Hledání nejkratší trasy aneb Sex, chlast a C++

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava

Prezentace ke dni 12. února 2020



Osnova přednášky

Hledání nejkratší trasy aneb Sex, chlast a C++

- Zadání úlohy – chlast
- Metoda řešení – sex
- Implementace – C++
- Výsledky

Motivační přednáška

Upozornění

1. Tato přednáška je pouze motivační. Má za cíl demonstrovat, co lze se znalostmi programování a algoritmů po prvním semestru zvládnout.
2. Náplň této přednášky **nebude** součástí žádného testu či písemné práce.

Život studentský, život veselý...

- Rodiče daleko...
- Plnoletost...
- Kapesné či vlastní příjem...
- Minimální povinnosti...
- Mladý organismus schopný rychlé regenerace...
- Tak proč nejít do hospody?!
- Důvod se vždycky najde...
- A když nenajde, tak se musí vymyslet...

Tour de pub

- klasická kolejní zábava,
- závod spočívá v návštěvě předem daných hospod,
- závodníci se přesunují výhradně pěšky,
- v každé hospodě je nutno vypít pivo nebo/a, panáka,
- závodí buď jednotlivci nebo týmy (většinou smíšené),
- trasa nemusí být přesně stanovena – je možné volit pořadí navštívených hospod,
- vítězí ten, kdo dorazí jako první do cíle s potvzenou konzumací.

Tour de pub Olomouc 2017

- předpokládejme, že se zúčastníme Tour de pub v Olomouci,
- máme daný seznam soutěžních hospod,
- v každé hospodě si dá závodník jedno pivo,
- start a cíl je v hospodě Doga,
- závodníci se přesunují výhradně pěšky, tramvaje či taxíky jsou zakázány,
- pořadí hospod je na každém závodníkovi.

Zadání úlohy

Navrhněte optimální trasu pro tento závod.

Soutěžní hospody

Id	Restaurace	Adresa, web
0	**Drápal**	Havlíčkova 1, 779 00 Olomouc `www.restauracedrapal.cz/`
1	**Plán B**	Palachovo nám. 620/1, 779 00 Olomouc `www.bar-planb.cz/`
2	**Pivovar Moritz**	Nešverova 683/2, 779 00 Olomouc `www.hostinec-moritz.cz/`
3	**U červeného volka**	Dolní nám. 26/39, 779 00 Olomouc `www.ucervenehovolka.cz/`
4	**U zlaté koule**	tř. Míru 168/15, 779 00 Olomouc `www.uzlatekoule.cz/`
5	**Mtrojka**	Masarykova tř. 889/3, 779 00 Olomouc `www.m3olomouc.cz/`
6	**U Musea**	1.máje 827/8, 779 00 Olomouc `www.ponorka.com/`
7	**U Bystřičky**	Masarykova tř. 802/ 10, 779 00 Olomouc

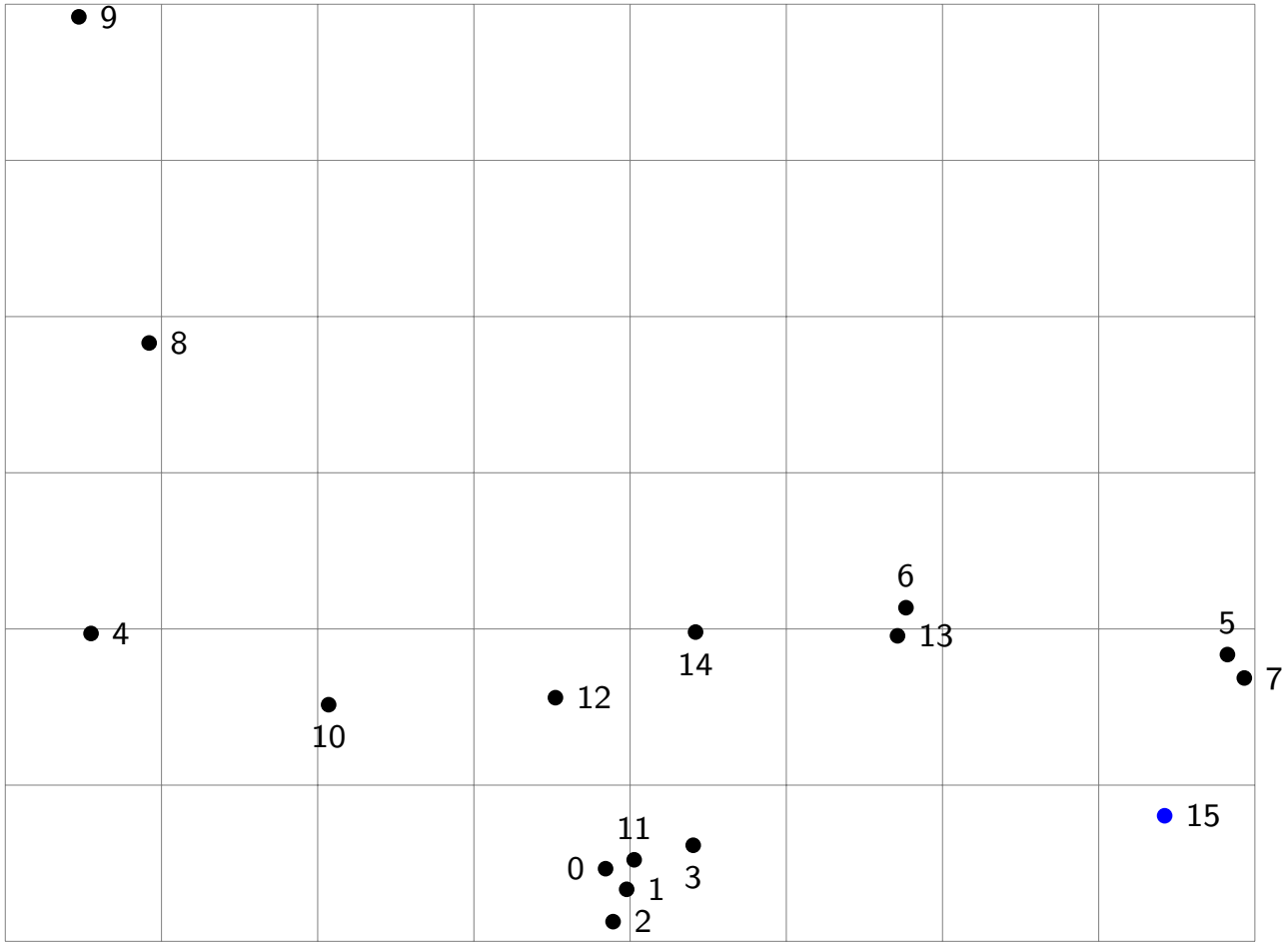
Soutěžní hospody (pokrač.)

Id	Restaurace	Adresa, web
8	**U Anči**	Na Trati 213/59, 779 00 Olomouc
9	**U kapličky**	Břetislavova 29/3, 779 00 Olomouc
10	**U Fleka**	Palackého 132/25, 779 00 Olomouc `www.ufleka.cz/`
11	**Morgan**	Mlýnská 2, 779 00 Olomouc `morgansrestaurant.cz/`
12	**Riegrovka**	Riegrova 22, 779 00 Olomouc `riegrovka.eu/`
13	**Svatováclavský pivovar**	Mariánská 845/4, 779 00 Olomouc `www.svatovaclavsky-pivovar.cz/`
14	**Jazz Tibet club**	Sokolská 48, Olomouc `www.jazzclub.olomouc.com/`
15	**Doga**	Šmeralova 1116/10, 779 00 Olomouc `www.pivnicedoga.cz/`

Mapa soutěžních hospod



Schematická mapa soutěžních hospod



Tour de pub Olomouc 2017 – analýza zadání

- Máme navrhnout optimální trasu pro závod. Co je to **optimální trasa**?
- Předpokládejme, že
  1. průměrná doba vypití jednoho piva je zhruba konstantní a
  2. průměrná rychlost chůze závodníka je také zhruba konstantní.
- Soutěžních piv je daný počet, tuto dobu asi příliš neovlivníme.
- Rychlost chůze asi nejsme schopni nijak výrazně zvýšit, ona se bude spíše snižovat. . .
- Můžeme ale volit pořadí navštívených hospod.
- Asi nebude vhodné pobíhat z jednoho konce města na druhý, ale brát hospody „nějak postupně“.
- Budeme-li brát hospody „nějak postupně“ ujdeme pravděpodobně menší vzdálenost, než když budeme chaoticky pobíhat po městě.

Tour de pub Olomouc 2017 – analýza zadání (pokrač.)

- Jinak řečeno, čím kratší trasu zvolíme, tím rychleji se nám podaří závod dokončit.
- Formálněji řečeno, hledáme nejkratší trasu ze startovní hospody přes všechny soutěžní hospody zpět do cílové hospody.

Nástroje pro hledání nejkratší soutěžní trasy

Musíme vymyslet následující:

- způsob, jak reprezentovat trasu,
- způsob, jak určit délku libovolné trasy a
- algoritmus řešení úlohy.

Formální specifikace zadání I.

- Počet soutěžních hospod N, v našem případě $N = 16$.
- Hospody budou očíslovány $0, \ldots, N - 1$.
- Startovní a cílová hospoda má číslo $N - 1$, v našem případě 15.
- Volit pořadí můžeme jen u hospod $0, \ldots, N - 2$, v našem případě $0, \ldots, 14$.

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.
- Takže záleží na pořadí hospod na trase.

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.
- Takže záleží na pořadí hospod na trase.
- Jak matematicky reprezentovat různé možnosti „přeházení“ či „přeuspořádání“ nějaké množiny? Konečné množiny čísel $0, \ldots, 14$?

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.
- Takže záleží na pořadí hospod na trase.
- Jak matematicky reprezentovat různé možnosti „přeházení“ či „přeuspořádání“ nějaké množiny? Konečné množiny čísel $0, \ldots, 14$?
- No přece pomocí **permutace**!

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.
- Takže záleží na pořadí hospod na trase.
- Jak matematicky reprezentovat různé možnosti „přeházení“ či „přeuspořádání“ nějaké množiny? Konečné množiny čísel $0, \ldots, 14$?
- No přece pomocí **permutace**!
- Permutace jednoznačně přiřazuje jednomu přirozenému číslu jiné přirozené číslo. Formálně jde o zobrazení $\pi : \mathbb{N} \to \mathbb{N}$.

Reprezentace trasy

- Co je pro reprezentaci nějaké trasy rozhodující? Čím se jednotlivé trasy od sebe liší?
- Startovní a cílová hospoda je neměnná.
- Délka? Ne! Délka trasy je jen důsledkem toho jak jsou hospody na trase uspořádány – jestli budeme pobíhat po městě nebo ne.
- Takže záleží na pořadí hospod na trase.
- Jak matematicky reprezentovat různé možnosti „přeházení“ či „přeuspořádání“ nějaké množiny? Konečné množiny čísel $0, \ldots, 14$?
- No přece pomocí **permutace**!
- Permutace jednoznačně přiřazuje jednomu přirozenému číslu jiné přirozené číslo. Formálně jde o zobrazení $\pi : \mathbb{N} \to \mathbb{N}$.
- V našem případě, permutace bude číslu označujícímu pořadí hospody na trase přiřazovat identifikační číslo hospody. Například takto:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 6 & 14 & 13 & 9 & 12 & 4 & 3 & 1 & 0 & 11 & 2 & 10 & 8 & 5 & 7 \end{pmatrix}$$

Tour de pub Olomouc 2017 – analýza zadání

Nástroje pro hledání nejkratší soutěžní trasy

Musíme vymyslet následující:

- způsob, jak reprezentovat trasu, ✓
- způsob, jak určit délku libovolné trasy a
- algoritmus řešení úlohy.

