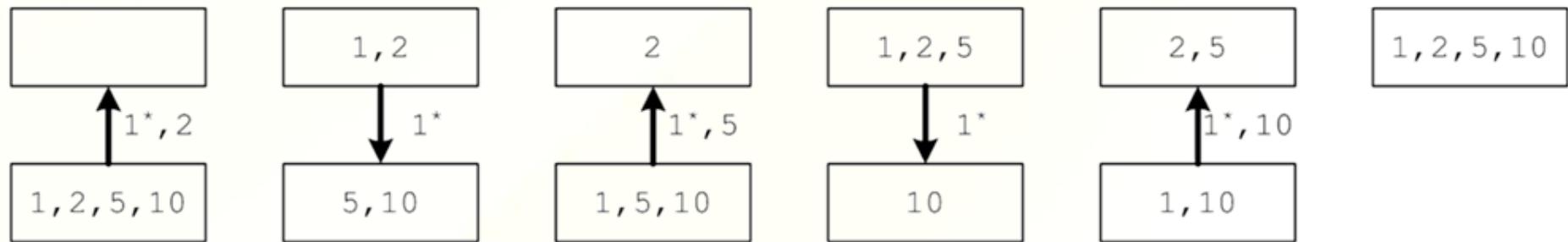
Четири особе морају да пређу са једне стране трошног моста на другу, током ноћи. Мост највише може да издржи две особе истовремено. За прелазак је неопходна лампа да би се осветлио сваки корак и избегле рупе (тј. при преласку у паровима иду брзином споријег). Постоји само једна лампа, коју није могуће пребацити са једне стране моста на другу (тј. увек неко мора да је носи). Први човек може да пређе мост за 1 min, други за 2 min, трећи за 5 min и четврти за 10 min. Пронађи минимално време потребно да сви пређу на другу страну моста.



Пример:

Основне идеје и једно решење

- Домен проблема је дискретан, а број могућности је пребројив и коначан.
- Прелази се у паровима, а лампу враћа једна особа.
- Да би четири особе прешли, потребно је 3 преласка и 2 повратка.
- Лампу са друге стране враћа увек најбржи који је на тој страни.
- Интуитивно решење, најбржи носи (враћа) лампу и преводи једног по једног $\{1,2,1,1,5,1,1,10\}$, $T = 19 \text{ min}$.



Пример:

Пребројавање решења и запис

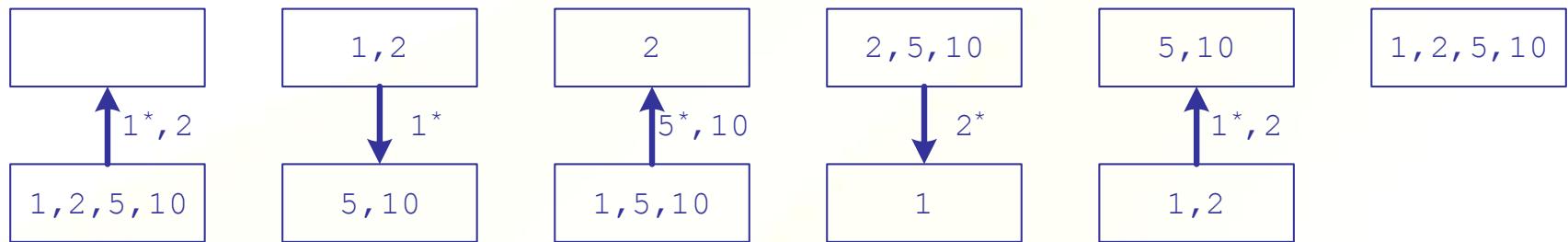
- При преласку (\uparrow) за избор првог паре постоји $\binom{4}{2}$ могућности, за избор другог паре $\binom{3}{2}$ могућности и за избор трећег паре $\binom{2}{2} = 1$ могућност.
- При повратку (\downarrow) имамо само једну могућност.
- Укупан број могућих комбинација прелазака је $N = \binom{4}{2} \cdot 1 \cdot \binom{3}{2} \cdot 1 \cdot \binom{2}{2} = 18$.
- Све комбинације можемо да запишемо у облику вектора $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ где су елементи вектора брзине одговарајућих људи, тј. $x_k \in \{1, 2, 5, 10\}$, $k = 1, 2, \dots, 8$. Први пар при преласку једнозначно одређују x_1, x_2 , други пар x_4, x_5 , трећи пар x_7, x_8 , а повратак једнозначно одређују x_3 и x_6 .
- Током прелазака потребно је водити евиденцију о томе ко је на којој страни (брзине једнозначно одређују људе). На почетку, на првој страни су сви $S_1 = \{1, 2, 5, 10\}$, а на другој нема људи $S_2 = \{\}$. После преласка првог паре и повратка једне особе са лампом $|S_1| = 3$ и $|S_2| = 1$.
- Укупно време преласка је $T = \max(x_1, x_2) + x_3 + \max(x_4, x_5) + x_6 + \max(x_7, x_8)$.

Пример: све могућности

\uparrow #1	\downarrow #1	\uparrow #2	\downarrow #2	\uparrow #3	T			
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	2	1	1	5	1	1	10	19
1	2	1	1	10	1	1	5	19
1	2	1	5	10	2	1	2	17
1	5	1	1	2	1	1	10	19
1	5	1	1	10	1	1	2	19
1	5	1	2	10	2	1	2	20
1	10	1	1	2	1	1	5	19
1	10	1	1	5	1	1	2	19
1	10	1	2	5	2	1	2	20
2	5	2	1	2	1	1	10	20
2	5	2	1	10	1	1	2	19
2	5	2	2	10	2	1	2	21
2	10	2	1	2	1	1	5	20
2	10	2	1	5	1	1	2	20
2	10	2	2	5	2	1	2	21
5	10	5	1	2	1	1	5	23
5	10	5	1	5	1	1	2	23
5	10	5	2	5	2	1	2	24

Пример: решење

- Минимално време преласка је 17 min



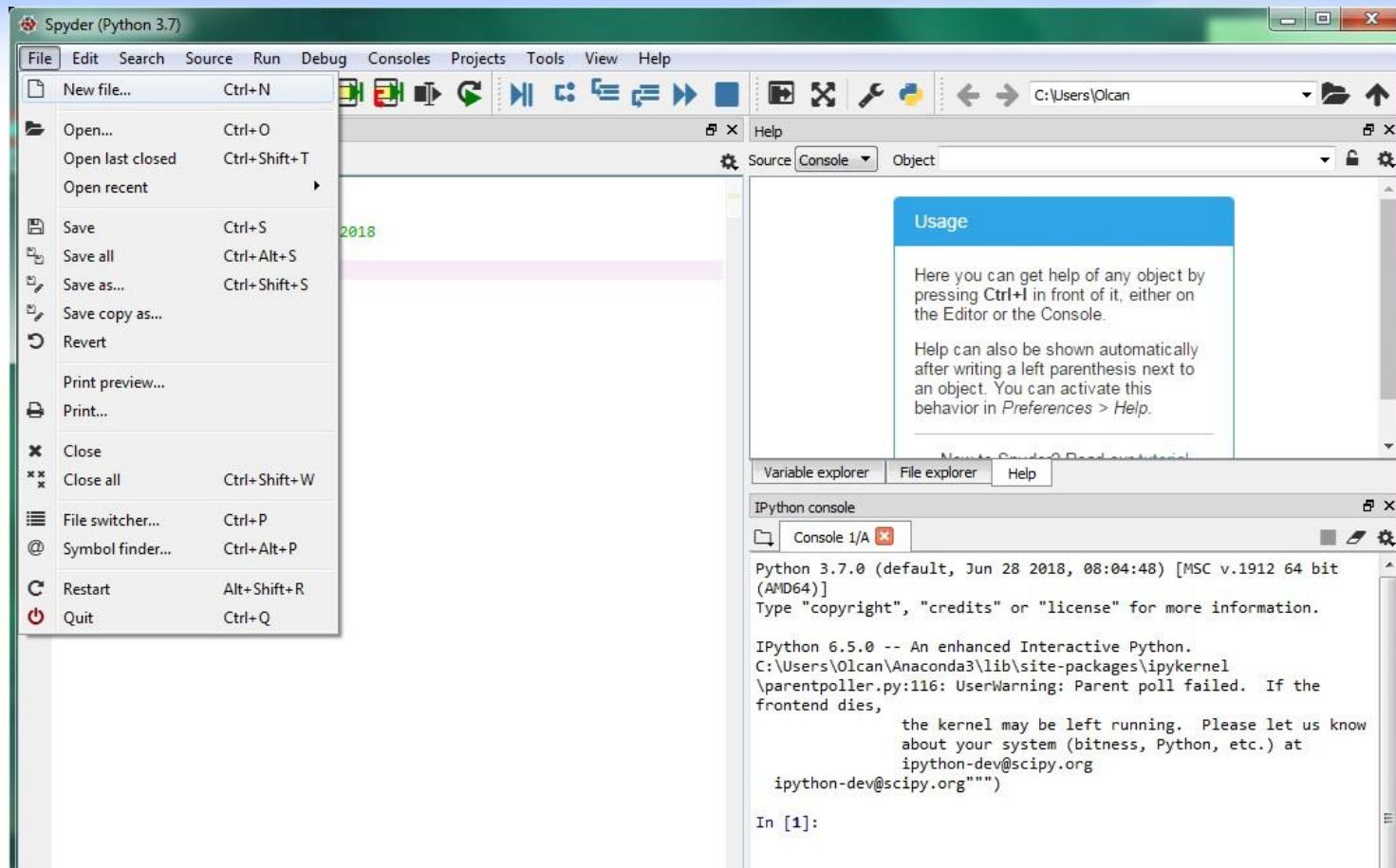
Задатак за вежбе

- Купац је купио четири производа. Ако се цене изразе у доларима, онда су збир и производ свих цена (бројчано) исти и износе 7,11. Цене се заокружују на \$0,01 (1 cent).
(а) Написати програм који извршава потпуну претрагу по све четири цене и помоћу њега одредити цене. Израчунати максималан број позива оптимизационе функције.
(б) Изразити једну цену преко осталих и написати програм који извршава потпуну претрагу по (преостале) три цене. Израчунати максималан број позива опт. функције и упоредити брзину програма у односу на програм из (а).
(в) Који од ова два програма је бржи?

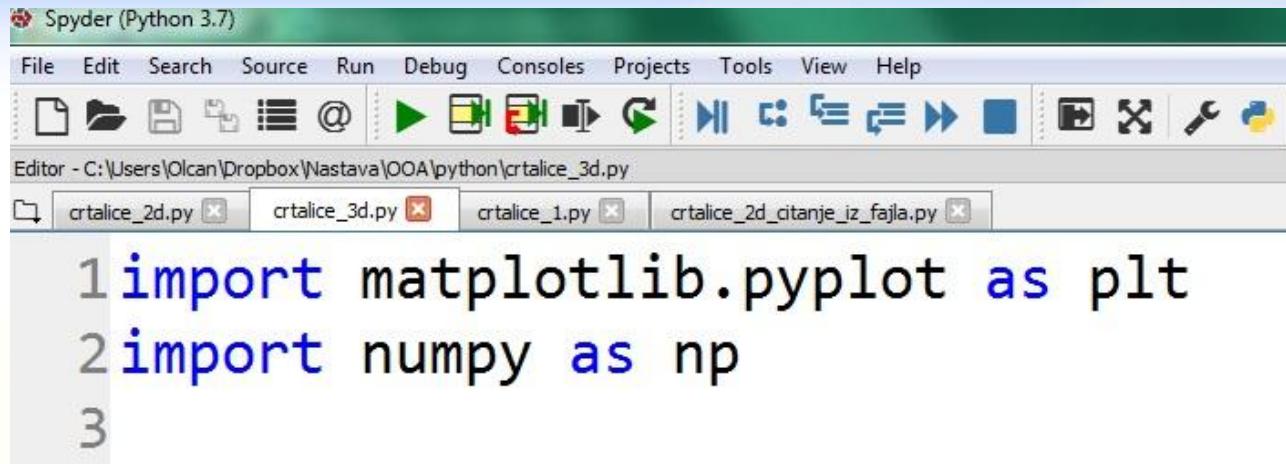
Вежбе на рачунару: Мотивација

- Упознавање са окружењима која ће бити коришћена на вежбама
 - **Python 3.7** (Spyder Anaconda 3):
приказ и обрада резултата
 - **C/C++** (Visual Studio 2017):
рачунарски захтевни делови програма
- Студенти се охрабрују да користе и своје рачунаре.

Python 3.7 (Spyder окружење са Anaconda 3)



Библиотеке



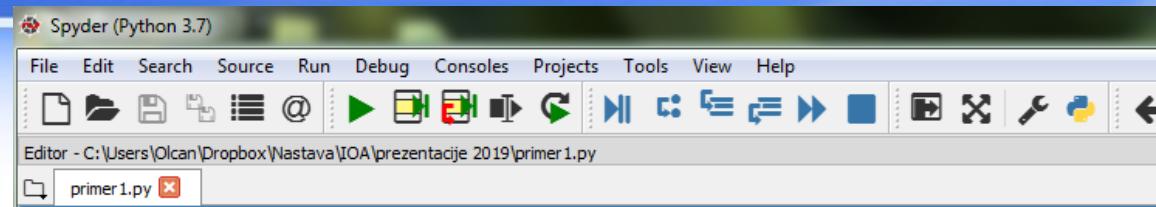
The screenshot shows the Spyder Python IDE interface. The title bar reads "Spyder (Python 3.7)". The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with various icons. The central area is an "Editor" window titled "C:\Users\Olcan\Dropbox\Nastava\OOA\python\crtalice_3d.py". The code editor contains the following Python code:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
```

The status bar at the bottom shows the file path "C:\Users\Olcan\Dropbox\Nastava\OOA\python\crtalice_3d.py". The bottom navigation bar has tabs for "crtalice_2d.py", "crtalice_3d.py", "crtalice_1.py", and "crtalice_2d_citanje_iz_fajla.py".

- **plt**: figure(), plot(), plot_surface(),
set_xlabel(), plt.legend()
- **np**: arange(), meshgrid()
- Користан сайт: <https://matplotlib.org/>

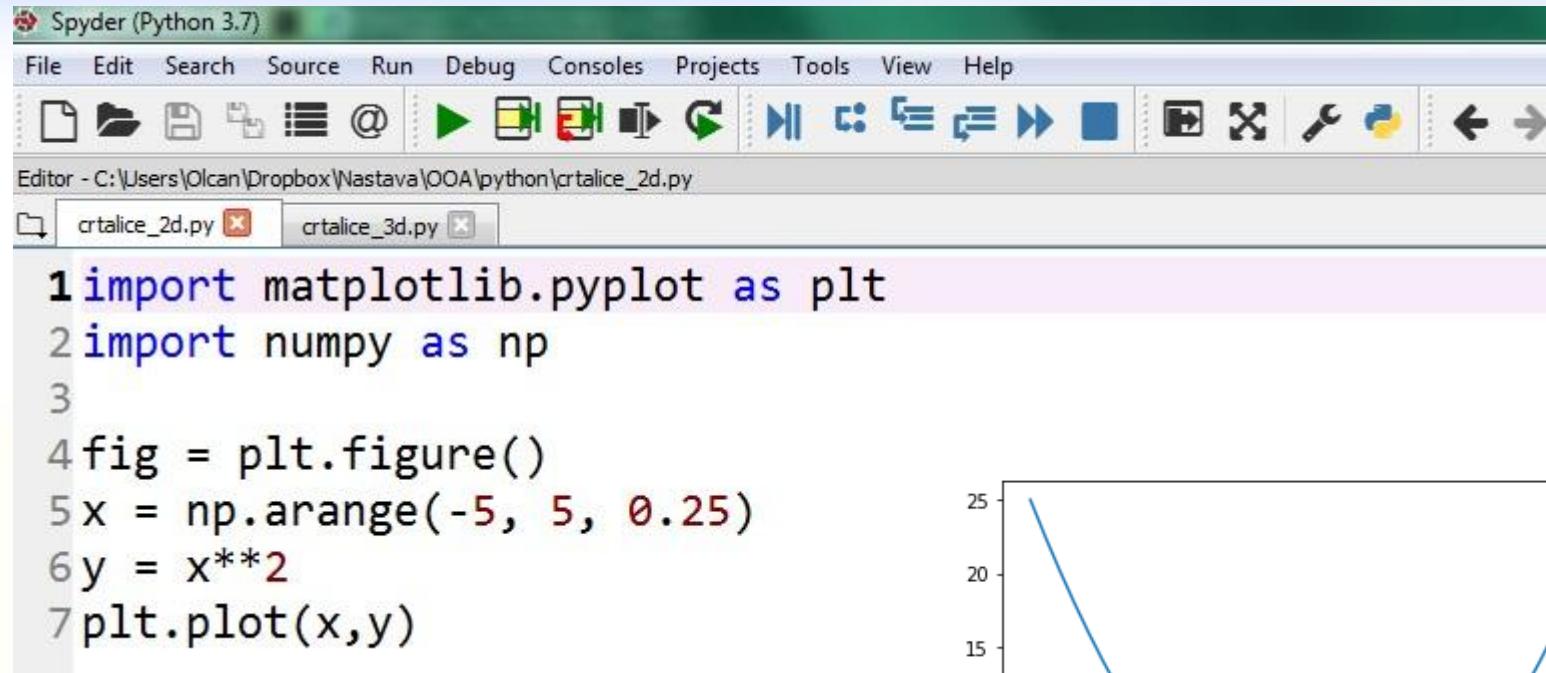
Дефинисање функција



```
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Olcان\Dropbox\Nastava\IOA\prezentacije 2019\primer1.py
primer1.py

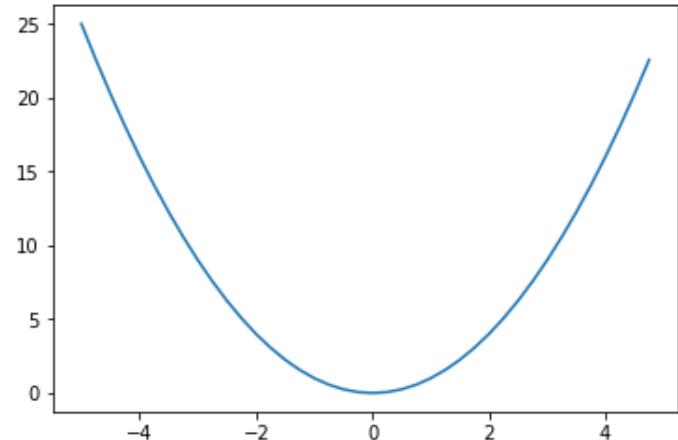
1 import numpy as np
2
3 def napon_otpornika(R, I):
4     return R*I
5
6 def prosto_kolo(R1, R2, E):
7     I = E/(R1+R2)
8     U1 = I*R1
9     U2 = I*R2
10    return np.array([I, U1, U2])
11
12 R = 2e3
13 I = 1e-3
14 U = napon_otpornika(R, I)
15 print(U)
16
17 R1 = 1e3
18 R2 = 3e3
19 E = 1
20 resenje = prosto_kolo(R1, R2, E)
21 print(resenje[1], resenje[2])
```

1D функција: *plot*



The screenshot shows the Spyder Python IDE interface. The title bar says "Spyder (Python 3.7)". The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar has various icons for file operations and debugging. The status bar indicates "Editor - C:\Users\Olcan\Dropbox\Nastava\OOA\python\crtalice_2d.py". Below the toolbar, there are two tabs: "crtalice_2d.py" and "crtalice_3d.py", with "crtalice_2d.py" being active. The code editor contains the following Python script:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 fig = plt.figure()
5 x = np.arange(-5, 5, 0.25)
6 y = x**2
7 plt.plot(x,y)
```



Легенда

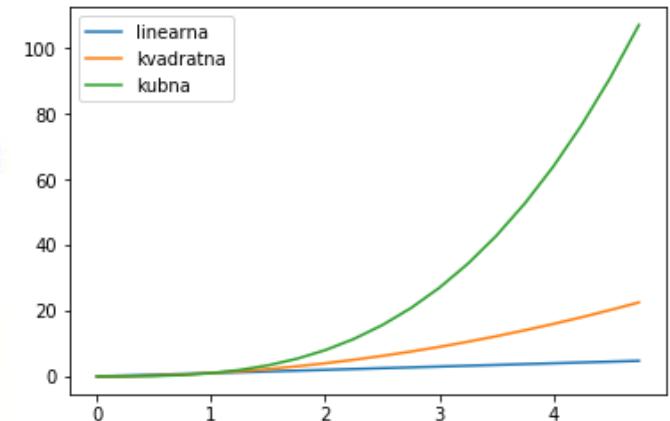
Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - C:\Users\Olcan\Dropbox\Nastava\OOA\python\crtalice_2d.py

crtalice_2d.py* crtalice_3d.py crtalice_1.py crtalice_2d_citanje_iz_fajla.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 fig = plt.figure()
5 x = np.arange(0, 5, 0.25)
6
7 plt.plot(x, x, label = 'linearna')
8 plt.plot(x, x**2, label = 'kvadratna')
9 plt.plot(x, x**3, label = 'kubna')
10 plt.legend()
```



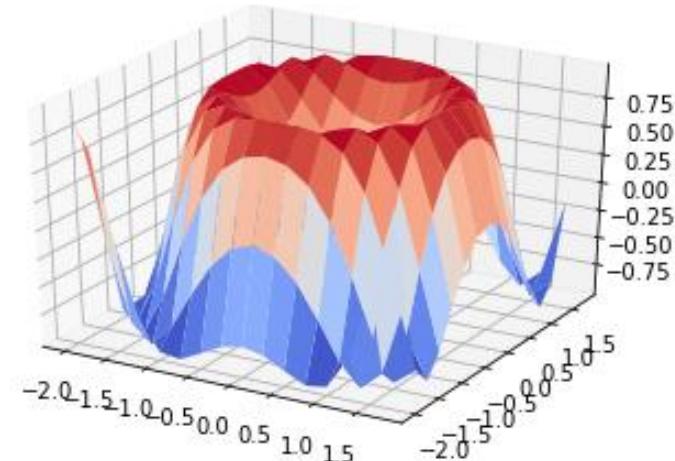
2D функција: *plot_surface* и *meshgrid*

Spyder (Python 3.7)

```
File Edit Search Source Run Debug Consoles Projects Tools View Help  
Editor - C:\Users\Olcان\Dropbox\Nastava\OOA\python\crtalice_3d.py  
crtalice_s_par.py crtalice_3d.py crtalice_2d.py
```

1 from mpl_toolkits.mplot3d import Axes3D
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 fig = plt.figure()
7 ax = fig.gca(projection = '3d')
8
9 x = np.arange(-2, 2, 0.25)
10 y = np.arange(-2, 2, 0.25)
11 x,y = np.meshgrid(x,y)
12 z = np.sin(x**2+y**2)
13 h = ax.plot_surface(x,y,z, cmap=plt.cm.coolwarm)

14



Означавање оса

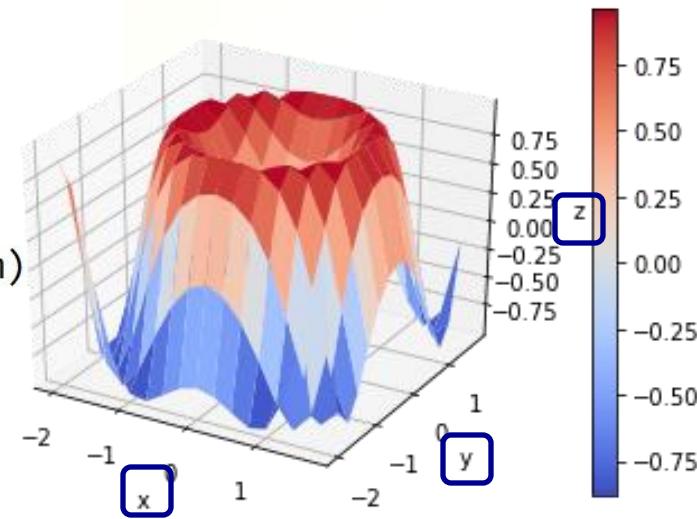
Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - C:\Users\Olcان\Dropbox\Nastava\OOA\python\crtalice_3d.py

crtanje_s_par.py crtalice_3d.py crtalice_2d.py

```
1 from mpl_toolkits.mplot3d import Axes3D
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 fig = plt.figure()
7 ax = fig.gca(projection = '3d')
8
9 x = np.arange(-2, 2, 0.25)
10 y = np.arange(-2, 2, 0.25)
11 x,y = np.meshgrid(x,y)
12 z = np.sin(x**2+y**2)
13 h = ax.plot_surface(x,y,z, cmap=plt.cm.coolwarm)
14
15 ax.set_xlabel('x')
16 ax.set_ylabel('y')
17 ax.set_zlabel('z')
18
19 fig.colorbar(h)
```



Учитавање података из датотеке

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - C:\Users\Olcان\Dropbox\Nastava\OOA\python\crtalice_2d_citanje_iz_fajla.py

crtanje_s_par.py crtalice_3d.py crtalice_2d.py crtalice_2d_citanje_iz_fajla.py

```
1 import matplotlib.pyplot as plt
2
3 file = open('funkcija.txt','r')
4 x = []
5 y = []
6 for line in file:
7     s = line.split()
8     if (len(s)!=0):
9         x.append(float(s[0]))
10        y.append(float(s[1]))
11 file.close()
12
13 plt.figure()
14 plt.plot(x,y)
15 plt.xlabel('x')
16 plt.ylabel('y')
17
```

funkcija.txt - No...

File Edit Format View Help

x	y
-5.0	25.0
-4.75	22.5625
-4.5	20.25
-4.25	18.0625
-4.0	16.0
-3.75	14.0625
-3.5	12.25
-3.25	10.5625
-3.0	9.0
-2.75	7.5625
-2.5	6.25
-2.25	5.0625
-2.0	4.0
-1.75	3.0625
-1.5	2.25
-1.25	1.5625
-1.0	1.0
-0.75	0.5625
-0.5	0.25
-0.25	0.0625
0.0	0.0
0.25	0.0625
0.5	0.25
0.75	0.5625
1.0	1.0
1.25	1.5625
1.5	2.25
1.75	3.0625
2.0	4.0
2.25	5.0625
2.5	6.25
2.75	7.5625

Пример минимизације оптимизационе функције

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - C:\Users\Olcان\Dropbox\Nastava\IOA\prezentacije 2019\primer2.py

primer2.py

```
1 from scipy.optimize import minimize
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 "Rosenbrock function"
6 def fun(x):
7     return (1-x[0])**2.0 + 100*(x[1]-x[0]**2)**2
8
9
10 "crtanje"
11 x = np.arange(-2, 2, 0.11)
12 y = np.arange(-1, 3, 0.11)
13 x,y = np.meshgrid(x, y)
14 z = np.log10(fun([x,y]))
15
16 h = plt.contourf(x,y,z)
17 plt.scatter(1.0, 1.0, facecolor='red')
18 plt.xlabel('x')
19 plt.ylabel('y')
20 plt.colorbar(h)
21 print(fun([1,1]))
22
23 "trazenje minimuma"
24 x0 = np.array([0,0])
25 res = minimize(fun, x0, method='Nelder-Mead', options = {'ftol':1e-8,'disp': True, 'maxfev':1e4})
26 xmin = np.array(res.x)
27 print("xmin = (",xmin[0], ", ", xmin[1],")")
```

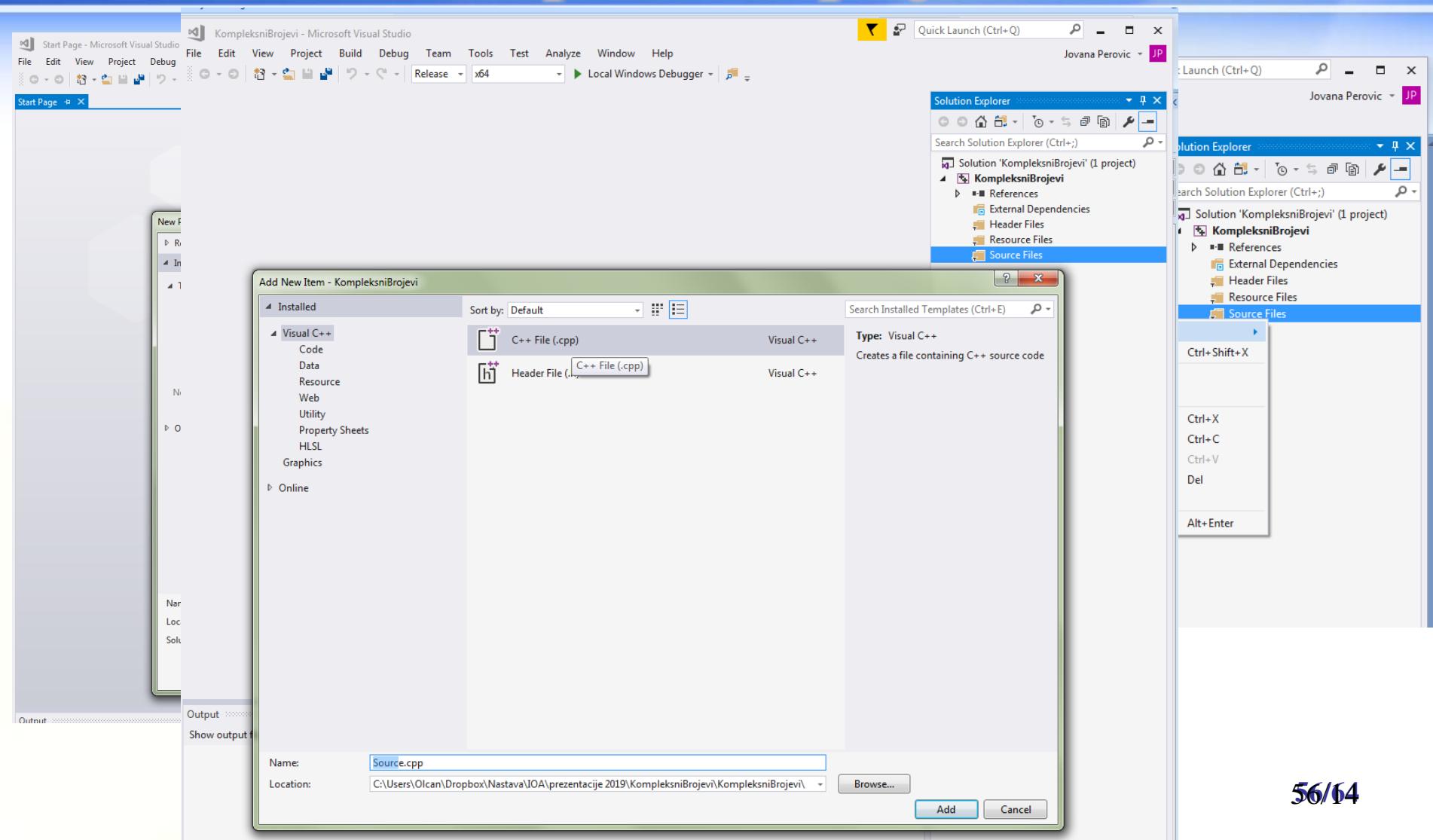
Optimization terminated successfully.
Current function value: 0.000000
Iterations: 79
Function evaluations: 146
xmin = (1.0000043858986165 , 1.0000106409916478)

A contour plot of the Rosenbrock function, which is a non-convex function with a narrow valley. The x-axis ranges from -2.0 to 1.5, and the y-axis ranges from -1.0 to 2.5. The plot shows contour lines of the function, with colors indicating the value of the function. A red arrow points to the minimum point at (1, 1), which is highlighted with a red dot. The word 'МИНИМУМ' (Minimum) is written in Russian above the arrow.

55/64

Visual Studio 2017

Отварање пројекта



Рад са комплексним бројевима

The screenshot shows the Microsoft Visual Studio interface with a project named "KompleksniBrojevi". The left pane displays the "Source.cpp" file containing C++ code for performing arithmetic operations on complex numbers. The right pane shows the terminal window displaying the program's output.

```
#include <stdio.h>
#include <iostream>
#include <complex>

using namespace std;

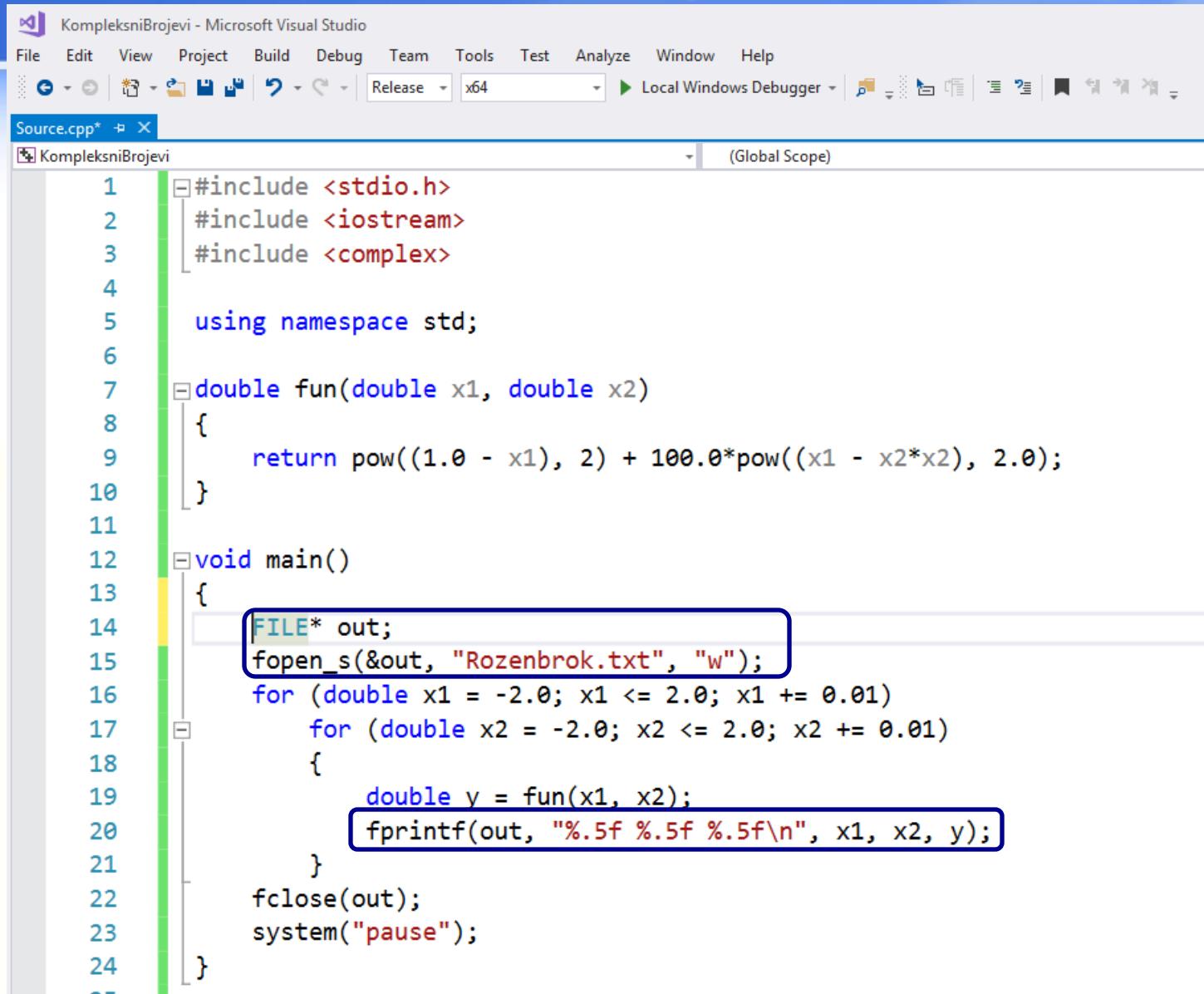
void main()
{
    //imaginarna jedinica
    complex<double> i;
    i.real(0.0);
    i.imag(1.0);

    complex<double> z1, z2, add, prod, div;
    double re, im;
    scanf_s("%lf %lf", &re, &im);
    z1.real(re);
    z1.imag(im);
    scanf_s("%lf %lf", &re, &im);
    z2.real(re);
    z2.imag(im);
    add = z1 + z2;
    prod = z1*z2;
    div = z1 / z2;
    printf("zbir: (%.2f, %.2f)\n", real(add), imag(add));
    printf("proizvod: (%.2f, %.2f)\n", real(prod), imag(prod));
    printf("kolicnik: (%.2f, %.2f)\n", real(div), imag(div));
    system("pause");
}
```

The terminal window output is:

```
1
2
3
4
5
zbir: <-1.00, 7.00>
proizvod: <-12.00, 1.00>
kolicnik: <0.28, -0.31>
Press any key to continue . . .
```

Уписивање у датотеку

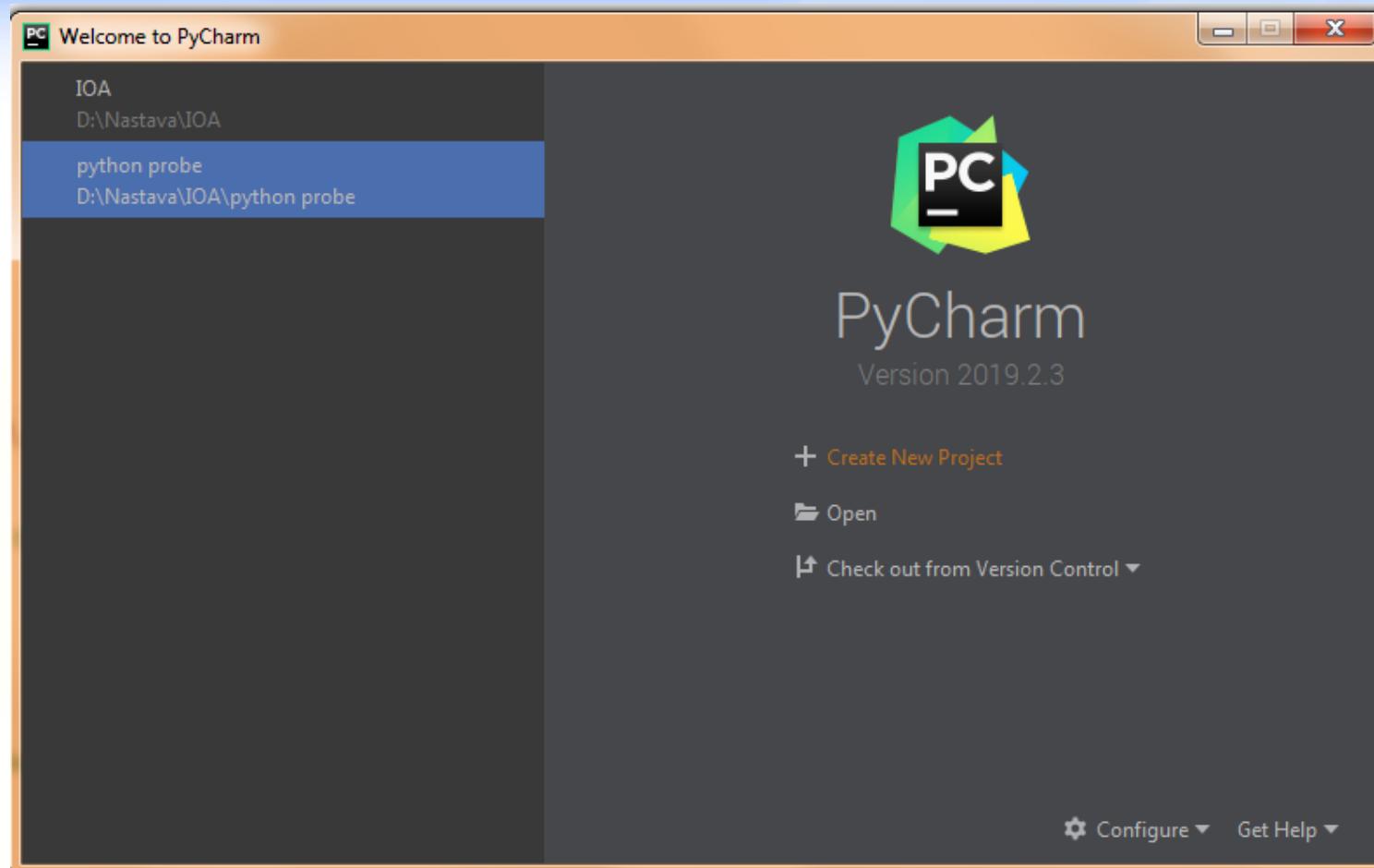


The screenshot shows the Microsoft Visual Studio interface with the title bar "KompleksniBrojevi - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar below has icons for file operations like Open, Save, and Build. The status bar indicates "Release x64 Local Windows Debugger". The code editor window displays "Source.cpp*" under the project "KompleksniBrojevi". The code itself is as follows:

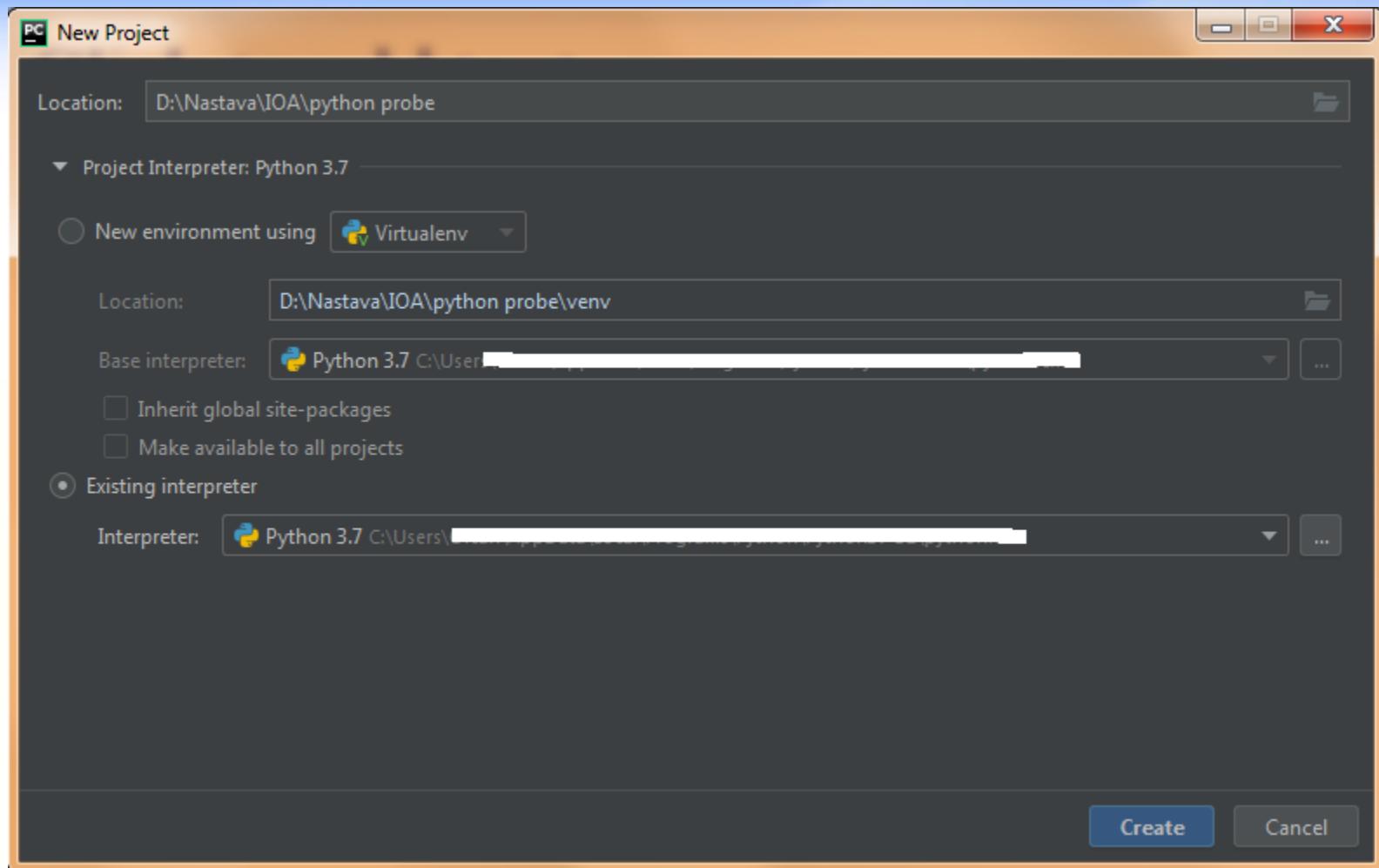
```
1 #include <stdio.h>
2 #include <iostream>
3 #include <complex>
4
5 using namespace std;
6
7 double fun(double x1, double x2)
8 {
9     return pow((1.0 - x1), 2) + 100.0*pow((x1 - x2*x2), 2.0);
10 }
11
12 void main()
13 {
14     FILE* out;
15     fopen_s(&out, "Rozenbrok.txt", "w");
16     for (double x1 = -2.0; x1 <= 2.0; x1 += 0.01)
17         for (double x2 = -2.0; x2 <= 2.0; x2 += 0.01)
18         {
19             double y = fun(x1, x2);
20             fprintf(out, "%.5f %.5f %.5f\n", x1, x2, y);
21         }
22     fclose(out);
23     system("pause");
24 }
```

The lines of code from 14 to 20 are highlighted with a red rectangle, and the line from 20 to 21 is highlighted with a blue rectangle, indicating specific sections of the code being discussed.

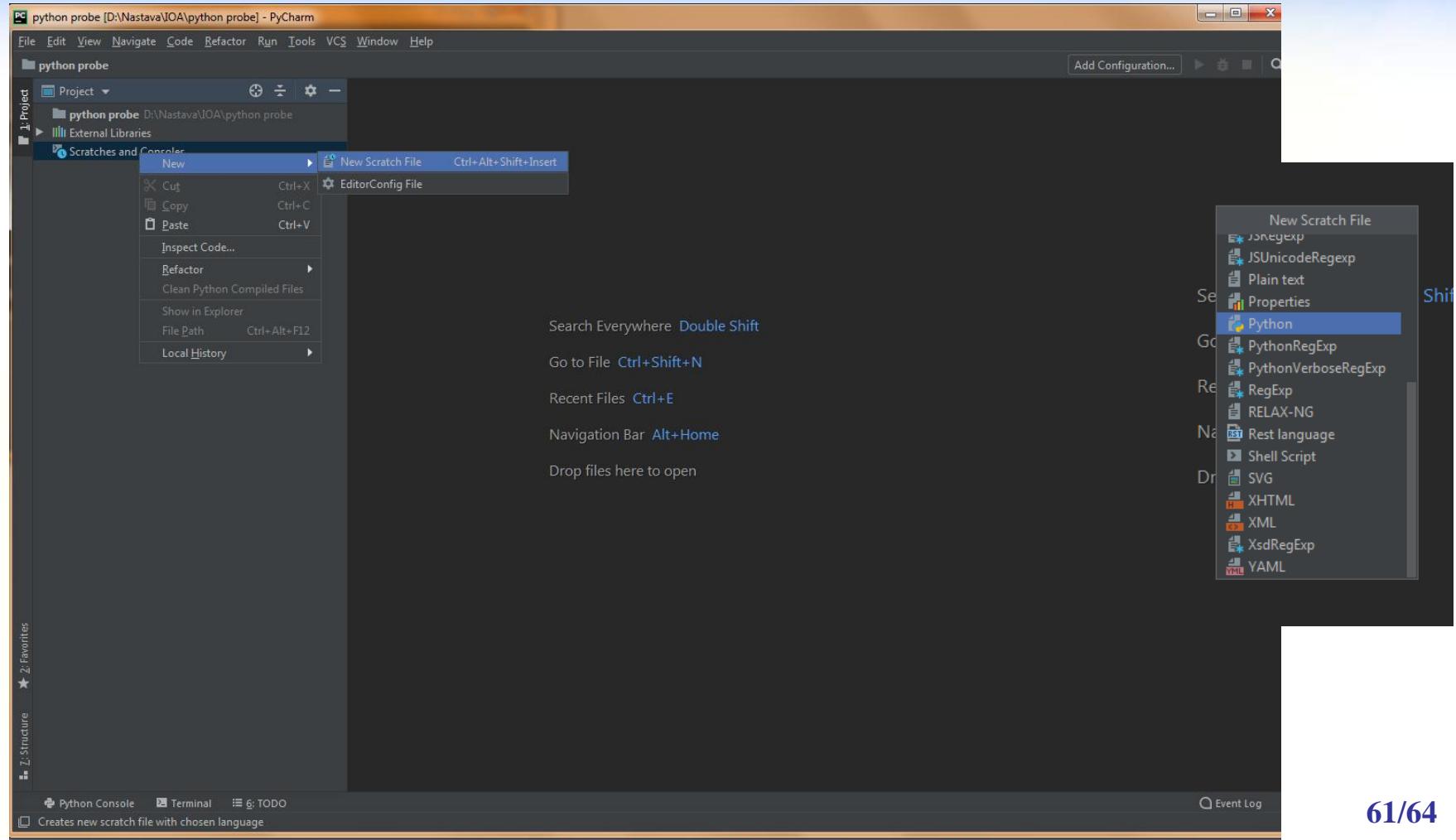
Python 3.7 (PyCharm окружење)



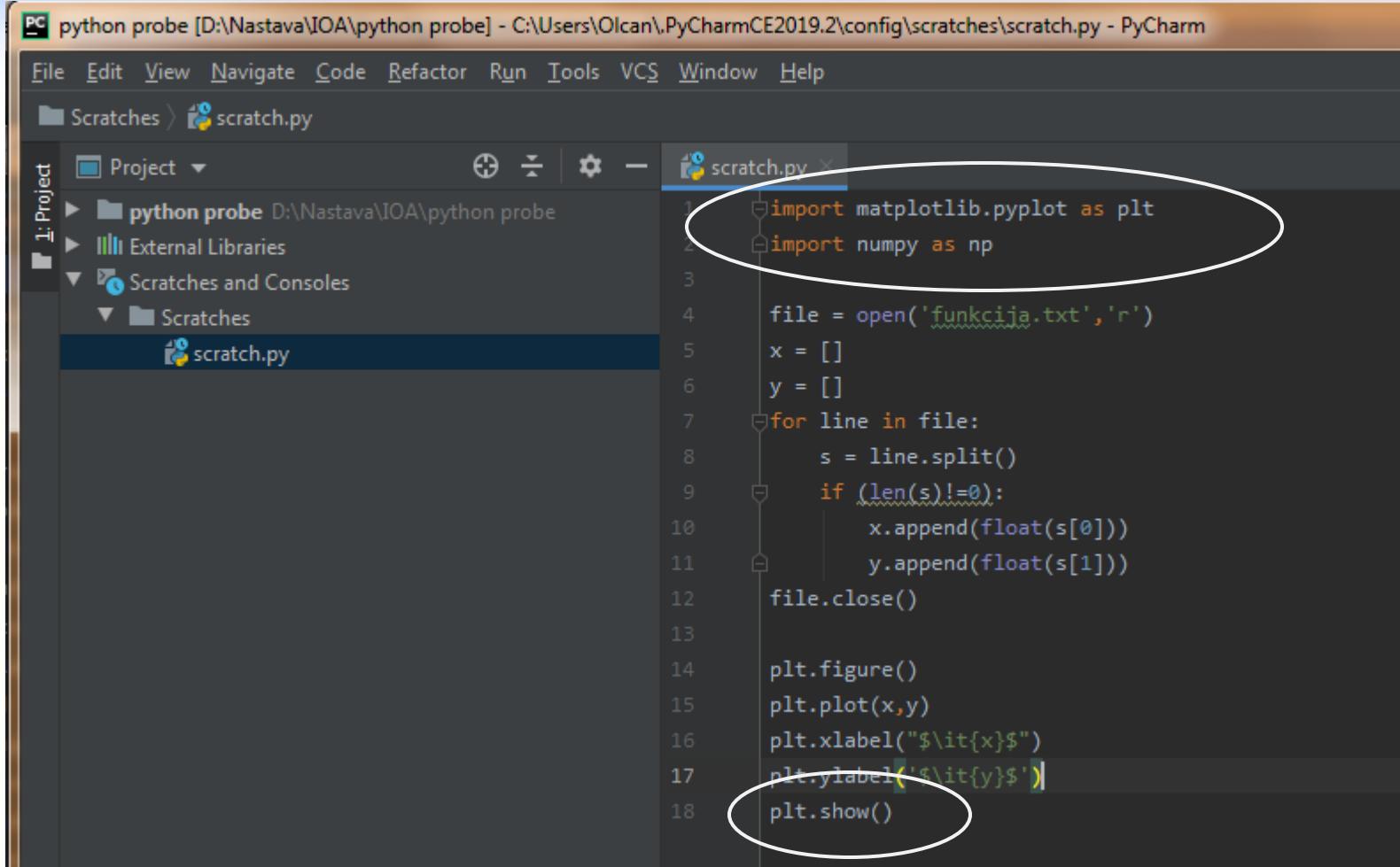
Нови пројекат



New scratch file



Додавање библиотека

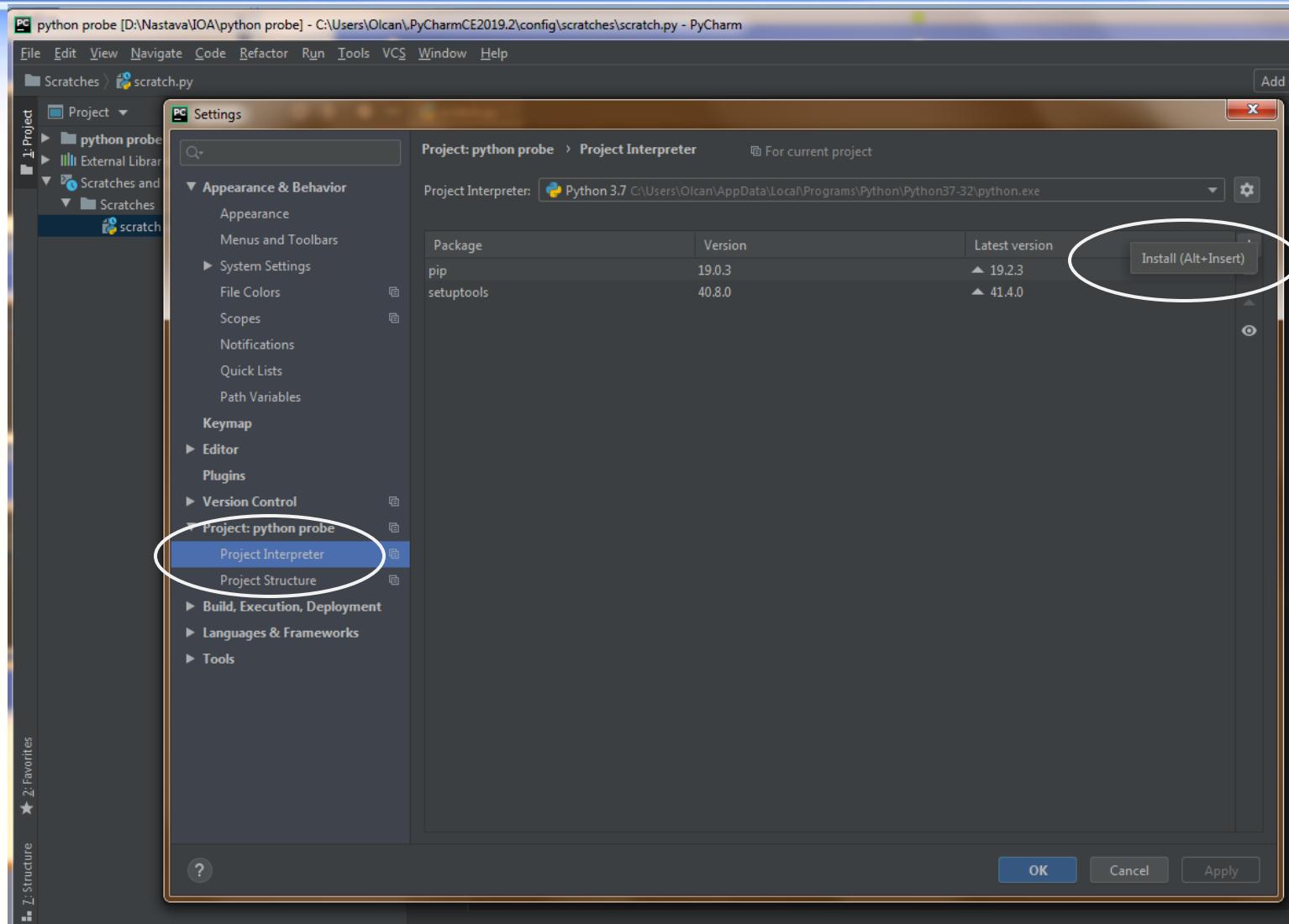


The screenshot shows the PyCharm IDE interface with the following details:

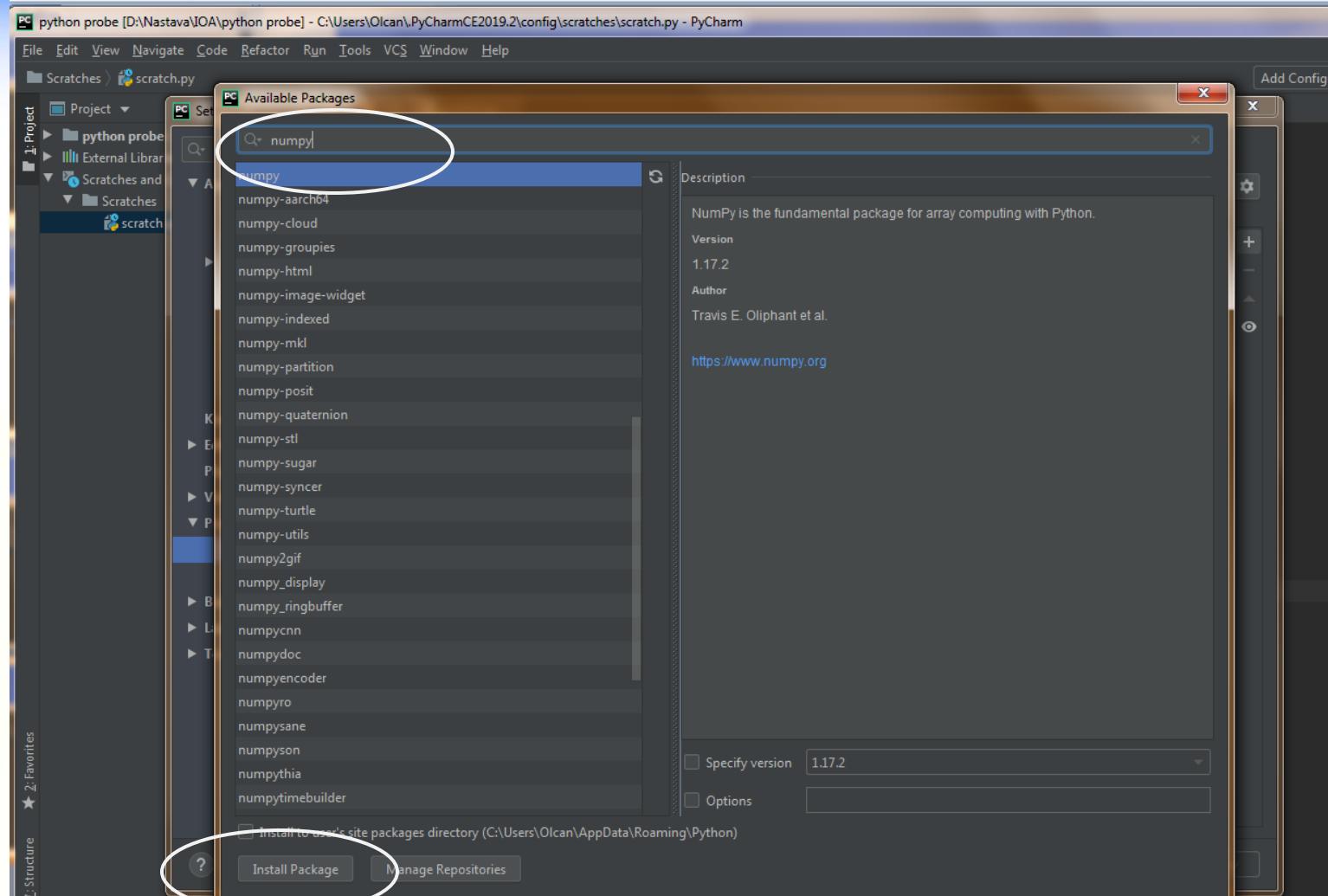
- Title Bar:** python probe [D:\Nastava\IOA\python probe] - C:\Users\Olcan\PyCharmCE2019.2\config\scratches\scratch.py - PyCharm
- Menu Bar:** File Edit View Navigate Code Refactor Run Tools VCS Window Help
- Toolbars:** Standard and Project
- Project Explorer:** Shows a project named "python probe" at D:\Nastava\IOA\python probe, an External Libraries section, and a Scratches and Consoles section containing a "Scratches" folder with "scratch.py".
- Code Editor:** The file "scratch.py" is open, displaying the following Python code:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 file = open('funkcija.txt','r')
5 x = []
6 y = []
7 for line in file:
8     s = line.split()
9     if (len(s)!=0):
10         x.append(float(s[0]))
11         y.append(float(s[1]))
12 file.close()
13
14 plt.figure()
15 plt.plot(x,y)
16 plt.xlabel("$\it{x}$")
17 plt.ylabel("$\it{y}$")
18 plt.show()
```
- Annotations:** Two white ovals highlight specific parts of the code:
 - The first oval encloses the two import statements: `import matplotlib.pyplot as plt` and `import numpy as np`.
 - The second oval encloses the label assignment in line 17: `plt.ylabel("\it{y}")`.

Добавање библиотека



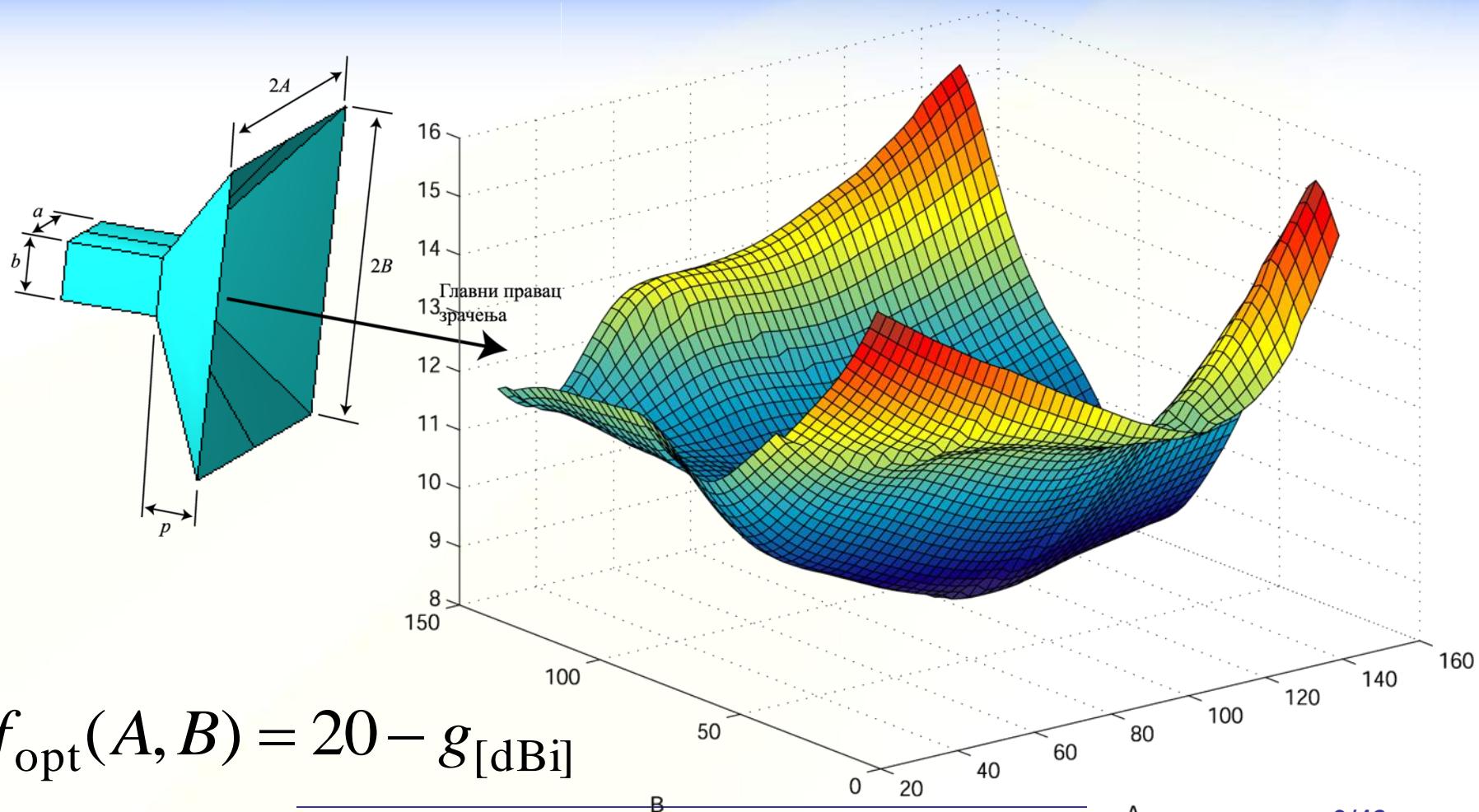
Инсталација библиотека



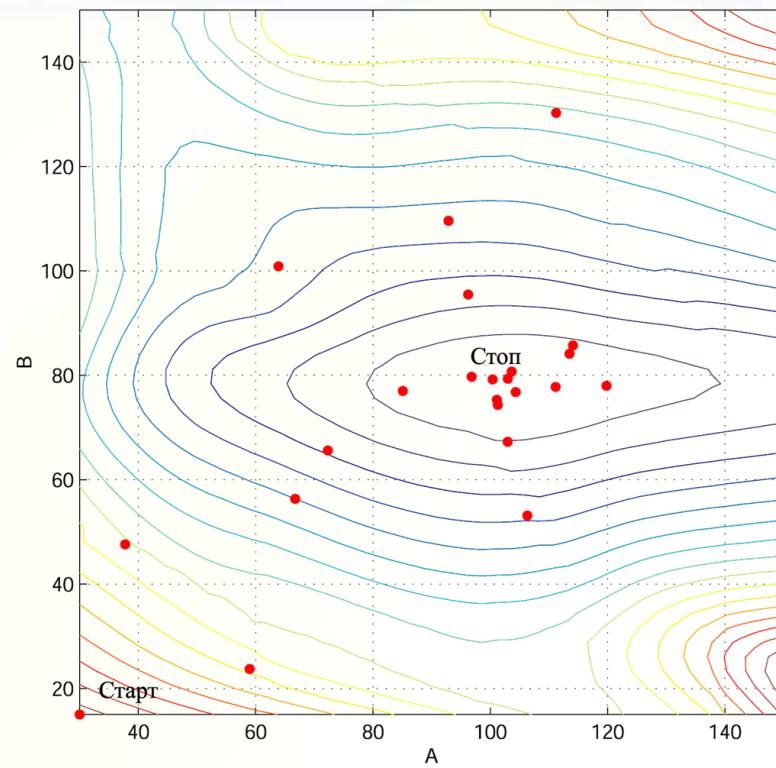
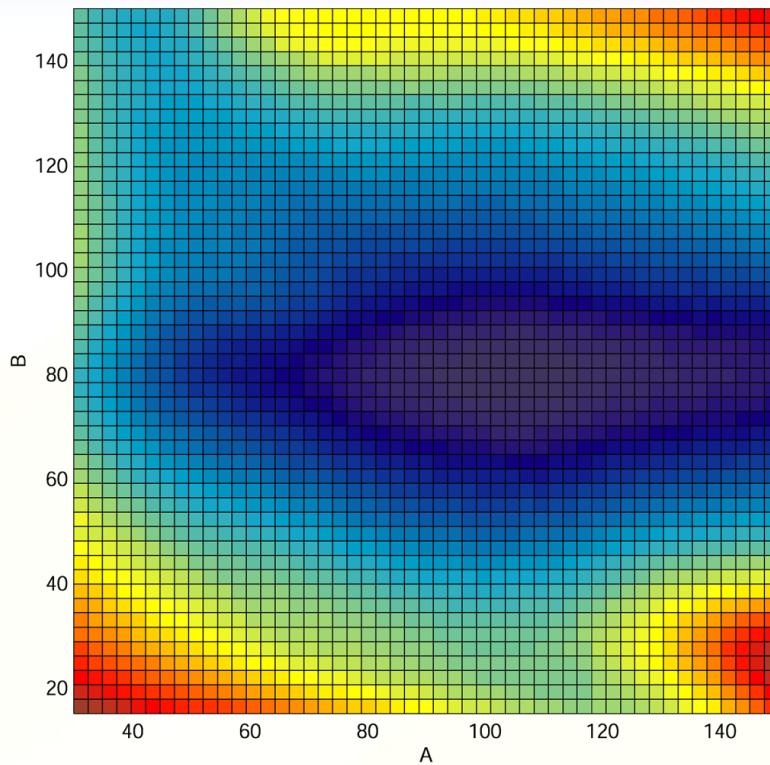
Оптимизациона функција

- 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 зnamо 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

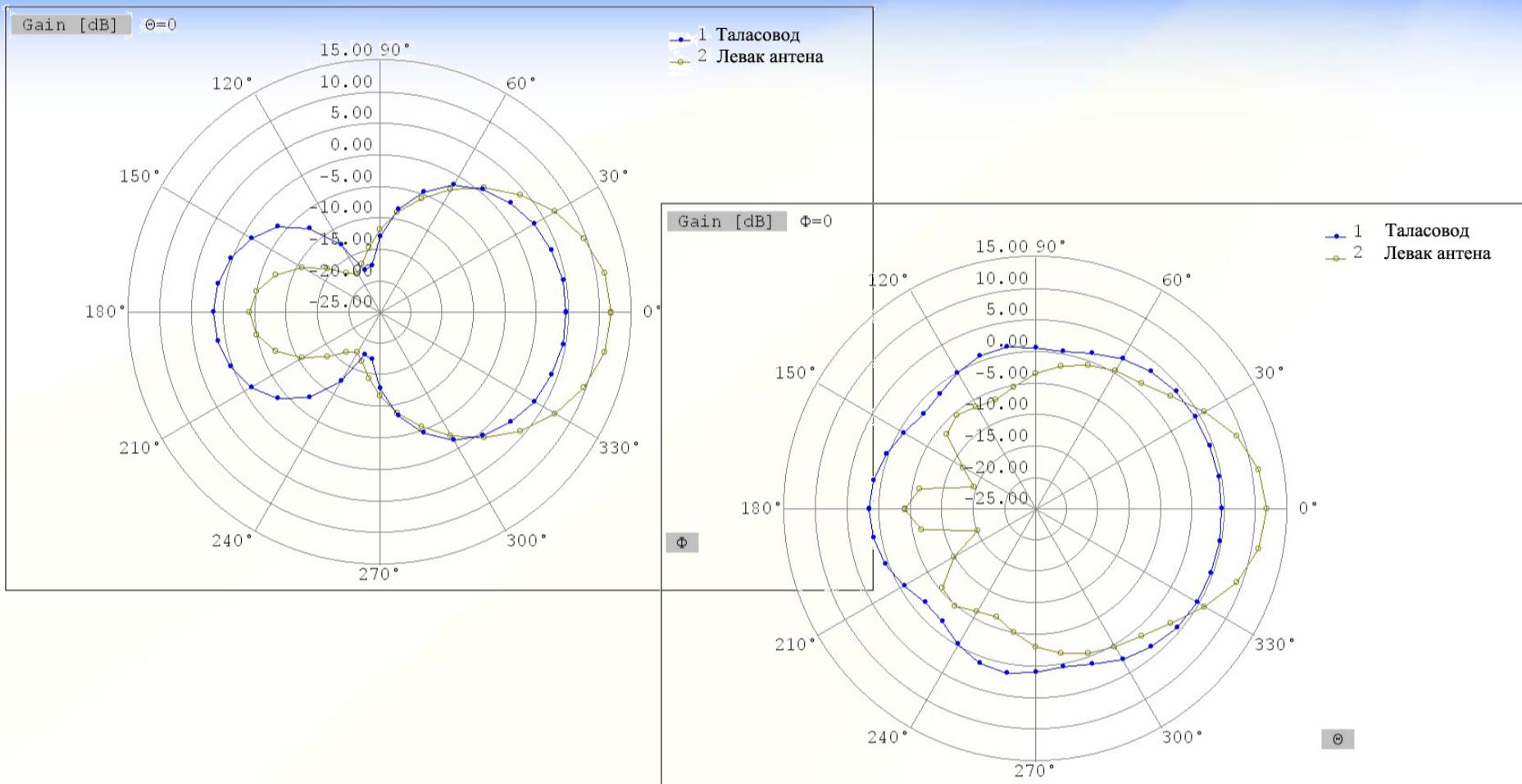
Пример из праксе једне оптимизационе функције (NLP)



Описна функција и оптимални отвор левак антене



Оптимална левак антена



Како проћенити особине и перформансе опт. алгоритма?

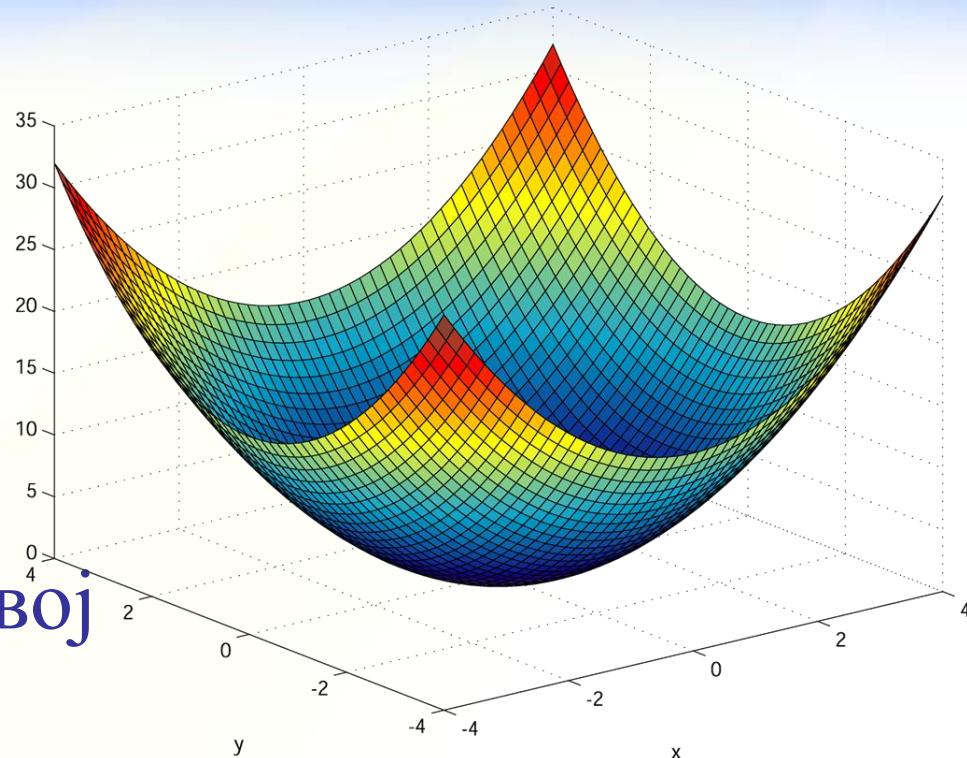
- Ситуација #1 (инжењерска пракса):
постављен је оптимизациони проблем и потребно је решити га
 - бирали смо алгоритам на основу претходног искуства и информација које имамо о датом проблему
 - пробали сме све доступне оптимизационе алгоритме и користили оне који имају најбоље перформансе
- Ситуација #2 (истраживање / наука):
потребно је сагледати особине оптимизационог алгоритма
 - тестирали смо алгоритме на познатим оптимизационим функцијама (проблемима) и поредили 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)

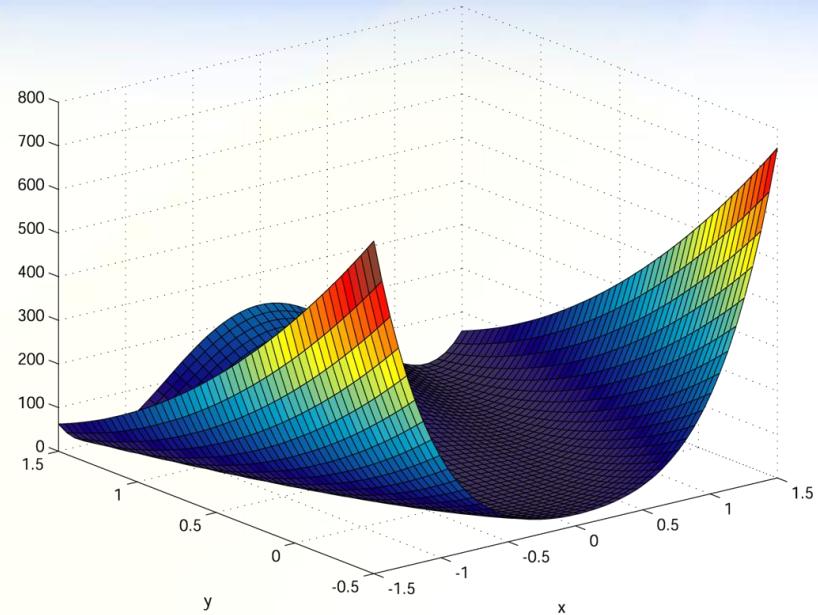


Розенброкова функција

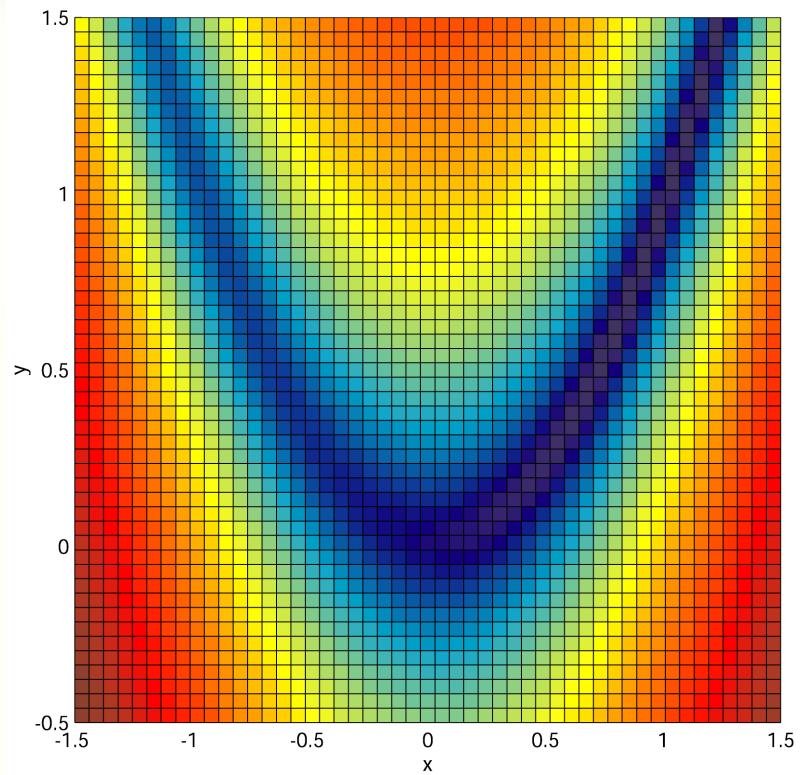
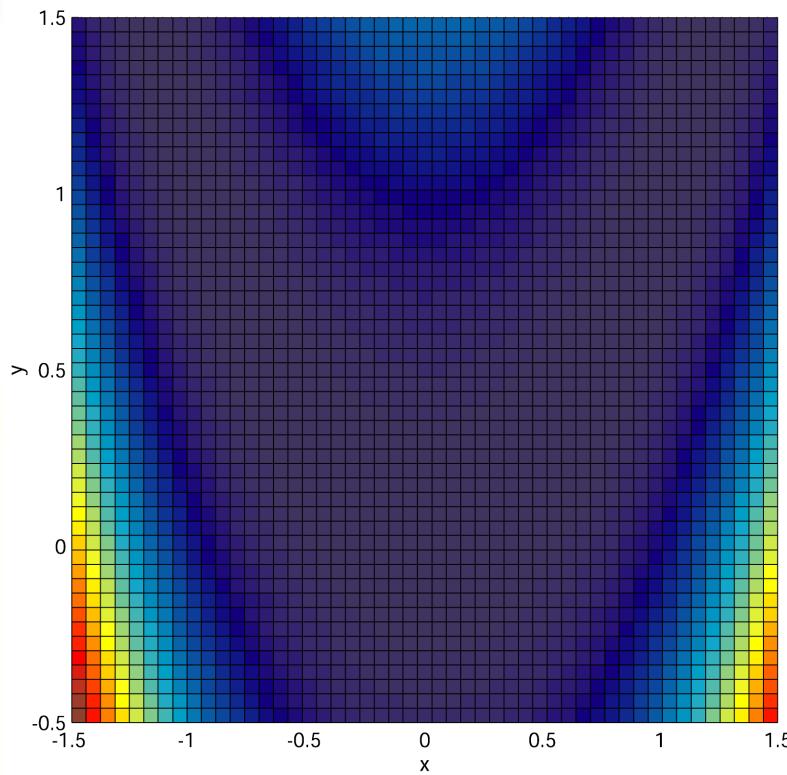
- Аналитички израз за D димензија

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

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

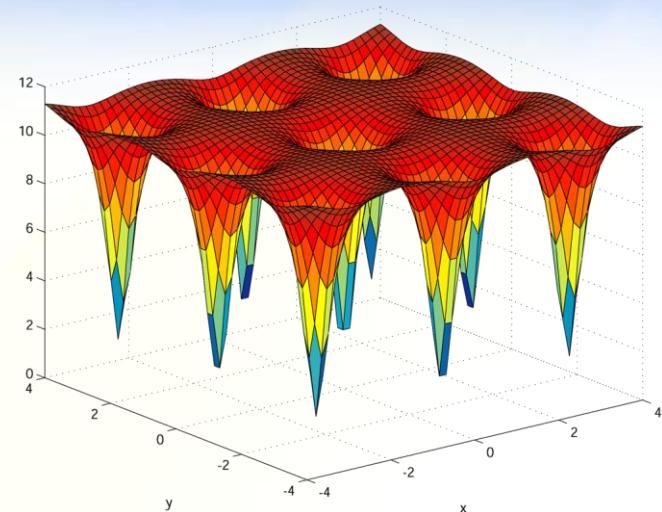


Розенброкова функција: поглед одозго и лог-размера

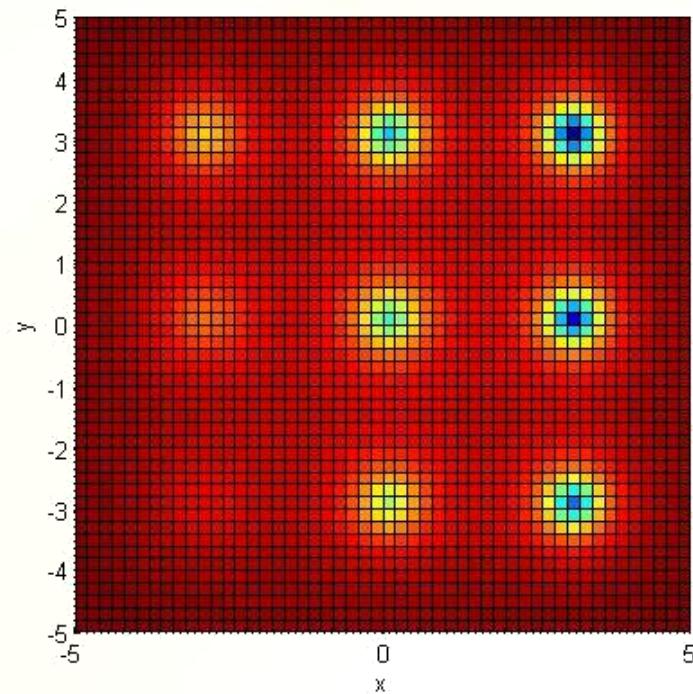
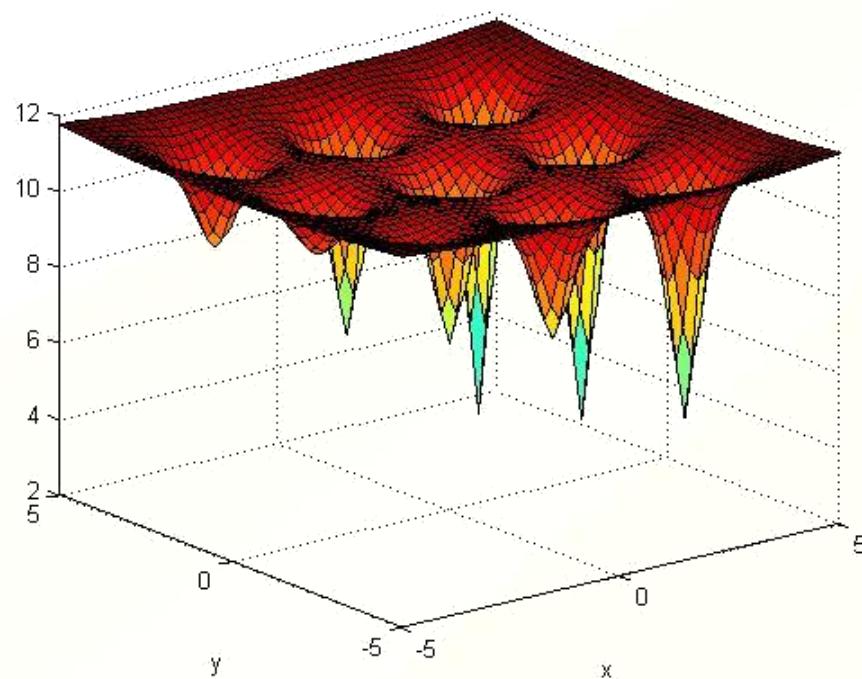


Шекелова функција (Fox-hole function)

- 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

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

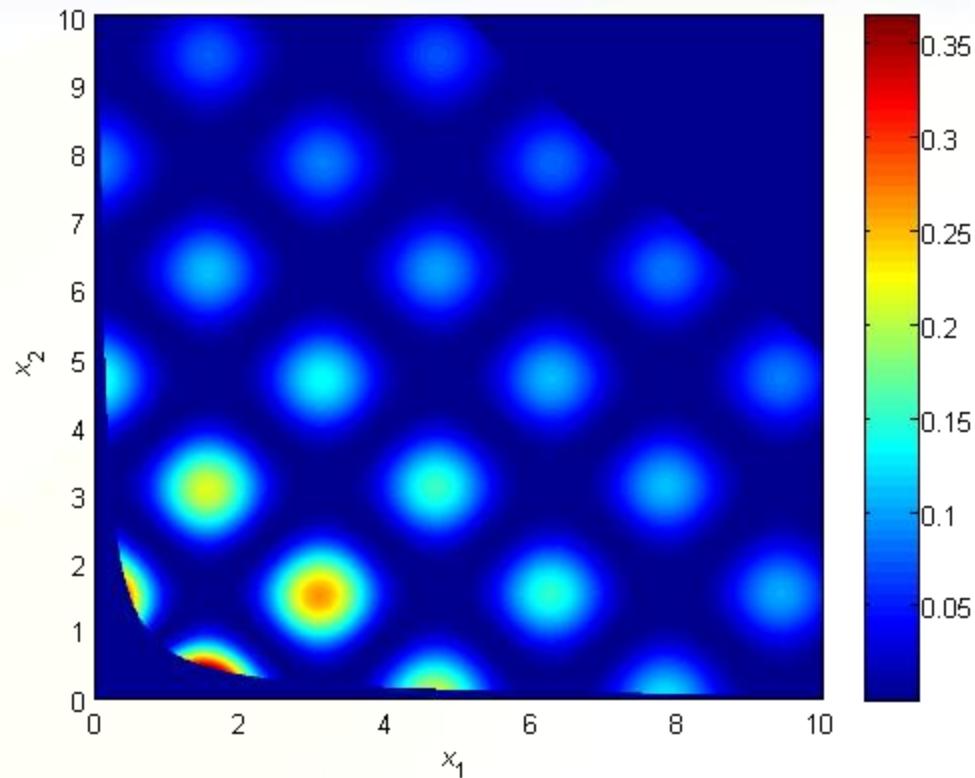
- Под условима:

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

- Нула ако нису испуњени услови

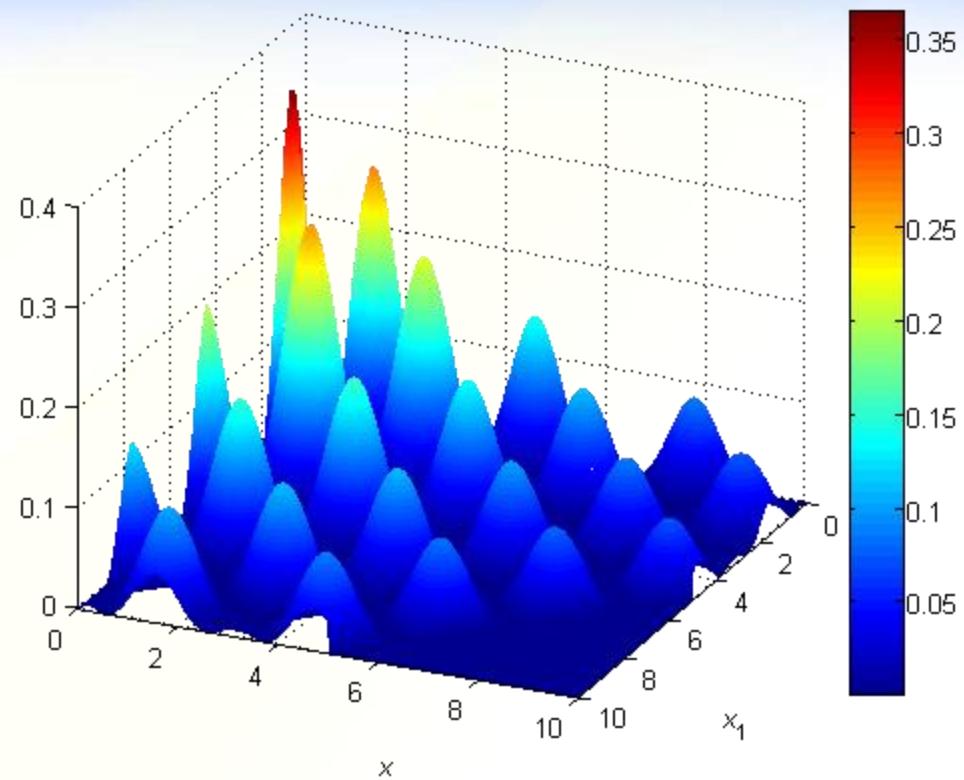
Функција G2 са 2 променљиве

- 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

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

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

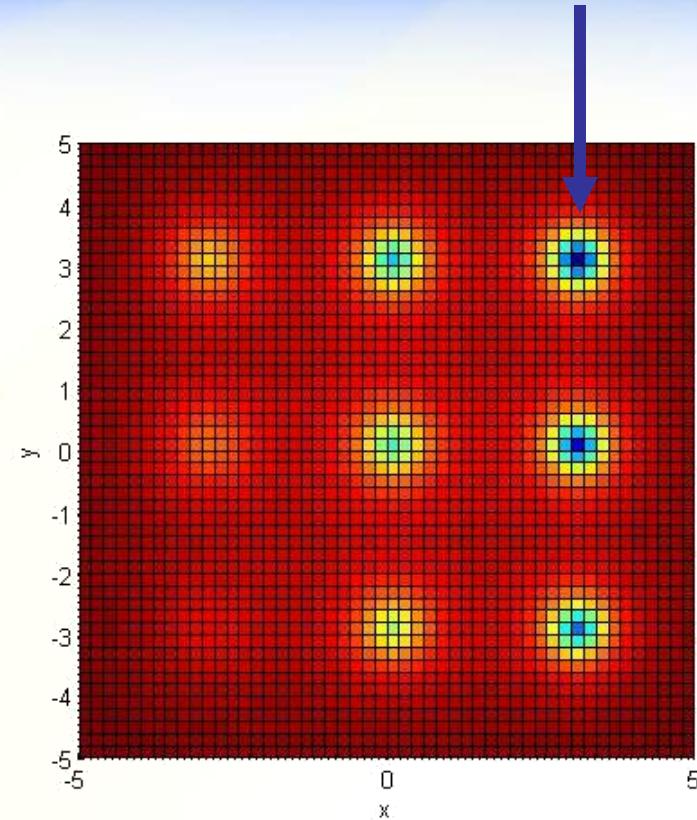
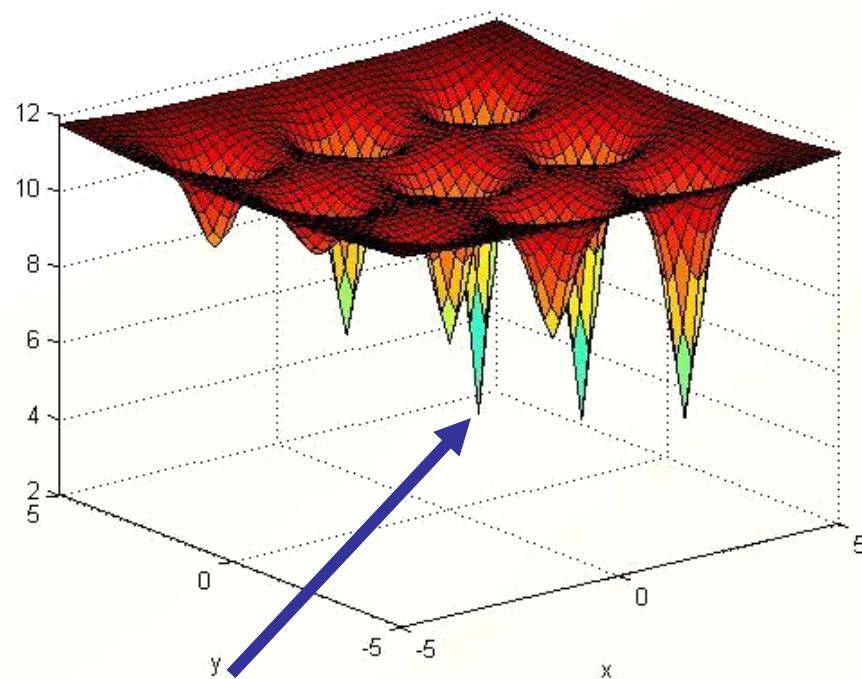
Дефиниција оптимизационог проблема са једним критеријумом

- Пронађи 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
$$\forall \mathbf{y} \in F, f(\mathbf{x}_g) \leq f(\mathbf{y})$$
- Глобални минимум је најбоље могуће решење
- Да ли постоји само један глобални минимум?
 - Може постојати један или више глобалних минимума

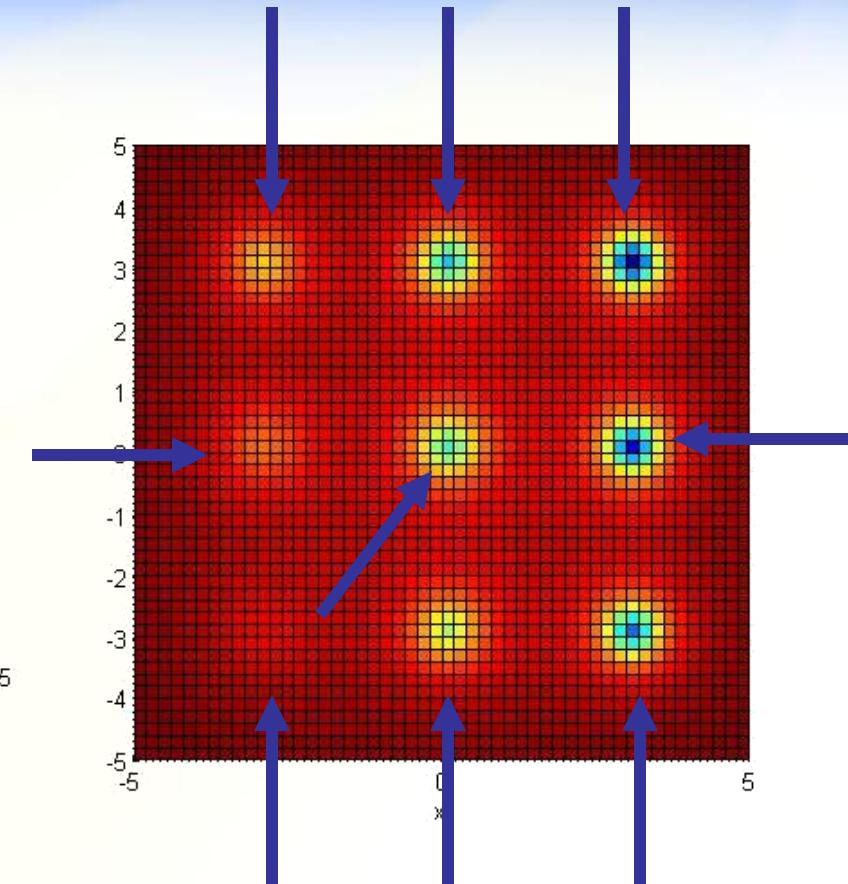
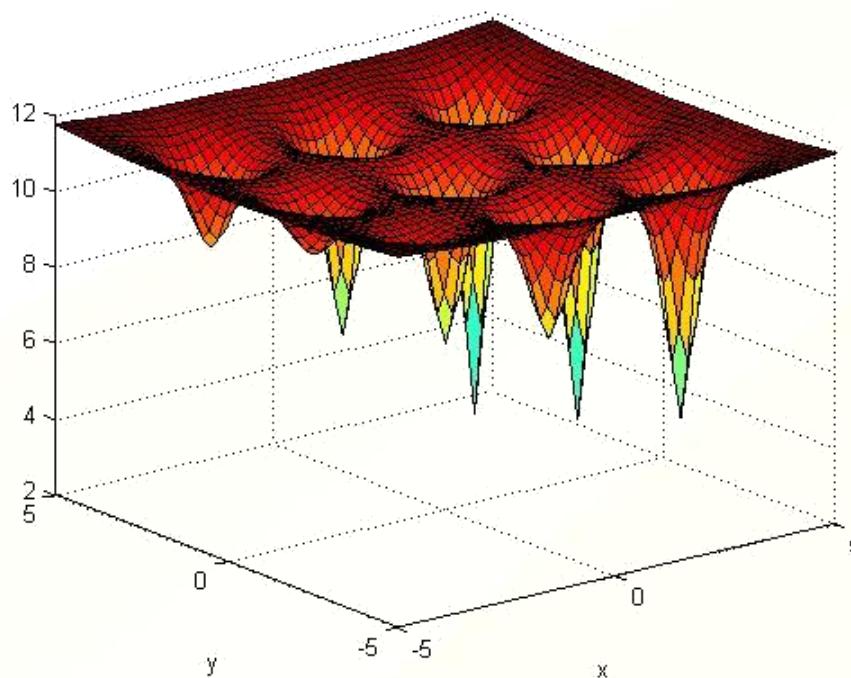
Пример глобалног минимума



Локално решење: формална дефиниција

- Епсилон околина решења \mathbf{x} је $N(\mathbf{x}) = \{\mathbf{y} \in S, dist(\mathbf{x}, \mathbf{y}) \leq \varepsilon\}$
- За NLP класу проблема:
Еуклидовско растојање $dist(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^D (x_k - y_k)^2}$
- За SAT класу проблема:
Хамингово растојање $dist(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^D x_k \oplus y_k$
- За TSP класу проблема:
број различитих прелазака $dist(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D-1} \begin{cases} 1, & x_k \leftrightarrow x_{k+1} \in \mathbf{y} \\ 0, & x_k \leftrightarrow x_{k+1} \notin \mathbf{y} \end{cases}$
- $dist(\mathbf{x}, \mathbf{y})$ 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!)



