

Динамичко “програмирање”

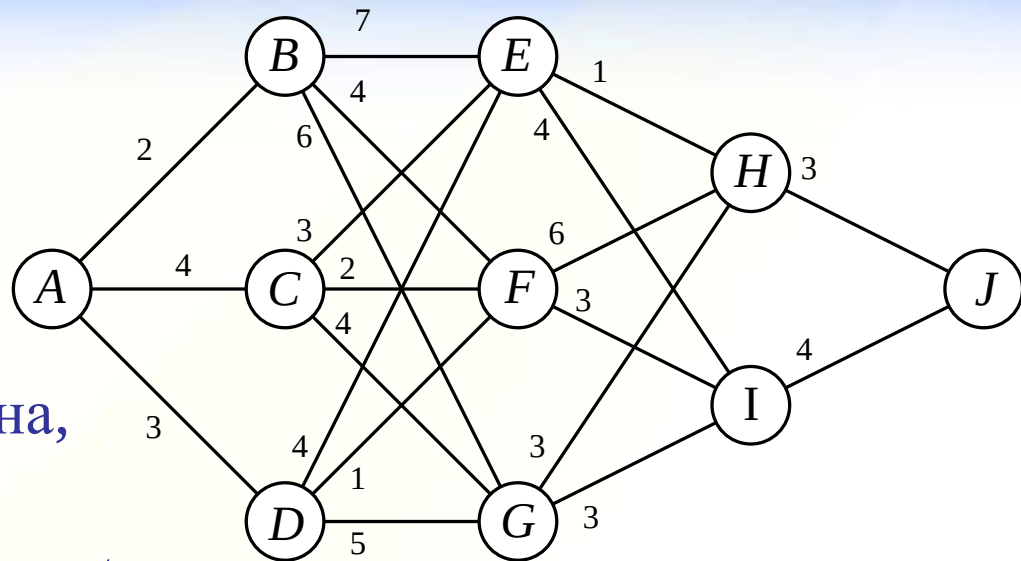
- Richard Bellman "The theory of dynamic programming" *Bulletin of the American Mathematical Society* 60 (6): 503–516 1954.
- Richard Bellman *Dynamic Programming* Princeton University Press 1957.
- D. P. Bertsekas *Dynamic Programming and Optimal Control* (4th ed.) Athena Scientific 2017.
- John von 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 формулација 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