Výpočet délky trasy

- Algoritmus výpočtu délky trasy bude přiřazovat každé trase nějaké kladné, patrně reálné, číslo.
- Z matematického pohledu se tedy bude jednat o funkci, nazvěme ji d, které každé permutaci π, reprezentující libovolnou trasu, přiřadí číslo z množiny $\mathbb{R}$.
- Je zřejmé, že délku celé trasy vypočteme jako součet jednotlivých úseků „od hospody k hospodě“.
- Budeme tedy potřebovat tabulku vzdáleností od hospody i k hospodě j.
- Jak reprezentovat matematicky tabulku? Budeme potřebovat něco, co má dva indexy – řádek a sloupec.
- Řešení? **Matice**! Matice D jednoznačně přiřazuje uspořádané dvojici přirozených čísel (i, j) jiné, většinou reálné, číslo $D_{i,j}$. Formálně jde opět o zobrazení, $D : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$.

Výpočet délky trasy (pokrač.)

- Matici D můžeme znázornit obvyklým způsobem tj. pomocí mřížky čísel ohraničených závorkami.

$$D = \begin{pmatrix}
0 & 100 & 160 & 320 & 1400 & 1600 & 890 & 1700 & 1800 & 2600 & 820 & 110 & 480 & 900 & 650 & 1600 \\
100 & 0 & 80 & 280 & 1500 & 1700 & 970 & 1700 & 1900 & 2700 & 890 & 190 & 560 & 980 & 730 & 1500 \\
160 & 80 & 0 & 310 & 1500 & 1800 & 1000 & 1800 & 1900 & 2800 & 900 & 270 & 640 & 1100 & 810 & 1500 \\
320 & 280 & 310 & 0 & 1700 & 1500 & 760 & 1500 & 2000 & 2700 & 1100 & 210 & 550 & 780 & 600 & 1300 \\
1400 & 1500 & 1500 & 1700 & 0 & 2600 & 1900 & 2600 & 780 & 1700 & 580 & 1500 & 1100 & 1900 & 1500 & 3100 \\
1600 & 1700 & 1800 & 1500 & 2600 & 0 & 750 & 60 & 2800 & 3200 & 2100 & 1600 & 1600 & 840 & 1300 & 1000 \\
890 & 970 & 1000 & 760 & 1900 & 750 & 0 & 770 & 2000 & 2600 & 1400 & 870 & 860 & 150 & 510 & 1900 \\
1700 & 1700 & 1800 & 1500 & 2600 & 60 & 770 & 0 & 2800 & 3200 & 2100 & 1600 & 1600 & 860 & 1300 & 1000 \\
1800 & 1900 & 1900 & 2000 & 780 & 2800 & 2000 & 2800 & 0 & 950 & 1000 & 1900 & 1400 & 2100 & 1700 & 3500 \\
2600 & 2700 & 2800 & 2700 & 1700 & 3200 & 2600 & 3200 & 950 & 0 & 2000 & 2600 & 2200 & 2600 & 2300 & 3900 \\
820 & 890 & 900 & 1100 & 580 & 2100 & 1400 & 2100 & 1000 & 2000 & 0 & 880 & 620 & 1400 & 980 & 2300 \\
110 & 190 & 270 & 210 & 1500 & 1600 & 870 & 1600 & 1900 & 2600 & 880 & 0 & 460 & 880 & 630 & 1400 \\
480 & 560 & 640 & 550 & 1100 & 1600 & 860 & 1600 & 1400 & 2200 & 620 & 460 & 0 & 880 & 570 & 1700 \\
900 & 980 & 1100 & 780 & 1900 & 840 & 150 & 860 & 2100 & 2600 & 1400 & 880 & 880 & 0 & 540 & 1500 \\
650 & 730 & 810 & 600 & 1500 & 1300 & 510 & 1300 & 1700 & 2300 & 980 & 630 & 570 & 540 & 0 & 1700 \\
1600 & 1500 & 1500 & 1300 & 3100 & 1000 & 1900 & 1000 & 3500 & 3900 & 2300 & 1400 & 1700 & 1500 & 1700 & 0
\end{pmatrix}$$

- Čtvercová matice D řádu N bude tedy představovat matici vzdáleností jednotlivých hospod.

Výpočet délky trasy (pokrač.)

- Prvek matice $D_{i,j}$ představuje vzdálenost mezi i-tou a j-tou soutěžní hospodou.
- Funkci d můžeme s pomocí matice D zapsat takto:

$$d(\pi) = D_{N-1,\pi(0)} + \sum_{i=0}^{N-3} D_{\pi(i),\pi(i+1)} + D_{\pi(N-2),N-1},$$

kde

- $D_{N-1,\pi(0)}$ je délka úseku ze startovací hospody do první hospody na trase,
- $\sum_{i=0}^{N-3} D_{\pi(i),\pi(i+1)}$ je součet všech úseků mezi hospodami na trase a
- $D_{\pi(N-2),N-1}$ je délka úseku z poslední hospody na trase do cílové hospody.

Tour de pub Olomouc 2017 – analýza zadání

Nástroje pro hledání nejkratší soutěžní trasy

Musíme vymyslet následující:

- způsob, jak reprezentovat trasu, ✓
- způsob, jak určit délku libovolné trasy a ✓
- algoritmus řešení úlohy.

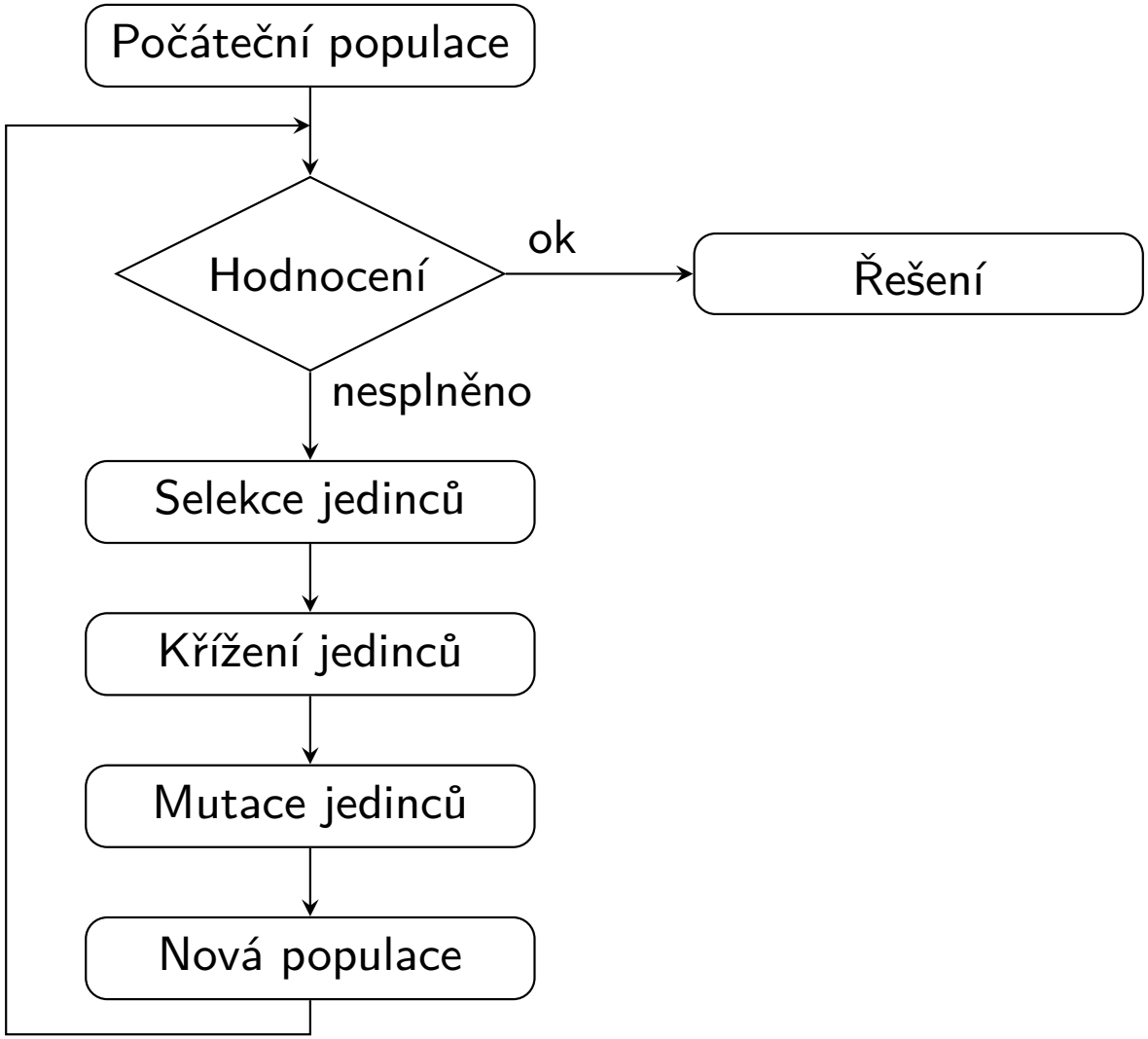
Metoda řešení – sex

- Přímočaré řešení hrubou silo nemá šanci na úspěch, permutací je příliš mnoho, $n!$. Výsledku bychom se nemuseli dočkat.
- Použít nějakou soustavu rovnic? Jakou? Co by mělo pro pořadí hospod platit?
- Umíme v podstatě jen měřit délku trasy a říci, které ze dvou tras je ta kratší.
- Nešlo by nějak začít u „normálního“ nezajímavého řešení a nějak ho vylepšovat?
- Nešlo by řešení nějak šlechtit?
- Když to funguje v přírodě, proč by to nemohlo fungovat v počítači.
- Řešením je použít darwinovskou evoluci.

Metoda řešení – evoluční algoritmy

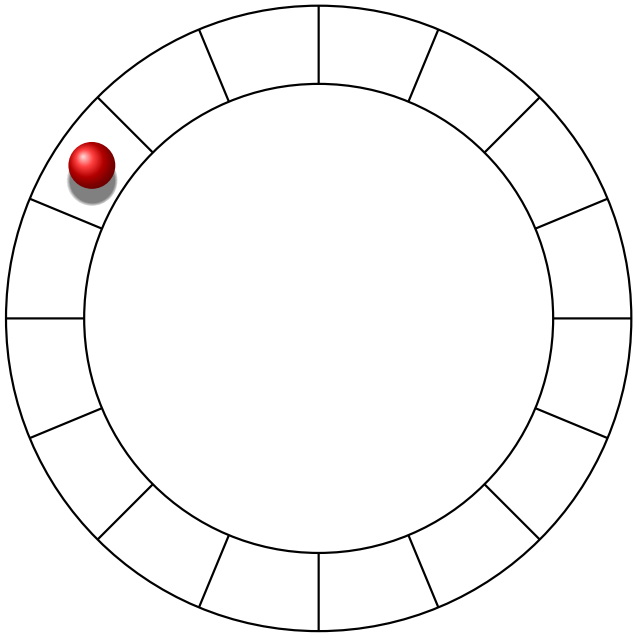
- Evoluční algoritmy se řadí mezi biologicky inspirované algoritmy.
- Zde použitý **genetický algoritmus** je jedním z mnoha evolučním algoritmů.
- Základem je jedinec, jehož genetická informace představuje řešení problému.
- Každý jedinec je ohodnocen svou zdatností – fitness. Snažíme se fitness maximalizovat.
- Algoritmus pracuje s populací jedinců, které přežívá s generace na generaci.
- Tři základní mechanismy pro vývoj populace:
  - Selekce – jedinci jsou vybráni pro křížení na základě své fitness, lepší jedinci mají přednost.
  - Křížení – kombinace genetické informace dvou jedinců.
  - Mutace – malá náhodná změna u nového jedince.
- Do nové generace postupují jen ti nejzdatnější jedinci.
- Algoritmus tedy napodobuje šlechtění rostlin, živočichů atd.

Genetický algoritmus – schéma



Selekce

- Jak provést selekci jedinců?
- Náhodně? Zcela náhodně? Prostě ruleta?