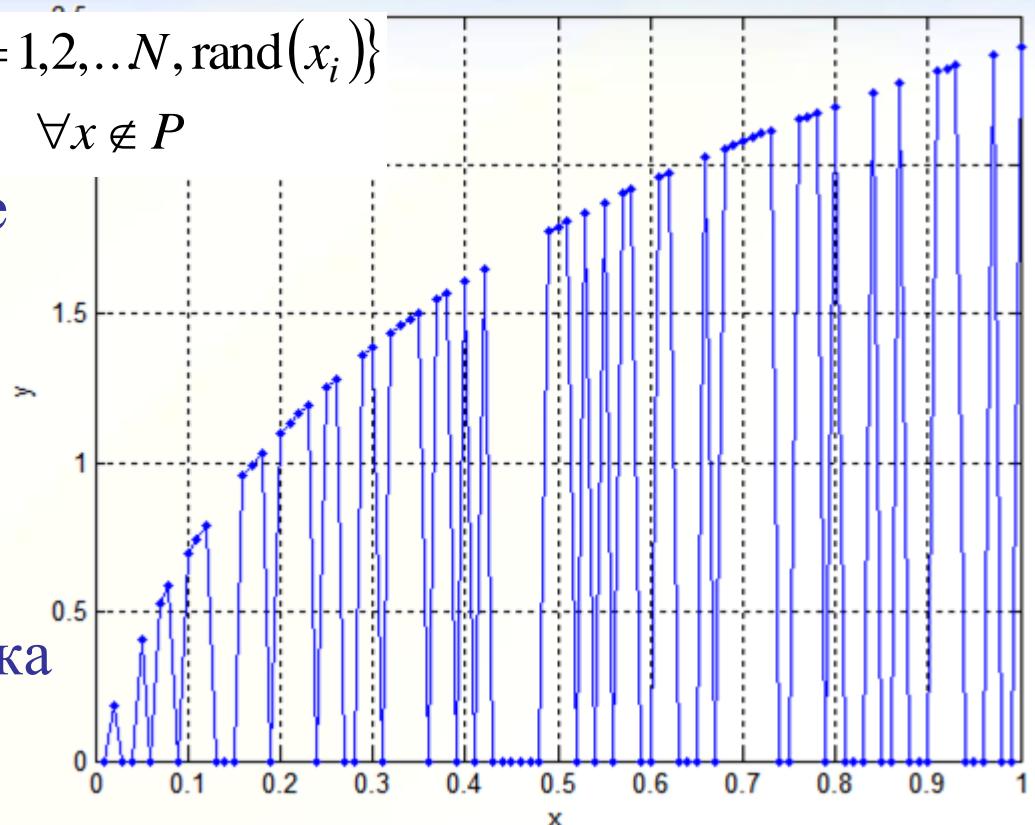
NFL теорема

- 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

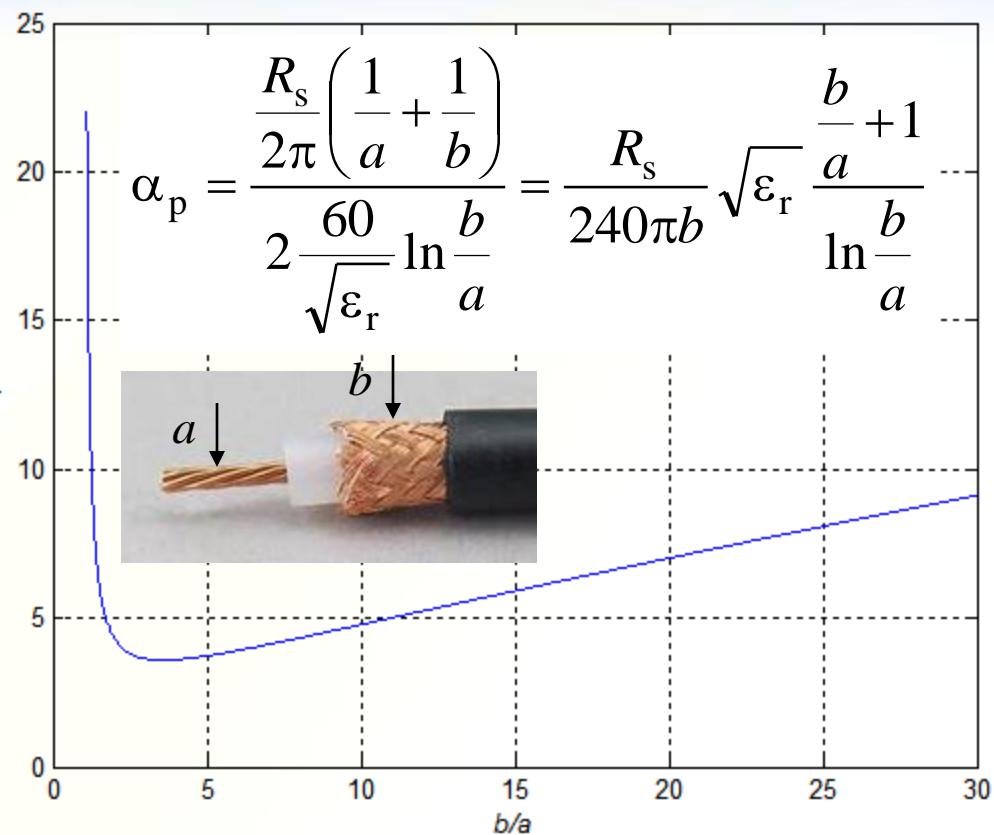


Критички осврт на NFL теорему (познати докази када НЕ важи)

- 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)
- Пронађи 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)



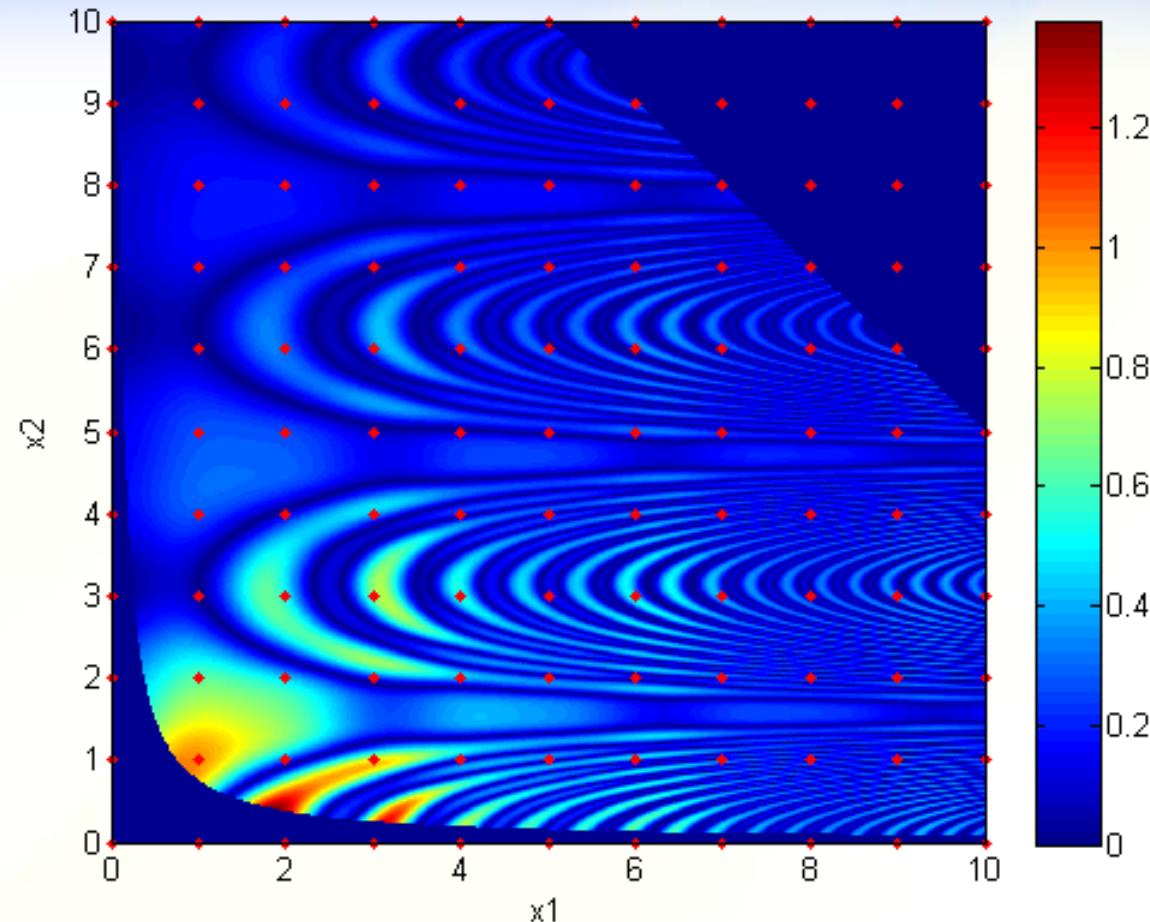
Инжењерска пракса: неки алгоритми су БОЉИ

- Пракса:
 - оптимизациона функција је смислено формулисана
 - оптимизациони проблем је једнозначен
 - 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 ... 111n
1121 1122 ... 112n
...
 n nn1 n nn2 ... n nnn

```
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;

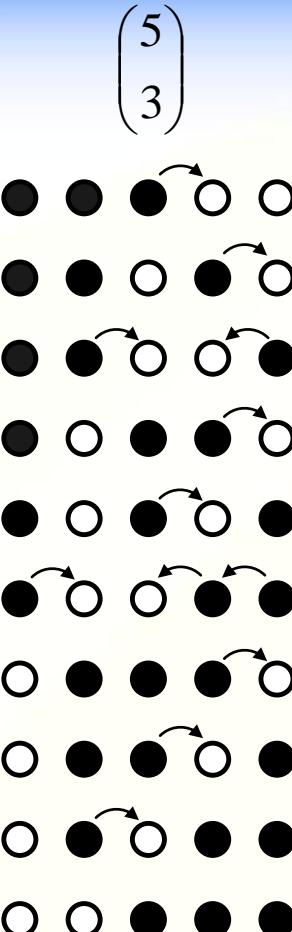
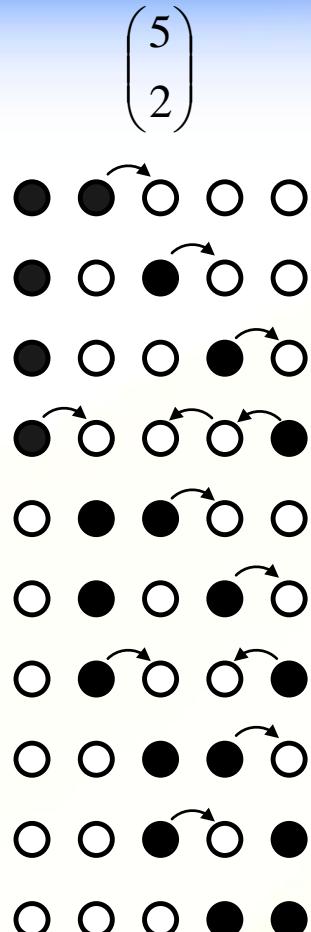
    do
    {
        for(int i=0; i<k; i++)
            printf("%5d ",P[i]+1);
        printf("\n");

        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 елемента
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 у десно 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



```
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)

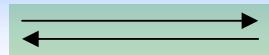
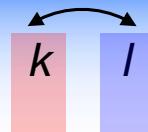
- На колико начина је могуће распоредити n елемената на n места (без понављања елемената)?
- Број различитих распореда је $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$
- Пун назив: пермутације скупа X_n (енг: permutation of set)
- Алгоритам изабран по критеријуму **најбржег извршавања** 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
 $P = \{P[0] \ P[1] \ \dots P[n - 1]\}$
- Пронађи највеће k ($0 \leq k < n - 1$)
тако да је $P[k] < P[k+1]$
- Пронађи 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	k	4	k	5		\rightarrow	1	2	3	5	4	\rightarrow	1	2	3	5	4
1	2	k	3	5	k	4		\rightarrow	1	2	4	5	3	\rightarrow	1	2	4	3	5
1	2	4	k	3	k	5		\rightarrow	1	2	4	5	3	\rightarrow	1	2	4	5	3
1	k	4	k	5	k	3		\rightarrow	1	2	5	4	3	\rightarrow	1	2	5	3	4
1	2	5	k	3	k	4		\rightarrow	1	2	5	4	3	\rightarrow	1	2	5	4	3
k			k		k			\rightarrow	1	3	5	4	2	\rightarrow	1	3	2	4	5
1	3	2	k	4	k	5		\rightarrow	1	3	2	5	4	\rightarrow	1	3	2	5	4

Имплементација у C/C++

```
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
48    return 1;
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])
```

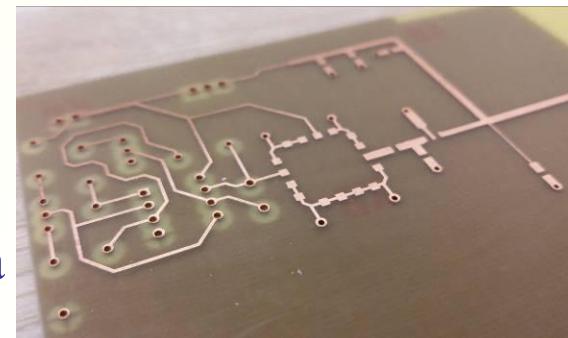
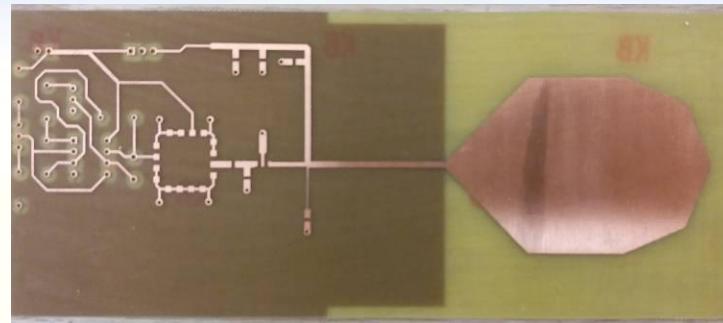
- **Wolfram Mathematica**

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

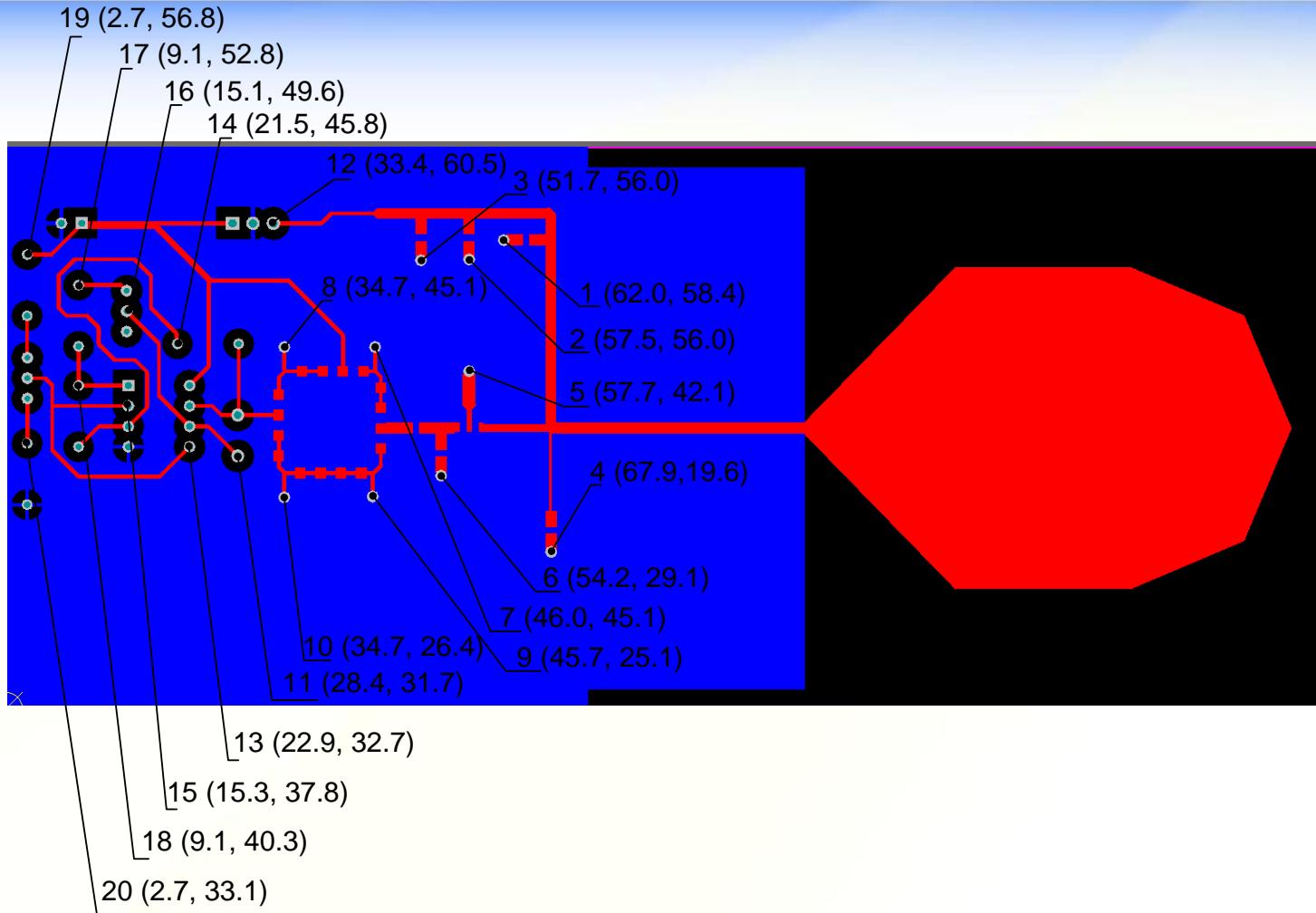
Задатак за вежбе

(Оптимално бушење плочице)

- Штампана плочица се израђује аутоматски
- На њој постоји више рупа истог пречника које је потребно избушити
- Аутоматска бургија 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 бургија (и минимизирати време израде)
- Бургија 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 [мм] на штампаниј плочици



Задатак за вежбе (детаљи 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 града је L_2 норма (Еуклидовско растојање)
 - Бургија 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
- * [5] БОНУС: написати ПАРАЛЕЛНУ имплементацију програма за потпуну претрагу TSP проблема и помоћу њега пронаћи најкраћи пут за првих 15 рупа ($15! \approx 1,3 \times 10^{12}$)

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

Партиције природног броја

- Партиција (подела/разбијање) природног броја n је низ позитивних природних бројева $a_1 \geq a_2 \geq \dots \geq a_m \geq 1$ тако да је $n = a_1 + a_2 + \dots + a_m$
- При томе је: $1 \leq m \leq n$ и $n \geq 1$
- Лексикографски поредак
 - прва партиција је $a_1 = n$
 - последња партиција је $1 + 1 + \dots + 1 = n$

Све партиције $n = 7$

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

C/C++ код за генерирање свих партиција природног броја

- Партиција се чува у низу $p[0], p[1], \dots, p[n-1]$
- Уколико је $p[k] > 0$ тада је $p[k]$ члан партиције
- Пронађи највеће k тако да је $p[k] > 1$ и нађи број јединица q
- $p[k] = p[k] - 1, q = q+1$
 - Ако је $p[k] \geq q$: $p[k+1] = q$
 - Ако је $q > p[k]$:
 $p[k+1] = p[k]$
 $q = q - p[k], k=k+1$
поновити корак докле год може

```
1 #include <stdio.h>
2 
3 bool next_partition(int* p, int n)
4 {
5     int k,i;
6     int q=0;
7 
8     if(p[0]==0) // the first one
9     {
10        p[0]=n;
11        return true;
12    }
13    else // finding k
14    {
15        for(i=0; i<n; i++)
16        {
17            if(p[i]!=0) k=i;
18            else break;
19        }
20    }
21    while (k>=0 && p[k]==1)
22    {
23        q++;
24        k--;
25    }
26 
27    if (k<0) return false;
28 
29    p[k]--;
30    q++;
31 
32    while(q > p[k])
33    {
34        p[k+1] = p[k];
35        q = q - p[k];
36        k++;
37    }
38 
39    p[k+1] = q;
40    k++;
41 
42    for(i=k+1; i<n; i++)
43        p[i]=0;
44 
45    return true;
46 }
47 
```

```
95 void driver_next_partition(void)
96 {
97     int n=7;
98     int* p=new int [n];
99 
100    for(int i=0; i<n; i++)
101        p[i]=0;
102 
103    while(next_partition(p, n))
104    {
105        //print current partition
106        for(int i=0; i<n && p[i]>0; i++)
107            printf("%2d ", p[i]);
108        printf("\n");
109    }
110    delete [] p;
111 }
```

Илустрација алгоритма

Све партиције $n = 7$

7	k=0, q=0+1: p[0]=7-1, p[1]=q=1
6 1	k=0, q=1+1: p[0]=6-1, p[1]=q=2
5 2	k=1, q=0+1: p[1]=2-1, p[2]=q=1
5 1 1	k=0, q=2+1: p[0]=5-1, p[1]=q=3
4 3	k=1, q=0+1: p[1]=3-1, p[2]=q=1
4 2 1	k=1, q=1+1: p[1]=2-1, p[2]=q=1
4 1 1 1	k=0, q=3+1: p[0]=4-1, p[1]=p[0], p[2]=q-p[1]
3 3 1	...
3 2 2	
3 2 1 1	
3 1 1 1 1	k=0, q=3+1: p[0]=3-1, p[1]=p[0], p[2]=q-p[1]
2 2 2 1	...
2 2 1 1 1	
2 1 1 1 1 1	
1 1 1 1 1 1 1	k=-1, q=7: нема наредне партиције

Укупан број партиција: не постоји аналитички израз

The man who
knew infinity

$$P(n) \approx \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$$

P(1)=	1	P(21)=	792	P(41)=	44583	P(61)=	1121505
P(2)=	2	P(22)=	1002	P(42)=	53174	P(62)=	1300156
P(3)=	3	P(23)=	1255	P(43)=	63261	P(63)=	1505499
P(4)=	5	P(24)=	1575	P(44)=	75175	P(64)=	1741630
P(5)=	7	P(25)=	1958	P(45)=	89134	P(65)=	2012558
P(6)=	11	P(26)=	2436	P(46)=	1055558	P(66)=	2323520
P(7)=	15	P(27)=	3010	P(47)=	124754	P(67)=	2679689
P(8)=	22	P(28)=	3718	P(48)=	147273	P(68)=	3087735
P(9)=	30	P(29)=	4565	P(49)=	173525	P(69)=	3554345
P(10)=	42	P(30)=	5604	P(50)=	204226	P(70)=	4087968
P(11)=	56	P(31)=	6842	P(51)=	239943	P(71)=	4697205
P(12)=	77	P(32)=	8349	P(52)=	281589	P(72)=	5392783
P(13)=	101	P(33)=	10143	P(53)=	329931	P(73)=	6185689
P(14)=	135	P(34)=	12310	P(54)=	386155	P(74)=	7089500
P(15)=	176	P(35)=	14883	P(55)=	451276	P(75)=	8118264
P(16)=	231	P(36)=	17977	P(56)=	526823	P(76)=	9289091
P(17)=	297	P(37)=	21637	P(57)=	614154	P(77)=	10619863
P(18)=	385	P(38)=	26015	P(58)=	715220	P(78)=	12132164
P(19)=	490	P(39)=	31185	P(59)=	831820	P(79)=	13848650
P(20)=	627	P(40)=	37338	P(60)=	966467	P(80)=	15796476

Партиције природног броја са тачно m чланова

- $a_1 \geq a_2 \geq \dots \geq a_m \geq 1$ тако да је $n = a_1 + a_2 + \dots + a_m$
- При томе је: $2 \leq m \leq n$ задато
- Једноставна имплементација:
прескочити партиције $P(n)$ чији је број чланова различит од m
 - Постоје и ефикасније имплементације

Све партиције

$n = 11 \ m = 4$

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

Партиције скупа

- Партиција скупа S је подела S на подскупове S_m , $1 \leq m \leq |S|$ тако да важи:

$$S = S_1 \cup S_2 \cup \dots \cup S_m$$

$$S_i \neq \{\emptyset\}, \quad 1 \leq i \leq m$$

$$i \neq j \Rightarrow S_i \cap S_j = \{\emptyset\}, \quad 1 \leq i, j \leq m$$

- $|S|$ је број елемената скупа S

Све партиције

$$S = \{1, 2, 3, 4\}$$

$$\{1, 2, 3, 4\}$$

$$\{1, 2, 3\}, \{4\}$$

$$\{1, 2, 4\}, \{3\}$$

$$\{1, 2\}, \{3, 4\}$$

$$\{1, 2\}, \{3\}, \{4\}$$

$$\{1, 3, 4\}, \{2\}$$

$$\{1, 3\}, \{2, 4\}$$

$$\{1, 3\}, \{2\}, \{4\}$$

$$\{1, 4\}, \{2, 3\}$$

$$\{1\}, \{2, 3, 4\}$$

$$\{1\}, \{2, 3\}, \{4\}$$

$$\{1, 4\}, \{2\}, \{3\}$$

$$\{1\}, \{2, 4\}, \{3\}$$

$$\{1\}, \{2\}, \{3, 4\}$$

$$\{1\}, \{2\}, \{3\}, \{4\}$$

Пресликање партиције скупа

- Посматрајмо низ a_1, a_2, \dots, a_n који задовољава следеће услове:
 $a_1 = 0$ и
 $a_{j+1} \leq 1 + \max(a_1, \dots, a_j),$
 $1 \leq j < n$
- Сваки следећи елемент низа може највише за 1 да буде већи од највећег елемента пре њега у низу
(енг: restricted growth string)
- Једнозначно пресликање RGS на партицију скупа!
- Број партиција: $\max(a_i) + 1$

Сви могући низови $n=4$		Све партиције $S = \{1, 2, 3, 4\}$
0 0 0 0	->	{1, 2, 3, 4}
0 0 0 1	->	{1, 2, 3}, {4}
0 0 1 0	->	{1, 2, 4}, {3}
0 0 1 1	->	{1, 2}, {3, 4}
0 0 1 2	->	{1, 2}, {3}, {4}
0 1 0 0	->	{1, 3, 4}, {2}
0 1 0 1	->	{1, 3}, {2, 4}
0 1 0 2	->	{1, 3}, {2}, {4}
0 1 1 0	->	{1, 4}, {2, 3}
0 1 1 1	->	{1}, {2, 3, 4}
0 1 1 2	->	{1}, {2, 3}, {4}
0 1 2 0	->	{1, 4}, {2}, {3}
0 1 2 1	->	{1}, {2, 4}, {3}
0 1 2 2	->	{1}, {2}, {3, 4}
0 1 2 3	->	{1}, {2}, {3}, {4}

C/C++ код за генерирање свих партиција скупа

- RGS: $a_i = 0, 0 \leq i \leq n - 1$
- Максималне вредности елемената $b_0 = 0$,
 $b_i = \max(b_{i-1}, a_{i-1} + 1)$
 $1 \leq i \leq n - 1$
- Полазејќи од последњег елемента a_i проверити да ли $a_i \leq b_i$
 - може: $a_i = a_i + 1$ и одредити нове вредности за све b_i
 - не може: $a_i = a_i + 1$ и поновити корак за елемент $i - 1$
- Крај је када нема елемента који се може повеќати за један

```
45 bool next_partition_of_set(int* a, int n)
46 {
47     int j;
48     int* b = new int [n];
49
50     for(j=0; j<n; j++)
51         b[j]=j==0 ? 0 : std::max(b[j-1],a[j-1]+1);
52
53     for(j=n-1; j>=0; j--)
54     {
55         if(a[j]<b[j])
56         {
57             a[j]++;
58             break;
59         }
60         else
61             a[j]=0;
62     }
63     if(j==-1)
64         return false;
65
66     delete [] b;
67     return true;
68 }

70 void driver_next_partition_of_set(void)
71 {
72     int n=4;
73     int* a = new int [n];
74     int i;

75     for(i=0; i<n; i++)
76         a[i]=0;

77     do
78     {
79         for(i=0; i<n; i++)
80             printf("%2d ",a[i]);
81         printf("\n");
82     }while(next_partition_of_set(a, n));

83
84     delete [] a;
85 }
```

Партиција скупа на тачно m подскупова

- Важи све што и за партицију скупа једино је m задато
 - Једноставна имплементација:
узети само RGS које задовољавају $\max(a_i) + 1 = m$
 - Постоје ефикасније имплементације које су алгоритамски сложеније
 - Бонус [5 поена]: осмислiti и имплементирati ефикасан алгоритам за проналажење свих партиција скупа од N елемената на тачно m подскупова ($m \leq N$). Проверити резултате са следећег слајда тим кодом.
- Све партиције
 $S = \{1, 2, 3, 4\}$
 $m = 2$
-
- $\{1, 2, 3\}, \{4\}$
 $\{1, 2, 4\}, \{3\}$
 $\{1, 2\}, \{3, 4\}$
 $\{1, 3, 4\}, \{2\}$
 $\{1, 3\}, \{2, 4\}$
 $\{1, 4\}, \{2, 3\}$
 $\{1\}, \{2, 3, 4\}$

Број партиција скупа

- Тачно k подскупова
 - Стирлингови бројеви друге врсте $S(n, k) = S_n^{(k)} = \binom{n}{k}$

$S(1, 1) =$	1
$S(2, 1) =$	1
$S(3, 1) =$	1
$S(4, 1) =$	1
$S(5, 1) =$	1
$S(6, 1) =$	1
$S(7, 1) =$	1
$S(8, 1) =$	1
$S(9, 1) =$	1
$S(10, 1) =$	1
$S(2, 2) =$	1
$S(3, 2) =$	3
$S(4, 2) =$	7
$S(5, 2) =$	15
$S(6, 2) =$	31
$S(7, 2) =$	63
$S(8, 2) =$	127
$S(9, 2) =$	255
$S(10, 2) =$	511
$S(3, 3) =$	1
$S(4, 3) =$	6
$S(5, 3) =$	25
$S(6, 3) =$	90
$S(7, 3) =$	301
$S(8, 3) =$	966
$S(9, 3) =$	3025
$S(10, 3) =$	9330
$S(4, 4) =$	1
$S(5, 4) =$	10
$S(6, 4) =$	65
$S(7, 4) =$	350
$S(8, 4) =$	1701
$S(9, 4) =$	7770
$S(10, 4) =$	34105
$S(5, 5) =$	1
$S(6, 5) =$	15
$S(7, 5) =$	140
$S(8, 5) =$	1050
$S(9, 5) =$	6951
$S(10, 5) =$	42525
$S(6, 6) =$	1
$S(7, 6) =$	21
$S(8, 6) =$	266
$S(9, 6) =$	2646
$S(10, 6) =$	22827
$S(7, 7) =$	1
$S(8, 7) =$	28
$S(9, 7) =$	462
$S(10, 7) =$	5880
$S(8, 8) =$	1
$S(9, 8) =$	36
$S(10, 8) =$	750
$S(9, 9) =$	1
$S(10, 9) =$	45
$S(10, 10) =$	1

- Све могуће партиције
 - Белови бројеви

$$B_n = \sum_{k=1}^n S(n, k)$$

$B(1) =$	1	$B(8) =$	4140
$B(2) =$	2	$B(9) =$	21147
$B(3) =$	5	$B(10) =$	115975
$B(4) =$	15	$B(11) =$	678570
$B(5) =$	52	$B(12) =$	4213597
$B(6) =$	203	$B(13) =$	27644437
$B(7) =$	877	$B(14) =$	190899322

Партиције код којих је редослед битан

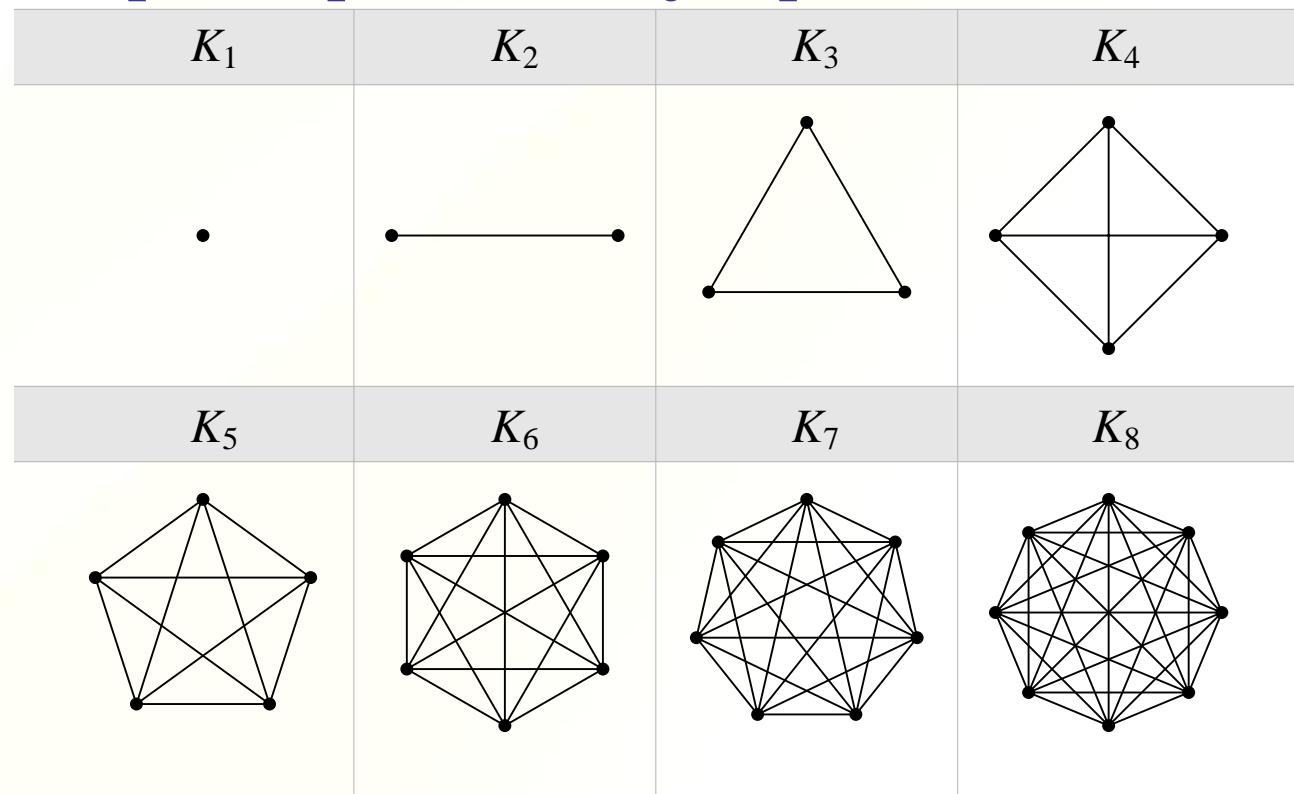
- Композиција природног броја је партиција код које је редослед сабирача битан
- Пример $n = 7$: $3+2+1+1$ или $1+1+2+3$
- Број композиција броја n је 2^{n-1}
(једнозначно пресликање $\{0,1\}^{n-1}$)
$$(1 \overbrace{* 1 * 1 * \dots * 1}^n), * \in \{+, |\}$$
- Партиција скупа са редоследом подскупова:
генерисати партицију скупа +
пермутација свих подскупова

Претраживање по стаблима графа: комплетан граф

- Комплетан неусмерен граф са n чворова (K_n): између сваког паре чворова постоји грана

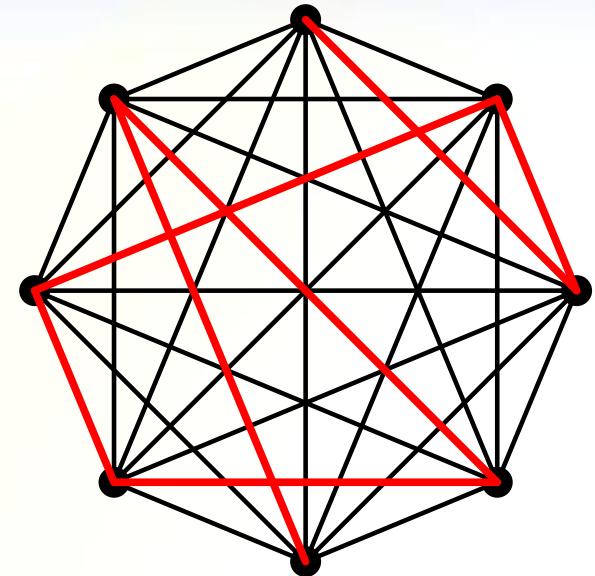
- Укупан број грана је

$$\binom{n}{2} = \frac{n(n-1)}{2}$$



Стабло графа

- Стабло графа (енг. spanning tree) је подскуп од $n - 1$ грана које
 - повезују све чворове графа (постоји пут између произвољног пара чворова) и
 - нема петљи (до једног чвора може се доћи само на један начин)
- Колико има различитих стабала комплетног графа K_n ?

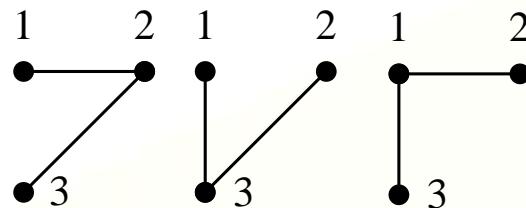


Примери стабла комплетних графова K_2 , K_3 и K_4

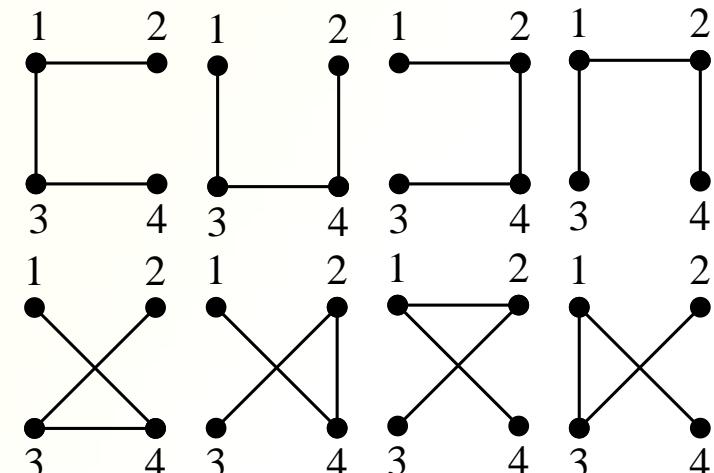
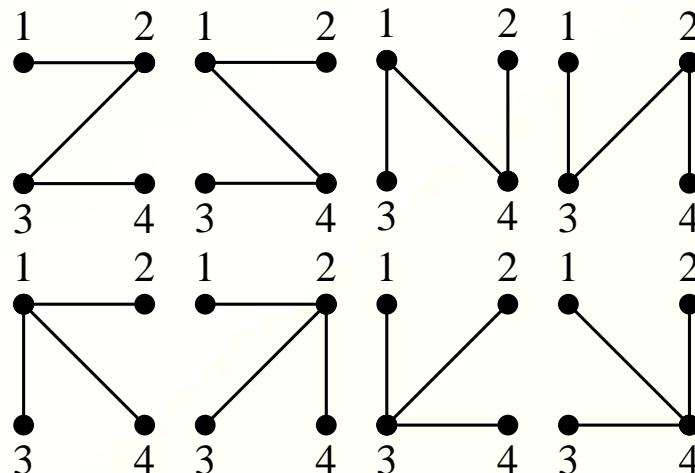
K_2 :



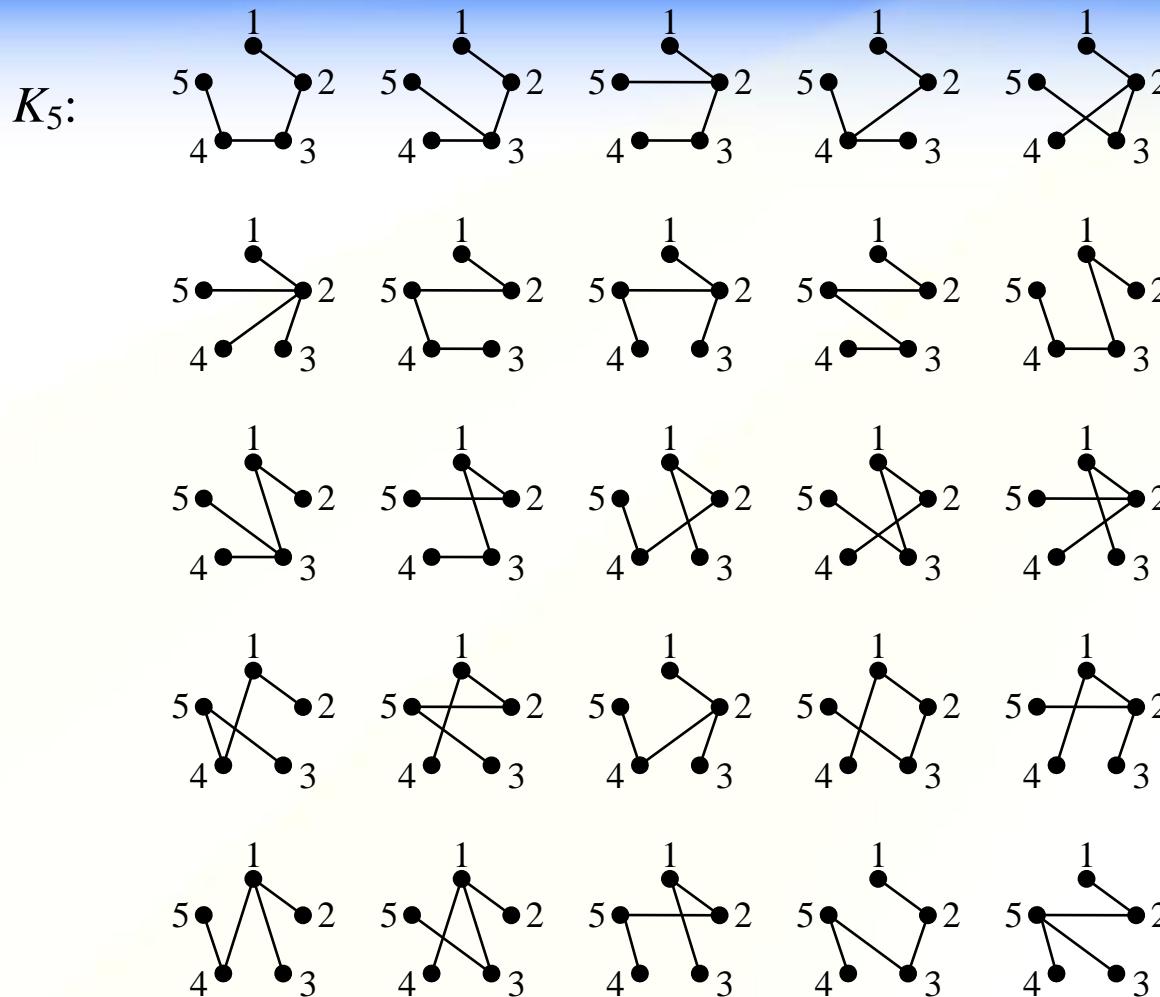
K_3 :



K_4 :



Пример стабла K_5 : 1/5 стабала (цикличка пермутација индекса)



Колики је број стабала графа?

- Број стабала комплетног графа је n^{n-2} (Cayley's formula)
- $n^{n-2} > n!$ ако је $n \geq 5$
- Претраживање по свим стаблима је сложенији проблем од TSP!
- Уколико граф није комплетан број стабала је мањи од n^{n-2}

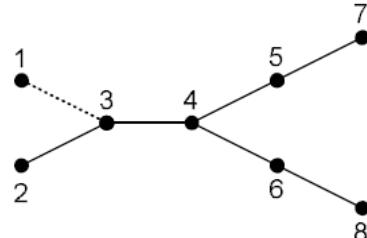
Како генерисати сва стабла комплетног графа?

- Секвенце: варијације са понављањем $n \geq 2$ елемената на $n - 2$ места
 - $n = 4$ (број стабала K_4 је 16):
 $(1\ 1)\ (1\ 2)\ (1\ 3)\ (1\ 4)\ (2\ 1)\ (2\ 2)\ (2\ 3)\ (2\ 4)$
 $(3\ 1)\ (3\ 2)\ (3\ 3)\ (3\ 4)\ (4\ 1)\ (4\ 2)\ (4\ 3)\ (4\ 4)$
- Постоји **једнозначно пресликавање** између оваквих секвенци и стабала комплетног графа са n чворова (Prüfer)

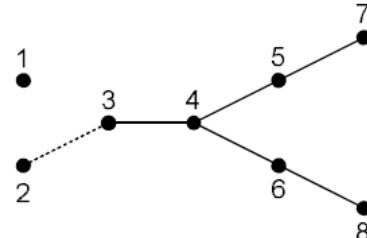
Кодовање стабала у секвенце: алгоритам

- Лист = чвор из кога полази само једна грана
- Пronaћи лист стабла са
најмањим редним бројем
- Записати редни број чвора са којим је
повезан нађени лист (јединствени сусед)
- Избацити лист из стабла
- Поновити процедуру $n - 2$ пута

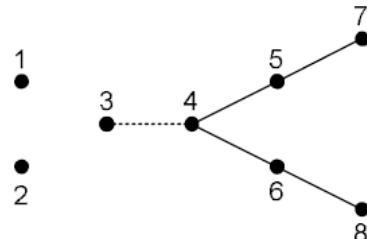
Кодовања стабла у секвенце: пример



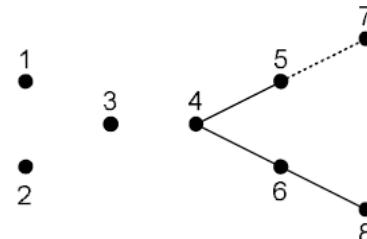
(a) $P = (3)$



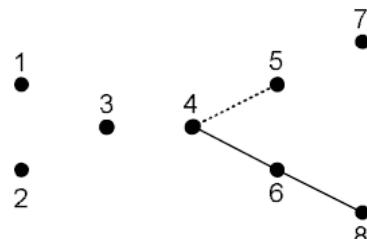
(b) $P = (3, 3)$



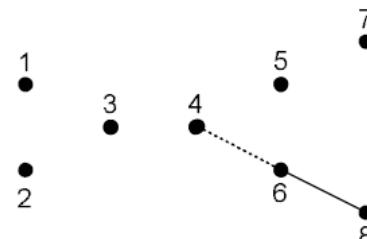
(c) $P = (3, 3, 4)$



(d) $P = (3, 3, 4, 5)$



(e) $P = (3, 3, 4, 5, 4)$

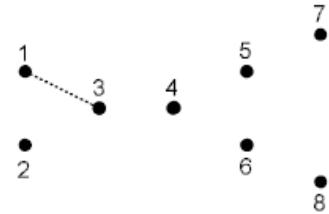


(f) $P = (3, 3, 4, 5, 4, 6)$

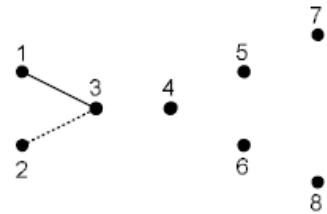
Декодовање секвенце у стабло: алгоритам

- Задата је секвенца $P = (p_1, p_2, \dots, p_{n-2})$
- Почети са низом чворова
 $V = (v_1, v_2, \dots, v_n) = (1, 2, \dots, n)$
- Понављати за $i = 1, 2, \dots, n - 2$
 - Пронаћи најмањи редни број чвора v_i који се не појављује у секвенци P
 - Повезати v_i са p_i
 - Избацити v_i из V и p_i из P
- Последњи корак је повезивање преостала два елемента из V

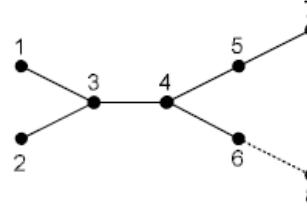
Декодовање секвенце у стабло: пример



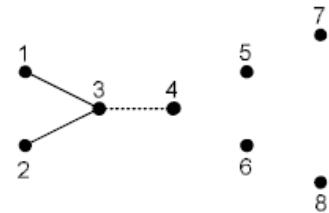
(a) $P = (\underline{3}, 3, 4, 5, 4, 6)$; $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$



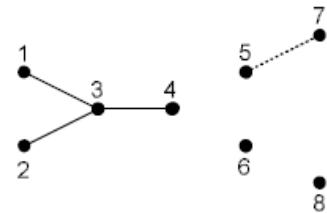
(b) $P = (\underline{3}, 4, 5, 4, 6)$; $V = \underline{\{2, 3, 4, 5, 6, 7, 8\}}$



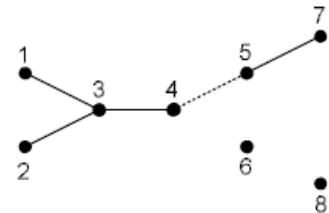
(g) $P = (\)$; $V = \underline{\{6, 8\}}$



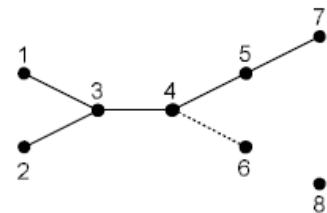
(c) $P = (\underline{4}, 5, 4, 6)$; $V = \underline{\{3, 4, 5, 6, 7, 8\}}$



(d) $P = (\underline{5}, 4, 6)$; $V = \{4, 5, 6, \underline{7}, 8\}$



(e) $P = (\underline{4}, 6)$; $V = \{4, \underline{5}, 6, 8\}$



(f) $P = (\underline{6})$; $V = \{\underline{4}, 6, 8\}$

С/С++ код за декодовање

```
1 #include <stdio.h>
2
3 void SequenceToSpanningTree(int* P, int len, int* T)
4 {
5     int i,j,q=0;
6     int n=len+2;
7     int* V=new int [n];
8
9     for(i=0; i<n; i++)
10        V[i]=0;
11
12    for(i=0; i<len; i++)
13        V[P[i]-1] += 1;
14
15    for(i=0; i<len; i++)
16    {
17        for (j=0; j<n; j++)
18        {
19            if (V[j]==0)
20            {
21                V[j]=-1;
22                T[q++]=j+1;
23                T[q++]=P[i];
24                V[P[i]-1]--;
25                break;
26            }
27        }
28    }
29
30    j=0;
31    for(i=0; i<n; i++)
32    {
33        if(V[i]==0 && j==0)
34        {
35            T[q++]=i+1;
36            j++;
37        }
38        else if(V[i]==0 && j==1)
39        {
40            T[q++]=i+1;
41            break;
42        }
43    }
44
45    delete [] V;
46 }
47
```

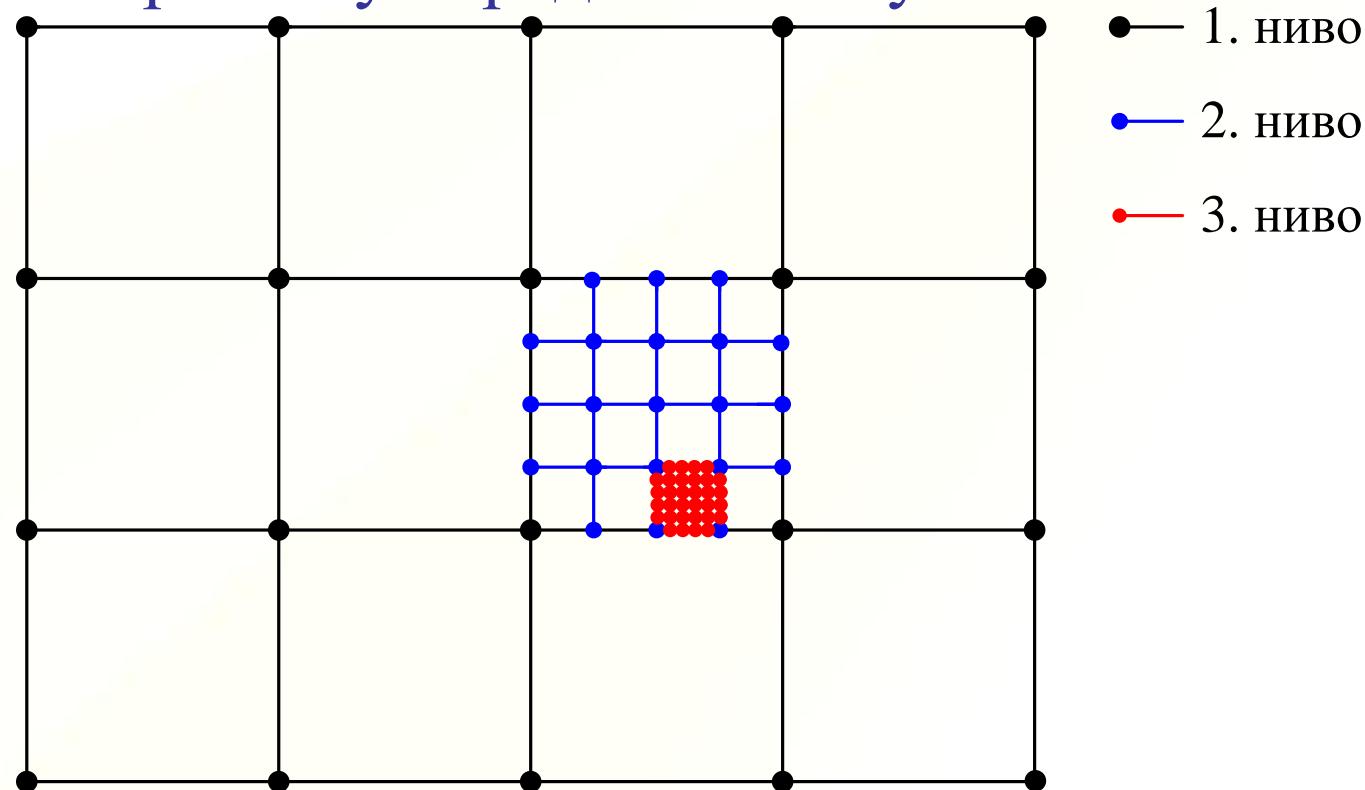
```
48 void main(void)
49 {
50     int P[] = { 1, 2, 2 };
51     int len = sizeof(P) / sizeof(P[0]);
52
53     int* T = new int [2*(len+1)];
54     SequenceToSpanningTree(P, len, T);
55
56     for(int i=0; i<2*(len+1); i++)
57     {
58         printf(" %d",T[i]);
59         if((i+1)%2==0 && i<2*len)
60             printf(" - ");
61     }
62     printf("\n");
63
64     delete [] T;
65
66 }
```

Систематско претраживање: особине и употреба

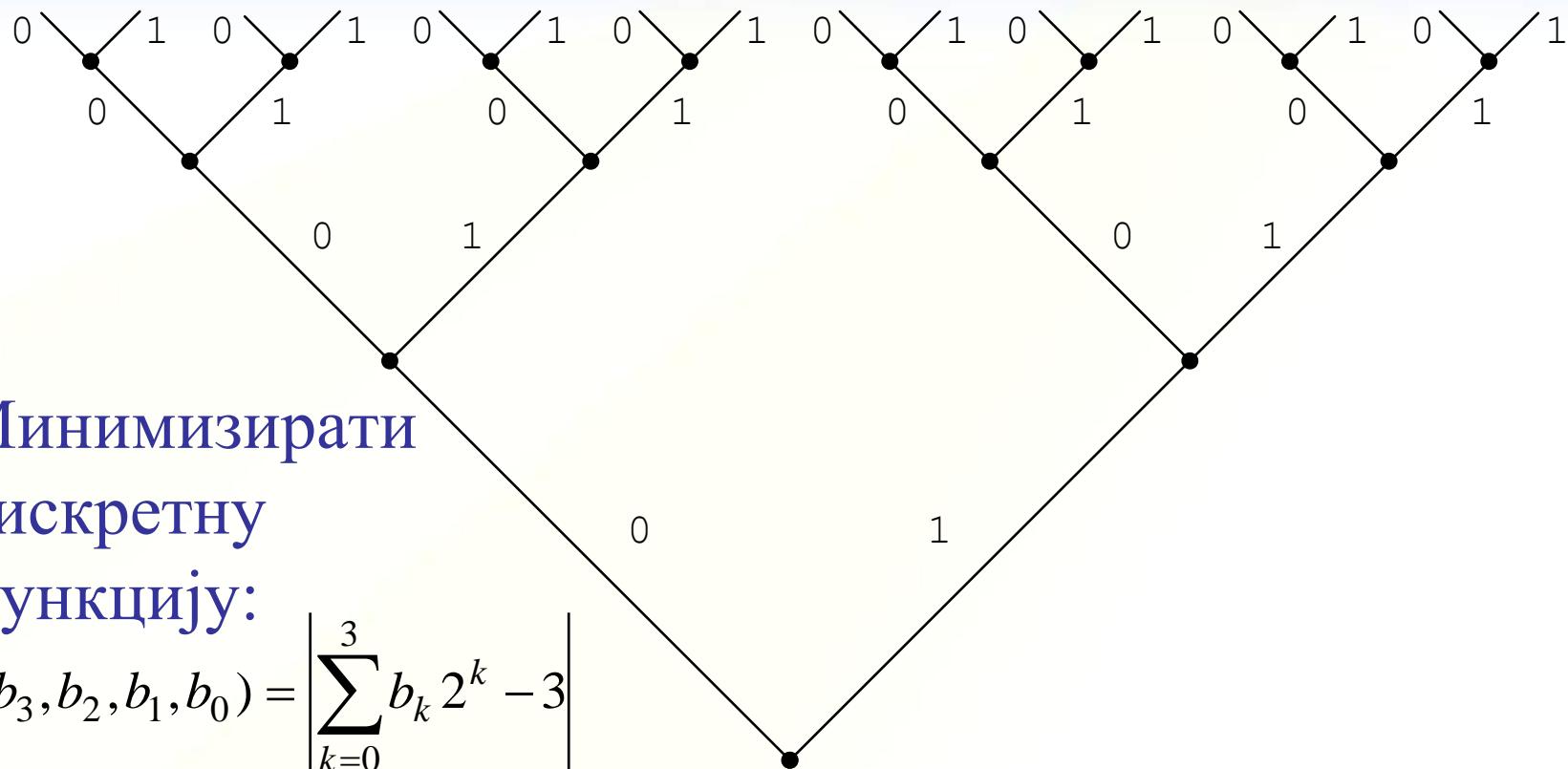
- Уколико желимо да докажемо да смо нашли **глобални оптимум** (најбоље могуће решење) једини начин је да **систематски претражимо** читав оптимизациони простор у општем случају
- Једно решење (једна тачка оптимизационог простора) се једном и само једном проверава
- Континуалне променљиве (NLP):
 - систематско претраживање подразумева коначан корак за сваку димензију (тачност)
 - решење смо нашли са тачношћу која је пропорционална кораку
- Дискретне променљиве (SAT и TSP):
 - сва решења су проверена

Варијације: хијерархијско систематско претраживање

- Проценити део простора и претражити га са мањим кораком у наредном нивоу

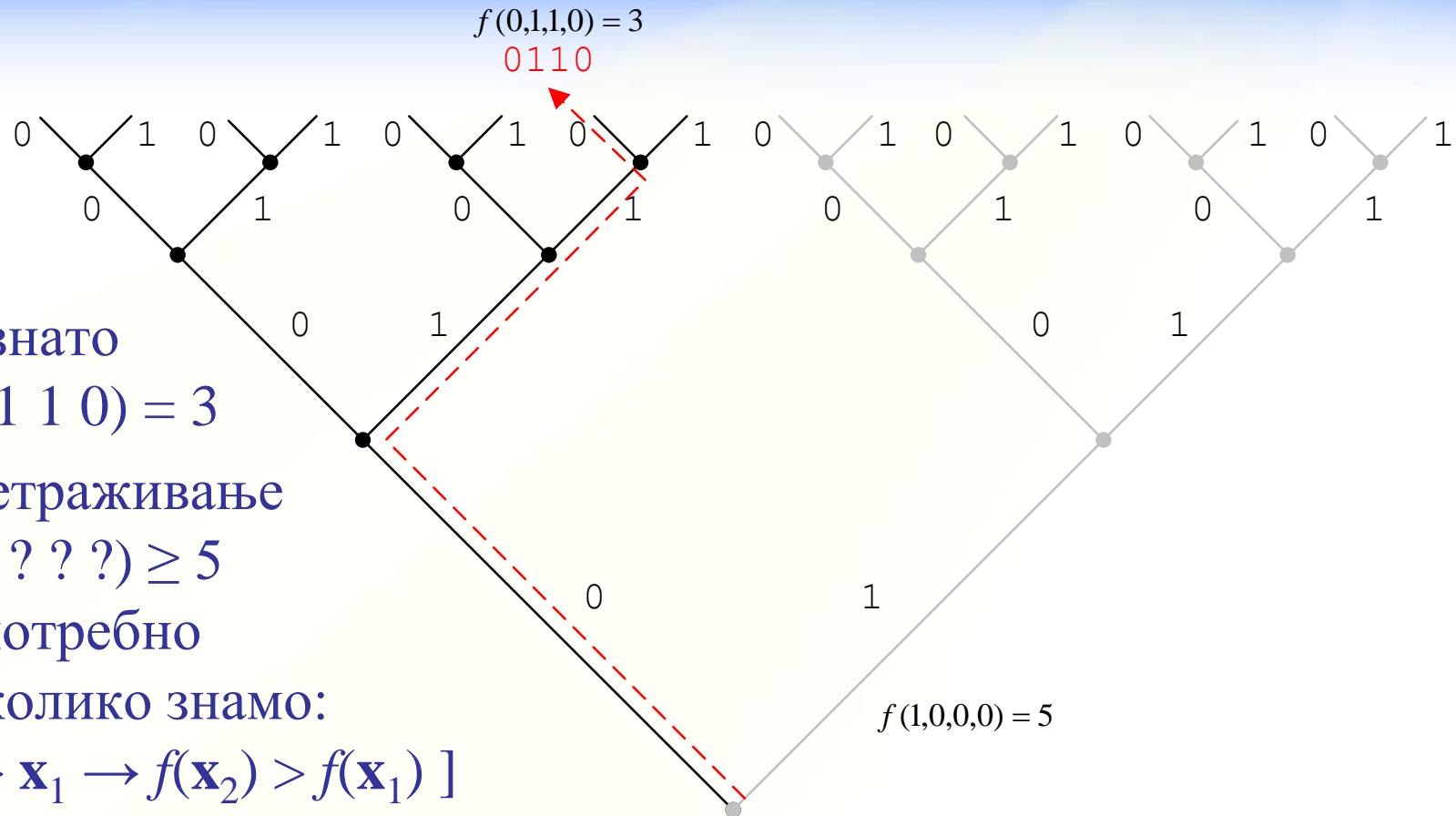


Варијације: гранање и одсецање (branch & cut backtracking)



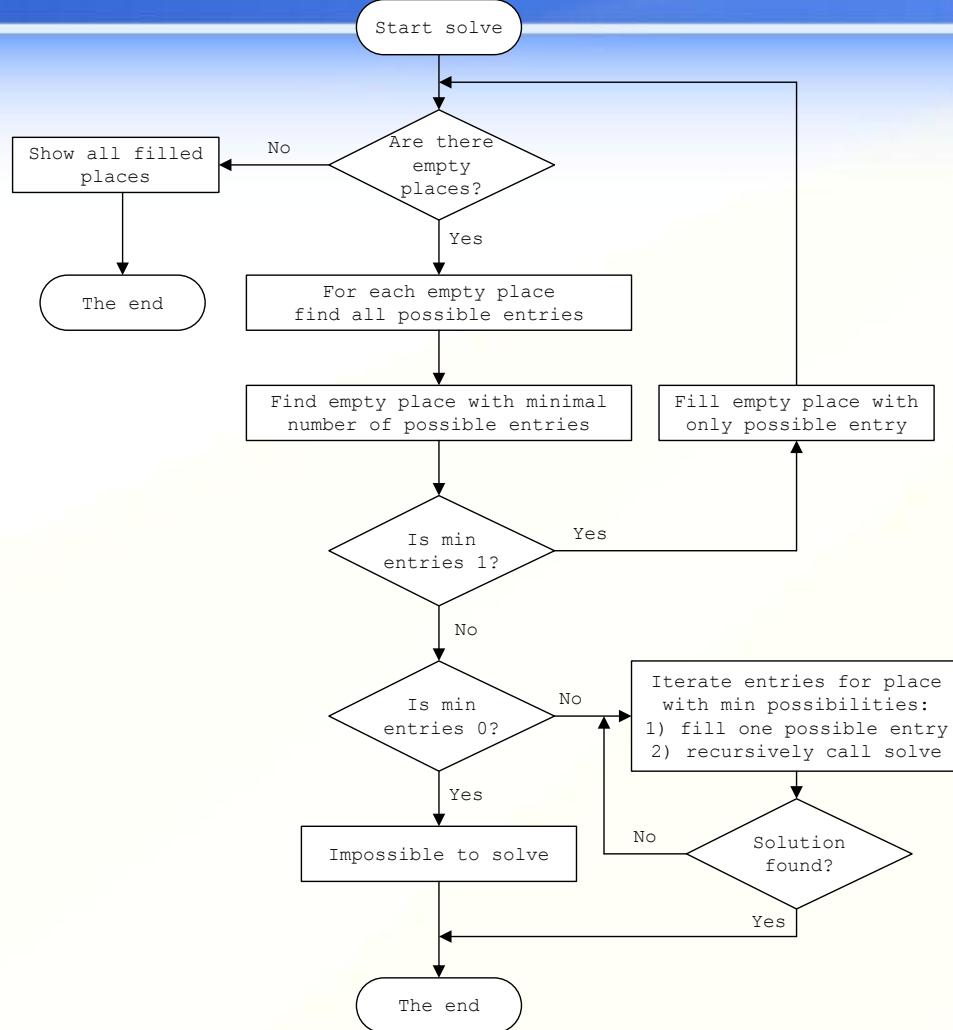
Гранање и одсецање: прескочити део простора

- Познато
 $f(0\ 1\ 1\ 0) = 3$
- Претраживање
 $f(1\ ?\ ?\ ?) \geq 5$
непотребно
[уколико знамо:
 $\mathbf{x}_2 > \mathbf{x}_1 \rightarrow f(\mathbf{x}_2) > f(\mathbf{x}_1)$]
- Простор смањен 2 пута!