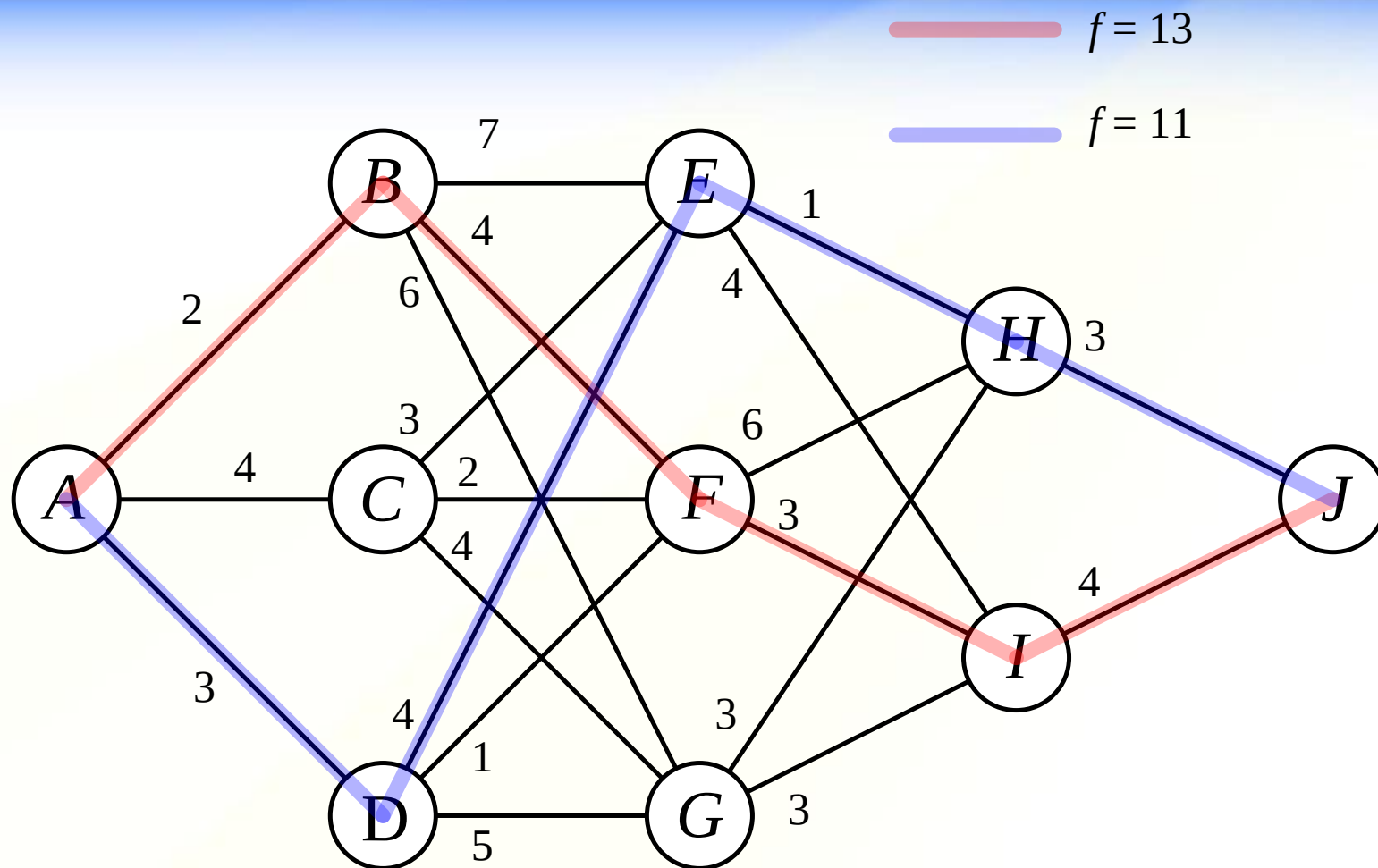
Оптимизациона функција

- 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 појединих 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)