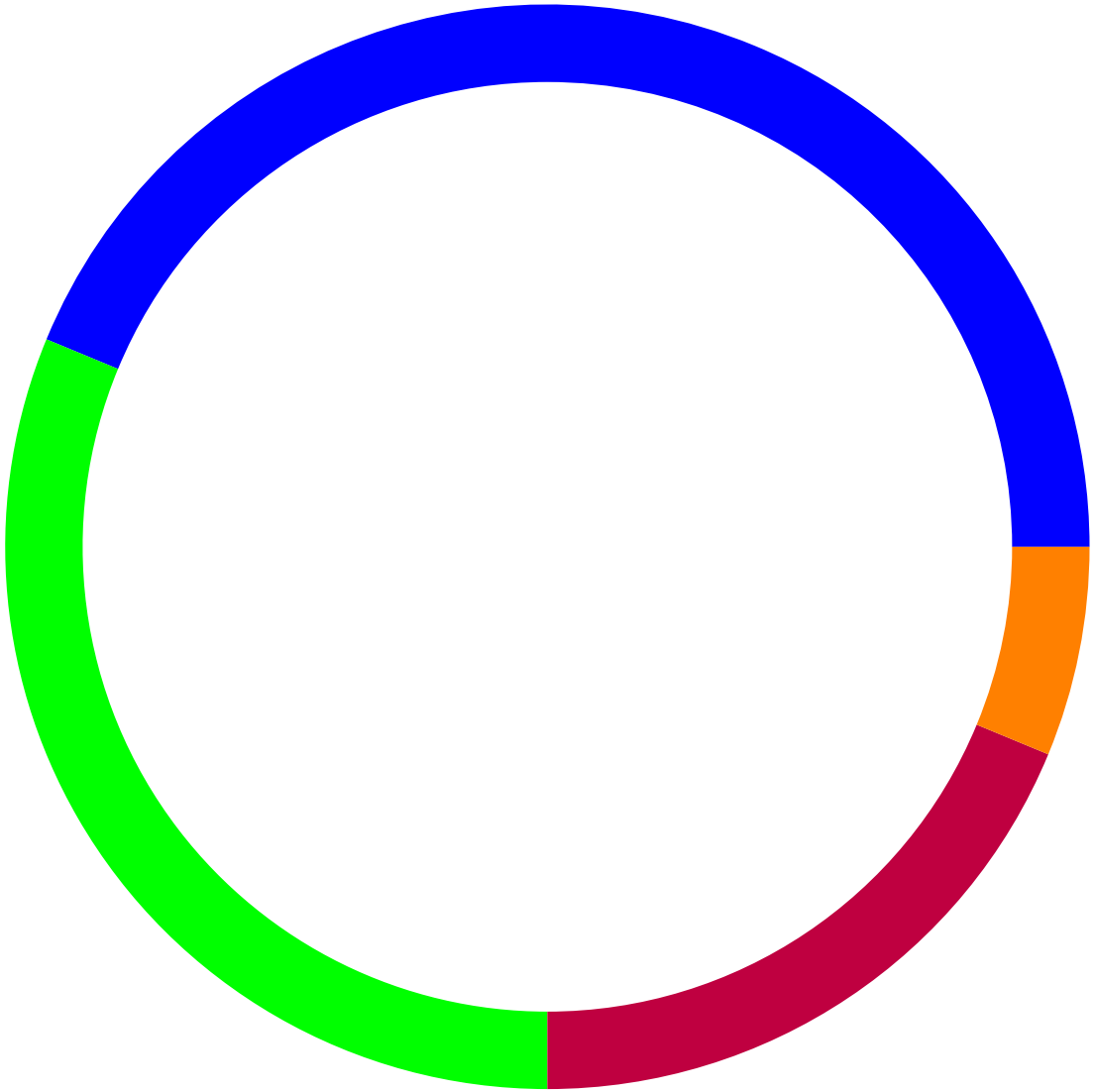


- Asi ne – se potřeba podporovat jedince s vysokým fitness, kvalitnější rodiče budou mít pravděpodobně kvalitní potomstvo.
- Vyřadit ale méně kvalitní jednice úplně z rozmnožování? Můžou nést malý kousek cenné genetické informace.

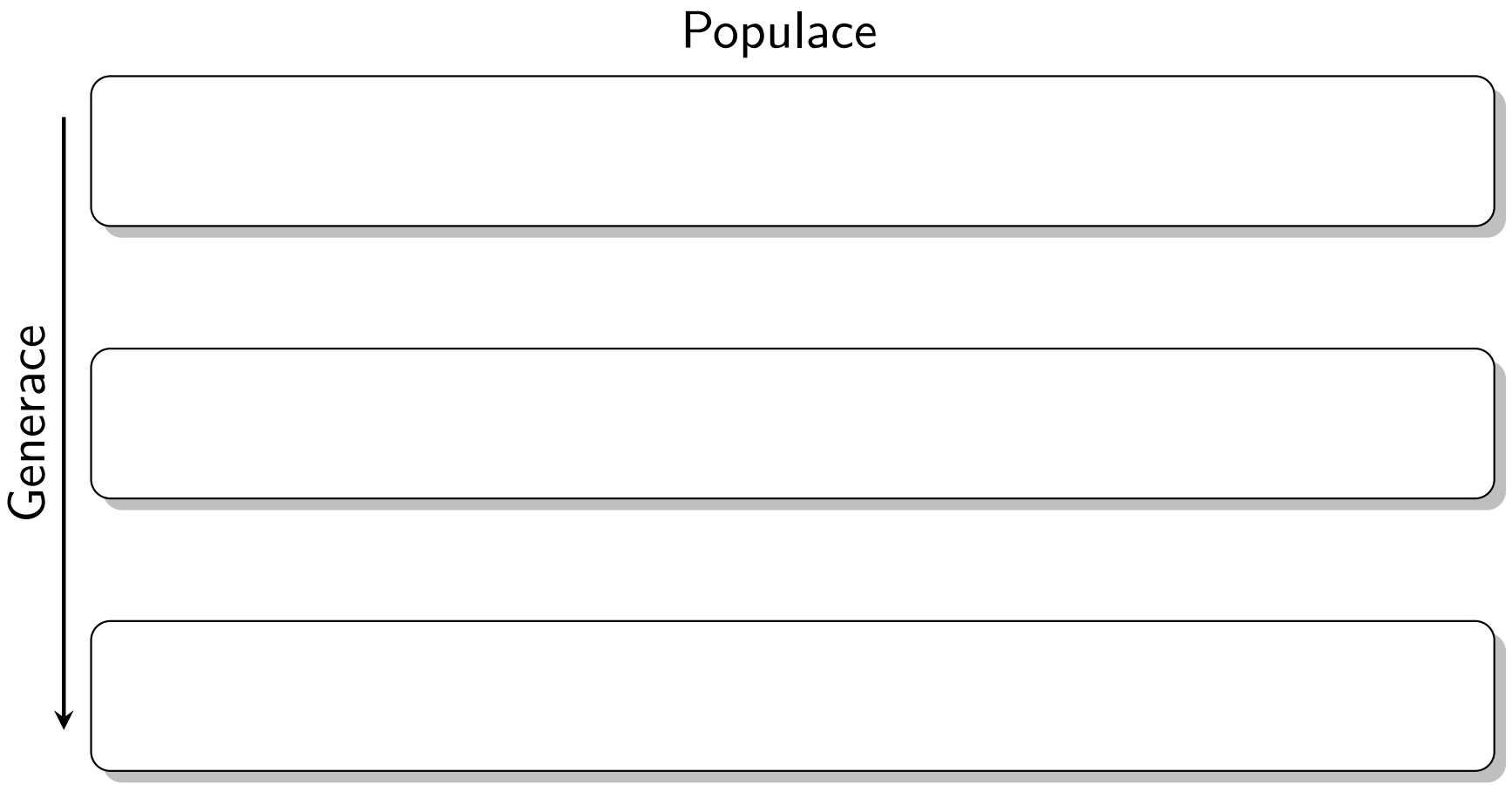
Selekce (pokrač.)

- Jen s kvalitními jedinci nám hrozí uváznutí na jednom místě, všechna řešení si budou podobná, nebudeme schopni provést větší změnu v řešení.
- Řešením se ukazuje ruletové kolo s různě velkými výsečemi. Velikost výšeče odpovídá fitness jedince.

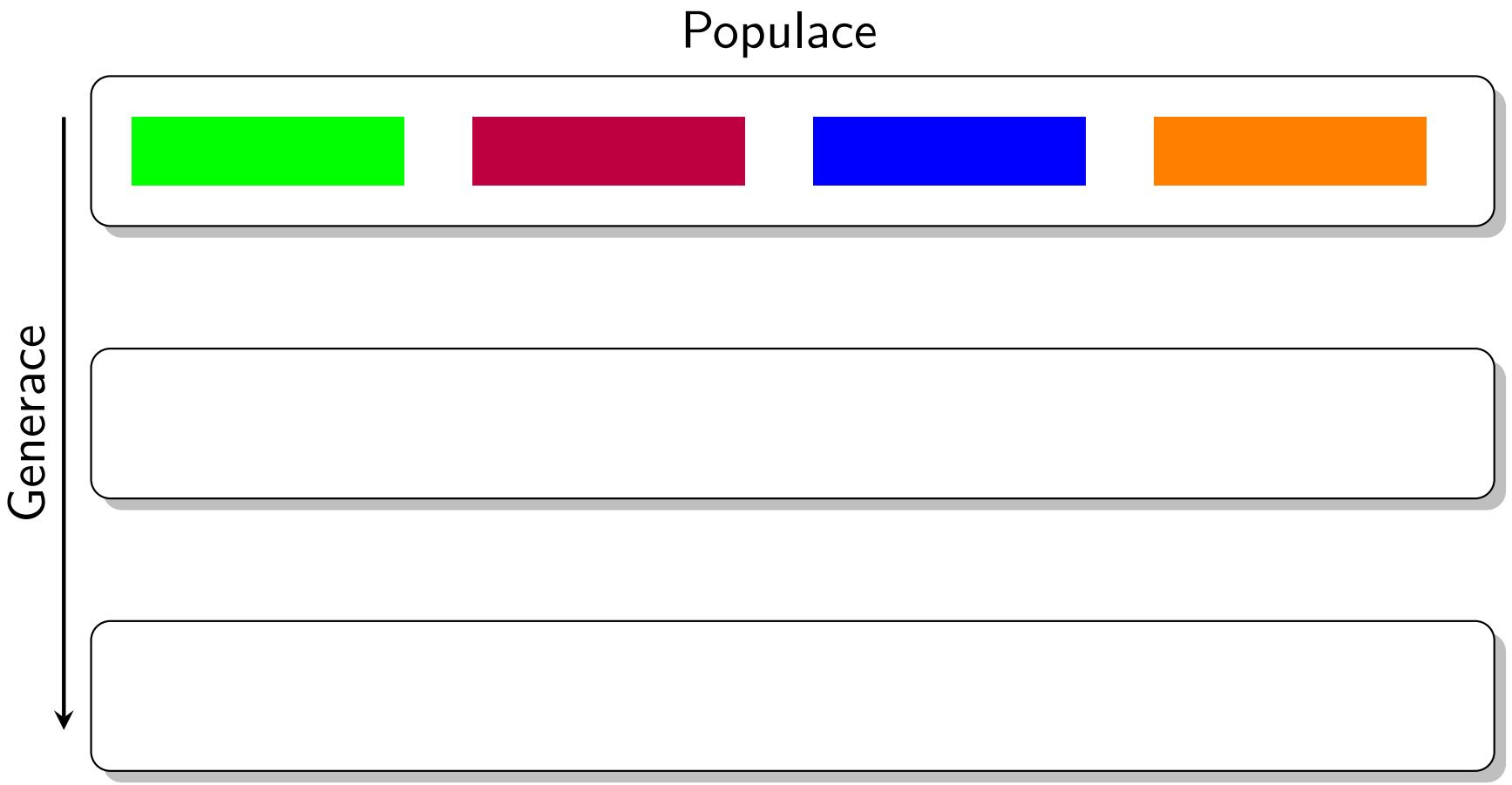
Selekce (pokrač.)