SUDOKU:

гранање и одсецање



1	4							
8	9							
2								
2	8	5						
1								
		7						
7		9	3					
6	1	4						
	2							
5	6	9	8	7	2	3	1	4
3	1	8	4	6	9	2	5	7
7	2	4	3	1	5	6	8	9
9	4	3	7	2	1	8	6	5
1	5	7	9	8	6	4	2	3
2	8	6	5	3	4	7	9	1
4	7	1	6	5	8	9	3	2
6	9	2	1	4	3	5	7	8
8	3	5	2	9	7	1	4	6

[33] Solving time: 0.061 [sec]

Други називи и особине систематског претраживања

- Други називи за систематско претраживање
 - Grid search
 - Brute-force search
 - Parameter sweep
 - Exhaustive search
 - Enumeration
 - Generate & test
- Најбољи могући приступ
уколико можемо да сачекамо да се претрага заврши
 - оптимизациони простор је мали у односу на расположиве ресурсе
- Уколико претрага траје недопустиво дugo није од користи
 - оптимизациони простор је велики у односу на рачунарске ресурсе
- Погодан за извршавање у
паралели на рачунарима са више процесора (језгара)

Случајно претраживање

- На случајан начин генеришу се тачке у којима се рачуна функција грешке
- Најчешће се користи генератор са униформном расподелом
- Неефикасан начин оптимизације јер су претходно и наредно израчунавање оптимизационе функције независни (иста тачка може да се испита више пута)
- Добар начин за грубу претрагу простора
- Сложеност $O(N)$
 N број одбирача оптимизационог простора

Генератори случајних бројева

- Рачунарско генерирање случајних бројева је нетривијалан задатак
- Генератори који постоје у библиотекама пролазе строге тестове случајности само делимично!
- Доступне функције:
 - C/C++: `rand()`, `srand()`, `boost/random...`
 - MATLAB: `rand()`, `randn()`, `randi()` ...
 - Python: `random.randint`, `random.uniform...`

Једноставна рутина за генерирање целих случајних бројева

- Опсег $a \leq \xi \leq b$
- Ограничение:
`RAND_MAX`
- Ограничение зависи од компајлера
(VS2017 `RAND_MAX = 32768`)
- Уколико је потребан већи опсег
 - генерисати произволјан низ бита {0, 1} и претворити га у цео број
 - генерисати два цела броја (или више), надовезати бите и интерпретирати нови низ као нови (већи) цео број

```
int random_int(int a, int b)
{
    int res;
    res = a + rand() % (b + 1 - a);
    return res;
}
```

Једноставна рутина за генеришење реалних случајних бројева

- Опсег $a \leq \xi \leq b$
- Разлика између два суседна броја који се генеришу (грануларност) је $1/\text{RAND_MAX}$
- Уколико је потребна већа грануларност:
 - Опсег $[a \ b]$ поделити на више подопсега и сабрати (случајне) бројеве из подопсега

```
double random_float(double a, double b)
{
    double res;
    res = a + (double)rand() / (double)(RAND_MAX / (b-a));
    return res;
}
```

C++11 <random>

```
#include <random>
#include <iostream>

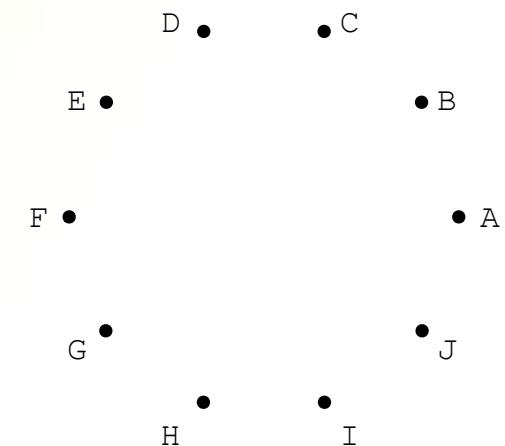
int main() {
    std::random_device rd;
    std::mt19937 mt(rd());

    std::uniform_real_distribution<double> random_float(1.0, 10.0);
    for (int i = 0; i<16; ++i)
        std::cout << random_float(mt) << "\n";

    std::uniform_int_distribution<int> random_int(1, 5);
    for (int i = 0; i<16; ++i)
        std::cout << random_int(mt) << "\n";
}
```

Задатак

- Десет градова потребно је повезати мрежом за дистрибуцију електричне енергије
 - Сви градови морају бити повезани
 - При повезивању није дозвољено да постоје затворени путеви (петље)
 - Цена повезивања сваког пара градова дата је у табели (табела је симетрична)
 - Свако гранање од једног града ка **четири или више** градова повећава цену за $(g - 3) \cdot 100$ јединица цене
 - Пронађи оптималан начин повезивања градова коришћењем **потпуне претраге** (минимизирати цену повезивања)



Цена повезивања (табела је симетрична)

	A	B	C	D	E	F	G	H	I	J
A	0	374	200	223	108	178	252	285	240	356
B	374	0	255	166	433	199	135	95	136	17
C	200	255	0	128	277	821	180	160	131	247
D	223	166	128	0	430	47	52	84	40	155
E	108	433	277	430	0	453	478	344	389	423
F	178	199	821	47	453	0	91	110	64	181
G	252	135	180	52	478	91	0	114	83	117
H	285	95	160	84	344	110	114	0	47	78
I	240	136	131	40	389	64	83	47	0	118
J	356	17	247	155	423	181	117	78	118	0

Поступак решавања

- Проблем се може решити претраживање по комплетном графу K_{10}
- Потребно је претражити сва стабла овог графа
- Максималан број претрага је $n^{n-2} = 10^8$
- Написати код за израчунавање оптимизационе функције: $f(\mathbf{x}) = \sum_{k=1}^9 c_k + \sum_{n=1}^{10} \begin{cases} g(n) < 4, & 0 \\ g(n) \geq 4, & 100(g(n)-3) \end{cases}$
 c_k је “цена” гране са редним бројем k
(тј., вредност из табеле за полазни и крајњи чвор k -те гране)
плус “пенал” за сваки чвор стабла у којем постоји гранање четири или више грана ($g(n)$ је број грана које се стичу у чвиру n)
- Написати код који генерише све секвенце које се могу једнозначно пресликати у стабла графа (варијације са понављањем)
- Искористити рутину за пресликање секвенце у стабло графа
- Написати код који извршава потпуну претрагу
- Пронаћи и записати у ASCII фајл
 - минималну цену мреже за повезивање
 - путању за повезивање (стабло графа)свака грана графа је дефинисана са два чвора X и Y
решење треба да садржи девет грана (стабло) графа

Случајно генерисана варијација са понављањем

- По један случајно изабран елемент из скупа од n елемената постављамо на свако од k места
 - Понављање елемената је дозвољено
- Имплементација: за свако од k места генеришемо један случајан цео број из скупа од n елемената
- Случајан вектор од n реалних бројева из опсега $[x_{\min}, x_{\max}]$
 - На формално идентичан начин генерише се случајно изабран вектор реалних бројева
 - Једина разлика је што се користи генератор случајних реалних бројева из датог опсега

```
38 void driver_random_variation(void)
39 {
40     srand((unsigned int)time(NULL));
41
42     int n=10;
43     int k=4;
44     int s=15;
45
46     int* r = new int [k];
47
48     for(int i=0; i<s; i++)
49     {
50         for(int j=0; j<k; j++)
51         {
52             r[j]=random_int(0,n-1);
53             printf("%2d ",r[j]);
54         }
55         printf("\n");
56     }
57
58     delete [] r;
59 }
60
61 void driver_random_vector(void)
62 {
63     srand((unsigned int)time(NULL));
64
65     double xmin=-1.0;
66     double xmax=+1.0;
67     int k=4;
68     int s=15;
69
70     double* r = new double [k];
71
72     for(int i=0; i<s; i++)
73     {
74         for(int j=0; j<k; j++)
75         {
76             r[j]=random_float(xmin,xmax);
77             printf("% .2f ",r[j]);
78         }
79         printf("\n");
80     }
81
82     delete [] r;
83 }
```

Случајно генерисана комбинација без понављања

- Из скупа од n елемената случајно изабрати k различитих елемената
 - Понављање није дозвољено
- За свако $i = 0, 1, \dots, k - 1$ случајно генерисати цео број $0 \leq r \leq n - 1 - i$
- Изабрати елемент на месту r који већ није изабран
- Записати изабране елементе

```
136 void random_combination(int n, int k, int* P)
137 {
138     if(k>n) return;
139
140     int i,j,r,c;
141     int* Q = new int [n];
142
143     for(i=0; i<n; i++)
144         Q[i]=0;
145
146     for(i=0; i<k; i++)
147     {
148         r = random_int(0,n-1-i);
149         c=0;
150         for(j=0; j<n; j++)
151         {
152             if(Q[j]==0)
153             {
154                 if(r==c)
155                 {
156                     Q[j]++;
157                     break;
158                 }
159                 c++;
160             }
161         }
162     }
163
164     c=0;
165     for(i=0; i<n; i++)
166     {
167         if(Q[i]==1)
168         {
169             P[c]=i+1;
170             c++;
171         }
172     }
173     delete [] Q;
174 }
```

```
176 void driver_random_combination(void)
177 {
178     int n=5;
179     int k=3;
180
181     int* P = new int [k];
182
183     for(int i=0; i<100; i++)
184     {
185         random_combination(n,k,P);
186
187         for(int i=0; i<k; i++)
188             printf("%2d ",P[i]);
189         printf("\n");
190
191     }
192     delete [] P;
193 }
```

Илустрација алгоритма

$Q = (0, 0, 0, 0, 0, 0, 0, 0)$

`random_int(0, 6) = 5` $Q = (0, 0, 0, 0, 0, \textcolor{red}{1}, 0)$

`random_int(0, 5) = 5` $Q = (0, 0, 0, 0, 0, \textcolor{red}{1}, \textcolor{red}{1})$

`random_int(0, 4) = 0` $Q = (\textcolor{red}{1}, 0, 0, 0, 0, \textcolor{red}{1}, \textcolor{red}{1})$

`random_int(0, 3) = 3` $Q = (\textcolor{red}{1}, 0, 0, 0, \textcolor{red}{1}, \textcolor{red}{1}, \textcolor{red}{1})$

$P = (1, 5, 6, 7)$

- Бројеви могу да се директно записују у P
 - сортира се P на крају (ефикасније али захтева сортирање)

Случајно генерисана пермутација

- Полазећи од низа a_0, a_1, \dots, a_{n-1} генерисати случајну пермутацију
- За свако $i = n - 1, n - 2, \dots, 1$
- Случајно генерисати цео број j , $0 \leq j \leq i$
- Заменити вредности $a_j \leftrightarrow a_i$
- n замена по паровима не даје добру статистику!

```
88 void random_permutation(int n, int *p)
89 {
90     int i,j,s;
91     for(i=n-1; i>0; i--)
92     {
93         j = random_int(0, i);
94         if(i!=j)
95         {
96             s = p[i];
97             p[i]=p[j];
98             p[j]=s;
99         }
100    }
101 }
102
103 void driver_random_permutation(void)
104 {
105     int n=3;
106     int* p=new int [n];
107     int* s=new int [n];
108     int T=1000;
109
110     // initialization
111     for(int i=0; i<n; i++)
112     {
113         p[i]=i+1;
114         s[i]=0;
115     }
116
117     for(int i=0; i<T; i++)
118     {
119         random_permutation(n, p);
120
121         // print-out
122         for(int j=0; j<n; j++)
123             printf("%2d ",p[j]);
124             printf("\n");
125
126         s[p[0]-1]++;
127     }
128
129     for(int i=0; i<n; i++)
130         printf("%2.5f\n",s[i]*1.0/T);
131
132     delete [] p;
133     delete [] s;
```

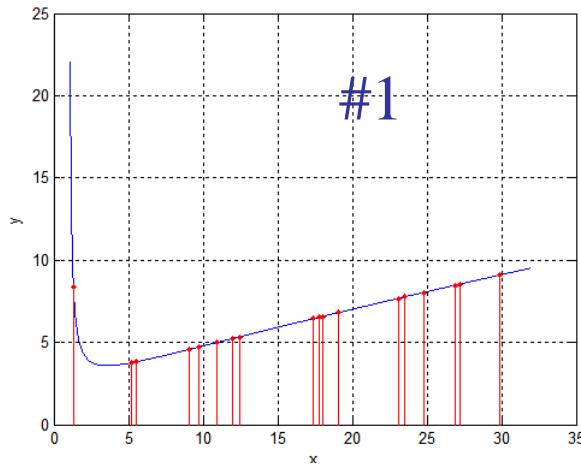
Илустрација алгоритма

		Основна идеја						Имплементација на задатом низу	
		Низ неискоришћених			Нови низ				
		1	2	3	4	5	6	7	
random_int(0, 6)=6		1	2	3	4	5	6	7	1 2 3 4 5 6 7
random_int(0, 5)=2		1	2	3	4	5	6	7	1 2 6 4 5 3 7
random_int(0, 4)=4		1	2	4	5	6		6 3 7	1 2 6 4 5 3 7
random_int(0, 3)=1		1	2	4	5			2 6 3 7	1 4 6 2 5 3 7
random_int(0, 2)=0		1	4	5				1 2 6 3 7	6 4 1 2 5 3 7
random_int(0, 1)=1		4	5					5 1 2 6 3 7	6 4 1 2 5 3 7
random_int(0, 0)=0		4						4 5 1 2 6 3 7	6 4 1 2 5 3 7

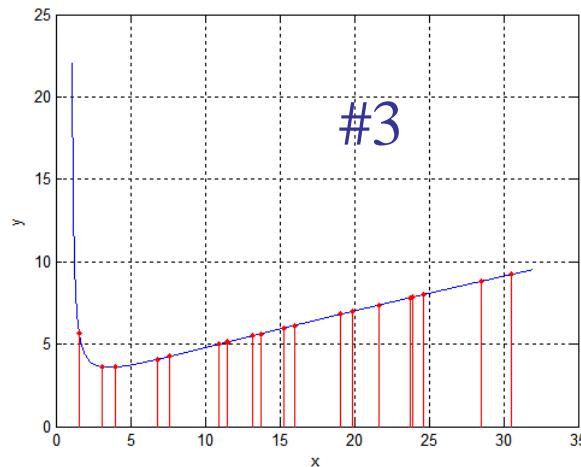
Случајно генерисане партиције и стабла графа

- Случајно генерисана партиција броја n :
 - Пронаћи укупан број партиција броја, генерисати случајан број од 1 до n и генерисати одговарајућу партицију алгоритмом за генерисање свих партиција
 - Једноставан али рачунарски захтеван приступ
- Случајно генерисана партиција скупа S
 - Генерисати случајан RGS (restricted growth string), сваки елемент је између нуле и највећи пре њега +1
- Случајно генерисано стабло потпуног графа K_n
 - Генерисати низ од $n - 2$ елемента на чијим местима може да буде $1, 2, \dots, n$ (видети пресликање таквих низова у стабла графа)
 - Ако граф није потпун, недозвољена стабла се изостављају
 - Постоје рачунарски ефикасније методе

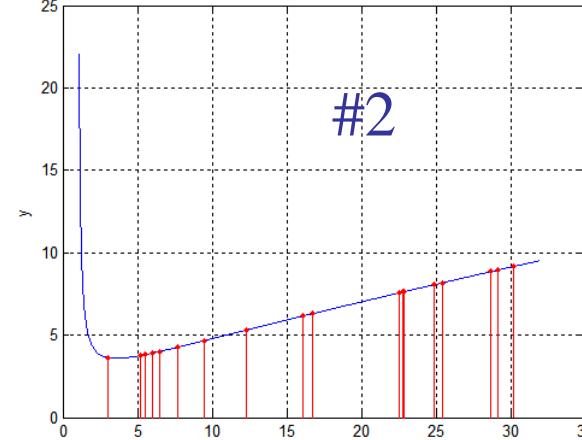
Пример: различита решења при сваком покретању



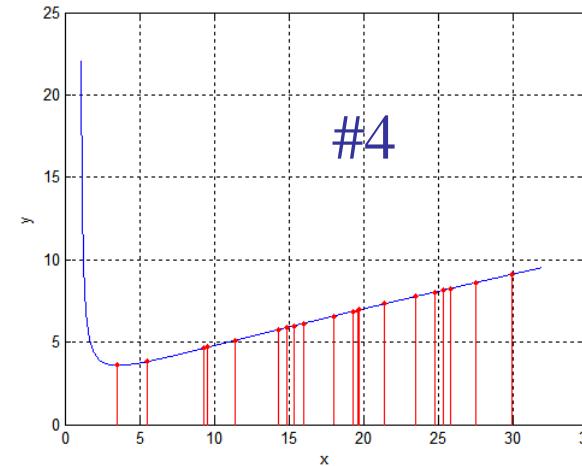
#1



#3

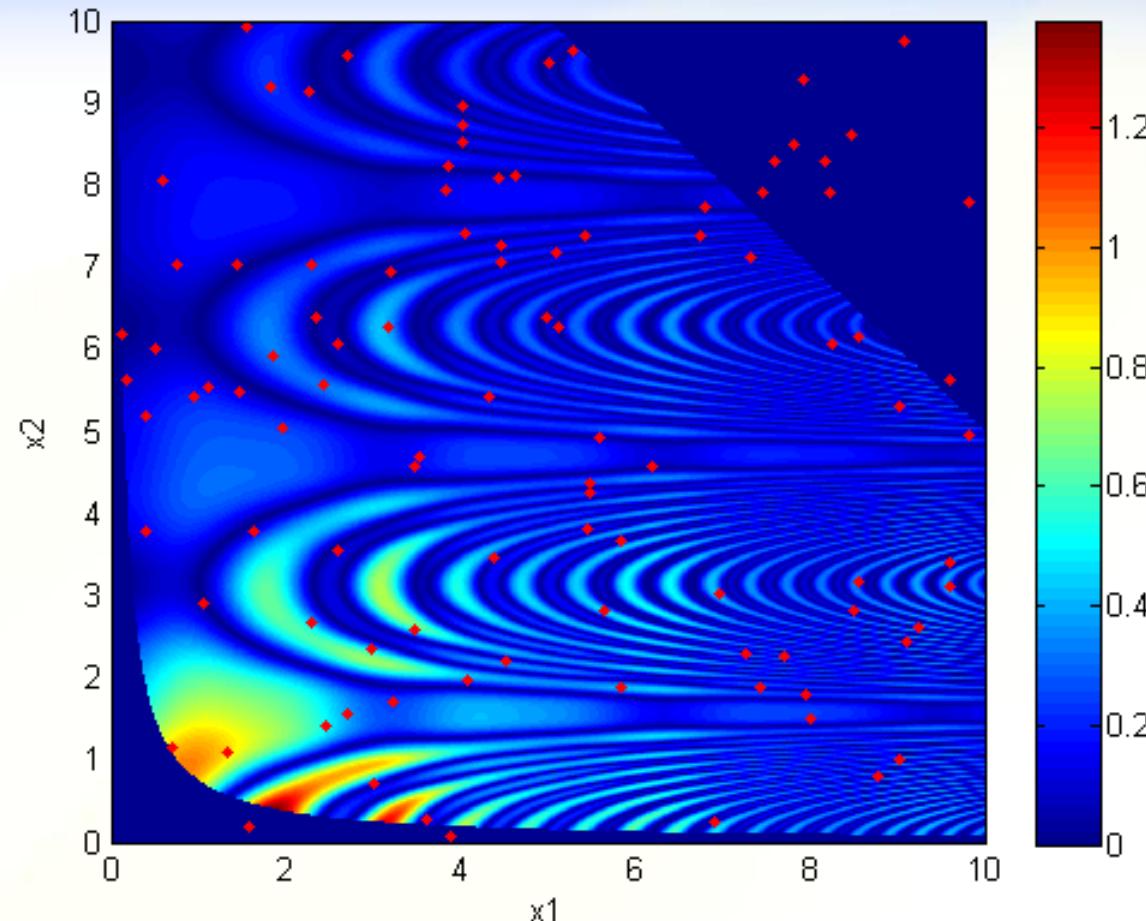


#2



#4

Пример случајног претраживања једне 2D функције (NLP)



О случајним процесима...

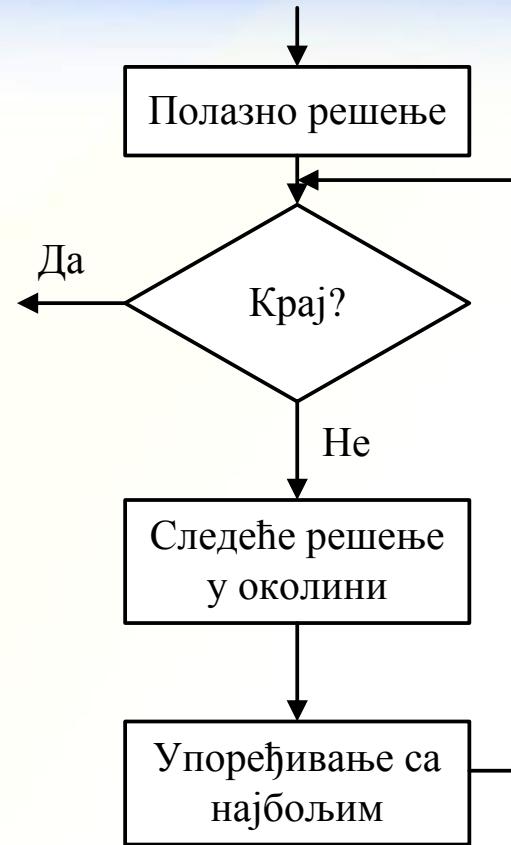
(by Piet Hein)

Whenever you're **called on to make up your mind**
and you're hampered by **not having any**
the best way to solve the dilemma you'll find
is simply by **spinning a penny**.

No - **not so that chance shall decide** the affair
while you're passively standing there moping;
but the moment the penny is up in the air
you suddenly **know what you're hoping**.

Локални оптимизациони алгоритми

- Енг: hill-climbing, down-hill, greedy, local optimization...
- Претражују околину полазног решења
- Заснивају се на итеративном поправљању полазног решења
- Генерализовани блок дијаграм је приказан десно



Слабости Hill-Climbing Алгоритама

- По правилу проналазе само локални оптимум
- Нема информације о томе колико смо далеко или близу глобалног оптимума
- Крајње решење зависи од полазног
- Генерално, није могуће предвидети максималан број потребних итерација (зависи од оптимизационе функције)

SAT локално претраживање

- Дефинисати околину која се претражује
- Хамингово растојање
(број бита, k , колико сме да се промени)
- Ако је број димензија D генерисати
све могуће промене
 - број избора k бита од D
 - пута број распореда бита на k места
- Памтити решења и не проверавати
више пута исто решење

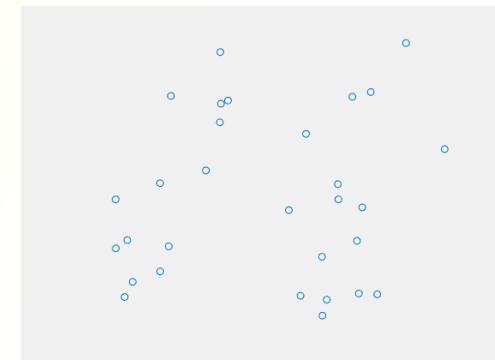
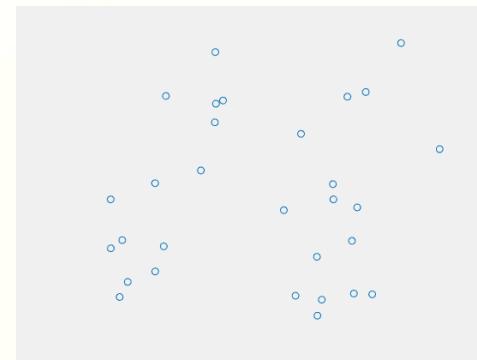
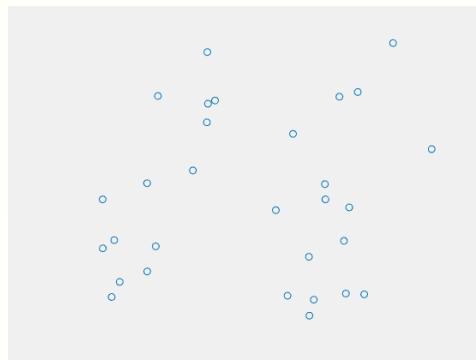
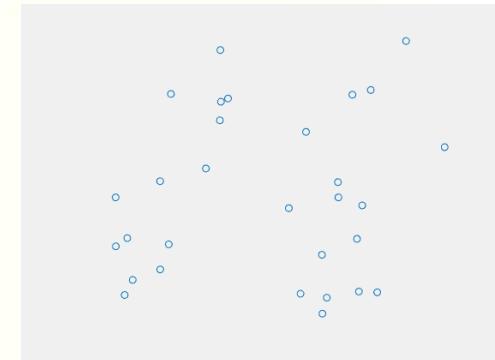
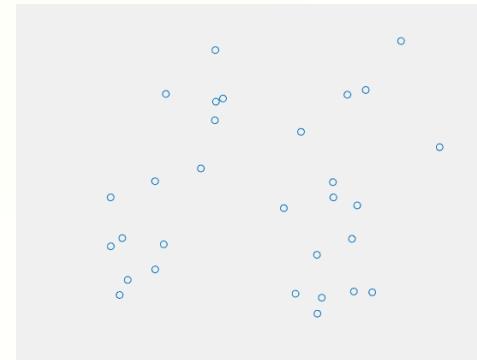
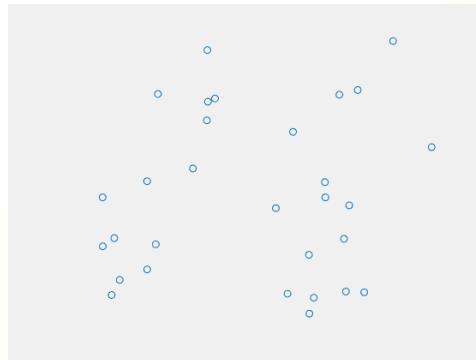
Претрага окolini $k=3$ бита	Све могуће вредности за промену $k=3$ бита
1 0 1 1 0	0 0 0
1 0 1 1 0	0 0 1
1 0 1 1 0	0 1 0
1 0 1 1 0	0 1 1
1 0 1 1 0	1 0 0
1 0 1 1 0	1 0 1
1 0 1 1 0	1 1 0
1 0 1 1 0	1 1 1
1 0 1 1 0	
1 0 1 1 0	

Графови: локално претраживање (minimum spanning tree)

- Prim's algorithm (Jarnik's algorithm, Prim-Dijkstra algorithm, DJP algorithm)
 - локално претраживање по графовима
 - проналажење MST
- Кораци алгоритма:
 - изабрати један чврт графа (произвољно) и уврстити га у стабло
 - пронаћи грану са најмањом тежином између било ког чвора стабла и чворова који нису у стаблу и уврстити одговарајући чврт и грану у стабло
 - поновити претходни корак док сви чворови нису у стаблу
- Ово је истовремено и алгоритам за проналажење MST
- Променом тежина грана графа могуће је генерисати различита стабла графа овим алгоритмом (ако се тежине грана додеље на случајан начин, добија се случајно генерисано стабло графа!)

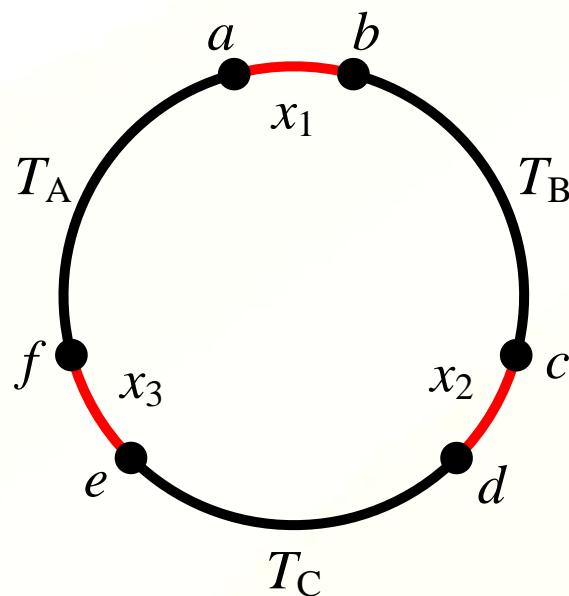
Илустрација проналажења стабла са минималним збиром тежина

- Полазак из различитог чвора даје исти резултат (исто стабло)
- У коришћеном примеру тежина гране је растојање између чворова
- Временска сложеност зависи од структуре података за чување графа



TSP Локално претраживање З-опт: основна идеја

- Проблем TSP класе, полазно решење (пермутација) T_0
- Изаберемо три везе (x_1, x_2, x_3) које прекинемо $(a,b), (c,d), (e,f)$
- Три преостала подниза T_A, T_B, T_C повезујемо на све могуће начине



$$T_0 = T_A + x_1 + T_B + x_2 + T_C + x_3$$

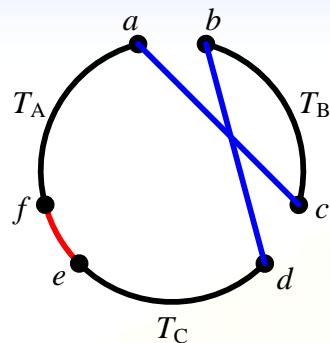
$$L(T) = f(T)$$

$$T_q = (n_1, n_2, \dots, n_k) \rightarrow R_q = (n_k, n_{k-1}, \dots, n_1)$$
$$q \in \{A, B, C\}$$

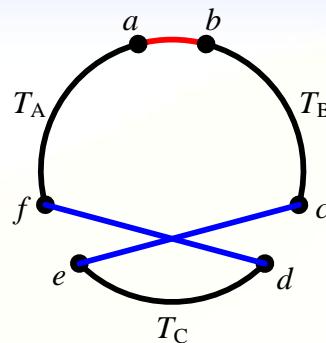
$$f(T) < f(T_0)$$

3-ОПТ: СВИ МОГУЋИ начини повезивања

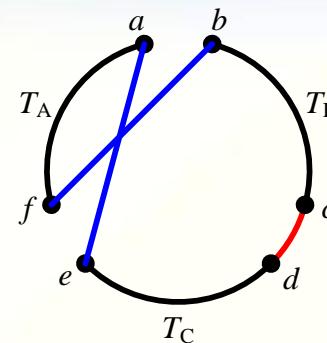
#1: $T_A R_B T_C$



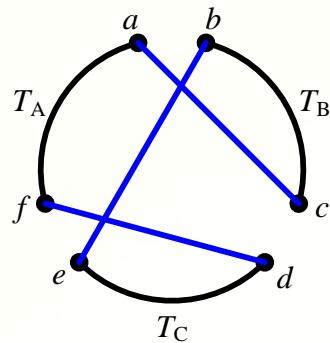
#2: $T_A T_B R_C$



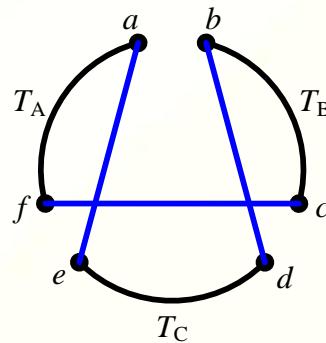
#3: $R_A T_B T_C$



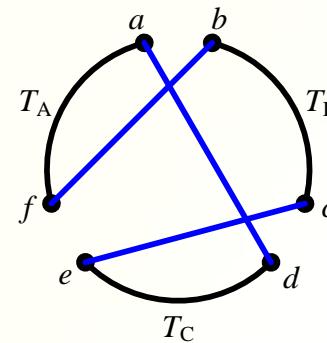
#4: $T_A R_B R_C$



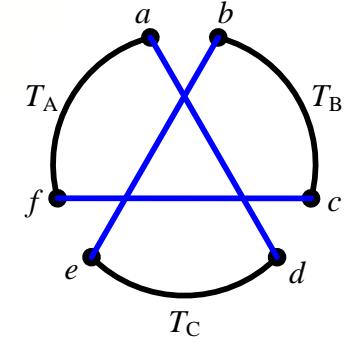
#5: $T_A R_C R_B$



#6: $T_A T_C R_B$

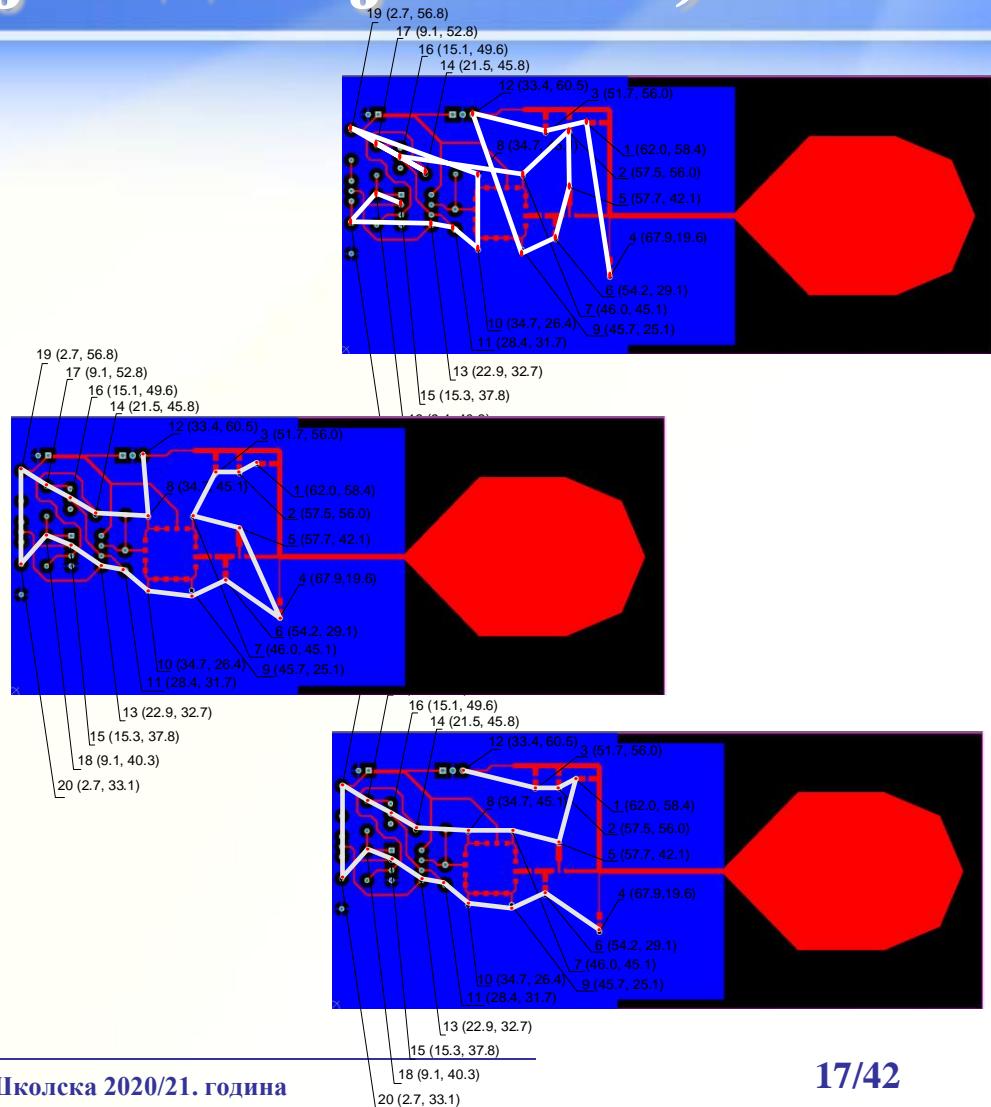


#7: $T_A T_C T_B$



3-опт: пример бушења 20 рупа (2,5 % лошије од најбољег)

- 1 000 000 случајних покушаја
 $f(\mathbf{x}) = 335,91 \text{ mm}$
- 10 000 случајних и 3-опт за најбољу
 $f(\mathbf{x}) = 210,58 \text{ mm}$
 - Свако покретање даје другачије решење!
- Најбоље решење
 $f(\mathbf{x}) = 205,136 \text{ mm}$



3-опт: примена

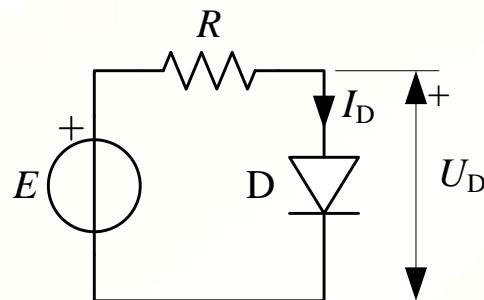
- 3-опт је специјални случај k -опт
 - $k = D$ (број градова) је потпуна претрага
- 3-опт укључује 2-опт (важи генерално)
- Које три везе пресећи?
 - највише $\binom{D}{3} \sim O(D^3)$ ако је D велико постаје непрактично
 - издвојити неке, али које?
- Генерализација
Lin-Kerningan-Helsgaun хеуристика

NLP: континуалне оптимизационе функције

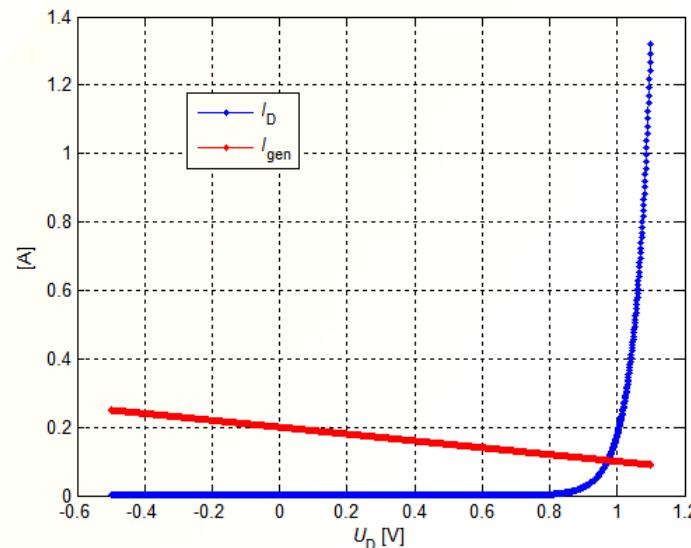
- Проблем:

Карактеристика диоде дата је изразом $I_D = I_0(e^{U_D/U_0} - 1)$, где је $U_0 = 50 \text{ mV}$ и $I_0 = 1 \text{ nA}$. Електромоторна сила генератора је $E = 2 \text{ V}$, а отпорност $R = 10 \Omega$.

- Израчунати радну тачку диоде



$$\frac{E - U_D}{R} = I_0 \left(e^{\frac{U_D}{U_0}} - 1 \right)$$



Оптимизација и решавање трансцендентних једначина

- Континуална функција са једном променљивом

- Трансцендентна једначина $\frac{E - U_D}{R} - I_0 \left(e^{\frac{U_D}{U_0}} - 1 \right) = 0$

- Знамо да постоји тачно једно решење

- Оптимизациони проблем, L_1 - норма

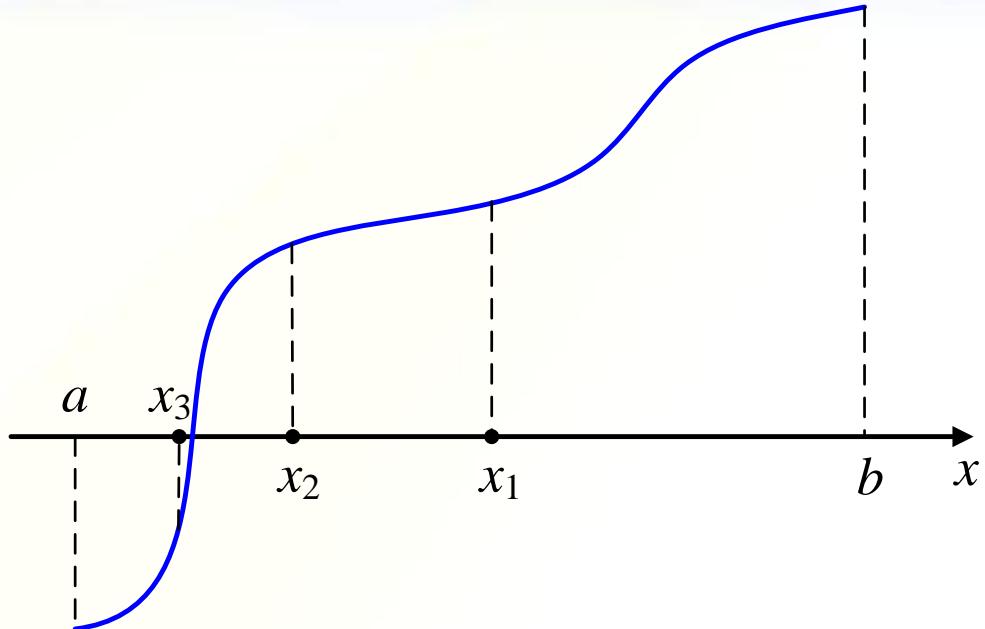
$$f_{\text{opt}}(U_D) = \left| \frac{E - U_D}{R} - I_0 \left(e^{\frac{U_D}{U_0}} - 1 \right) \right| \quad \min(f_{\text{opt}}(U_D)), U_D \in \Re$$

Класични методи (NLP: континуалне функције)

- Метод половљења интервала (енглески: bisection/bracketing method)
- Метод сечице (regula-falsi)
- Њутнов метод (апроксимација тангентама)
- Брентов метод (апроксимација параболама)
- Лагранжови мултипликатори
- ККТ услови
- Градијентни метод
- Хесијан
 - BFGS (Broyden–Fletcher–Goldfarb–Shanno) метод

Половљење интервала

- Услов:
постоји интервал
 $f(a)f(b) < 0$
- $x_1 = 0,5 (a + b)$
- Нови интервал:
 $[a, x_1]$ ако $f(a)f(x_1) < 0$
 $[x_1, b]$ ако $f(x_1)f(b) < 0$
- Поновити процедуру
док се нула не одреди са задатом тачношћу



Половљење интервала брзо конвергира

- У кораку n интервал је $\frac{b-a}{2^n}$
- Свака итерација смањује интервал 2 пута
- Изузетно робустан
- Једноставан за програмирање
- Не захтева познавање извода функције!
- Може да се примени и ако је функција са шумом или са целим бројевима
- Захтева познавање почетног интервала
- Генерализација на вишедимензионе проблеме?

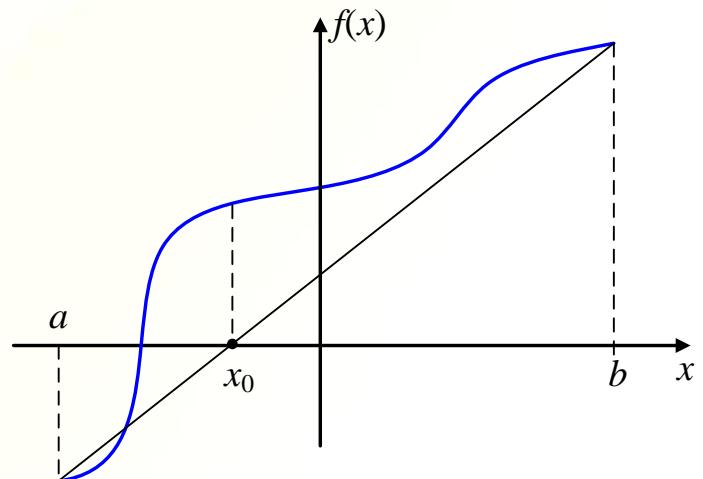
Regula falsi: основна идеја и скица извођења

- Нула се тражи у пресеку линеарне апроксимације функције и апсцисе

$$f(x) = \frac{f(b) - f(a)}{b - a}(x - a) + f(a)$$

$$0 = \frac{f(b) - f(a)}{b - a}(x_0 - a) + f(a)$$

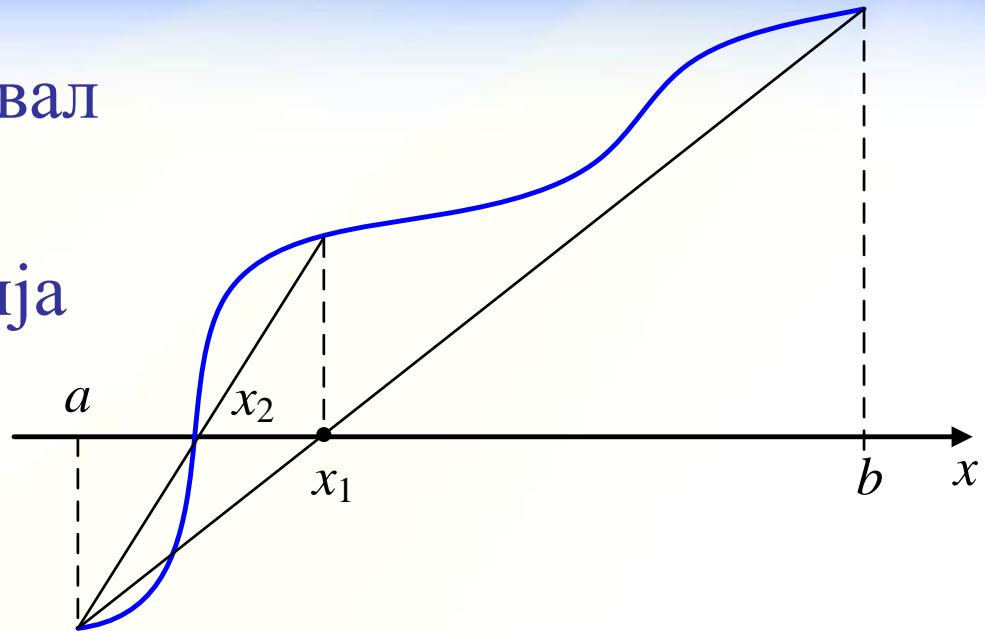
$$x_0 = \frac{af(b) - bf(a)}{f(b) - f(a)}$$



Regula Falsi (метод сечице)

- Услов: постоји интервал $f(a)f(b) < 0$
- Следећа апроксимација

$$x_1 = \frac{af(b) - bf(a)}{f(b) - f(a)}$$



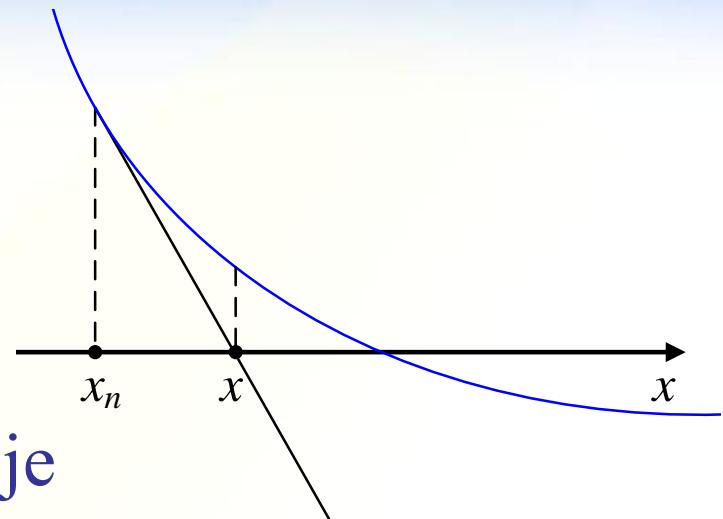
- Нови интервал:
 $[a, x_1]$ ако $f(a)f(x_1) < 0$ или $[x_1, b]$ ако $f(x_1)f(b) < 0$
- Поновити док се нула не одреди са задатом тачношћу

Regula falsi: употреба

- Потребно је знати почетни интервал у којем постоји једна нула, а функција мења знак
- Конвергенција је нешто бржа од метода половине интервала
- Мало сложенији за програмирање од половине интервала
(постоји формула која се програмира)
- Генерализација на вишедимензионе проблеме?

Њутнов метод: основна идеја и скица извођења

- Такође је познат под именом Newton-Raphson
- Позната функција $f(x)$ и њен први извод $f'(x)$
- Једначина тангенте у тачки x_n је



$$y(x) = f'(x_n)(x - x_n) + f(x_n)$$

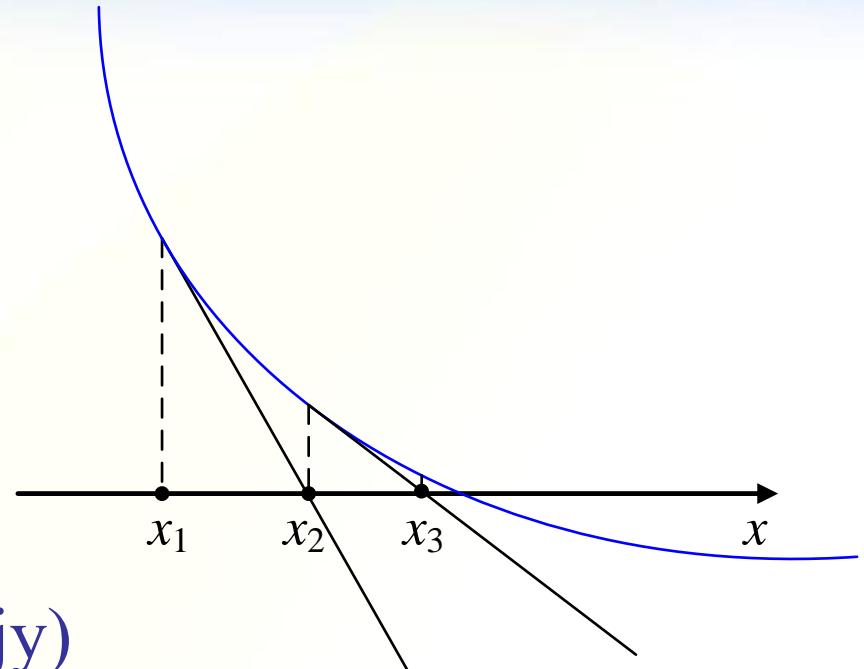
$$0 = f'(x_n)(x - x_n) + f(x_n) \quad \Rightarrow x = x_n - \frac{f(x_n)}{f'(x_n)}$$

Њутнов метод (апроксимација тангентом)

- Следећа апроксимација:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

- Захтева добро полазно решење (тангента “води” ка нули, није тако у општем случају)



- Потребно је израчунати извод функције, што није увек једноставно

Њутнов метод: употреба

- Изузетно брза конвергенција
- Тачно или апроксимативно израчунавање извода ограничава употребу
- Уколико функција садржи локални минимум, алгоритам може да дивергира
- Функције са нумеричким шумом нису погодне за овај метод због израчунавања извода
- Генерализација на вишедимензионе проблеме?

Њутнов метод: генерализација

- Полазимо од система

D нелинеарних једначина, са D непознатих

$$\left. \begin{array}{l} f_1(x_1, x_2, \dots, x_D) = f_1(\mathbf{x}) = 0 \\ f_2(x_1, x_2, \dots, x_D) = f_2(\mathbf{x}) = 0 \\ \vdots \\ f_D(x_1, x_2, \dots, x_D) = f_D(\mathbf{x}) = 0 \end{array} \right\} \rightarrow \mathbf{f}(\mathbf{x}) = 0$$

- Развијемо једну једначину у Тејлоров ред

$$f_p(\mathbf{x} + \Delta\mathbf{x}) = f_p(\mathbf{x}) + \sum_{k=1}^D \frac{\partial f_p}{\partial x_k} \Delta x_k + O(\Delta\mathbf{x}^2) \quad p = 1, 2, \dots, D$$

Њутнов метод: генерални итеративни поступак

- Занемарујући чланове $\Delta\mathbf{x}^2$, пресек тангенте по једној координати је

$$\sum_{k=1}^D \frac{\partial f_p}{\partial x_k} \Delta x_k = -f_p(\mathbf{x}) \Rightarrow \Delta x_k = -\left(\sum_{k=1}^D \frac{\partial f_p}{\partial x_k} \right)^{-1} f_p(\mathbf{x})$$

- По свим координатама

$$\Delta\mathbf{x} = -\mathbf{J}_f^{-1}\mathbf{f}(\mathbf{x}) \Rightarrow \mathbf{x} + \Delta\mathbf{x} = \mathbf{x} - \mathbf{J}_f^{-1}\mathbf{f}(\mathbf{x})$$

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_D} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_D}{\partial x_1} & \frac{\partial f_D}{\partial x_2} & \dots & \frac{\partial f_D}{\partial x_D} \end{bmatrix}$$

- Итеративно

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{J}_f^{-1}\mathbf{f}(\mathbf{x}_i)$$

Брентов метод (квадратна интерполяција)

- Услов: функција има само један минимум (или максимум) у посматраном интервалу
- Квадратна апроксимација на основу 3 тачке
- Конструисање параболе
- Рачунање минимума параболе
- Понављање метода
док се не постигне потребна тачност
- Брентов метод обједињује
 - половљење интервала,
 - метод сечице и
 - квадратну интерполяцију

Квадратна апроксимација

- Три (различита) одбирка функције $f_k = f(x_k)$

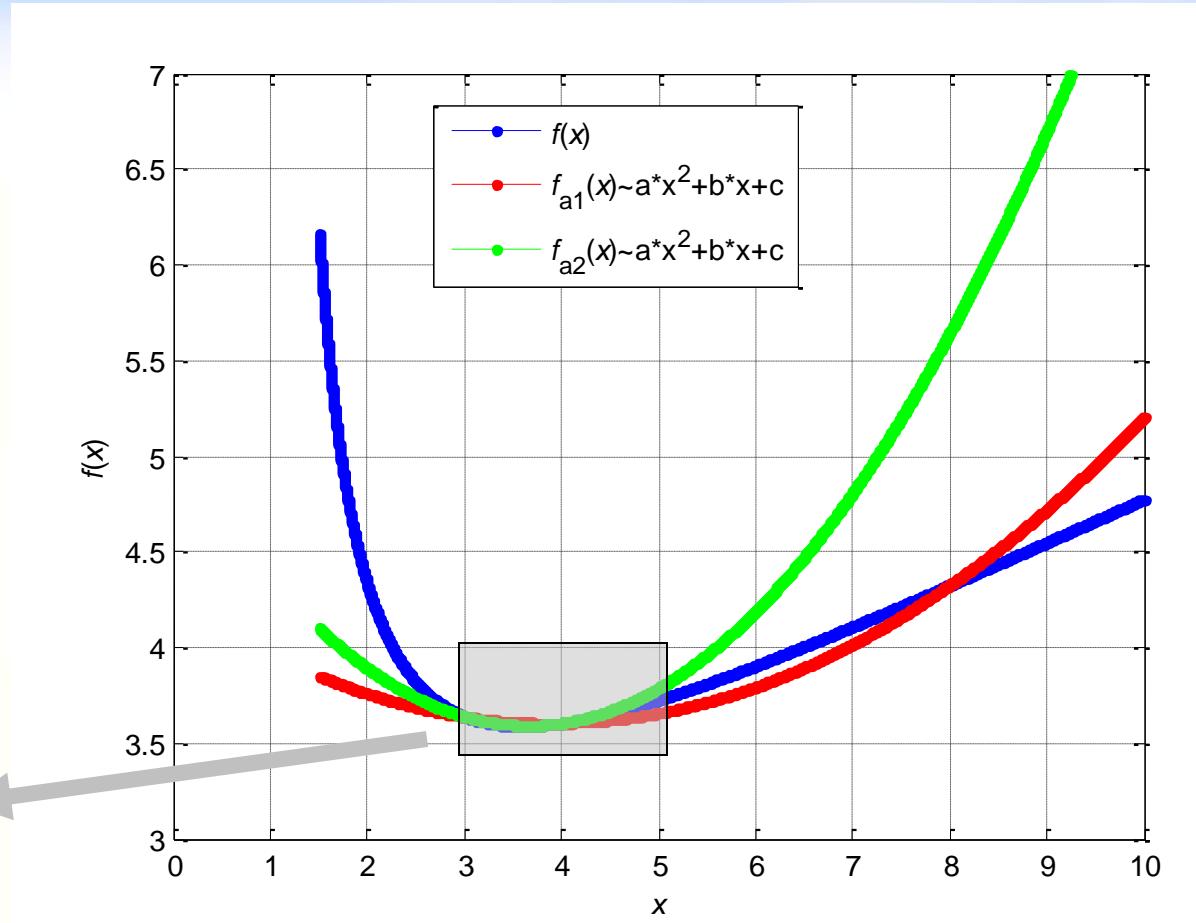
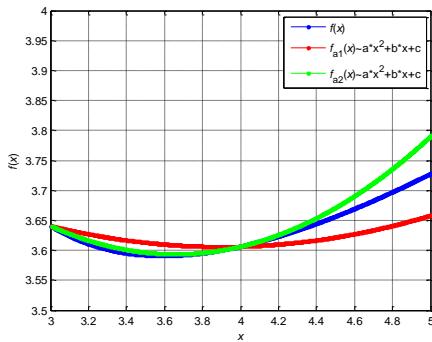
$$\begin{aligned}f_1 &= ax_1^2 + bx_1 + c \\f_2 &= ax_2^2 + bx_2 + c \\f_3 &= ax_3^2 + bx_3 + c\end{aligned}\Rightarrow \begin{bmatrix}x_1^2 & x_1 & 1 \\x_2^2 & x_2 & 1 \\x_3^2 & x_3 & 1\end{bmatrix} \begin{bmatrix}a \\b \\c\end{bmatrix} = \begin{bmatrix}f_1 \\f_2 \\f_3\end{bmatrix}$$

- Коефицијенти $\begin{bmatrix}a \\b \\c\end{bmatrix} = \begin{bmatrix}x_1^2 & x_1 & 1 \\x_2^2 & x_2 & 1 \\x_3^2 & x_3 & 1\end{bmatrix}^{-1} \begin{bmatrix}f_1 \\f_2 \\f_3\end{bmatrix}$

- Квадратна апроксимација $f \approx ax^2 + bx + c$
- Пронађемо минимум $x_{\min} = -\frac{b}{2a}$

Пример квадратне апроксимације

- Функција f
- Прва аппрокс. f_{a1}
- Друга аппрокс. f_{a2}



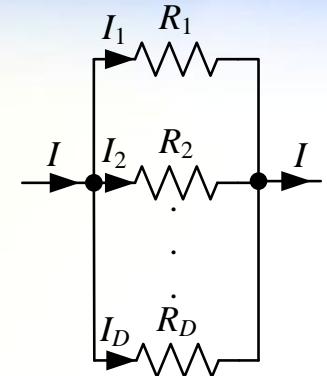
Лагранжови мултипликатори

- Решавамо проблем $\text{minimize } f(\mathbf{x}), \mathbf{x} = (x_1, x_2, \dots x_D)$
$$g_k(\mathbf{x}) = 0, k = 1, 2, \dots m \quad (m < D)$$
- $f(\mathbf{x})$ и $g_k(\mathbf{x})$ су непрекидна и имају све парцијалне изводе
- Формирали помоћну (нову опт.) функцију $F(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{k=1}^m \lambda_k g_k(\mathbf{x})$
- Коефицијенти λ_k су Лагранжови мултипликатори
- Опт. проблем са условима сведен на проблем без услова
$$\frac{\partial F}{\partial x_1} = 0, \frac{\partial F}{\partial x_2} = 0, \dots, \frac{\partial F}{\partial x_D} = 0; \quad g_1 = 0, g_2 = 0, \dots, g_m = 0$$

компактан запис:
$$\nabla F = 0 \Leftrightarrow \frac{\partial F}{\partial x_1} \mathbf{i}_{x_1} + \frac{\partial F}{\partial x_2} \mathbf{i}_{x_2} + \dots + \frac{\partial F}{\partial x_D} \mathbf{i}_{xD} = 0$$
- Тачке које задовољавају овај систем једначина су екстремуми (минимуми, максимуми или седласте тачке)
- Проверити вредности свих добијених екстремума

Пример примене Лагранжових мултипликатора

- У једном чвору електричног кола струја I дели се у D паралелно везаних грана
- У свакој грани је један отпорник познате отпорности R_1, R_2, \dots, R_D
- Пронађи струје грана I_1, I_2, \dots, I_D . тако да је електрична енергија (и снага) овог кола минимална
- Формални запис опт. проблема $\begin{aligned} & \text{minimize } f(\mathbf{x}) = R_1 I_1^2 + R_2 I_2^2 + \dots + R_D I_D^2 \\ & \mathbf{x} = (I_1, I_2, \dots, I_D) \\ & g_1(\mathbf{x}) = I_1 + I_2 + \dots + I_D - I = 0 \end{aligned}$



Решење: други Кирхофов закон одговара минимизацији енергије

Помоћна функција је

$$F(I_1, I_2, \dots, I_D, \lambda) = R_1 I_1^2 + R_2 I_2^2 + \dots + R_D I_D^2 - \lambda(I_1 + I_2 + \dots + I_D - I)$$

Парцијални изводи по оптимизационим променљивима су

$$\frac{\partial F}{\partial I_1} = 2R_1 I_1 - \lambda = 0, \quad \frac{\partial F}{\partial I_2} = 2R_2 I_2 - \lambda = 0, \dots \quad \frac{\partial F}{\partial I_D} = 2R_D I_D - \lambda = 0$$

$$I_1 = \frac{\lambda}{2R_1}, \quad I_2 = \frac{\lambda}{2R_2}, \dots, \quad I_D = \frac{\lambda}{2R_D}$$

Уврштавањем израза за струје у услов $I = g_1 = 0$ добија се

$$\frac{\lambda}{2} \left(\frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_D} \right) = I \Rightarrow \lambda = 2I \left(\frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_D} \right)^{-1} = 2IR_e$$

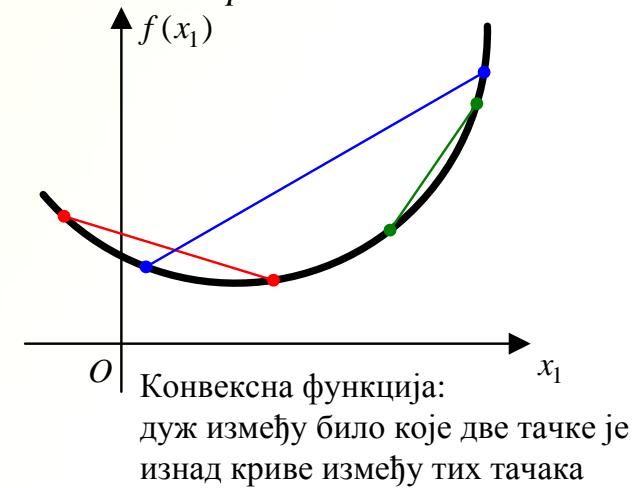
Коначно

$$I_1 = \frac{R_e}{R_1} I, \quad I_2 = \frac{R_e}{R_2} I, \dots, \quad I_D = \frac{R_e}{R_D} I$$

Исти резултат добија се применом другог Кирхофовог закона!

Karush–Kuhn–Tucker (генерализација Лагранжових мул.)

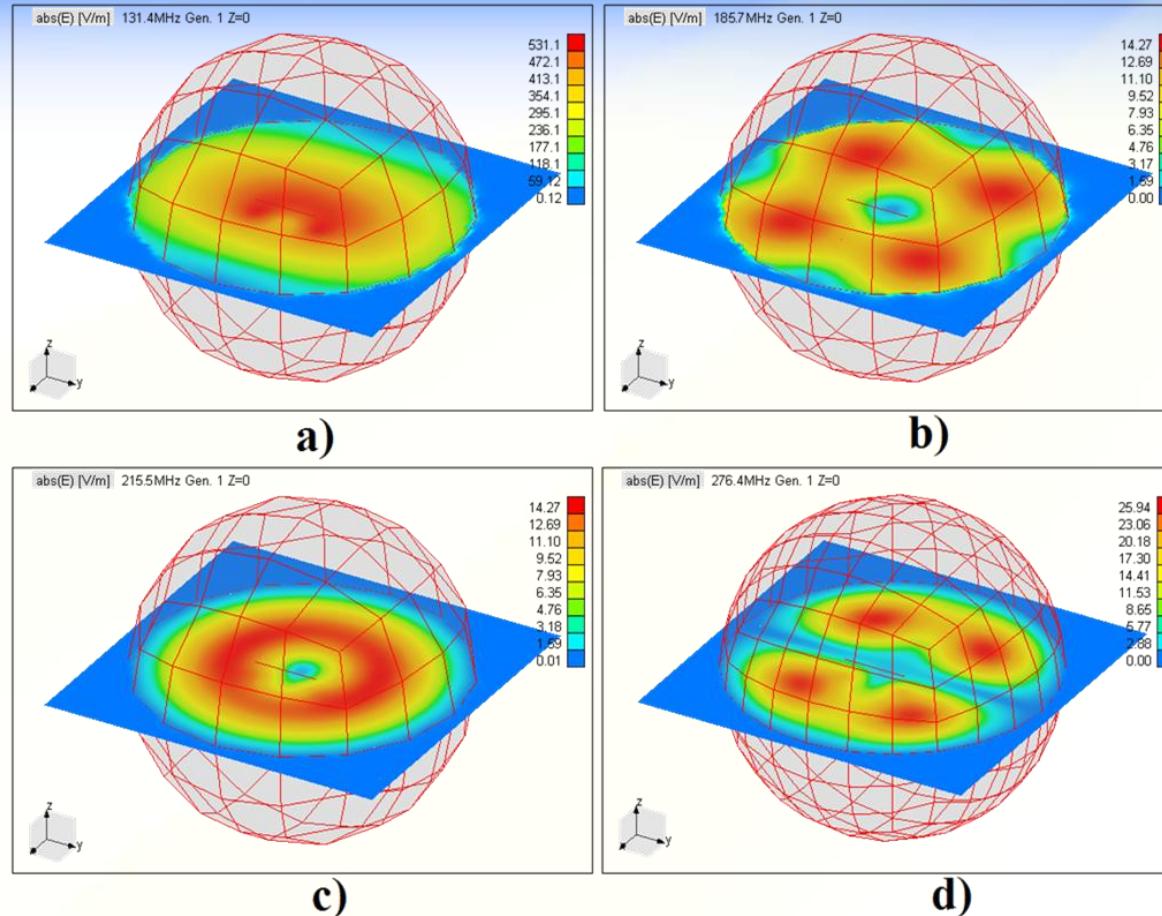
- Решавамо проблем: $\begin{aligned} & \text{minimize } f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \dots, x_D) \\ & g_k(\mathbf{x}) \leq 0, \quad k = 1, 2, \dots, m \end{aligned}$
- f, g, h су континуалне диференцијабилне функције, а f је конвексна функција
- Помоћна функција $F(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{k=1}^m \mu_k g_k(\mathbf{x}) + \sum_{p=1}^l \lambda_p h_p(\mathbf{x})$
- \mathbf{x}_0 је оптимално решење:
 - (1) $\nabla F(\mathbf{x}_0) + \sum_{k=1}^m \mu_k \nabla g_k(\mathbf{x}_0) + \sum_{p=1}^l \lambda_p \nabla h_p(\mathbf{x}_0) = 0$
 - (2) $g_k(\mathbf{x}_0) \leq 0, \quad h_p(\mathbf{x}_0) = 0$
 - (3) $\mu_k \geq 0$
 - (4) $\sum_{k=1}^m \mu_k g_k(\mathbf{x}_0) = 0$



Ограничења класичних метода

- Оптимизациона функција не мора имати нулу (ако је потребно пронаћи минимум, онда се тражи нула извода)
- Изводи оптимизационе функције не морају нужно да буду познати, а могуће је и да не постоје!
- Генерализација у случају више променљивих није једноставна за све наведене алгоритме
- За све наведене, сем метода половљења, постоје ситуације када методи дивергирају или стану

Задатак: одређивање резонантних модова у сферној шупљини



A.J. Krneta, D.I. Olcan, D.H. Trout, "On calculating resonant frequencies using general-purpose method-of-moments code," *29th Annual Review of Progress in Applied Computational Electromagnetics ACES 2013*, March 24-28, 2013., Monterey, CA, pp. 804-809.

Поставка

- Одређивање резонантних модова у сферном резонатору своди се на израчунавање нула сферних Беселових функција
- Сферне Беселове функције прве врсте реда n , $j_n(x)$, дефинисане су као

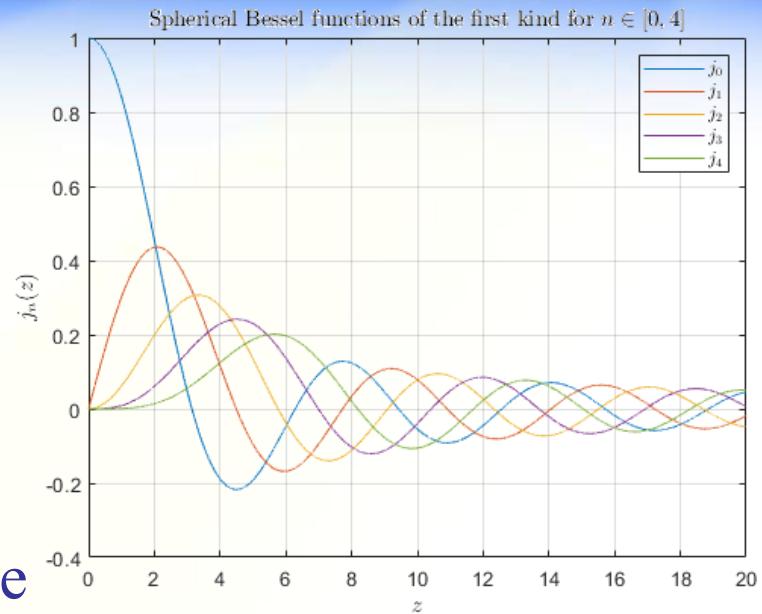
$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+\frac{1}{2}}(x),$$

где је $J_{n+\frac{1}{2}}(x)$ (обична) Беселова функција прве врсте реда $n + \frac{1}{2}$

- Израчунати нуле $x_{np0} > 0$, $j_n(x_{np0}) = 0$ **сферних Беселових** функција за редни број нуле $p = 1, 2$ и ред функције $n = 1, 2$
- Нуле је потребно одредити са апсолутном тачношћу $\pm 10^{-12}$

Имплементација

- Сферне Беселове функције прве врсте реда n
 - C/C++17: `<cmath>`
`std::sph_bessel`
(`unsigned int n, double z`)
 - Python:
`scipy.special.spherical_jn`
(`n, z, derivative=False`)
- Нацртати сферне Беселове функције задатог реда за аргумент из опсега [0,20]
 - Решење је Python код који црта одговарајући график
- Коришћењем једног од класичних метода оптимизације израчунати све потребне нуле са задатом тачношћу
 - Решење је код који израчунава тражене нуле и
 - ASCII фајл са добијеним вредностима за тражене нуле



Градијентни метод (Gradient)

- Полазна тачка $\mathbf{x}^0 = (x_1^0, x_2^0, \dots, x_D^0)$
- Нумеричка апроксимација градијента

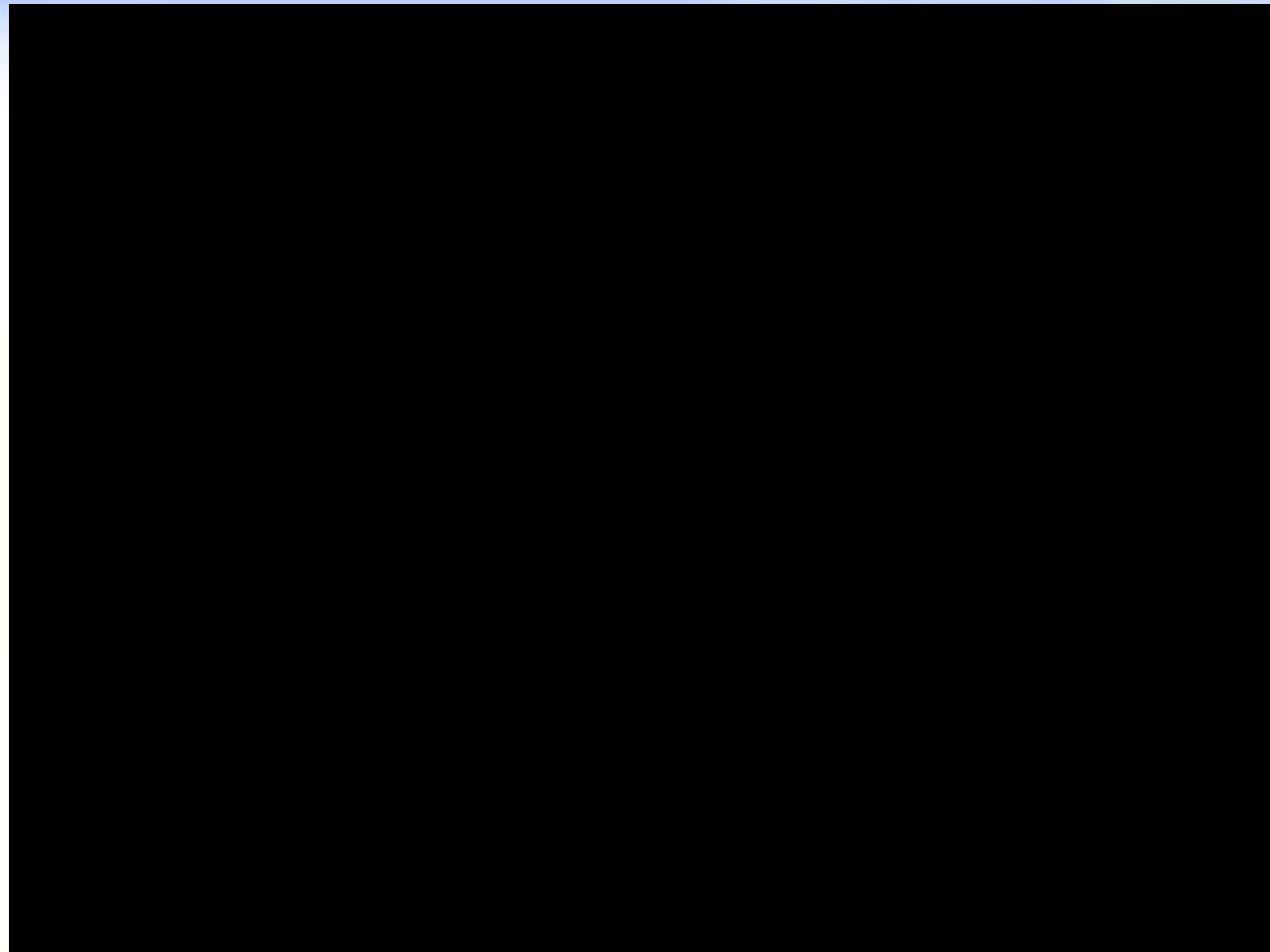
$$\text{grad}(f) \Big|_{\mathbf{x}=\mathbf{x}_0} \approx \left(\frac{\Delta f}{\Delta x_1} \mathbf{i}_{x_1} \right) \Big|_{\mathbf{x}=\mathbf{x}^0} + \left(\frac{\Delta f}{\Delta x_2} \mathbf{i}_{x_2} \right) \Big|_{\mathbf{x}=\mathbf{x}^0} + \dots + \left(\frac{\Delta f}{\Delta x_D} \mathbf{i}_{x_D} \right) \Big|_{\mathbf{x}=\mathbf{x}^0}$$

- Наредна тачка

$$x_k^1 = x_k^0 - s \frac{\left(\frac{\Delta f}{\Delta x_k} \right) \Big|_{\mathbf{x}=\mathbf{x}^0}}{\|\text{grad}(f)\|_{\mathbf{x}=\mathbf{x}^0}}, \quad k = 1, 2, \dots, D$$

- s је дужина корака која се задаје

Илустрација (10 пута поновљен градијентни метод)



Предности и мане градијентног метода

- Локални оптимизациони алгоритам
- Изузетно брза конвергенција
- Додатно повећање брзине конвергенције:
уколико је добро процењен правац
повећати корак у следећој итерацији
- Проналази само најстрмији минимум
у околини полазног решења
(то није нужно и најдубљи минимум!)
- Ефикасан уколико знамо добро полазно решење
- У D -димензионом простору захтева
 $D+1$ итерација за процену градијента
- Корак за процену градијента Δx_i зависи од проблема!

Хесијан матрица

- Претпоставимо
 $f(\mathbf{x}): R^D \rightarrow R$ и
постоји други извод
- Хесијан:
 $H(f(\mathbf{x}_k)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_D} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_D} & \frac{\partial^2 f}{\partial x_2 \partial x_D} & \dots & \frac{\partial^2 f}{\partial x_D^2} \end{bmatrix}_{\mathbf{x}_k}$
је матрица
израчуната у \mathbf{x}_k
- Алгоритам за рачунање наредне итерације:
$$\mathbf{x}_{k+1} = \mathbf{x}_k - H(f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$$

Пример коришћења Хесијана

- Функција $f(a,b) = a^2 + b^2$
- Полазно решење $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$
- Градијент $\nabla f(\mathbf{x}_1) = \begin{bmatrix} 2a \\ 2b \end{bmatrix}_{\mathbf{x}_1} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$
- Хесијан $H(f(\mathbf{x}_1)) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$
- Наредно решење $\mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Предности и мане оптимизације засноване на Хесијану

- У претходном примеру даје решење у једном кораку за произвољну полазну тачку
- Уколико оптимизациона функција има непрекидан први и други извод, оптимизација заснована на Хесијану је ефикаснија од градијентне методе
- Које се решење добија уколико оптимизациона функција има више од једног минимума?
- Израчунавање двоструких парцијалних извода може да буде проблематично код практичних оптимизационих проблема
- ...или први и други извод не морају да постоје

Практична употреба Хесијана

- Оптимизациона функција са више променљивих
- Тејлоров развој

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}) \Delta\mathbf{x} + \dots$$

- Проблем је рачунање Хесијана
- Најпознатији је Broyden–Fletcher–Goldfarb–Shanno алгоритам

Broyden–Fletcher–Goldfarb–Shanno (BFGS) алгоритам

- \mathbf{x}_0 је полазна тачка, \mathbf{B}_0 је нулта ест. Хес., типично \mathbf{I}
- У \mathbf{x}_k одреди се правац претраге \mathbf{p}_k

$$\mathbf{B}_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

- Промени се позиција $\mathbf{s}_k = \alpha_k \mathbf{p}_k \quad \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$
- Нека је $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
- Естимација Хесијана у наредној тачки је

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k}$$

- Понавља се за наредну тачку у простору

Комплетно и парцијално решење

- Алгоритми који раде са комплетним решењем
 - потпуно дефинисани проблем
 - заустављање алгоритма даје до тада најбоље решење
 - примери: систематско претраживање, случајно претраживање, hill-climbing, градијентни метод...
- Алгоритми који раде са парцијалним решењем
 - апроксимативно решење (surrogate models, fitness fitting, fitness approximation, space mapping, Kriging, response surface methodology...)
 - заустављени алгоритам не мора да има корисно решење (случајно претраживање без памћења најбољег пронађеног...)
- Инжењерски употребљиви су алгоритми који дају најбоље решење до тада

Constrained vs. Unconstrained optimization

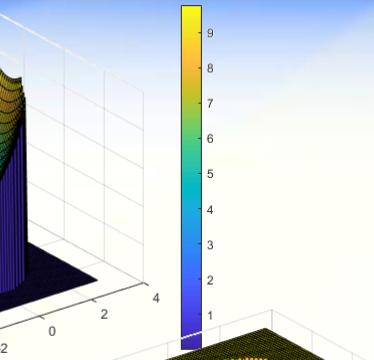
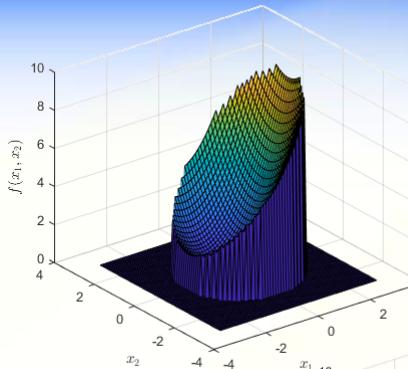
- Како алгоритам који решава оптимизационе проблеме без ограничења искористити за решавање опт. проблема са ограничењима?
- Најједноставнији приступ:
уколико ограничења нису задовољена додати “пенал” на оптимизациону функцију
$$F(\mathbf{x}) = f(\mathbf{x}) + f_p(g(\mathbf{x}))$$
где је $g(\mathbf{x})$ услов који није испуњен, а $f_p(g(\mathbf{x}))$ додатна функција (“пенал”) који се додаје

Укључивање ограничења у оптимизациону функцију

- Пример

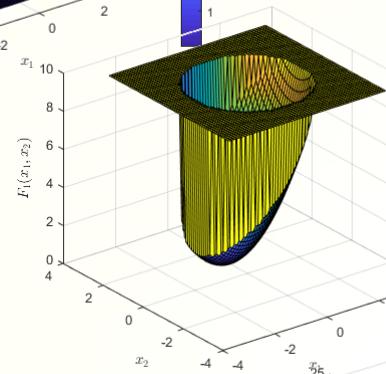
$$\text{minimize } f(x_1, x_2) = (x_1 + 1)^2 + x_2^2 + 1$$

$$g(x_1, x_2) : x_1^2 + x_2^2 \leq 4$$



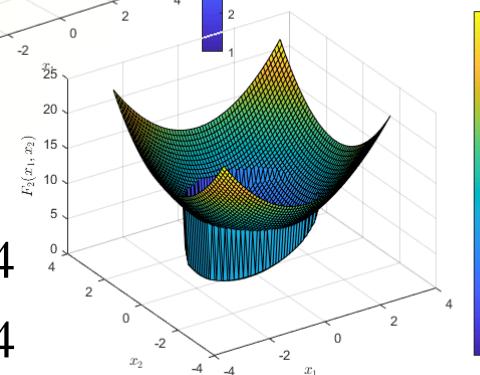
- Једноставно решење

$$\text{minimize } F_1(x_1, x_2) = f(x_1, x_2) + \begin{cases} 0, & x_1^2 + x_2^2 \leq 4 \\ 10, & x_1^2 + x_2^2 > 4 \end{cases}$$



- Боље (сложеније) решење

$$\text{minimize } F_2(x_1, x_2) = f(x_1, x_2) + \begin{cases} 0, & x_1^2 + x_2^2 \leq 4 \\ 10 + (x_1^2 + x_2^2 - 4), & x_1^2 + x_2^2 > 4 \end{cases}$$



Nelder-Mead

Симплекс Алгоритам

- Основне референце
 - J. A. Nelder, R. Mead “A Simplex Method for Function Minimization”, *The Computing Journal* 7, pp. 308-313, 1965.
 - Lagarias, J. C., J. A. Reeds, M. H. Wright, and P. E. Wright. “Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions.” *SIAM Journal of Optimization*. Vol. 9, Number 1, 1998, pp. 112–147.
 - Gao, F. and Han, L. “Implementing the Nelder-Mead simplex algorithm with adaptive parameters,” *Computational Optimization and Applications*, 51:1, 2012, pp. 259-277.
- Један од најчешће коришћених алгоритама за локалну NLP оптимизацију
- Често се референцира само под именом симплекс или Nelder-Mead
- Понекад се назива и “амоева” алгоритам

Имплементације

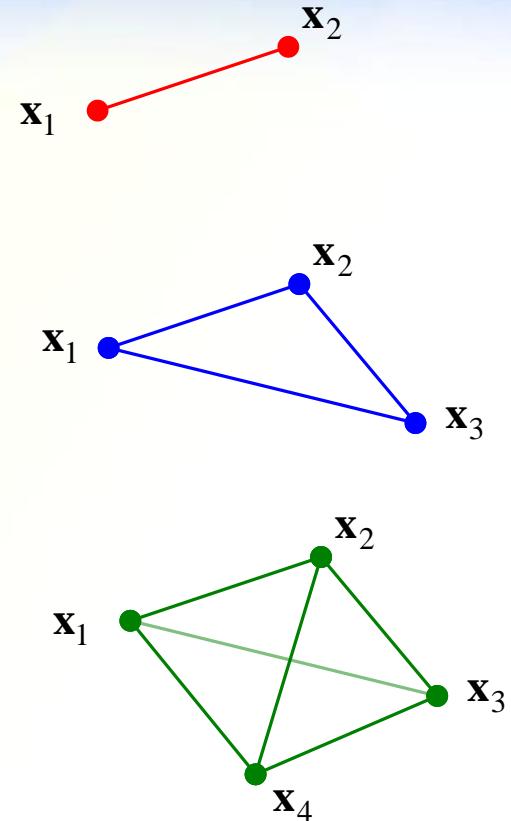
- MATLAB:
 - `fminsearch` uses the **Nelder-Mead** simplex algorithm as described in Lagarias et al.
- Mathematica:
 - The Nelder-Mead method is implemented as `NMinimize[f, vars, Method -> "NelderMead"]`.
- Python:
 - `scipy.optimize.minimize(fun, x0, args=(), method='Nelder-Mead', ...)`
- Numerical recipes in C: The art of scientific computing
 - `void amoeba(float **p, float y[], int ndim, float ftol, float (*funk)(float []), int *nfunk)`
[Multidimensional minimization of the function `funk(x)` based on Nelder, J.A., and Mead, R. 1965, Computer Journal, vol. 7, pp. 308-313]

Искључиво се користе вредности описне функције

- Алгоритам користи искључиво вредности описне функције
- Не рачуна изводе описне функције (ни експлицитно ни имплицитно)
- Спада у класу директних алгоритама (алгоритми који не користе изводе опт. ф.)
- Алгоритам се заснива на трансформацијама геометријске фигуре под именом симплекс

Симплекс фигура

- Симплекс фигура има ненулту “запремину”
- Састоји се од $D+1$ тачке у D димензионом оптимизационом простору
 - за 1-D проблем симплекс је дуж (“запремина” је дужина дужи)
 - за 2-D проблем симплекс је троугао (“запремина” је површина троугла)
 - за 3-D проблем симплекс је тетраедар итд. (“запремина” је запремина тетраедра)
 - ...
- Идеја: поступно се мења једна по једна тачка симплекса док се он не смањи тако да се све тачке симплекса налазе у епсилон околини решења



Параметри симплекс алгоритма

- У општем случају постоје четири коефицијента који се задају унапред
 - коефицијент рефлексије α ,
 - коефицијент контракције β ,
 - коефицијент експанзије γ и
 - коефицијент сажимања σ
- Коефицијент сажимања није експлицитно дефинисан у оригиналном раду у коме је алгоритам описан (усвојен је као константа)

Избор параметара симплекс алгоритма

- Ови параметри задовољавају следеће услове

$$\alpha > 0, \quad 0 < \beta < 1, \quad \gamma > 1, \quad 0 < \sigma < 1$$

- Готово универзалан избор вредности ових параметара

$$\alpha = 1, \quad \beta = \frac{1}{2}, \quad \gamma = 2, \quad \sigma = \frac{1}{2}$$

- Ове вредности установљене су још у оригиналном раду, а даља истраживања су потврдила да је овакав избор врло добар за велики број различитих проблема

Почетак симплекс алгоритма

- Алгоритам почиње формирањем $D+1$ тачке симплекса (обавезно ненулте запремине) у простору и израчунавањем оптимизационе функције у тим тачкама
- На почетку k -тог корака алгоритма, задато је $D+1$ тачака које заједно формирају симплекс
- Сваки корак алгоритма почиње сортирањем и обележавањем ових тачака $\mathbf{x}_1(k), \mathbf{x}_2(k), \dots, \mathbf{x}_{D+1}(k)$

$$f_i^{(k)} = f(\mathbf{x}_i^{(k)}) \quad f_1^{(k)} \leq f_2^{(k)} \leq \dots \leq f_{D+1}^{(k)}$$

Најбоље и најгоре тачке и функције у оквиру алгоритма

- У сваком кораку алгоритма формира се нови скуп тачака (симплекс) који је различит од симплекса у претходном кораку
- Циљ алгоритма је проналажење минимума функције f
 - тачка $\mathbf{x}_1(k)$ се назива најбоља тачка,
 - $\mathbf{x}_{D+1}(k)$ је најгора тачка, а
 - $\mathbf{x}_D(k)$ је друга најгора тачка
- Најбоља функција грешке је $f_1^{(k)}$, а најгора функција грешке је $f_{D+1}^{(k)}$

Један корак симплекс алгоритма

- Један корак алгоритма састоји се из следећих пет операција
 - (1) сортирање
 - (2) рефлексија
 - (3) експанзија
 - (4) контракција
 - (5) сажимање
- Само неке операције се извршавају у оквиру једног корака
- Која ће се операција (или операције) извршити зависи од текућег стања

1. Сортирање

(1) Сортирање: сортирати темена симплекса тако да је испуњен услов

$$f_1 \leq f_2 \leq \dots \leq f_{D+1}$$

Алгоритам за сортирање није критичан, јер симплекс по правилу има до највише неколико стотина тачака

2. Рефлексија

(2) **Рефлексија:** Израчунати центроид D најбољих тачака,

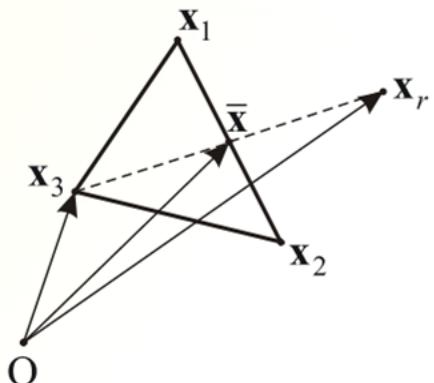
$$\bar{\mathbf{x}} = \frac{1}{D} \sum_{i=1}^D \mathbf{x}_i$$

и у односу на њега израчунати тачку рефлексије \mathbf{x}_r према формулама

$$\mathbf{x}_r = \bar{\mathbf{x}} + \alpha \cdot (\bar{\mathbf{x}} - \mathbf{x}_{D+1}) = (1 + \alpha) \cdot \bar{\mathbf{x}} - \alpha \cdot \mathbf{x}_{D+1}$$

Израчунати f_r

Уколико је $f_1 \leq f_r < f_D$ изоставити најгору тачку, а уместо ње уврстити \mathbf{x}_r у симплекс и завршити корак



3. Експанзија

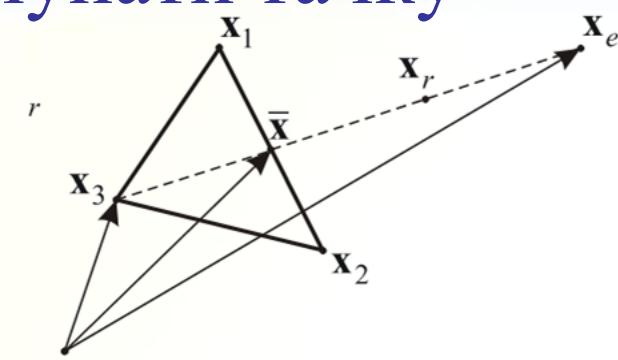
(3) Експанзија: ако је $f_r < f_1$ израчунати тачку експанзије \mathbf{x}_e према формулама

$$\mathbf{x}_e = \bar{\mathbf{x}} + \gamma \cdot (\mathbf{x}_r - \bar{\mathbf{x}}) = \gamma \cdot \mathbf{x}_r + (1 - \gamma) \cdot \bar{\mathbf{x}}$$

Израчунати f_e

Ако је $f_e < f_r$ изоставити најгору тачку и уместо ње уврстити \mathbf{x}_e у симплекс и завршити корак

У супротном, ако је $f_e \geq f_r$ уврстити \mathbf{x}_r и завршити корак



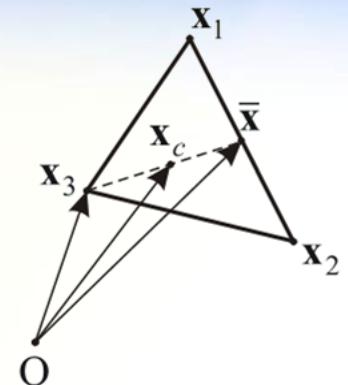
4. Контракција

(4) Контракција: Ако је $f_r \geq f_{D+1}$ доделити

$$\mathbf{x}_{D+1} = \begin{cases} \mathbf{x}_r, & f_r \leq f_{D+1} \\ \mathbf{x}_{D+1}, & f_r > f_{D+1} \end{cases}$$

израчунати тачку контракције \mathbf{x}_c као

$$\mathbf{x}_c = \bar{\mathbf{x}} + \beta \cdot (\mathbf{x}_{D+1} - \bar{\mathbf{x}}) = \beta \cdot \mathbf{x}_{D+1} + (1 - \beta) \cdot \bar{\mathbf{x}}$$



Израчунати f_c , уврстити \mathbf{x}_c и завршити корак ако $f_c \leq \min(f_{D+1}, f_r)$,

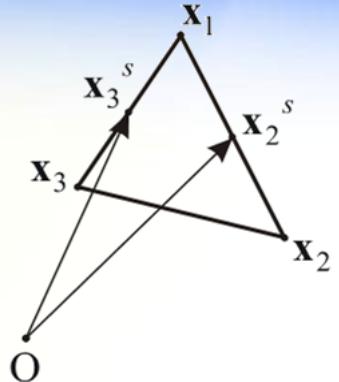
(тј. тачка контракције је боља од \mathbf{x}_{D+1} и \mathbf{x}_r)

У супротном извршити сажимање

5. Сажимање

(5) Сажимање: заменити све тачке осим најбоље са

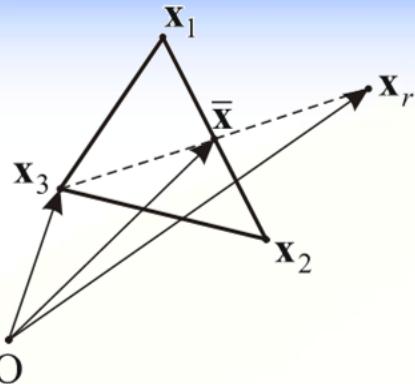
$$\mathbf{x}_i = \mathbf{x}_1 + \sigma \cdot (\mathbf{x}_i - \mathbf{x}_1), \quad i = 2, 3, \dots, D+1$$



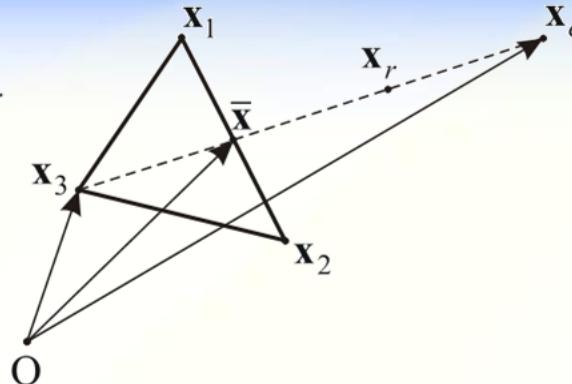
израчунати функцију грешке у свим новим тачкама и завршити корак

- * По завршетку једног корака, а пре преласка у наредни корак потребно је извршити проверу да ли је алгоритам испунио услове за завршетак оптимизације

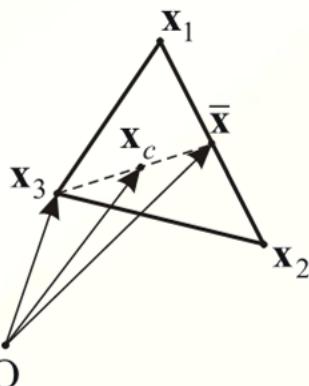
Графички приказ трансформација симплекса у 2-D простору



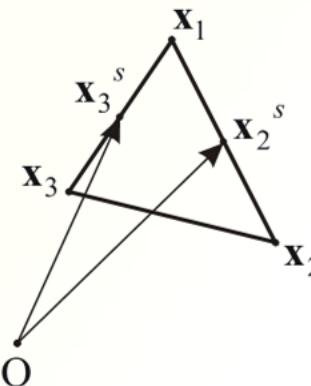
(а) Рефлексија



(б) Експанзија

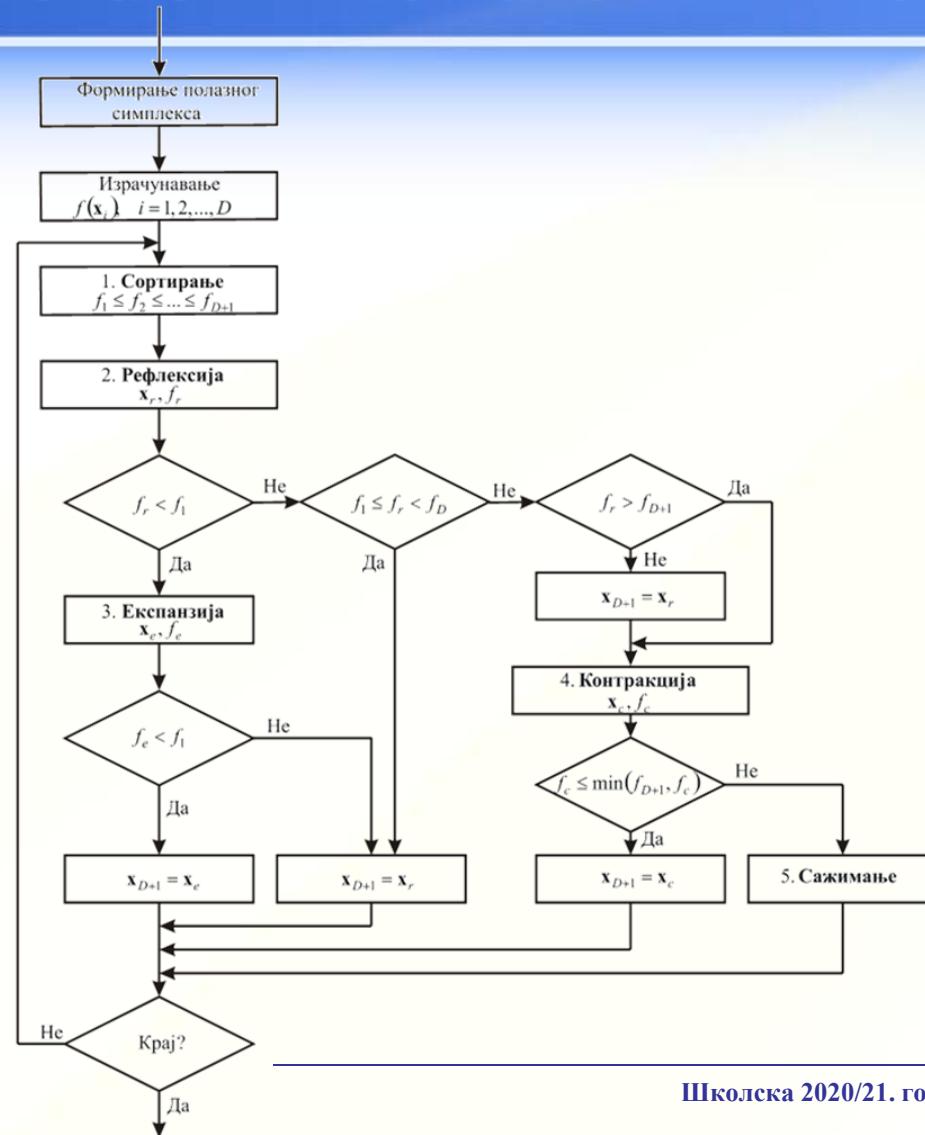


(в) Контракција



(г) Сажимање

Блок дијаграм Nelder-Mead симплекс алгоритма



Формирање полазног симплекса

- Типично је позната једна тачка \mathbf{x}_0 (почетно решење), осталих D је потребно формирати
- Најједноставнији приступ (хипер-правоугаоник):
$$\mathbf{x}_1 = \mathbf{x}_0 + (\Delta x_1, 0, \dots, 0)$$
$$\mathbf{x}_2 = \mathbf{x}_0 + (0, \Delta x_2, \dots, 0)$$

 \dots
$$\mathbf{x}_D = \mathbf{x}_0 + (0, 0, \dots, \Delta x_D)$$
- Проблеми:
 - Како изабрати Δx_k ?
 - Неке тачке могу изаћи из оптимизационог простора
- Провера да ли је тачка у оптимизационом простору и ако није промена знака Δx_k
- Могуће је користити и друге D димензионе фигуре (сфера, коцка, итд.)