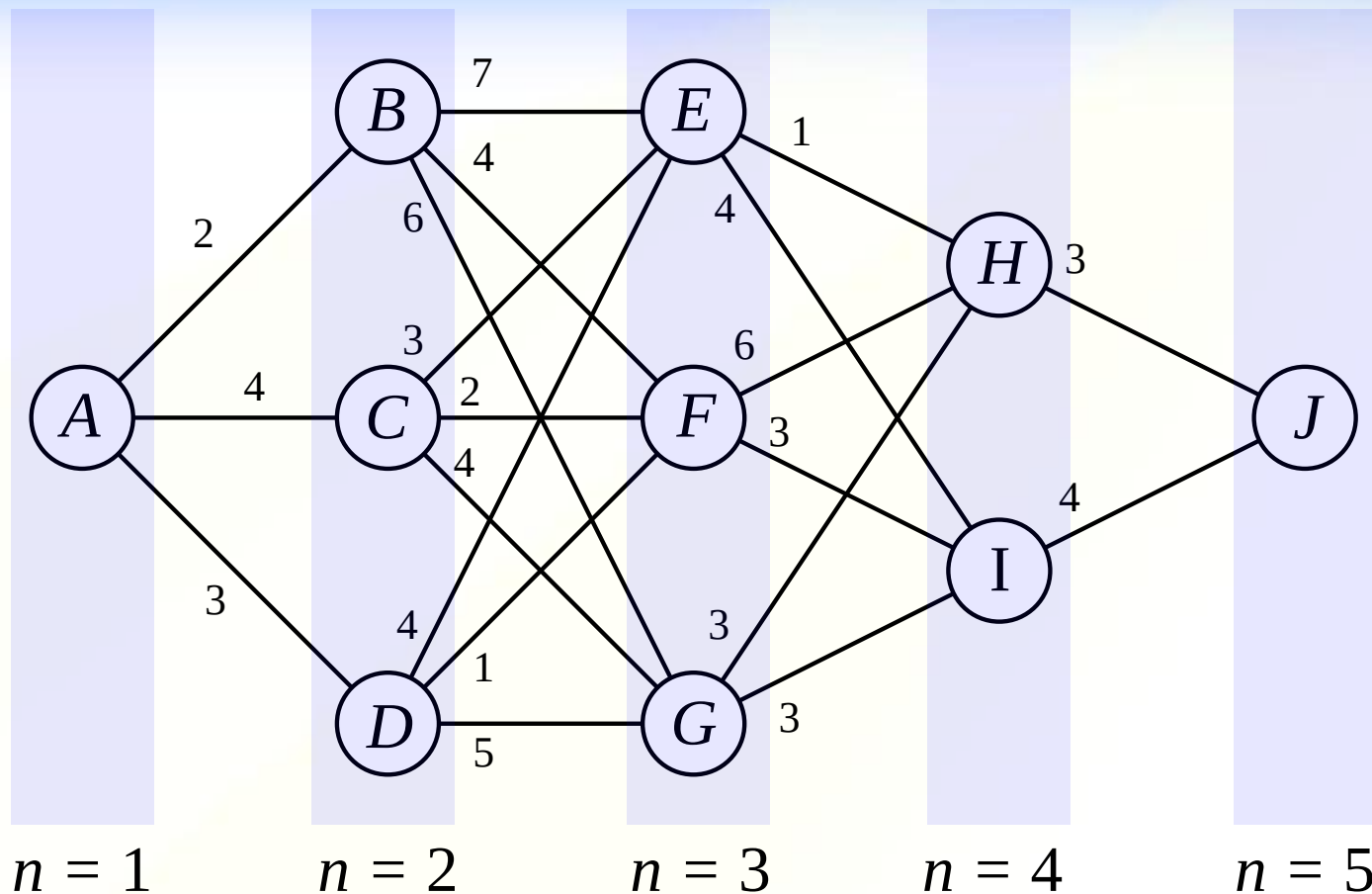
Формални запис динамичког програмирања

- 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

$n = 4$

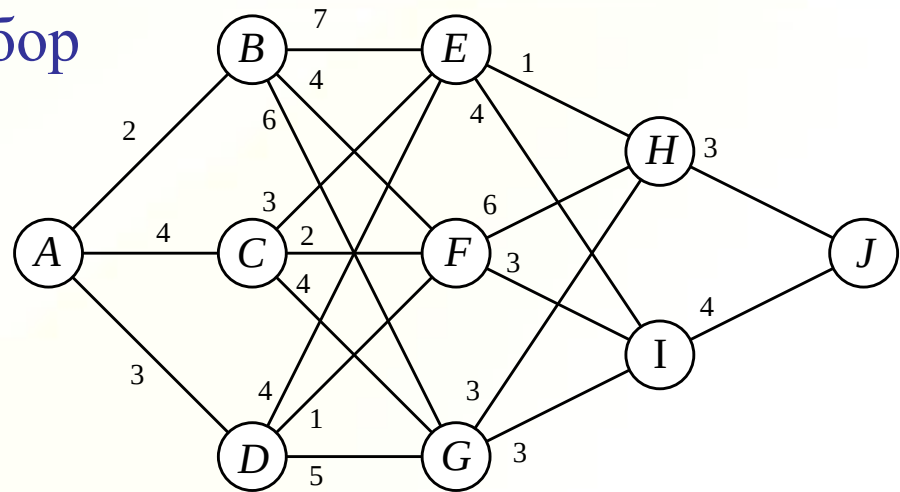
$$f_4^*(s) = f_4(s, x_4) = f_4(s, J) = d(s, J)$$