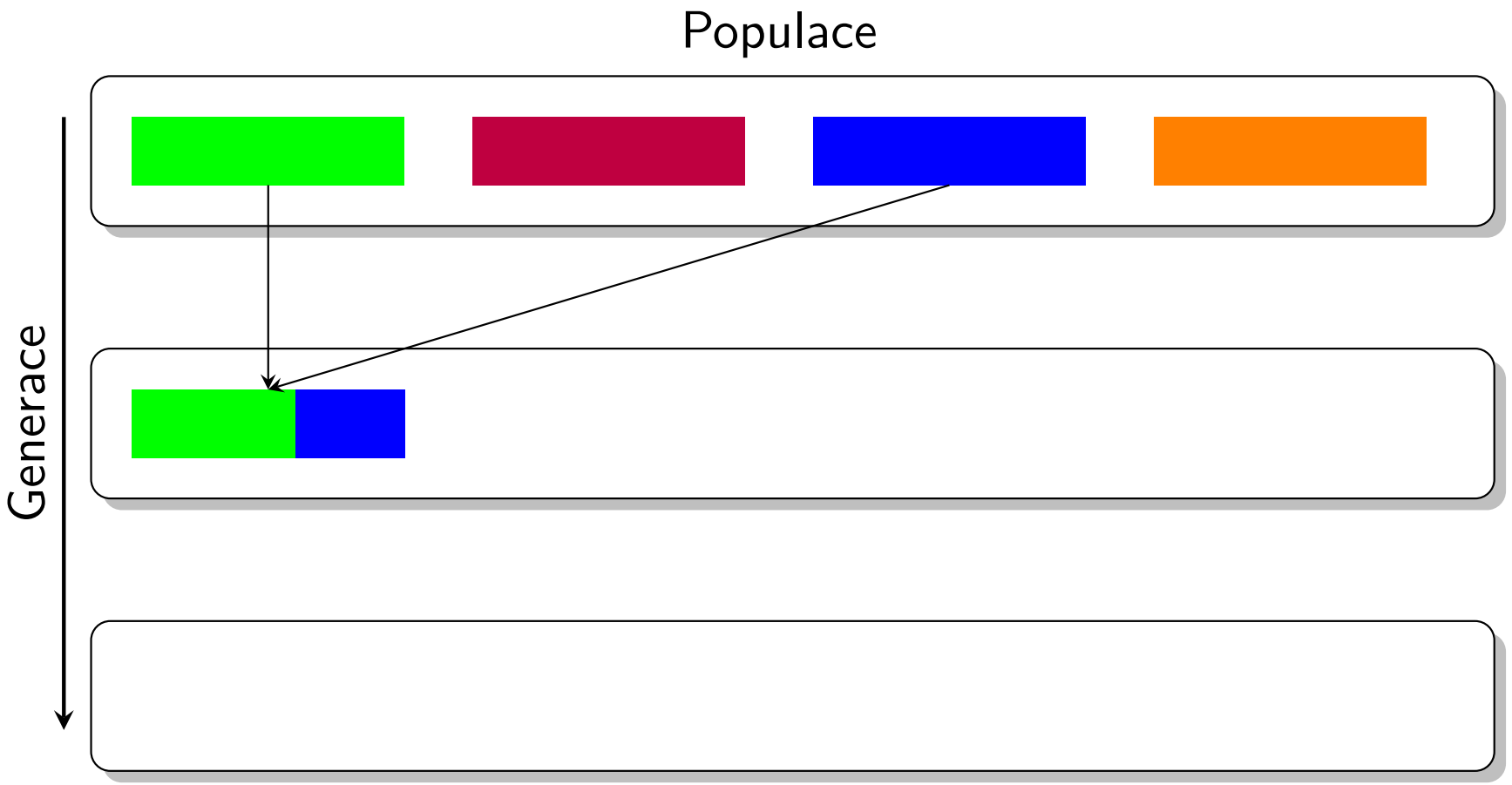
Křížení jedinců



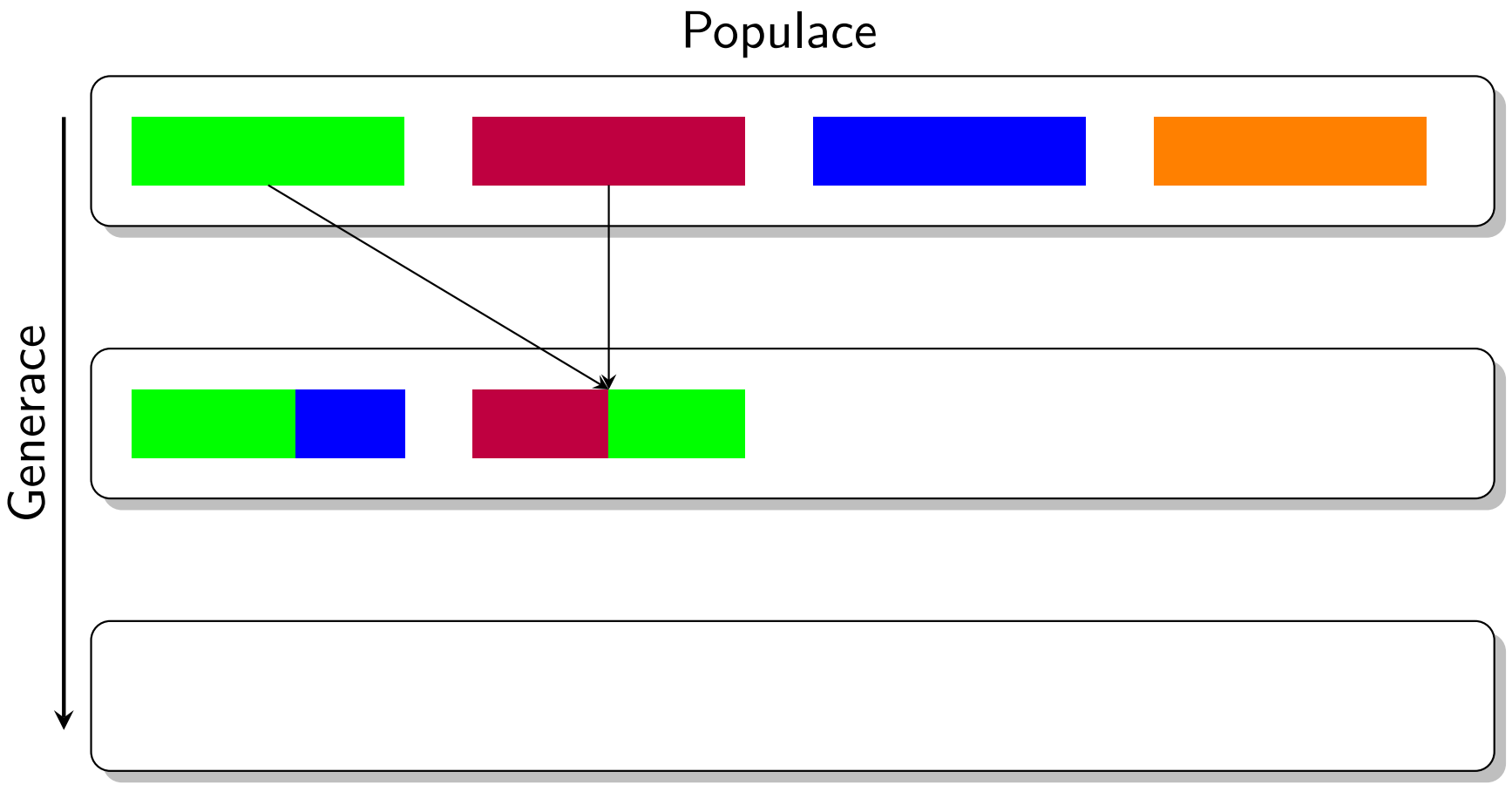
Křížení jedinců



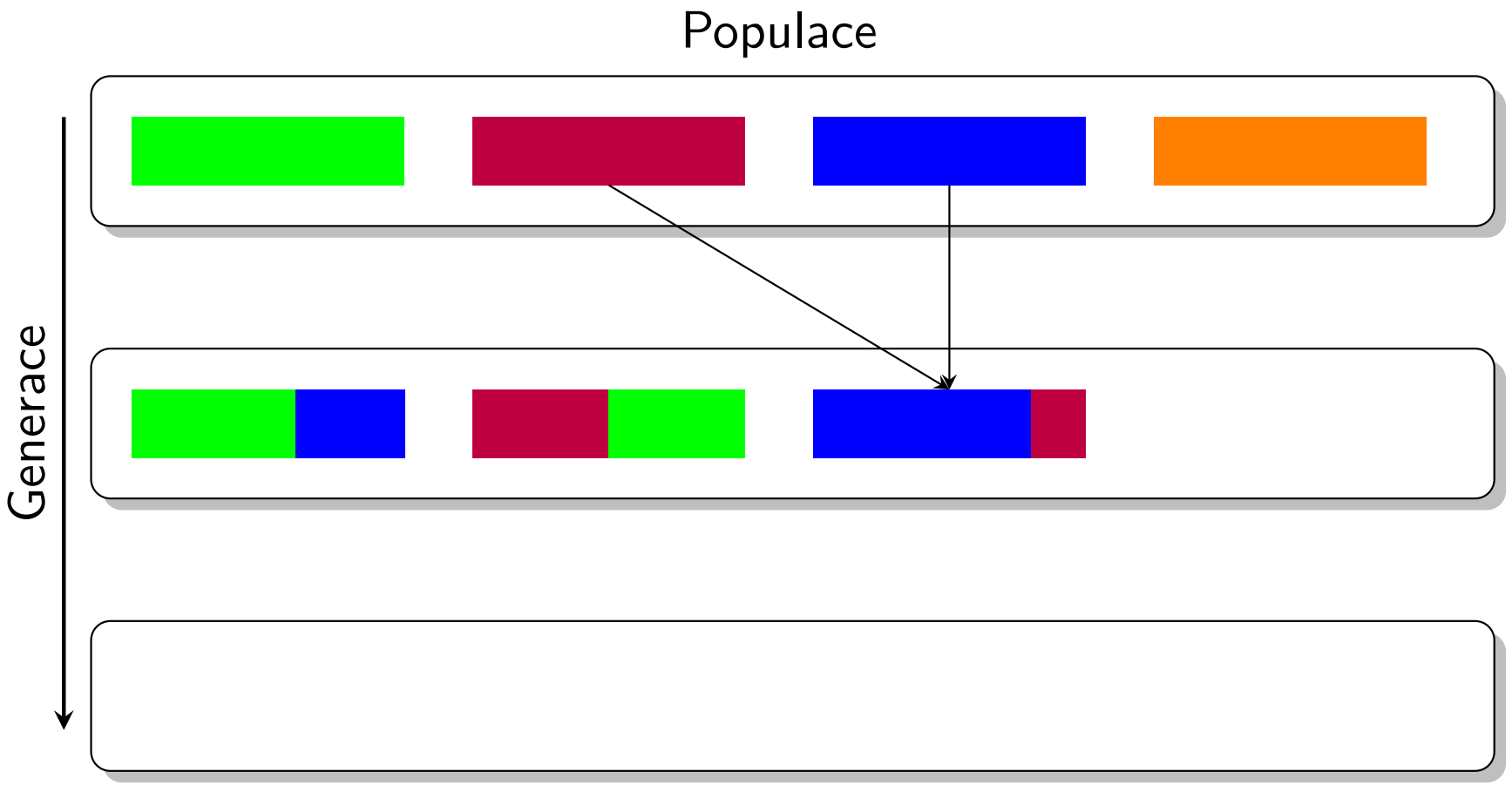
Křížení jedinců



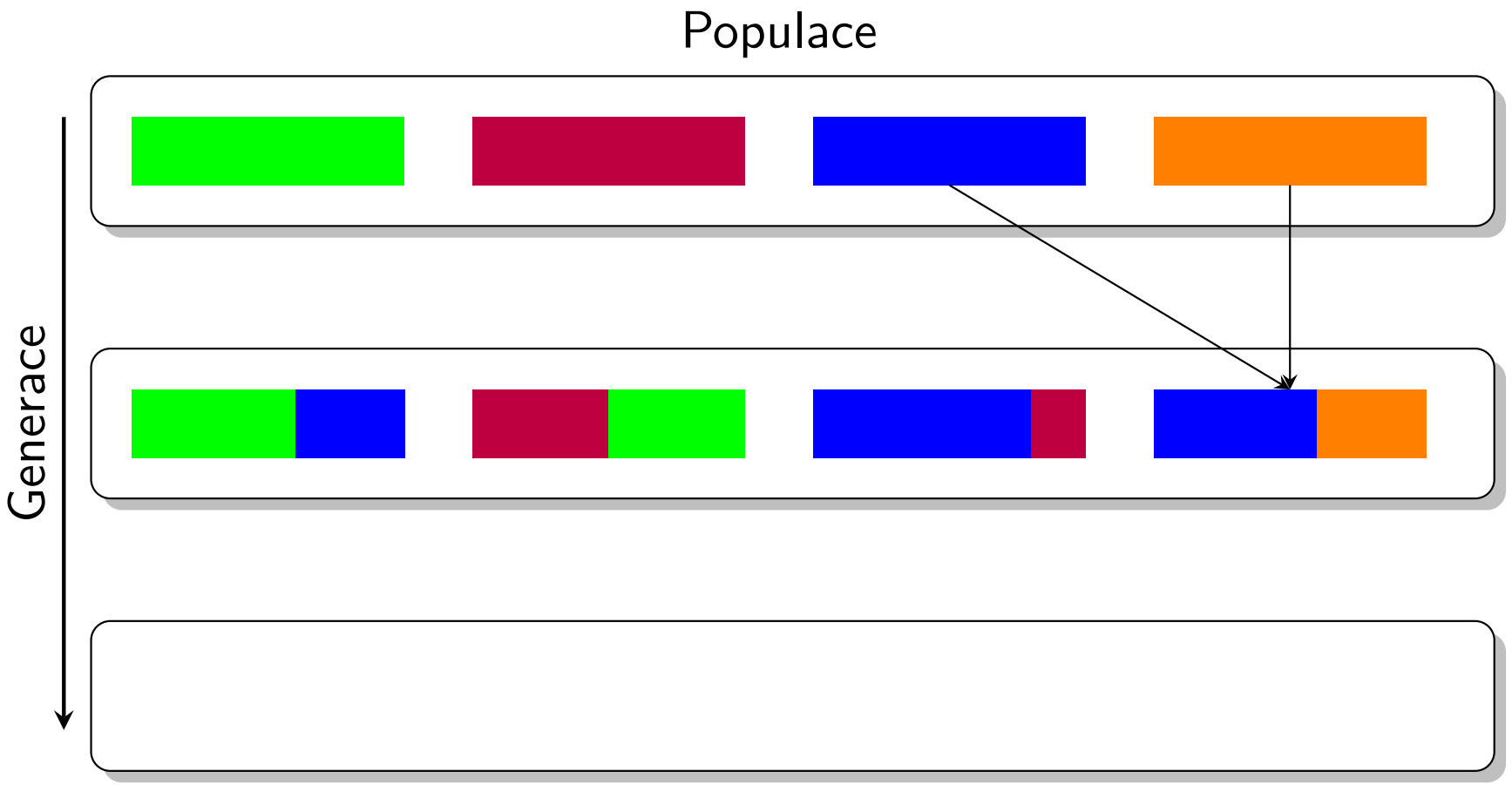