Адаптивна промена параметара

- Параметри алгоритма могу се подешавати у односу на број димензија оптимизационог простора D

- Један могући приступ

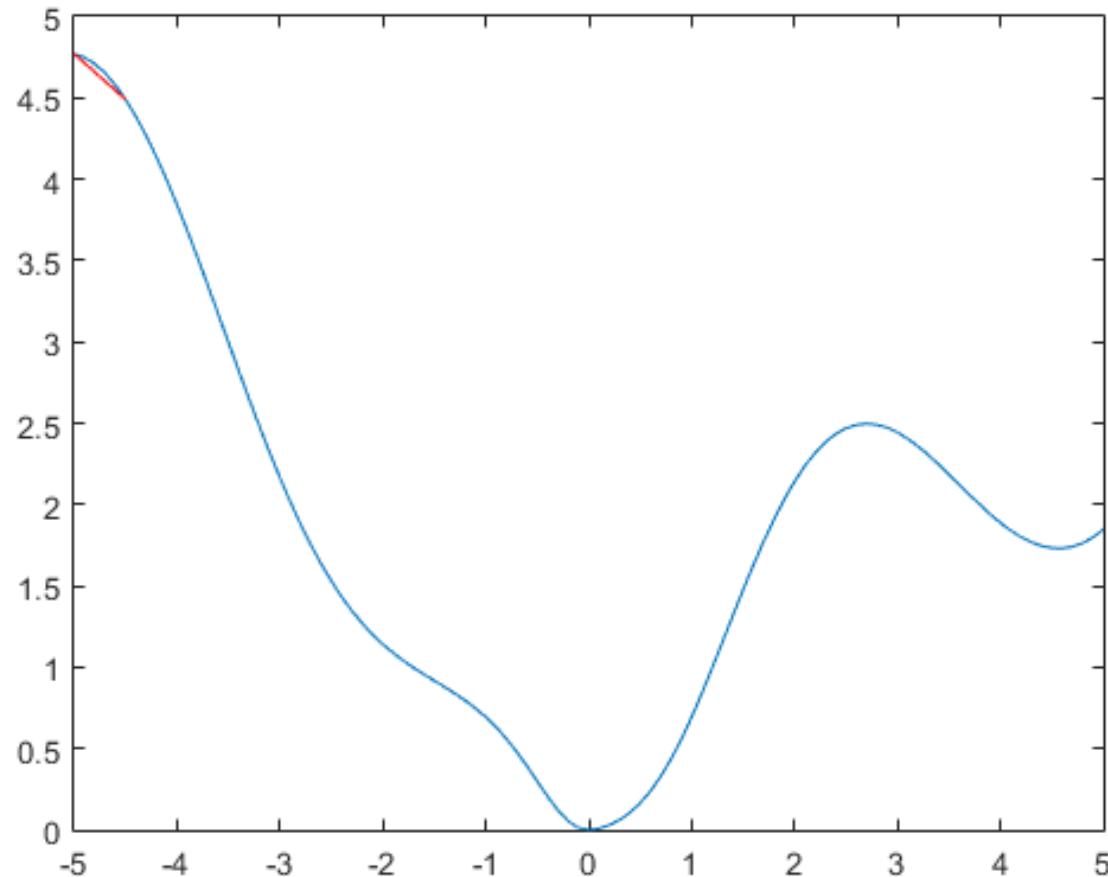
$$\alpha = 1, \quad \beta = \frac{3}{4} - \frac{1}{2D}, \quad \gamma = 1 + \frac{2}{D}, \quad \sigma = 1 - \frac{1}{D}$$

- Повољно у случајевима “великог” D (од ~ 10 до ~ 30)
- Python имплементација адаптира параметре у зависности од D
- Оптимизација параметара оптимизационог алгоритма (енглески: *metaheuristics*)

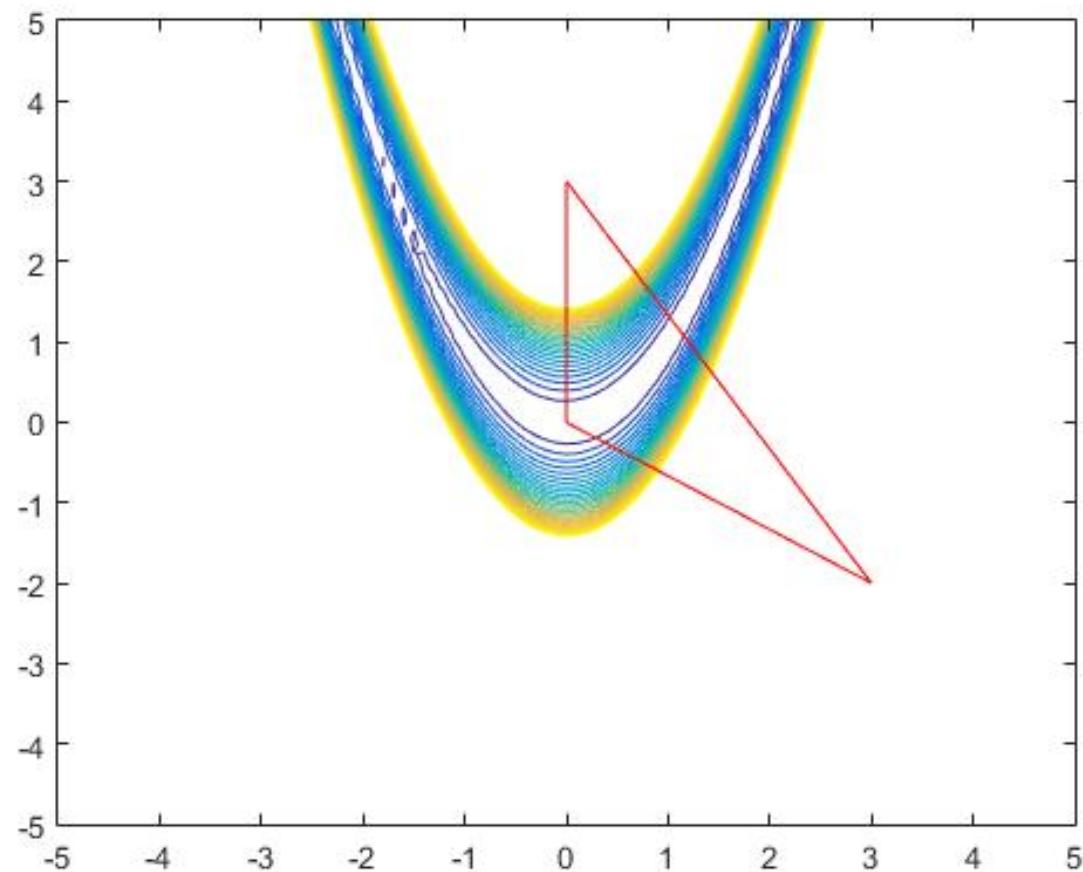
Колика је сложеност симплекс алгоритма?

- У пракси алгоритам типично конвергира за $O(D^2)$ итерација тј. $\sim D^2$ израчунавања оптимизационе функције
- У литератури постоје теоријска разматрања само за мањи број димензија оптимизационог простора, типично до 10
- У општем случају,
није позната сложеност овог алгоритма

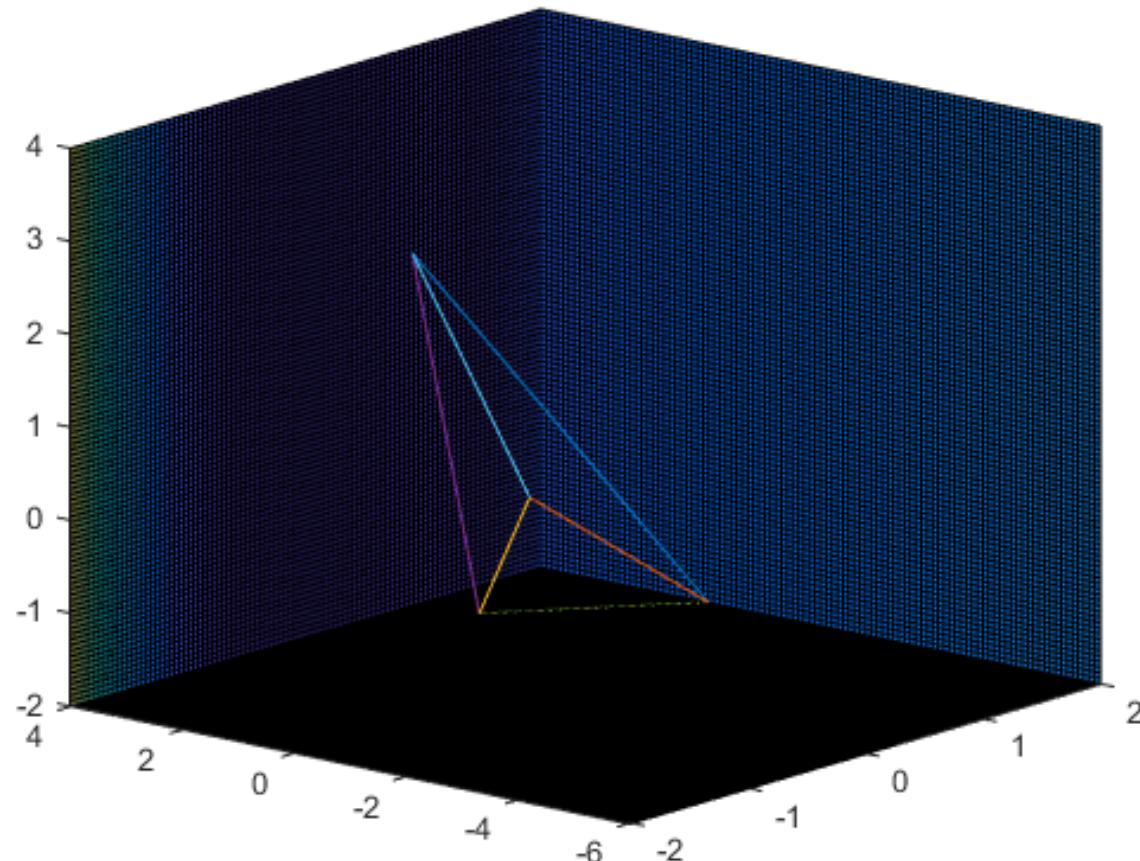
Илустрација рада симплекс алгоритма (1D)



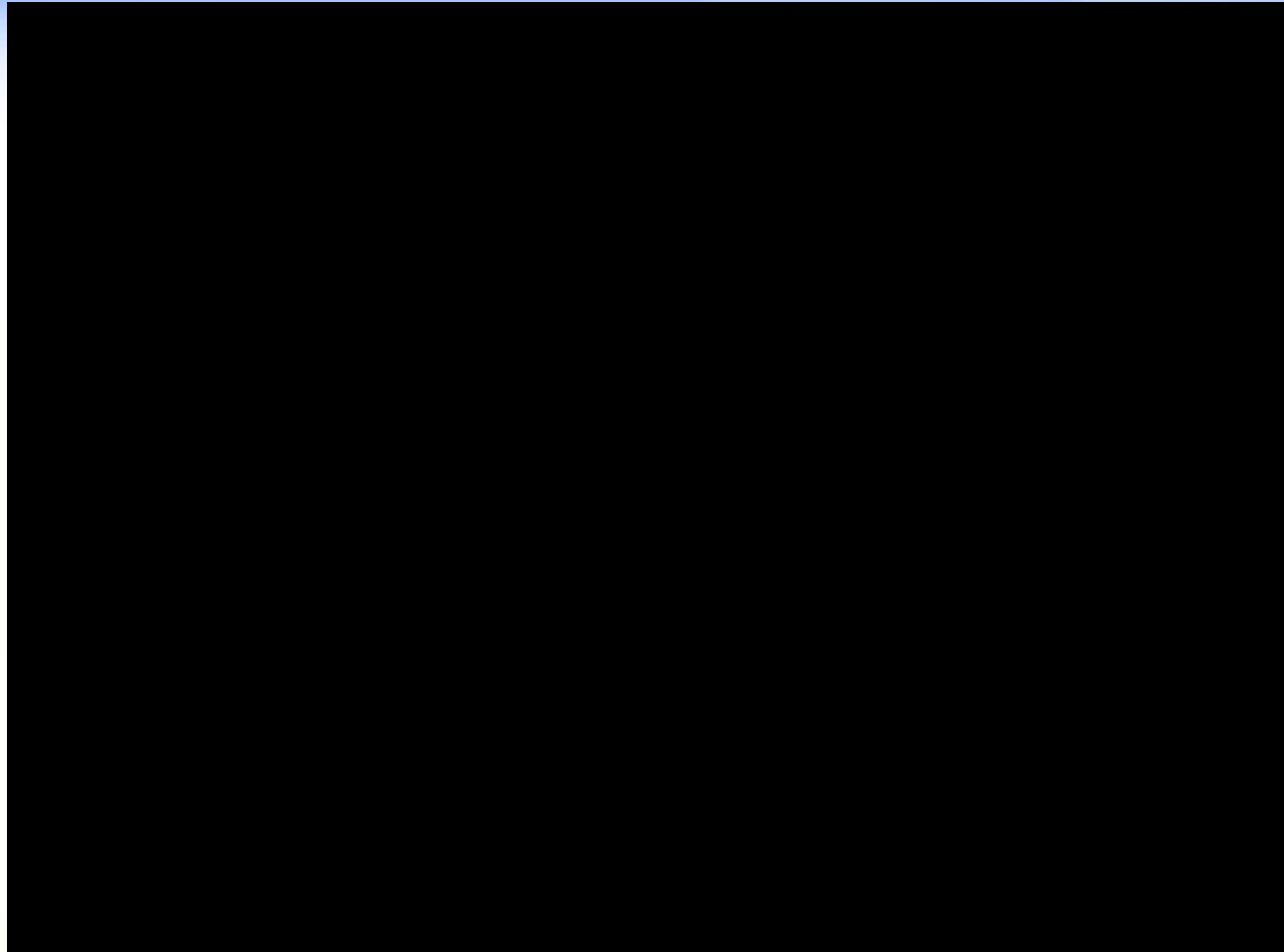
Илустрација (2D)



Илустрације (3D)



Ток симплекс алгоритма 10 пута

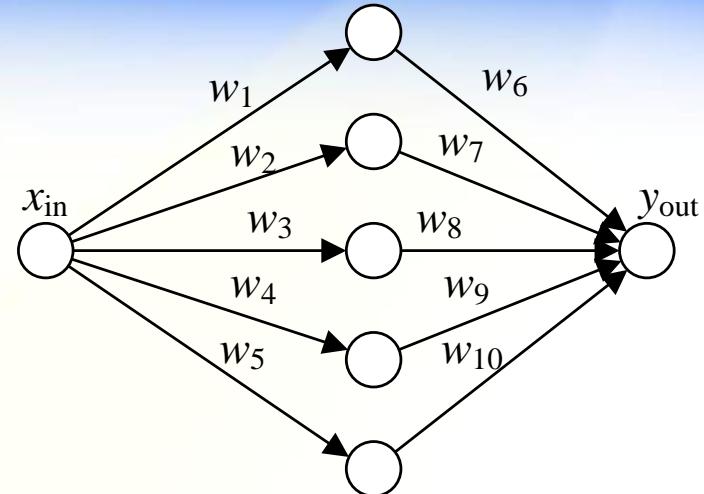


Практични закључци о симплекс алгоритму

- Спада у групу локалних оптимизационих метода
- Поседује способност да, у одређеним ситуацијама, прескочи из једног локалног минимума у други уколико се тиме добија боље решење
- Ефекат “прескакања” се не дешава увек и зависи од
 - конкретног проблема и
 - полазног симплекса
- Ова особина даје предност симплекс алгоритму у односу на (све) друге локалне оптимизационе алгоритме
- Постоје напори да се строго математички докаже због чега симплекс алгоритам постиже врло добре резултате у пракси, међутим генералног доказа нема
- Симплекс се може посматрати и као генерализација метода половљења интервала

Задатак за вежбе (поставка)

- На слици је приказана једноставна неурална мрежа са једним улазом, једним излазом и једним скривеним слојем са 5 чврода (неурона)
- Оптимизацијом је потребно пронаћи вредности тежинских коефицијената $\mathbf{w} = (w_1, w_2, \dots, w_{10})$ тако да мрежа представља апроксимацију следеће функције $y_{\text{training}}(x)$
- Активациона функција сваког чвора је дата изразом



$$y_{\text{training}}(x) = \frac{1}{2} \sin(\pi x)$$

$$-1 \leq x \leq +1$$

$$a(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Задатак за вежбе (детаљи)

- Вредност на улазу (x_k) и излазу чвора (y_k) је
 $x_k = w_k x_{in}, \quad k = 1, 2, \dots, 5$
 $y_k = a(x_k)$
- Вредност на излазу мреже дата је изразом
$$y_{out}(x_{in}) = a\left(\sum_{k=1}^5 w_{k+5} a(w_k x_{in})\right)$$
- Оптимизациона функција је дата изразом

$$f(\mathbf{w}) = \sqrt{\sum_{\substack{x_{in}=-1 \\ \Delta x_{in}=0,1}}^{+1} (y_{out}(x_{in}) - y_{training}(x_{in}))^2}$$

Задатак (имплементација)

- Написати код који рачуна излаз задате мреже, на основу улаза
- Написати код који рачуна оптимизациону функцију за један избор тежинских фактора
- Користити једну од следећих ставки
 - C/C++ код за Nelder-Mead симплекс
<http://mtt.etf.rs/si/IOA/NMSimplex.zip>
 - Python функција
`scipy.optimize.minimize(..., method='nelder-mead', ...)`
 - или код неког другог класичног метода обрађеног на курсу
- Одредити коефицијенте $\mathbf{w} = (w_1, w_2, \dots, w_{10})$ тако да оптимизациона функција буде мања од 10^{-2}
- Дозвољени опсег за коефицијенте је $-10 \leq w_1, w_2, \dots, w_{10} \leq +10$
- У пратећи ASCII фајл записати
 - коефицијенте \mathbf{w} са 15 цифара после децималне тачке у облику $(w_1, w_2, \dots, w_{10})$,
 - написати вредност минималне пронађене оптимизационе функције
- Нацртати на истом графику $y_{\text{out}}(x)$, $y_{\text{training}}(x)$, $-1 \leq x \leq +1$

The Best of 20th Century: Top 10 Algorithms

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Algos is the Greek word for pain. *Algor* is Latin, to be cold. Neither is the root for *algorithm*, which stems instead from al-Khwarizmi, the name of the ninth-century Arab scholar whose book *al-jabr wa'l muqabalah* devolved into today's high school algebra textbooks. Al-Khwarizmi stressed the importance of methodical procedures for solving problems. Were he around today, he'd no doubt be impressed by the advances in his eponymous approach.

Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory and Francis Sullivan of the Center for Computing Sciences at the Institute for Defense Analyses put together a list they call the "Top Ten Algorithms of the Century."

"We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century," Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here's the CiSE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

The Best of 20th Century: Top 10 Algorithms

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.



In terms of widespread use, George Dantzig's simplex method is among the most successful algorithms of all time.

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of **Krylov subspace iteration methods**.

These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that A is a huge $n \times n$ matrix, so that the algebraic answer $x = b/A$ is not so easy to compute. (Indeed, matrix "division" is not a particularly useful concept.) Iterative methods—such as solving equations of the form $Kx_{i+1} = Kx_i + b - Ax_i$ with a simpler matrix K that's ideally "close" to A —lead to the study of Krylov subspaces. Named for the Russian mathematician Nikolai Krylov, Krylov subspaces are spanned by powers of a matrix applied to an initial "remainder" vector $r_0 = b - Ax_0$. Lanczos found a nifty way to generate an orthogonal basis for such a subspace when the matrix is symmetric. Hestenes and Stiefel proposed an even niftier method, known as the conjugate gradient method, for systems that are both symmetric and positive definite. Over the last 50 years, numerous researchers have improved and extended these algorithms. The current suite includes techniques for non-symmetric systems, with acronyms like GMRES and Bi-CGSTAB. (GMRES and Bi-CGSTAB premiered in *SIAM Journal on Scientific and Statistical Computing*, in 1986 and 1992,

The Best of 20th Century: Top 10 Algorithms

- 1946: Monte Carlo method
- **1947: Simplex method for linear programming (Dantzig)**
- 1950: Krylov subspace iteration method
- 1951: Decompositional approach to matrix computation
- 1957: Fortran optimizing compiler
- 1959–61: computing eigenvalues, QR algorithm
- 1962: Quicksort
- 1965: Fast Fourier Transformation
- 1977: Integer relation detection algorithm
- 1987: Fast multipole algorithm

Линеарно “програмирање”

- Линеарно програмирање је специјалан случај нелинеарног програмирања (NLP)
- Оптимизациона функција је линеарна функција

$$f(x_1, x_2, \dots, x_D) = c_1 x_1 + c_2 x_2 + \dots + c_D x_D = \sum_{k=1}^D c_k x_k$$

- Услови су линеарне функције (једнакости или неједнакости)

$$a_1 x_1 + a_2 x_2 + \dots + a_D x_D \begin{cases} \leq \\ = \\ \geq \end{cases} b_1$$

Формализација LP (стандардни облик)

- Максимизације оптимизационе функције

$$f(x_1, x_2, \dots, x_D) = c_1 x_1 + c_2 x_2 + \dots + c_D x_D = \sum_{k=1}^D c_k x_k$$

- Услови (укупно m услова, $b_1, b_2, \dots, b_m \geq 0$)

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1D}x_D \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2D}x_D \leq b_2$$

⋮

,

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mD}x_D \leq b_m$$

$$x_1, x_2, \dots, x_D \geq 0$$

Превођење у стандардни облик

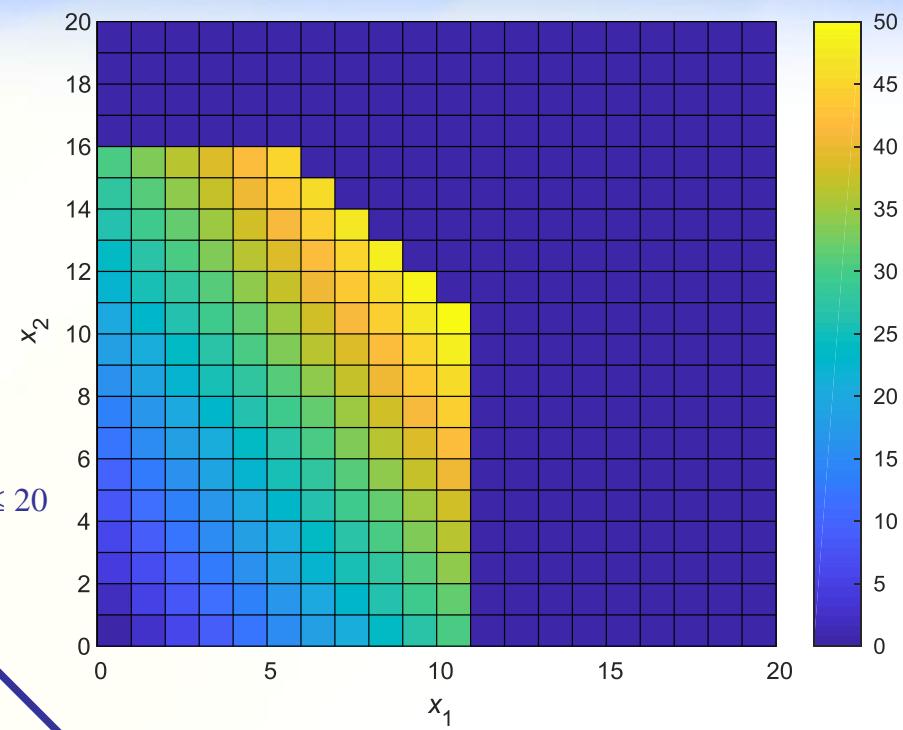
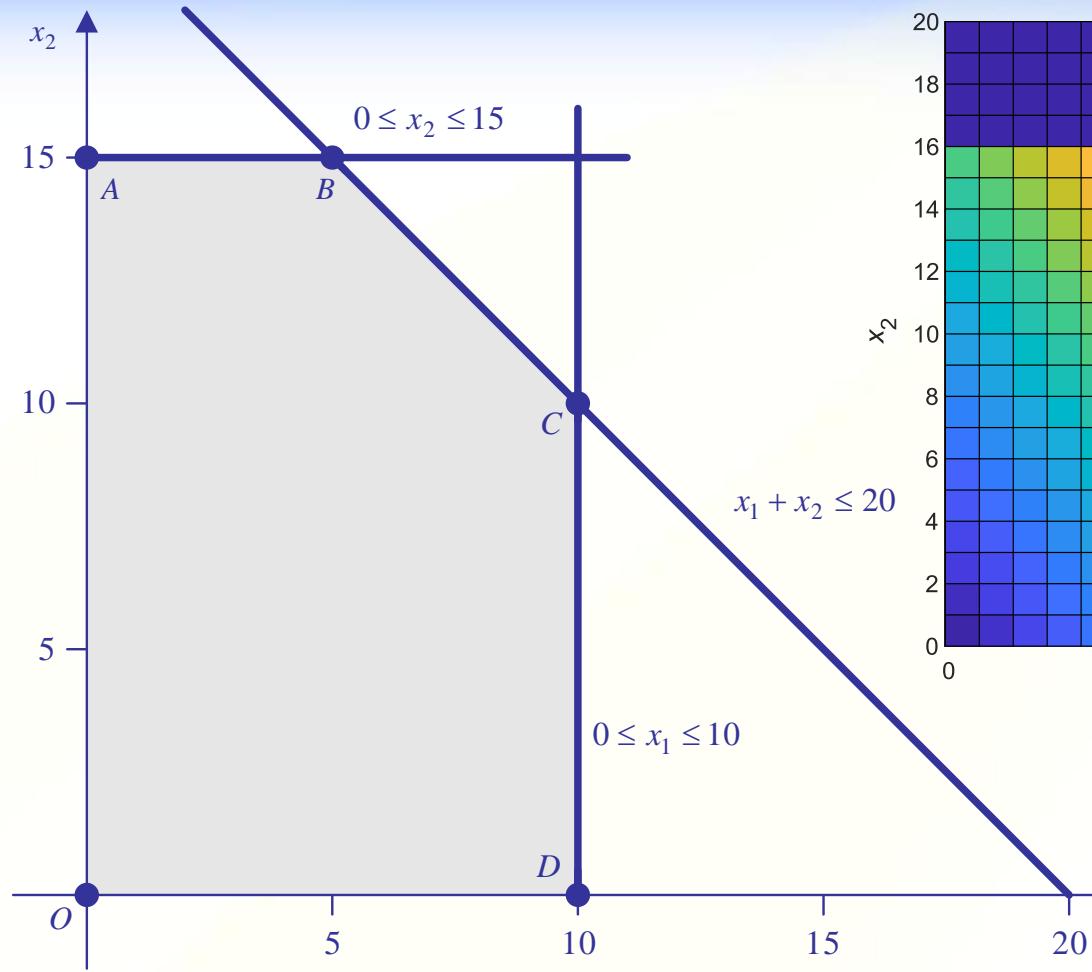
- $\min f \rightarrow \max (-f)$
- Све променљиве морају бити ненегативне
 - неограничена променљива
 $x \rightarrow x = x_1 - x_2$ при чему је $x_1, x_2 \geq 0$
 - непозитивна променљива:
 $x \leq 0 \rightarrow x = x_1 - x_2$ при чему је $x_1, x_2 \geq 0$
(најједноставније $x_1=0$ и $x_2 \geq 0$)
- Једнакости
 - Редукција броја непознатих
 $ax_1 + bx_2 = c \rightarrow x_1 = (c - bx_2)/a$ и замена у свим изразима где се појављује
(може да буде компликовано у случају великог броја услова) или
 - Претварање у две неједнакости
 $ax_1 + bx_2 = c \rightarrow ax_1 + bx_2 \leq c$ и $ax_1 + bx_2 \geq c$
- Све неједнакости се преводе у облик
 $ax_1 + bx_2 \leq c$

Пример: проблем интернет провайдера

Интернет провайдер са ограниченим пропусним опсегом. Две опције за корисника (1) неограничен приступ (flat-rate) и (2) приступ са ограниченим протоком и количином података. Нека је број корисника са неограниченом приступом означен са x_1 , а број корисника са ограниченим приступом x_2 . Бројеви корисника припадају скупу целих бројева. Корисници који приступају интернету са неограниченом приступом плаћају месечно α , а корисници који приступају интернету са ограниченим приступом плаћају β , при чему је $\alpha > \beta$. Услед ограничених капацитета провайдера, постоје горње границе за x_1 и x_2 , као и за $x_1 + x_2$. Циљ провайдера је да максимизира зараду $f(x_1, x_2) = \alpha x_1 + \beta x_2$. Који је оптималан избор x_1 и x_2 ?

Ради илустрације, сматраћемо да је $\alpha = 3$, $\beta = 2$, $0 \leq x_1 \leq 10$, $0 \leq x_2 \leq 15$ и $x_1 + x_2 \leq 20$.

Графички приступ



Домен решења и екстремне тачке

- Домен решења је означен сивом бојом
- Домен је ограничен тачкама O, A, B, C, D које представљају крајње (екстремне) тачке
- Колико има екстремних тачака у општем случају са N услова?
- Решење проблема мора бити у једној од екстремних тачака домена решења

Решење проблема

- Како решити проблем у општем случају?
- Одређивање свих екстремних тачака домена и провера описне функције у њима

$$f_O(0,0) = 0$$

$$f_A(0,15) = 30$$

$$f_B(5,15) = 45$$

$$f_C(10,10) = 50$$

$$f_D(10,0) = 30$$

Dantzig simplex алгоритам

- Најпознатији алгоритам за решавање LP
- G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1959.
- George B. Dantzig and Mukund N. Thapa. 1997. *Linear programming 1: Introduction*. Springer-Verlag.
- George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.
- 1975. Нобелова награда за LP програмирање у економији L. Kantorovich и T.C. Koopmans

Почетак алгоритма

- Припрема за примену алгоритма, превођење неједнакости у једнакости
увођење помоћне (изравнавајуће) променљиве (енг: slack variable)

$$x_1 + x_2 \leq 20 \rightarrow x_1 + x_2 + s_1 = 20, \quad s_1 \geq 0$$

$$x_1 \leq 10 \rightarrow x_1 + s_2 = 10, \quad s_2 \geq 0$$

$$x_2 \leq 15 \rightarrow x_2 + s_3 = 15, \quad s_3 \geq 0$$

$$x_1, x_2 \geq 0$$

$$s_1, s_2, s_3 \geq 0$$

- Све неједнакости претворене у једнакости,
све променљиве су ненегативне
- Могуће је добити
 - поддетерминисан систем линеарних једначина
 - систем линеарних једначина
 - предетерминисан систем линеарних једначина

Компактан (матрични) запис услови и опт. функције

- Услови

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 15 \end{bmatrix} \rightarrow \mathbf{Ax} = \mathbf{b}$$

- Оптимизациона функција (max)

$$f(\mathbf{x}) = Z = \sum_{k=1}^D c_k x_k = \mathbf{c}^T \mathbf{x} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Канонични облик

- Циљ: максимизирати Z
- Канонични облик LP (simplex tableau)

$$\begin{bmatrix} 1 & -\mathbf{c}^T \\ 0 & \mathbf{A} \end{bmatrix} \begin{bmatrix} Z \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}$$

- У општем случају може се свести на овај облик
(све једнакости и све променљиве ненегативне)

Dantzig simplex

- Поћи из једне екстремне тачке
(подразумева се да је бар једна таква тачка позната, што у општем случају не мора да буде тривијално)
- Проверити да ли је могуће повећати Z преласком у другу суседну екстремну тачку
- Завршити алгоритам уколико није могуће повећати Z (решење је пронађено)

Dantzig алгоритам за проблем интернет провајдера: почетак

- $\max f = 3x_1 + 2x_2 = Z$
 $x_1 + x_2 + s_1 = 20$
 $x_1 + s_2 = 10$
 $x_2 + s_3 = 15$
- зависне променљиве:
 $x_1, x_2 \geq 0$
- независне (основне) променљиве:
 $s_1, s_2, s_3 \geq 0$
- Одређивање полазне тачке:
све зависне променљиве су 0
 $x_1 = x_2 = 0$ следи $Z = 0$

Dantzig алгоритам за проблем интернет провајдера

Полазимо из тачке $(x_1, x_2) = (0,0)$, на слици је то тачка O , а $Z = 0$.

Основне променљиве су оне које чине јединичну субматрицу, s_1, s_2, s_3 .

$$\begin{bmatrix} 1 & -3 & -2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 20 \\ 10 \\ 15 \end{bmatrix}$$

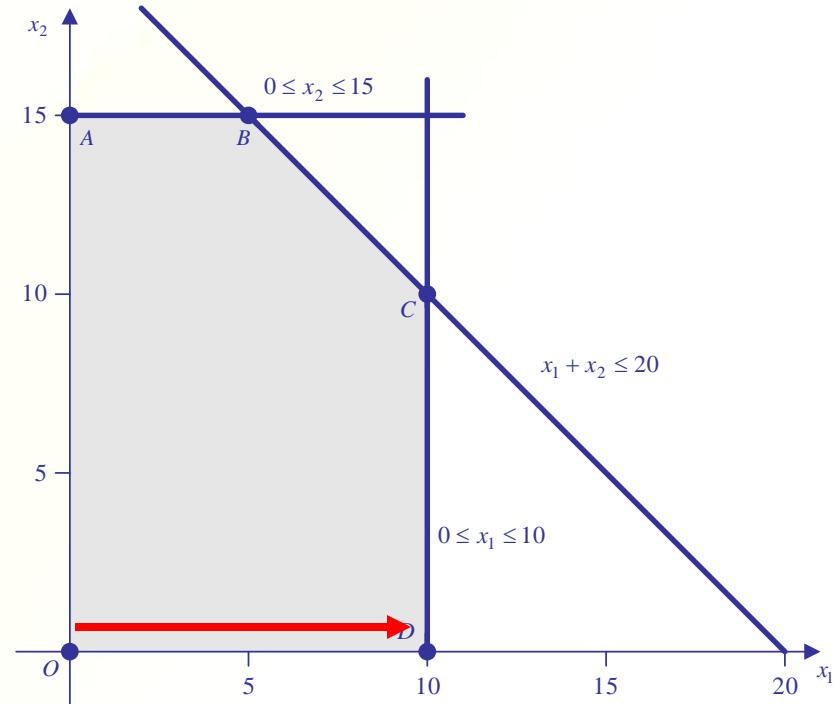
Основне променљиве (оне које чине јединичну субматрицу)
називају се базисни вектор

Пивотизација (основни корак алгоритма)

- Пронаћи зависну променљиву са позитивним коефицијентом у Z (повећање те променљиве повећава Z)
- Повећати променљиву максимално могуће на основу услова који постоје
- Пронаћи најстрожи услов (најмање повећање)
- Заменити основну и зависну променљиву

Испитивање за први корак

- $Z = 3x_1 + 2x_2$
 $x_1 + x_2 + s_1 = 20$
 $x_1 + s_2 = 10$
 $x_2 + s_3 = 15$
- x_1 има позитивни коефицијент
 $x_1 + 0 + 0 = 20 \rightarrow x_{1\max} = 20$
 $x_1 + 0 = 10 \rightarrow x_{1\max} = 10$
 $x_2 + s_3 = 15$
- Најстрожи услов $x_{1\max} = 10$
- $Z = 3(10 - s_2) + 2x_2 = 30 - 3s_2 + 2x_2$
 $- s_2 + x_2 + s_1 = 10$
 $x_1 = 10 - s_2$
 $x_2 + s_3 = 15$
- Ново решење $(x_1, x_2) = (10, 0)$, $Z_{\max} = 30$



Еквивалентне матричне манипулације

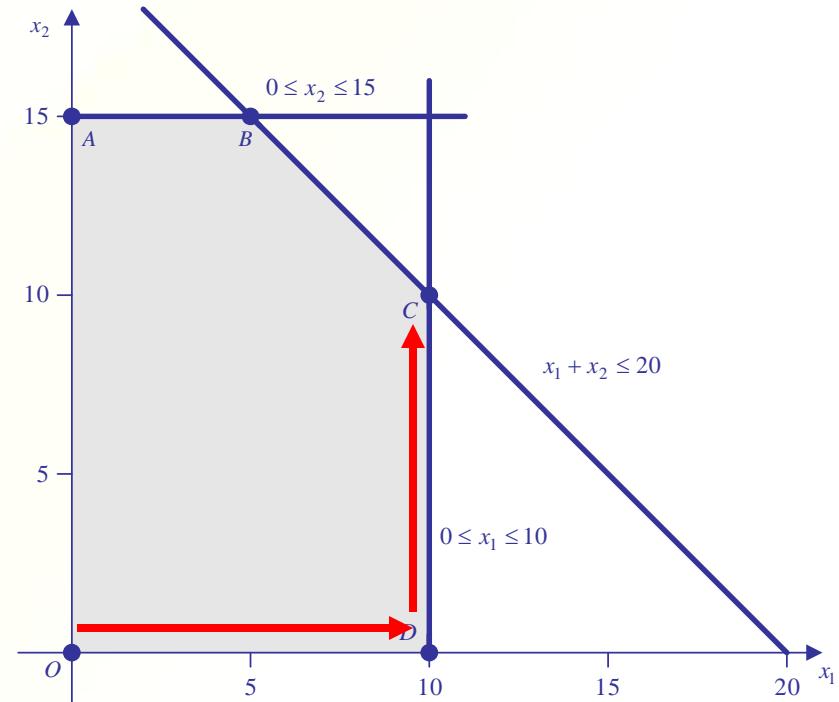
$$\begin{array}{c}
 \xrightarrow[3]{+} \\
 \left[\begin{array}{cccccc} 1 & -3 & -2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{c} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{array} \right] = \left[\begin{array}{c} 0 \\ 20 \\ 10 \\ 15 \end{array} \right] \rightarrow \left[\begin{array}{cccccc} 1 & 0 & -2 & 0 & 3 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{c} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{array} \right] = \left[\begin{array}{c} 30 \\ 20 \\ 10 \\ 15 \end{array} \right] \xleftarrow[-1]{+} \\
 \end{array}$$

$$\rightarrow \left[\begin{array}{cccccc} 1 & 0 & -2 & 0 & 3 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{c} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{array} \right] = \left[\begin{array}{c} 30 \\ 10 \\ 10 \\ 15 \end{array} \right]$$

- Табло се типично пише без вектора $[Z \ x \ s]^T$ и знака једнакости
- Они су овде написан ради јасног сагледавања везе са полазним системом

Испитивање за наредни корак

- $Z = 3(10-s_2) + 2x_2 = 30 - 3s_2 + 2x_2$
 $-s_2 + x_2 + s_1 = 10$
 $x_1 = 10 - s_2$
 $x_2 + s_3 = 15$
- x_2 има позитивни коефицијент
 $-0 + x_2 + 0 = 10 \rightarrow x_{2\max} = 10$
 $x_1 = 10 - s_2$
 $x_2 + s_3 = 15 \rightarrow x_{2\max} = 15$
Најстрожи услов $x_{2\max} = 10$
- $Z = 50 - 2s_1 - s_2$
 $x_2 = 10 + s_2 - s_1$
 $x_1 = 10 - s_2$
 $-s_1 + s_2 + s_3 = 5$
- Ново решење $(x_1, x_2) = (10, 10)$, $Z_{\max} = 50$



Еквивалентне матричне манипулације

$$\begin{array}{c}
 \text{+} \\
 \text{2} \times \text{C}_1 \rightarrow \\
 \left[\begin{array}{cccccc} 1 & 0 & -2 & 0 & 3 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \begin{bmatrix} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 30 \\ 10 \\ 10 \\ 15 \end{bmatrix} \rightarrow \begin{array}{l} \left[\begin{array}{cccccc} 1 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \begin{bmatrix} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 50 \\ 10 \\ 10 \\ 15 \end{bmatrix} \\ -1 \times \text{C}_2 \\
 \end{array}
 \end{array}$$

$$\rightarrow \left[\begin{array}{cccccc} 1 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 \end{array} \right] \begin{bmatrix} Z \\ x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 50 \\ 10 \\ 10 \\ 5 \end{bmatrix}$$

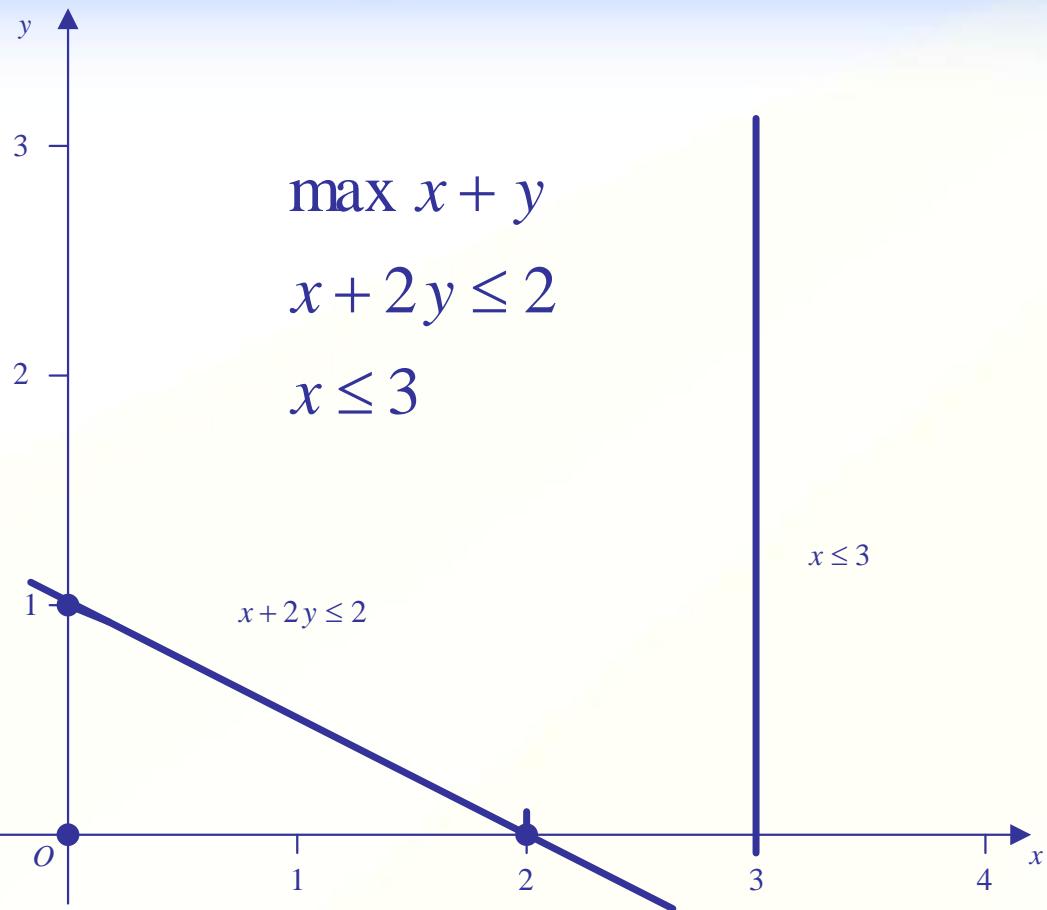
Анализа наредног корака

- $Z = 50 - 2s_1 - s_2$
- Нема више променљивих које имају позитивни коефицијент у Z
- Нема могућности за даље повећање Z
- Решење је пронађено
- Крај алгоритма

Генерализација

- Уколико су чланови са десне стране услова b_i ненегативни, постоји решење
- Како организовати алгоритам уколико су b_i **негативни**?
- Описани поступак назива се “фаза 2” симплекс алгоритма
- Претпроцесирање (“фаза 1”) своди све друге LP проблеме у облик који се може решити “фазом 2”
- Потребно је пронаћи базисни вектор
 - додају се “вештачке променљиве”
 - pivotизацијом се решава “додатни проблем” који даје формулацију полезног проблема који се може решити “фазом 1”
- Може се догодити да полазни проблем
 - нема решење
 - оптимизациона функција (max) није ограничена са горње стране
 - ...
- Претпроцесирање је сложено у општем случају

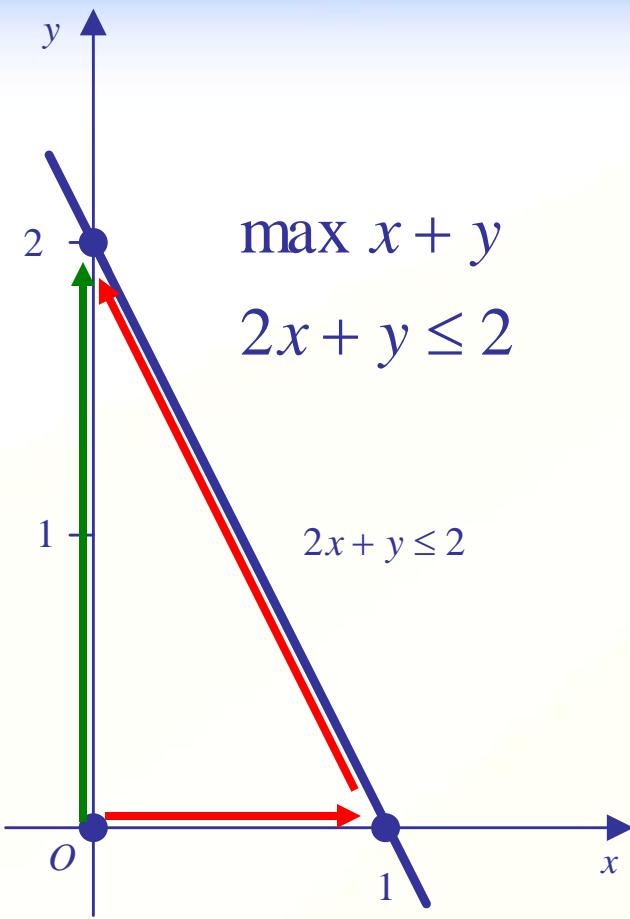
Генерализација: Сувишни услов



$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 & 1 & 3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 2 \\ 0 & 1 & 2 & 1 & 0 & 2 \\ 0 & 0 & -2 & -1 & 1 & 1 \end{bmatrix}$$

- Појављује се негативни коефицијент
- Прескочити такав услов

Ефикасност: Избор наредног корака



$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 2 & 1 & 1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & -0,5 & 0,5 & 1 \\ 0 & 2 & 1 & 1 & 2 \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 2 \\ 0 & 2 & 1 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 2 & 1 & 1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 0 & 1 & 2 \\ 0 & 2 & 1 & 1 & 2 \end{bmatrix}$$

- У првом случају **два корака** у другом случају **један корак**
- Од избора наредне тачке зависи ефикасност

Транспортни проблем (услови су искључиво једнакости)

- Компанија производи штампаче у две фабрике F1 и F2 на различитим местима.
- Поред тога, има три продајна објекта C1, C2 и C3.
- Месечни захтеви објеката (у пакетима од по 10 штампача) су 8, 5 и 2.
- Фабрике F1 и F2 могу да произведу 6 и 9 пакета месечно, редом.
- Преносе се искључиво пакети (није могуће дељење пакета на мање јединице).
- Приметити да је укупна производња једнака укупној потражњи.
- Транспорт из фабрика до продајних објеката има различиту цену због различитих места на којима се они налазе.
- Цене транспорта су дате у табели и изражене су у хиљадама динара.
- Пронаћи план испоруке (колико пакета иде у који продајни центар и из које фабрике) тако да трошкови транспорта буду минимални.
- Израчунати минималне трошкове транспорта у том случају.

Цене преноса из Fx у Cy

	C1	C2	C3
F1	5	5	3
F2	6	4	1

Формулација

- Трошкови испоруке су

$$z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$$

где је x_{ij} број пакета који иде из фабрике i у објекат j ($i \in \{1,2\}$ и $j \in \{1,2,3\}$)

- Потребно је поронаћи све x_{ij} тако да функција z има минималну вредност, при задатим условима
- x_{ij} мора бити цео број и ненегативан ($x_{ij} \geq 0$)

Формулација са условима

- $\min z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$
- Услови
 - $x_{11} + x_{21} = 8$
 - $x_{12} + x_{22} = 5$
 - $x_{13} + x_{23} = 2$
 - $x_{11} + x_{12} + x_{13} = 6$
 - $x_{21} + x_{22} + x_{23} = 9$
- Сви услови су једнакости!
Нема помоћних променљивих

Једноставно решење елиминацијом променљивих

- Сваку једнакост можемо искористити да елиминишемо по једну променљиву

Елиминација x_{11}

$$\min z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23} \Rightarrow z = 40 + 5x_{12} + 3x_{13} + x_{21} + 4x_{22} + x_{23}$$

$$x_{11} + x_{21} = 8 \Rightarrow x_{11} = 8 - x_{21}$$

$$x_{12} + x_{22} = 5$$

$$x_{13} + x_{23} = 2$$

$$x_{11} + x_{12} + x_{13} = 6 \Rightarrow -x_{21} + x_{12} + x_{13} = -2$$

$$x_{21} + x_{22} + x_{23} = 9$$

Наредни кораци елиминације (x_{12})

- Жуто означене једначине су већ искоришћене

Елиминација x_{12}

$$z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23} \Rightarrow z = 65 + 3x_{13} + x_{21} - x_{22} + x_{23}$$

$$x_{11} + x_{21} = 8 \Rightarrow x_{11} = 8 - x_{21}$$

$$x_{12} + x_{22} = 5 \Rightarrow x_{12} = 5 - x_{22}$$

$$x_{13} + x_{23} = 2$$

$$x_{11} + x_{12} + x_{13} = 6 \Rightarrow -x_{21} + x_{12} + x_{13} = -2 \Rightarrow -x_{21} - x_{22} + x_{13} = -7$$

$$x_{21} + x_{22} + x_{23} = 9$$

Наредни кораци елиминације (x_{13})

- Жуто означене једначине су већ искоришћене

Елиминација x_{13}

$$z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23} \Rightarrow z = 71 + x_{21} - x_{22} - 2x_{23}$$

$$x_{11} + x_{21} = 8 \Rightarrow x_{11} = 8 - x_{21}$$

$$x_{12} + x_{22} = 5 \Rightarrow x_{12} = 5 - x_{22}$$

$$x_{13} + x_{23} = 2 \Rightarrow x_{13} = 2 - x_{23}$$

$$x_{11} + x_{12} + x_{13} = 6 \Rightarrow -x_{21} + x_{12} + x_{13} = -2 \Rightarrow -x_{21} - x_{22} - x_{23} = -9$$

$$x_{21} + x_{22} + x_{23} = 9$$

Последње две једначине су идентичне!

Наредни кораци елиминације (x_{21})

- Жуто означене једначине су већ искоришћене

Елиминација x_{21}

$$z = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23} \Rightarrow z = 80 - 2x_{22} - 3x_{23}$$

$$x_{11} + x_{21} = 8 \Rightarrow x_{11} = 8 - x_{21}$$

$$x_{12} + x_{22} = 5 \Rightarrow x_{12} = 5 - x_{22}$$

$$x_{13} + x_{23} = 2 \Rightarrow x_{13} = 2 - x_{23}$$

$$x_{11} + x_{12} + x_{13} = 6 \Rightarrow -x_{21} + x_{12} + x_{13} = -2 \Rightarrow -x_{21} - x_{22} - x_{23} = -9$$

$$x_{21} + x_{22} + x_{23} = 9 \Rightarrow x_{21} = 9 - x_{22} - x_{23}$$

Решење

- После ове елиминације остаје само
$$z = 80 - 2x_{22} - 3x_{23}$$
- Ову функцију је потребно минимизовати, што се постиже узимањем максималних вредности за $x_{22} = 5$ и $x_{23} = 2$
- Све остале променљиве добијају се из једнакости које су искоришћене за њихово "елиминисање"
- Коначно $z_{\min} = 64$,
 $(x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}) = (6,0,0,2,5,2)$
- У општем случају решавања LP проблема са једнакостима постоје две могућности
 - елиминисање једне променљиве једном једначином
 - увођење вештачке (eng: artificial) променљиве тада се може применити Dantzig алгоритам, али са обе фазе

Основне чињенице о Dantzig simplex алгоритму

- Најчешће коришћен алгоритам у пракси за LP оптимизацију
- У зависности од полазног решења, могуће је обићи све екстреме да би се стигло до најбољег (што је еквивалентно систематском претраживању)
- Време потребно за претраживање је **експоненцијално** (у најнеповољнијим случајевима, постоји строг доказ)
- Без обзира на то, овај алгоритам добро ради у пракси
- Сложеност
 - у пракси је типично $O(N^3)$,
 - теоријски експоненцијална (обилазак свих екстремних тачака, могу се конструисати примери где алгоритам мора да прође све екстреме пре него што пронађе решење)

Имплементације Dantzig симплекс алгоритма

- MATLAB:
 - linprog,
 - intlinprog
- Mathematica:
 - LinearProgramming
- Numerical Recipes In C: The Art Of Scientific Computing: `void simplex(...)`
- Python: `scipy.optimize.linprog`

Други алгоритми за LP

- Criss-cross algorithm
нешто једноставнији од Dantzig simplex
- Fourier–Motzkin elimination
- Karmarkar's algorithm
један од ретких алгоритама који је био патентиран,
прилично компликован
(перформансе у пракси нису боље од Dantzig симплекса)
- Методи унутрашње тачке (interior-point methods)
проналазе решење крећући се по
унутрашњости симплекса
(за разлику од Dantzig алгоритма који иде по ивицама!)
- Сви алгоритми који решавају NLP проблеме могу да се
примене (при томе, некада је потребно променити
оптимизациону функцију $f_0 = -\|f\|_p + \text{const.}$)

Задатак за вежбе

- Потребно је поставити сервере у ормаре тако да се **максимизира** рачунски капацитет система који користи све сервере у ормарима
- Сваки од 3 ормара има ограничење по броју сервера и укупној снаги сервера
- На располагању стоји различит број сервера, за сваки од 4 типа
- Одредити распоред сервера по ормарима и израчунати максимални остварени рачунски капацитет система

	Максималан број сервера	Максимална снага [W]
Ормар 1	10	6800
Ормар 2	16	8700
Ормар 3	8	4300

	Укупан број сервера на располагању	Снага појединачног сервера [W]	Рачунски капацитет појединачног сервера [GFlops]
Сервер 1	18	480	310
Сервер 2	15	650	380
Сервер 3	23	580	350
Сервер 4	12	390	285

Поступак решавања

- Написати оптимизациону функцију и задате услове у стандардном LP облику
 - Непознате су бројеви сервера постављени у ормаре x_{rs} , где је r редни број ормара (1,2,3), а s редни број типа сервера (1,2,3,4)
- Написати програм који решава задати проблем
 - Користити Python функцију за линеарно програмирање или написати одговарајући код
 - Водити рачуна да су променљиве цели (ненегативни) бројеви! (уколико се за решење x_{rs} добије број који није цео, а различит од нуле, проверити и три мања и три већа цела броја)
- ASCII фајл уз решење би требало да садржи:
 - Оптималан распоред сервера по ормарима записан у облику $(x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}, x_{31}, x_{32}, x_{33}, x_{34})$
 - Максималан остварени рачунски капацитет система
 - Уколико се за променљиве прво добију нецелобројна решења, записати и то решење у истом облику са одговарајућим максималним рачунским капацитетом у том случају

Динамичко “програмирање”

- Richard Bellman "The theory of dynamic programming" *Bulletin of the American Mathematical Society* 60 (6): 503–516 1954.
- Richard Bellman *Dynamic Programming* Princeton University Press 1957.
- D. P. Bertsekas *Dynamic Programming and Optimal Control* (4th ed.) Athena Scientific 2017.
- John von Neumann Harold William Kuhn *Theory of Games and Economic Behavior: 60th Anniversary Commemorative Edition* 2007.

Основне идеје

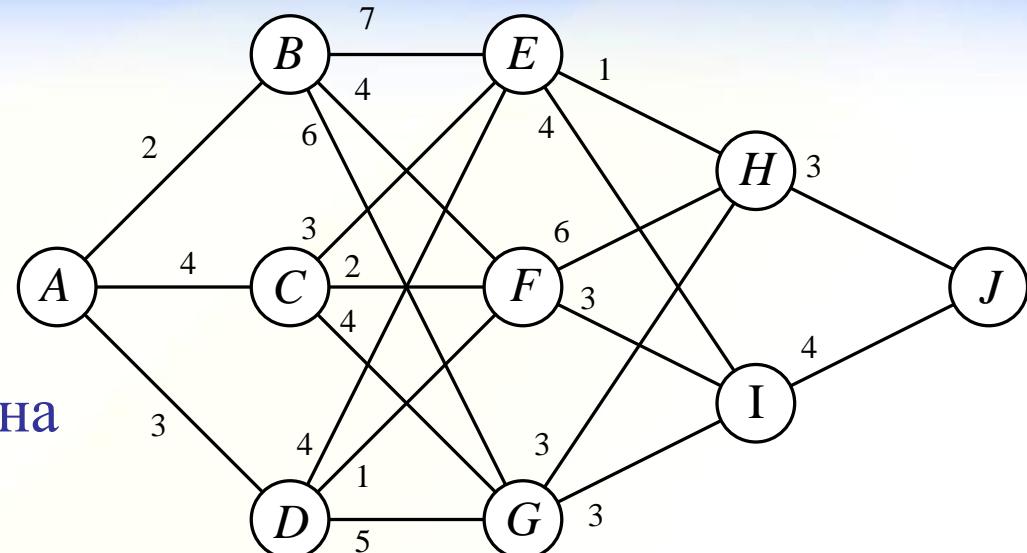
- Динамичко програмирање је техника за проналажење оптималне комбинације догађаја који су међусобно повезани
- За разлику од линеарног програмирања не постоји стандардна математичка формулатија динамичког програмирања
- Постоји принцип који се модификује према конкретном оптимизационом проблему

Илустрација динамичког програмирања: пример

- Табеларни приказ са вредностима прелаза
- Вредност прелаза из A у B је 2 итд.
- Вредност може бити: цена растојање време итд.
- Циљ је пронаћи путању од A до J са најмањом сумом вредности

	B	C	D
A	2	4	3

	E	F	G
B	7	4	6
C	3	2	4
D	4	1	5



	H	I
E	1	4
F	6	3
G	3	3

	J
H	3
I	4

Формални запис проблема

- Сматраћемо да постоји 4 оптимизационе променљиве које описују путању

$$A \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$$

- Могуће вредности оптимизационих променљивих су

$$x_0 \in \{A\} \quad x_1 \in \{B, C, D\} \quad x_2 \in \{E, F, G\} \quad x_3 \in \{H, J\} \quad x_4 \in \{J\}$$

- Оптимизациони простор је дискретан и има $3 \times 3 \times 2 = 18$ могућих решења

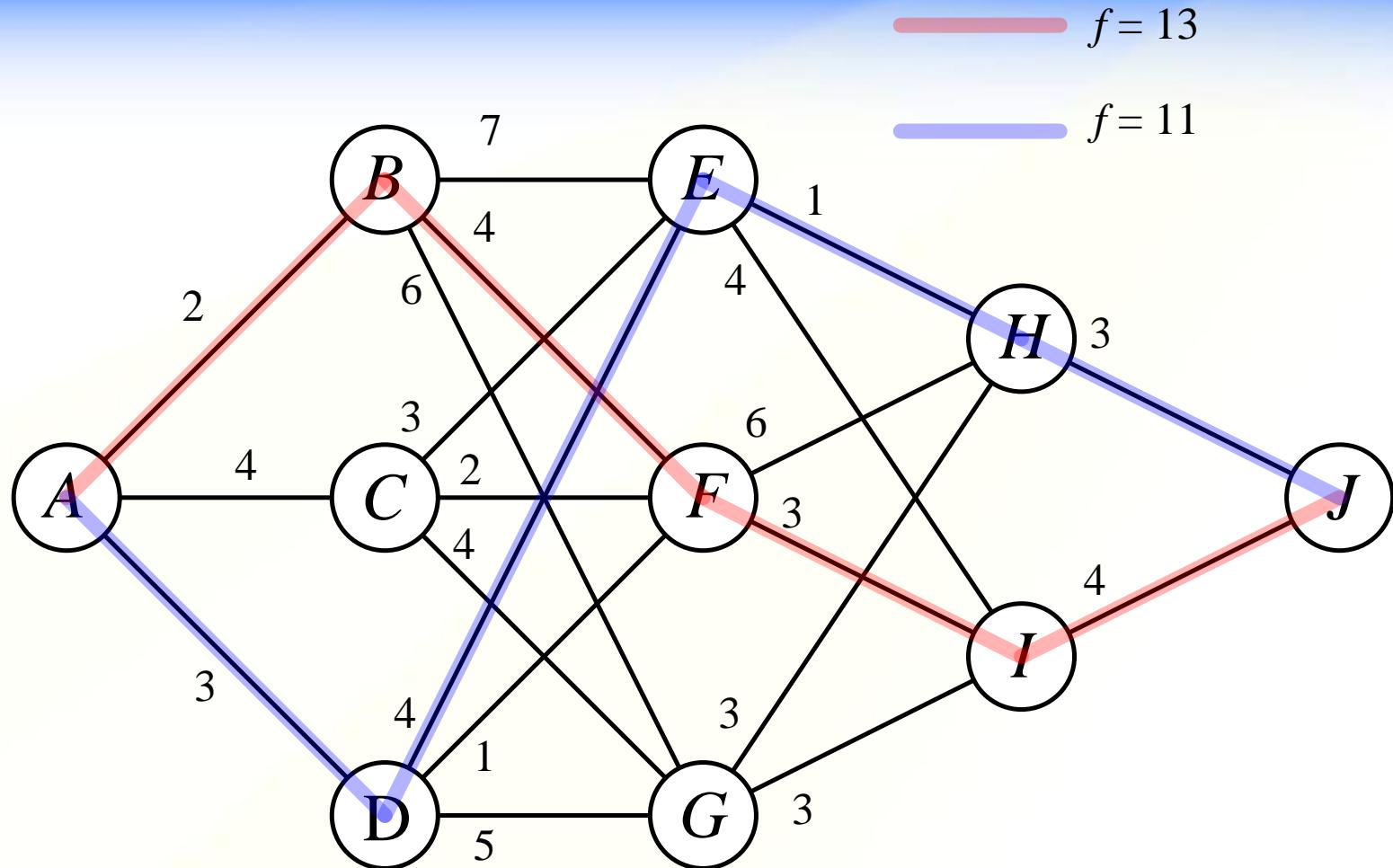
Оптимизациона функција

- Оптимизациона функција је

$$f(\mathbf{x}) = \sum_{k=1}^4 d(x_{k-1}, x_k)$$

- $d(a,b)$ је вредност путање од a до b и $x_0=A$
- Вредност поједињих путања је задата табеларно или на неки други начин
- Нема ограничења у погледу линеарности/нелинеарности оптимизационе функције

Хеуристичко решење проблема



Хеуристичко решење и потпуна претрага

- Једноставан приступ – у сваком кораку изабрати прелаз са најмањом вредношћу (енг: greedy тј. локални алгоритам)
 - Тако добијено решење има оптимизациону функцију 13
- Најбоље решење можемо да нађемо и потпуном претрагом
 - За оптимално решење вредност оптимизационе функције је 11

Динамичко програмирање: идеја

- Основна идеја је поделити проблем на мање потпроблеме
- Потпроблеми:
 - су међусобно повезани
 - решавају се одвојено
 - крајње решење се добија на основу решења појединачних потпроблема
- Сваки потпроблем одговара једној оптимизационој променљивој
(текућа променљива)
- Проблем се решава у корацима
- Кораци могу бити и дискретни одбирци времена
(одатле “динамичко” програмирање)

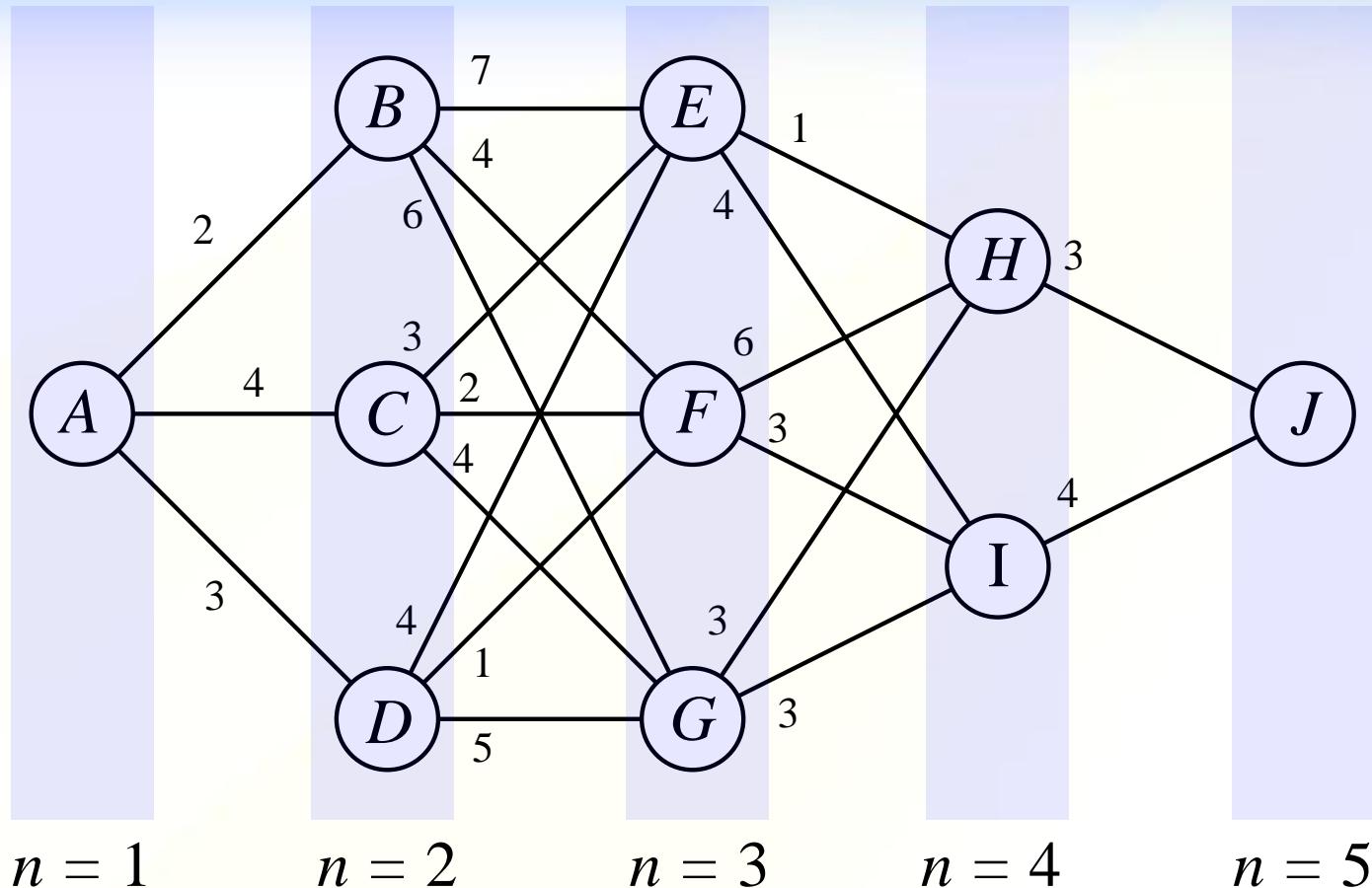
Формални запис динамичког програмирања

- Нека је текућа (оптимизациона) променљива s у једном кораку алгоритма
- Из s прелазимо у променљиву x_n
- Вредност оптимизационе функције за сва преостала стања услед поласка из s
 $f_n(s, x_n)$
- Оптимална вредност ове функције је
$$f_n^*(s) = \min_{x_n} f_n(s, x_n) = f_n(s, x_n^*)$$
- где x_n^* оптимизациона променљива x_n која минимизира $f_n(s, x_n)$

Рекурзивни приказ оптимизационе променљиве

- Оптимизациона функција се може приказати као
$$f_n(s, x_n) = d(s, x_n) + f_{n+1}^*(x_n)$$
- $d(s, x_n)$ је повећање функције у текућем кораку
- Минимална оптимизациона функција за сва наредна стања (од стања x_n па надаље) је $f_{n+1}^*(x_n)$
- Када смо стигли на крај $f_5^*(J) = 0$
- Циљ је пронаћи $f_1^*(A)$
- Оптимална путања се проналази сукцесивним одређивањем $f_4^*(s), f_3^*(s), f_2^*(s)$

Графички приказ корака



Алгоритам динамичког програмирања

- Полазимо из претпоследњег стања када постоји још само један избор

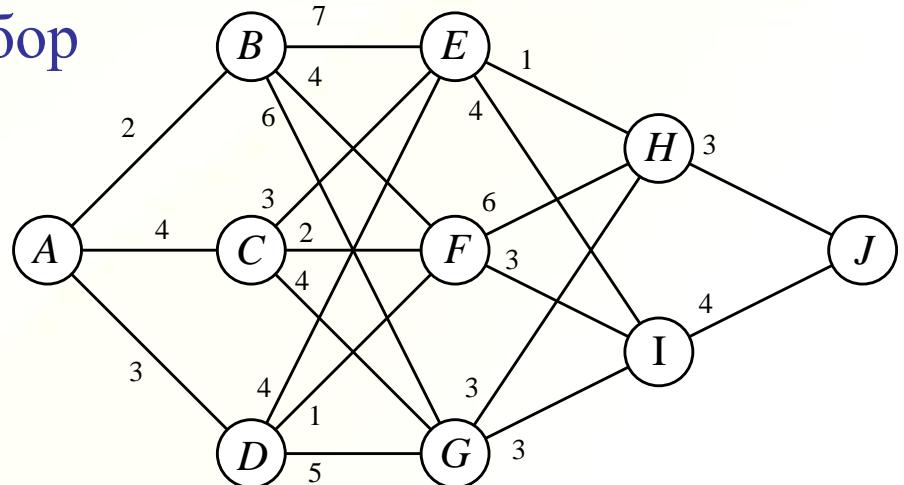
$$n = 4$$

$$f_4^*(s) = f_4(s, x_4) = f_4(s, J) = d(s, J)$$

s	$f_4^*(s)$	x_4^*
H	3	J
I	4	J

$$n = 3$$

$$f_3(s) = d(s, x_3) + f_4^*(x_3)$$



s	$f_3(s) = d(s, x_3) + f_4^*(x_3)$		$f_3^*(s)$	x_3^*
	H	I		
E	$1+3=4$	$4+4=8$	4	H
F	$6+3=9$	$3+4=7$	7	I
G	$3+3=6$	$3+4=7$	6	H

Алгоритам динамичког програмирања

- Настављамо алгоритам преласком у претходне кораке

$n = 2$

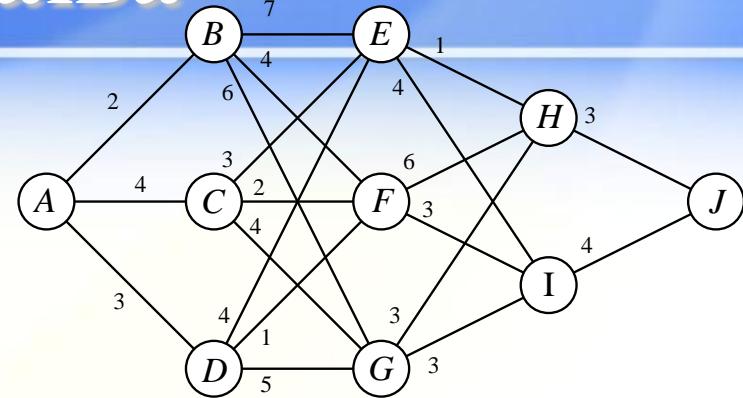
$$f_2(s) = d(s, x_2) + f_3^*(x_2)$$

	$f_2(s) = d(s, x_2) + f_3^*(x_2)$			$f_2^*(s)$	x_2^*
s	E	F	G		
B	$7+4=11$	$4+7=11$	$6+6=12$	11	E/F
C	$3+4=7$	$2+7=9$	$4+6=10$	7	E
D	$4+4=8$	$1+7=8$	$5+6=11$	8	E/F

$n = 1$

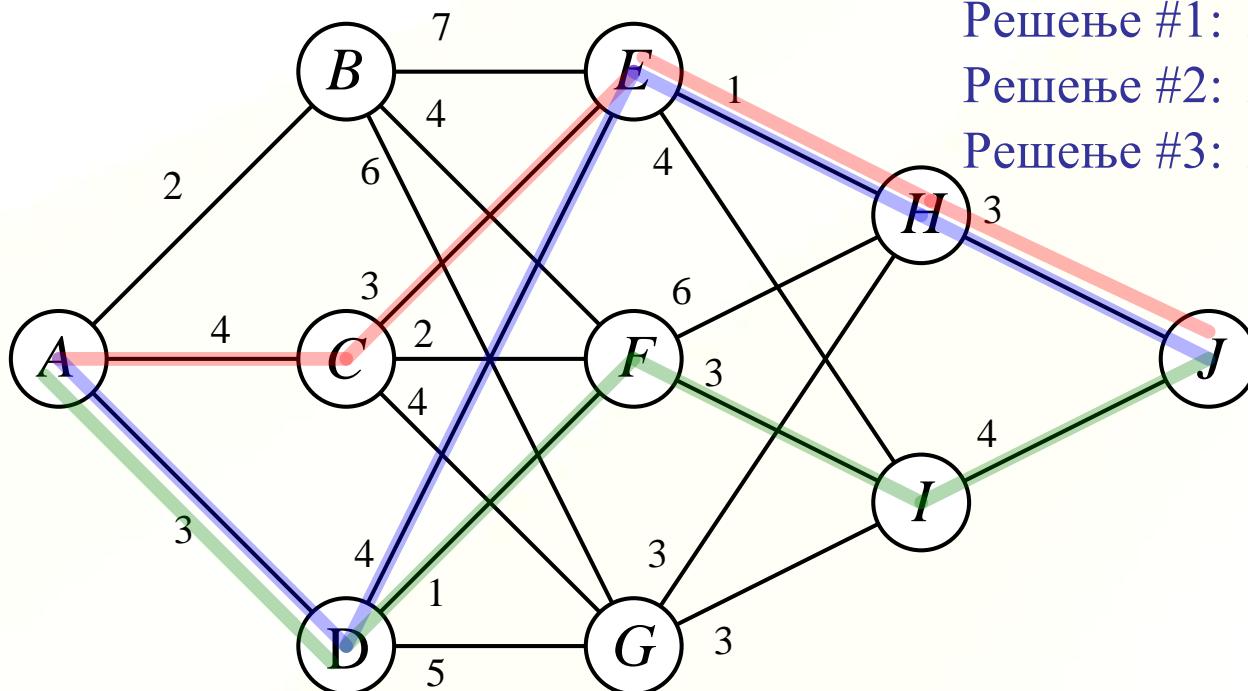
$$f_1(s) = d(s, x_1) + f_2^*(x_1)$$

	$f_1(s) = d(s, x_1) + f_2^*(x_1)$			$f_1^*(s)$	x_1^*
s	B	C	D		
A	$2+11=13$	$4+7=11$	$3+8=11$	11	C/D



Оптимална путања се прочита из табела

- Тиме је решен проблем $f_1^*(A) = 11$
- Оптималне путање проналазимо из табела



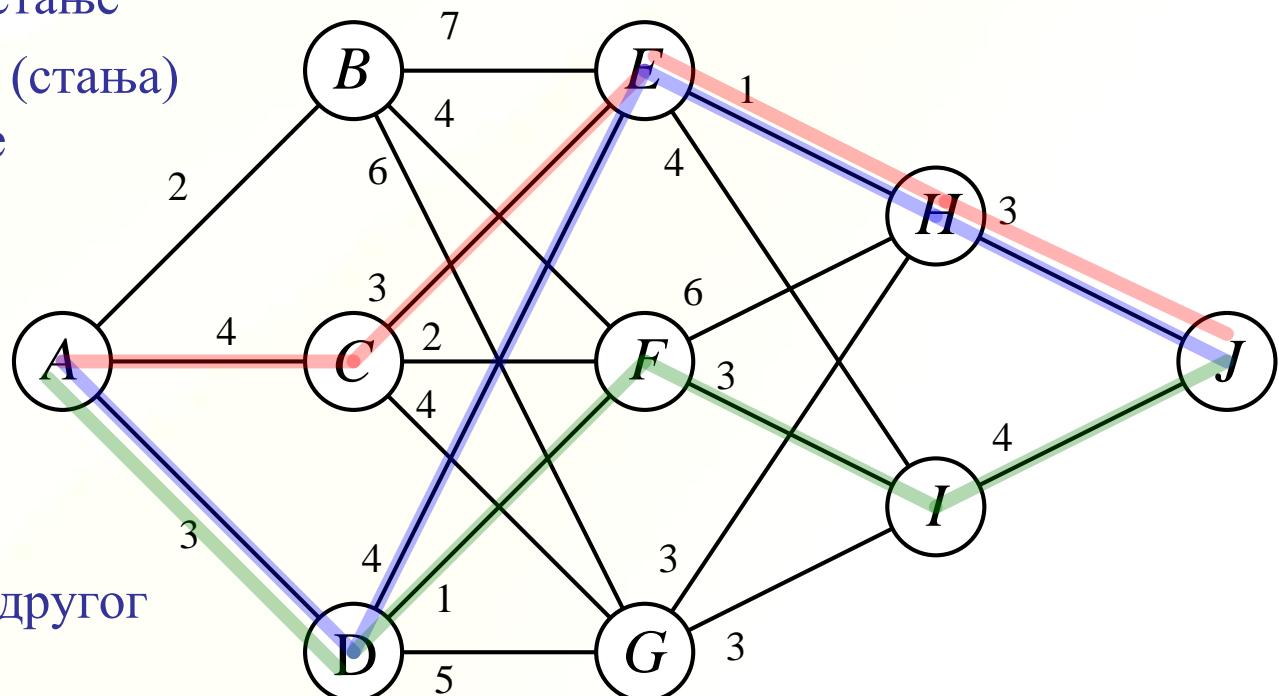
Решење #1: $A \rightarrow C \rightarrow E \rightarrow H \rightarrow J$.

Решење #2: $A \rightarrow D \rightarrow E \rightarrow H \rightarrow J$.

Решење #3: $A \rightarrow D \rightarrow F \rightarrow I \rightarrow J$.

Белманов принцип оптималности

- Оптимални избор корака за читав проблем има особину да без обзира на почетан избор избор преосталих корака мора бити оптималан у односу на текуће стање
- Из било ког чвора (стања) оптималне путање да кренемо преостали пут је оптималан!
- Пример:
пут $D-F-I-J$ је бољи од било ког другог пута $D-\dots-J$



Број операција

- Уколико се проблем решава потпуном претрагом потребно је проверити 18 путања а за сваку извршити 4 сабирања (72 операције)
- Чувањем табеле за сваки корак решили смо проблем са 21 рачунском операцијом и 9 провера (онолико колико има укупно редова у табелама)
- Пример: проблем са 10 корака 10 стања по кораку и 10 могућности по стању
 - Број провера би био 10^{10} уз памћење само најбољег пронађеног решења
 - Динамичким програмирањем потребно је 10^3 провера уз памћење одговарајућих табела

Каррактеристике проблема динамичког програмирања

- Проблем се може поделити на **потпроблеме који се решавају сукцесивно**
- Сваки потпроблем има **коначан број избора** који се могу направити
- Сваки чвор графа одговара једном могућем стању. Колона чворова одговара свим могућим стањима у једном кораку. (Проблем динамичког програмирања може се приказати као граф у којем скуп чворова припада једном кораку).
- Тражимо **оптималну путању кроз граф**
- У текућем стању **оптимални избор наредног корака** зависи само од преосталих стања а **не зависи од претходних стања** (Белманов принцип оптималности)
- Први корак је одређивање оптималног последњег (могућег) избора
- **Постоји рекурзивна релација** којом се одређује оптимални избор у текућем кораку уколико је познат оптимални избор за све наредне кораке
- До решења се долази одређивањем оптималног избора полазећи од последњег корака преко претпоследњег и тако до првог корака

Примери алгоритама који користе динамичко програмирање

- **Dijkstra's algorithm** (проналажење најкраћег пута у дискретном простору)
- **Viterbi algorithm** (проналажење највероватније секвенце скривеног процеса)
- **Bellman–Held–Karp algorithm** (проналажење оптималне путање TSP проблема динамичким програмирањем)

[Бонус: решити проблем путање бургије са СВИХ 20 рупа коришћењем динамичког програмирања
Bellman–Held–Karp алгоритам]

Закључци о динамичком програмирању

- Број провера и рачунских операција се смањује на рачун памћења претходно решених потпроблема (табела)
- Код великих проблема
меморијски ресурси постају критични!
- Решавање се може организовати генерално на два начина
 - од kraja ka почетку (енглески: top-down) или
 - од почетка ka kraju (енглески: bottom-up)
- Динамичко програмирање = рекурзија + “здрава памет”
 - рекурзија: изражавамо текућу вредност преко претходних
 - “здрава памет”: организујемо поступак тако да се памте све вредности које су већ једном одређене

Алгоритми који проналазе локални минимум (NLP)

- Алгоритми које смо до сада радили или
 - захтевају недопустиво много времена да пронађу најбоље решење
 - заглаве се у локалном оптимуму
- Немогуће је направити алгоритам који у кратком времену проналази глобални оптимум у општем случају
- Практичан приступ: итеративни hill-climbing
 - када се пронађе један оптимум алгоритам се поново покрене из другог почетног решења

Симулирано каљење

- Симулирано каљење
 - Монте Карло каљење (simulated annealing
Monte Carlo annealing statistical cooling
probabilistic hill-climbing stochastic relaxation
probabilistic exchange algorithm)
- S. Kirkpatrick, C. D. Gelatt, Jr. M. P. Vecchi
“Optimization by Simulated Annealing”
Science, Vol. 220, May 1983, pp. 671-680.
- Аналогија са термодинамиком

Аналогија између термодинамике и оптимизације

- Приликом хлађења загрејаног система честица систем тежи да заузме стабилно стање које представља минимум у простору могућих енергетских стања система
- Овај поступак се примењује током каљења челика ради добијања бољих физичких карактеристика материјала одакле потиче назив алгоритма
- Енергија система аналогна је оптимизационој функцији а свако енергетско стање у којем се налази систем одговара једној тачки оптимизационог простора



Основни кораци симулираног каљења

- Генерирање наредне тачке (у околини која се претражује)
- Прелазак у наредну тачку (прелазак се одиграва стохастички)
- Формирање следеће температуре (хлађење) и промена околине у којој се генерише наредна тачка



1. Генерисање наредне тачке

- Наредна тачка у оптимизационом простору се бира на **случајан начин** из околине текућег решења
- Најчешће се користе генератори
 - унiformне или
 - Гаусове расподеле случајних бројева
- Током хлађења запремина околине у којој се генеришу наредне тачке се сужава и теоријски на крају оптимизације постаје нула
 - NLP околина сфера полупречника R
 - SAT околина Хамингово растојање
 - TSP околина број градова којима заменимо места

2. Прелазак у наредну тачку

- Наредна тачка се приhvата тј. постаје текућа тачка оптимизације у зависности од прираштаја функције грешке
- Уколико је прираштај опт. функције ΔE негативан тј. опт. функција је мања у наредној тачки тачка нужно постаје текућа тачка оптимизације
- **Међутим дозвољени су и преласци у тачке у којима је ΔE позитивно!**

2. Прелазак у наредну тачку: пораст функције грешке

- Уколико је прираштај опт. функције ΔE позитиван наредна тачка се приhvата са вероватноћом p
$$p = \exp\left(-\frac{\Delta E}{T(k)}\right)$$
- $T(k)$ је “температура” у k -том кораку алгоритма
- На овај начин алгоритам не бива у просеку заглављен у локалне минимуме већ проналази глобални минимум

Температура $T(k)$: границне вредности

- Вероватноћа преласка одговара Болцмановој расподели те се овај алгоритам назива и Болцманово хлађење
- Уколико је $T \gg \Delta E$ тада је $p \approx 1$ сви преласци су дозвољени алгоритам се понаша као случајно претраживање
- Уколико је $T \ll \Delta E$ тада је $p \approx 0$ само преласци у боље решења су дозвољени тј. понаша се као локални оптимизациони алгоритам

Температура $T(k)$: табеларни приказ за $\Delta E > 0$

$T(k)$	$p = \exp\left(-\frac{\Delta E}{T(k)}\right)$
$10^{-6} \cdot \Delta E$	0
$0,1 \cdot \Delta E$	0,000045
$1 \cdot \Delta E$	0,37
$5 \cdot \Delta E$	0,82
$10 \cdot \Delta E$	0,90
$100 \cdot \Delta E$	0,99
$1000 \cdot \Delta E$	0,999
$10^{+6} \cdot \Delta E$	1,0

- Без обзира на $T(k)$ уколико је $\Delta E = 0$ вероватноћа преласка је 0,5!