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

$n = 2$

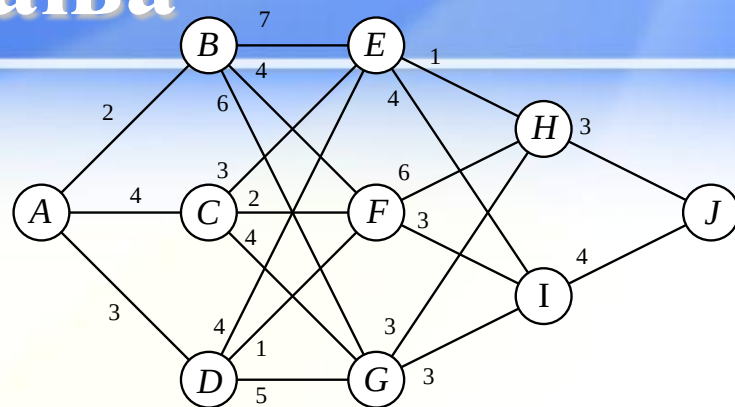
$$f_2(s) = d(s, x_2) + f_3^*(x_2)$$

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

$n = 1$

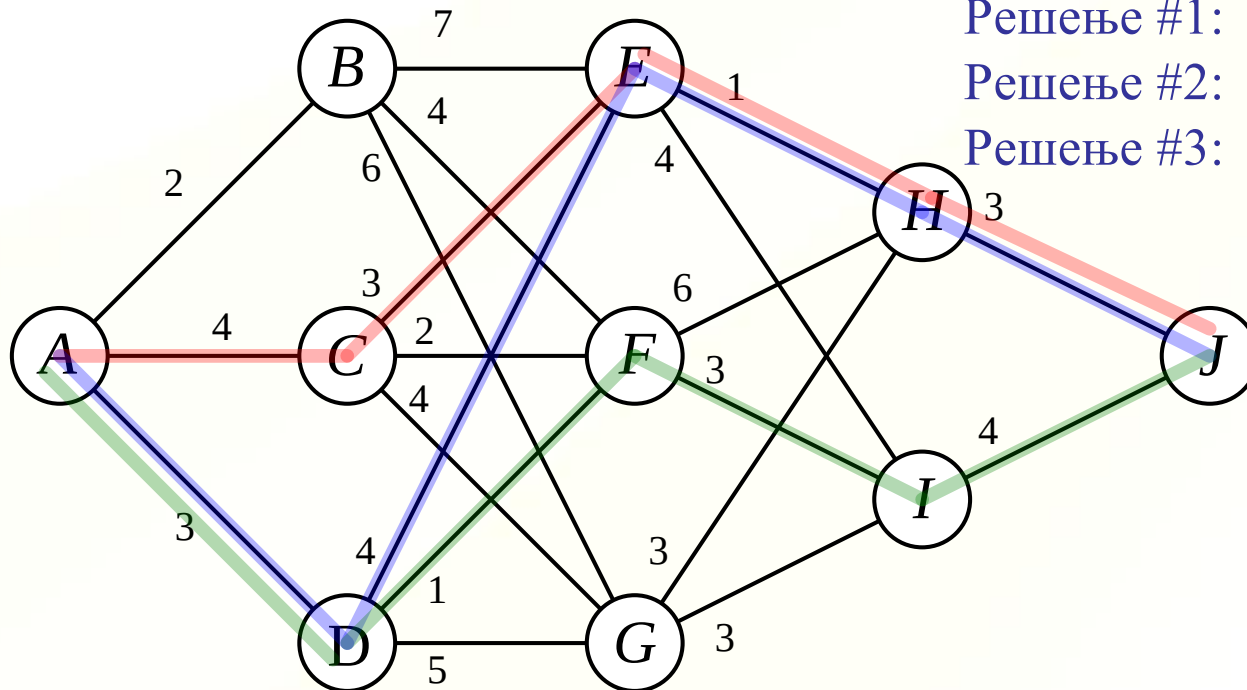
$$f_1(s) = d(s, x_1) + f_2^*(x_1)$$

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



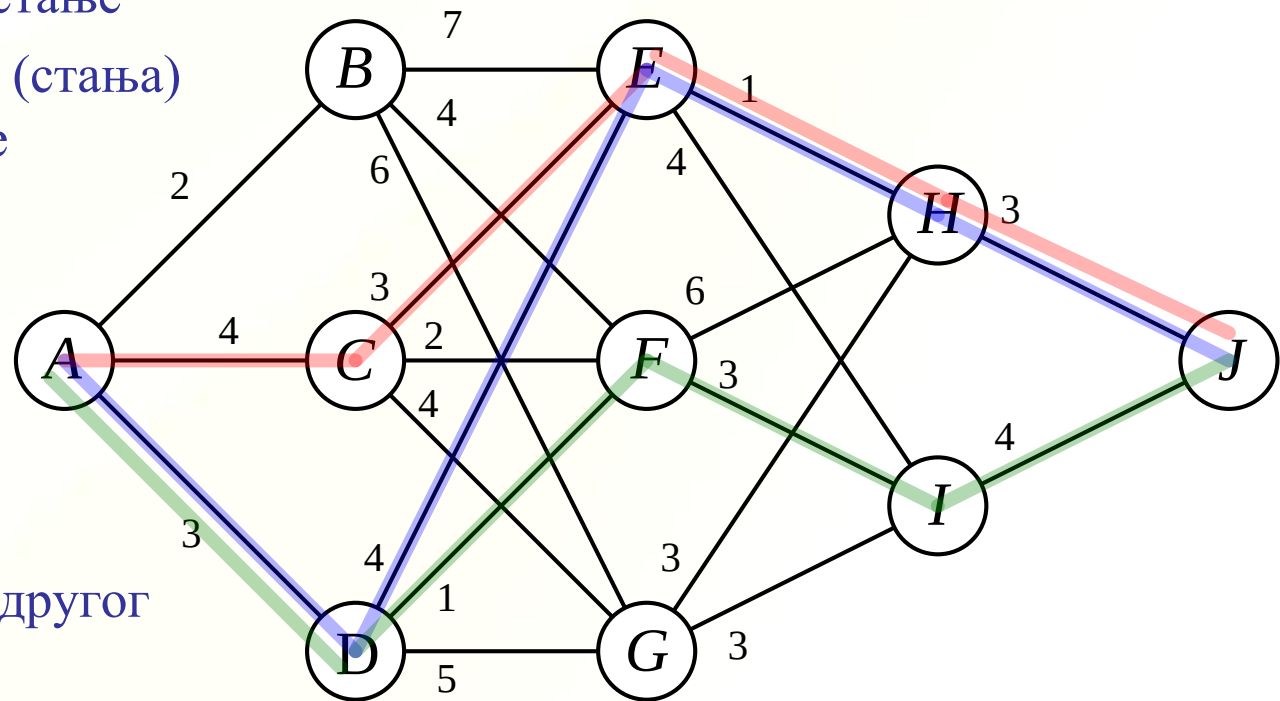
Решење #1: $A \rightarrow C \rightarrow E \rightarrow H \rightarrow J$.