Křížení jedinců



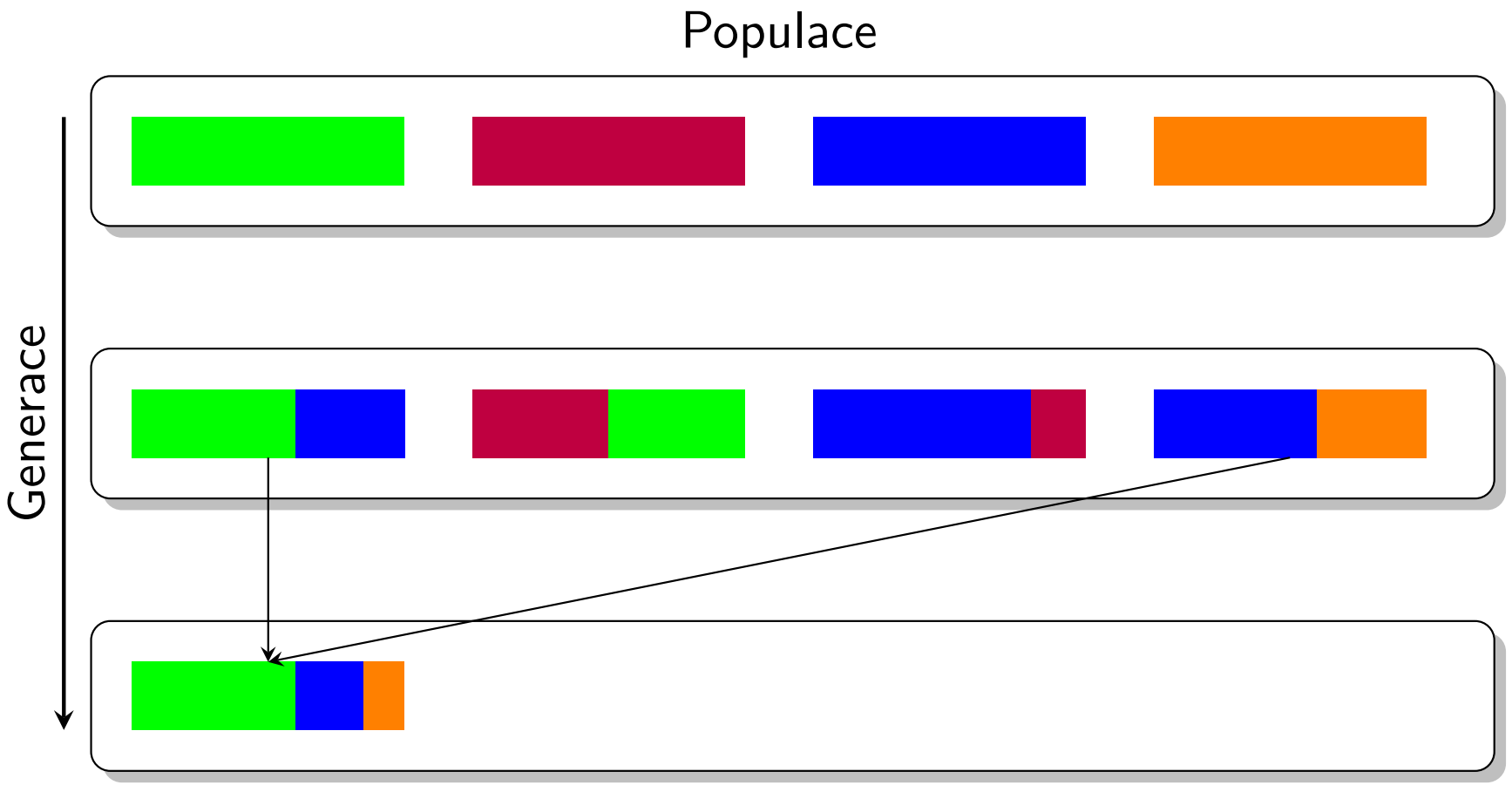
Křížení jedinců



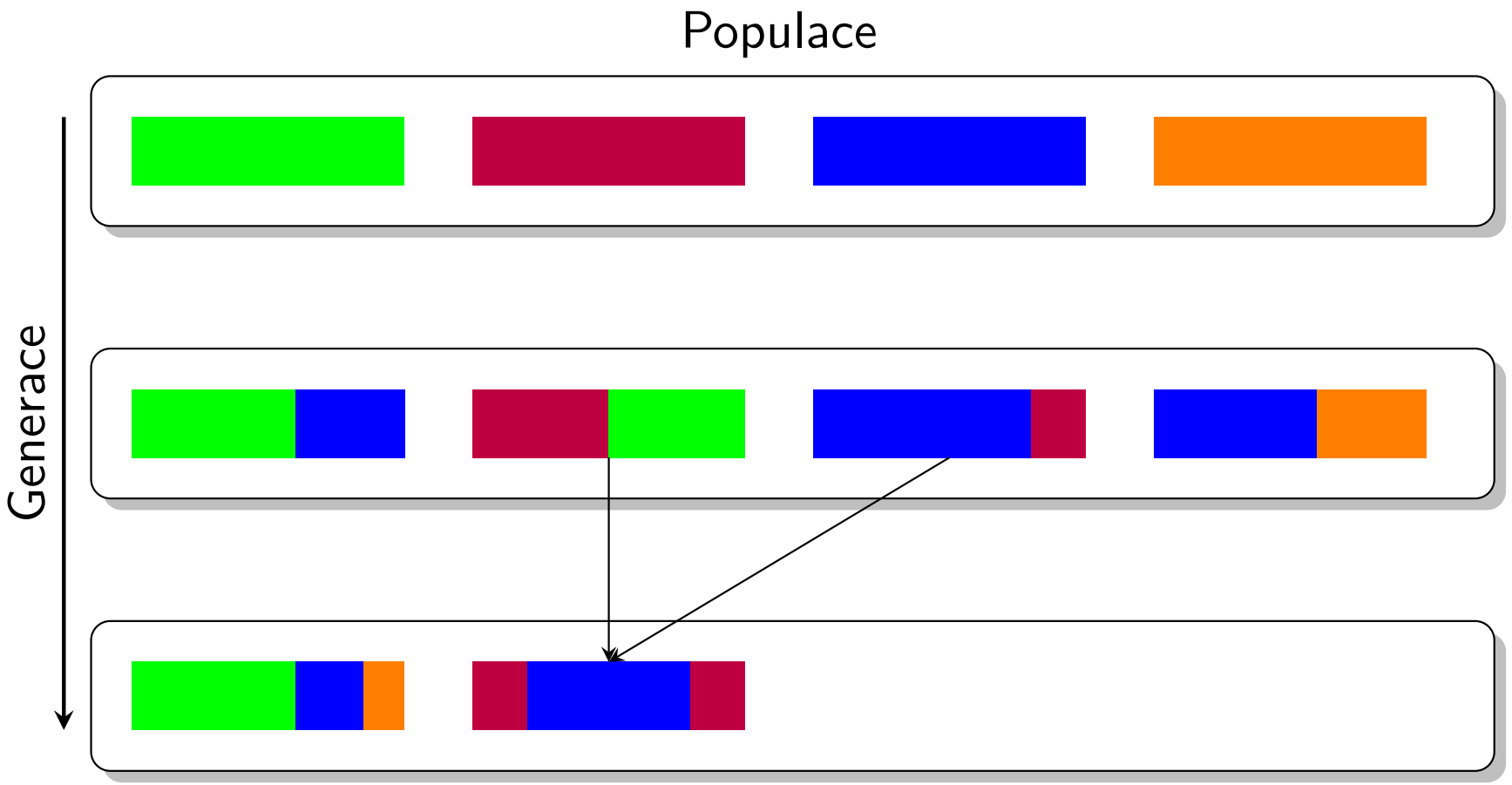
Křížení jedinců



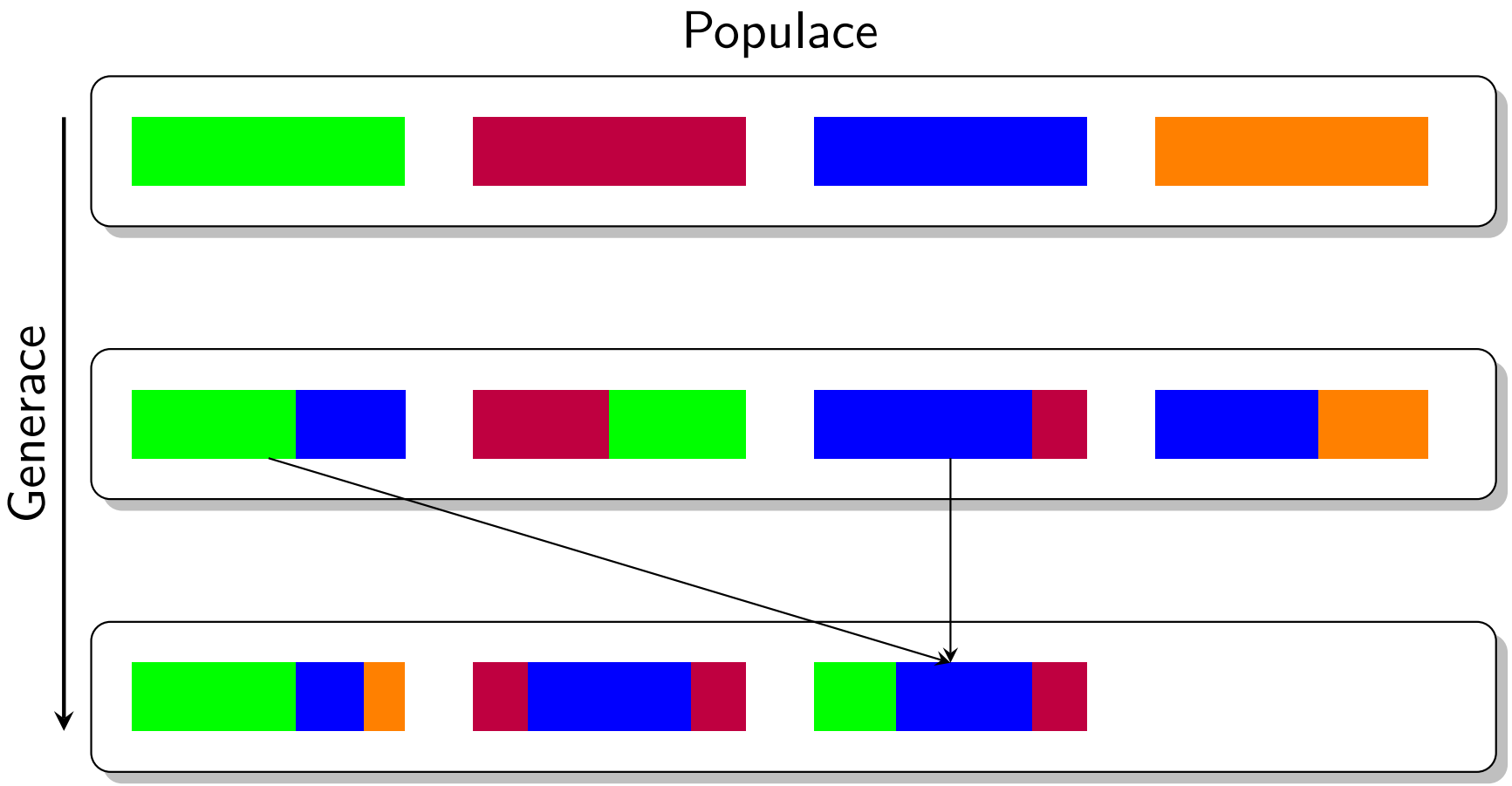
Křížení jedinců