3. Формирање следеће температуре (хлађење)

- Температура у следећем кораку је увек мања или једнака од температуре у текућем кораку
- Следећа температура се формира по шеми хлађења
- Постоји неколико прихваћених шема хлађења од којих су две најчешће референциране

Промена температуре: логаритамско хлађење

- Логаритамско хлађење

$$T(k+1) = \frac{T_0}{\ln(k+2)}, \quad k \geq 1$$

- Постоји строг математички доказ да алгоритам конвергира ка глобалном минимуму али после бесконачно великог броја корака тј. када температура тежи нули
- Температура се врло споро смањује
- Потребан изузетно велики број итерација
- Овакво хлађење се у пракси ретко примењује

Промена температуре: експоненцијално хлађење

- Експоненцијално хлађење

$$T(k+1) = a \cdot T(k), \quad 0 < a < 1$$

- Овакав начин смањивања температуре се најчешће примењује у пракси
- Брже конвергира од логаритамског хлађења
- Даје задовољавајуће резултате
- Типично се узима $a \sim 0,95$
(или се израчуна имајући у виду T_0 жељену крајњу температуру и максималан број итерација)

Како изабрати почетну температуру T_0 ?

- Од избора T_0 зависе перформансе алгоритма
- Пример: оптимизациона функција $f(\mathbf{x})$
- Скалирана оптимизациона функција је суштински исти проблем (решења се налазе на истим местима у оптимизационом простору)
 - $f_1(\mathbf{x}) = 10 f(\mathbf{x})$
 - $f_2(\mathbf{x}) = 10^6 f(\mathbf{x})$
 - $f_3(\mathbf{x}) = 10^{-6} f(\mathbf{x})$
- Нумерички $\Delta E = \Delta f$ или $\Delta f_2 \gg \Delta f_1 \gg \Delta f_3$ ако је T_0 исто у свим случајевима вероватноћа $p(\Delta ET_0)$ се значајно разликује!
- **Добар избор** $T_0 \sim \Delta f_{\text{srednje}}$ израчунати f више пута у различитим тачкама и проценити $\Delta f_{\text{srednje}}$
$$f_{\text{sr}} = \frac{1}{N} \sum_{k=1}^N f(\mathbf{x}_k) \quad \Delta f_{\text{sr}} = \frac{1}{N} \sum_{k=1}^N |f(\mathbf{x}_k) - f_{\text{sr}}|$$
- **Лош избор** T_0 претвара симулирано каљење у случајно претраживање ($T_0 \gg \Delta f_{\text{srednje}}$) или у стохоатички локални оптимизациони алгоритам $T_0 \ll \Delta f_{\text{srednje}}$
- Избор T_0 зависи од оптимизационе функције!
- Неке имплементације користе случајно претраживање за подешавање T_0

Смањивање околине у којој се проверава наредно решење

- Током оптимизације “запремина” околине у којој се генерише и проверава наредно решење би требало да буде монотоно опадајућа функција
- Обично се користи линеарно смањивање са бројем итерација али може и било која друга (нерастућа) функција
- У примерима који следе сматраћемо да је околина функција броја итерација i је број текуће итерације а укупан (максималан дозвољен број итерација) биће означен са i_{tot}

Примери смањивања окoline

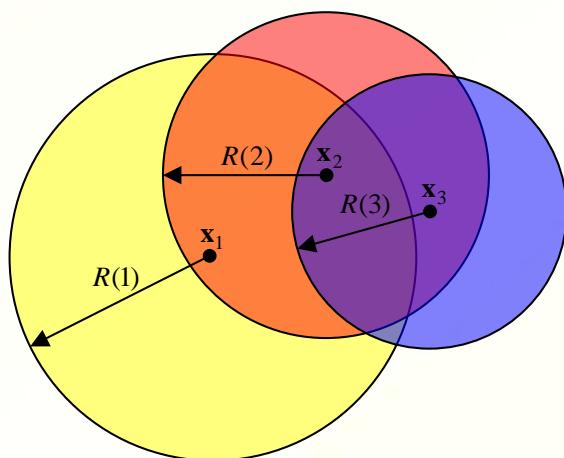
NLP:

$$R(i) = \|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2$$

$$R(i) = \frac{R_{\min} - R_{\max}}{i_{\text{tot}} - 1}(i - 1) + R_{\max}$$

$$i = 1 \Rightarrow R(i = 1) = R_{\max}$$

$$i = i_{\text{tot}} \Rightarrow R(i = i_{\text{tot}}) = R_{\min}$$



SAT:

$$h(i) = \text{hamming distance}(\mathbf{x}_{i+1}, \mathbf{x}_i)$$

$$h(i) = \frac{h_{\min} - h_{\max}}{i_{\text{tot}} - 1}(i - 1) + h_{\max}$$

$$i = 1 \Rightarrow h(i = 1) = h_{\max}$$

$$i = i_{\text{tot}} \Rightarrow h(i = i_{\text{tot}}) = h_{\min}$$

$$\mathbf{x}_1 = 10010110$$

$$h(1) = 5$$

$$\mathbf{x}_2 = 1\underline{1001101}$$

$$h(2) = 3$$

$$\mathbf{x}_3 = \underline{01011001}$$

TSP:

$$d(i) = \text{number_of_pairs}(\mathbf{x}_{i+1}, \mathbf{x}_i)$$

$$d(i) = \frac{d_{\min} - d_{\max}}{d_{\text{tot}} - 1}(i - 1) + d_{\max}$$

$$i = 1 \Rightarrow d(i = 1) = d_{\max}$$

$$i = i_{\text{tot}} \Rightarrow d(i = i_{\text{tot}}) = d_{\min}$$

$$\mathbf{x}_1 = (1, 2, 3, 4, 5, 6, 7, 8)$$

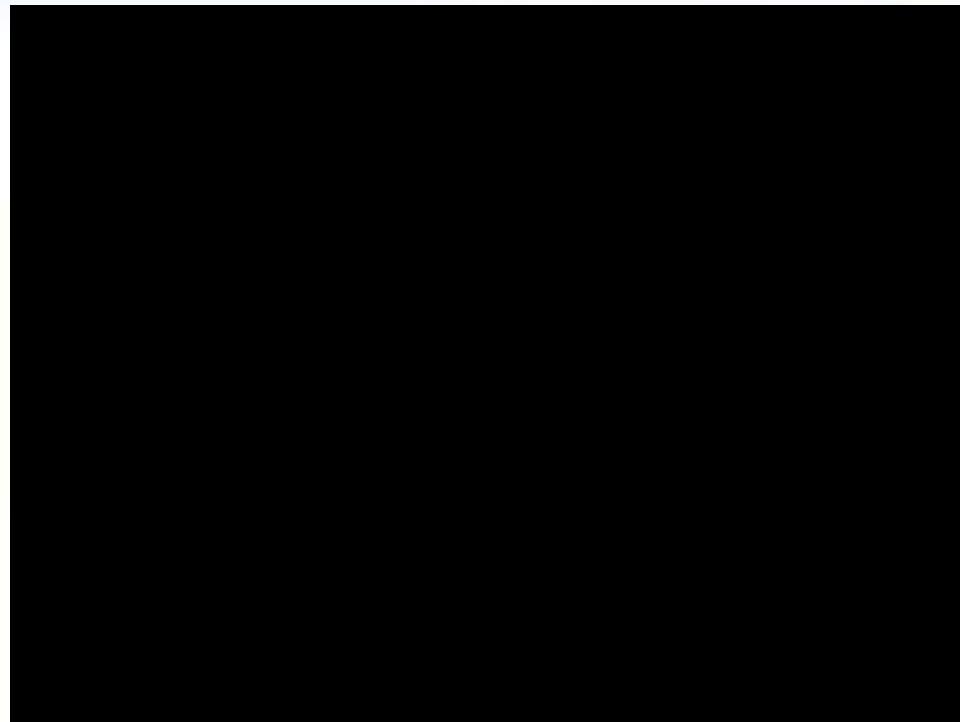
$$d(1) = 2$$

$$\mathbf{x}_2 = (\underline{3}, 2, \underline{1}, 4, \underline{8}, 6, 7, \underline{5})$$

$$d(2) = 1$$

$$\mathbf{x}_3 = (3, \underline{8}, 1, 4, \underline{2}, 6, 7, 5)$$

Ток симулираног каљења



Варијације симулираног каљења

- На једној температури је могуће урадити више итерација
- Понављање алгоритма из претходно добијеног минимума је још једна од честих модификација алгоритма
- Понављање се у литератури назива поновљено каљење (reannealing)

Закључци о симулираном каљењу

- Добре особине
 - први глобални оптимизациони алгоритам
 - постоји строг доказ да проналази глобални оптимум
- Критике
 - споро конвергира
 - потребан је врло велики број итерација
 - одабир температуре зависи од промене описне функције тј. од облика и вредности опт. функције!

Имплементације Симулираног каљења

- MATLAB: `simulannealbnd`
(Global optimization toolbox)
- Mathematica: `NMinimize[f vars`
`Method -> "SimulatedAnnealing"]`
- Numerical Recipes In C: The Art Of Scientific Computing: `void anneal (...)`
- Python:
 - `scipy.optimize.anneal`
 - `scipy.optimize.basinhopping`

Табу претраживање

- Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research* 13 (5): 533–549.
- Основна идеја слична као код симулираног каљења: дозвољен је прелаз у стање са лошијом описном функцијом
- Основна разлика:
детерминистички алгоритам

Основна идеја Табу претраживања: меморија

- Идеја: памте се претходна стања и њихове околине се проглашавају за забрањене (табу) зоне
- Уколико се алгоритам дуже времена задржи у локалном минимуму тај део простора се прогласи за табу
- Дефинисање меморије?

Различите меморије

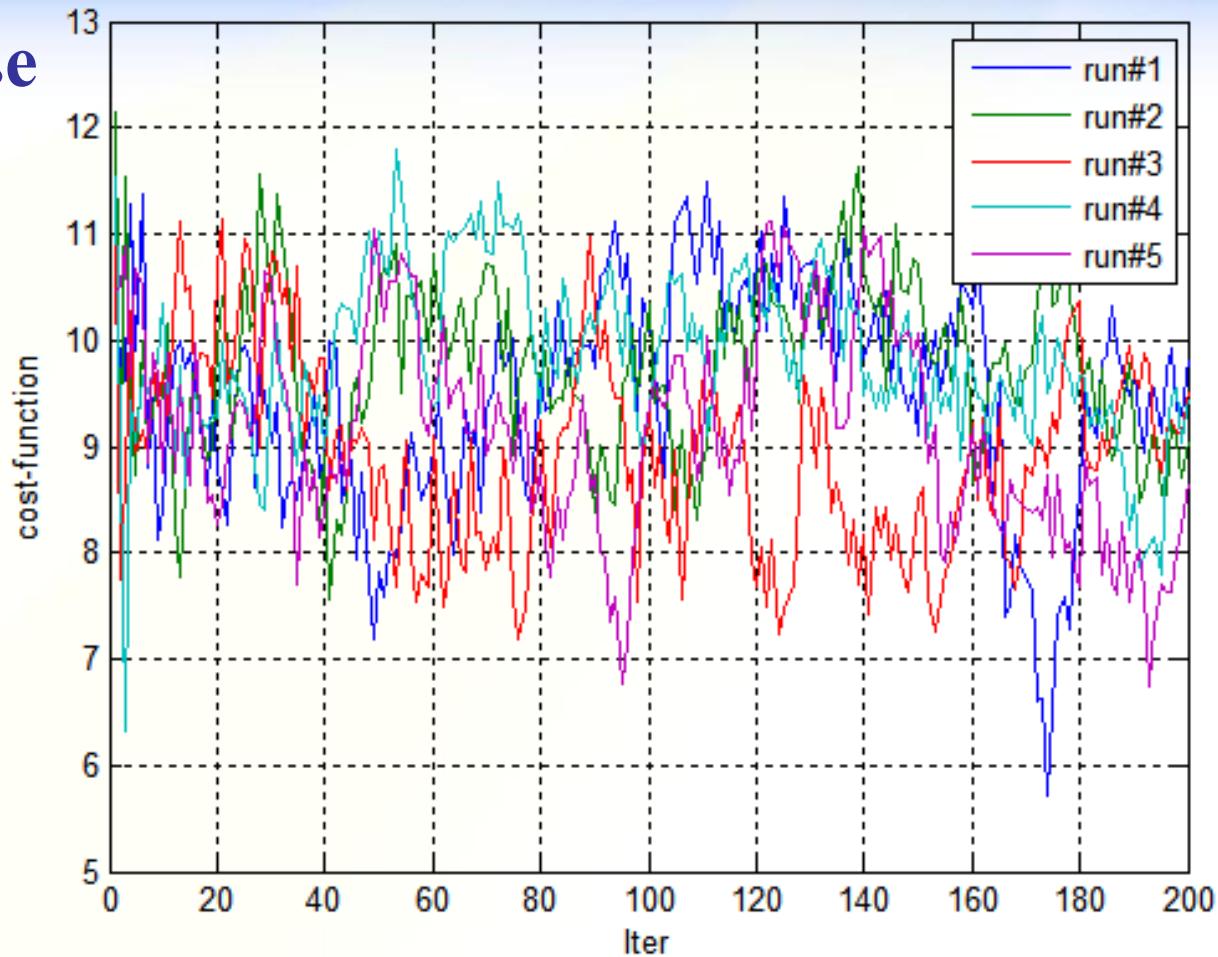
- Краткорочна меморија
 - памти се неколико последњих испитаних решења и њихове околине
- Дугорочна меморија
 - памте се (скоро) сва решења која су испитана
- Средњорочна меморија
 - памти се већи број решења која су пробана
- Избор меморије и дефинисање шта меморија представља зависе од проблема!

Закључци о Табу претраживању

- Табу претраживање се једноставније примењује на дискретне проблеме (TSP, SAT)
- За дискретне проблеме се меморија и табу зоне једноставније дефинишу
- Табу претраживање се релативно ретко примењује на генералне NLP оптимизационе проблеме

Типичан ток оптимизације стохастичког алгоритма

- Свако покретање стохастичког алгоритма даје различите резултате
- Формално решење је случајна променљива
- Како разматрати особине оваквих алгоритама?



Кумулативни минимум (дефиниција)

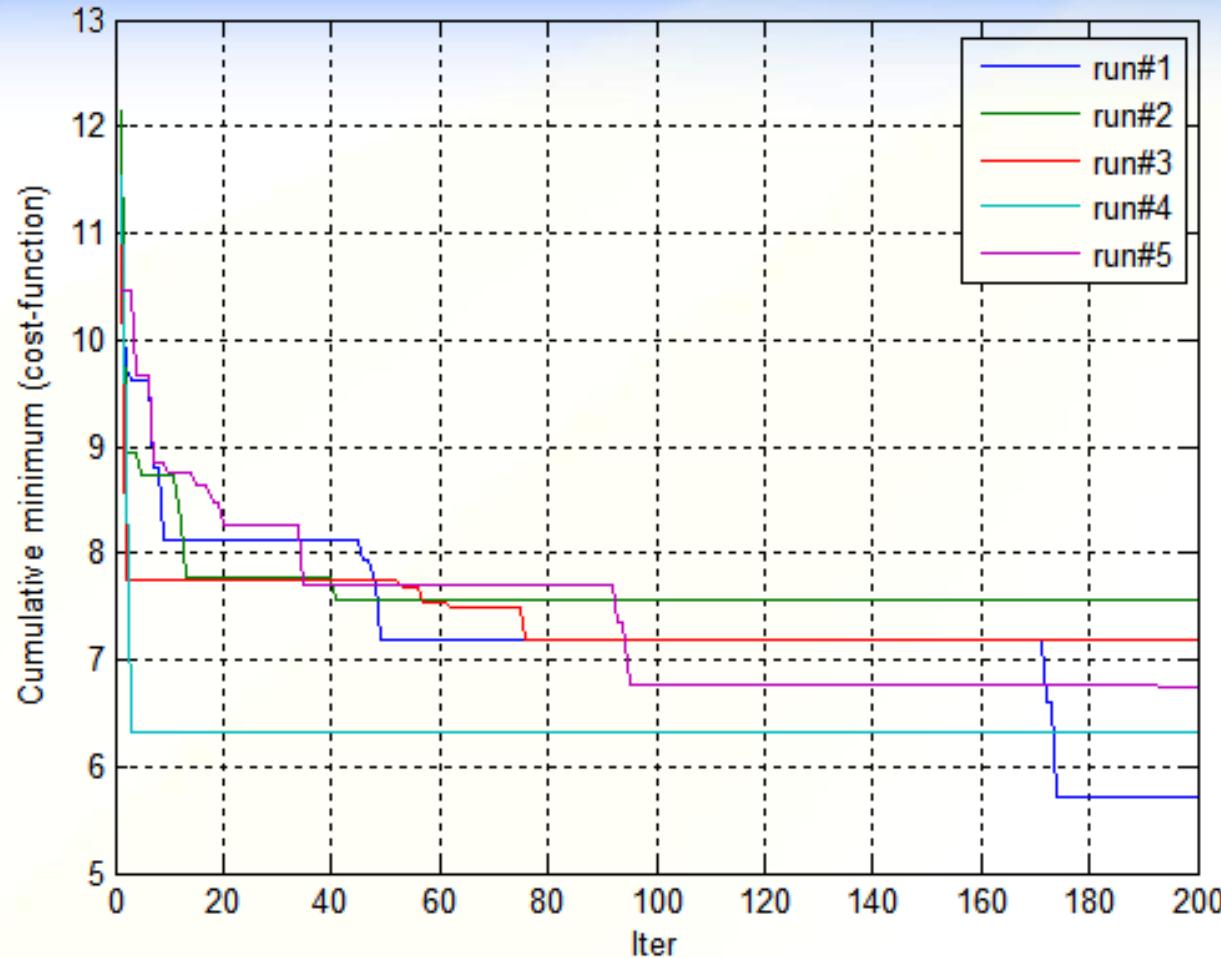
- Нека је задат низ вредности оптимизационе функције израчунате током оптимизација
$$(f_1, f_2, f_3, \dots, f_{k-1}, f_k, f_{k+1}, \dots, f_N)$$
- Кумулативни минимум је

$$c(k) = \min(f_1, f_2, \dots, f_k)$$

$$k = 1, 2, \dots, N$$

- Кумулативни минимум је низ дужине N који на позицији k има вредност најмање оптимизационе функције пронађене закључно са итерацијом k
- За стохастички оптимизациони алгоритам f_k и $c(k)$ су случајне променљиве
- $c(k)$ је монотоно нерестућа функција

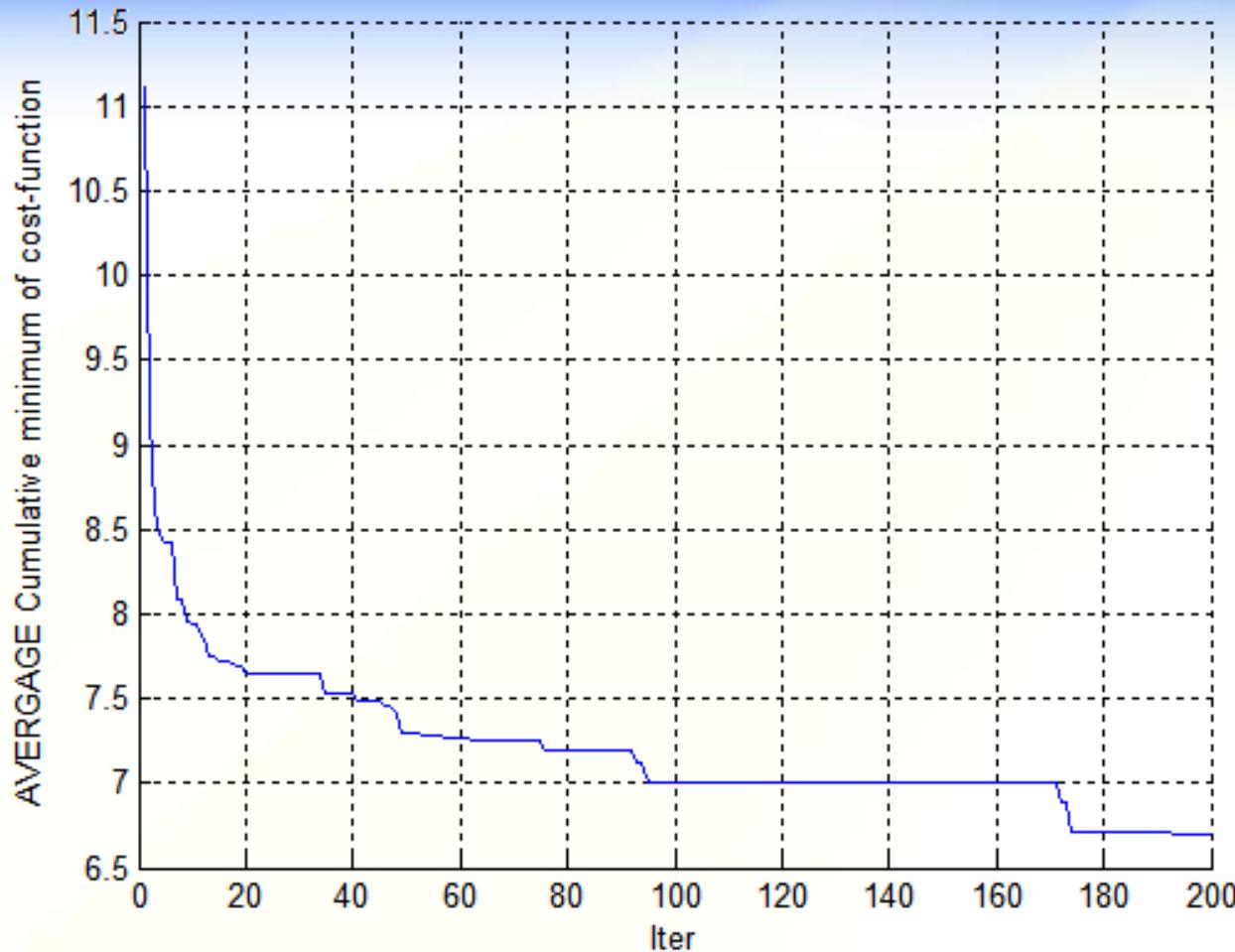
Кумулативни минимум (пример)



Средње најбоље пронађено решење (дефиниција)

- Нека су познати кумулативни минимуми за M различитих узастопних покретања оптимизације $c_1(k), c_2(k), \dots, c_M(k)$ $k = 1, 2, \dots, N$
- Средње најбоље пронађено решење је
$$\bar{c}(k) = \frac{1}{M} \sum_{p=1}^M c_p(k)$$
- Процена средње вредности оптимизационе функције коју проналази алгоритам за k итерација

Средње најбоље пронађено решење (пример)



Задатак за вежбе

- Дат је скуп од $D = 64$ фајла. Величине фајлова у [В] су редом
 $\mathbf{s} = (173669, 275487, 1197613, 1549805, 502334, 217684, 1796841, 274708, 631252, 148665, 150254, 4784408, 344759, 440109, 4198037, 329673, 28602, 144173, 1461469, 187895, 369313, 959307, 1482335, 2772513, 1313997, 254845, 486167, 2667146, 264004, 297223, 94694, 1757457, 576203, 8577828, 498382, 8478177, 123575, 4062389, 3001419, 196884, 617991, 421056, 3017627, 131936, 1152730, 2676649, 656678, 4519834, 201919, 56080, 2142553, 326263, 8172117, 2304253, 4761871, 205387, 6148422, 414559, 2893305, 2158562, 465972, 304078, 1841018, 1915571);$
- Потребно је што боље искористити меморију величине $64 \text{ MiB} = 2^{26} \text{ B}$, за складиштење ових фајлова
- Усвојени запис решења овог проблема је $\mathbf{x} = (x_1, x_2, \dots, x_D)$, $x_k \in \{0,1\}$, $k = 1, 2, \dots, D$ (0: фајл се не складишти, 1: фајл се складишти у датој меморији)
- Оптимизациона функција је: $f(\mathbf{x}) = \begin{cases} F, & F \geq 0 \\ 2^{26}, & F < 0 \end{cases}$, где је $F = 2^{26} - \sum_{k=1}^D x_k s_k$
- Тражи се минимум (проблем SAT класе)

Имплементација

- Написати имплементацију симулираног каљења
- За почетну температуру узети $T_0 \sim 32 * 1024 * 1024 = 32$ [MiB] (процењена средња вредност оптимизационе функције у оптимизационом простору)
- Максималан број итерација (израчунавања оптимизационе функције) је 100 000 за једно покретање
- Пустити 20 независних покретања и сачувати ток оптимизације (вредности оптимизационе функције у свакој итерацији)
- Израчунати и нацртати кумулативни минимум за свако покретање (20 кривих на једном графику)
- Израчунати и нацртати средње најбоље решење на основу претходних резултата
- Оба графика представити у log-log размери
- ASCII фајл уз решење би требало да садржи низ 64 бита који одговара најбољем пронађеном решењу и минималну вредност оптимизационе функције (решење које задовољава услов $f(\mathbf{x}) \leq 32$ је довољно добро)

Генетички алгоритам: основне референце

- Goldberg, David (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley Professional
- Fraser, Alex S. (1957). "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction," *Australian Journal of Biological Sciences* 10: 484–491.

Еволуција и оптимизација

- У природи, еволуција се заснива на
 - размножавању
 - мутацији
 - такмичењу
 - селекцији
- Овај процес се примењује на све живе организме, генерацију за генерацијом
- Немогуће је за једну врсту (живог света) да буде потпуно прилагођена на околину, јер се околина стално мења
- **Еволуција је оптимизациони процес**
 - Ко је започео овај процес оптимизације?
 - Ко је крајњи корисник резултата те оптимизације (еволуције)?

Дарвинова теорија еволуције и ГА

- ГА је настао из покушаја симулација живог света (нумеричка биологија)
 - Game-of-life
- Теорија еволуције (Дарвин) је објединила биологију, али је корак даље укључивање интелигенције
- Данас се сматра да се интелигенција и еволуција не могу развојити
- Нивои учења
 - филогенетички (кроз наследство)
 - онтогенетички (кроз искуство током живота јединке)
 - социогенетички (кроз интеракцију у групи)
- Локални оптимизациони алгоритми се могу схватити као апстракција онтогенетичког учења: једно решење се поправља на основу неких правила
- ГА се може схватити као апстракција филогенетичког учења – кроз искуство претходних генерација које је сачувано у генима

Генетички алгоритам: симулација опстанка

- Основна идеја: имплементације алгоритама опстанка из природе
- Генетички алгоритам (ГА) је један од генералних приступа оптимизацији
- Заснива се на принципу природне селекције и опстанка **најприлагођенијих** јединки околини
- ГА спада у **стохастичке** алгоритме за **глобалну** оптимизацију
- Алгоритам се најчешће описује
 - строге математичке дефиниције захтевају апстрактне дефиниције оператора

Елементи ГА: индивидуе, гени и способности

- Једно могуће решење оптимизационог проблема (једна тачка оптимизационог простора) назива се **индивидуа** и састоји се од **гена**
- Вектор оптимизационих променљивих је индивидуа $\mathbf{x} = (x_1, x_2, \dots, x_D)$
- Свака индивидуа има онолико гена колико променљивих има оптимизациона функција (број гена једнак је броју димензија оптимизационог простора, D)
- **Једна оптимизациона променљива**, за изабрани запис оптимизационог проблема, представља **један ген**
- Ген је најмања јединица за рекомбинацију (претрагу простора)
- Вредност оптимизационе функције, $f(\mathbf{x})$, у једној тачки оптимизационог простора, назива се **способност** (енг: fitness) индивидуе
 - Један ген нема f
 - Група гена која дефинише индивидуу има f

Елементи ГА: популација и генерација

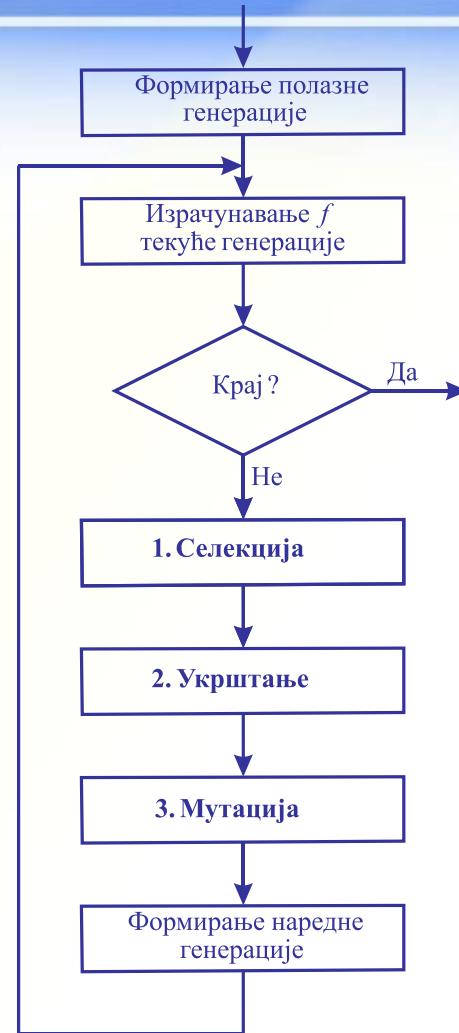
- ГА врши операције над **скупом решења** у оптимизационом простору
 - сви други претходно описани алгоритми оперишу само са једним (најбољим) решењем које “поправљају”
 - NM симплекс ради са скупом од тачно $D+1$ решења али се једно (најбоље) решење стално прати
- Скуп индивидуа (решења) назива се **популација**
- Број чланова популације је нужно већи од један
 - типично 100, 1000, или (много) више
- Током оптимизације, текућа популација замењује се наредном популацијом решења
- Популација у једном кораку алгоритма назива се генерација

Основна идеја ГА и формирање почетне генерације

- Идеја: од најбољих решења из претходне генерације формира се наредна генерација решења која (би требало да) су боља
- Формирањем сукцесивних генерација долази се до све бољих решења оптимизационог проблема
- ГА почиње формирањем почетне популације (полазног скупа решења) које се обично назива полазна или нулта генерација
 - најчешће је то скуп случајних вектора у опт. простору
 - може се формирати и на неки други начин (детерминистички, униформно, полазећи од процене, итд.)
 - ГА не залази у начин формирања нулте генерације

Блок дијаграм и формални кораци ГА

- Један корак ГА се састоји од 3 операције
 - селекција
 - укрштање
 - мутација
- Коришћењем ових операција од претходне генерације добија се наредна генерација
- Ово је најопштији случај алгоритма, прескацањем појединих оператора добијају се посебни случајеви



Бинарни и континуални ГА

- У зависности од представе гена, разликујемо две класе ГА:
 - бинарни ГА и
 - континуални ГА
- У литератури се под називом ГА најчешће подразумева бинарни ГА
- Од представе гена зависи реализација оператора укрштања и оператора мутације
- У литератури постоје опречна мишљења о томе које је представљање боље (зависи од конкретног проблема)

Представљање гена

- У природи су гени живог света представљени дискретним скупом елемената
 - број могућих комбинација људског генома је $4^{3\ 000\ 000\ 000}$ (макс.)
- Из тог разлога су у првим применама ГА оптимизациони параметри представљани бинарно
- Бинарна представа је врло погодна за дискретне комбинаторно-оптимизационе проблеме (SAT, TSP)
- Реалне променљиве:
потребно је извршити конверзију и дискретизацију
 - компликује имплементацију алгоритма
- Гени могу бити и реални бројеви, са тачношћу до машинске
- Сви оптимизациони проблеми (SAT, TSP, NLP) могу се решавати помоћу ГА

1. Оператор селекције

- У оквиру текуће генерације селектује се скуп родитељских решења
 - У природи су то они чланови популације који су оставили потомство
- Тај скуп служи за стварање наредне генерације, док остали чланови текуће генерације не учествују у формирању наредне генерације
- На овај начин се фаворизује простирање добрих решења и омогућава се њихово даље побољшавање
- Селекција се може реализовати
 - стохастички или
 - детерминистички

Стратегије селекције

- Најчешће коришћене стратегије селекције
 - Децимација
 - Пропорционална (рулет) селекција
 - Турнир селекција
- Постоји варијација за сваку од ових стратегија
- Постоје и многе друге стратегије које су предложене у литератури и коришћене у пракси
- Не постоји закључак која стратегија је (нај)боља

Стратегије селекције: Децимација

- Децимација популације
 - сортирање популације према оптимизационој функцији
 - затим се из тог скупа изабере подскуп првих k (најбољих) решења
- Предност овог приступа је **једноставност имплементације**
- Мана је сувише **брз губитак разноликости** генетског материјала унутар популације

Bilbo: I have... I have never used a sword in my life.

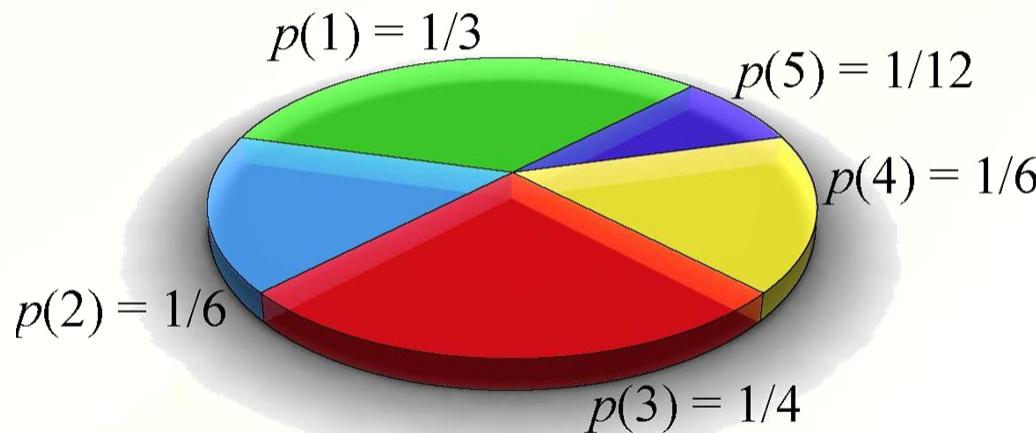
Gandalf: And I hope you never have to. But if you do, remember this: true courage lies in knowing not when to take a life, but when to spare one.

Елитизам и децимација

- При формирању следеће генерације могу се
 - задржати најбоља изабрана решења (елитистички приступ)
 - потпуно избацити решења из претходне генерације
- Иако се на први поглед може чинити да је добро оставити бар најбоље решење из претходне генерације, нумерички експерименти (статистички) показују **супротно**
- Елитистичка стратегија
 - доприноси бржој конвергенцији алгоритма
 - повећава вероватноћу конвергенције читаве генерације ка локалном минимуму из ког је алгоритму потребно много времена да изађе
- Елитистички приступ није погодан у општем случају, али се може користи као опција за убрзање конвергенције алгоритма

Стратегије селекције: Пропорционална (рулет) селекција

- Свакој индивидуи се додељује вероватноћа $p_i = \frac{f_i}{\sum_i f_i}$ избора p_i пропорционална способности индивидуе f_i
- Случајним избором селектује се једна индивидуа која учествује у стварању наредне популације



- Илустрација: популација са 5 индивидуа, случајном променљивом из скупа [0,1] бира се једна од индивидуа

Стратегије селекције: Пропорционална (рулет) селекција

- Вишеструким понављањем избора добија се група индивидуа за укрштање
- **Фаворизују се најбоља решења**, али постоји коначна вероватноћа да **понеко лоше** (али “срећно”) решење учествује у укрштању
- Повећава се разноликост генетског материјала током оптимизације, али се конкретна имплементација селекције усложњава
- Конвергенција ГА се успорава на овај начин

Стратегије селекције: Турнир селекција

- Из популације се на случајан начин изабере подскуп од N индивидуа
- Најчешће се користи бинарни турнир код кога је $N = 2$
- У изабраном подскупу пронађе се индивидуа са најбољом способношћу која је селектована за укрштање
- Изабрана индивидуа се избацује из основне популације како се не би догодило да поново буде изабрана
- Понављањем се добију све индивидуе за укрштање
- Смањује се вероватноћа брзог губитка генетске разноликости
- Има мању сложеност и рачунарске захтеве од пропорционалне селекције што га чини погоднијим за имплементацију

Оператор селекције: закључци

- Селекција се врши на основу оптимизационе функције (способности)
- Избором **стратегије селекције** се прави компромис између брзине конвергенције алгоритма и вероватноће проналажења локалног уместо глобалног минимума
- Избор стратегије селекције **не зависи од** записа оптимизационог проблема
- Све описане стратегије селекције могу се применити на било који оптимизациони проблем (SAT, TSP, NLP)
- Формално, било која операција која издваја подскуп решења из текуће генерације је **оператор селекције**
 - Селекција на случајан начин: ГА = случајно претраживање

2. Оператор укрштања

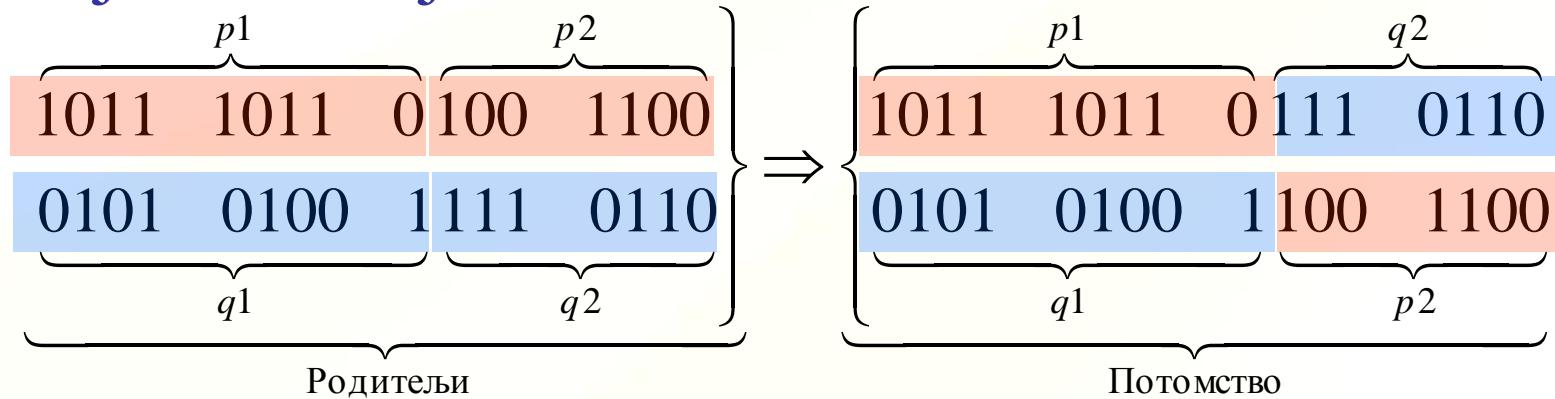
- Избором два или више решења из скупа најбољих формира се група решења за укрштање
- Њиховим укрштањем формирају се нова решења
- Укрштање се може реализовати
 - стохастички или
 - Вероватноћа укрштања је вероватноћа са којом се изабрана група индивидуа укршта и типично износи од 0,6 до 0,9
 - детерминистички
- Конкретна реализација оператора укрштања зависи од записа проблема

Бинарно укрштање (SAT проблеми)

- Гени су у овом случају бити $x_k = \{0,1\}$
- Индивидуе (решења) су секвенце бита (низ карактера поређаних један за другим)
 $x = (0,1,1,0,1,0,0,1)$
- Нека постоје два селектована решења за укрштање
- На случајан начин се изабере тачка (бит) укрштања у којој се пресеку низови оба родитељска решења
- Надовезивањем добијених поднизова добијају се два нова решења “потомка”
- На први подниз првог родитеља надовеже се други подниз другог родитеља и обрнуто

Пример бинарног укрштања

- Укрштање два решења која се састоје од по 16 бита



- Описани поступак назива се укрштање са једном тачком пресека (one-point crossover)
- Могућа је модификација тако да се прекидање генетског низа врши у две, три или више тачака

Континално укрштање (NLP проблеми)

- Континална представа гена
(реалне променљиве, $x_k \in \mathbb{R}$)
- Укрштање се најчешће врши линеарном комбинацијом вектора који представљају родитеље (\mathbf{r}_1 и \mathbf{r}_2), параметар $0 < \alpha < 1$

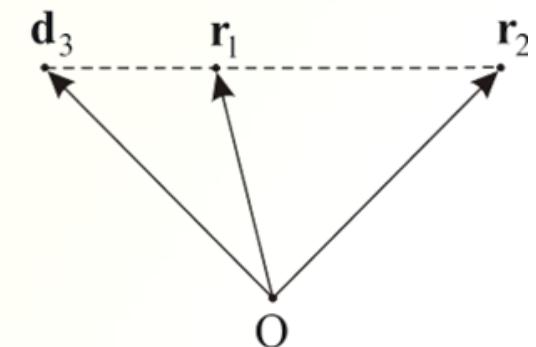
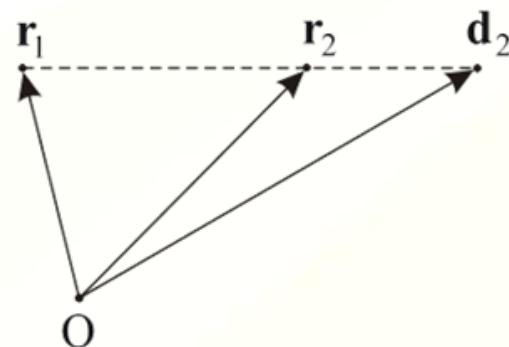
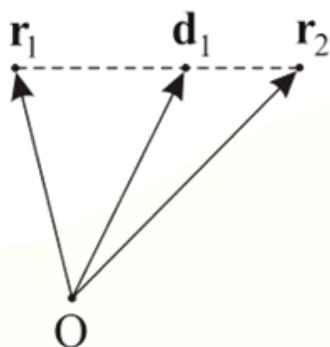
$$\mathbf{d}_1 = \mathbf{r}_2 + \alpha \cdot (\mathbf{r}_1 - \mathbf{r}_2) = \alpha \cdot \mathbf{r}_1 + (1 - \alpha) \cdot \mathbf{r}_2,$$

$$\mathbf{d}_2 = \mathbf{r}_2 - \alpha \cdot (\mathbf{r}_1 - \mathbf{r}_2) = -\alpha \cdot \mathbf{r}_1 + (1 + \alpha) \cdot \mathbf{r}_2,$$

$$\mathbf{d}_3 = \mathbf{r}_1 + \alpha \cdot (\mathbf{r}_1 - \mathbf{r}_2) = (1 + \alpha) \cdot \mathbf{r}_1 - \alpha \cdot \mathbf{r}_2,$$

Пример континуалног укрштања

- Пример континуалног укрштања
(два родитеља r_1, r_2 , три потомка d_1, d_2, d_3)



- У литератури су предложене различите реализације оператора укрштања за NLP
- Другачија укрштања нису драстично побољшала конвергенцију алгоритма на већем броју оптимизационих примера

Укрштање за TSP проблеме

- Запис TSP проблема је низ пермутованих елемената
- Два решења $\mathbf{x}_1=\{1,3,2,4,5\}$ $\mathbf{x}_2=\{3,2,5,4,1\}$
- Потребна је креативност за укрштање две пермутације
- Најједноставније је користити једног родитеља и заменити места неколико гена
- Замена места гена k парова
- Може се увести вероватноћа са којом се замена места извршава

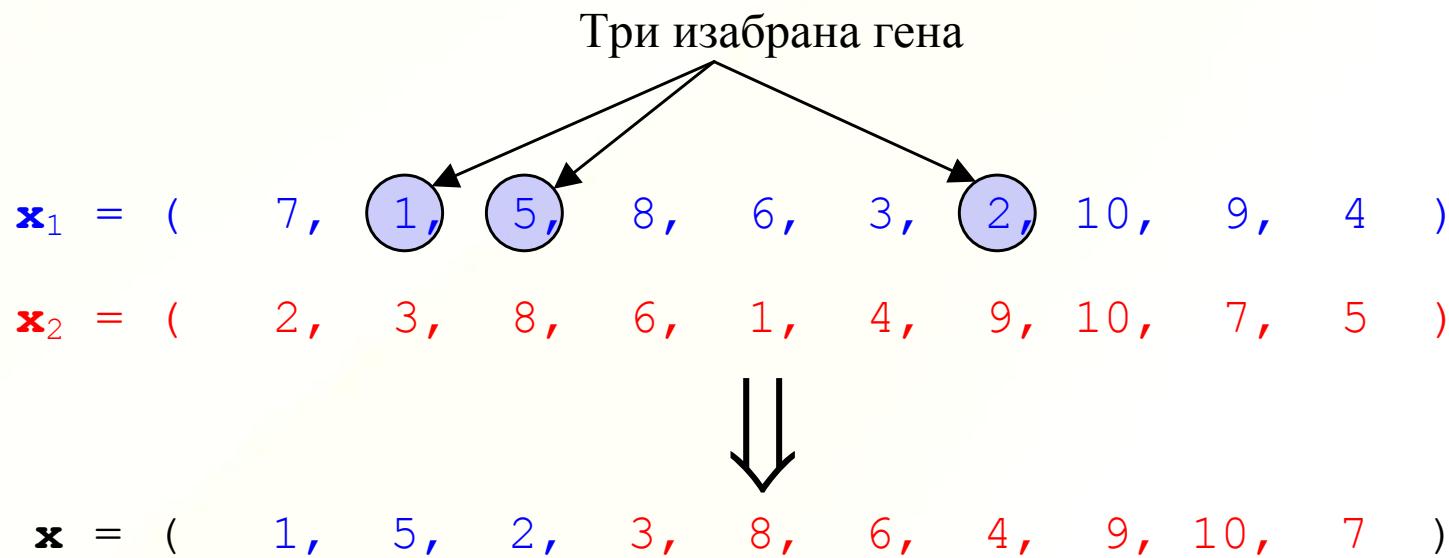
Илустрација укрштања за TSP проблеме, два пресека

- Запис са 10 променљивих
- Изаберу се две тачке пресека првог родитеља
- Гени између тачка пресека препишу се у ново решење, а остали гени се препишу из другог

$$\begin{array}{c} \text{прва тачка} \\ \text{пресека} \\ \text{друга тачка} \\ \text{пресека} \\ \hline \mathbf{x}_1 = (\quad 7, \quad 1, \quad | \quad 5, \quad 8, \quad 6, \quad 3, \quad | \quad 2, \quad 10, \quad 9, \quad 4 \quad) \\ \mathbf{x}_2 = (\quad 2, \quad 3, \quad 8, \quad 6, \quad 1, \quad 4, \quad 9, \quad 10, \quad 7, \quad 5 \quad) \\ \downarrow \\ \mathbf{x} = (\quad 5, \quad 8, \quad 6, \quad 3, \quad 2, \quad 1, \quad 4, \quad 9, \quad 10, \quad 7 \quad) \end{array}$$

Илустрација укрштања за TSP проблеме, k гена

- Запис са $D = 10$ променљивих, избор $k = 3$ гена
- Изабре се k гена из првог родитеља и они се препишу у ново решење
- Осталих $D-k$ гена се препише из другог родитеља



Илустрација укрштања за TSP проблеме, k замена

- Запис са 10 променљивих, $k = 2$ (две замене)

$$\mathbf{x} = (7, 1, 5, 8, 6, 3, 2, 10, 9, 4)$$

$$\mathbf{x} = (7, 1, 5, 8, \textcolor{red}{6}, 3, 2, 10, 9, \textcolor{red}{4})$$

$$\mathbf{x} = (7, 1, 5, 8, \textcolor{red}{4}, 3, 2, 10, 9, \textcolor{red}{6})$$

} Замена
места гена
првог паре

$$\mathbf{x} = (\textcolor{blue}{7}, 1, 5, 8, \textcolor{red}{4}, 3, 2, \textcolor{blue}{10}, 9, \textcolor{red}{6})$$

$$\mathbf{x} = (\textcolor{blue}{10}, 1, 5, 8, \textcolor{red}{4}, 3, 2, \textcolor{blue}{7}, 9, \textcolor{red}{6})$$

} Замена
места гена
другог паре

Оператор укрштања: закључци

- Укрштањем се формирају нова решења оптимизационог проблема (нове индивидуе) полазећи од селектованих решења
- Укрштање се понавља онолико пута колико је потребно да се формирају сва решења (индивидуе) за наредну генерацију
- Избор начина укрштања **зависи од записа** оптимизационог проблема
- За сваки тип записа (SAP, TSP, NLP) постоје посебне реализације укрштања
- Формално, било која операција која, на основу једног или више селектованих решења, формира нова решења је **оператор укрштања**

3. Мутација

- Реализује се искључиво стохастички
- Са унапред задатом вероватноћом изаберу се гени унутар новоформиране генерације који се на случајан начин промене
- Мутацијом се додаје нови генетски материјал у следећу генерацију (уноси се нова информација о оптимизационом простору)
- Мутација представља могућност да се истраже неистражени делови оптимизационог простора
- Вредност за вероватноћу мутације је обично мала, типично од 0,01 до 0,15
- Границе вредности вероватноће мутације:
 - 1: ГА = случајно претраживање
 - 0: ГА = локално претраживање

Мутација: SAT, NLP, TSP

- SAT: своди се на промену бита у генетском низу са унапред задатом вероватноћом
 - оригинални низ $\mathbf{x} = (1, 0, 1, 1, 0, 1, 0, 0)$
 - мутирани низ $\mathbf{x} = (1, 0, 0, 1, 0, 1, \textcolor{red}{1}, 0)$
- NLP: изводи се тако што се цео ген замени случајном вредношћу, у дозвољеном опсегу гена, са унапред задатом вероватноћом мутације
 - оригинални низ $\mathbf{x} = (0.81, 0.24, 0.93, -0.35)$
 - мутирани низ $\mathbf{x} = (0.81, \textcolor{red}{-0.62}, 0.93, -0.35)$
- TSP: изабере се подскуп гена којима се на случајан начин промене вредности (слично као укрштање са k замена!)
- Оператор мутације: случајна промена вредности гена једног решења

Критеријуми за завршетак

- Алгоритам се прекида онда када је решење пронађено или када се гени читаве генерације разликују за мање од неког унапред задатог прага
- Уколико се **оптимизација врши довољно дugo**, пре или касније ће се десити мутације које ће довести до проналажења **глобалног минимума**
- У пракси је готово увек недопустиво чекати толико дugo и алгоритам се прекида после одређеног броја генерација уколико критеријуми за завршетак нису пре тога достигнути
- Експериментално је утврђено да после ~50 генерација алгоритам конвергира за популације до 200 000 индивидуа

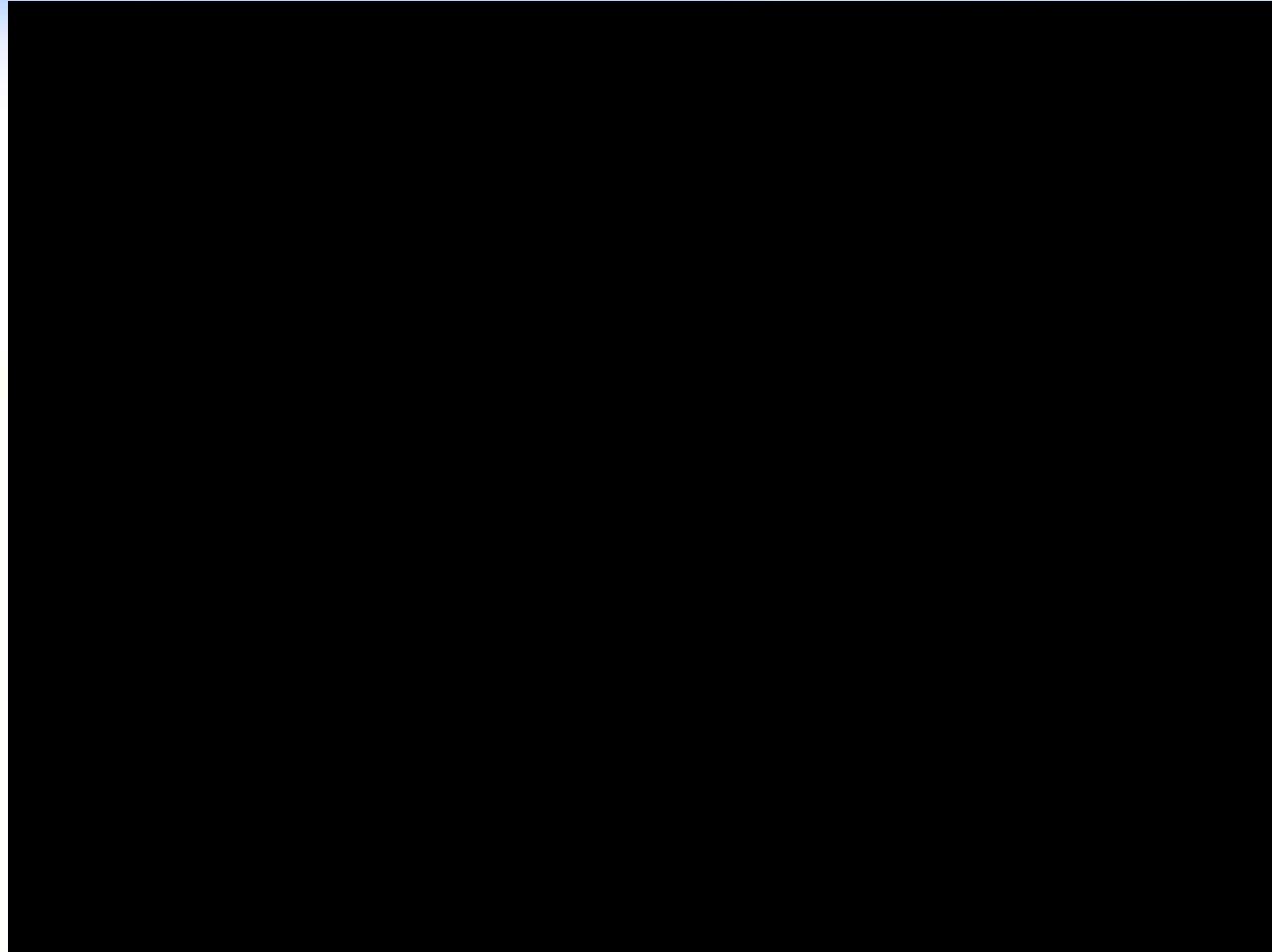
Параметри ГА

- Величина популације (N_{pop})
- Број генерација (N_{gen})
 - Укупан број итерација (израчунавања опт. функције) је $N_{\text{pop}} * N_{\text{gen}}$
- Запис променљивих
- Начин реализације сваког оператора (селекција, укрштање и мутација)
- Избор вероватноћа код стохастичких оператора

Који су оптимални параметри ГА?

- Величина популације треба да буде већа од броја оптимизационих променљивих
- Што већа популација то је боље решење на крају, али већа популација значи оптимизацију са више итерација
- Број генерација 30-100 довољан је за релативно мале популације (до 200 000)
 - за веће популације и даље је ?
- Кодовање:
бинарно или континуално у зависности од проблема
- Вероватноћа укрштања: 0,8
- Вероватноћа мутације: 0,1
- Број индивидуа које се укрштају ~популација / 5

Пример оптимизације са ГА



Имплементација ГА

- MATLAB: genetic algorithm
(Global Optimization Toolbox)
- Mathematica, Python, C/C++
 - постоје само “незванични” кодови
- Алгоритам је сложен и имплементације имају пуно степени слободе
- Тешко је користити ГА имплементацију коју добро не познајете, због великог броја степени слободе

Варијације ГА

- ГА са више популација које коегзистирају заједно са разменом од неколико индивидуа у свакој генерацији (енг: multipopulation GA)
- Микро ГА: вишеструко понављање ГА са релативно малим популацијама
- Коришћење локалних оптимизационих алгоритама у спрези са ГА (вишестепене оптимизације)
- ГА код кога су параметри алгоритма придржени оптимизационим променљивима $\mathbf{x}_{\text{tot}} = (\mathbf{x}_{\text{opt}}, \mathbf{x}_{\text{GA}})$
 - параметри ГА су променљиве које се мењају током оптимизације
 - ГА сам подешава своје параметре (могуће осцилације и дивергенција!)

Закључци о ГА

- Група решења (популација) уместо једног решења
- Добре стране:
 - Проналази глобално решење
(после довољно дуго времена појавиће се мутација која алгоритам пребацује у околину глобалног минимума)
 - Алгоритам се изузетно лако паралелизује
(прилагођен архитектури модерних рачунара)
 - Што већи број оптимизационих варијабли то је алгоритам ефикаснији у поређењу са другим алгоритмима
- Лоше стране:
 - Алгоритам има доста параметара
 - Релативно комплексан за имплементацију
 - Споро конвергира ка решењу
 - Захтева пуно израчунавања оптимизационе функције

Генерализација ГА: Evolutionary Algorithm (EA)

- EA = Genetic algorithm, Genetic programming, evolution strategies, evolutionary programming...
- Марковљев ланац (енг: Markov chain) будуће стање (стохастичког) система зависи само од текућег (садашњег) стања, а не зависи од претходних стања
- Наредна генерација решења добија се само на основу претходне, стога ГА/ЕА представља Марковљев ланац

Генерални опис EA

- Генерално $\mathbf{x}[n + 1] = s(v(\mathbf{x}[n]))$
- $\mathbf{x}[n]$ је генерација n ,
тј. скуп свих индивидуа у n -тој популацији
- $\mathbf{x}[0]$ је почетна популација (нулта генерација)
- $v()$ је **стохастички** оператор варијације (промене)
- $s()$ је оператор селекције
- Могуће је изоставити $s()$
није могуће изоставити $v()$

Стандардни запис ЕА

- (μ, λ) -ЕА
 - μ је број селектованих решења (родитеља)
 - λ је број нових решења која се генеришу (деца)
 $\lambda \geq \mu$
 - из скупа од λ нових решења селекцијом се бира μ
(претходна популација се не памти)
- $(\mu+\lambda)$ -ЕА
 - λ решења се генерише на основу μ решења
 - сви се заједно пореде
 - из скупа од $\mu+\lambda$ решења селекцијом се бира μ
(претходна популација се памти и пореди са наредном)

Примери ЕА: претходни алгоритми

- $(\mu+\lambda)$ -ЕА уз $N_{\text{pop}} = \mu + \lambda$ даје ГА са елитизмом
 - оператор варијације = оператор (ГА) укрштања + оператор (ГА) мутације
 - оператор селекције је исти за ЕА и ГА
- (μ, λ) -ЕА уз услов $N_{\text{pop}} = \lambda$, $\mu < \lambda$ је ГА без елитизма
 - оператор варијације = оператор (ГА) укрштања + оператор (ГА) мутације
 - оператор селекције је исти за ЕА и ГА
- $(1,1)$ -ЕА је генерализовано симулирано каљење
 - оператор варијације је избор наредне тачке, као што је дефинисано за симулирано каљење
 - оператор селекције је начин прихватања (или одбацивања) наредне тачке, као што је дефинисано за симулирано каљење

Примери ЕА: могуће имплементације

- $(1, \lambda)$ -ЕА
 - на основу једног решења генерише се λ решења у којима се израчунава f , а затим се бира једно најбоље решење и понавља се процес
 - нових λ решења се генерише на случајан начин са Гаусовом расподелом око основног решења (селектованог из прошле генерације)
- Једноставне имплементације ЕА су могуће
 - Избор оператора селекције и варијације је потпуно слободан
- Резултати показују да су једноставне ЕА имплементације сличних перформанси као и ГА (који је сложенији)

Закључци о ЕА

- ЕА је “швајцарски ножић” међу оптимизационим алгоритмима
- Може да се примени на СВЕ оптимизационе проблеме
- Избором оператора
 - Могу да се добију и локални и глобални алгоритми
 - Може да се мења брзина конвергенције алгоритма
- Избор параметара алгоритма
 - Величина популације
 - Број генерација
 - Оператори варијације и селекције
- Могућности имплементација су практично неограничене

Задатак за вежбе

- Дат је скуп од $D = 64$ фајла. Величине фајлова у [В] су редом
 $\mathbf{s} = (173669, 275487, 1197613, 1549805, 502334, 217684, 1796841, 274708, 631252, 148665, 150254, 4784408, 344759, 440109, 4198037, 329673, 28602, 144173, 1461469, 187895, 369313, 959307, 1482335, 2772513, 1313997, 254845, 486167, 2667146, 264004, 297223, 94694, 1757457, 576203, 8577828, 498382, 8478177, 123575, 4062389, 3001419, 196884, 617991, 421056, 3017627, 131936, 1152730, 2676649, 656678, 4519834, 201919, 56080, 2142553, 326263, 8172117, 2304253, 4761871, 205387, 6148422, 414559, 2893305, 2158562, 465972, 304078, 1841018, 1915571);$
- Потребно је што боље искористити меморију величине $64 \text{ MiB} = 2^{26} \text{ B}$, за складиштење ових фајлова
- Усвојени запис решења овог проблема је $\mathbf{x} = (x_1, x_2, \dots, x_D)$, $x_k \in \{0,1\}$, $k = 1, 2, \dots, D$ (0: фајл се не складишти, 1: фајл се складишти у датој меморији)
- Оптимизациона функција је: $f(\mathbf{x}) = \begin{cases} F, & F \geq 0 \\ 2^{26}, & F < 0 \end{cases}$, где је $F = 2^{26} - \sum_{k=1}^D x_k s_k$
- Тражи се минимум (проблем SAT класе)

Имплементација

- Написати имплементацију генетичког алгоритма
- За величину популације узети 2000
- Максималан број итерација (израчунавања оптимизационе функције) је 100 000 за једно покретање, тј. 50 генерација
- Пустити 20 независних покретања и сачувати ток оптимизације (вредности оптимизационе функције у свакој итерацији)
- Израчунати и нацртати кумулативни минимум за свако покретање (20 кривих на једном графику)
- Израчунати и нацртати средње најбоље решење на основу претходних резултата
- Оба графика представити у log-log размери
- ASCII фајл уз решење би требало да садржи низ 64 бита који одговара најбољем пронађеном решењу и минималну вредност оптимизационе функције (решење које задовољава услов $f(\mathbf{x}) \leq 32$ је довољно добро)

Диференцијална еволуција

- Storn, R. (1996) "On the usage of differential evolution for function optimization," *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*. pp. 519–523.
- R. Storn, K. Price, (1997) "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization* 11: 341–359.

Елементи диференцијалне еволуције

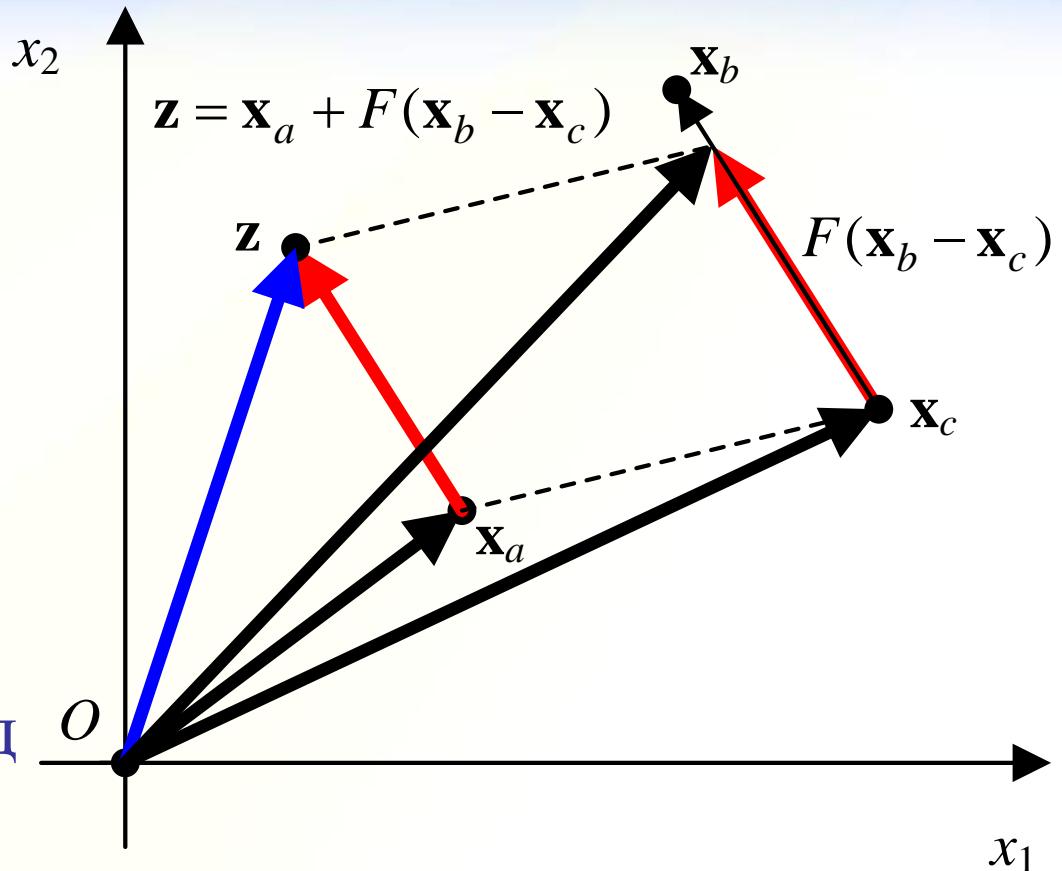
- У суштини је једна имплементација ГА/ЕА
 - ГА: селекција и укрштање (нема мутације!)
 - ЕА: селекција и варијација
- Алгоритам ради са скупом решења (популацијом)
- Почетни корак је иницијализација полазне популације
 - типично на случајан начин
 - израчунавање описне функције за полазну популацију
- У основној варијанти све оптимизационе променљиве су РЕАЛНИ бројеви (прилагођен за NLP проблеме)

Селекција

- За свако решење \mathbf{x} из текуће популације изаберу се три различита вектора (\mathbf{x}_a , \mathbf{x}_b и \mathbf{x}_c)
 - популација мора да има 4 или више елемента
- \mathbf{x}_a , \mathbf{x}_b и \mathbf{x}_c морају бити различити међусобно и различити од \mathbf{x}
- Дефинишу се параметри
 - Диференцијални тежински фактор F из интервала од 0 до 2
 - Вероватноћа укрштања CR из интервала од 0 до 1

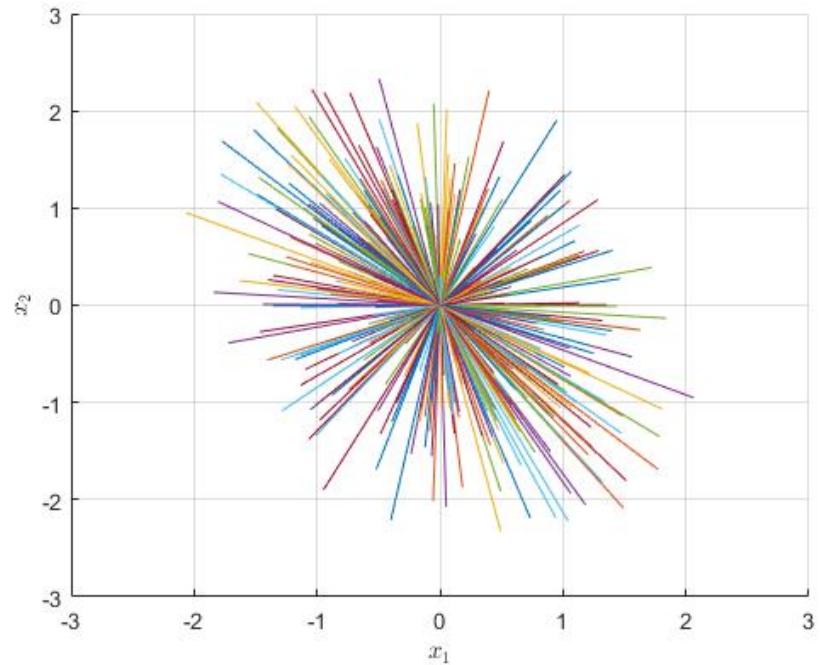
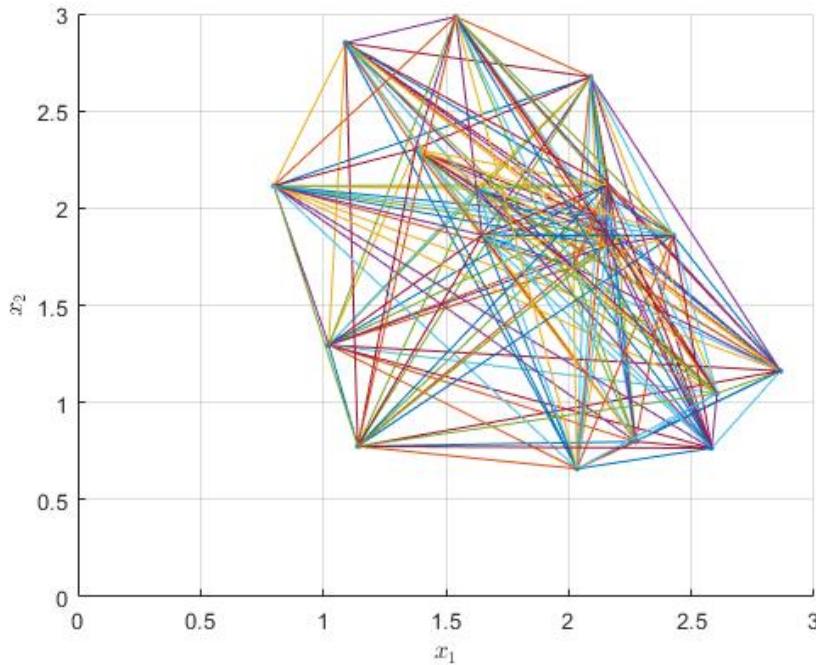
Међурешење

- Полазећи од изабраних \mathbf{x}_a , \mathbf{x}_b и \mathbf{x}_c
- Формира се међурешење $\mathbf{z} = \mathbf{x}_a + F^*(\mathbf{x}_b - \mathbf{x}_c)$
- Како је $\mathbf{x}_b \neq \mathbf{x}_c$ нови вектор \mathbf{z} је сигурно различит од \mathbf{x}_a , \mathbf{x}_b и \mathbf{x}_c



Расподела вектора разлика

- Слика лево: 20 тачака и сви вектори разлика
- Слика десно: сви вектори разлика у координатном почетку(симетрија)



Укрштање (или оператор варијација)

- Изабере се случајан цео број R између 1 и D , где је D број димензија оптимизационог простора
 - Смисао случајног броја R је да ће променљива под редним бројем R сигурно бити замењена
 - Тиме се обезбеђује да резултат укрштања увек буде различит од полазног решења!
- За сваки елемент вектора решења (променљиву) изабере се случајан број r_i са униформном расподелом од 0 до 1

Укрштање: услов и илустрација

- Нека је $\mathbf{y} = (y_1, y_2, \dots, y_D)$ вектор новог решења
- Уколико је
 $r_i < \text{CR}$ или $i = R$:
 $y_i = z_i$
- У супротном
 $y_i = x_i$
- Вектор бита приказан изнад слике је логичка вредност наведених услова (0-остаје x_i , 1-мења се на z_i)

$r_i < \text{CR}$ или $i = R$:

0	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

$\mathbf{x} = (\text{x1}, \text{x2}, \text{x3}, \text{x4}, \text{x5}, \text{x6}, \text{x7}, \text{x8})$

$\mathbf{z} = (\text{z1}, \text{z2}, \text{z3}, \text{z4}, \text{z5}, \text{z6}, \text{z7}, \text{z8})$

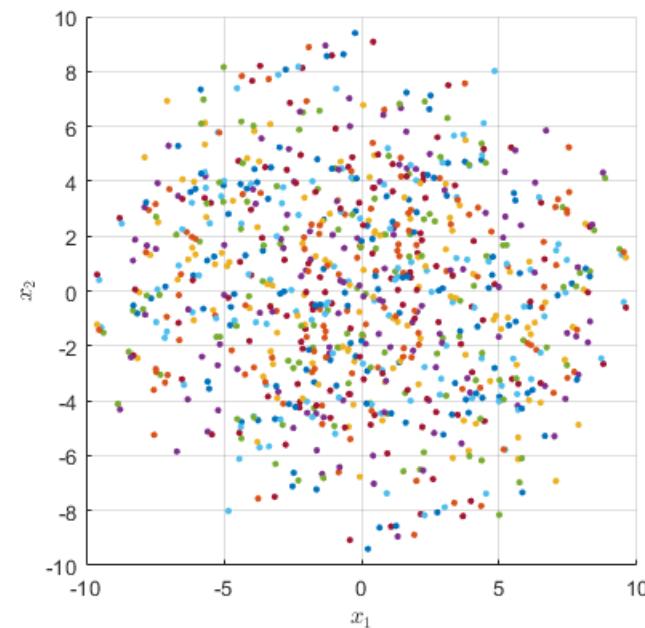
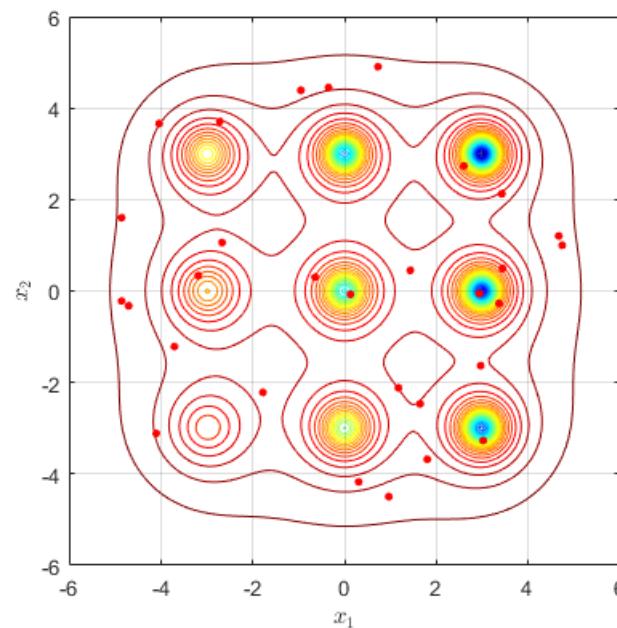
$\mathbf{y} = (\text{x1}, \text{z2}, \text{z3}, \text{x4}, \text{z5}, \text{x6}, \text{x7}, \text{z8})$

Услов за прихватање новог решења

- За свако решење \mathbf{x}_k из популације формира се ново решење $\mathbf{y}_k, k = 1, 2, \dots N_{\text{pop}}$
- У меморији се чувају две популације
 - текућа популација решења $\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_{N_{\text{pop}}}$ и
 - популација са кандидатима $\mathbf{y}_1, \mathbf{y}_2, \dots \mathbf{y}_{N_{\text{pop}}}$
- Уколико је $f(\mathbf{y}_k) < f(\mathbf{x}_k)$
 \mathbf{x}_k се замењује са \mathbf{y}_k у текућој популацији
- Уколико је $f(\mathbf{y}_k) \geq f(\mathbf{x}_k)$
 \mathbf{x}_k остаје у текућој популацији

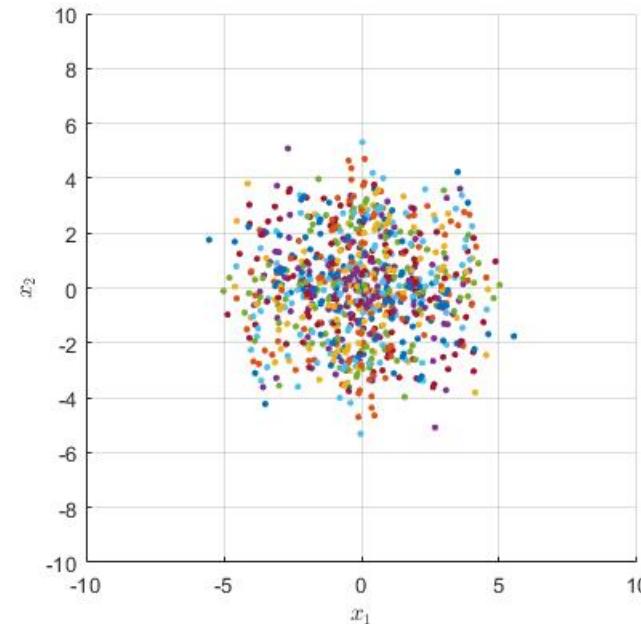
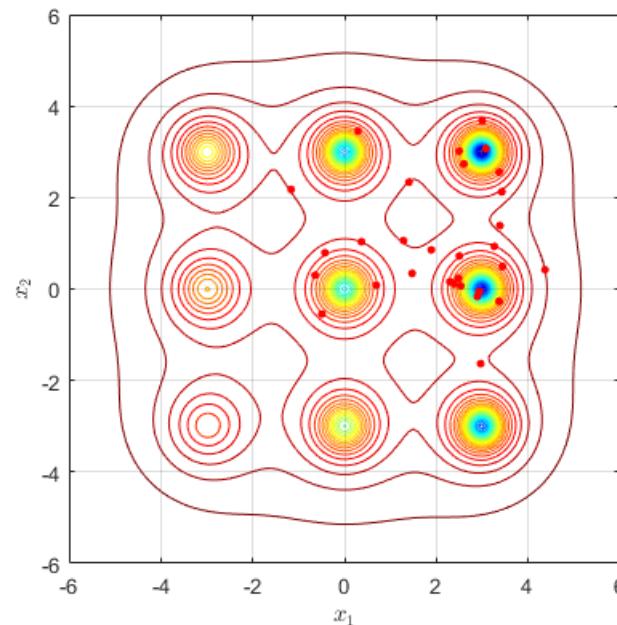
Ток оптимизације: Шекелова функција са 9 различитих мин.

- Величина популације је 20
- Нулта генерација и сви вектори разлика транслирани у координатни почетак



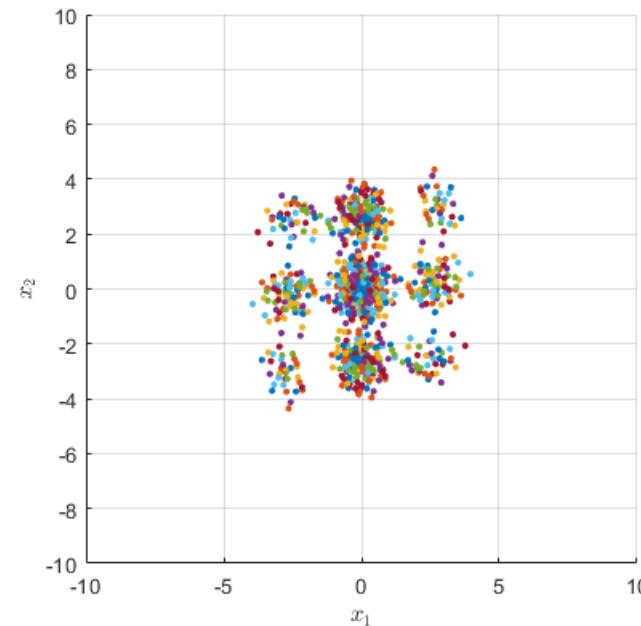
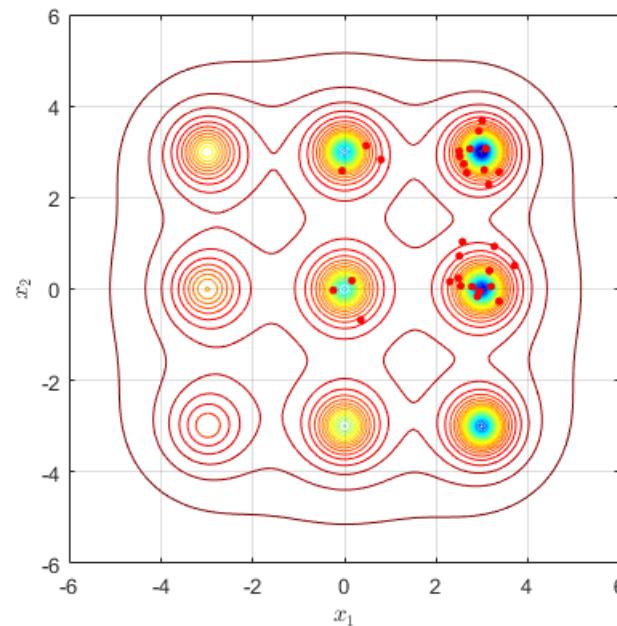
Ток оптимизације: Шекелова функција са 9 различитих мин.

- Генерација #5 и сви вектори разлика транслирани у координатни почетак
- Вектори разлика се смањују



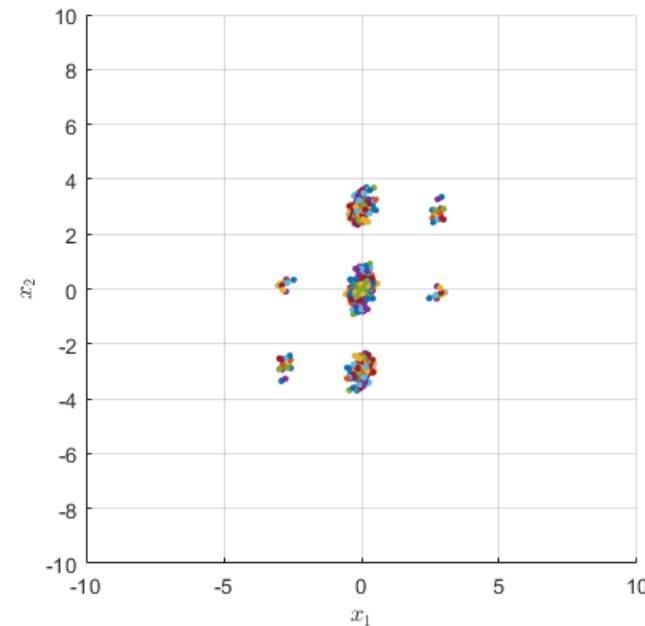
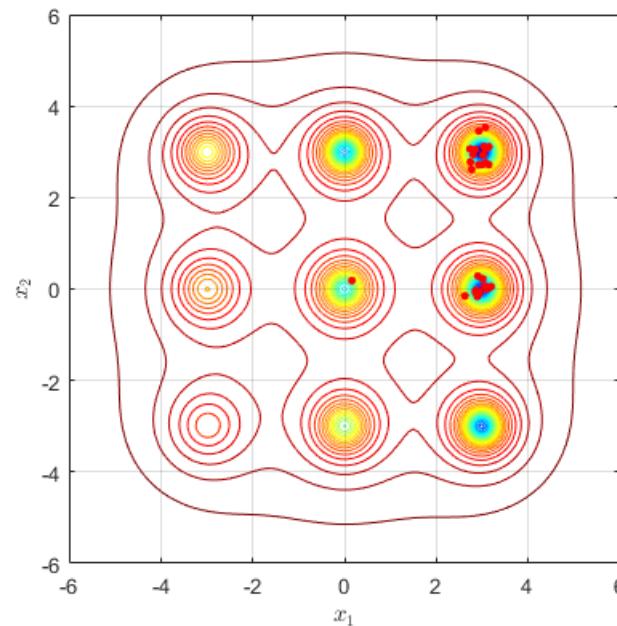
Ток оптимизације: Шекелова функција са 9 различитих мин.

- Генерација #10 и сви вектори разлика транслирани у координатни почетак
- Груписани вектори разлика: претрага СВИХ минимума!



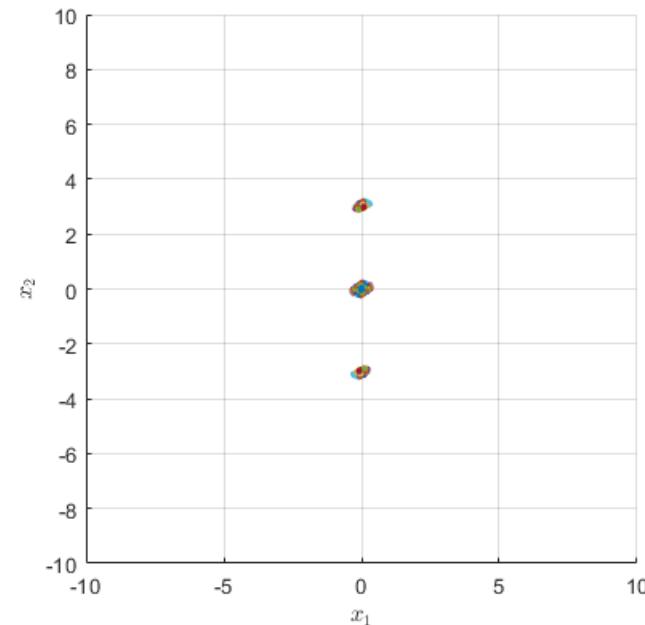
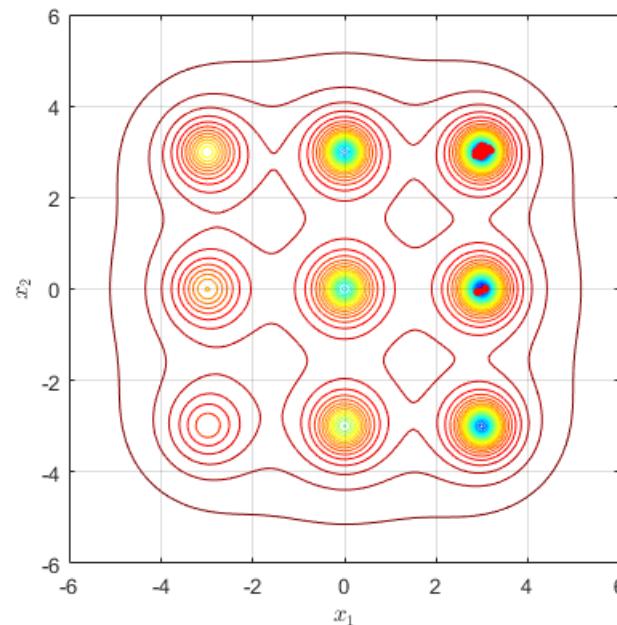
Ток оптимизације: Шекелова функција са 9 различитих мин.