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 $D-...-J$



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

Карактеристике 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)

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 краја ка почетку (енглески: top-down) или
 - од почетка ка крају (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

- Генерисање 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
 - униформне 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 прихвата 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 прихвата 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 $T(k)$: граничне 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 $\Delta E = \Delta f$ али $\Delta f_2 \gg \Delta f_1 \gg \Delta f_3$
ако је T_0 исто у свим случајевима
вероватноћа $p(\Delta E T_0)$ 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 монотono 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) је i_{tot}

Примери смањивања околине

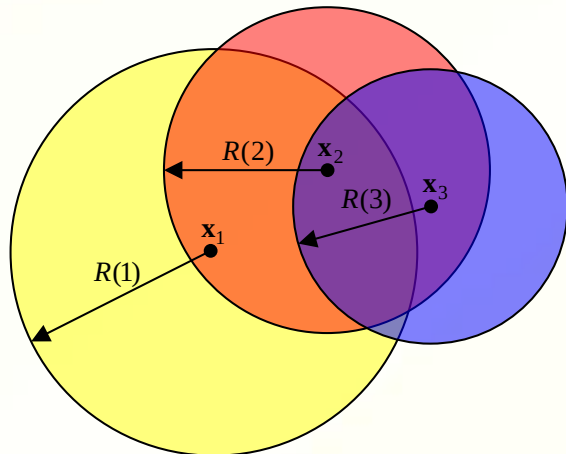
NLP:

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

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

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

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



SAT:

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

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

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

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

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

$$h(1) = 5$$

$$\mathbf{x}_2 = 1\underline{1}0\underline{0}\underline{1}\underline{1}\underline{0}\underline{1}$$

$$h(2) = 3$$

$$\mathbf{x}_3 = \underline{0}10\underline{1}\underline{1}\underline{0}01$$

TSP:

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

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

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

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

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

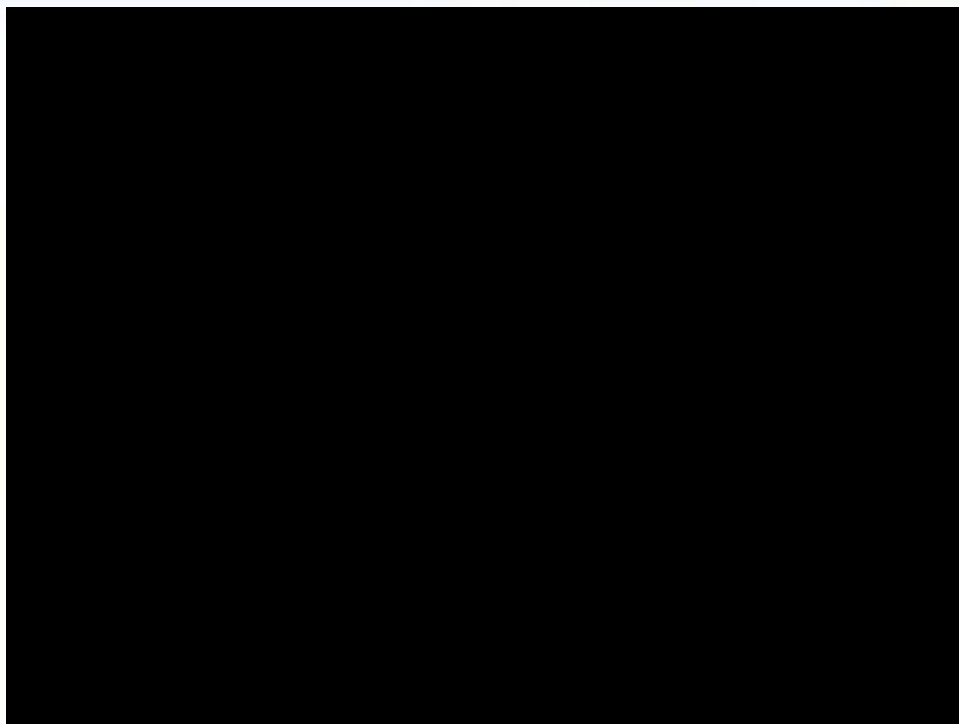
$$d(1) = 2$$

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

$$d(2) = 1$$

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

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

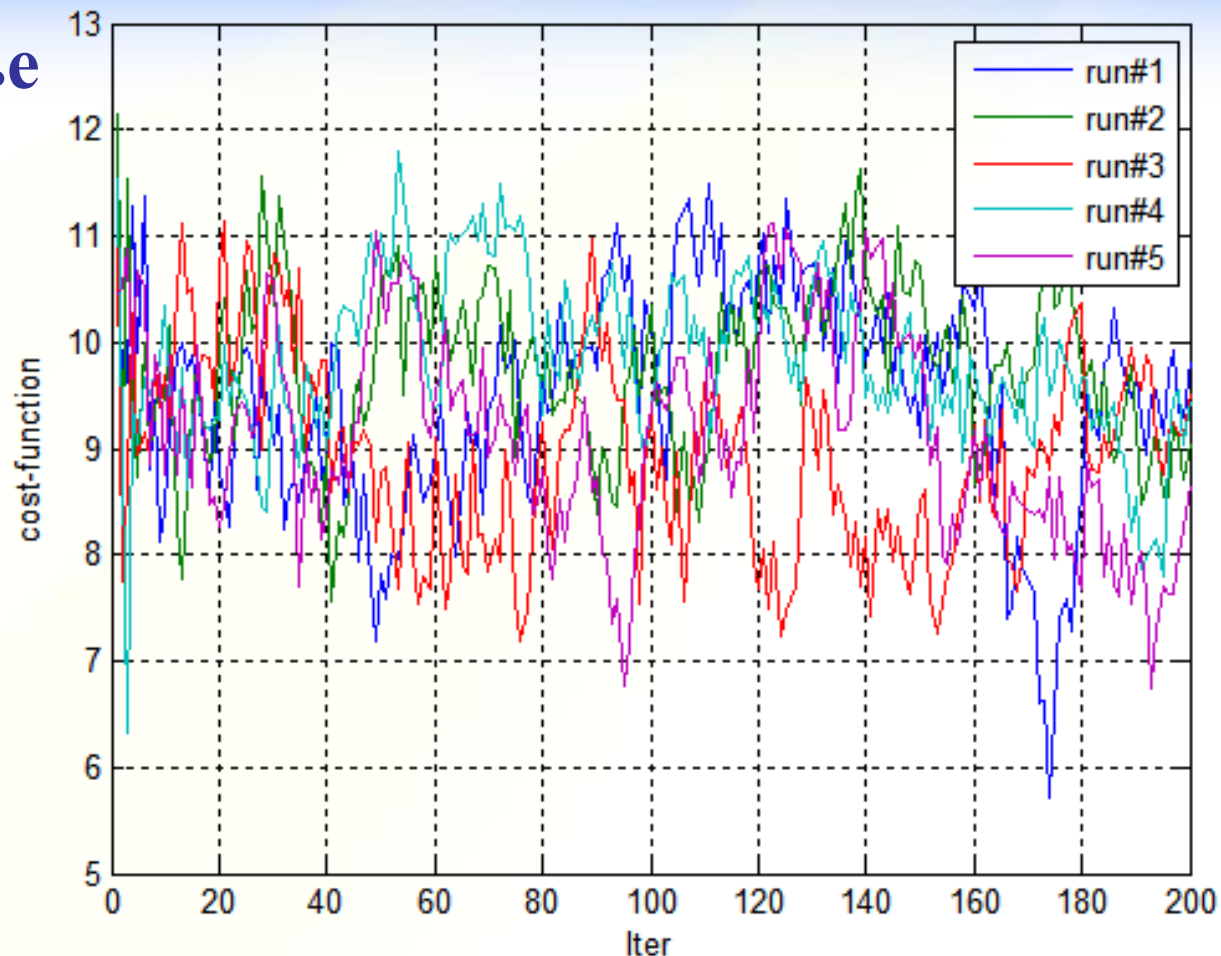
- Краткорочна меморија
 - памти се неколико последњих испитаних решења и њихове околине
- Дугорочна меморија
 - памте се (скоро) сва решења која су испитана
- Средњорочна меморија
 - памти се већи број решења која су пробана
- Избор меморије и дефинисање шта меморија представља зависе од проблема!