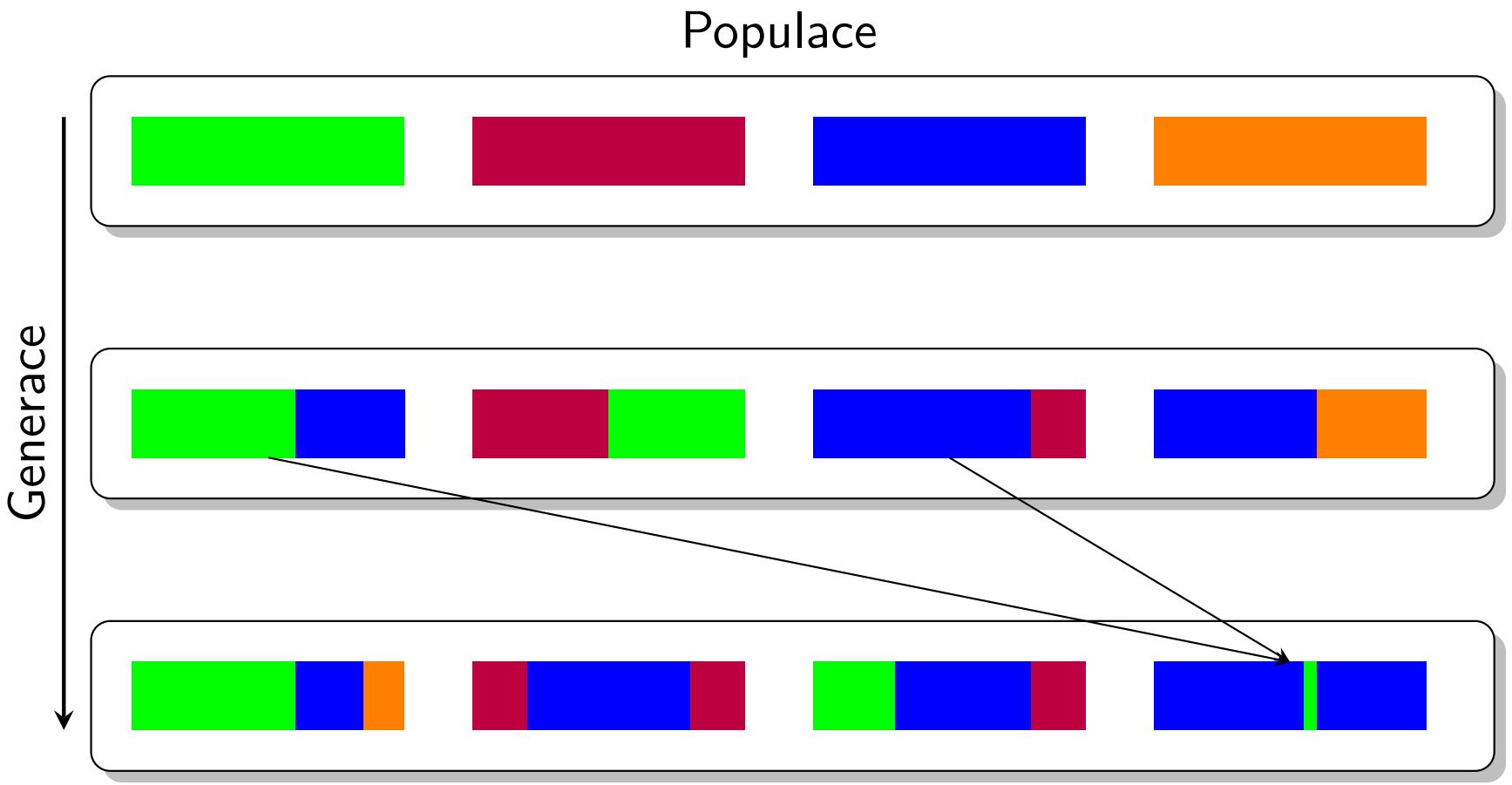
Křížení jedinců



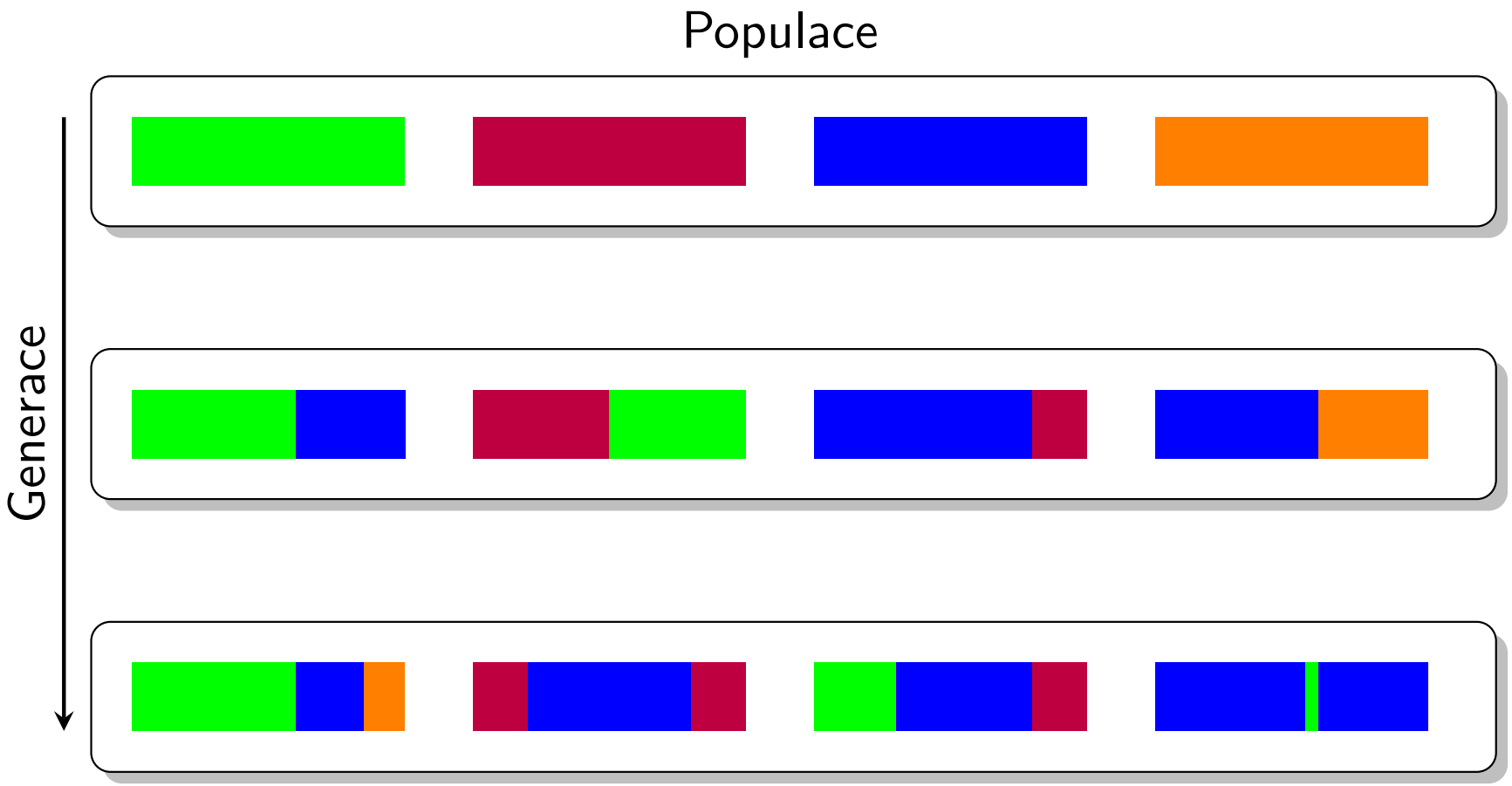
Křížení jedinců



Křížení jedinců



Křížení jedinců



Mutace jednice



Jak reprezentovat jedince a potřebné operace?