- Генерација #20 и сви вектори разлика транслирани у координатни почетак
- Смањује се могућност претраге, почиње конвергенција



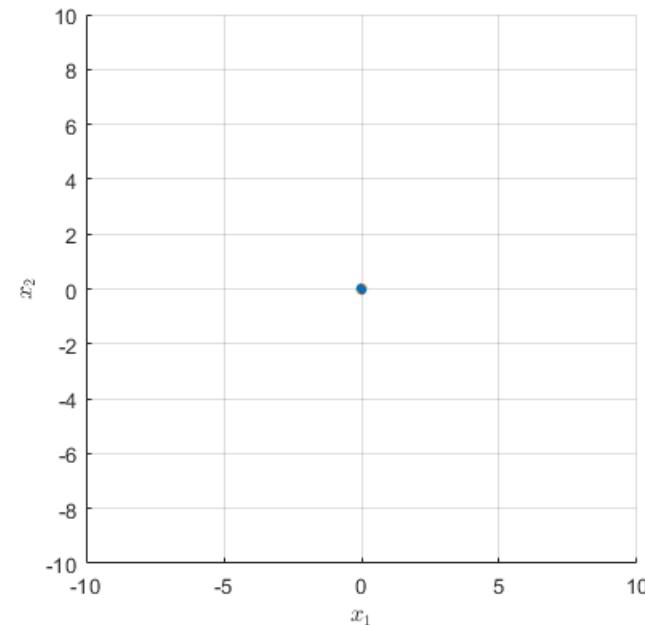
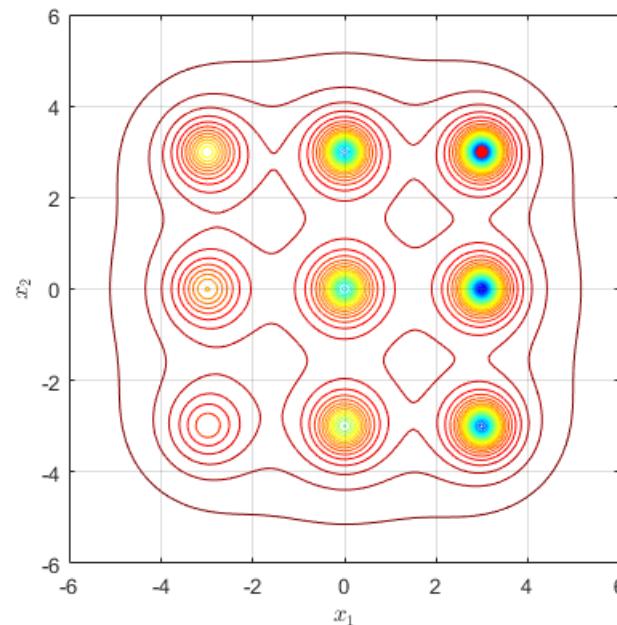
Ток оптимизације: Шекелова функција са 9 различитих мин.

- Генерација #30 и сви вектори разлика транслирани у координатни почетак
- Претражују се само још 2 минимума, конвергенција



Ток оптимизације: Шекелова функција са 9 различитих мин.

- Генерација #40 и сви вектори разлика транслирани у координатни почетак
- Претражују се само глобални минимум!



Пример оптимизације са диференцијалном еволуцијом

Означавање варијација алгоритма

- Класична диференцијална еволуција
- Стандардна ознака **DE/rand/1/bin**
 - Differential Evolution
 - random избор вектора \mathbf{x}_a , \mathbf{x}_b , \mathbf{x}_c
 - 1 разлика
 - binomial расподела бројева који се добијају од вектора разлике
- $(\lambda+\mu)$ -EA са $\lambda = \mu = N_{\text{pop}}$
 - оператор селекције: \mathbf{x} , \mathbf{x}_a , \mathbf{x}_b , \mathbf{x}_c
 - оператор варијације: $\mathbf{x} \wedge \mathbf{z} = \mathbf{x}_a + F^*(\mathbf{x}_b - \mathbf{x}_c) \rightarrow \mathbf{y}$
- Постоје и другачије имплементације алгоритма диференцијалне еволуције

Примена за SAT и TSP проблеме?

- SAT проблеми:
 - најједноставнија имплементација је тумачити бите као реалне бројеве
 - извршити израчунавње и
 - заокружити резултат за сваки бит на 0 или 1
- TSP проблеми:
 - потребно је дефинисати разлику два пермутована низа
 - добијени алгоритам више личи на ГА него на диферецијалну еволуцију
 - отворено је питање ефикасности за TSP проблеме

Закључци о диференцијалној еволуцији

- Стандардан избор параметара
 - Популација 10 пута D , неки сматрају да је ~ 50 овољно и за изузетно сложене проблеме
 - $F = 0,8$
 - $CR = 0,9$
- Диференцијална еволуција има знатно мање параметара од ГА
- Представља једну могућу имплементацију ЕА
- Ефикасност је врло слична ГА
- У пракси је изузетно добар алгоритам за решавање сложених NLP и SAT проблема са пуно променљивих
- Нема могућност проналажења глобалног оптимума после теоријски бесконачно много времена јер нема мутације

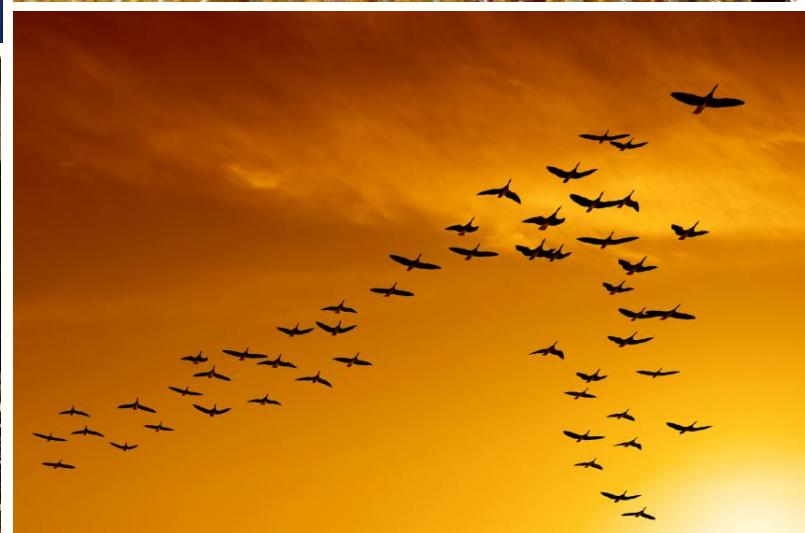
Имплементације диференцијалне еволуције

- Mathematica:
`NMinimize[f, vars, Method -> "DifferentialEvolution"]`
- Python:
`scipy.optimize.differential_evolution(func, bounds, ...)`
- Постоје разни “незванични” кодови доступни на Интернету

Particle Swarm Optimization (Оптимизација јатом): Референце

- J. Kennedy, R. Eberhart (1995), "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks IV*. pp. 1942–1948.
- J. Kennedy, R.C. Eberhart, (2001), *Swarm Intelligence*, Morgan Kaufmann, ISBN 1-55860-595-9

PSO аналогије



PSO и симулација одлучивања у групи

- Психологија: размишљање је процес који се нужно одиграва у групи!
- **Настајање норми и култура у оквиру једног друштва је оптимизациони процес!**
- Симулације друштвених процеса су истовремено и алгоритми за решавање оптимизационих проблема
- Мисли појединца могу се схватити као један елемент друштва (скупа појединаца)
- У комплексним ситуацијама, када је немогуће за појединца да рационално сагледа ситуацију која је сувише комплексна, људи прибегавају коришћењу веровања/убеђења/предрасуда која могу бити далеко од рационалног закључивања

Дељење информација и имитација као оптимизациони алгоритам

- Дељење информација појединих јединки је корисно за опстанак и напредак групе тих јединки
 - Јединка има своје знање/информације
 - Знање се преноси кроз групу (кроз веровања, предрасуде, понашање, социјалне мреже итд.)
 - Култура настаје као резултат тог процеса (култура оптимизира спознају – резултати се преносе на јединке које су далеко у групи)
- Имитирање других јединки, које раде неку операцију боље, је један (могући) алгоритам за оптимизацију

Оптимијациони проблем: избор изборног предмета

- Пример једног избора као оптимационог проблема
- **Одлука се доноси на основу два скупа информација**
 - Информације које **појединац сам прикупи** (градиво, наставник, начин полагања итд.)
 - Информације које **појединац добије од** других чланова групе (познаника)
- Без обзира који конкретан избор је у питању, увек постоје ова два скупа информација
- Утицај ова два скупа информација на коначну одлуку је различит за различите појединце и различите ситуације
- Симулација овог процеса је оптимизација јатом

PSO терминологија

- Агент (елемент, честица) је једно могуће решење у оптимизационом простору
 - вектор координата решења, $\mathbf{x} = (x_1, x_2, \dots, x_D)$
- Јато (група) је скуп агената помоћу којих се врши оптимизација (исто као популација код ГА)
- PSO је оптимизациони алгоритам који врши операције над скупом решења

Оптимизација и кретање јата

- Агент води рачуна о најбољем решењу које је пронашао током оптимизације (p_{best})
- Јато води рачуна о најбољем решењу које су пронашли сви агнети јата (g_{best})
- Агенти мењају своју позицију у (оптимизационом) простору на основу ова два решења (p_{best} и g_{best})

Промена позиције

- Позиција агента \mathbf{x}_n у n -том кораку је
$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mathbf{v}_n \Delta t$$
- \mathbf{x}_{n-1} је претходна позиција агента
- Δt је прираштај времена
 - у оптимизацији је 1 ради једноставности
- \mathbf{v}_n је брзина агента у n -том кораку
 - брзина је D димензиони вектор који одговара прираштају позиције
 - како се рачуна?

Промена брзине

- Промена брзине се рачуна према формулама

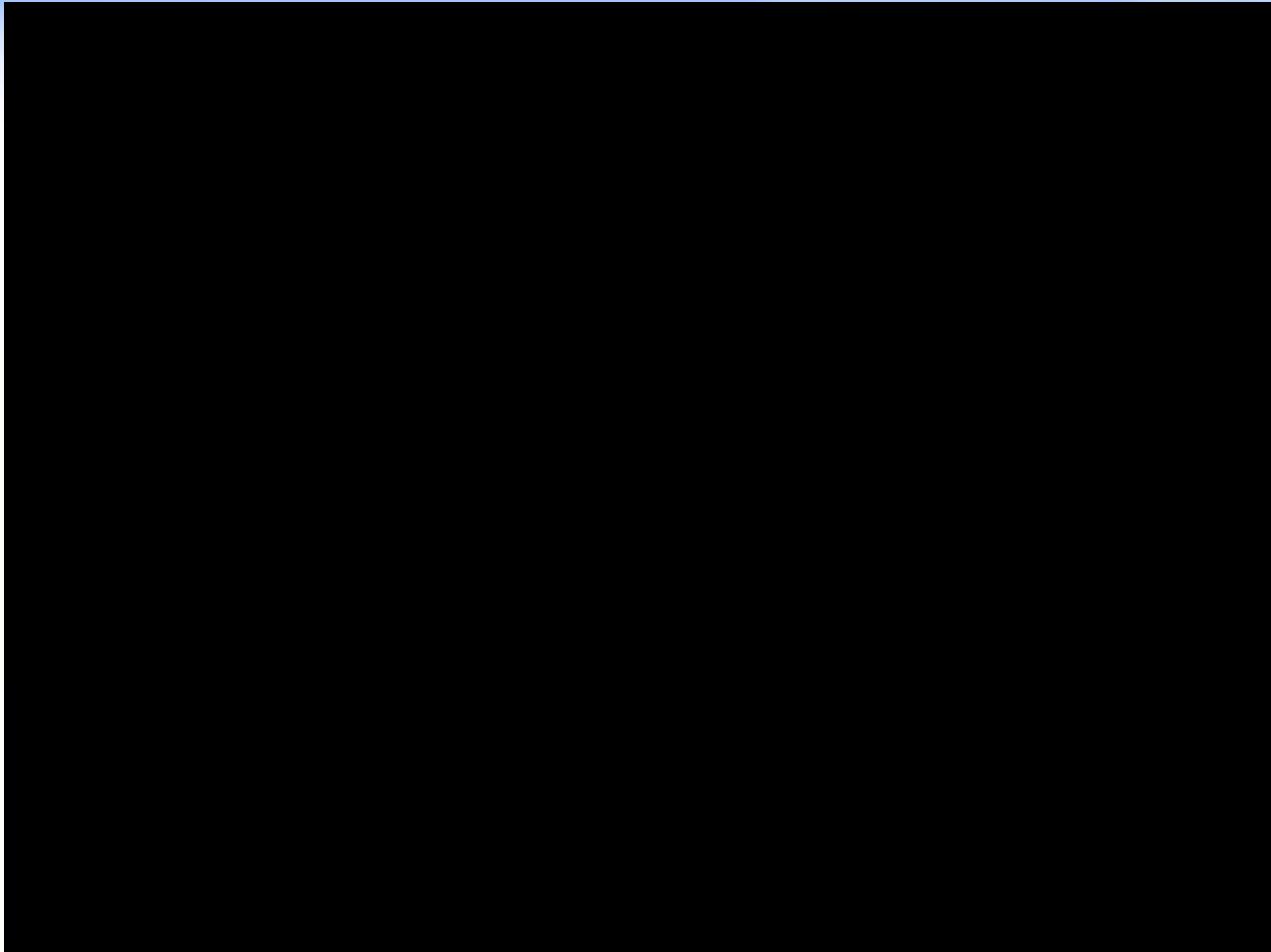
$$\mathbf{v}_n = w \cdot \mathbf{v}_{n-1} + c_1 \cdot \text{rand}() \cdot (\mathbf{p}_{\text{best}} - \mathbf{x}_{n-1}) + c_2 \cdot \text{rand}() \cdot (\mathbf{g}_{\text{best}} - \mathbf{x}_{n-1})$$

- \mathbf{v}_{n-1} је брзина агента у $n-1$ кораку
- $\text{rand}()$ је функција која генерише случајан број у интервалу $[0,1]$
- w је коефицијент инерције
- c_1 је когнитивни коефицијент
- c_2 је социјални коефицијент

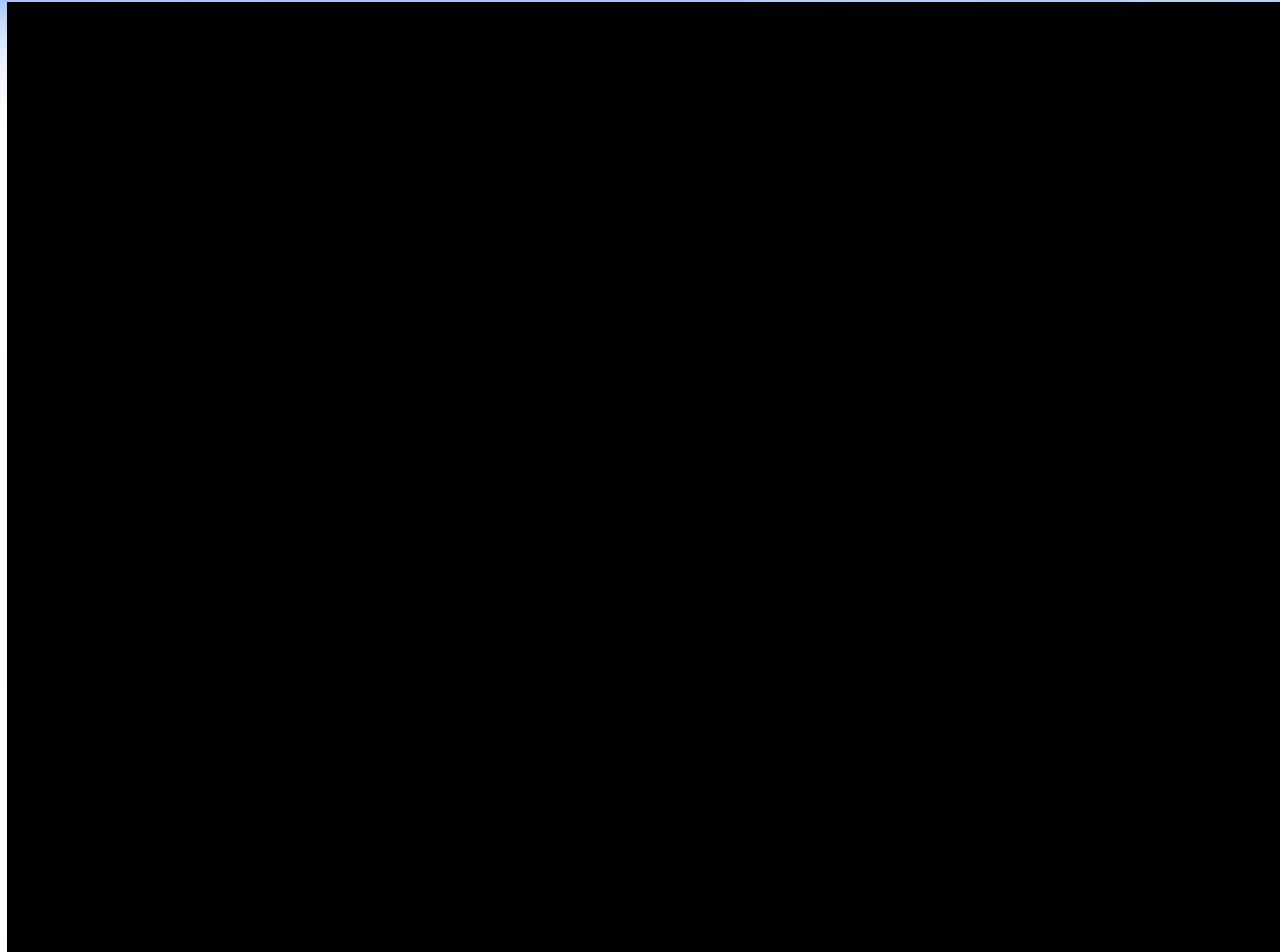
Параметри PSO алгоритма и њихове вредности

- Број агената у јату
- Коефицијент инерције $w = 0,729$
- Когнитивни коефицијент
- Социјални коефицијент $(c_1, c_2) = (1,494, 1,494)$
- Максимална брзина $v_{\max} = 0,2$
[под условом да су променљиве
у опсегу од -1 до 1]
тј. 10% опсега за сваку променљиву

Илустрација: 15 агената

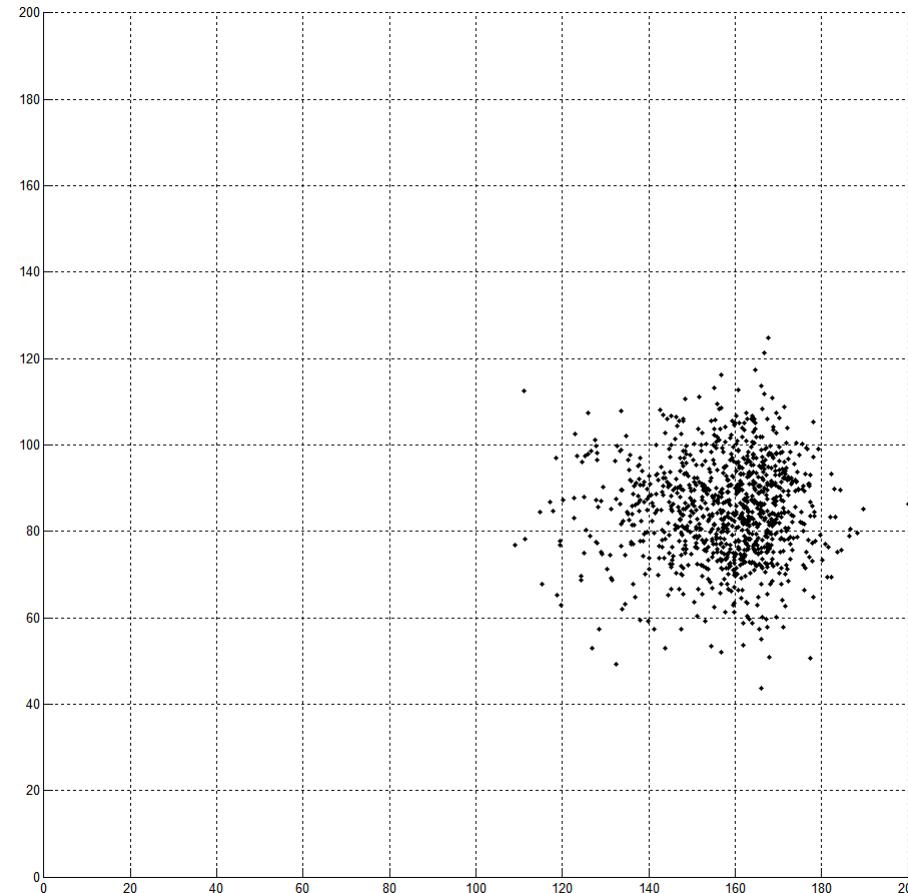


Илустрација: 50 агената



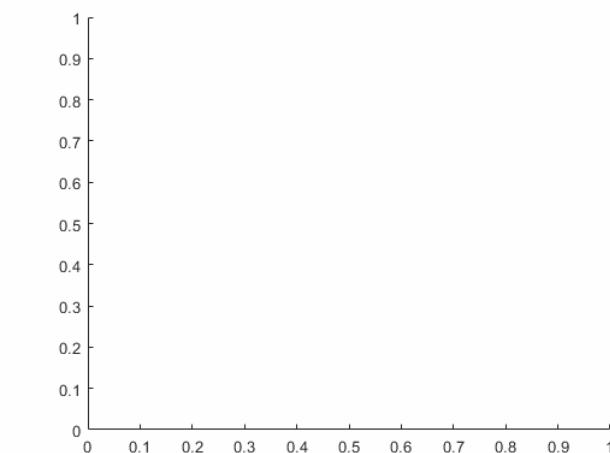
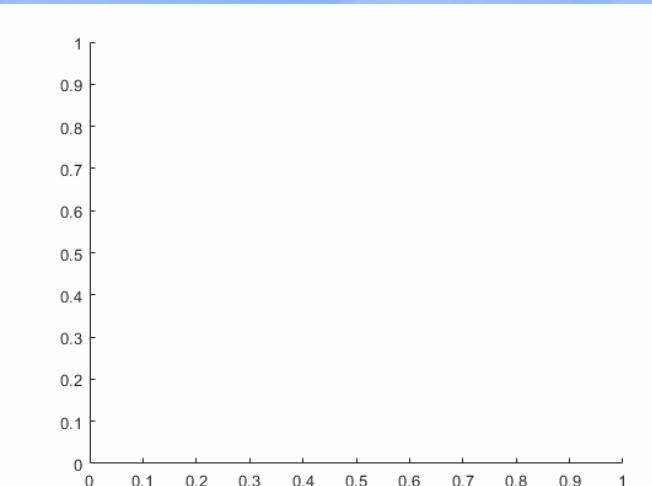
Particle Swarm Optimization

Илустрација



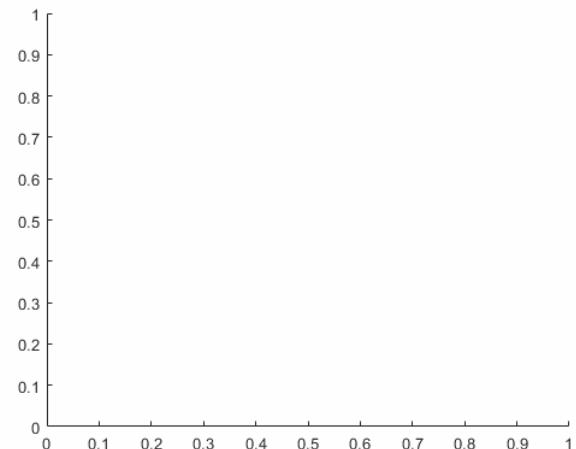
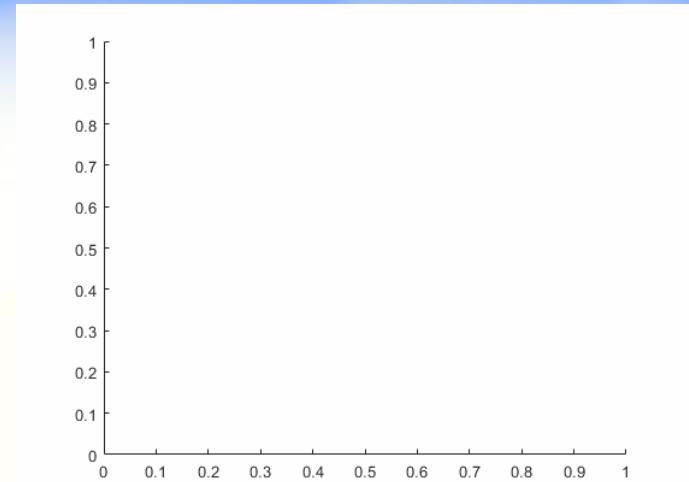
PSO параметри и стабилност: без случајних променљивих

- $N=1000, v_{\max}=0,2,$
 $c_1=c_2=1,5, w=0,8$
све случајне
променљиве су 1
- $N=1000, v_{\max}=0,2,$
 $c_1=c_2=4,0, w=0,8$
све случајне
променљиве су 1



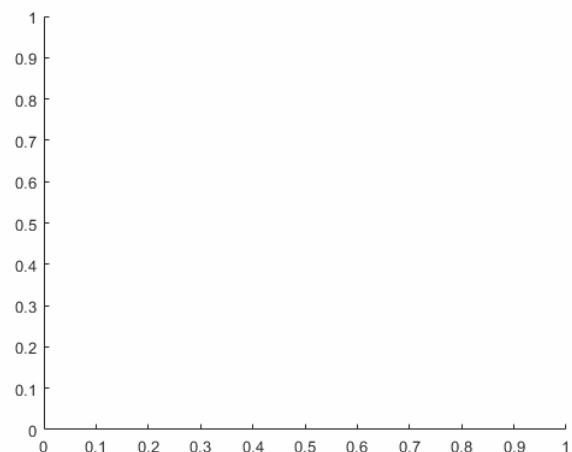
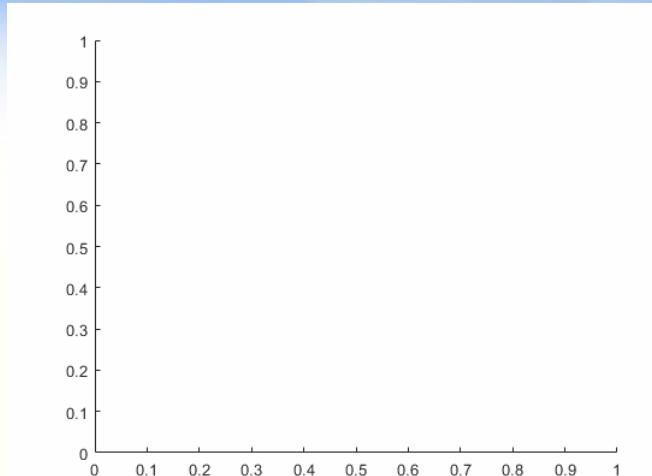
PSO параметри и стабилност: са случајним променљивима

- $N=1000$, $v_{\max}=0,2$,
 $c_1=c_2=1,5$, $w=0,8$
све случајне
променљиве су $\text{rand}(0,1)$
- $N=1000$, $v_{\max}=0,2$,
 $c_1=c_2=4,0$, $w=0,8$
све случајне
променљиве су $\text{rand}(0,1)$



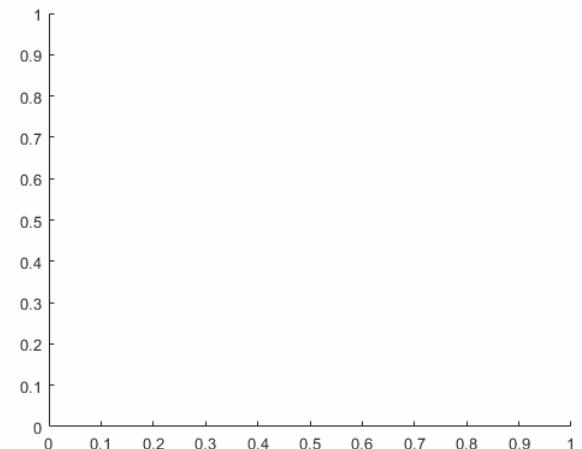
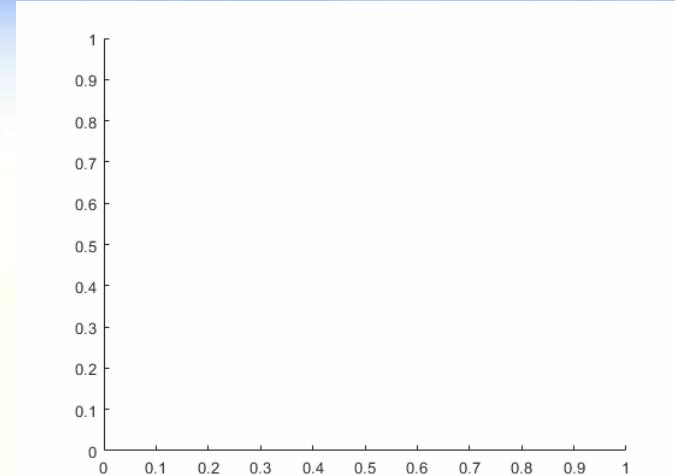
PSO параметри и стабилност: са случајним пром. без v_{\max}

- $N=1000$, без v_{\max}
 $c_1=c_2=1,5$, $w=0,8$
све случајне
променљиве су $\text{rand}(0,1)$
- $N=1000$, без v_{\max}
 $c_1=c_2=4,0$, $w=0,8$
све случајне
променљиве су $\text{rand}(0,1)$



PSO параметри и стабилност: са случајним пром. промена w

- $N=1000$, $v_{\max}=0,2$,
 $c_1=c_2=1,5$, $w=0,2$
све случајне
променљиве су $\text{rand}(0,1)$
- $N=1000$, $v_{\max}=0,2$,
 $c_1=c_2=1,5$, $w=1,2$
све случајне
променљиве су $\text{rand}(0,1)$

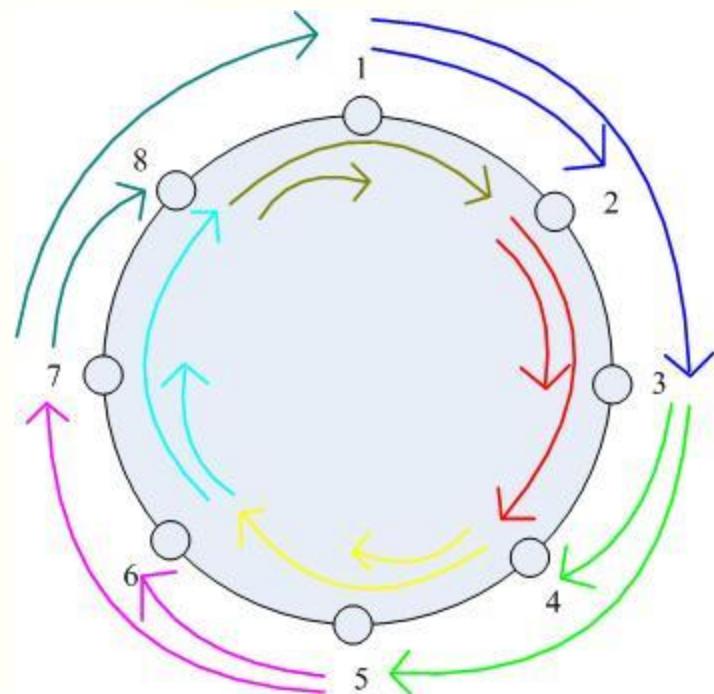


PSO параметри и стабилност: закључци

- PSO није апсолутно стабилан алгоритам
- “Експлозија” јата дешава се уколико параметри нису добро подешени
- Посебно, коефицијенти c_1 и c_2 морају бити мањи од 4
 - Ова граница је теоретски показана
- Смањивање w води ка брзој конвергенцији повећавање w води ка расипању јата

Тополошки PSO: варијације

- Уместо читавог јата агент има информације само о околини
- “Ring” топологија је најчешћа, са 2 суседа
- Може се генерализовати на “ m -околину”



PSO и EA

- Да ли се PSO може схватити као једна имплементација EA?
- (μ, λ) -EA, $\mu = \lambda$ = укупан број агената
 - Оператор варијације је начин промене “позиције” агента
 - Нема селекције
 - p_{best} и g_{best} су меморија!
- Формално EA нема меморију!
- Проширење EA уз дефиницију меморије?

PSO закључци

- Једноставан оптимизациони алгоритам
- Стохастички оптимизациони алгоритам
 - случајно генерисане полазне позиције
 - `rand()` у сваком кораку алгоритма
- Може да се заустави у локалном оптимуму, ако сви агенти исконвергирају ка њему (нема даљег начина за излазак!)
- На граници између локалних и глобалних

Имплементације PSO

- MATLAB:
`particleswarm(fun, nvars)`
- Python: посебна библиотека
`pyswarms`
- Разни кодови на Интернету...

Оптимизација која опонаша колонију мрава

- Ant colony optimization (ACO)
- A. Colomni, M. Dorigo et V. Maniezzo, *Distributed Optimization by Ant Colonies*, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.
- M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italie, 1992.

Основна идеја и терминологија

- Колонија мрава = популација
- Позиција мрава = једно могуће решење оптимизационог проблема
- Координате позиције одговарају оптимизационим променљивима
- Идеја: што више мрава прође неком путањом то је та путања боља

Наредна решења

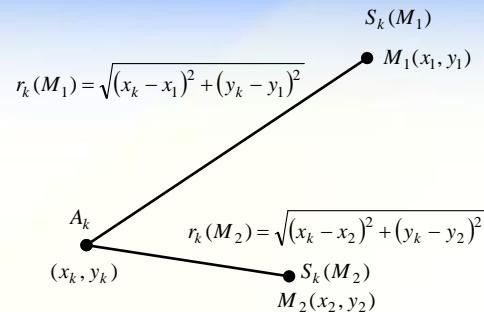
- Вероватноћа испитивања наредних решења (позиција) зависи од два параметра
 - Атрактивности решења
 - Нивоа пута (колико је мрава прошло туда)
- Ови параметри се мењају током оптимизације
- Релативно компликовано рачунање параметара

Закључци о АСО

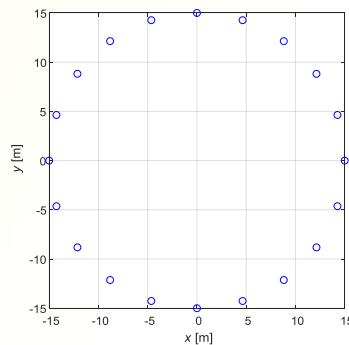
- Оптимизација са АСО алгоритмом је пре свега применљива за дискретне проблеме
- Алгоритам је прво примењен за проналажење пута у графовима
- Постоје варијанте алгоритма које се примењују на НЛР проблеме
- Комплексност алгоритма, чини се, није праћена значајним побољшањем ефикасности

Задатак за вежбе

- Сигнал, S_k , који еmitује предајник k дат је изразом $S_k = \frac{A_k}{r_k}$, где је A_k константа предајника (реалан број), а r_k је растојање (у метрима) између тачке у којој се налази предајник и тачке у којој се мери сигнал (слика 1).
- Ради одређивања локације и константи два непозната извора сигнала, извршена су мерења у $N = 20$ тачака.
- Мерне тачке су унiformно распоређене на кружници полупречника $R_0 = 15\text{ m}$, а координате мерних тачака су дате изразом $(x_i, y_i) = \left(R_0 \cos \frac{2\pi i}{N}, R_0 \sin \frac{2\pi i}{N} \right)$, $i = 0, 1, 2, \dots, N-1$ (слика 2).
- Извори сигнала се налазе у равни тог круга, у њему.
- Вредност сигнала у једној тачки простора једнака је збиру вредности сигнала које еmitују појединачни извори (важи принцип суперпозиције).



Слика 1. Предајник A_k који се налази у тачки (x_k, y_k) и две мерне тачке: $M_1(x_1, y_1)$ и $M_2(x_2, y_2)$.



Слика 2. Распоред мерних тачака.

Задатак за вежбе

- Вредности измереног сигнала у тим тачкама су, редом:

$S = (2.424595205726587e-01, \quad 1.737226395065819e-01, \quad 1.315612759386036e-01,$
 $1.022985539042393e-01, \quad 7.905975891960761e-02, \quad 5.717509542148174e-02,$
 $3.155886625106896e-02, \quad -6.242228581847679e-03, \quad -6.565183775481365e-02,$
 $-8.482380513926287e-02, \quad -1.828677714588237e-02, \quad 3.632382803076845e-02,$
 $7.654845872485493e-02, \quad 1.152250132891757e-01, \quad 1.631742367154961e-01,$
 $2.358469152696193e-01, \quad 3.650430801728451e-01, \quad 5.816044173713664e-01,$
 $5.827732223753571e-01, \quad 3.686942505423780e-01)$

- Усвојени запис решења овог проблема је $\mathbf{x} = (x_{P_1}, y_{P_1}, x_{P_2}, y_{P_2}, A_1, A_2)$.
- Оптимизациона функција је

$$f_{\text{opt}}(\mathbf{x}) = \begin{cases} \sum_{i=0}^{N-1} \left(\frac{A_1}{\sqrt{(x_i - x_{P_1})^2 + (y_i - y_{P_1})^2}} + \frac{A_2}{\sqrt{(x_i - x_{P_2})^2 + (y_i - y_{P_2})^2}} - s_i \right)^2, & \sqrt{x_{P_1}^2 + y_{P_1}^2} < R_0 \text{ и } \sqrt{x_{P_2}^2 + y_{P_2}^2} < R_0 \\ 100, & \sqrt{x_{P_1}^2 + y_{P_1}^2} \geq R_0 \text{ или } \sqrt{x_{P_2}^2 + y_{P_2}^2} \geq R_0 \end{cases}$$

Задатак за вежбе

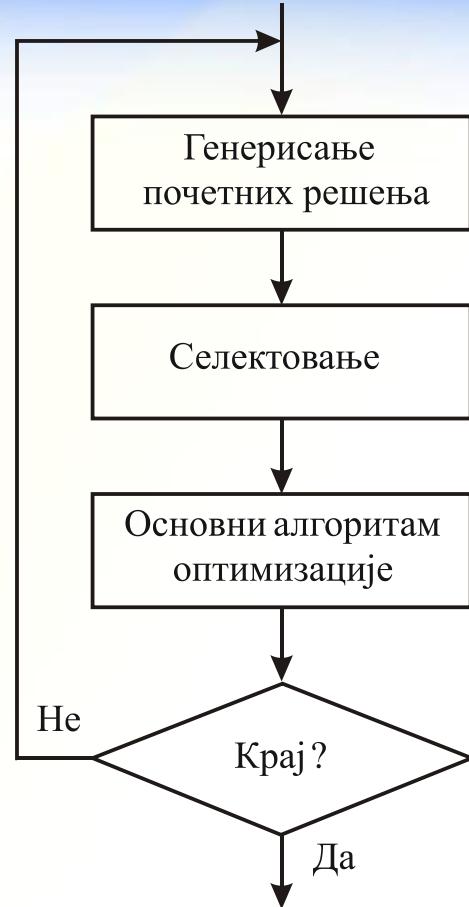
- Потребно је пронаћи координате позиција и константе непознатих извора
- Написати своју имплементацију алгоритма **диференцијалне еволуције**
- Уколико се користи готова функција за диференцијалну еволуцију (из Pythona или неког другог извора), задатак носи **2 поена**
- Пронаћи и у ASCII фајл записати решење задатог проблема за које је $f_{\text{opt}} \leq 10^{-14}$

Методе више минимума засноване на понављању

- У пракси, оптимизациони алгоритми се покрећу више пута са различитим почетним решењима
- Такав приступ се примењује уколико је:
 - потребно пронаћи више различитих решења или
 - доказати да больих решења (највероватније) нема
- Три основна корака

Блок дијаграм

- Генерисање скупа почетних решења
- Селекција (нај)бољих решења
- Оптимизација са полазним решењем добијеним у претходном кораку



Практични примери: случајно полазно решење

- Понављање основних оптимизационих алгоритама са случајно изабраном почетном тачком
 - градијентна метода или симплекс са случајно изабраним полазним решењем
 - симулирано каљење са случајном полазном тачком
 - генетички алгоритам са случајно изабраном првом генерацијом
 - PSO или диференцијална еволуција са случајно изабраним полазним популацијама

Практични примери: скуп случајно одабраних решења

- Понављање основних оптимизационих метода са најбољом тачком из претходно генерисаног скупа
 - комбиноване двостепене методе:
 - локални оптимизациони алгоритам покренут са најбољим решењем из случајно генерисаног скупа
 - децимација прве генерације ГА (прва генерација има знатно више решења од N_{pop} , изаберу се најбоља решења)

Вишеструка понављања: литература и примена

- Вишеструко понављање ГА се у литератури често назива микро-генетички алгоритам (microgenetic algorithm)
- Вишеструко понављање симулираног калења се назива прекаливање (reannealing)
- Ови алгоритми се могу користити за:
 - проналажење једног (глобалног) оптимума и
 - проналажење више решења

Особине поновљених оптимизација

- Добра страна метода заснованих на понављању је лака имплементација
- **Основна мана алгоритама ове класе је недостатак корелације између појединачних покретања**
- Ово има за последицу то да су **крања решења** која се добијају после појединачних покретања алгоритама **међусобно независна**
- Вероватноћа **проналажења истог решења** је релативно велика, што је неефикасан начин оптимизације

Теоријска ефикасност алгоритама заснованих на понављању

- Претпоставимо да оптимизациона функција има N минимума
 - унiformно распоређених у простору и
 - једнаких дубина
- Вероватноћа проналажења сваког минимума је стога једнака и износи $1/N$
- Колика је вероватноћа проналажења свих N минимума после k независних покушаја?

Вероватноћа проналажења свих N минимума из k покушаја, $P(N,k)$

$$P(N,k) = \frac{Q(N,k)}{N^k}$$

$$Q(N,k) = \begin{cases} N^k - \sum_{p=1}^{N-1} \binom{N}{p} \cdot Q(p,k), & N > 1 \\ 1, & N = 1 \end{cases}$$

$N = 10$	$N = 20$	$N = 30$
$P(10, 10) = 3.63 \cdot 10^{-4}$	$P(20, 20) = 2.32 \cdot 10^{-8}$	$P(30, 30) = 1.33 \cdot 10^{-12}$
$P(10, 27) = 0.520$	$P(20, 67) = 0.500$	$P(30, 113) = 0.503$
$P(10, 44) = 0.905$	$P(20, 103) = 0.902$	$P(30, 168) = 0.903$

Процена локалних минимума

- Генерирање полазног скупа решења која се налазе у целом оптимизационом простору
- Процењивање положаја локалних минимума
- Покретање “основног” оптимизационог алгоритма



Минимизација вероватноће проналажења истих решења

- Проценом положаја локалних минимума на основу свих елемената скупа генерисаног у првом кораку, или на основу неког његовог подскупа
- Остварује се минимална вероватноћа вишеструких претраживања истих делова оптимизационог простора
- Ова особина је од кључног значаја за ефикасност алгоритама ове класе

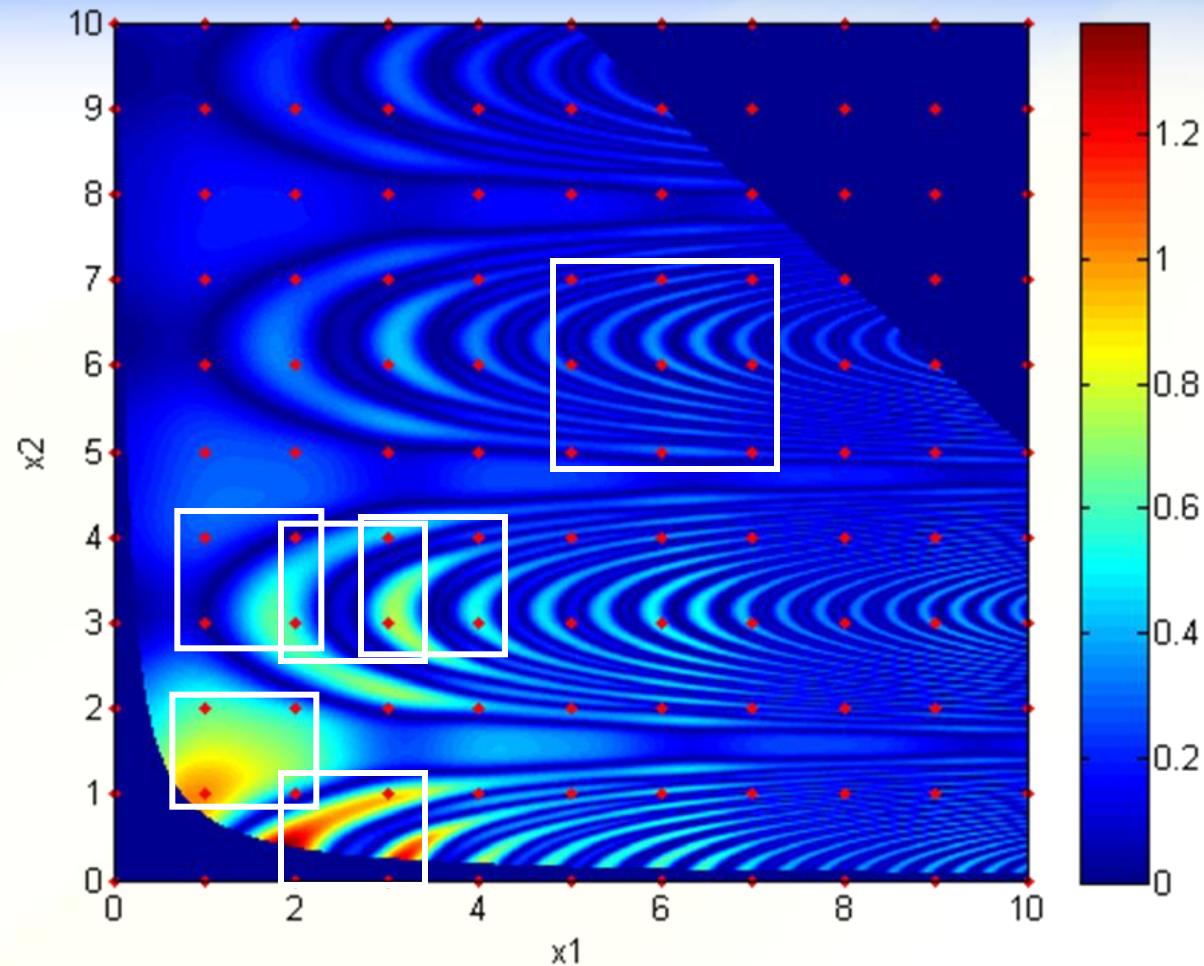
Систематско и случајно претраживање као први корак

- Систематско и случајно претраживање су природни кандидати за први корак алгоритма
- Оба алгоритма генеришу тачке по читавом оптимизационом простору
- Систематско претраживање има предност само за просторе са малим бројем димензија, грубо, до највише три димензије
- Код простора са већим бројем димензија овакав приступ генерисању тачака није погодан због огромног броја потребних итерација
- Стога, случајно претраживање са унiformном расподелом остаје као **једини практичан избор** за вишедимензионалне просторе

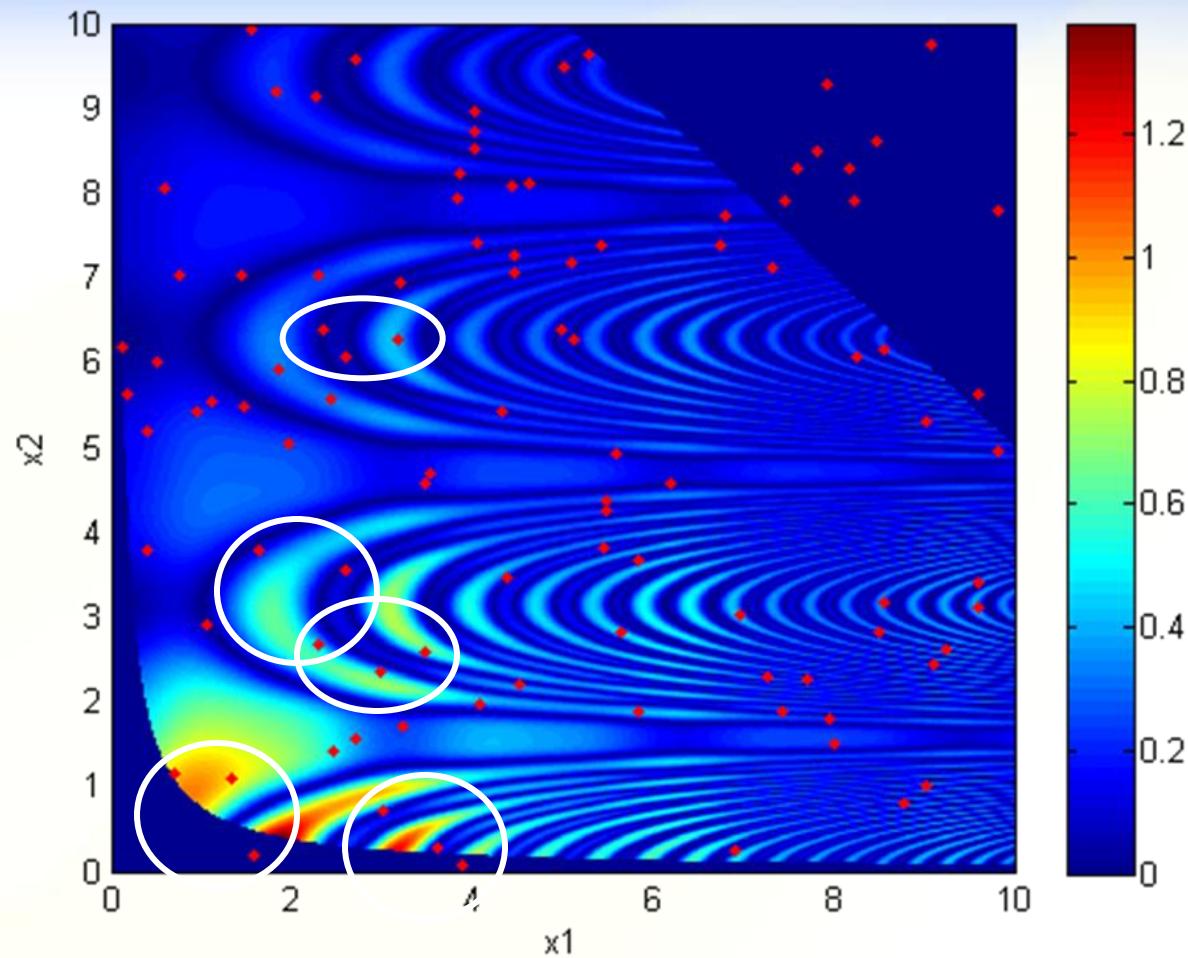
Процена локалних минимума

- Проценом локалних минимума пожељно је пронаћи околине свих минимума са најмањим могућим бројем рачунских операција
- Процена се може извршити на много различитих начина
- Један једноставан начин да се то уради јесте да се за сваку тачку из основног скупа пронађе M најближих тачака и уколико је вредност функције грешке у тој тачки мања од свих M суседних, тачка се прогласи локалним минимумом

Процена локалних минимума систематским претраживањем



Процена локалних минимума случајним претраживањем



Параметри процене локалних минимума

- Број елемената скупа случајних решења N је први параметар овако организованог алгоритма
- Потребно је да буде довољно велик да је могућа процена свих, или бар већине локалних минимума
- Овај број зависи од конкретног проблема оптимизације
- Експериментално је утврђено да би број суседних тачака M за процену локалних минимума требало да буде бар два пута већи од броја димензија простора
- За добро изабране параметре N и M број процењених локалних минимума конвергира стварном броју локалних минимума
- Сложеност обраде $O(N^2)$, у најгорем случају

Поређење метода на аналитички задатим функцијама

- Циљ: сагледати особине и утицај параметара алгоритама
- Стохастички алгоритми:
результат је **случајна променљива**
 - посматрамо средњу вредност најбољег решења после одговарајућег броја итерација
 - може се посматрати и девијација средњег најбољег решења

Сума квадрата: градијентни метод, константан корак

Број димензија	Дужина корака				
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1	9	32	257	2507	25007
2	17	44	367	3549	35364
3	15	54	444	4341	43312
4	18	63	513	5013	50013
5	28	71	574	5605	55924
10	33	104	828	7943	79082
15	62	132	1003	9717	96877
20	56	157	1163	11225	111848

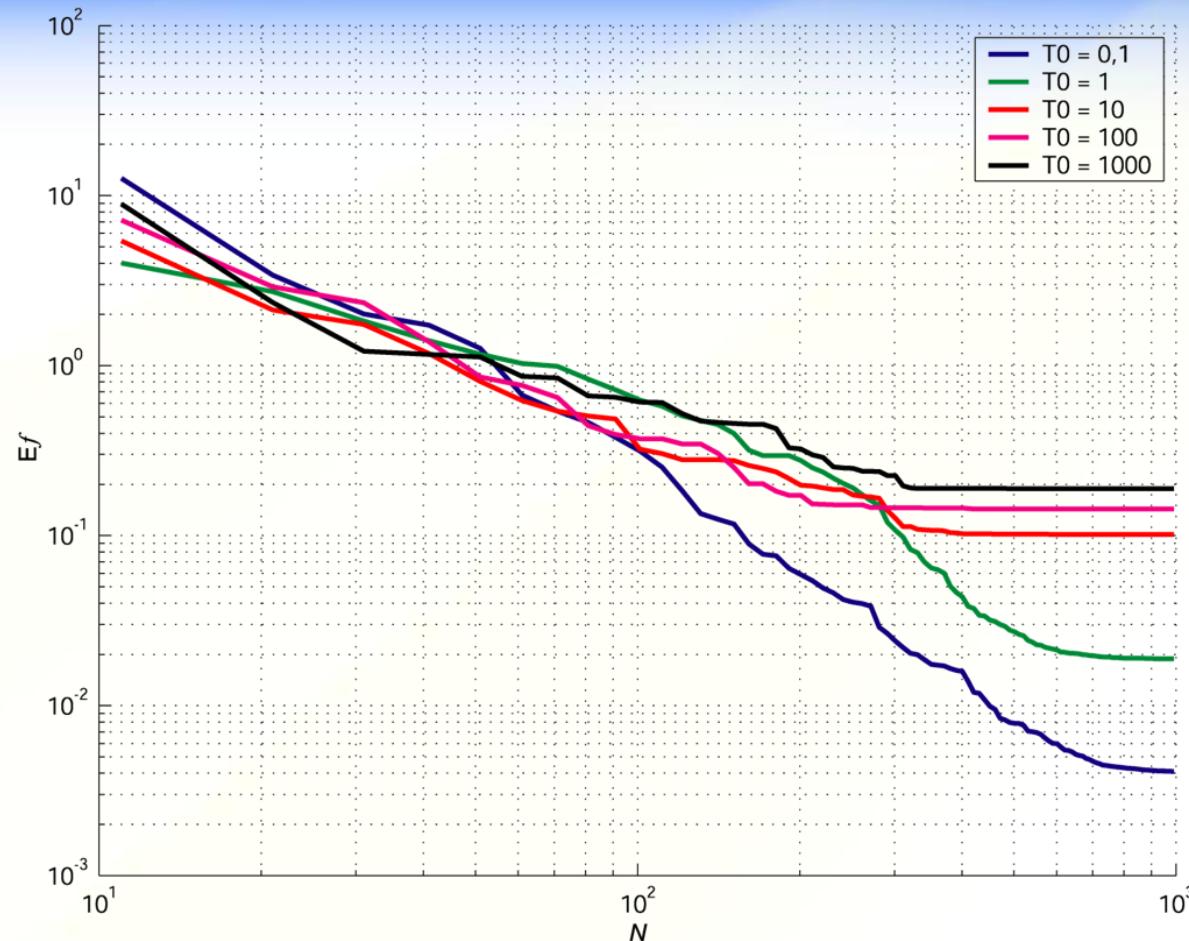
Сума квадрата: градијентни метод, удвојујуће корака

Дим.	Дужина почетног корака									
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
1	9	17	20	36	61	64	100	103	147	157
2	11	24	34	46	60	99	106	120	172	195
3	19	29	42	52	74	107	94	153	173	133
4	22	28	38	58	80	93	131	130	185	188
5	25	37	45	73	102	96	156	175	170	260
10	28	59	93	84	158	123	156	222	216	263
15	56	63	103	151	149	181	177	270	289	267
20	137	99	130	211	228	184	321	317	297	435

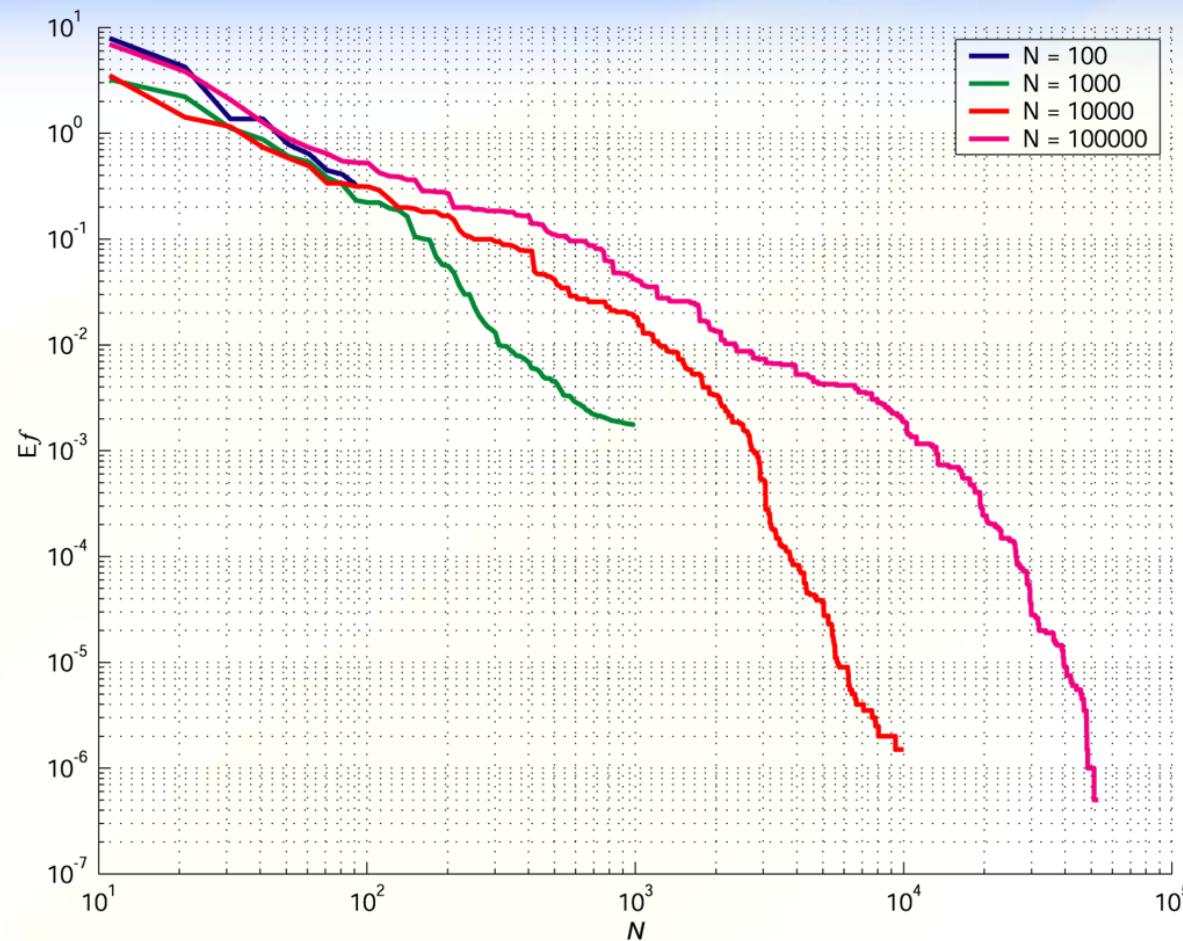
Сума квадрата: симплекс алгоритам

Дим.	Толеранција за функцију грешке									
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
1	6	10	14	18	20	25	29	33	35	39
2	18	26	33	37	45	53	57	68	72	78
3	33	41	52	62	80	89	102	114	122	130
4	40	63	74	90	104	117	133	152	166	187
5	54	63	99	123	159	183	202	241	289	317
10	115	167	209	260	302	345	379	433	466	532
15	203	289	332	411	458	533	600	664	733	804
20	338	459	558	663	787	928	1102	1289	1428	1623

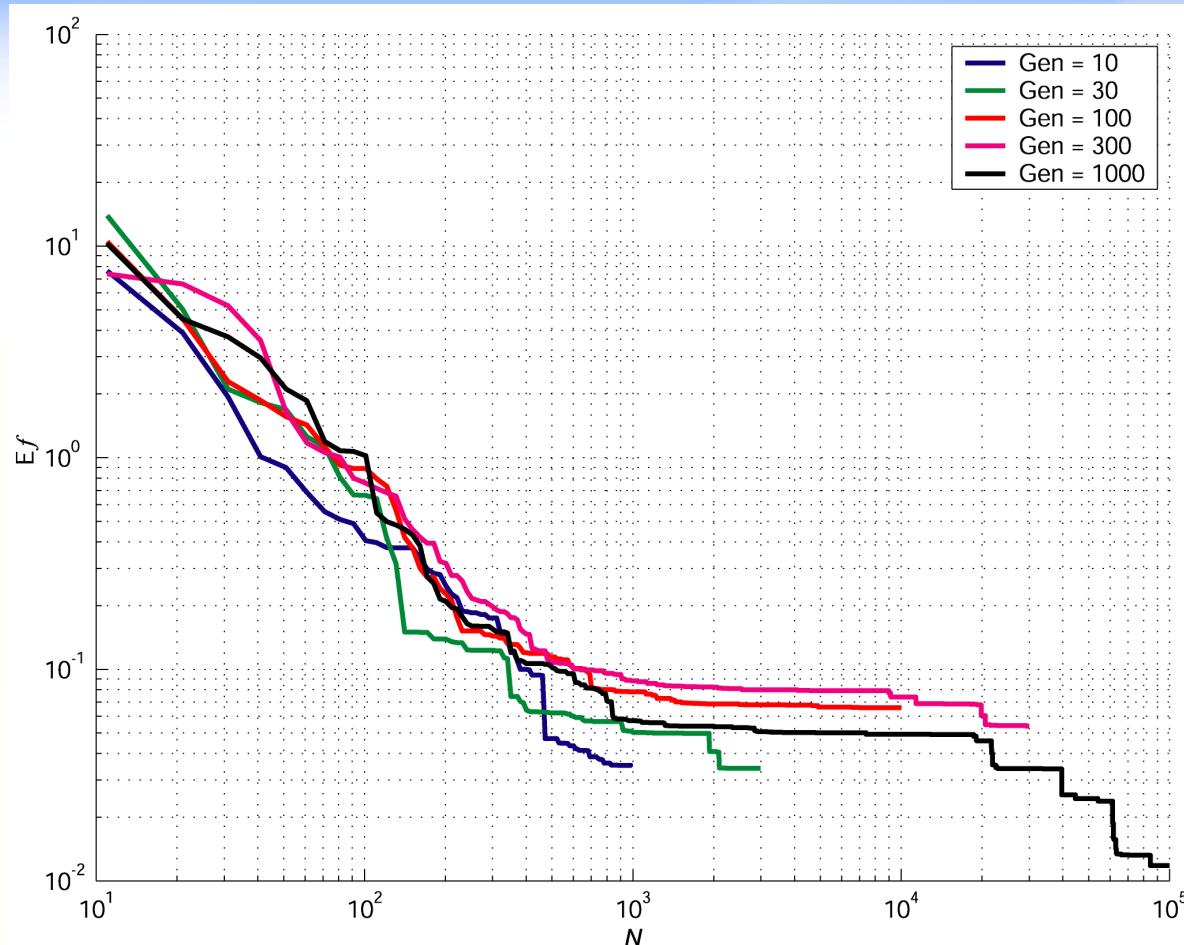
2D Розенброкова функција: СА – почетна температура



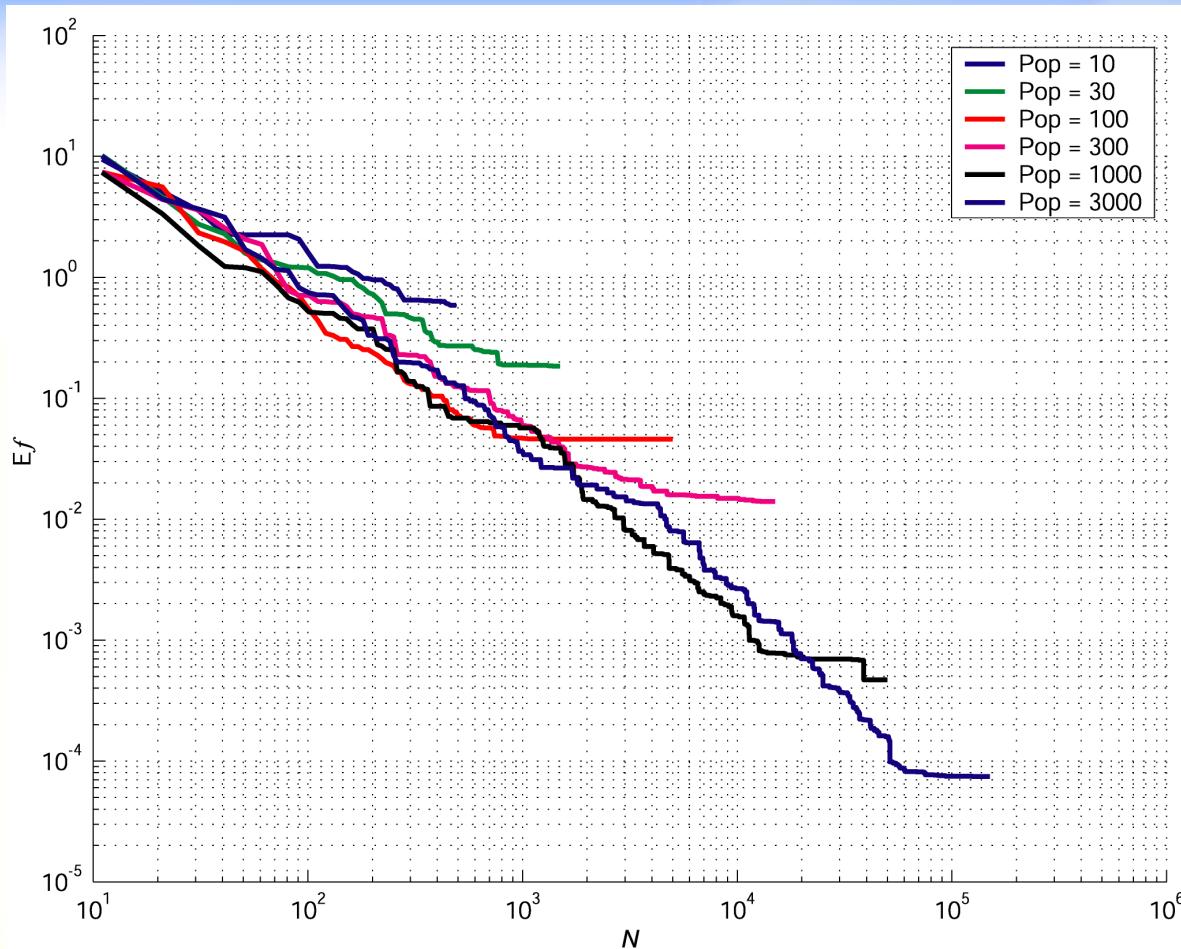
2D Розенброкова функција: СА – број итерација



2D Розенброкова функција: ГА – број генерација



2D Розенброкова функција: ГА – величина популације



Шекелова функција: алгоритми засновани на понављању

- **Рандом(1)+симплекс(100).** Симплекс алгоритам са максимално 100 итерација је стартован из случајно изабране тачке у оптимизационом простору.
- **Рандом(50)+симплекс(50).** Симплекс алгоритам са максимално 50 итерација је стартован из најбоље тачке из скупа од 50 случајно изабраних тачака.
- **ГА(50)+симплекс(50).** Полазна тачка за симплекс алгоритам са највише 50 итерација је проналажена генетичким алгоритмом. Параметри кориштеног генетичког алгоритма су: популација од 10 решења, 3 најбоља решења из генерације формирају наредну генерацију, укупно има 5 генерација, вероватноћа мутације $P_{mut} = 0,1$, вероватноћа укрштања $P_{cross} = 0,8$ и ниједно решење из претходне генерације не прелази у наредну. Прва генерација је бирана на случајан начин.
- **ПЛМ – процена локалних минимума.** Симплекс алгоритам са највише 50 итерација је стартован из свих процењених локалних минимума. Процена је вршена из најближих $M = 8$ тачака.

Шекелова 2D функција са једнаким минимумима (9 мин.)

Алгоритам	Укупан број итерација				
	500	1000	1500	2000	2500
Рандом(1) + симплекс(100)	3,6	6,0	7,8	8,0	8,4
Рандом(50) + симплекс(50)	4,2	6,2	7,8	8,6	9,0
ГА(50) + симплекс(50)	4,0	7,4	8,2	8,4	9,0
ПЛМ	7,8	9,0	9,0	9,0	9,0

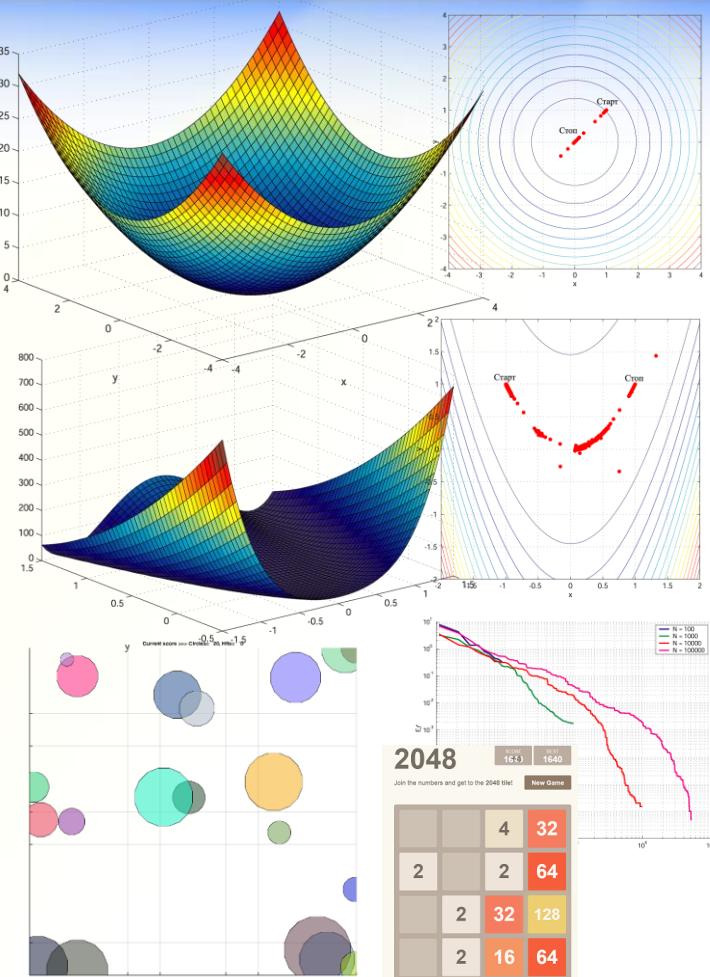
Шекелова 2D функција са неједнаким минимумима (9 мин.)

Алгоритам	Укупан број итерација				
	500	1000	1500	2000	2500
Рандом(1) + симплекс(100)	4,3	6,1	6,8	7,9	8,2
Рандом(50) + симплекс(50)	3,8	4,9	6,1	6,6	6,9
ГА(50) + симплекс(50)	4,0	5,5	6,7	6,8	6,7
ПЛМ	8,3	9,0	9,0	9,0	9,0

Инжењерске оптимизације

- поређење алгоритама -

- Оптимизације комплексних описних функција са више димензија
 - изабрати један оптимизациони алгоритам са курса
 - тест оптимизационе функције http://infinity77.net/global_optimization/test_functions.html#test-functions-index
 - сумирати резултате понашања алгоритма за различит број димензија оптимизационог проблема и различите параметре алгоритма
- Програмирање једноставних рачунарских игара
 - поналажење пута на шаховској табли
 - управљање једноставним играма коришћењем оптимизационих алгоритама



Инжењерске оптимизације - такмичарски проблеми -

The screenshot shows a web browser window with the URL codingcompetitions.withgoogle.com/hashcode/archive. The page displays past problem statements for three competitions: 2016, 2017, and 2018. Each competition section contains two problems: 'Final Round' and 'Qualification Round'. The 2016 section has two problems: 'City Plan' and 'Router placement'. The 2017 section has two problems: 'Self-driving rides' and 'Streaming videos'. The 2018 section has two problems: 'Final Round' and 'Qualification Round'. Each problem statement includes a 'View problem' and 'Download file' link.

IEEE WORLD CONGRESS ON
COMPUTATIONAL INTELLIGENCE
(WCCI)

[http://www.gecad.isep.ipp.pt/
WCCI2018-SG-COMPETITION/](http://www.gecad.isep.ipp.pt/WCCI2018-SG-COMPETITION/)

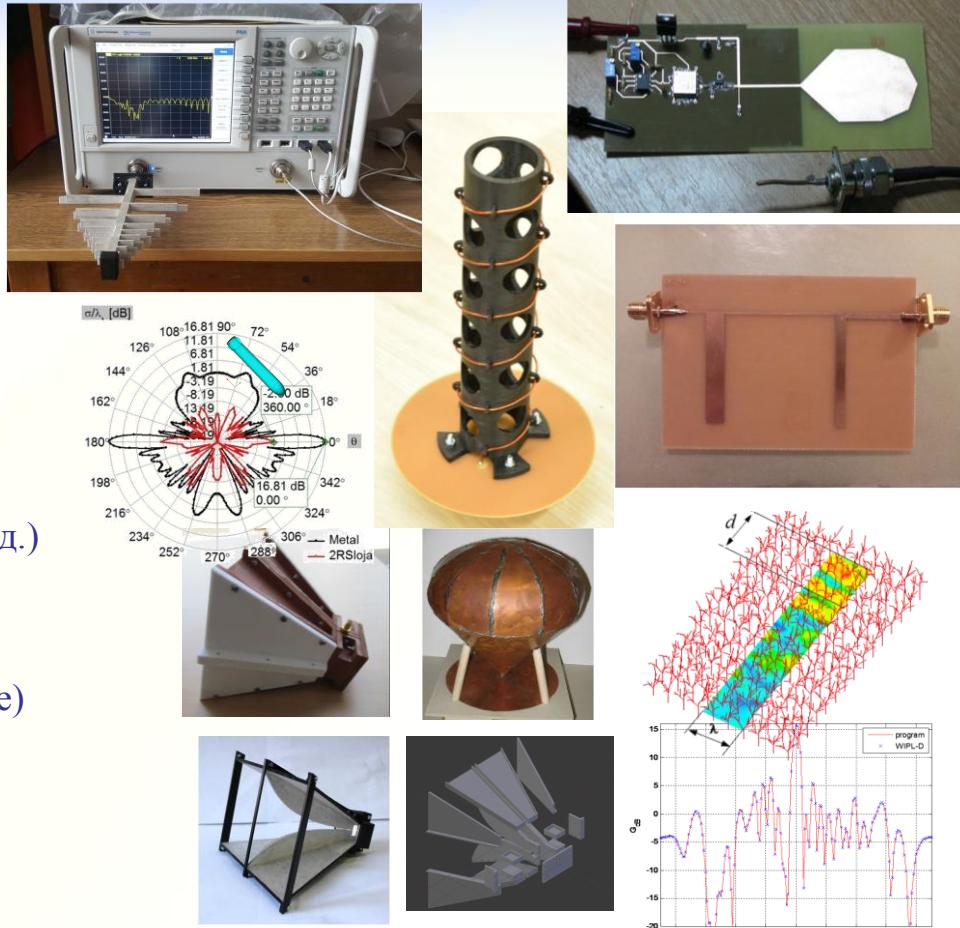
[http://www.gecad.isep.ipp.pt/
ERM-competitions/2020-2/](http://www.gecad.isep.ipp.pt/ERM-competitions/2020-2/)

<https://codingcompetitions.withgoogle.com/hashcode/archive>

Инжењерске оптимизације

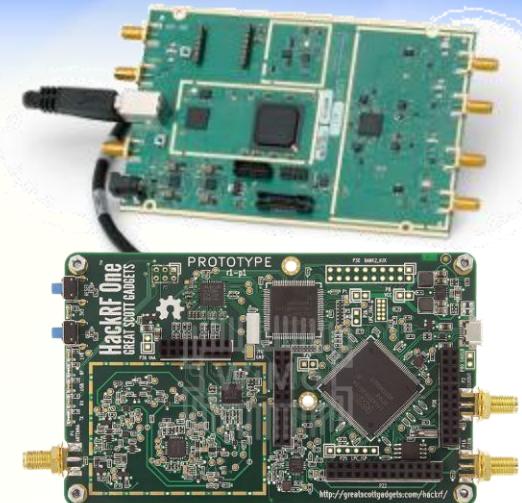
- примењена електромагнетика -

- Оптимизације антена и низова
 - Микротракасте антене
 - Низ микротракастих антена
 - Лог-периодична антена
 - Јаги антена
 - Хеликоидална антена
 - Вивалди антена
 - Широкопојасни монопол
 - Левак антена,...
- Оптимизација микроталасних кола
 - микроталасни филтри
(микротракасти, тракасти, таласоводни, итд.)
 - кола за прилагођење
(ускопојасна или широкопојасна)
 - трансформатори импеданси
(парето фронт вода променљиве импедансе)
- Оптимизација израчунавања нумеричких ЕМ језгара за симулације
- Оптимизовати, направити и премерити
(теме за радове)



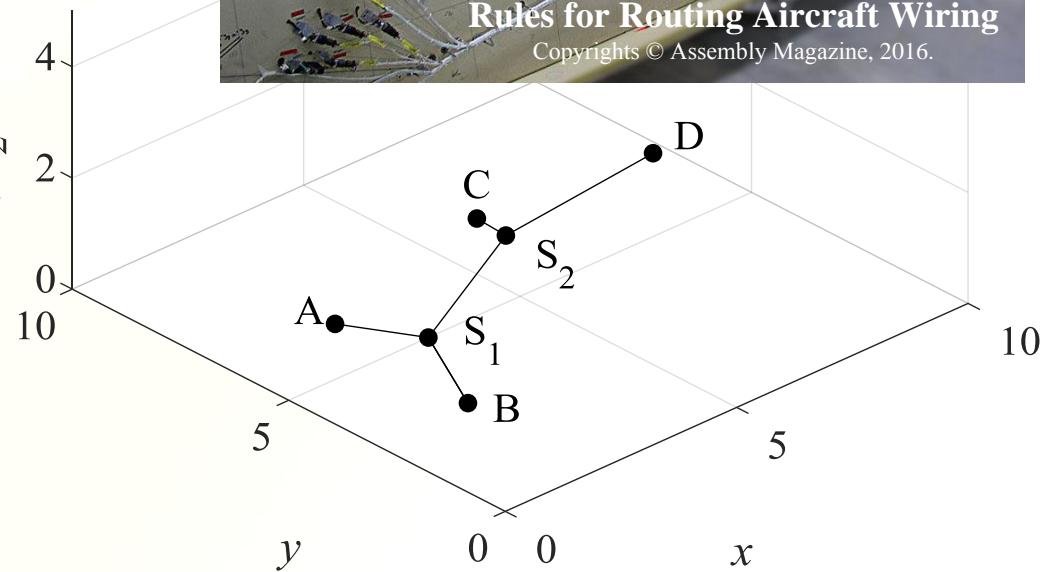
Инжењерске оптимизације - ЕМС, системи, електроника -

- Оптимизација сигнала генератора шума у опсезима WiFi, GPS, GSM, LTE коришћењем SDR
- Оптимално препознавање на основу спектра израченог ЕМ поља снимљеног помоћу SDR или NARDA SRM3006
- Анализа аналогних нелинеарних кола коришћењем оптимизационих алгоритама
 - кола са више диода (усмерачи и исправљачи)
 - кола са транзисторима (појачавачи итд.)
- Оптимизације управљања хардвером и софтвером

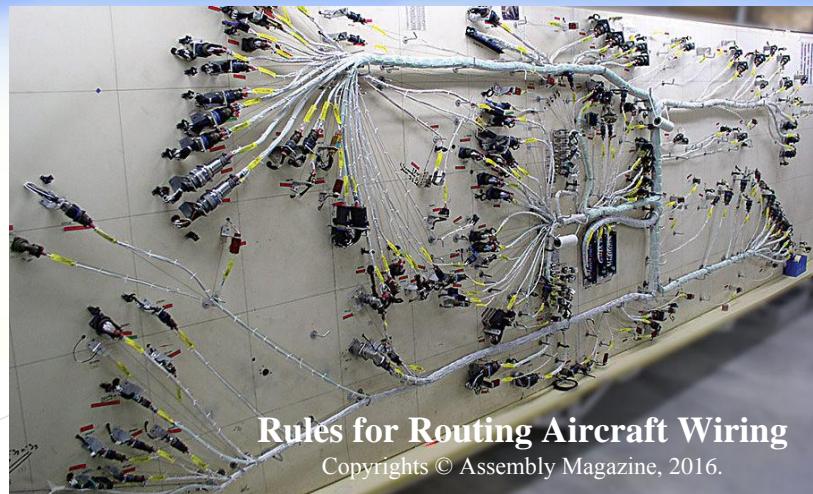


Задатак за вежбе: поставка

- Дате су четири тачке у Декартовом координатном систему:
- $A(1,5,1)$, $B(3,2,0)$, $C(5,7,1)$ и $D(6,3,3)$
- Потребно их је повезати праволинијским путевима тако да **укупна дужина пута буде минимална**

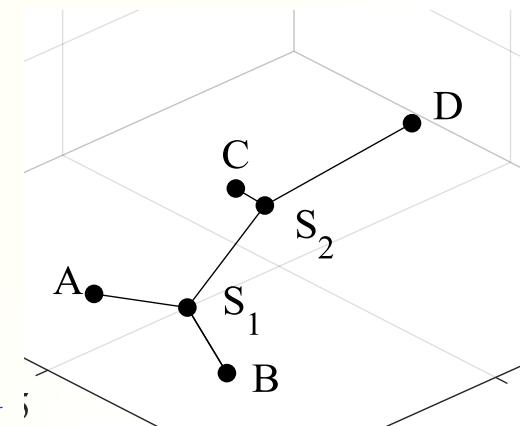


*** Steiner tree problem



Задатак за вежбе: детаљи

- Тачке се повезују додавањем двеју тачака $S_1(x_1, y_1, z_1)$ и $S_2(x_2, y_2, z_2)$
- Тачке A и B повезују се прво са S_1 , а тачке C и D прво са S_2 , као што је приказано на слици
- Одредити и записати у ASCII фајл
 - оптималне координате тачака S_1 и S_2 и ;
 - израчунати минималну дужину пута
- Имплементирати алгоритам за **оптимизацију јатом (PSO)** и искористити га за проналажење оптималних координата



Оптимизациона функција

- До сада смо сматрали да је позната или дефинисана **једна оптимизациона функција**
- То је тачно уколико постоји само **један критеријум оптимизације**
- Како се дефинише оптимизациона функција уколико постоји више критеријума оптимизације?
- Критеријуми могу бити по суштински различитим величинама
 - појачање и прилагођење,
 - струја/напон и физичке димензије,
 - техничке карактеристике и цена,
 - време и остварена добит,
 - број линија кода и функционалност рачунарског програма, итд.

Неки примери оптимизације са више критеријума

- Запаковати што више апликативних опција у што мањи електрични уређај
- Направити што је могуће бржи чип са што је могуће мањим Цуловим губицима
- Написати што је могуће бржи код са што мање линија кода у програмском језику
- Пронаћи оптимално управљање системом са што је могуће једноставнијим алгоритмом
- Урадити испитни задатак за што краће време, а добити највишу оцену

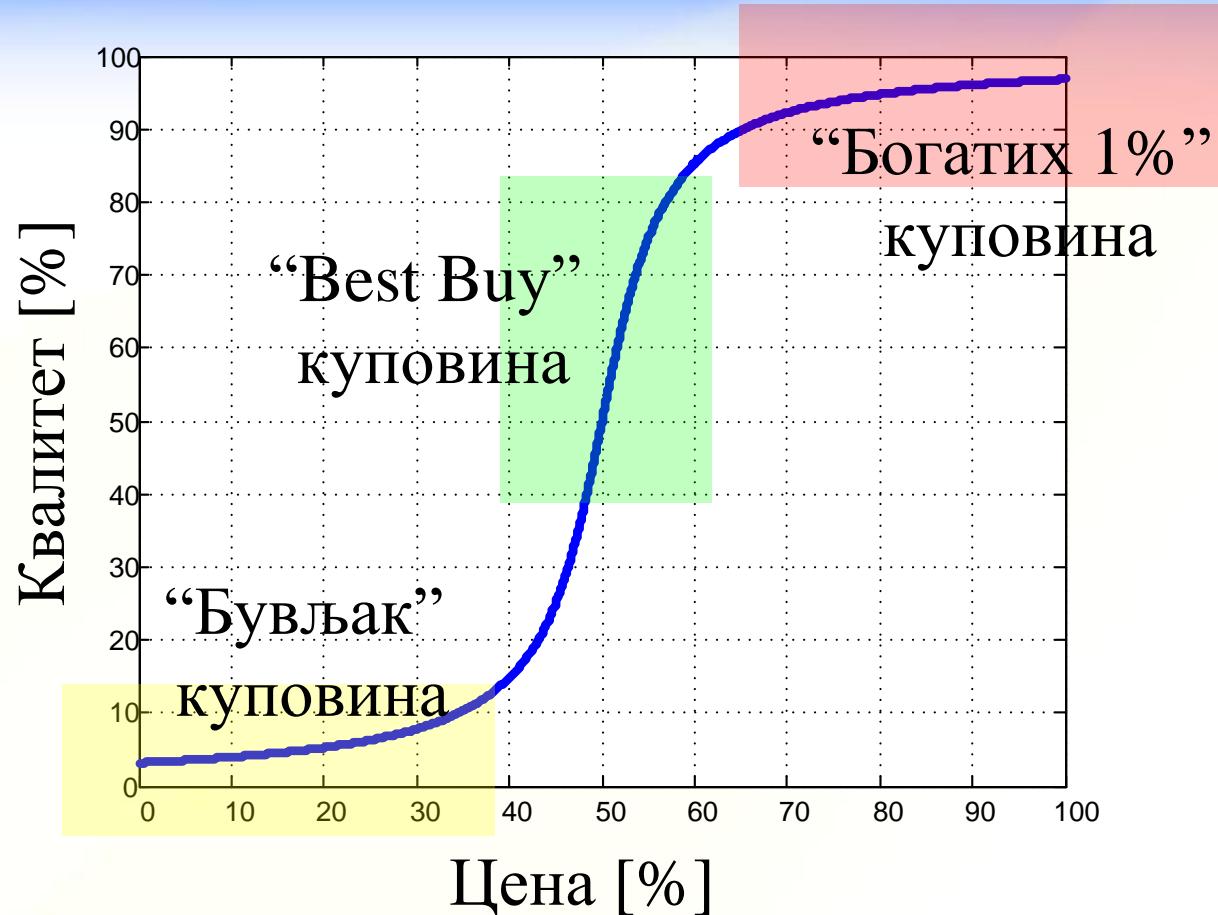
Куповина: илустративни пример супротстављених критеријума

- Посматраћемо пример куповине
- Купац, у принципу, увек води рачуна о два критеријума који су супротстављени
 - Критеријум #1: максимизирати добит (купити што квалитетније или што више)
 - Критеријум #2: минимизирати количину новца
- Да ли у таквим случајевима постоји једно најбоље (оптимално) решење?

Оптимално решење постоји само ако постоји ЈЕДАН критеријум

- Ако фиксирамо други критеријум (количину новца)
 - тада је оптимално решење оно које максимизира добит (први критеријум)
- Ако фиксирамо први критеријум (добит)
 - тада је оптимално решење оно које минимизира дату количину новца (други крит.)
- Уколико дозволимо постојање опсега вредности по оба критеријума, нема једног оптималног решења!