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

$$(f_1, f_2, f_3, \dots, f_{k-1}, f_k, f_{k+1}, \dots, f_N)$$

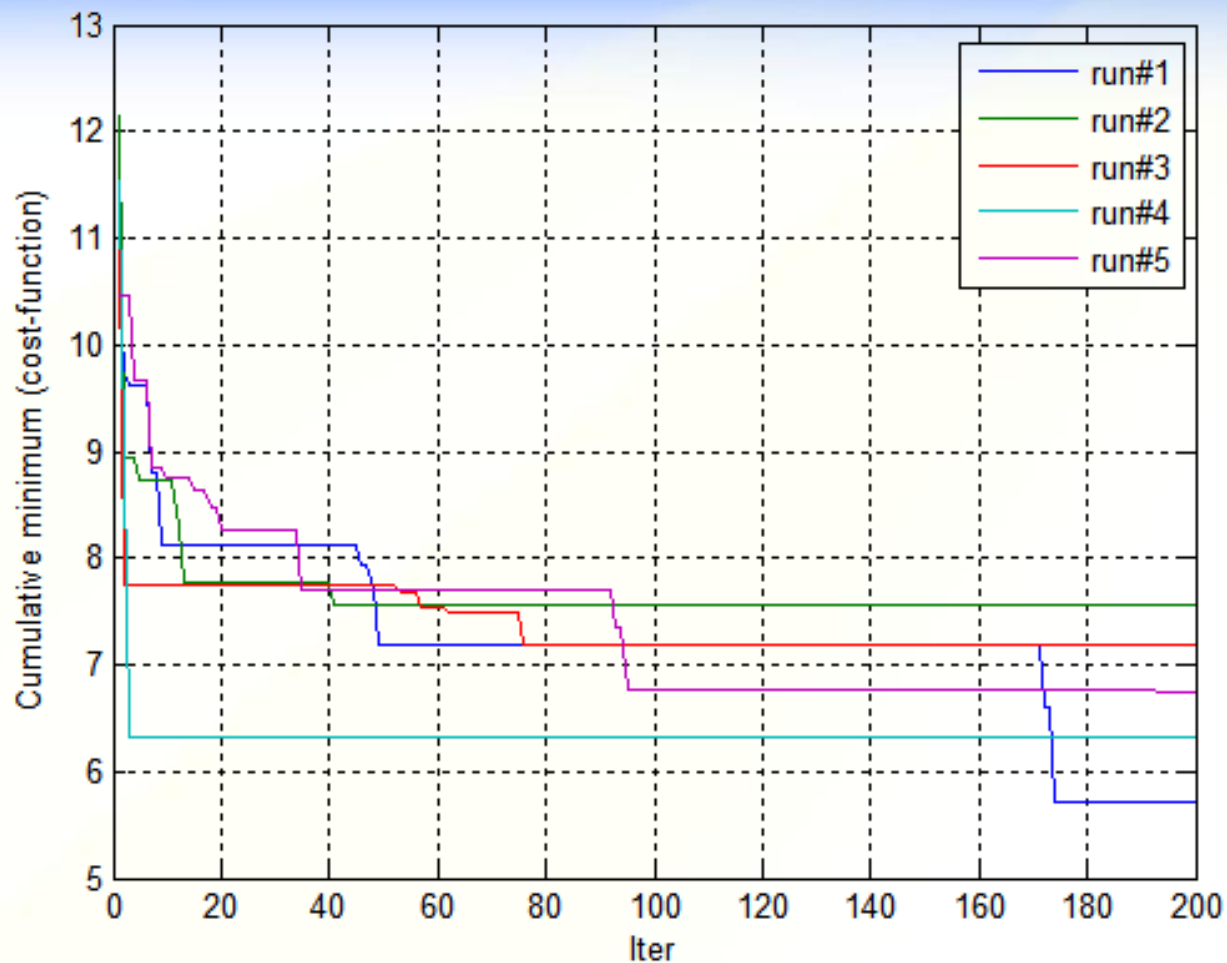
- Кумулативни минимум је

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

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

- 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 монотono нерестућа функција