- Trasa po hospodách je dána permutací – jedinec tedy bude reprezentován touto permutací.
- Zdatnost jedince bude dána délkou této trasy. Otázka maximalizace fitness.
- Jak provést křížení permutací? A jak provést mutaci v permutaci?
- Mutace by šla řešit prohozením dvojice čísel v permutaci.
- Při přímočaré reprezentaci může dojít při křížení k opakování hodnot v permutaci.
- Řešení nabízí reprezentace pomocí tzv. náhodných klíčů.
  - Pro každý prvek v permutaci vygenerujeme náhodné číslo v rozsahu $\langle 0, 1)$.

$$\begin{pmatrix} 0,7155 & 0,2547 & 0,8695 & 0,4034 & 0,7228 & 0,7678 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

Jak reprezentovat jedince a potřebné operace? (pokrač.)

- Permutaci získáme jejich setříděním.

$$\begin{pmatrix} 0,2547 & 0,4034 & 0,7155 & 0,7228 & 0,7678 & 0,8695 \\ 1 & 3 & 0 & 4 & 5 & 2 \end{pmatrix}$$

- Křížení – spojíme dvě části posloupnosti náhodných klíčů a klíče setřídíme.
- Mutace je přímočará – nepatrně změníme jeden k klíčů a setřídíme.

Fitness jedince

- GA maximalizuje fitness.
- My chceme ale naopak co nejkratší trasu!
- Jak tedy převést mimimalizaci délky trasy na maximalizaci?
- Vhodnou formulací fitness funkce.
- Musíme použít přímo délku trasy? Nemůžeme ji nějak transformovat?
- Velice jednoduše – převrácené číslo, „jedna lomeno“.
- Čím kratší trasa bude, tím bude její převrácená hodnota větší.

Tour de pub Olomouc 2017 – analýza zadání

Nástroje pro hledání nejkratší soutěžní trasy

Musíme vymyslet následující:

- způsob, jak reprezentovat trasu, ✓
- způsob, jak určit délku libovolné trasy a ✓
- algoritmus řešení úlohy. ✓

Soubor GlobalConstants.h

```
1  #pragma once
2
3  const int TotalNumberOfPubs = 16;
4
5  const int NumberOfMutablePubs = TotalNumberOfPubs - 1;
6
7  const int StartFinishPubIndex = TotalNumberOfPubs - 1;
8
9  const int DistanceMatrix[TotalNumberOfPubs][TotalNumberOfPubs] =
10 {
11    {0, 100, 160, 320, 1400, 1600, 890, 1700, 1800, 2600, 820, 110, 480,
          900, 650, 1600},
12    { 100, 0, 80, 280, 1500, 1700, 970, 1700, 1900, 2700, 890, 190, 560,
          980, 730, 1500 },
13    { 160, 80, 0, 310, 1500, 1800, 1000, 1800, 1900, 2800, 900, 270,
        640, 1100, 810, 1500 },
14    { 320, 280, 310, 0, 1700, 1500, 760, 1500, 2000, 2700, 1100, 210,
        550, 780, 600, 1300 },
```

Soubor GlobalConstants.h (pokrač.)

```
15    { 1400, 1500, 1500, 1700, 0, 2600, 1900, 2600, 780, 1700, 580, 1500,
          1100, 1900, 1500, 3100 },
16    { 1600, 1700, 1800, 1500, 2600, 0, 750, 60, 2800, 3200, 2100, 1600,
          1600, 840, 1300, 1000 },
17    { 890, 970, 1000, 760, 1900, 750, 0, 770, 2000, 2600, 1400, 870,
          860, 150, 510, 1900 },
18    { 1700, 1700, 1800, 1500, 2600, 60, 770, 0, 2800, 3200, 2100, 1600,
          1600, 860, 1300, 1000 },
19    { 1800, 1900, 1900, 2000, 780, 2800, 2000, 2800, 0, 950, 1000, 1900,
          1400, 2100, 1700, 3500 },
20    { 2600, 2700, 2800, 2700, 1700, 3200, 2600, 3200, 950, 0, 2000,
          2600, 2200, 2600, 2300, 3900 },
21    { 820, 890, 900, 1100, 580, 2100, 1400, 2100, 1000, 2000, 0, 880,
          620, 1400, 980, 2300 },
22    { 110, 190, 270, 210, 1500, 1600, 870, 1600, 1900, 2600, 880, 0,
          460, 880, 630, 1400 },
23    { 480, 560, 640, 550, 1100, 1600, 860, 1600, 1400, 2200, 620, 460,
          0, 880, 570, 1700 },
```

Soubor GlobalConstants.h (pokrač.)

```
24    { 900, 980, 1100, 780, 1900, 840, 150, 860, 2100, 2600, 1400, 880,
          880, 0, 540, 1500 },
25    { 650, 730, 810, 600, 1500, 1300, 510, 1300, 1700, 2300, 980, 630,
          570, 540, 0, 1700 },
26    { 1600, 1500, 1500, 1300, 3100, 1000, 1900, 1000, 3500, 3900, 2300,
          1400, 1700, 1500, 1700, 0 }
27 };
28
29 const int PopulationSize = 250;
30
31 const int NumberOfGenerations = 100;
32
33 const double MutationProbability = 1.0 / NumberOfMutablePubs;
34
35 const double MutationDelta = 0.2;
```

Soubor Random.h

```
1  #pragma once
2
3  #include <cstdlib>
4
5  using namespace std;
6
7  inline double Random()
8  {
9     double dRet = rand();
10    dRet /= RAND_MAX + 1;
11    return dRet;
12 }
13
14 inline int Random(const int UpperLimit)
15 {
16    return rand() % UpperLimit;
17 }
```

Soubor Utils.h

```
1 #pragma once
2
3 inline void Exchange(int& a, int& b)
4 {
5    int aux = a;
6    a = b;
7    b = aux;
8 }
9
10 inline void Exchange(double& a, double& b)
11 {
12    double aux = a;
13    a = b;
14    b = aux;
15 }
```

Soubor Individual.h

```
1  #pragma once
2
3  #include <sstream>
4  #include <string>
5  #include "GlobalConstants.h"
6  #include "Random.h"
7  #include "Utils.h"
8
9  using namespace std;
10
11 struct Individual
12 {
13    double RandomKeys[NumberOfMutablePubs];
14    int Route[NumberOfMutablePubs];
15    int RouteLength;
16    double Fitness;
17 };
18
19
```

Soubor Individual.h (pokrač.)

```
20 Individual CreateNewIndividual();
21
22 Individual CreateOffspringIndividual(const Individual& Parent1, const
       Individual& Parent2);
23
24 void PerformCrossover(const double ParentRandomKeys1[], const double
       ParentRandomKeys2[], double OffspringRandomKeys[]);
25
26 void PerformMutation(double RandomKeys[]);
27
28 void CreateRoute(const double RandomKeys[], int Route[]);
29
30 int ComputeRouteLength(const int Route[]);
31
32 double ComputeFitness(const int RouteLength);
33
34 string IndividualToString(const Individual& Individual);
35
36 inline void Exchange(Individual& a, Individual& b)
```

Soubor Individual.h (pokrač.)

```
37 {
38     Individual aux = a;
39     a = b;
40     b = aux;
41 }
```

Soubor Individual.cpp

```
1 #include "Individual.h"
2
3 Individual CreateNewIndividual()
4 {
5    Individual I;
6    for (int i = 0; i < NumberOfMutablePubs; i++)
7    {
8       I.RandomKeys[i] = Random();
9    }
10   CreateRoute(I.RandomKeys, I.Route);
11   I.RouteLength = ComputeRouteLength(I.Route);
12   I.Fitness = ComputeFitness(I.RouteLength);
13   return I;
14 }
15
16 Individual CreateOffspringIndividual(const Individual& Parent1, const
       Individual& Parent2)
17 {
18   Individual Offspring;
```

Soubor Individual.cpp (pokrač.)

```
19    PerformCrossover(Parent1.RandomKeys, Parent2.RandomKeys, Offspring.
          RandomKeys);
20    PerformMutation(Offspring.RandomKeys);
21    CreateRoute(Offspring.RandomKeys, Offspring.Route);
22    Offspring.RouteLength = ComputeRouteLength(Offspring.Route);
23    Offspring.Fitness = ComputeFitness(Offspring.RouteLength);
24    return Offspring;
25 }
26
27 void PerformCrossover(const double ParentRandomKeys1[], const double
       ParentRandomKeys2[], double OffspringRandomKeys[])
28 {
29    int breakpoint;
30    do
31    {
32       breakpoint = Random(NumberOfMutablePubs);
33    } while (breakpoint == 0);
34    for (int i = 0; i < NumberOfMutablePubs; i++)
35    {
```

Soubor Individual.cpp (pokrač.)

```
36        OffspringRandomKeys[i] = i < breakpoint ? ParentRandomKeys1[i] :
              ParentRandomKeys2[i];
37    }
38 }
39
40 void PerformMutation(double RandomKeys[])
41 {
42    int i = Random(NumberOfMutablePubs);
43    int sgn = Random() < 0.5 ? -1 : 1;
44    RandomKeys[i] += sgn * MutationDelta * RandomKeys[i];
45 }
46
47 void CreateRoute(const double RandomKeys[], int Route[])
48 {
49    double TempKeys[NumberOfMutablePubs];
50    for (int i = 0; i < NumberOfMutablePubs; i++)
51    {
52       TempKeys[i] = RandomKeys[i];
53       Route[i] = i;
```

Soubor Individual.cpp (pokrač.)

```
54     }
55     bool exchange;
56     do
57     {
58        exchange = false;
59        for (int i = 0; i < NumberOfMutablePubs - 1; i++)
60        {
61           if (TempKeys[i] > TempKeys[i + 1])
62           {
63              Exchange(TempKeys[i], TempKeys[i + 1]);
64              Exchange(Route[i], Route[i + 1]);
65              exchange = true;
66           }
67        }
68     } while (exchange);
69 }
70
71 int ComputeRouteLength(const int Route[])
72 {
```

Soubor Individual.cpp (pokrač.)

```
73    int pub1 = Route[0];
74    int pub2 = Route[NumberOfMutablePubs - 1];
75    int length = DistanceMatrix[StartFinishPubIndex][pub1] +
          DistanceMatrix[pub2][StartFinishPubIndex];
76    for (int i = 0; i < NumberOfMutablePubs - 1; i++)
77    {
78       pub1 = Route[i];
79       pub2 = Route[i + 1];
80       length += DistanceMatrix[pub1][pub2];
81    }
82    return length;
83 }
84
85 double ComputeFitness(const int RouteLength)
86 {
87    return 1.0 / RouteLength;
88
89 }
90
```

Soubor Individual.cpp (pokrač.)

```
91 string IndividualToString(const Individual& Individual)
92 {
93    ostringstream os;
94    os << "Random keys (";
95    os << Individual.RandomKeys[0];
96    for (int i = 1; i < NumberOfMutablePubs; i++)
97    {
98       os << ", " << Individual.RandomKeys[i];
99    }
100   os << ")" << endl;
101   os << "Route (";
102   os << Individual.Route[0];
103   for (int i = 1; i < NumberOfMutablePubs; i++)
104   {
105      os << ", " << Individual.Route[i];
106   }
107   os << ")" << endl;
108   os << "RouteLength " << Individual.RouteLength << endl;
109   os << "Fitness " << Individual.Fitness << endl;
```

Soubor Individual.cpp (pokrač.)

```
110     return os.str();
111 }
```

Soubor GeneticAlgorithm.h

```
1  #pragma once
2
3  #include <ctime>
4  #include <fstream>
5  #include <iostream>
6  #include "GlobalConstants.h"
7  #include "Individual.h"
8  #include "Random.h"
9
10 using namespace std;
11
12 struct PopulationStatistics
13 {
14    int BestRoute[NumberOfMutablePubs];
15    int BestRouteLength;
16    double BestFitness;
17
18    int MeanRouteLength;
19    double MeanFitness;
```