Пример из економије: зависност квалитета од цене



Теорија оптимизације са више критеријума

- Вилфредо Парето (1848–1923)
- Инжењер и економиста
- Бавио се теоријом оптимизације искошишћења ресурса у економији
- Вишекритеријумска оптимизација или парето оптимизација
- Синоними: парето скуп = парето фронт = парето граница

Дефиниција парето фронта

- Посматрамо оптимизациони проблем са G критеријума, f^k , које минимизирамо
- Формално, формиралимо вектор критеријума $f = (f^1, f^2, \dots, f^G)$
- Посматрамо два (могућа) решења: f_1 и f_2
- f_1 доминира f_2 уколико $f_1^i \leq f_2^i$ за свако i и за бар једно i је $f_1^i < f_2^i$, где i иде по свих G критеријума
- Парето фронт је скуп свих решења за која не постоје (друга) доминантна решења (енглески: set of nondominated solutions)
- Парето фронт представља скуп најбољих компромиса
- Најједноставније је проверити да ли за испитивано решење постоји бар једно доминатно решење – ако постоји, онда испитивано решење није тачка парето фронта

Илустрација: скуп од 12 решења проблема са 2 критеријума

Испитивано решење

Доминантно решење

Решење за које не постоји доминантно (nondominated solution)

$$\begin{aligned}f_{01} &= (1.682, 6.283) & f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) & \textcolor{blue}{f_{02}} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) & f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) & f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) & f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) & f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) & f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) & f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) & \textcolor{red}{f_{09}} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) & f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) & f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983) & f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}\textcolor{red}{f_{01}} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

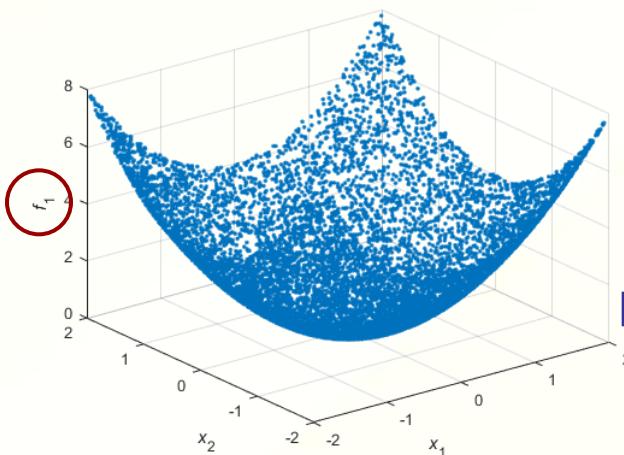
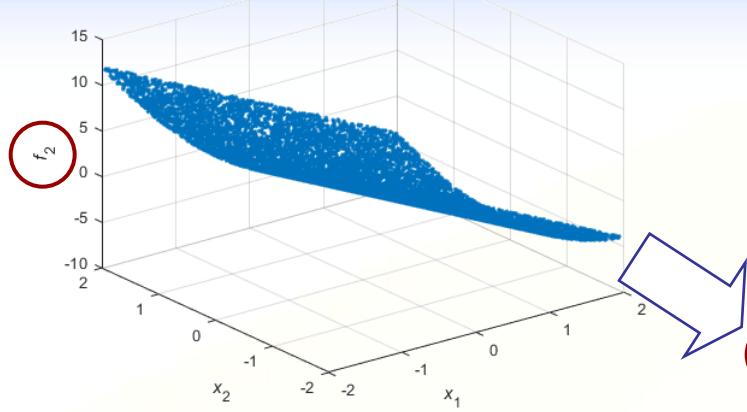
$$\begin{aligned}f_{01} &= (1.682, 6.283) \\f_{02} &= (4.125, 3.751) \\f_{03} &= (1.697, 5.212) \\f_{04} &= (4.699, 6.803) \\f_{05} &= (4.995, 18.992) \\f_{06} &= (0.422, 11.877) \\f_{07} &= (2.416, 9.357) \\f_{08} &= (4.721, 18.455) \\f_{09} &= (3.677, 3.134) \\f_{10} &= (1.436, 5.455) \\f_{11} &= (0.202, 11.480) \\f_{12} &= (0.265, 11.983)\end{aligned}$$

Парето фронт је...

- Геометријско место тачака у простору оптимизационих критеријума чији је број димензија највише $G-1$, где је G број критеријума
 - уколико постоји 2 критеријума парето фронт је крива,
 - уколико постоји 3 критеријума парето фронт је површ,
 - уколико постоји 4 критеријума парето фронт је запремина, итд.
- Уколико постоје несупротстављени критеријуми, број димензија парето фронта се (у принципу) смањује

Домен опт. променљивих и домен критеријума

Домен опт. променљивих



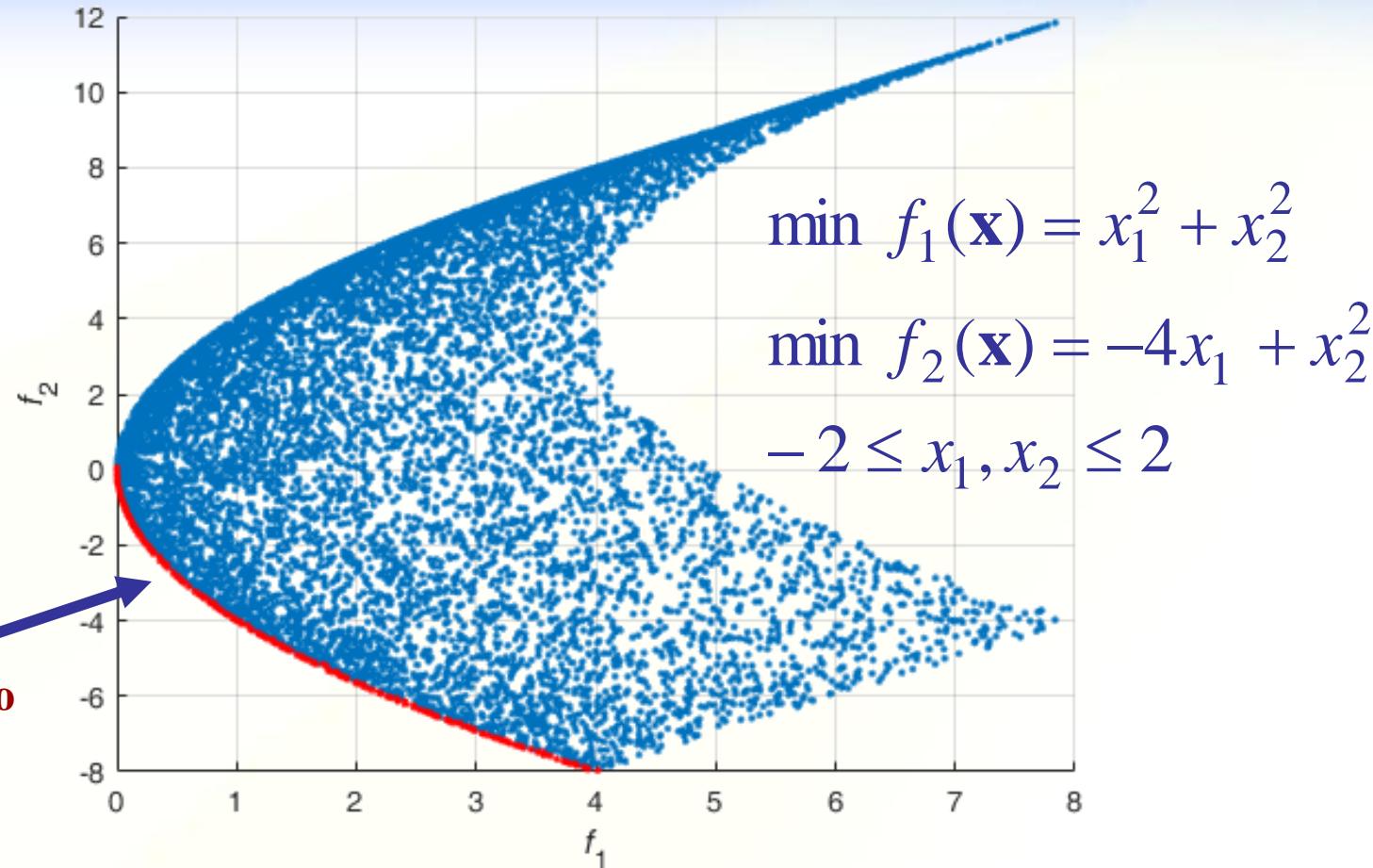
Домен опт. критеријума

$$\mathbf{f}(x_1, x_2) = (f_1, f_2)$$

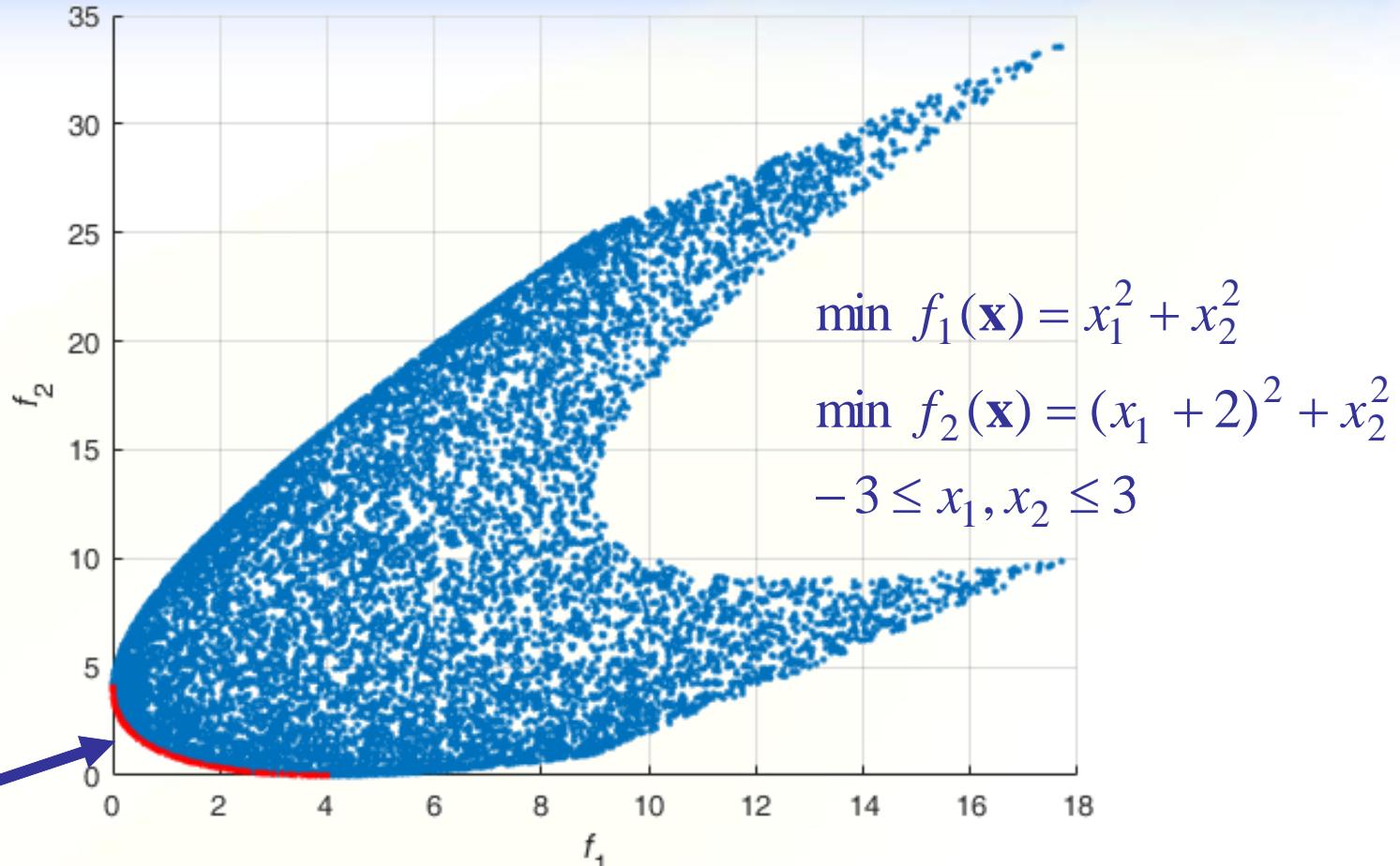
$$\mathbf{f}(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$$

Број критеријума и број променљивих
је различит у оштем случају!

Пример парето фронта #1: два критеријума

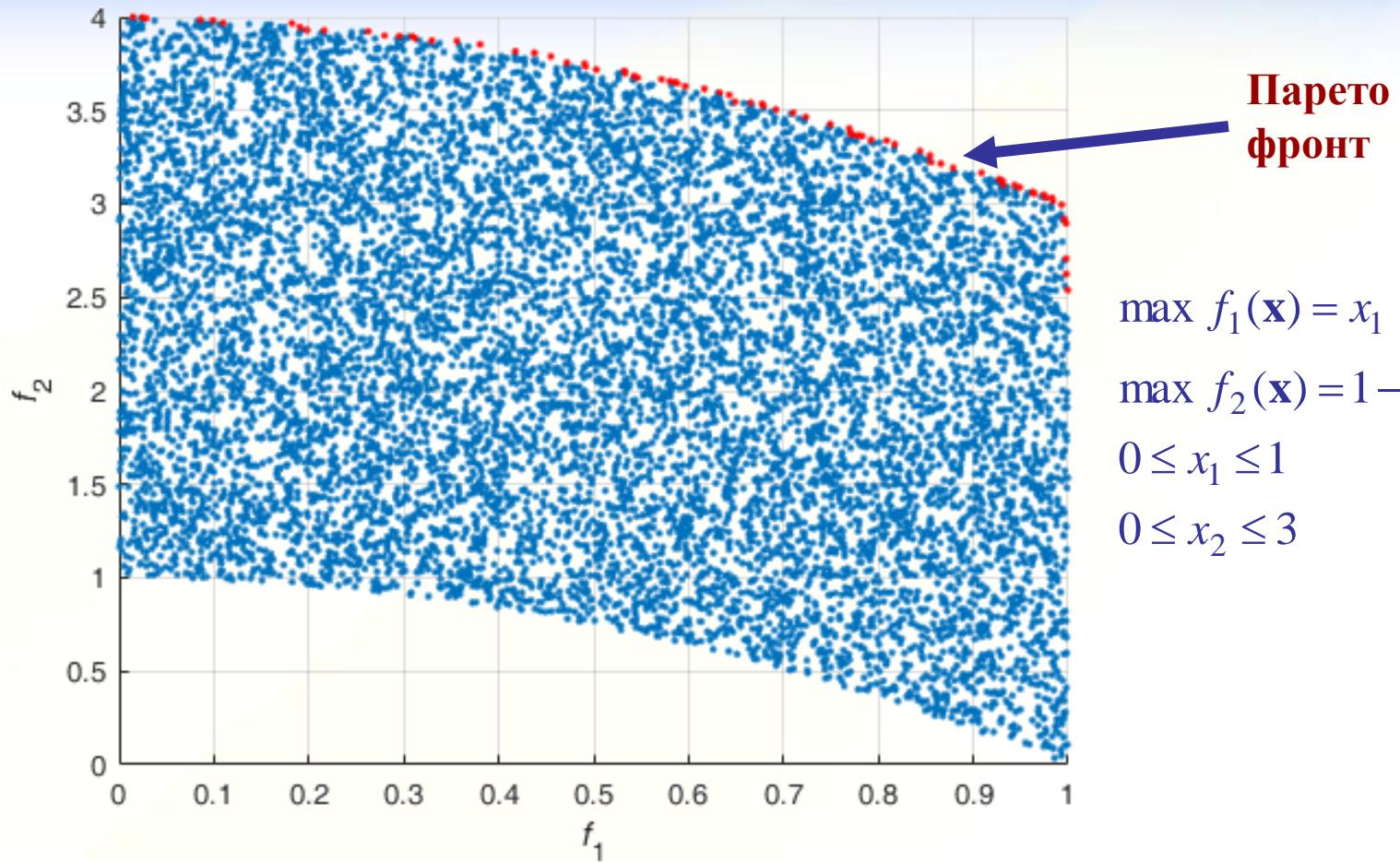


Пример парето фронта #2: два (друга) критеријума

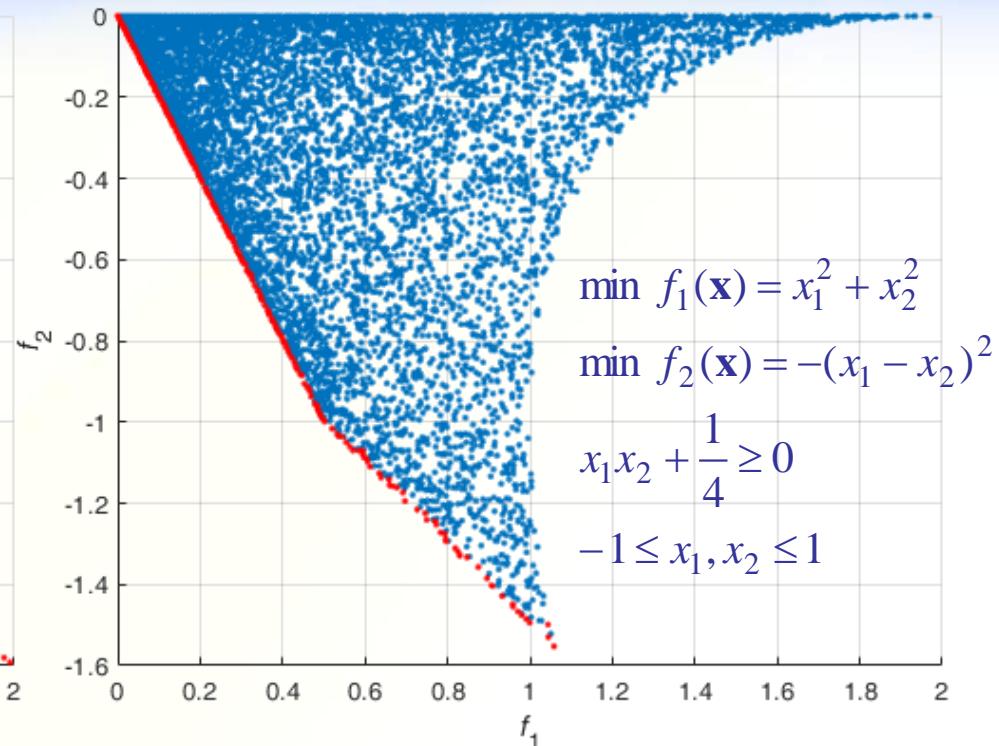
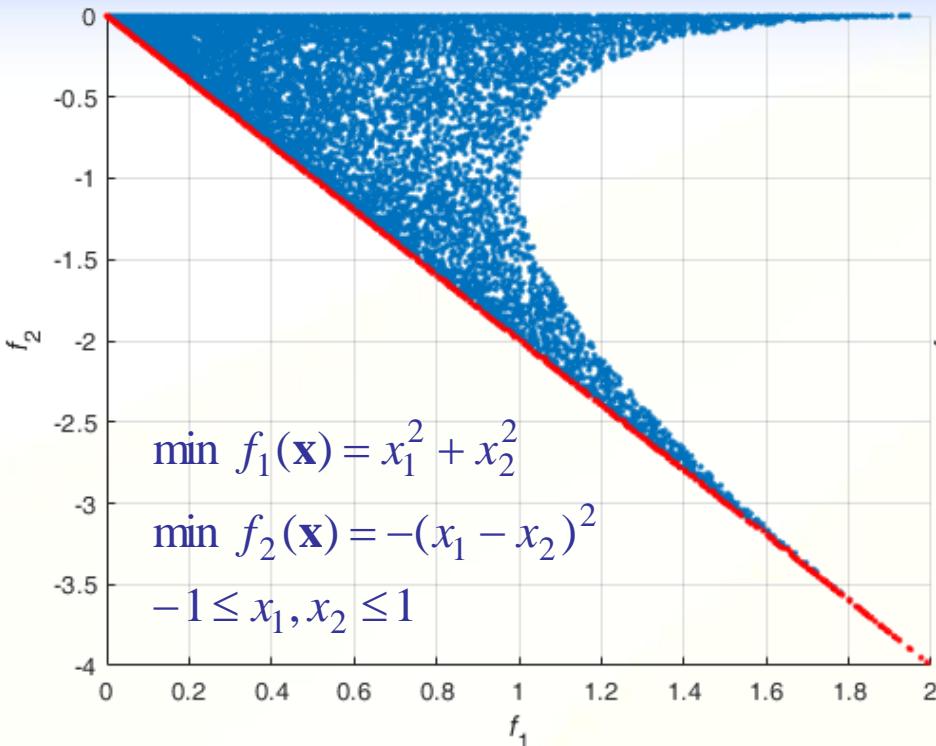


Парето
фронт

Пример парето фронта #3: максимизација



Пример парето фронта #4: додатни услови



- Додатни услови МЕЊАЈУ парето фронт

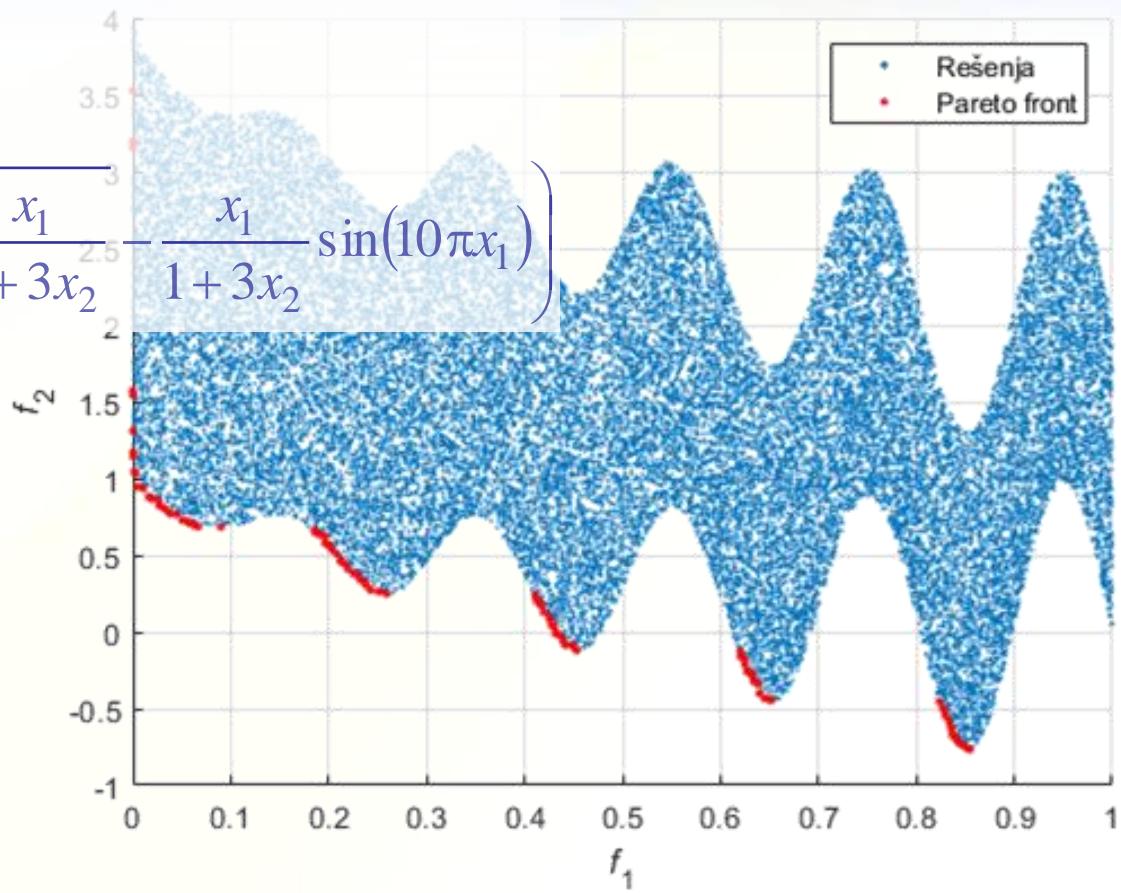
Пример парето фронта #5: фронт може имати прекиде!

$$0 \leq x_1, x_2 \leq 1$$

$$\min f_1(\mathbf{x}) = x_1$$

$$\min f_2(\mathbf{x}) = (1 + 3x_2) \left(1 - \sqrt{\frac{x_1}{1 + 3x_2}} - \frac{x_1}{1 + 3x_2} \sin(10\pi x_1) \right)$$

- Парето фронт
може имати
прекиде!



Пример парето фронта #6: фронт може бити неконвексан

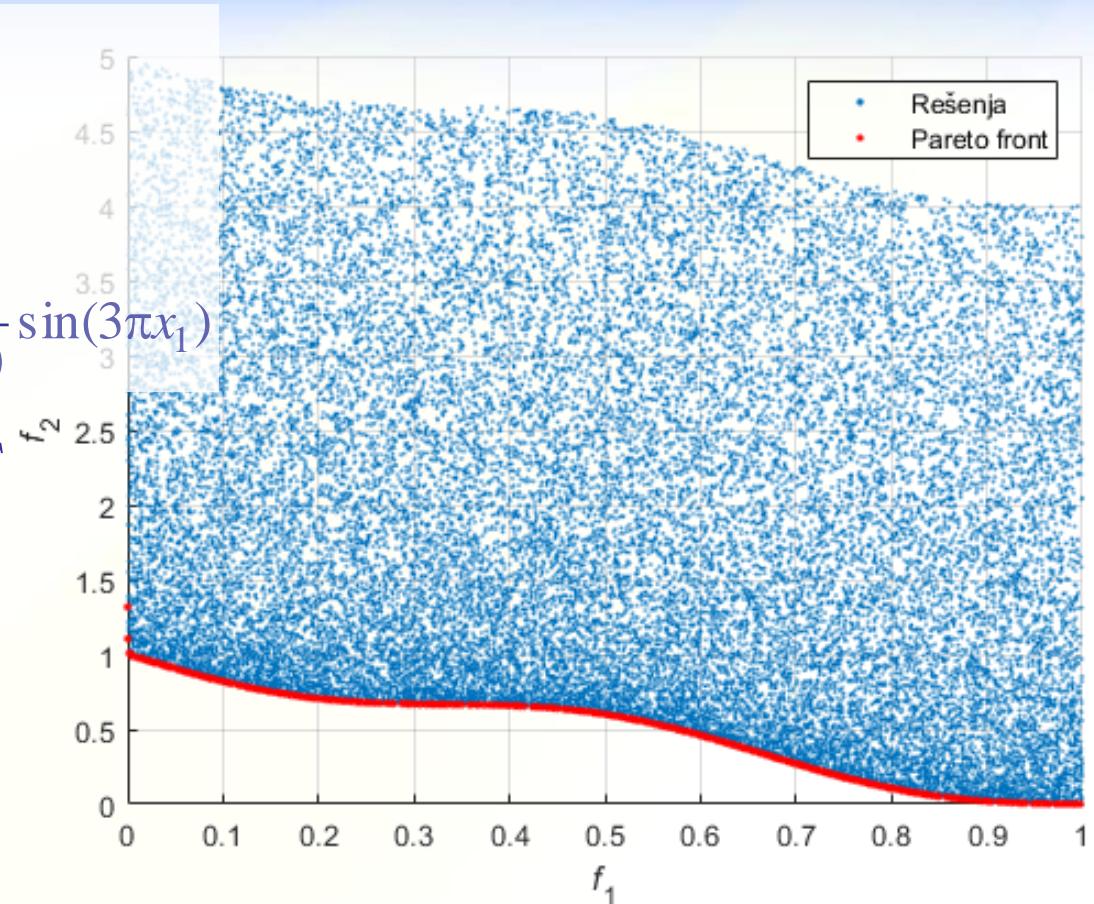
$$0 \leq x_1 \leq 1$$

$$-2 \leq x_2 \leq 2$$

$$\min f_1(\mathbf{x}) = x_1$$

$$\min f_2(\mathbf{x}) = 1 + x_2^2 - x_1 - \frac{1}{10} \sin(3\pi x_1)$$

- Парето фронт
може бити
неконвексан



Пример парето фронта #7: 2D парето фронт

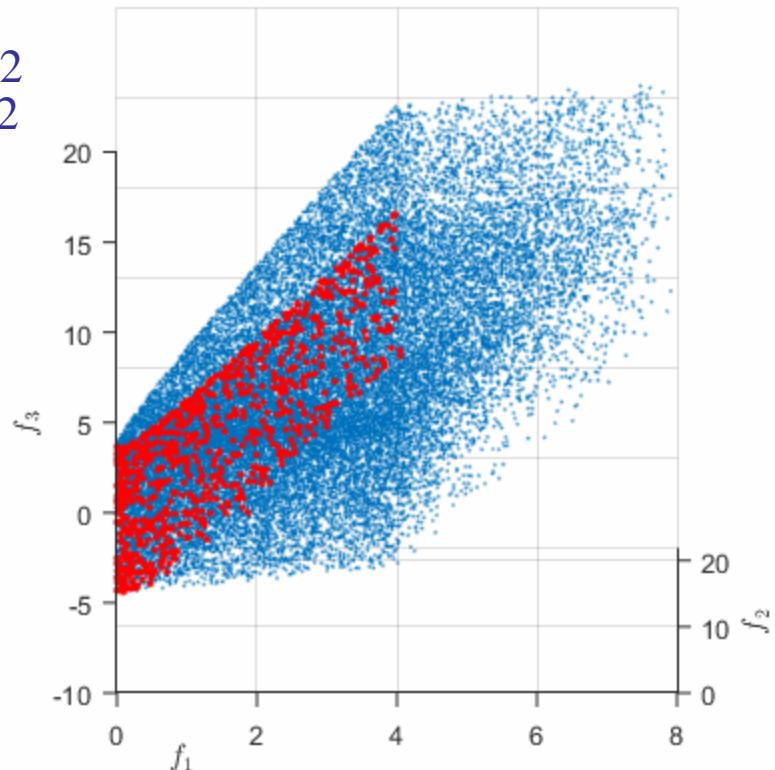
$$\min f_1(x_1, x_2, x_3) = x_1^2 + x_2^2$$

$$\min f_2(x_1, x_2, x_3) = 10 - 4x_1 + x_2^2$$

$$\min f_3(x_1, x_2, x_3) = 4x_1^2 - 2x_3^2$$

$$-2 \leq x_1, x_2, x_3 \leq 2$$

- Оптимизација са три критеријума
- Парето фронт је 2D површ



Проналажење Парето фронта

- Оптимизациони проблеми који се срећу у инжењерској пракси по правилу имају више критеријума
- По (неписаном) правилу критеријуми оптимизације у реалним проблемима су супротстављени
- Теоријски: за све вишекритеријумске проблеме најбоље што можемо је да пронађемо парето фронт
- Како пронаћи парето фронт?
- Како то урадити ефикасно?

Потпуна претрага и провера да ли решење припада парето фронту

- Најједноставнији начин за проналажење парето фронта је провера свих тачака у оптимизационом простору да ли припадају парето фронту или не
- Тачке се могу генерисати
 - потпуним претраживањем
(за опт. просторе са коначним бројем решења)
 - систематским претраживањем
 - случајним претраживањем
- Ово је **изузетно неефикасан** начин, јер у пракси најчешће није могуће потпуно претражити оптимизациони простор
- Ефикаснија решења:
коришћење оптимизационих алгоритама

#1: Комбиновање критеријума са тежинским факторима

- Комбиновање више критеријума у јединствену оптимизациону функцију коришћењем тежинских фактора

$$f = w_1 * f_1 + w_2 * f_2$$

- Овај приступ води ка сабирању величина које су по природи различите
- Примери:
цена и квалитет, брзина процесора и температура, функције електронског уређаја и његова запремина, итд.
- Избор тежинских фактора зависи од конкретног проблема и природе величина које се оптимизују
(инжењер мора сам да их изабере на основу “искуства”)

Теорија и инжењерска пракса

- Теорија: парето фронт се добија ако узмемо све могуће вредности за тежинске факторе и урадимо оптимизацију за сваку комбинацију
- Бесконачно вредности коефицијената
+ бесконачно комбинација тежинских фактора
= непрактичан приступ
- Које тежинске коефицијенте узети у пракси?
- Више покушаја или морамо имати предзнање о појединачним критеријумима
- Генерално, неефикасан начин и зависан од предзнања корисника

#2: Nondominated sorting GA (NSGA)

- Заснива се на стандардном ГА са изменама у делу за рачунање описне функције
- Израчунају се описне функције за сваки критеријум, за свако решење у генерацији
- Проналазе се сва (nondominated solutions) решења која одговарају процењеном парето фронту у текућој генерацији
- Додељује се rank#1 и понавља се док сва решења нису рангирана

NSGA: одређивање парето фронта

- Описна функција на основу које се бирају решења за укрштање се рачуна на основу ранга (први ранг је бољи од осталих)
- Укрштање и мутација се раде на стандардан начин примерен ГА
- Парето фронт се памти и побољшава током генерација (тј. током оптимизације)

Рачунање описне функције за NSGA

- Решења се организују у скупове (нише)

- Број нише:

$$m_i = \sum_{j=1}^{N_{\text{pop}}} Sh(d_{i,j})$$

Растојање у простору критеријума између решења i и j

$$Sh(d_{i,j}) = \begin{cases} 1 - \frac{d_{i,j}}{\sigma_{\text{share}}} & d_{i,j} < \sigma_{\text{share}} \\ 0 & d_{i,j} \geq \sigma_{\text{share}} \end{cases}$$

Полупречник скупа (нише)

- Оптимизациона функција: $F_i = Rank_i \cdot \left(1 + \frac{m_i}{q} \right)$

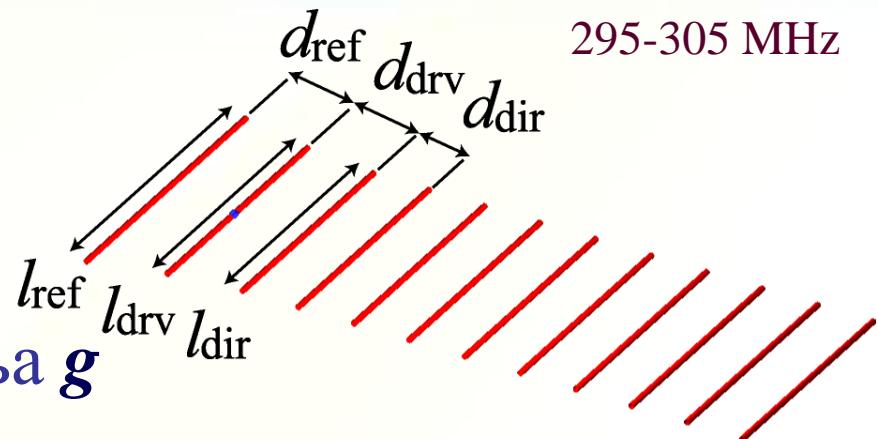
Укупан број решења у скупу (ниши)

#3: Процена локалних минимума за проналажење парето фронта

- Комбиновање свих критеријума у једну описну функцију коришћењем тежинских фактора
- Вишеструко покретање оптимизације за различите комбинације тежинских фактора
- ПЛМ проналази више оптимума у једном покретању
- Та решења увек представљају процену парето фронта у датом покретању
- Памћење парето фронта за вишеструко покретање и одређивање решења која доминирају у фронту

Инжењерски пример: оптимизација антене

- Минимизирати коефицијент рефлексије s_{11}
- Максимизирати усмерено појачање у главном правцу зрачења g
- 6 оптимизационих параметара:
 $0,2 \text{ m} \leq l_{\text{ref}}, l_{\text{drv}}, l_{\text{dir}}, d_{\text{ref}}, d_{\text{drv}}, d_{\text{dir}} \leq 0,8 \text{ m}$
- **Парето фронт:** колико појачање морамо да “жртујемо” да бисмо истовремено имали и добро прилагођену антenu?



Поставка оптимизације

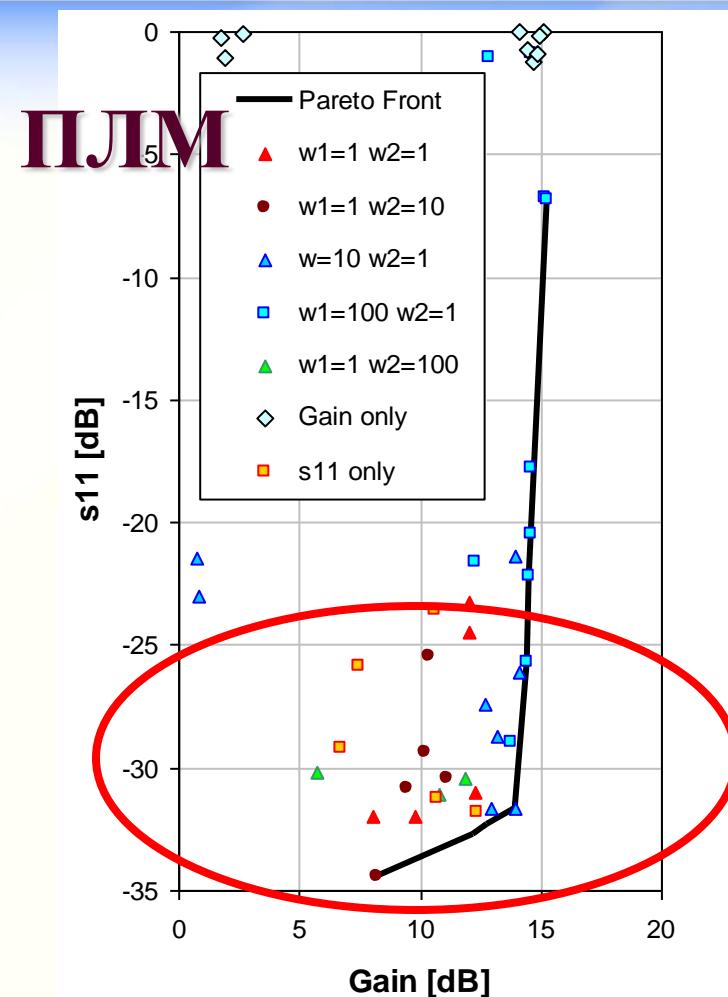
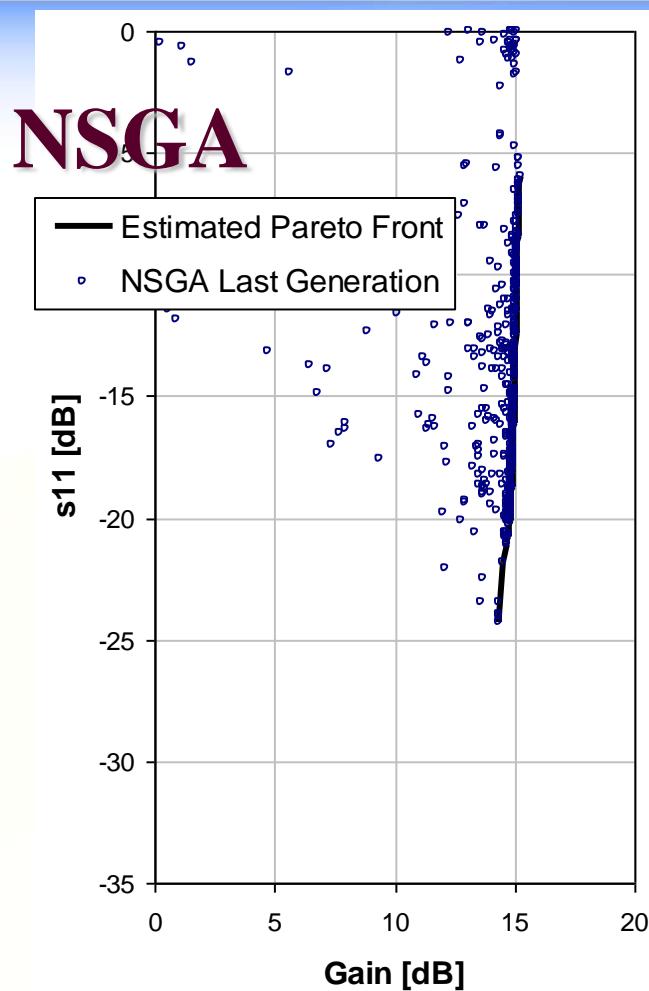
- **NSGA:** 500 решења, 40 генерација, 20 000 итерација укупно
- **ПЛМ:** 7 комбинација тежинских фактора w_1, w_2

w_1	1	1	10	1	100	0	1
w_2	1	10	1	100	1	1	0

~3 000 итерација за једну комбинацију
20 000 итерација укупно

- 20 независних покретања
како би се проценило понашање алгоритама у статистичком смислу

Процењени парето фронт: резултати су комплементарни!



Предности и мане NSGA и ПЛМ алгоритама

- NSGA:
 - нема тежинских фактора
 - парето фронт се проналази у једном покретању
 - спора конвергенција (дуготрајна оптимизација)
 - потенцијални проблеми са оштрим минимумима
- ПЛМ:
 - бржа конвергенција од NSGA
 - мање зависан од тежинских коефицијената од вишеструког покретања основних опт. алгоритама
 - избор тежинских фактора зависи од проблема

Друге опције за проналажење парето фронта

- Постоје и други алгоритми за проналажење парето фронта
- Практично сви “еволутивни алгоритми” имају верзије за вишекритеријумску оптимизацију
- У литератури се често помиње и NSGA2
- Генерално, проналажење парето фронта је знатно сложенији оптимизациони проблем од оптимизације по једном критеријуму
- Пре почетка оптимизације увек проверити да ли се ради о оптимизацији са једним или више критеријума и на основу тога организовати оптимизацију

Алгоритми за проналажење парето фронта

- MOGA: Multi-Objective Genetic Algorithm
- NPGA: Niched Pareto Genetic Algorithm
- Predator-Prey Evolution Strategy
- Distributed Reinforced Learning Approach
 - агентима додељене различите опт. функције
- NCGA: Neighborhood Constrained GA
 - критеријуми се претварају у ограничења
- Nesh GA
 - поправљање једног по једног критеријума
(док се остали не диражуј)
- MOEA: Multi-Objective Evolutionary Algorithms

Отворена питања за проблеме са више критеријума

- Како онемогућити груписање решења у једном делу оптимизационог простора?
- Како се перформансе алгоритама за одређивање парето фронта скалирају са повећањем броја критеријума?
- Како обезбедити конвергенцију алгоритма ка свим деловима парето фронта?
- Како поредити перформансе алгоритама?
- Како приказати парето фронт?

Парето оптималност и компромиси

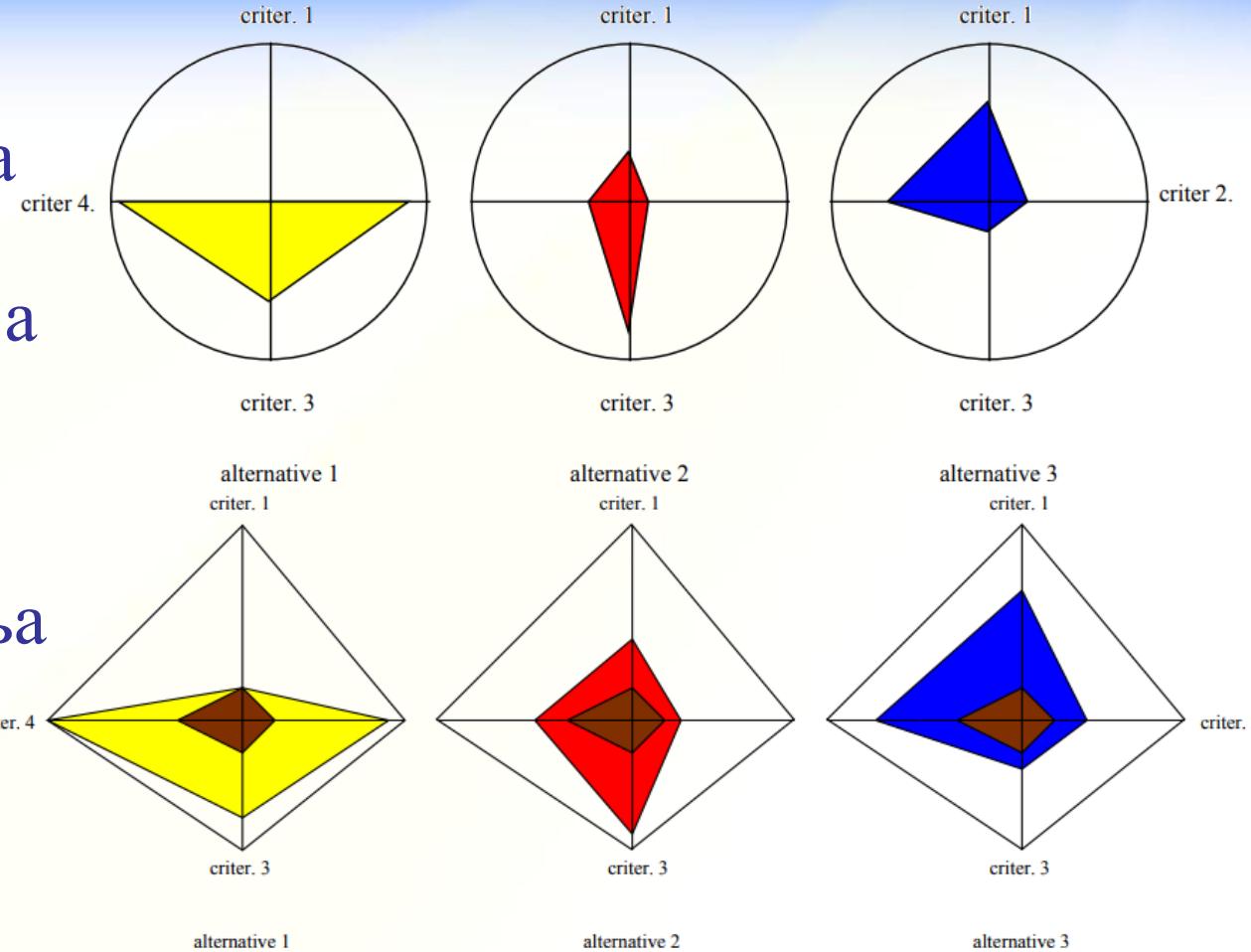
- Већина људи, ван математике, инжењерства и економије НИЈЕ чула за појам парето оптималности
- Међутим, најбољи могући компромиси се јасно (интуитивно) разумеју
 - A good compromise is like a good sentence, or a good piece of music. Everybody can recognize it. They say: 'Huh. It works. It makes sense.'
- Одређивање једне тачке парето фронта је једнако решавању једног проблема са једним критеријумом
- Одређивање парето фронта у већини инжењерских апликација је данас на граници (или изван) доступних рачунарских ресурса!

Како искористити парето фронт?

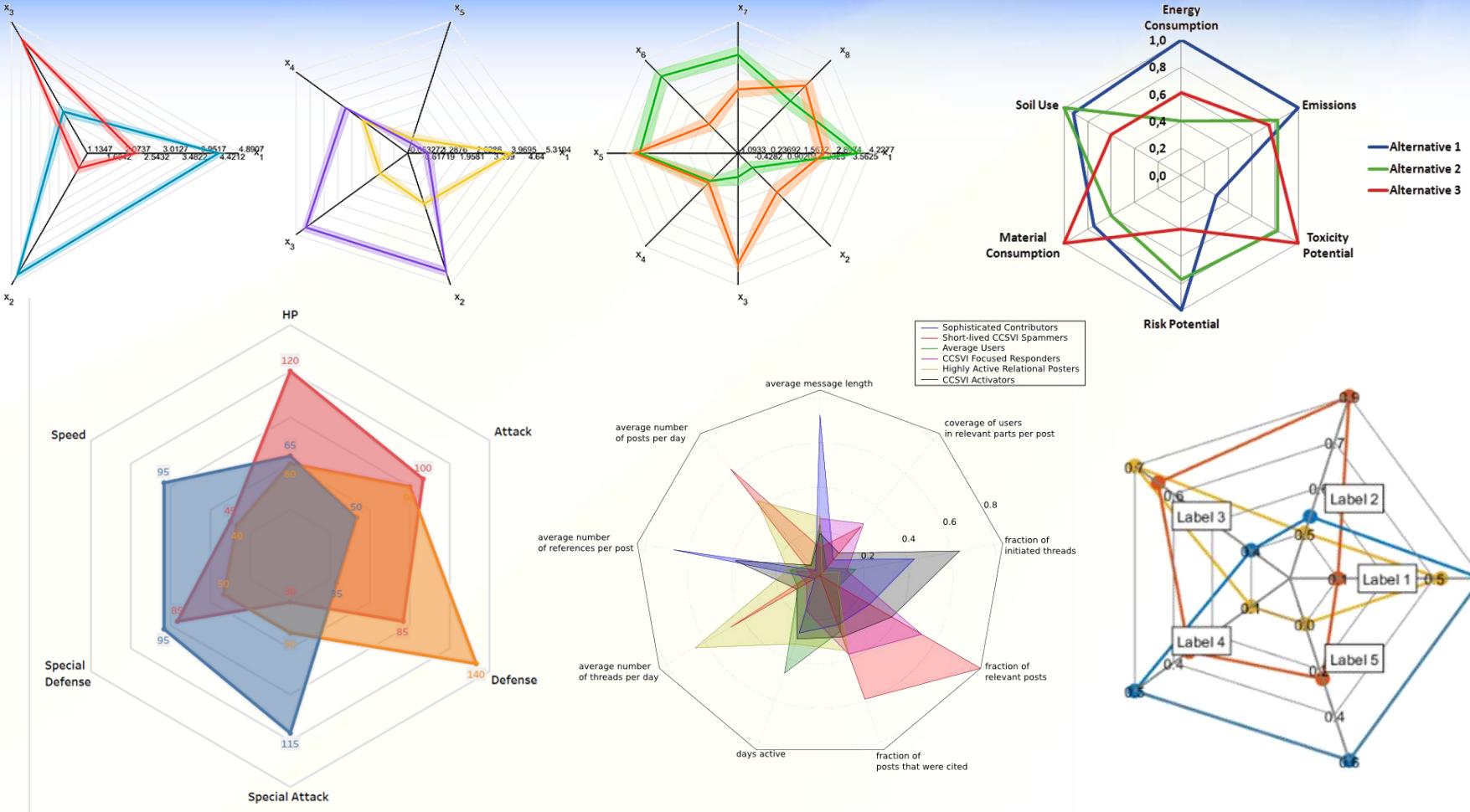
- Уколико је познат **тачан парето фронт** или је позната “најбоља” процена парето фронта имамо увид у скуп свих најбољих компромиса које можемо да направимо за задати проблем
- Шта даље?
- Како одабрати једно од компромисних решења?
- Избор “најбољег” компромиса зависи од онога ко је у позицији да доноси одлуку
- Теорија игара и аутоматско одлучивање

Како визуелно приказати најбоље могуће компромисе?

- Пример:
4 критеријума
- У питању је
максимизација
по сваком
критеријуму
- Графички
приказ решења
 - Radar chart
 - Spider chart
 - Star chart



Примери графичког приказа тачака парето фронта



Задатак за вежбе

- Проценити парето фронт вишекритеријумске оптимизације функција двеју променљивих
- Два проблема: без и са додатних услова

$$\min f_1(\mathbf{x}) = 2x_1^2 + x_2^2$$

$$\min f_1(\mathbf{x}) = 2x_1^2 + x_2^2$$

$$\min f_2(\mathbf{x}) = -(x_1 - x_2)^2$$

$$\min f_2(\mathbf{x}) = -(x_1 - x_2)^2$$

$$-1 \leq x_1, x_2 \leq 1$$

$$x_1 x_2 + \frac{1}{4} \geq 0$$

$$-1 \leq x_1, x_2 \leq 1$$

Задатак детаљи

- Генерисати 10000 тачака случајно изабраних из домена оптимизационих променљивих са униформном расподелом
- Израчунати вредности оптимизационих функција у свим тачкама
- Пронаћи доминантна решења
- Нацртати:
 - све генерисане тачке у домену критеријума и
 - јасно означити парето фронт (доминантна решења)

Бонус задатак

- Текст и подаци из задатка са трећих вежби (одређивање стабла минималног збира тежина грана са “пеналима” за гранања)
- Одредити парето фронт (два критеријума)
 - Минимизирати збир тежина (растојања) грана стабла
 - Минимизирати број гранања у стаблу
(ако три или више грана полазе из истог чвора,
додати број грана минус два у оптимизациону функцију)
- Потпуно претражити оптимизациони простор и пронаћи све тачке парето фронта (nondominated sorting)
- Записати све пронађене тачке парето фронта у ASCII фајл заједно са вредностима опт. функција за критеријуме

Оптимизациони алгоритми са инжењерским проблемима

- Циљ: сагледати особине оптимизационих алгоритама када су примењени на инжењерске оптимизационе проблеме
- Примери
 - Јаги антена
 - Трансверзални антенски низ
 - Неуниформна хеликоида
 - Одређивање поља око процесора за различите инструкције које се извршавају
- Кључна тачка: оптимизациона функција
- NLP и SAT проблеми

Пример #1: Јаги антена

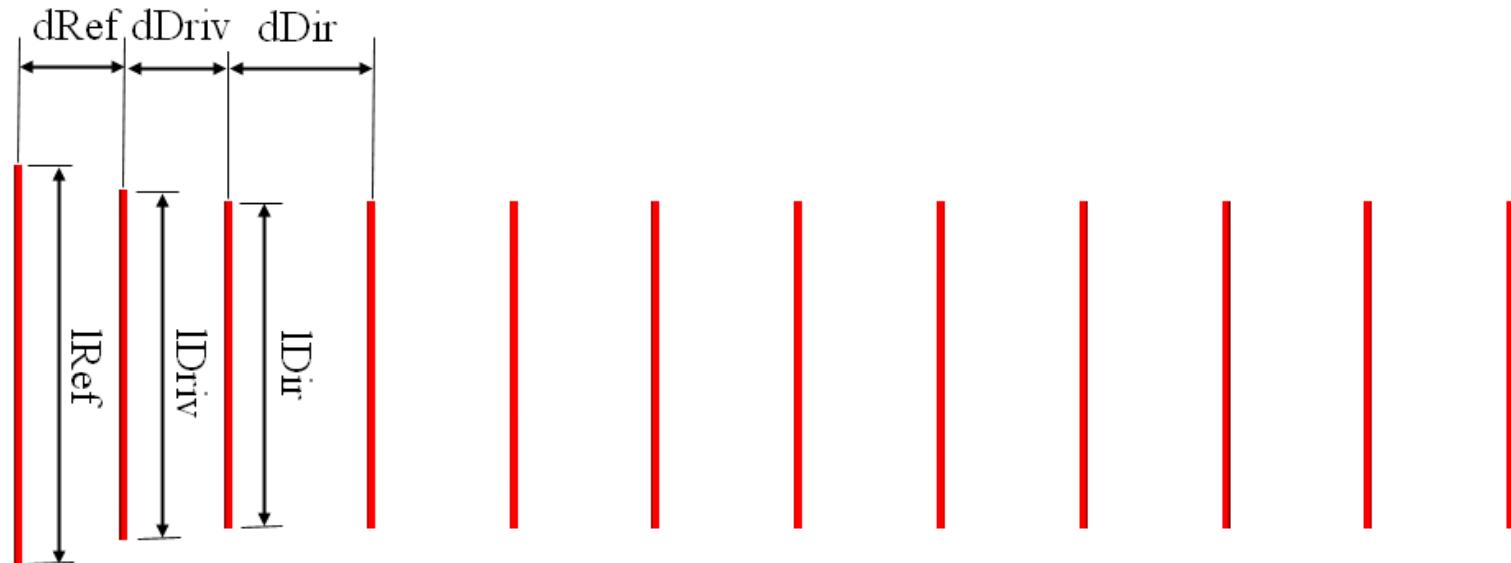
- Јаги антена се састоји из неколико линеарних дипола
- Само један дипол се директно напаја
- Остали диполи раде као паразитни радијатори чије струје су индуковане због међусобне спреге
- Антене овога типа се користе у опсегу 3 MHz - 3000 MHz

Модел јаги антене коришћен за оптимизацију

- Шест оптимизационих параметара:
 - l_{ref} – дужина рефлектора,
 - l_{drv} – дужина напајаног елемента,
 - l_{dir} – дужина сваког од директора (сви директори су једнаке дужине),
 - d_{ref} – растојање између рефлектора и напајаног елемента,
 - d_{drv} – растојање између рефлектора и првог директора,
 - d_{dir} – растојање између суседних директора

Оптимизациони простор и модел

- Оптимизациони простор:
параметри су мењани у
границама од 0,2 м до 0,8 м



Описна функција

- Описна функција
$$f = \sqrt{\frac{1}{n} \sum_{k=1}^n (gain - G_0)^2}$$
- n је број фреквенција у којима је рачунат добитак (*gain*) антене у децибелима
- У свим приказаним резултатима $G_0 = 20$ dB
- Добитак је рачунат на $n = 5$ еквидистантних участаности
- Опсег участаности 295-305 MHz
- За $G_0 = 0$ f је средњи добитак антене у посматраном опсегу участаности

Поставка за поређење алгоритама

- За оптимизацију је коришћено шест различитих алгоритама
- Сваки метод је изнова покретан до укупно 10 000 итерација
- Додатним нумеричким експериментима је показано да је усвојени максималан број итерација довољан за поуздану статистику

Алгоритми који су упоређени

- Случајно претраживање
- Понављање градијентног алгоритма са случајно изабраним почетним решењем
- Понављање симплекс алгоритма са случајно изабраним почетним решењем
- Најбоље решење из 300 случајно изабраних као почетно решење за симплекс
- Хибридни ГА+симплекс
- Процена локалних минимума

Случајно претраживање

- Униформно шестодимензионално случајно претраживање
- Метод пружа увид у расподелу добитака јаги антене у оптимизационом простору
- Средње растојање међу решењима би требало да буде једнако теоријској вредности за униформну расподелу тачака (теоријска вредност 0,6 м)

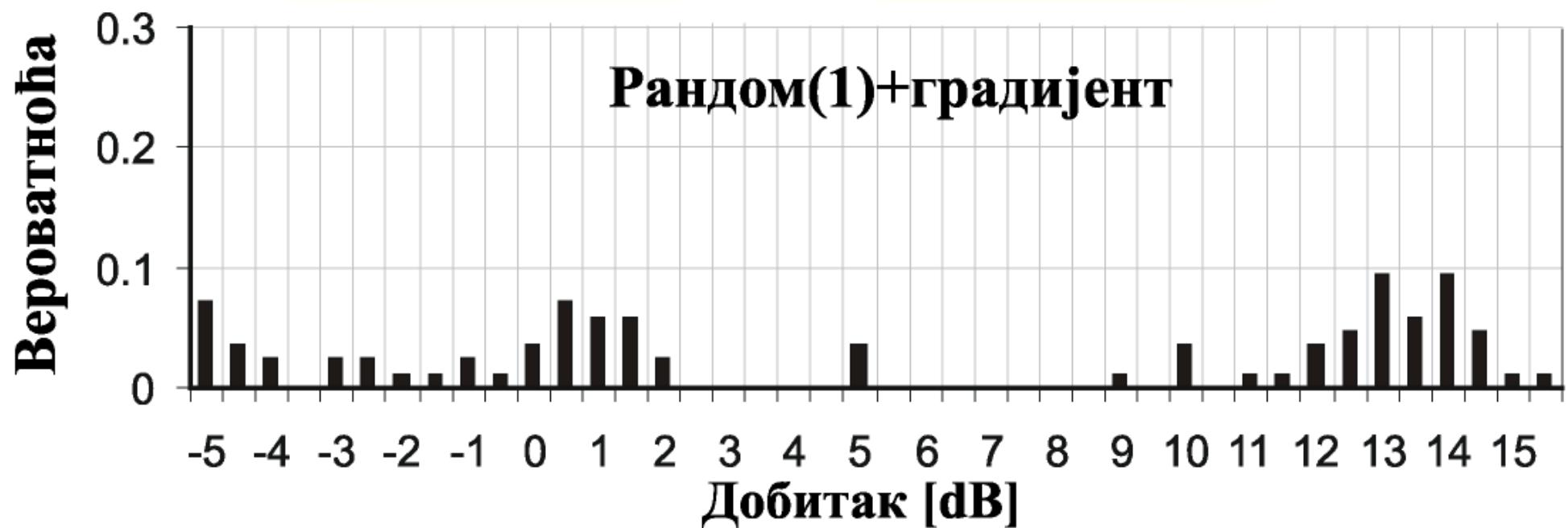
Случајно претраживање: резултати

- 25% решења: средњи добитак мањи од -5 dB
- Ниједно решење са добитком већим од 14 dB



Рандом(1) + градијент

- Релативни корак за процену градијента 0,05% координате оптимизационог простора



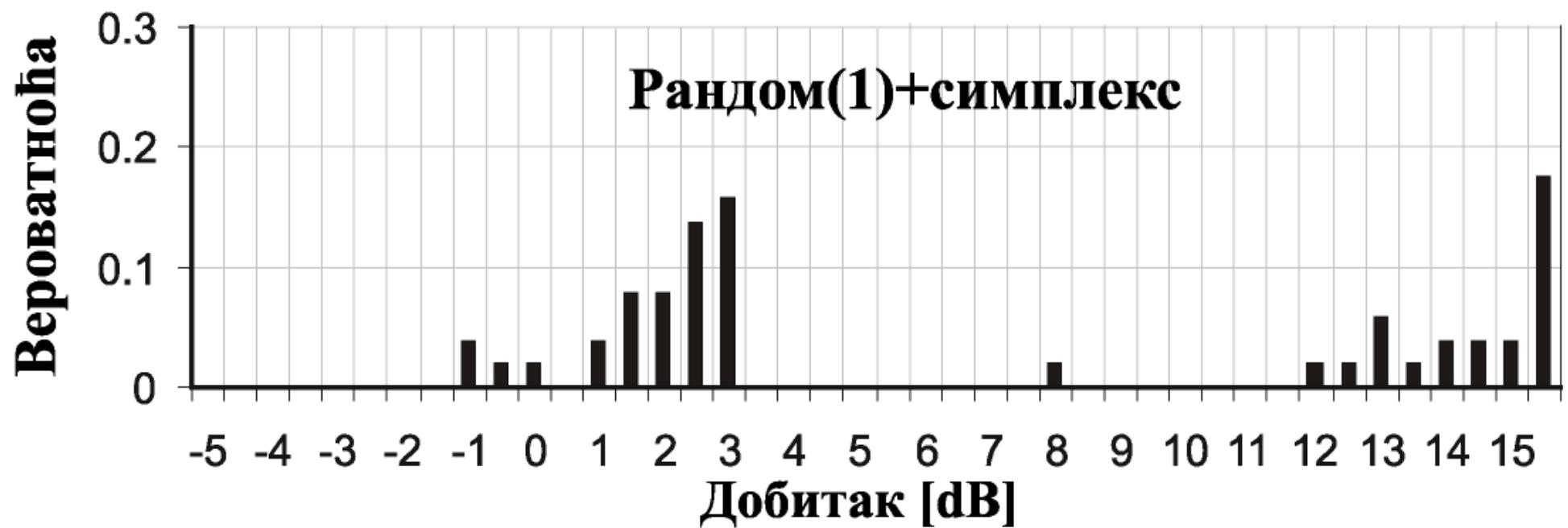
Рандом(1) + градијент

- Хистограм показује да постоји много минимума различитих дубина
- Градијентна метода конвергира ка једном од најближих локалних минимума у околини полазног решења
- Пронађени су минимуми у свим деловима оптимизационог простора у којем они постоје, без обзира на дубину минимума
- Решења добијена овом методом дају увид у расподелу положаја локалних минимума и њихових дубина у оптимизационом простору

Параметри симплекс алгоритма

- Коефицијенти износе:
рефлексија $\alpha = 1$, експанзија $\gamma = 2$,
контракција $\beta = 0,5$ и сажимање $\sigma = 0,5$
- Критеријуми за завршетак алгоритма су:
(а) релативна разлика, по свакој од координата,
свих тачака симплекса мања од 0,05%
(б) апсолутна разлика функције грешке у свим
тачкама симплекса мања од 10^{-3}
(в) укупно 300 итерација

Рандом(1) + симплекс

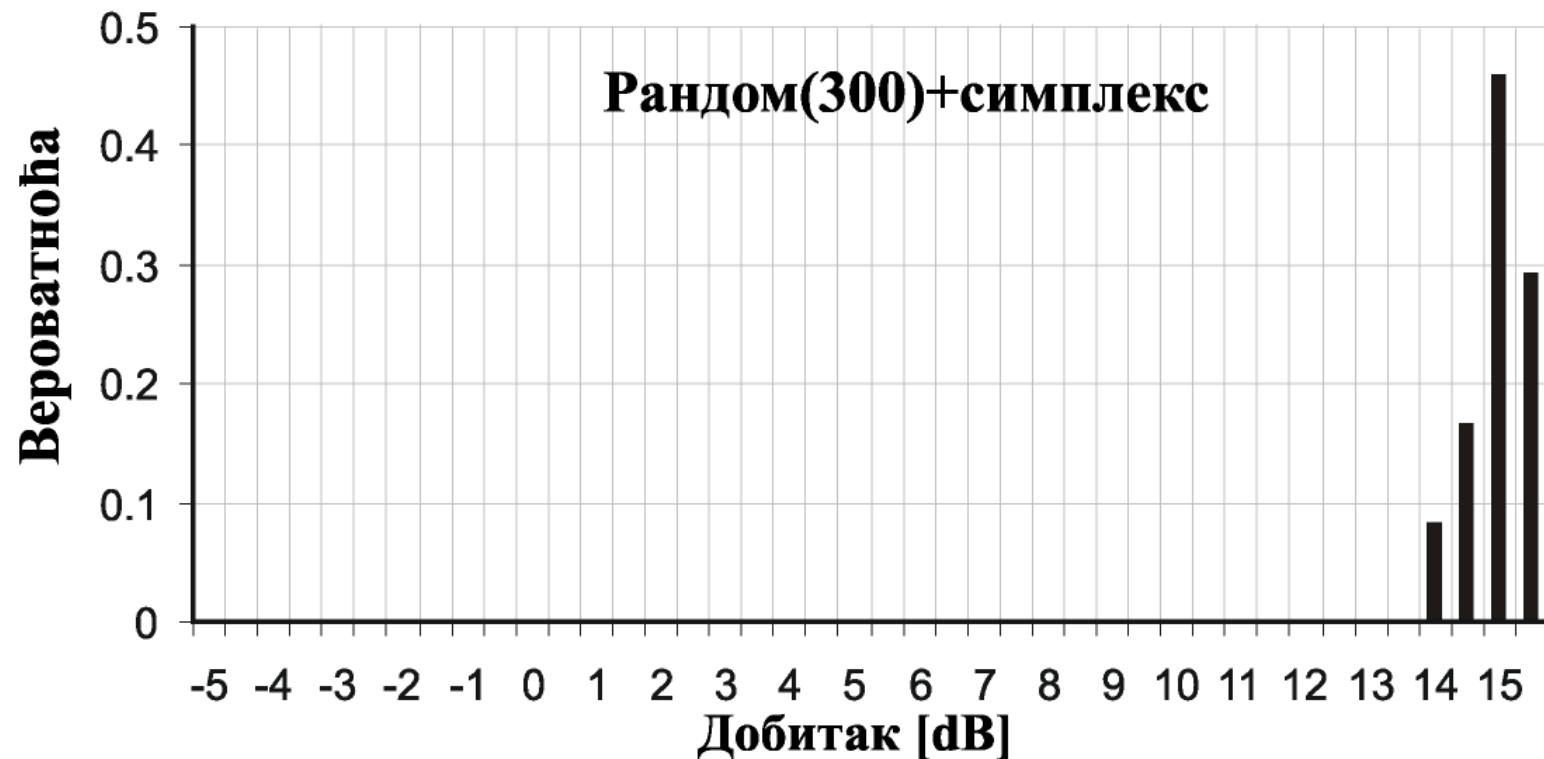


Рандом(1) + симплекс

- Понављањем симплекса проналазе се боља решења него понављањем градијентног алгоритма
- Овакви резултати су последица особине симплекса да успешно пронађе најбољи минимум у околини полазног решења

Рандом(300) + симплекс

- Прескачу се делови простора у којима постоје локални минимуми мањих дубина

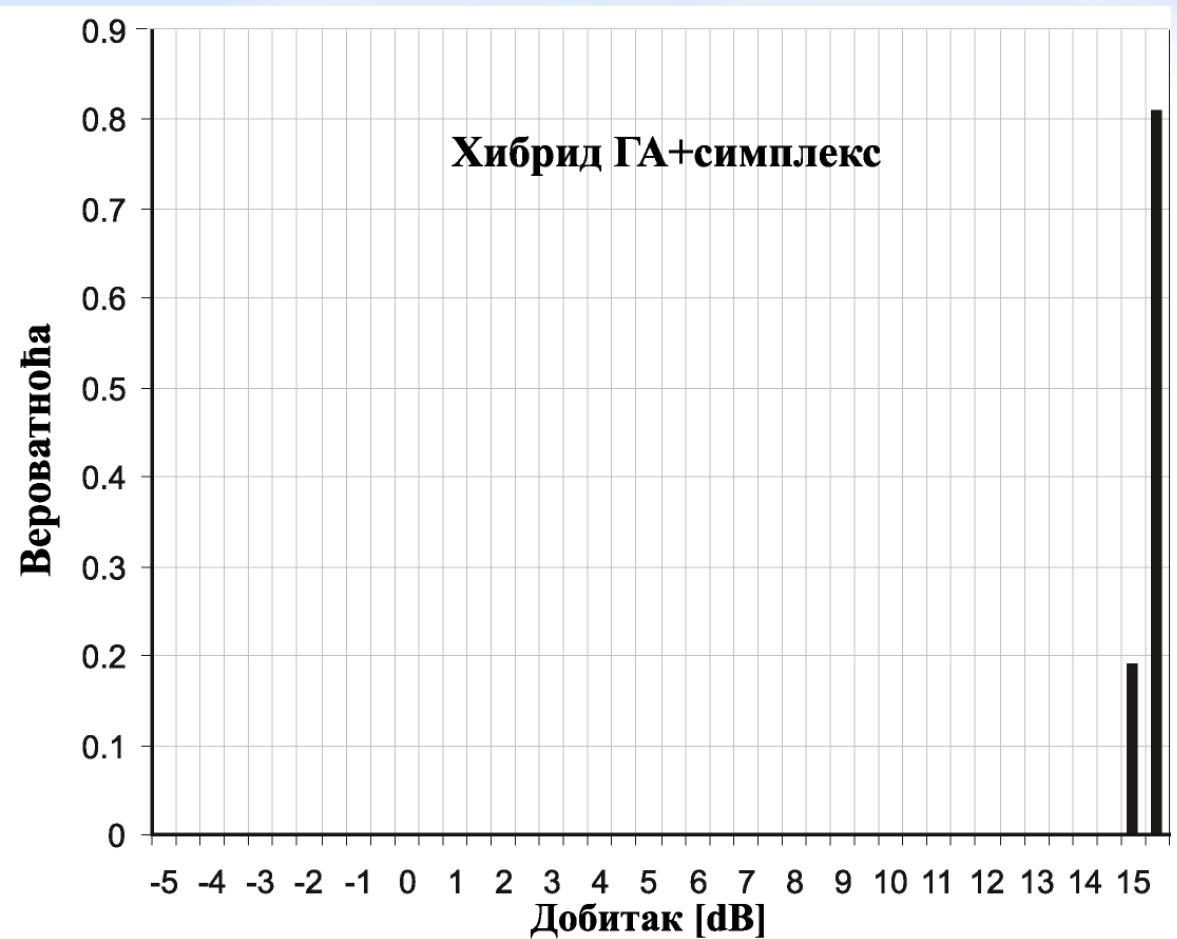


Хибридни ГА+симплекс

- Больје од обичног ГА у смислу много мањег броја потребних итерација
- Параметри ГА су: шеснаестобитно кодовање оптимизационих параметара, величина популације 60, у свакој генерацији 20 најбољих решења учествује у стварању наредне генерације, укупан број генерација 5, вероватноћа укрштања је 0,8 и вероватноћа мутације 0,1
- ГА има 300 итерација и најбоље решење после тога је полазна тачка за симплекс

ГА + СИМПЛЕКС

- Вероватноћа проналажења најбољег решења 80% (глобални минимум)



Процена локалних минимума

- За сваку тачку у скупу од 100 случајно изабраних, суседних 10 тачака је кориштено за проверу да ли је посматрана тачка локални минимум
- Шест независних покретања је било потребно да би се добио укупан потребан број итерација
- У оквиру сваког покретања процењено је у просеку 9 минимума

Процена локалних минимума



Рачунање растојања између решења

- У циљу стицања увида у распоред решења у оптимизационом простору, средње растојање између решења је рачувано као

$$R = \sqrt{\frac{2 \sum_{i=1}^N \sum_{j=i+1}^N \sum_{k=1}^6 (x_k^i - x_k^j)^2}{N(N-1)}}$$

- N је укупан број пронађених решења
- x_k^i је k -ти параметар i -тог решења

Средње растојање између пронађених решења

Алгоритам	Средње растојање [m]
Рандом	0,5676
Рандом(1)+градијент	0,5694
Рандом(1)+симплекс	0,5654
Рандом(300)+симплекс	0,4022
Хибрид ГА+симплекс	0,3189
Процена локалних минимума ПЛМ	0,5412

Број итерација за антену са средњим добитком већим од 15 dB

Алгоритам	Просечан број итерација
Случајно претраживање	>10.000
Рандом(1)+градијент	~10.000
Рандом(1)+симплекс	2.400
Рандом(300)+симплекс	1.200
Хибридни ГА+симплекс	690
Процена локалних минимума – ПЛМ	650

Пример #2: Потискивање лобова трансверзалног антенског низа

- Трансверзални антенски низ сачињен од 42 електрички кратка дипола (тачкасти извори зрачења)
- Растојање између суседних дипола је једна половина таласне дужине
- Амплитуде простопериодичних побуда дипола су оптимизоване тако да се добију минимални бочни лобови у дијаграму зрачења

Поставка оптимизације

- Симетрија коефицијената: само једна половина коефицијената низа је оптимизована
- Да би се избегао бесконачан број решења, који је последица могућег скалирања коефицијената, једном коефицијенту је дата апсолутна вредност
- На овај начин је добијен двадесетодимензиони оптимизациони проблем

Дијаграм зрачења низа

- Карактеристична функција зрачења

$$F(\theta) = 2 \cdot \sum_{k=1}^N a_k \cos\left(\left(k - N - \frac{1}{2}\right) \cdot \beta d \cos(\theta)\right)$$

- θ је угао између осе низа и правца у коме се рачуна функција
- $2N$ је укупан број елемената низа
- a_k је амплитуда побуде k -тог елемента
- β је коефицијент зрачења и
- d је растојање између суседних елемената

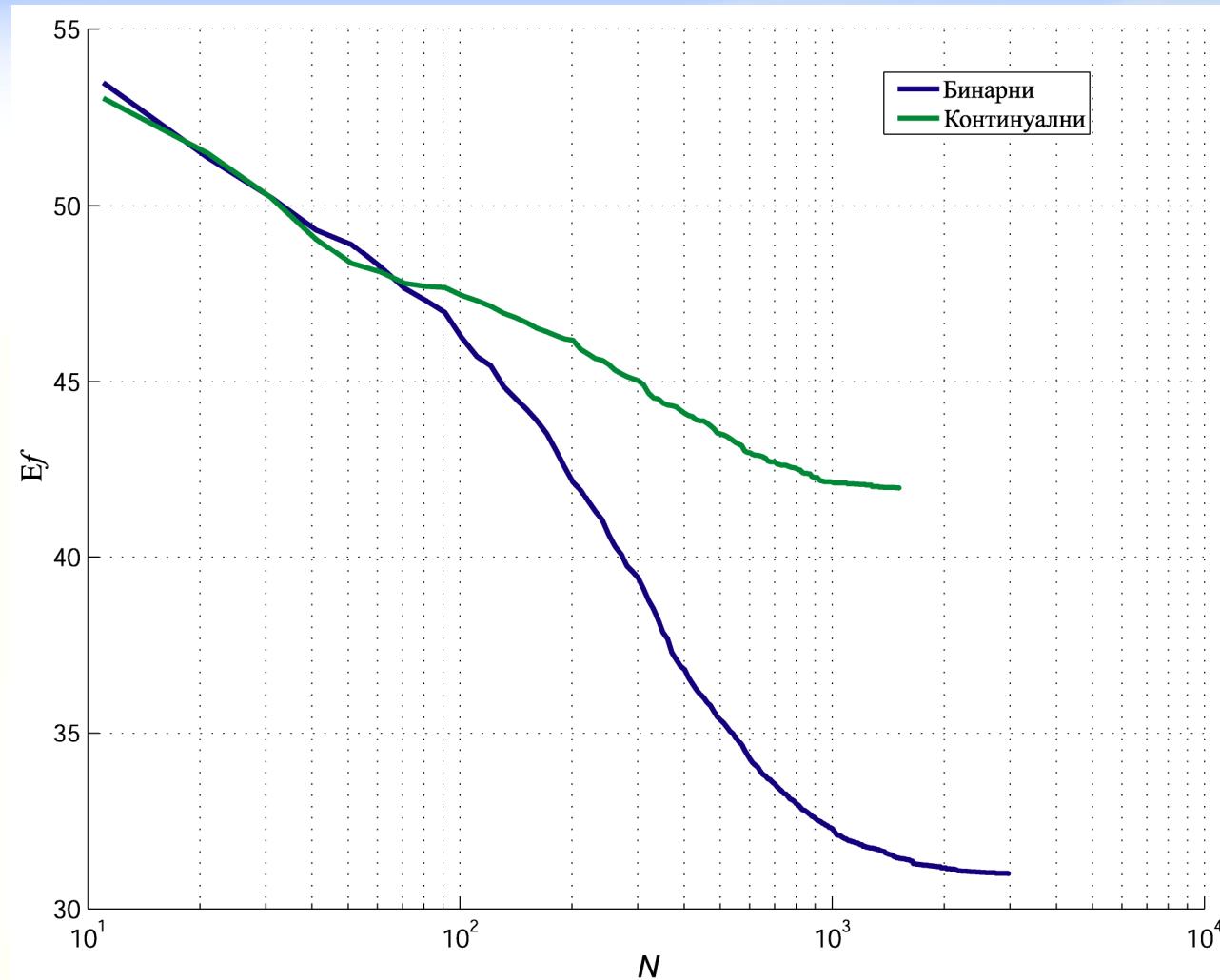
Описна функција

- Критеријум за оптимизацију је да нивои бочних листова буду нижи од -80 dB за $\theta > 25^\circ$ (мерено од осе низа)
- Оптимизациона функција

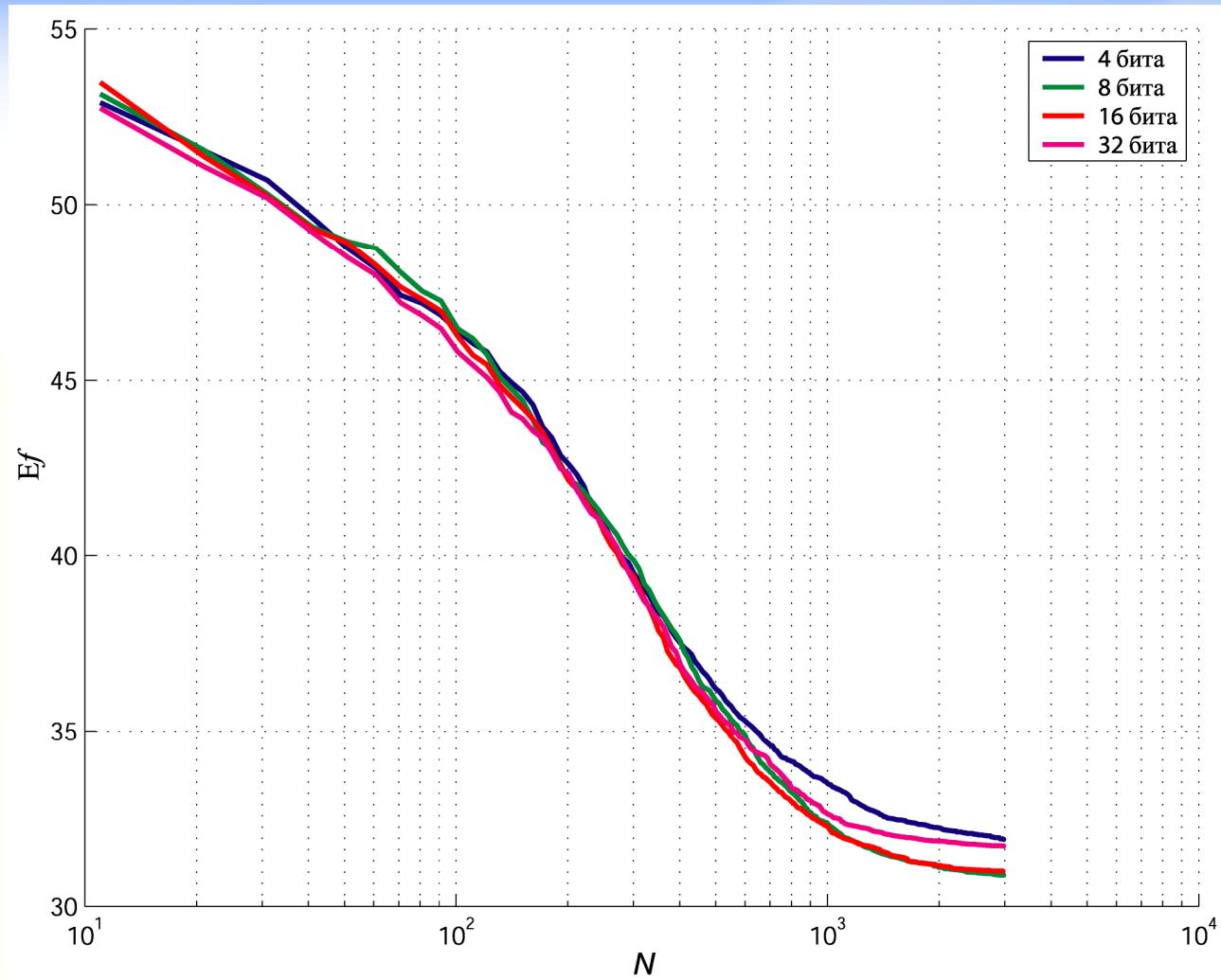
$$Cost\ Func. = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} \left\{ \max[0, -80 - F_{\max} - F(\theta_i)] \right\}^2}$$

- 100 статистички независних покретања алгоритма

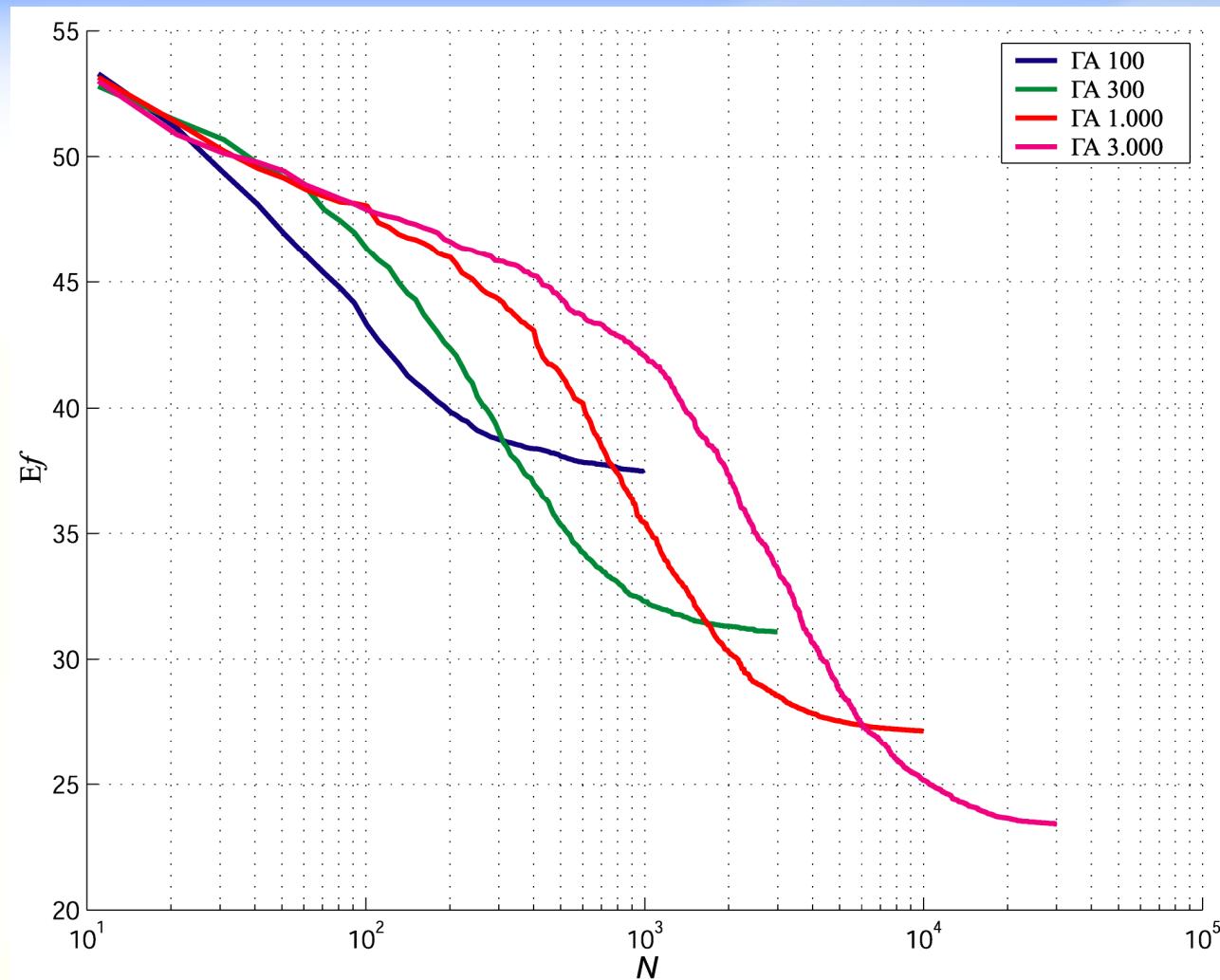
ГА: бинарно / континуално представљање



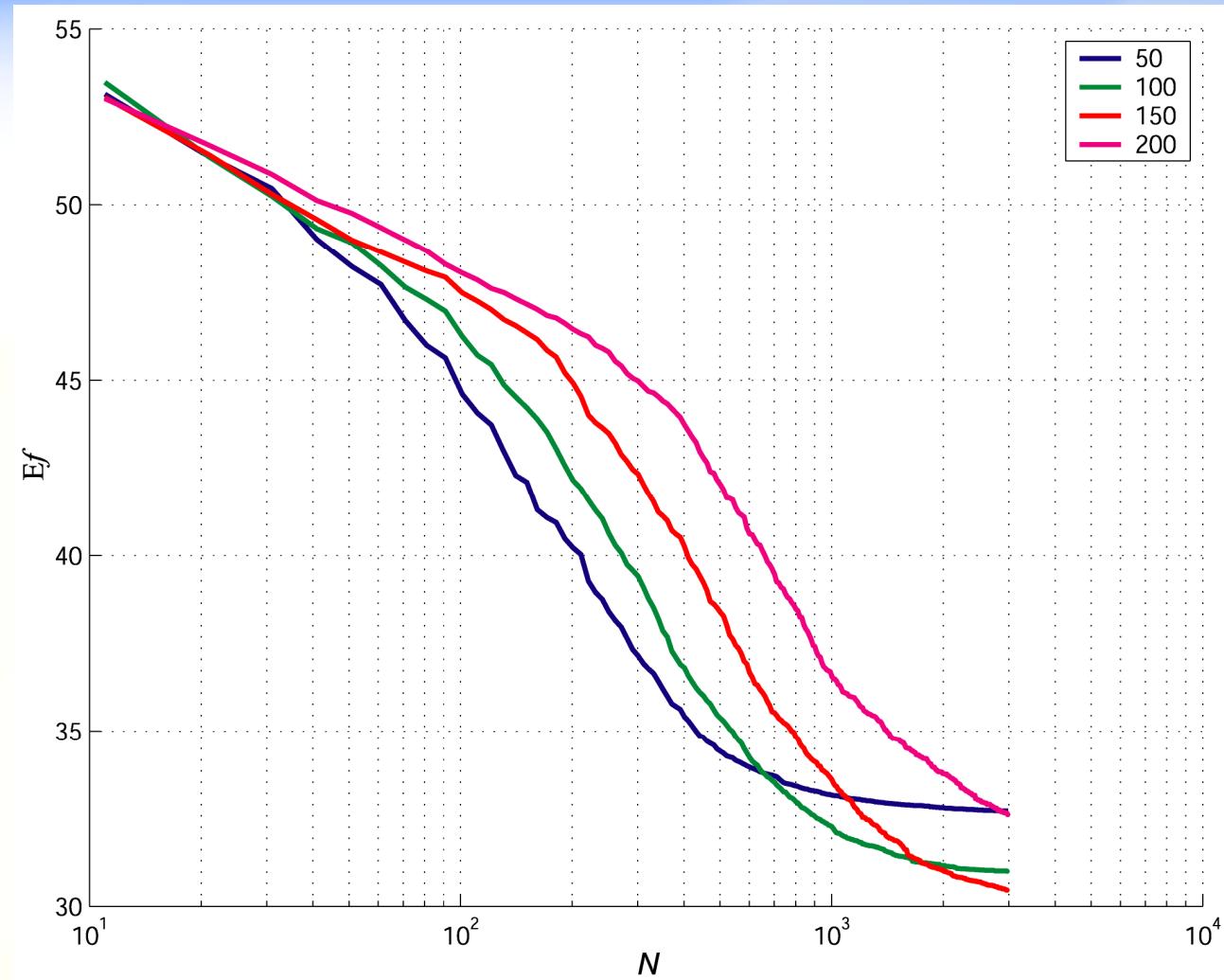
Утицај броја бита за представљање променљивих