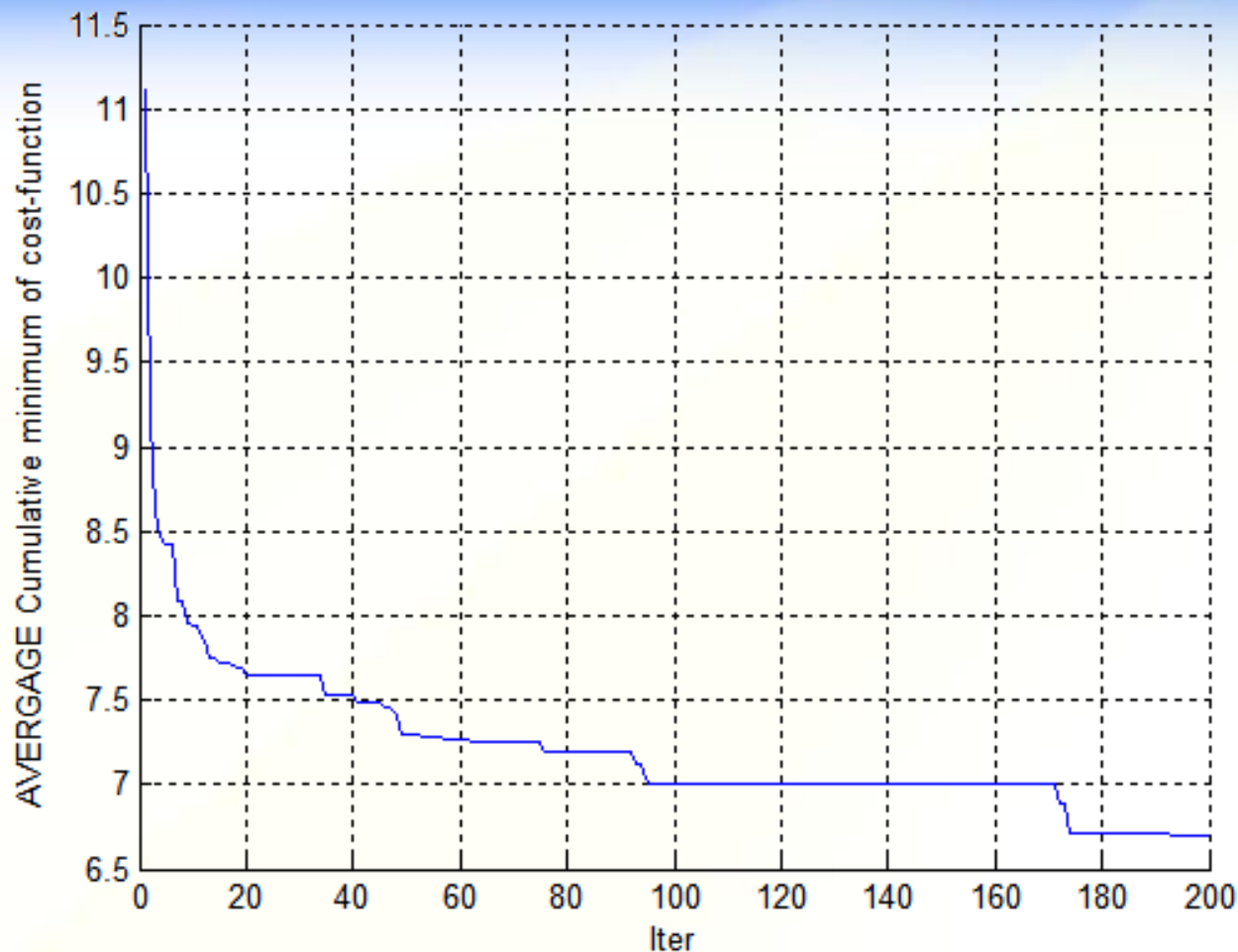
Кумулативни минимум (пример)



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

- На процесору који се састоји од 4 језгра потребно је извршити 100 процеса. Времена трајања процеса, у μs , записана су у низу t

$t=(227, 316, 797, 676, 391, 332, 598, 186, 672, 941, 248, 948, 667, 95, 441, 886, 697, 326, 733, 220, 81, 159, 340, 465, 266, 815, 193, 129, 91, 598, 854, 601, 931, 724, 860, 929, 546, 937, 494, 273, 451, 665, 330, 903, 257, 339, 258, 355, 5, 628, 282, 68, 616, 176, 304, 440, 150, 217, 474, 476, 255, 297, 279, 260, 482, 211, 495, 246, 838, 180, 862, 178, 750, 611, 209, 759, 249, 85, 618, 536, 634, 174, 248, 684, 80, 875, 428, 618, 313, 178, 9, 210, 870, 972, 441, 378, 275, 966, 58, 408)$

- Потребно је распоредити процесе на језгра, тако да за извршавање свих процеса буде потребно најкраће време

Задатак за вежбе

(поступак решавања)

- Усвојени запис решења овог проблема је $\mathbf{x} = (x_1, x_2, \dots, x_D)$, $x_k \in \{1, 2, 3, 4\}$, $k = 1, 2, \dots, D$ (1: процес се извршава на првом језгру, 2: процес се извршава на другом језгру, 3: процес се извршава на трећем језгру, 4: процес се извршава на четвртом језгру)
- Оптимизациона функција је: $f(\mathbf{x}) = \max\{t_1, t_2, t_3, t_4\}$, где је t_{x_k} укупно време извршавања процеса на језгру x_k
- Тражи се минимум

Имплементација

- Написати имплементацију симулираног каљења
- За почетну температуру узети $T_0 \sim 100 \cdot 500$
- Максималан број итерација (израчунавања оптимизационе функције) је 10 000 за једно покретање симулираног каљења
- Пустити 10 пута симулирано каљење из претходно добијене најбоље тачке (поновљено каљење)
- Пустити 20 независних покретања поновљеног каљења 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 у semilogx размери
- Решење садржи
 - ASCII фајл: низ дужине 100 који одговара најбољем пронађеном решењу (записано искључиво у дефинисаном формату $\mathbf{x}=(x_1, x_2, \dots x_D)$) и минималну вредност оптимизационе функције
 - графике у png или jpg формату
 - код