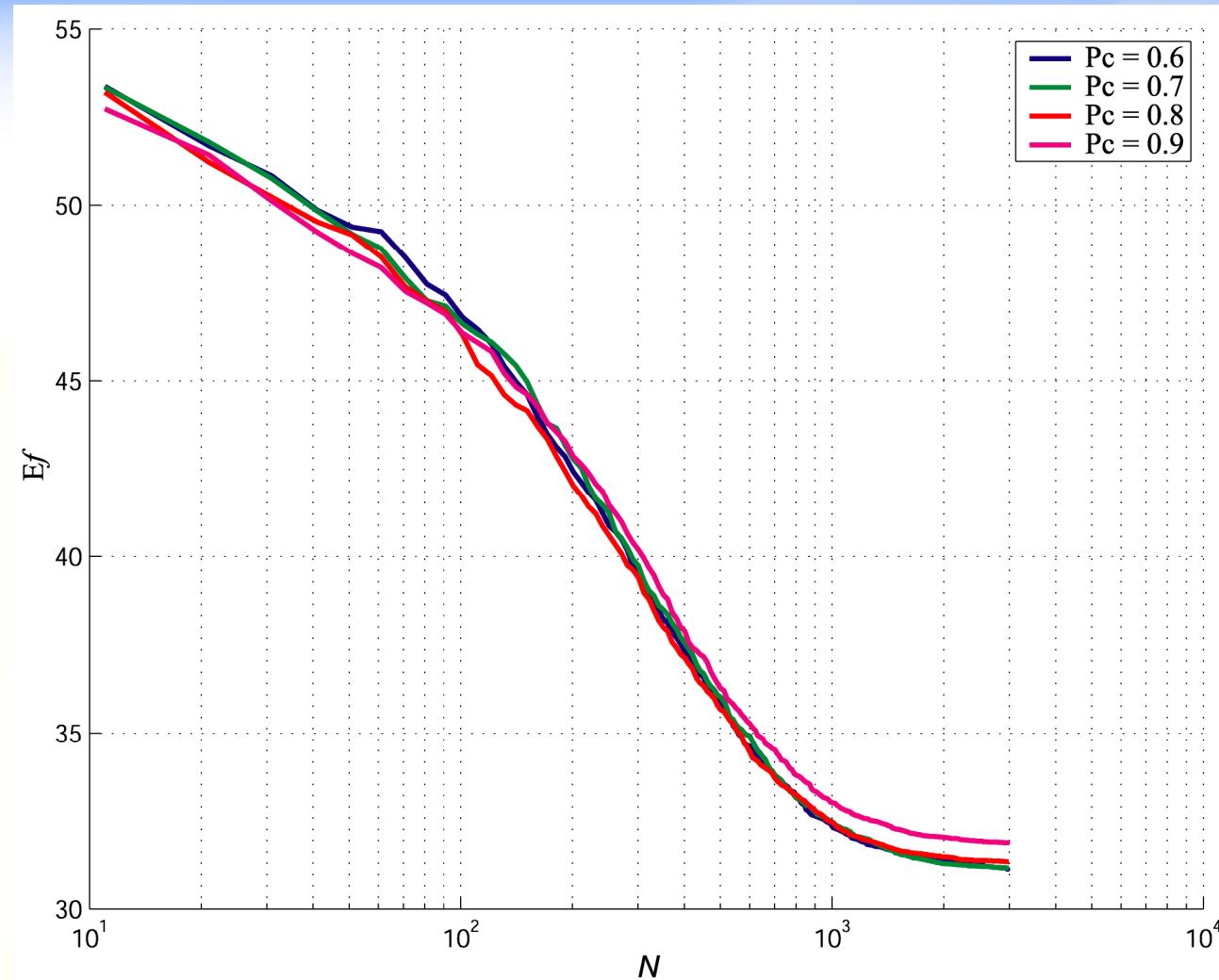
Утицај величине популације



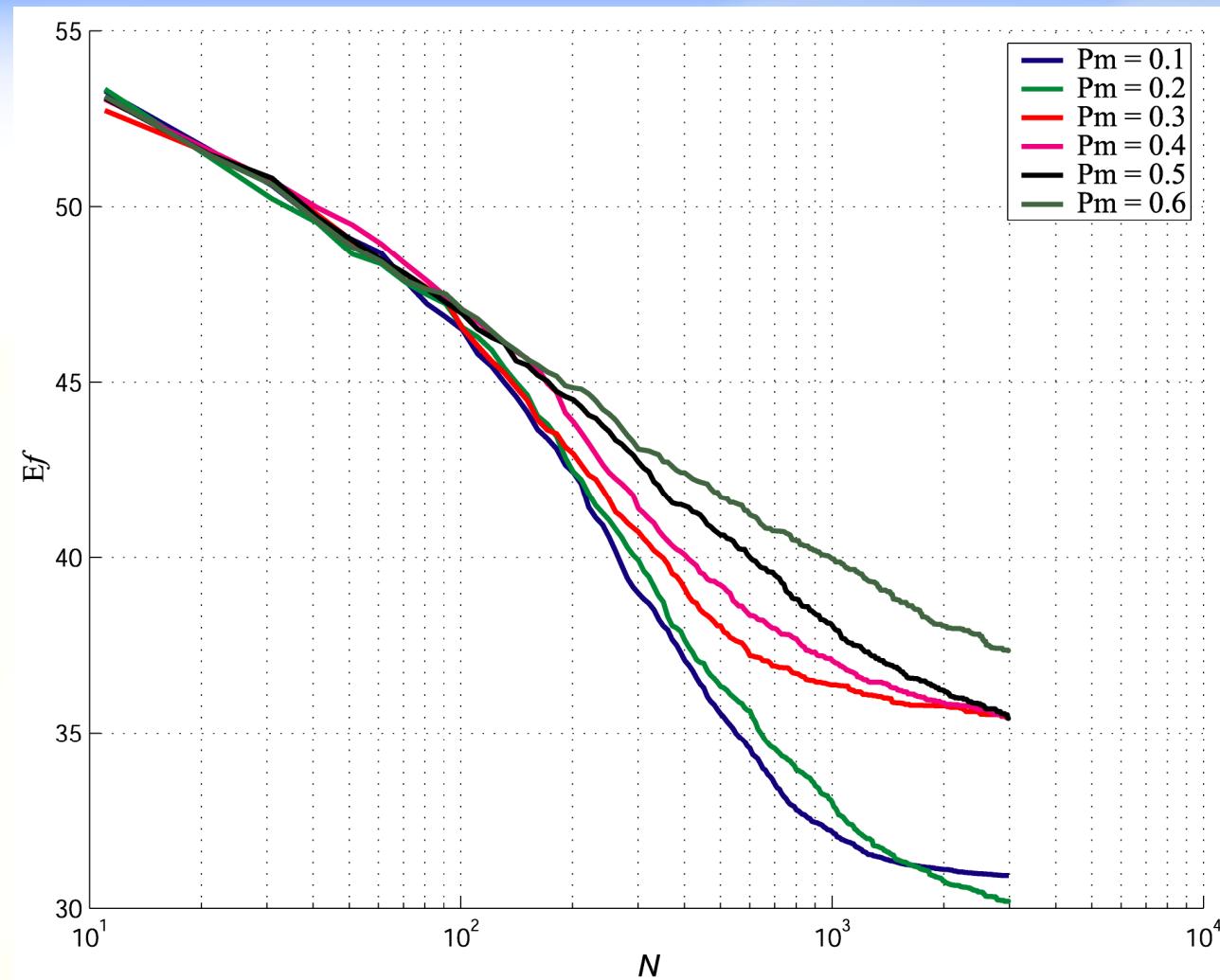
Број решења за формирање наредне генерације (30%)



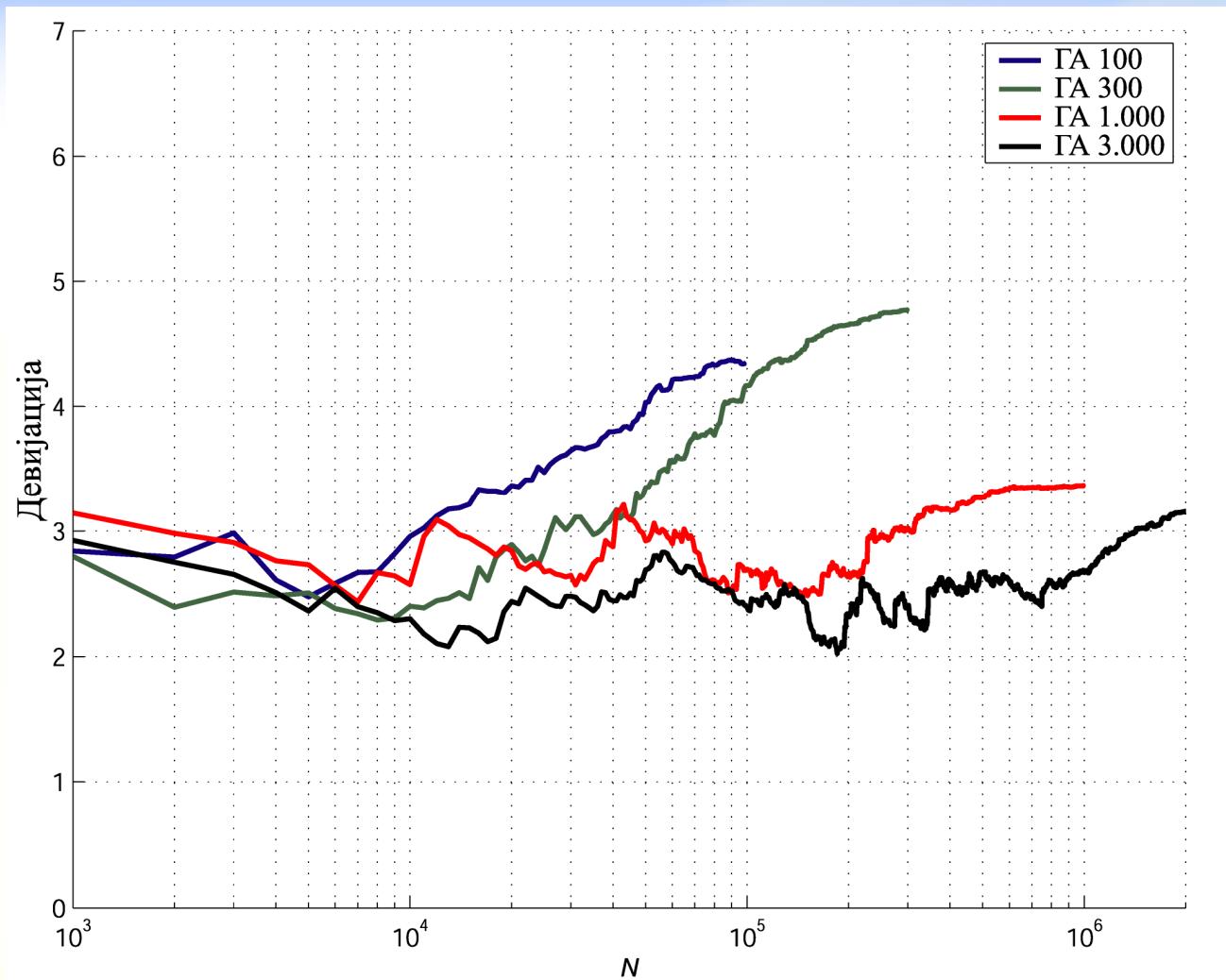
Утицај вероватноће укрштања



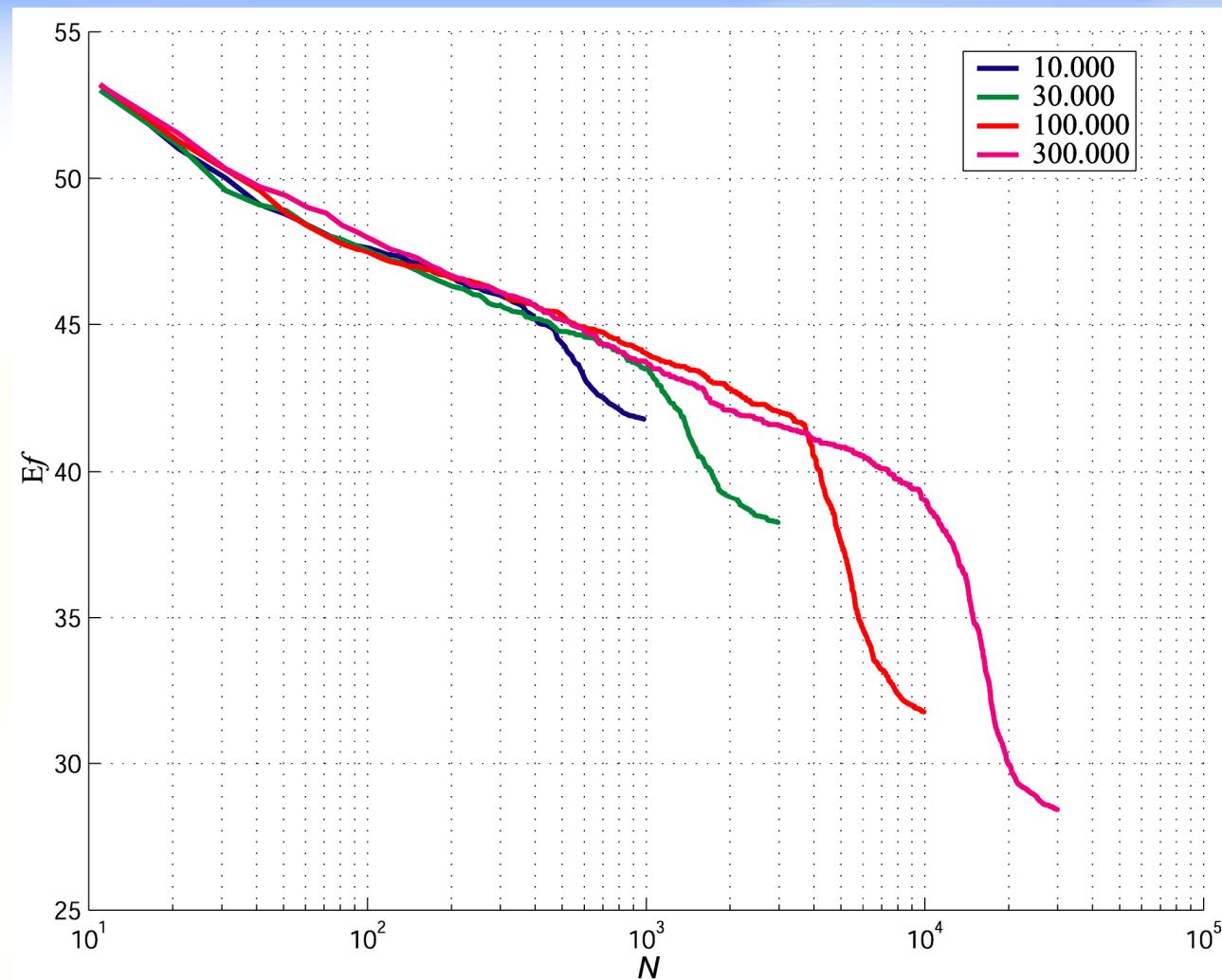
Утицај вероватноће мутације



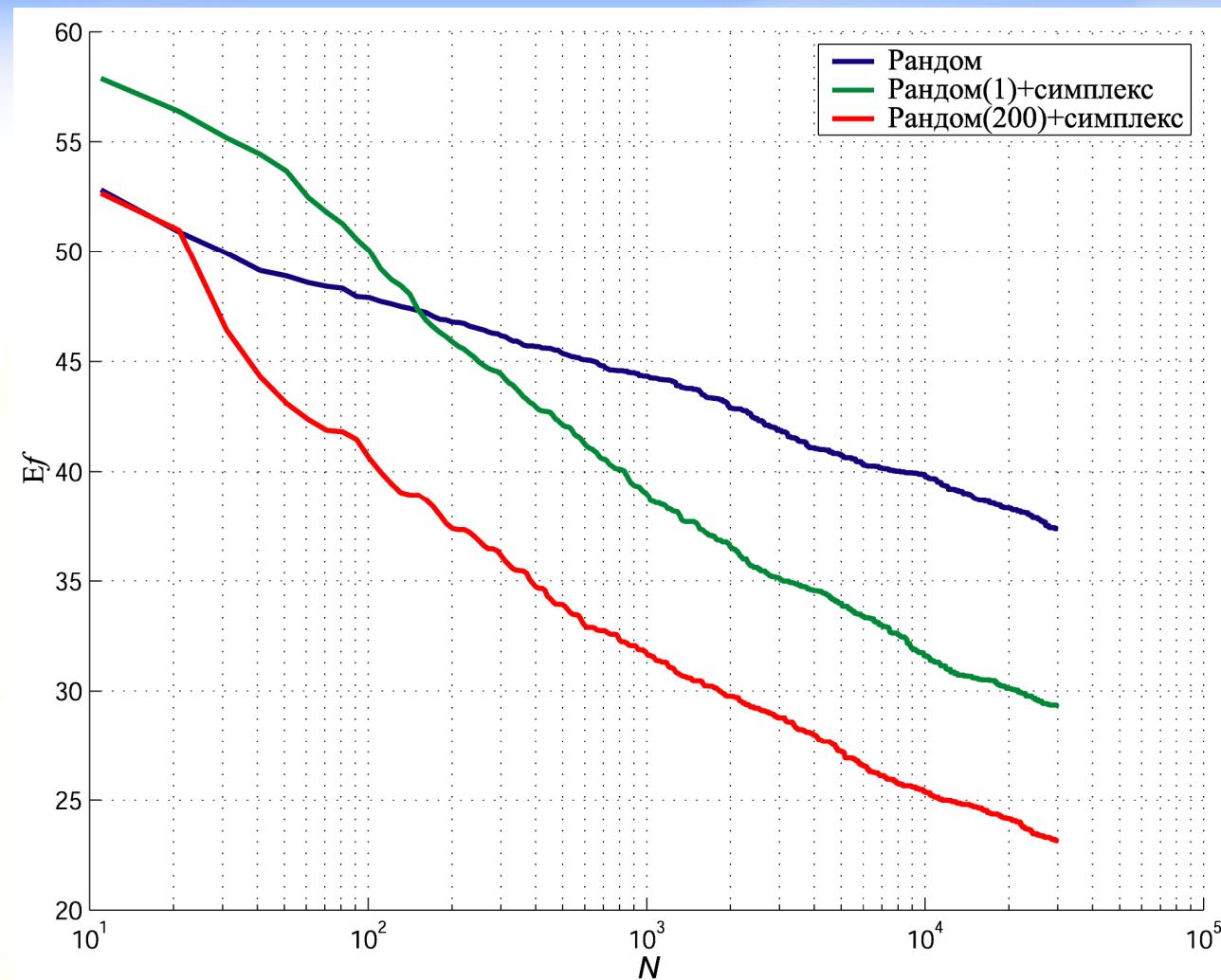
Девијација средњег најбољег решења (популације 100-3000)



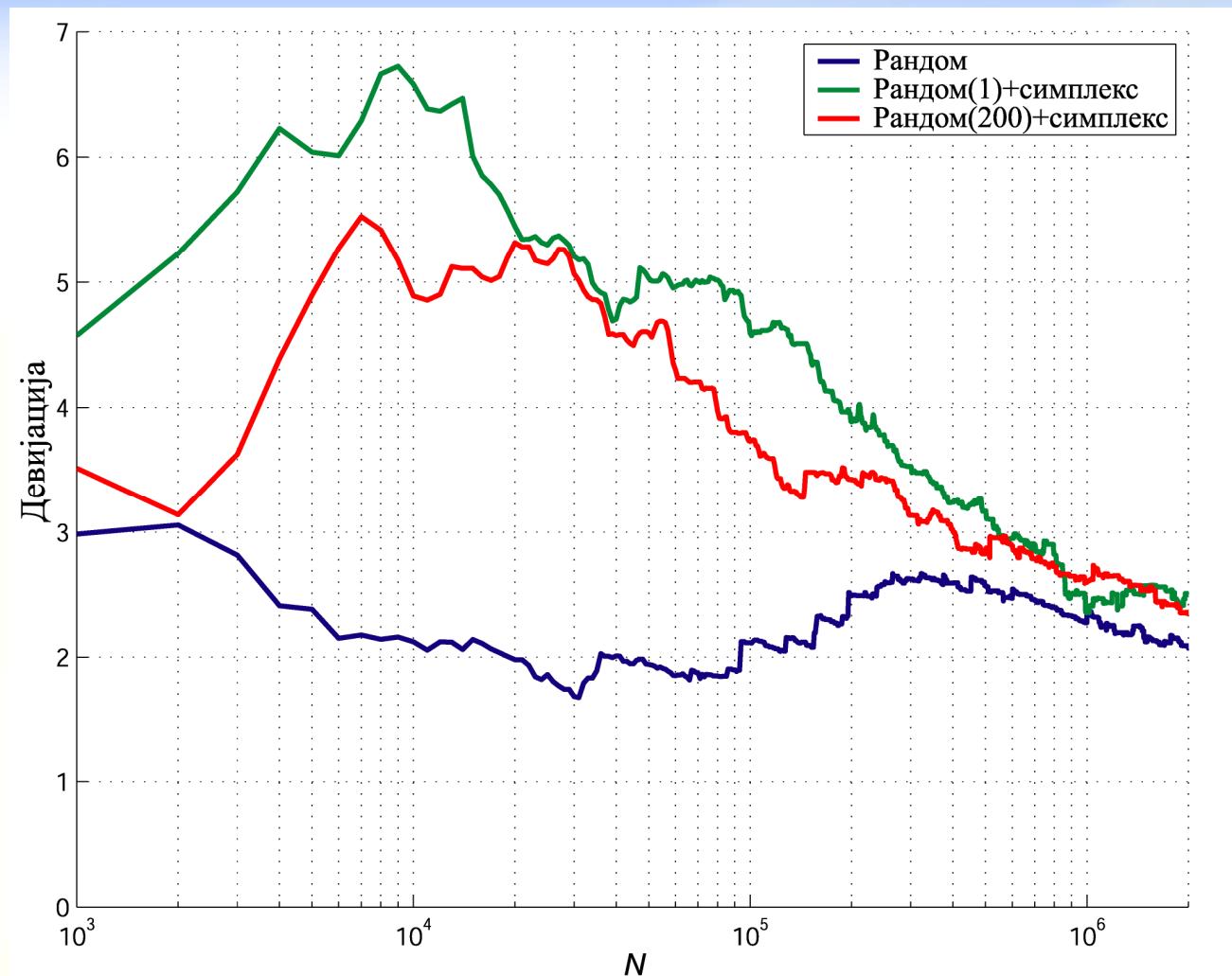
Симулирано каљење (ГА је ефикаснији)



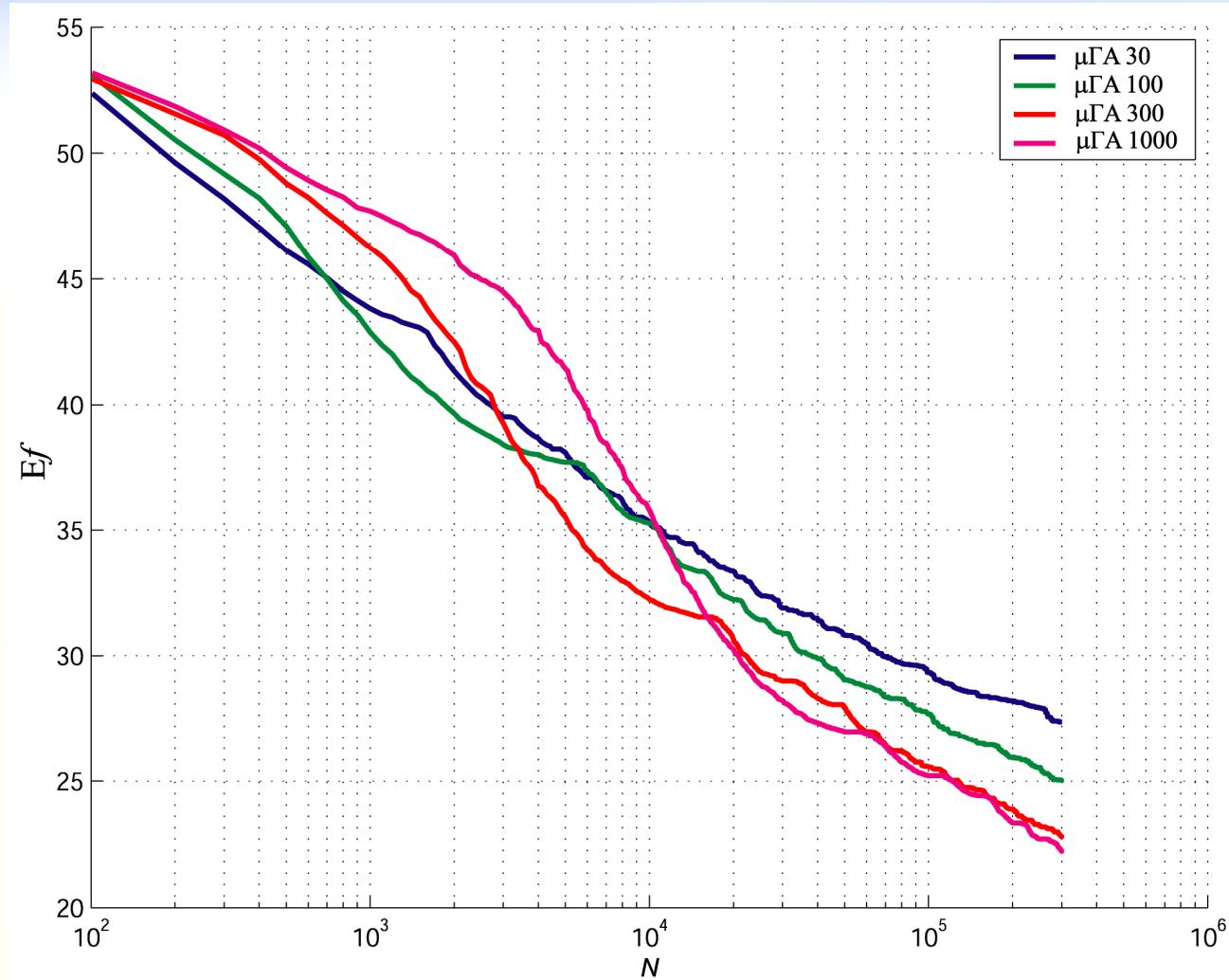
Методи понављања (ГА није много бољи)



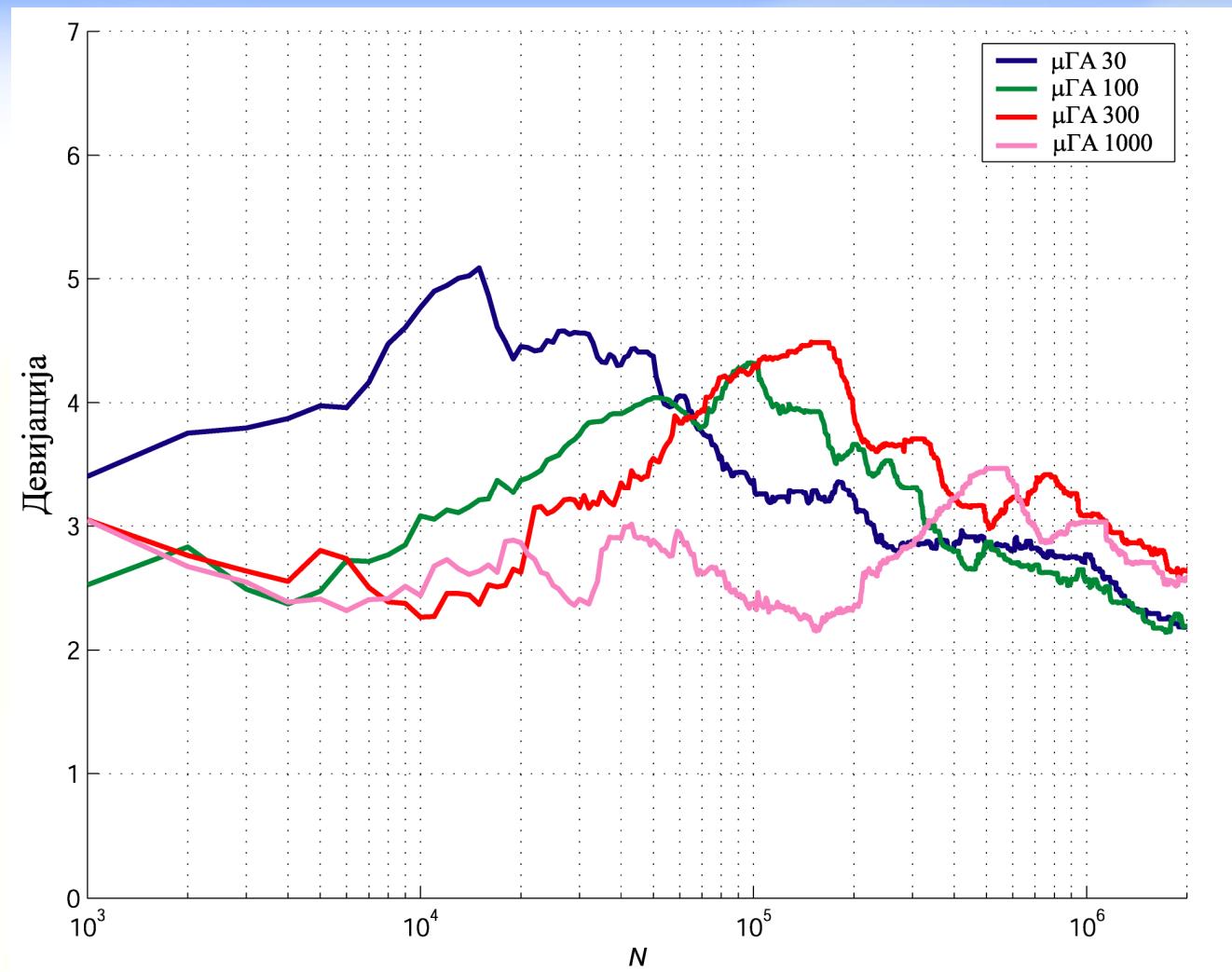
Методи понављања: девијација



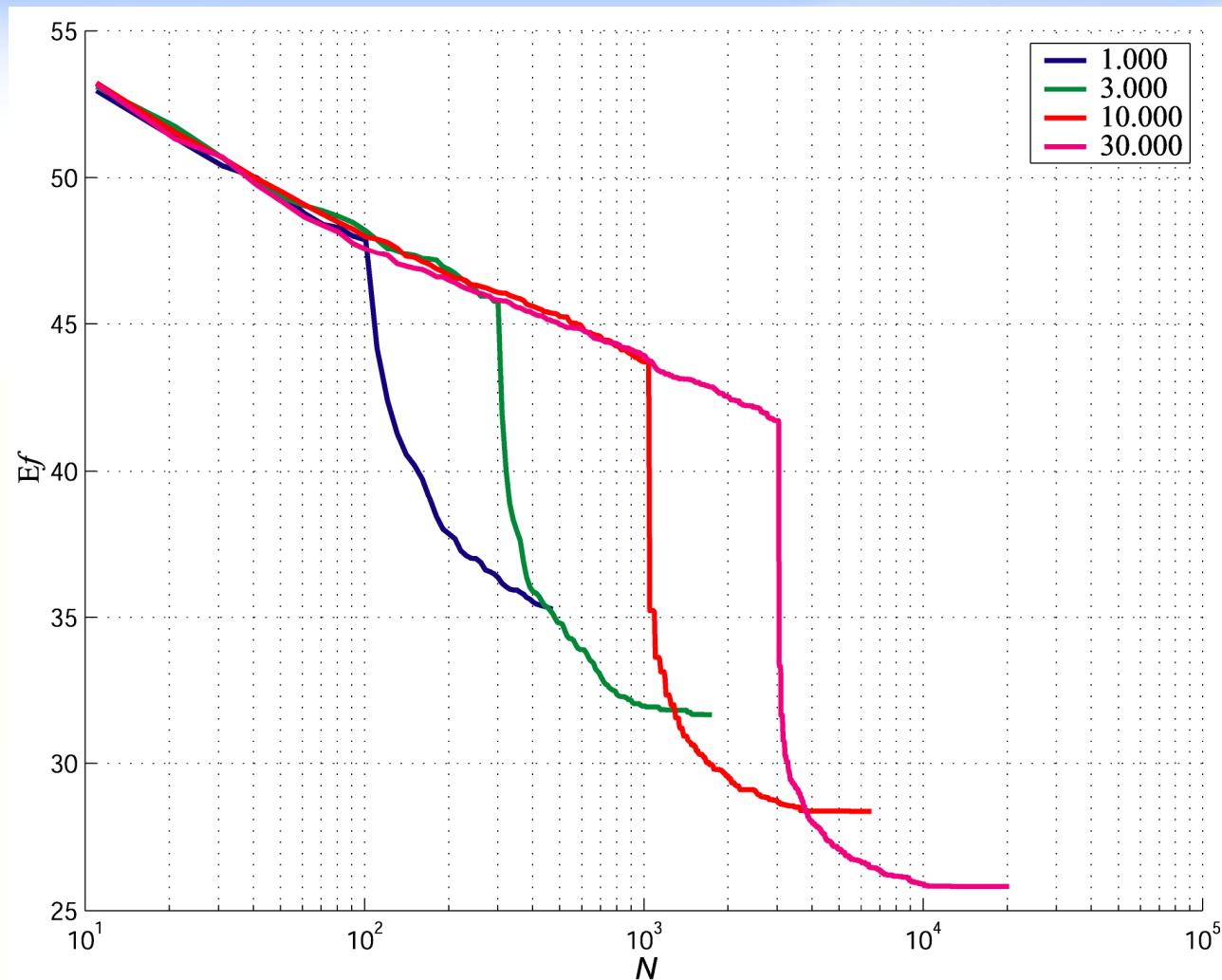
Микро ГА (популација)



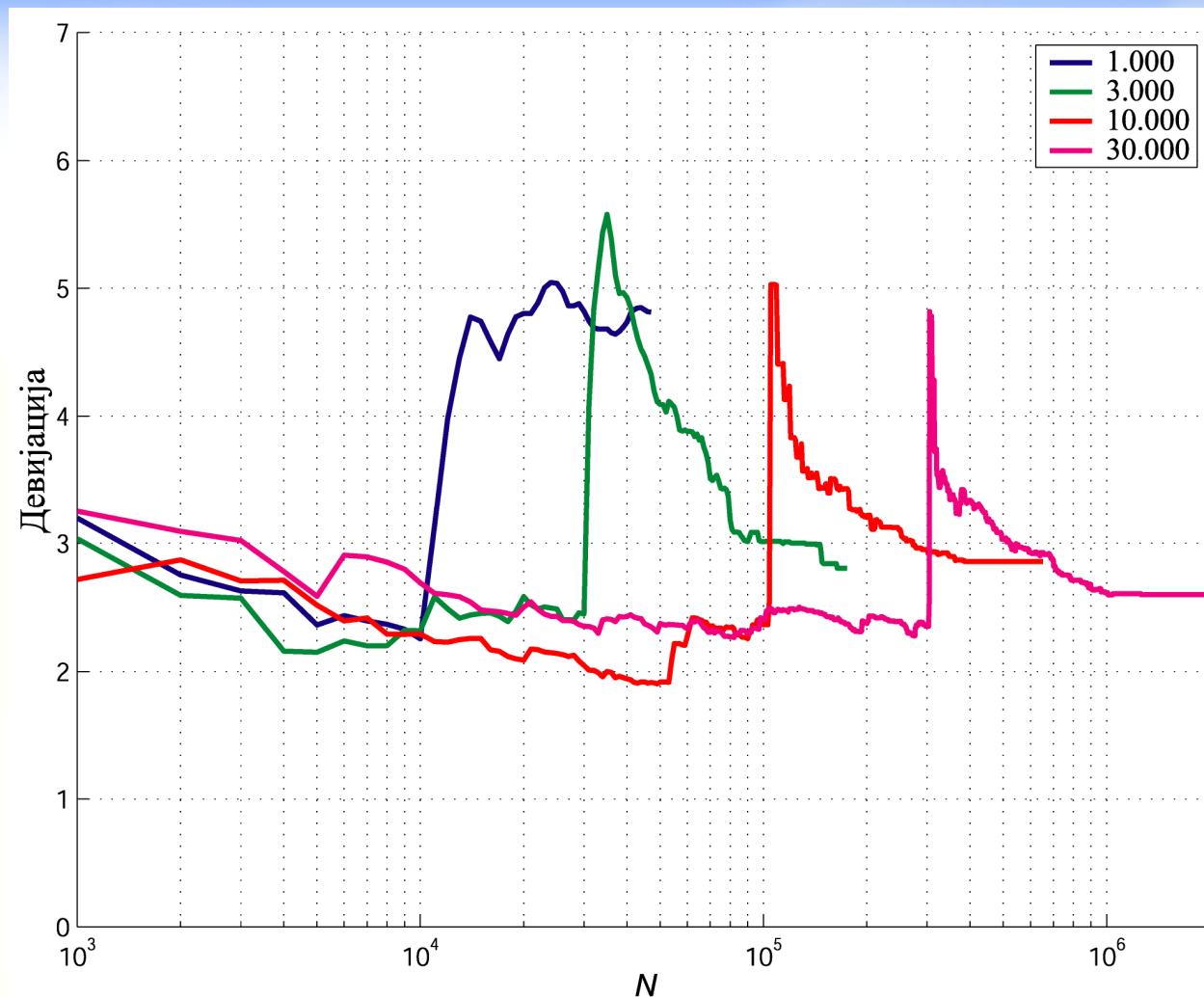
Микро ГА: девијација



Метод процене локалних минимума (полазни скуп)



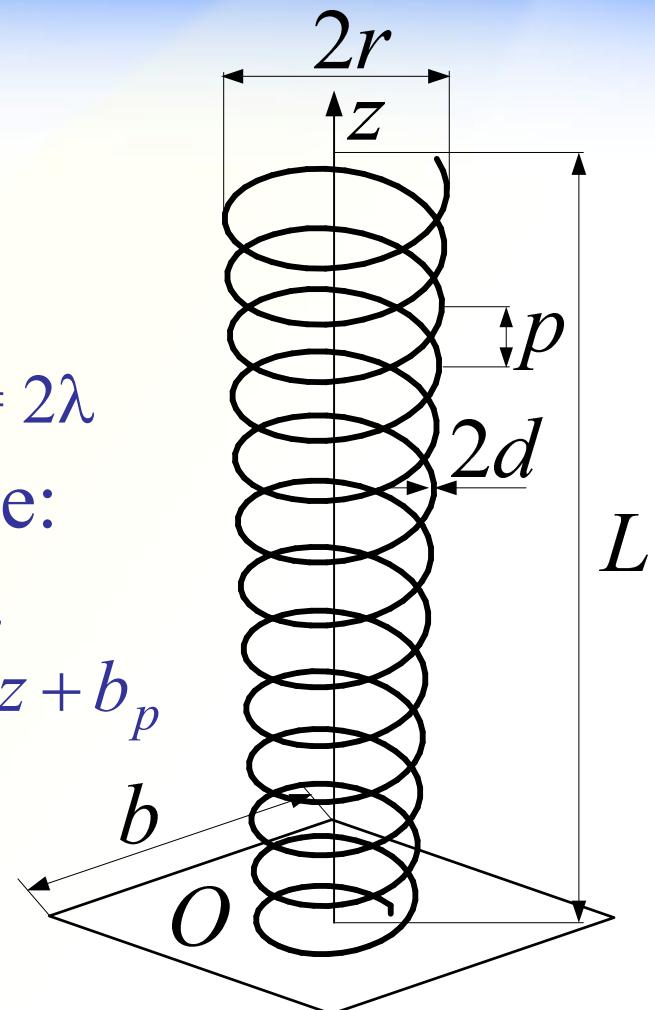
ПЛМ: девијација



Пример #3:

Неуниформна хеликоида

- Предефинисано:
 - полуупречник жице, $d = \lambda/10^4$
 - укупна висина, L ,
 - страница квадратног рефлектора, $b = 2\lambda$
- Линеарна промена од z -координате:
 - полуупречник завојка, $r(z) = a_r z + b_r$
 - растојање између завојака, $p(z) = a_p z + b_p$
- Губици узети у обзир
- Опт. теорија: NLP проблем са 4 опт. променљиве



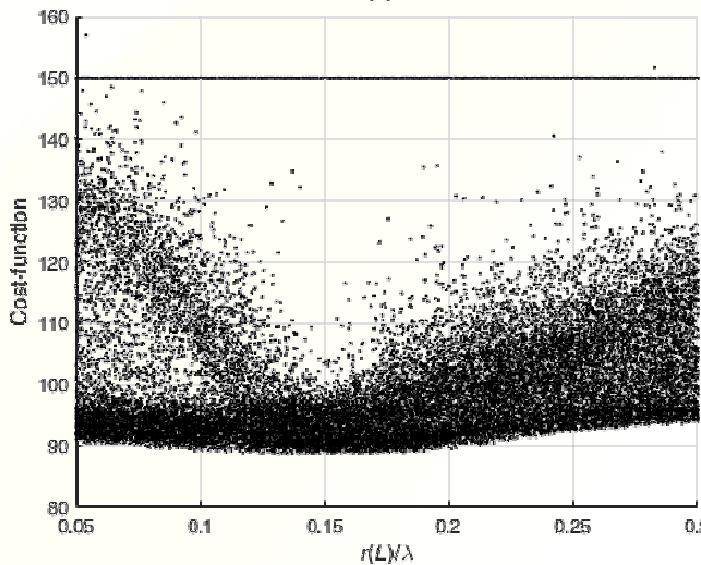
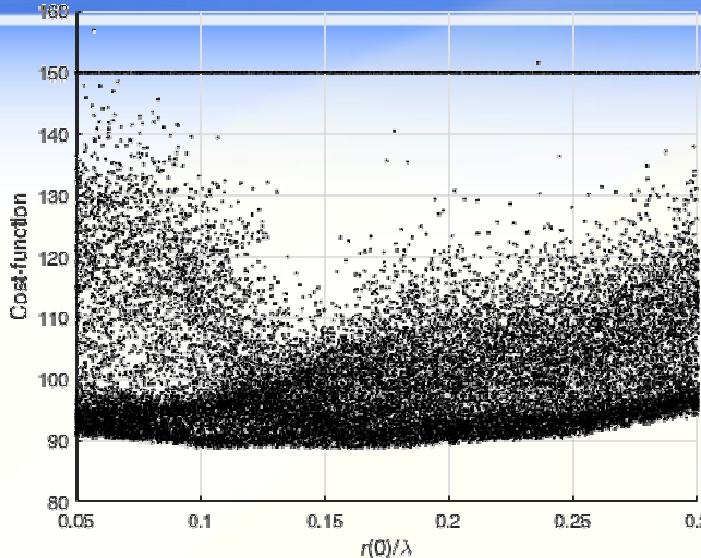
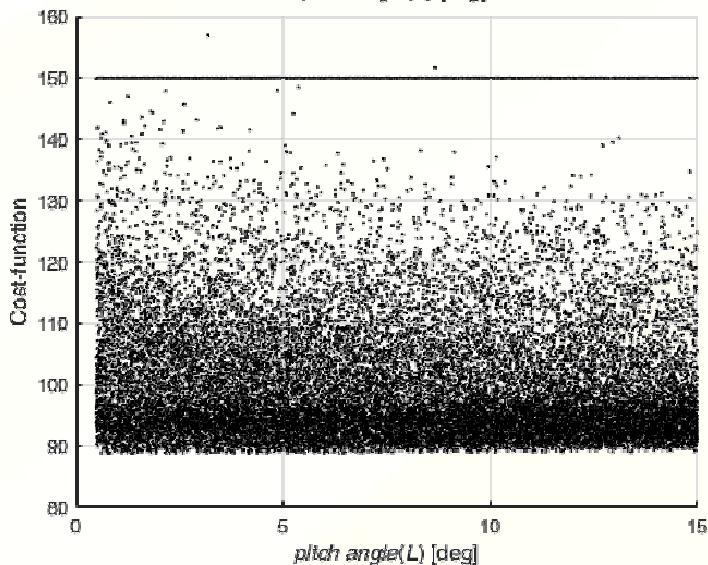
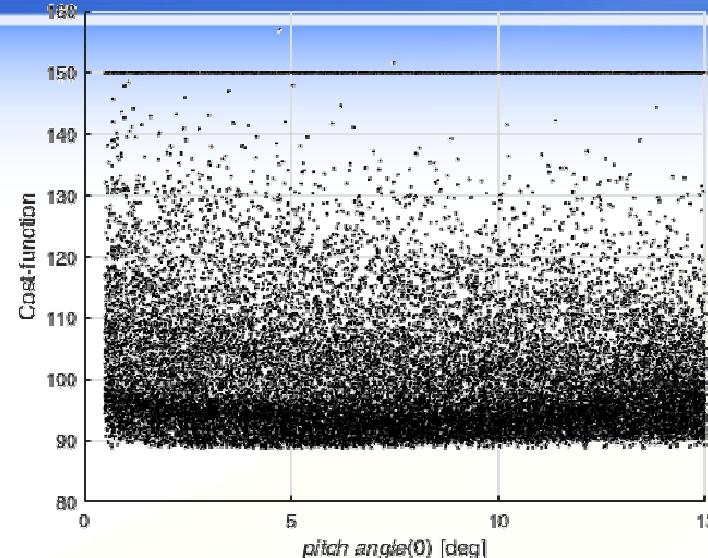
Оптимизационе променљиве

- Оптимизационе променљиве: $r(0)$, $r(L)$, $p(0)$ и $p(L)$
- Полупречници су нормализовани на таласну дужину, λ , на централној учестаности
- Одговарајући коефицијенти a_r , b_r , a_p и b_p су прорачунати и на основу њих је дефинисана геометрија структуре
- Границе оптимизационих променљивих:
 - $20 \cdot 10^{-3} \leq r(0)/\lambda \leq 500 \cdot 10^{-3}$
 - $20 \cdot 10^{-3} \leq r(L)/\lambda \leq 500 \cdot 10^{-3}$
 - $0.5^\circ \leq p(0), p(L) \leq 15^\circ$

Критеријуми и дефиниција оптимизационе функције

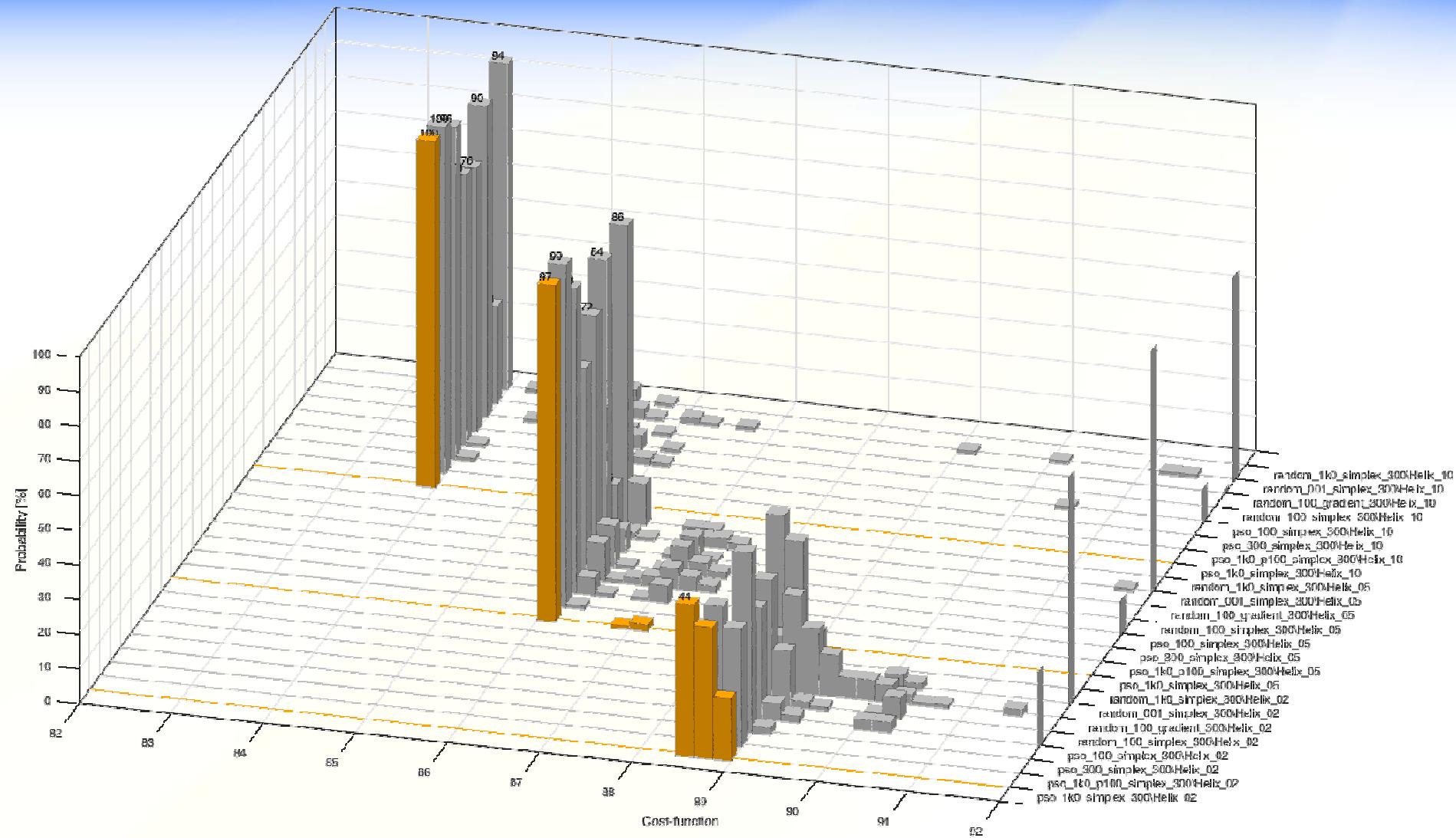
- Антена је моделована у прерађеним верзијама програма AWAS и WIPL-D како би могло да се изврши много итерација
- Посебно је аутоматски прављена геометрија и задавање оптимизационе функције
- Два критеријума:
 - (1) максимално појачање у главном правцу
 - (2) кружна поларизација таласа
- Оптимизациона функција: $f = 100 - g_{[\text{dBi}]} - rgr$
 $E_{\text{tot}} = \sqrt{E_{\theta}^2 + E_{\phi}^2 + \delta} \quad E_r = \left| 0.5(-E_{\theta} + jE_{\phi}) \right|$
 $rgr = 20 \log_{10} \left(E_r / E_{\text{tot}} + 10^{-12} \right)$
- f је 150 за геометрије које су физички неоствариве

Увид у оптимизациони простор 100 000 случајних одбира

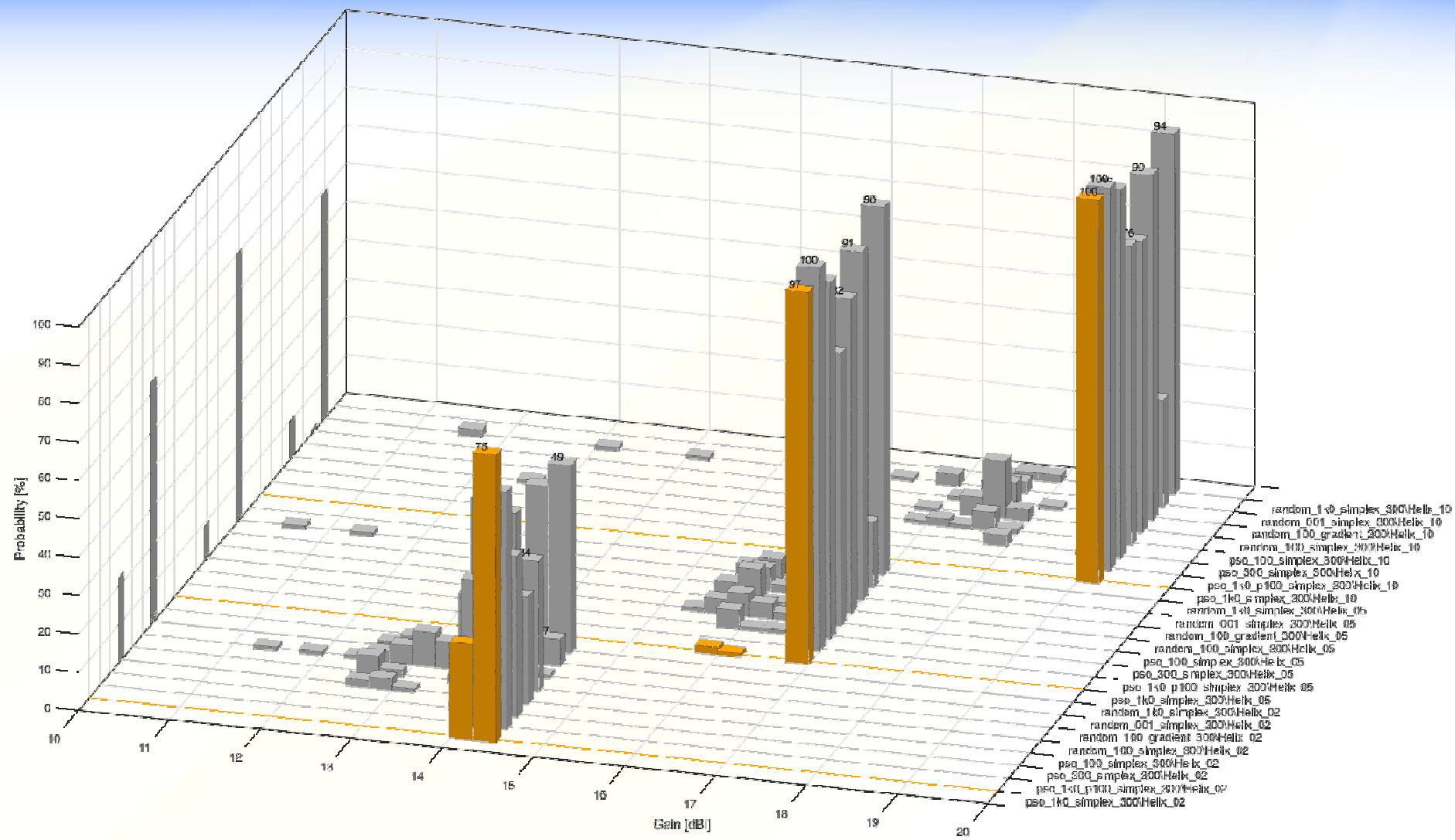


- Много локалних минимума
- Тежак опт. проблем
- Одређивање делова опт. простора у којима се налазе боља решења је тешко!

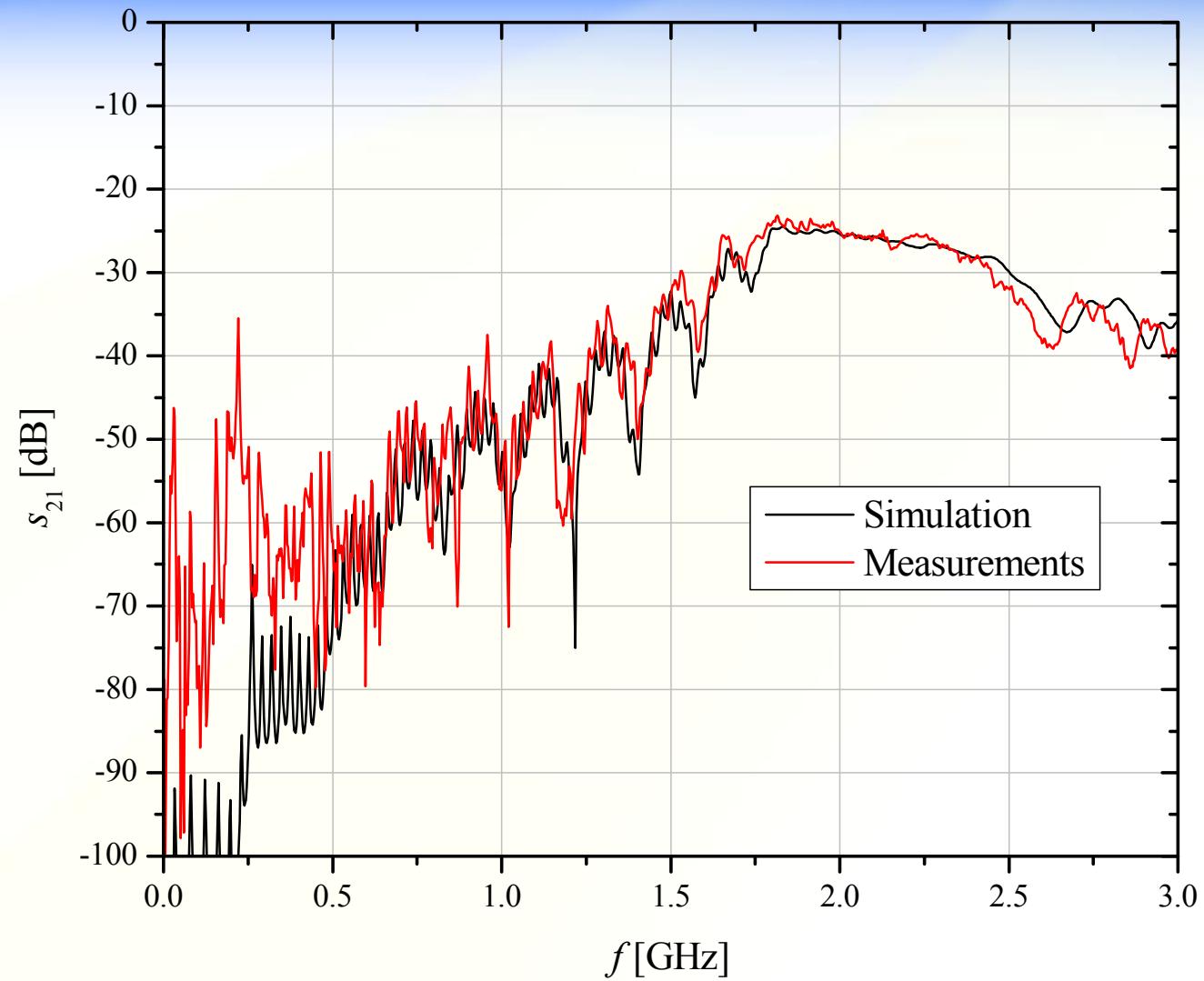
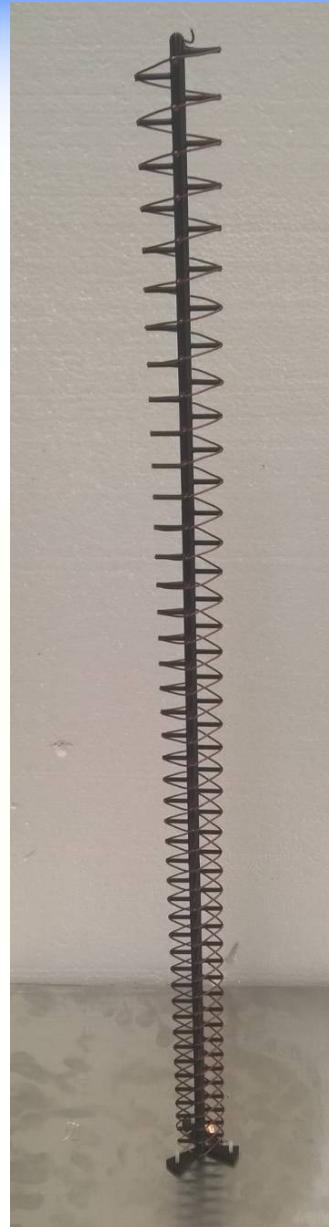
Минимизација опт. функције за 2 λ , 5 λ и 10 λ хеликоиде



Максимизација појачања за 2λ , 5λ и 10λ хеликоиде

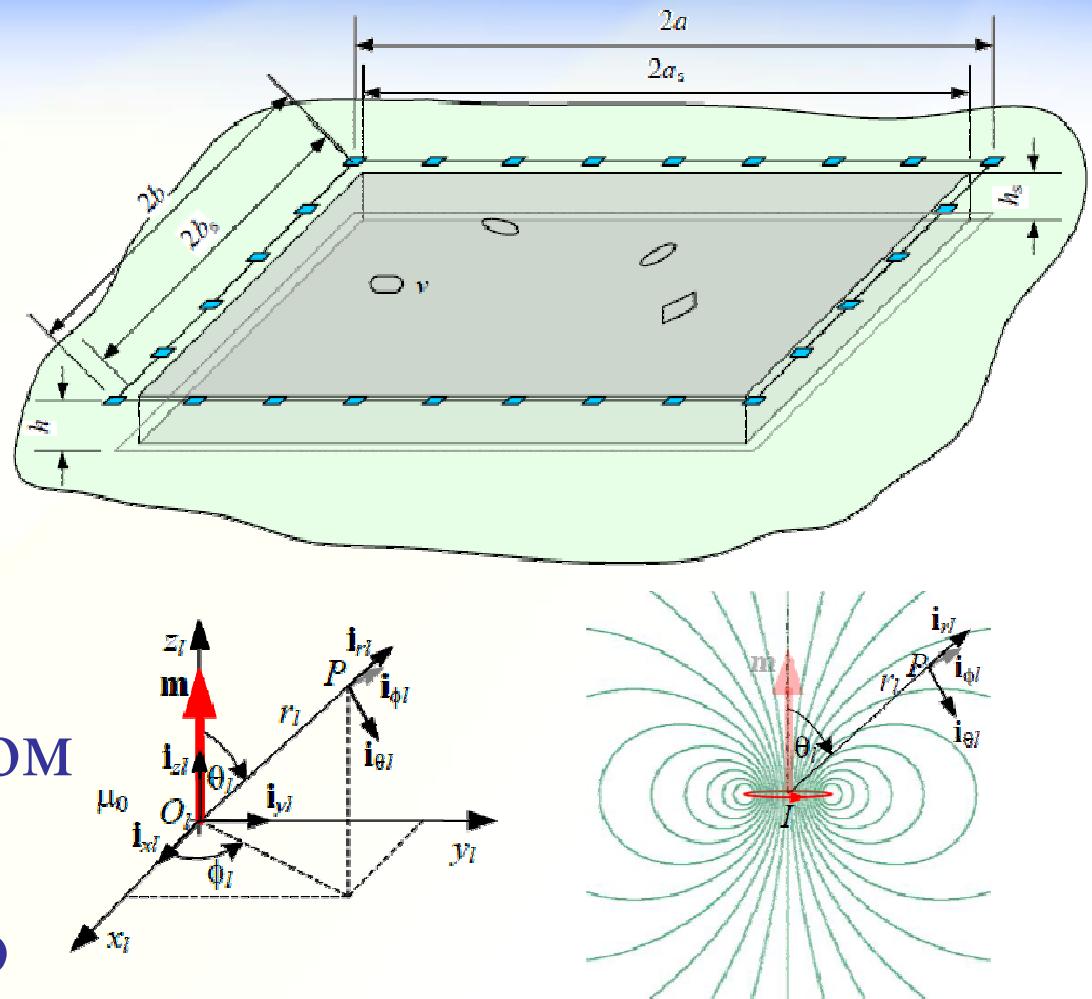


3-D одштампан прототип и верификација



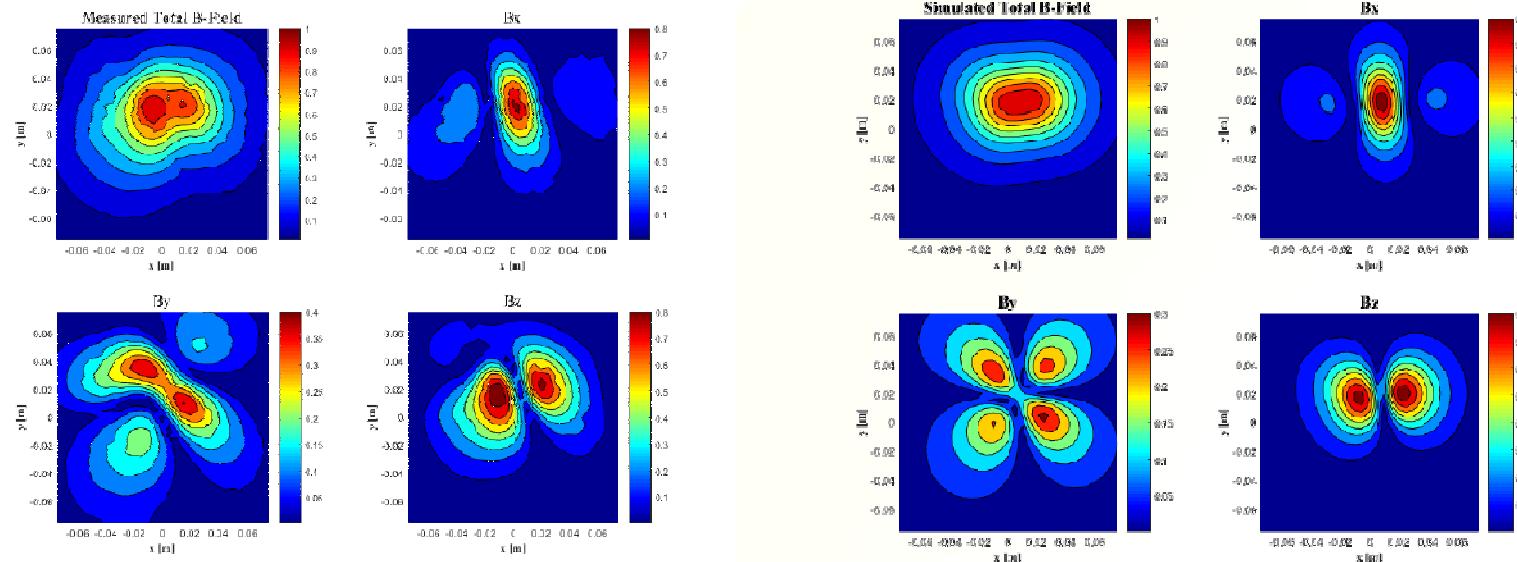
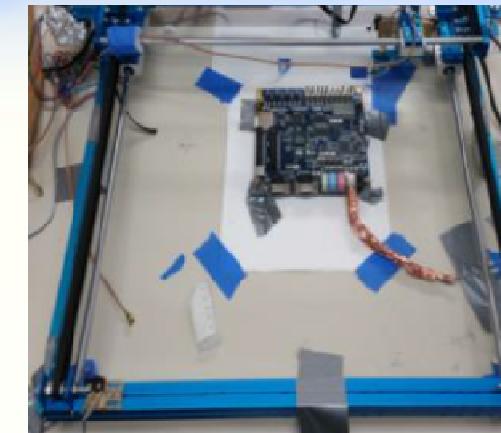
Пример #4: Одређивање процесорског кода на основу ЕМ

- У зависности од инструкција које се извршавају, различити делови матичне плоче су активни
- Сваки струјни пут моделује се магнетском контуром
- Мерење ЕМ поља око процесора



Експериментална поставка и мерење поља

- На FPGA процесору се извршава DIV/ADD пар инструкција
- Измерено поље и оптимизацијом пронађени извори

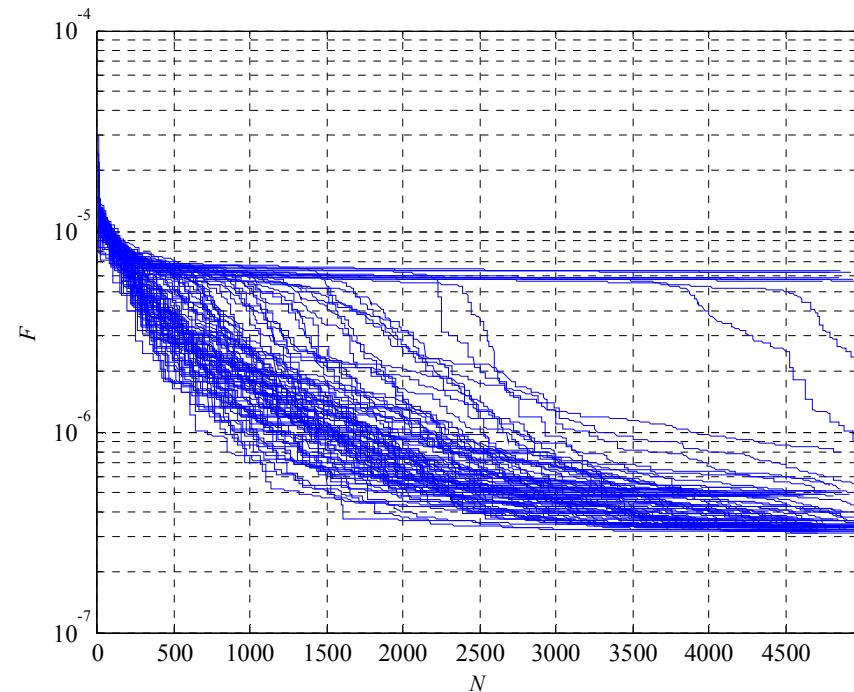
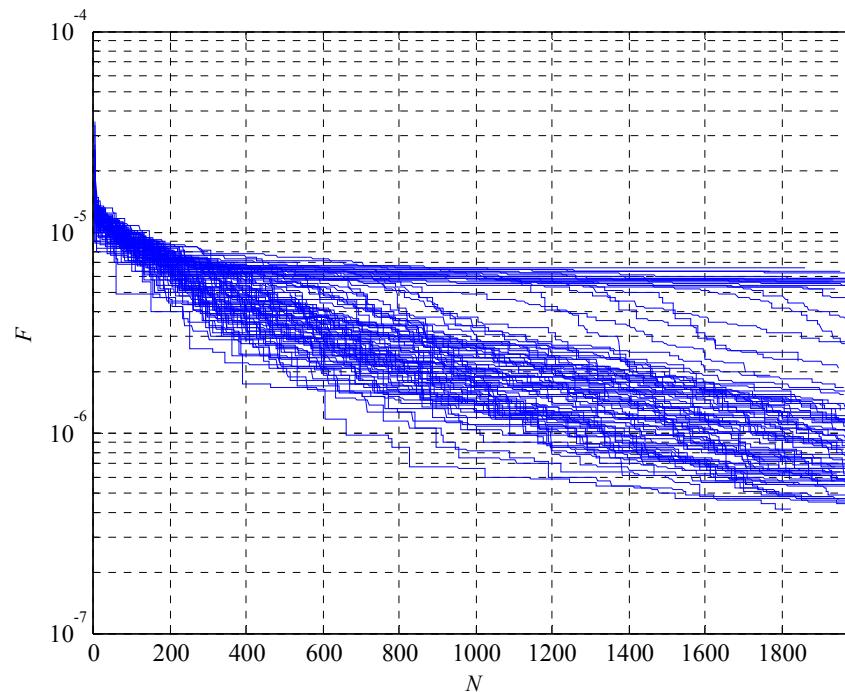


Оптимизациони проблем

- Број извора је оптимизациона променљива и тај број је дискретан
- Позиције извора поља и њихове струје су реални бројеви
- За сваки извор по 7 променљивих: x, y, z , углови ротације ϕ и θ , ефективна вредност струје и фаза (укупно 7 до 21 променљивих)
- У питању је комбинација SAT и NLP оптимизационих проблема!
- Коришћени су PSO, NM симплекс и случајно претраживање

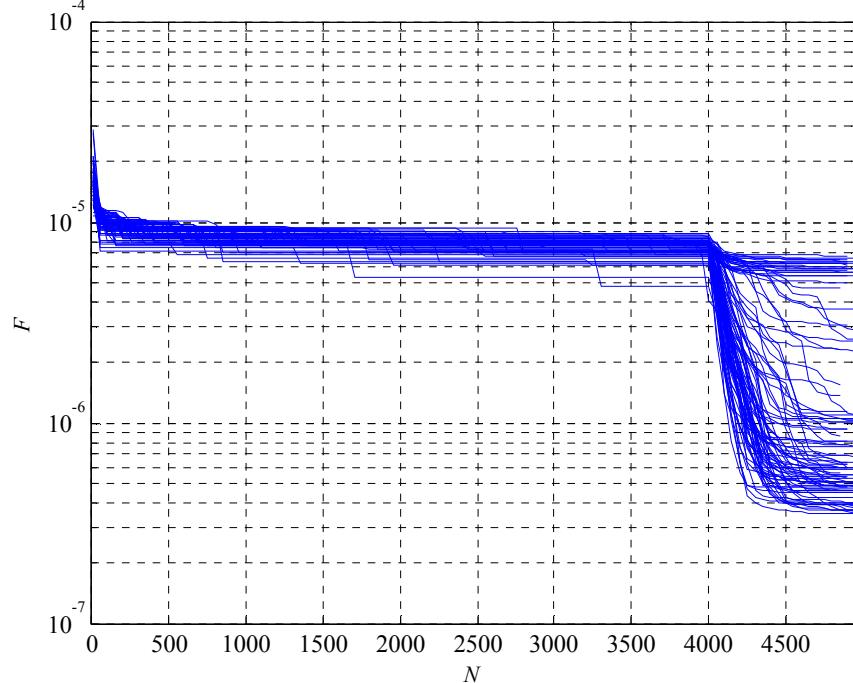
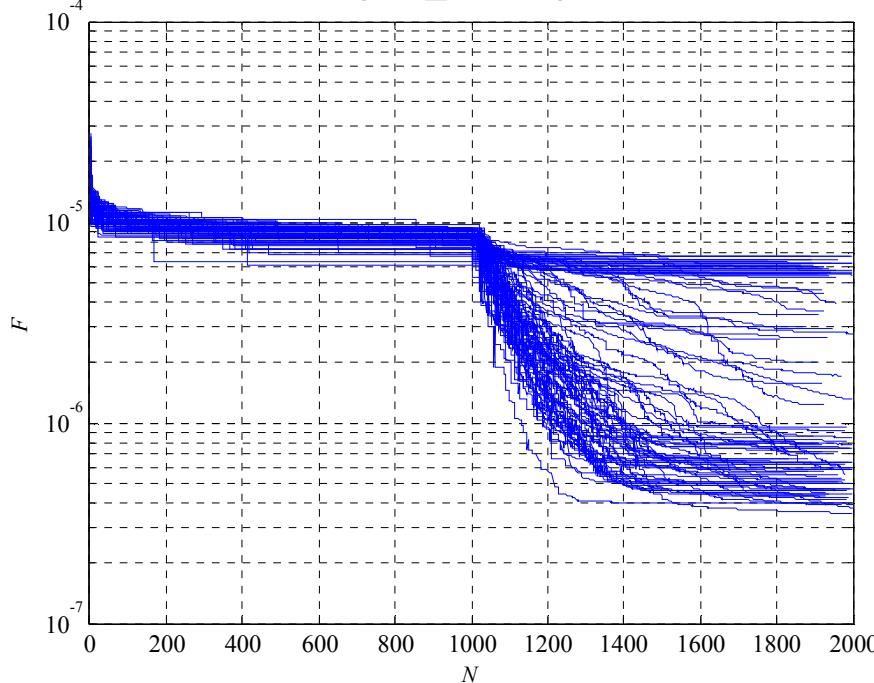
PSO максималан број итерација

- Број агената 25, итерација 2000 и 5000 (100 независних опт.)
- Неке оптимизације и са $2k$ пронађу добро решење, али са $5k$ се у већини случајева пронађе добро решење

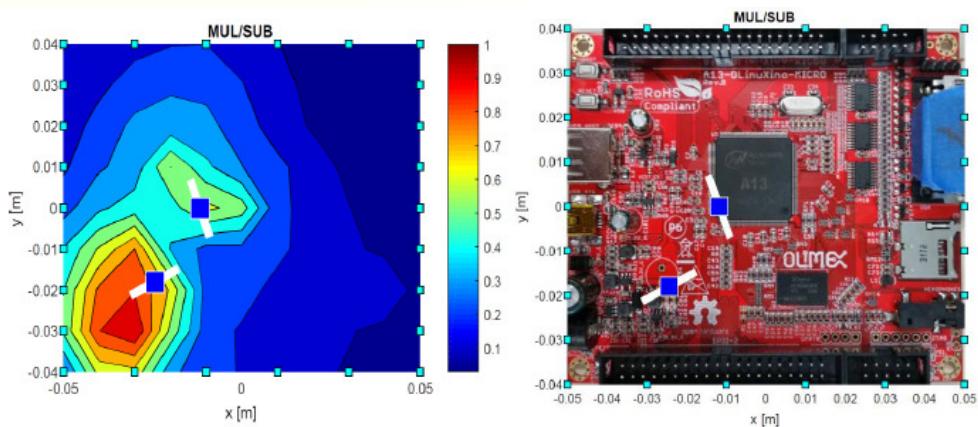
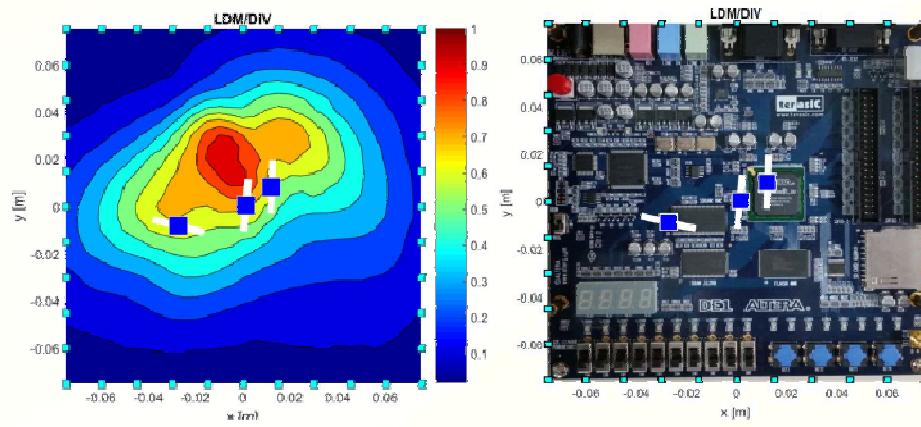
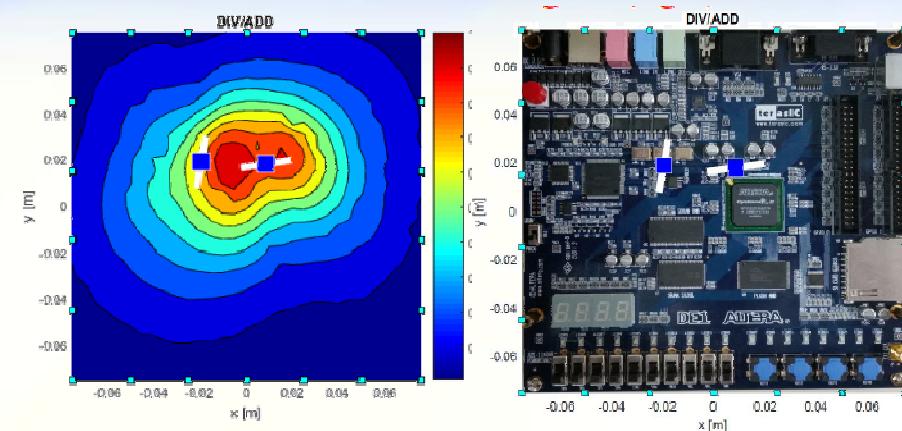
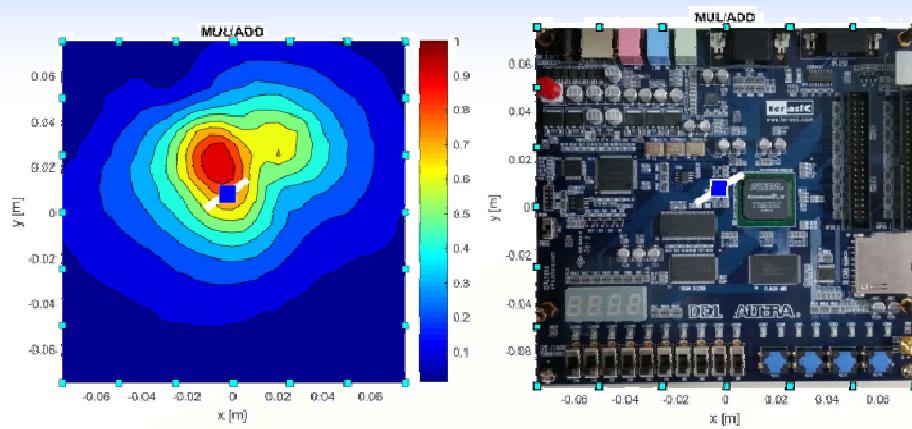


Случајно претраживање и NM симплекс

- Random(1k)+NMsimplex(1k) и
Random(4k)+NMsimplex(1k)
- Да ли су резултати бољи од PSO?



Успешно идентификоване ситуације са 1, 2, 3 активна дела



Оптимизациона хеуристика

Алгоритам	D	Коментар	Главни параметар
Случајно претраживање	-	Грубо претраживање простора. Споро конвергенција. Само у комбинацији са другим алгоритмима.	Унiformна или Гаусова расподела
Систематско претраживање	1-3	Детаљно претраживање простора. Неefикасно за просторе са више од ~3 димензије	Корак за претрагу може бити исти или различит за сваку димензију.
Градијентни алгоритам	1-7	Најбржа конвергенција, уколико је познато добро полазно решење.	Корак за процену градијента.
Симплекс	1-7	Најробуснији локални оптимизациони алгоритам. Може да пронађе најбољи од неколико околних минимума (статистички).	Димензије полазног симплекса.
Симулирано каљење	10+	Глобални оптимизациони алгоритам са спором конвергенцијом.	Почетна температура.
Генетички алгоритам	10+	Тренутно најефикаснији глобални оптимизациони алгоритам, уколико се посматрају проблеми са више од 10 димензија.	Величина популације.
PSO и DE	5-15	Алгоритми који су између глобалних и локалних оптимизационих алгоритама.	Број решења у јату (популацији).

Задатак за вежбе

- Дата је бинарна секвенца дужине 31:
 $X_0 = 1010\ 0001\ 0001\ 1011\ 0000\ 1100\ 1110\ 011$
- Пронађи бинарну секвенцу X , за коју важе следећи услови
 - Секвенца је дужине 31 бит
 - Број нула и јединица у секвенци разликује се за тачно 1
 - Кроскорелација секвенци X_0 и X је већа од -4 и мања од 6 , за сваки цикликчи померај $k = 0, 1, 2, \dots, 30$
 - Аутокорелација секвенце је већа од -18 и мања од 12 , за сваки циклички померај $k = 1, 2, \dots, 30$
- **Кроскорелација** секвенци X_0 и X , једнаких дужина, дефинише се као разлика броја позиција са истим битима и броја позиција са различитим битима, за свако циклично померање секвенце X_0
- **Аутокорелација** секвенце X је кроскорелација секвенце са самом собом

Задатак: пример

- Кроскорелација секвенци
 $\mathbf{x}_0 = 1010 \ 0001 \ 0001 \ 1011 \ 0000 \ 1100 \ 1110 \ 011$
 $\mathbf{x} = 1001 \ 1101 \ 0010 \ 1010 \ 1010 \ 1001 \ 0101 \ 101$
за циклички померај $k = 0$ једнака је -1
- За циклички померај $k = 5$, секвенце \mathbf{X}_0 на десну страну, кроскорелација је једнака 11
 $\mathbf{x}'_0 = 1001 \ 1101 \ 0000 \ 1000 \ 1101 \ 1000 \ 0110 \ 011$
 $\mathbf{x} = 1001 \ 1101 \ 0010 \ 1010 \ 1010 \ 1001 \ 0101 \ 101$
- Аутокорелација секвенце \mathbf{X} , за циклички померај $k = 3$ у десно, је -5
 $\mathbf{x}' = 1011 \ 0011 \ 1010 \ 0101 \ 0101 \ 0101 \ 0010 \ 101$
 $\mathbf{x} = 1001 \ 1101 \ 0010 \ 1010 \ 1010 \ 1001 \ 0101 \ 101$

Задатак: детаљи

- Задатак је испитног типа, по обиму и поставци
 - Запис решења (кодовање оптимизационих променљивих у вектор решења \mathbf{x}) не мора бити експлицитно задат, стога је потребно изабрати један могући запис
 - Описан је оптимизациони проблем, али оптимизациона функција није усвојена (потребно је изабрати једну могућу дефиницију)
 - Оптимизациони алгоритам за решавање није наведен, стога је потребно изабрати један алгоритам или комбинацију алгоритама са курса
- При избору записа најједноставније је изабрати природан запис (SAT, TSP или NLP) за задати оптимизациони проблем
- Проценити број различитих решења у оптимизационом простору и време израчунавања оптимизационе функције, и на основу тога изабрати оптимизациони алгоритам
 - Уколико потпуна претрага траје недопустиво дugo, изабрати оптимизациони алгоритам (искористити кодове и приступе са предавања и вежби) и решити задати проблем
 - Ако је могућа потпуна претрага, урадити је!
- Потребно је пронаћи **једно решење** овог проблема и записати га у пратећи ASCII/TXT фајл
- Решење представити **и у бинарном и у децималном запису!**
- Поред ASCII/TXT фајла проследити и код коришћен за решавање