Soubor GeneticAlgorithm.h (pokrač.)

```
20
21     int WorstRouteLength;
22     double WorstFitness;
23 };
24
25
26 void PerformGeneticAlgorithm();
27
28 void ReportPopulation(const Individual Population[], const int PopSize,
       const string& ReportFilePath);
29
30 void ComputeStatistics(const Individual Population[],
      PopulationStatistics& Stat);
31
32 void ReportStatistics(const PopulationStatistics Stats[], const string&
       ReportFilePath);
33
34 void CreateInitialPopulation(Individual Population[]);
35
```

Soubor GeneticAlgorithm.h (pokrač.)

```
36 void CreateRouletteWheel(const Individual Population[], double
       RouletteWheel[]);
37
38 void PerformSelection(const double RouletteWheel[], int& Parent1, int&
       Parent2);
39
40 void CopyPopulation(const Individual SourcePopulation[], Individual
       DestinationPopulation[]);
41
42 int TurnRouletteWheel(const double RouletteWheel[]);
43
44 void QuickSort(Individual A[], const int Left, const int Right);
```

Soubor GeneticAlgorithm.cpp

```
1  #include "GeneticAlgorithm.h"
2
3  void PerformGeneticAlgorithm()
4  {
5     Individual Population[PopulationSize];
6     PopulationStatistics PopStat[NumberOfGenerations + 1];
7     srand((unsigned int)time(NULL));
8     CreateInitialPopulation(Population);
9     QuickSort(Population, 0, PopulationSize - 1);
10    ReportPopulation(Population, PopulationSize, "InitialPopulation.txt"
          );
11    ComputeStatistics(Population, PopStat[0]);
12    for (int g = 1; g <= NumberOfGenerations; g++)
13    {
14       cout << "Generation #" << g << endl;
15       cout << "Best individual" << endl;
16       cout << IndividualToString(Population[0]);
17       cout << endl << endl;
18       double RouletteWheel[PopulationSize];
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
19        CreateRouletteWheel(Population, RouletteWheel);
20        Individual TempPopulation[2 * PopulationSize];
21        CopyPopulation(Population, TempPopulation);
22        for (int i = 0; i < PopulationSize; i++)
23        {
24           int Parent1Index, Parent2Index;
25           PerformSelection(RouletteWheel, Parent1Index, Parent2Index);
26           TempPopulation[i + PopulationSize] = CreateOffspringIndividual(
                 Population[Parent1Index], Population[Parent2Index]);
27        }
28        QuickSort(TempPopulation, 0, 2 * PopulationSize - 1);
29        CopyPopulation(TempPopulation, Population);
30        ReportPopulation(Population, PopulationSize, "NewPopulation" +
              to_string(g) + ".txt");
31        ComputeStatistics(Population, PopStat[g]);
32     }
33     cout << "Finish" << endl;
34     cout << "----------------------------------------" << endl;
35     cout << "Best individual" << endl;
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
36     cout << IndividualToString(Population[0]);
37     cout << endl;
38     ReportStatistics(PopStat, "PopulationStatistics.txt");
39 }
40
41
42 void ReportPopulation(const Individual Population[], const int PopSize,
         const string& ReportFilePath)
43 {
44     ofstream Output(ReportFilePath);
45     for (int i = 0; i < PopSize; i++)
46     {
47         Output << "Individual #" << i << endl;
48         Output << IndividualToString(Population[i]);
49         Output << endl;
50     }
51     Output.close();
52 }
53
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
54 void ComputeStatistics(const Individual Population[],
       PopulationStatistics& Stat)
55 {
56    for (int i = 0; i < NumberOfMutablePubs; i++)
57    {
58       Stat.BestRoute[i] = Population[0].Route[i];
59    }
60    Stat.BestRouteLength = Population[0].RouteLength;
61    Stat.BestFitness = Population[0].Fitness;
62    Stat.MeanRouteLength = 0;
63    Stat.MeanFitness = 0;
64    for (int i = 0; i < PopulationSize; i++)
65    {
66       Stat.MeanRouteLength += Population[i].RouteLength;
67       Stat.MeanFitness += Population[i].Fitness;
68    }
69    Stat.MeanRouteLength /= PopulationSize;
70    Stat.MeanFitness /= PopulationSize;
71    Stat.WorstRouteLength = Population[PopulationSize - 1].RouteLength;
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
72    Stat.WorstFitness = Population[PopulationSize - 1].Fitness;
73 }
74
75 void ReportStatistics(const PopulationStatistics Stats[], const string&
       ReportFilePath)
76 {
77    ofstream Output(ReportFilePath);
78    for (int i = 0; i <= NumberOfGenerations; i++)
79    {
80       Output << i << ";";
81       Output << "(";
82       Output << Stats[i].BestRoute[0];
83       for (int j = 1; j < NumberOfMutablePubs; j++)
84       {
85          Output << ", " << Stats[i].BestRoute[j];
86       }
87       Output << ");";
88       Output << Stats[i].BestRouteLength << ";";
89       Output << Stats[i].BestFitness << ";";
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
90          Output << Stats[i].MeanRouteLength << ";";
91          Output << Stats[i].MeanFitness << ";";
92          Output << Stats[i].WorstRouteLength << ";";
93          Output << Stats[i].WorstFitness << ";";
94          Output << endl;
95       }
96       Output.close();
97    }
98
99
100   void CreateInitialPopulation(Individual Population[])
101   {
102      for (int i = 0; i < PopulationSize; i++)
103      {
104         Population[i] = CreateNewIndividual();
105      }
106   }
107
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
108 void CreateRouletteWheel(const Individual Population[], double
        RouletteWheel[])
109 {
110    double sum = 0;
111    for (int i = 0; i < PopulationSize; i++)
112    {
113       sum += Population[i].Fitness;
114    }
115    for (int i = 0; i < PopulationSize; i++)
116    {
117       RouletteWheel[i] = Population[i].Fitness / sum;
118    }
119    for (int i = 1; i < PopulationSize; i++)
120    {
121       RouletteWheel[i] += RouletteWheel[i - 1];
122    }
123 }
124
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
125 void PerformSelection(const double RouletteWheel[], int& Parent1, int&
        Parent2)
126 {
127    Parent1 = TurnRouletteWheel(RouletteWheel);
128    do
129    {
130       Parent2 = TurnRouletteWheel(RouletteWheel);
131    } while (Parent1 == Parent2);
132 }
133
134 void CopyPopulation(const Individual SourcePopulation[], Individual
        DestinationPopulation[])
135 {
136    for (int i = 0; i < PopulationSize; i++)
137    {
138       DestinationPopulation[i] = SourcePopulation[i];
139    }
140 }
141
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
142 int TurnRouletteWheel(const double RouletteWheel[])
143 {
144    double r = Random();
145    for (int i = 0; i < PopulationSize; i++)
146    {
147       if (r < RouletteWheel[i])
148       {
149          return i;
150       }
151    }
152    return 0;
153 }
154
155 void QuickSort(Individual A[], const int Left, const int Right)
156 {
157    int i = Left;
158    int j = Right;
159    double pivot = A[(Left + Right) / 2].Fitness;
160    do
```

Soubor GeneticAlgorithm.cpp (pokrač.)

```
161     {
162        while (A[i].Fitness > pivot)
163           i += 1;
164        while (pivot > A[j].Fitness)
165           j -= 1;
166        if (i <= j)
167        {
168           Exchange(A[i], A[j]);
169           i += 1;
170           j -= 1;
171        }
172     } while (i <= j);
173     if (Left < j)
174        QuickSort(A, Left, j);
175     if (i < Right)
176        QuickSort(A, i, Right);
177  }
```

Soubor main.cpp

```
1 #include "GeneticAlgorithm.h"
2
3 void main()
4 {
5    PerformGeneticAlgorithm();
6 }
```

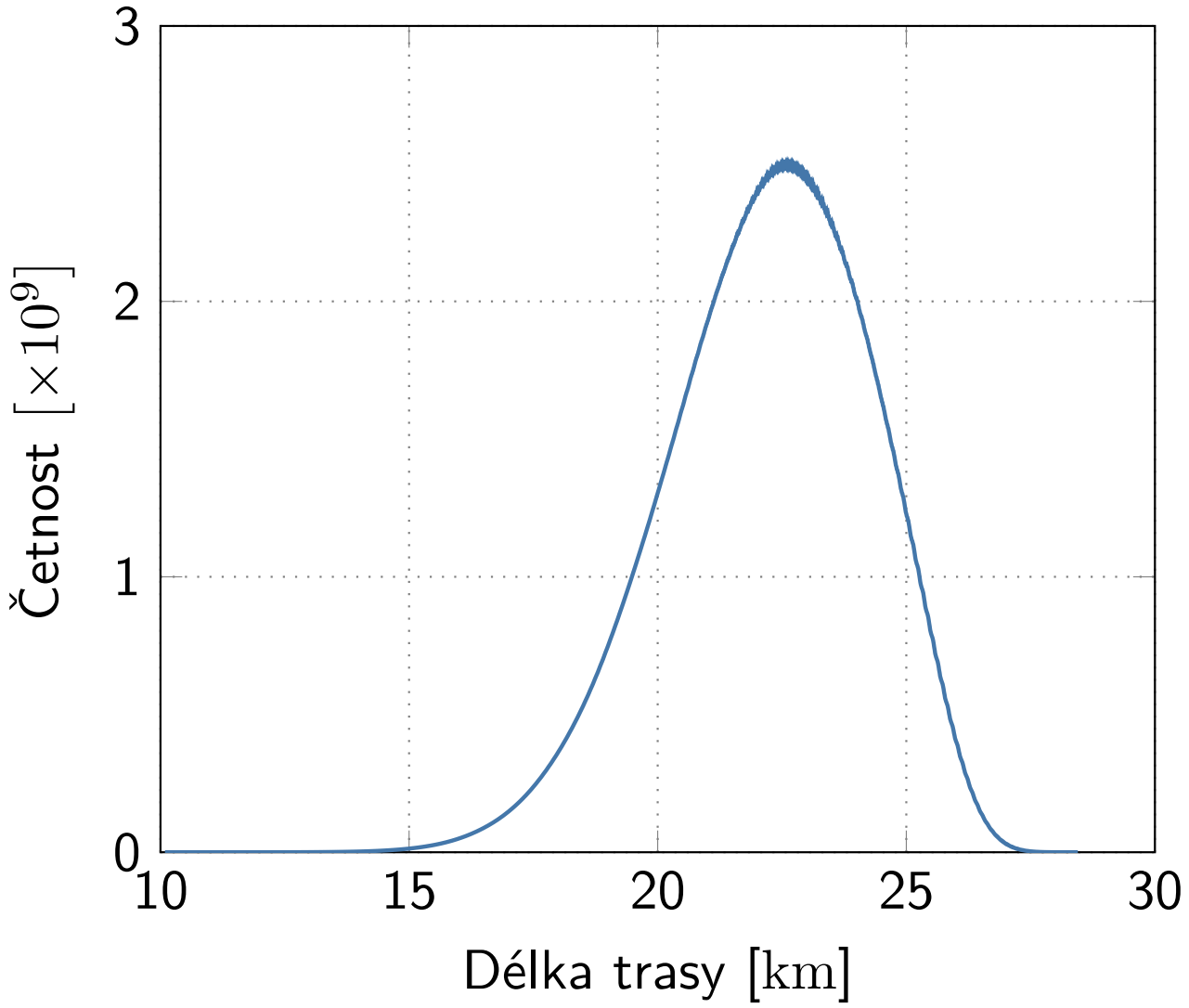
Výsledky – statistické údaje

- Celkový počet možných tras: $15! = 1,30767 \times 10^{12}$ tj. $\approx 1,31$ bilionu.
- Celkem existuje 1833 navzájem různých, tj. unikátních, délek tras.
- Nejkratší trasa měří 10090 m, nejdelší 28450 m – rozdíl činí 18360 m.
- Průměrná délka trasy je cca 22194 m.
- Medián délek tras má hodnotu 22335 m.
- Modus délek tras má hodnotu 2 499 847 424 pro délku trasy 22600 m. Modus označuje nejčastější hodnotu v datech.

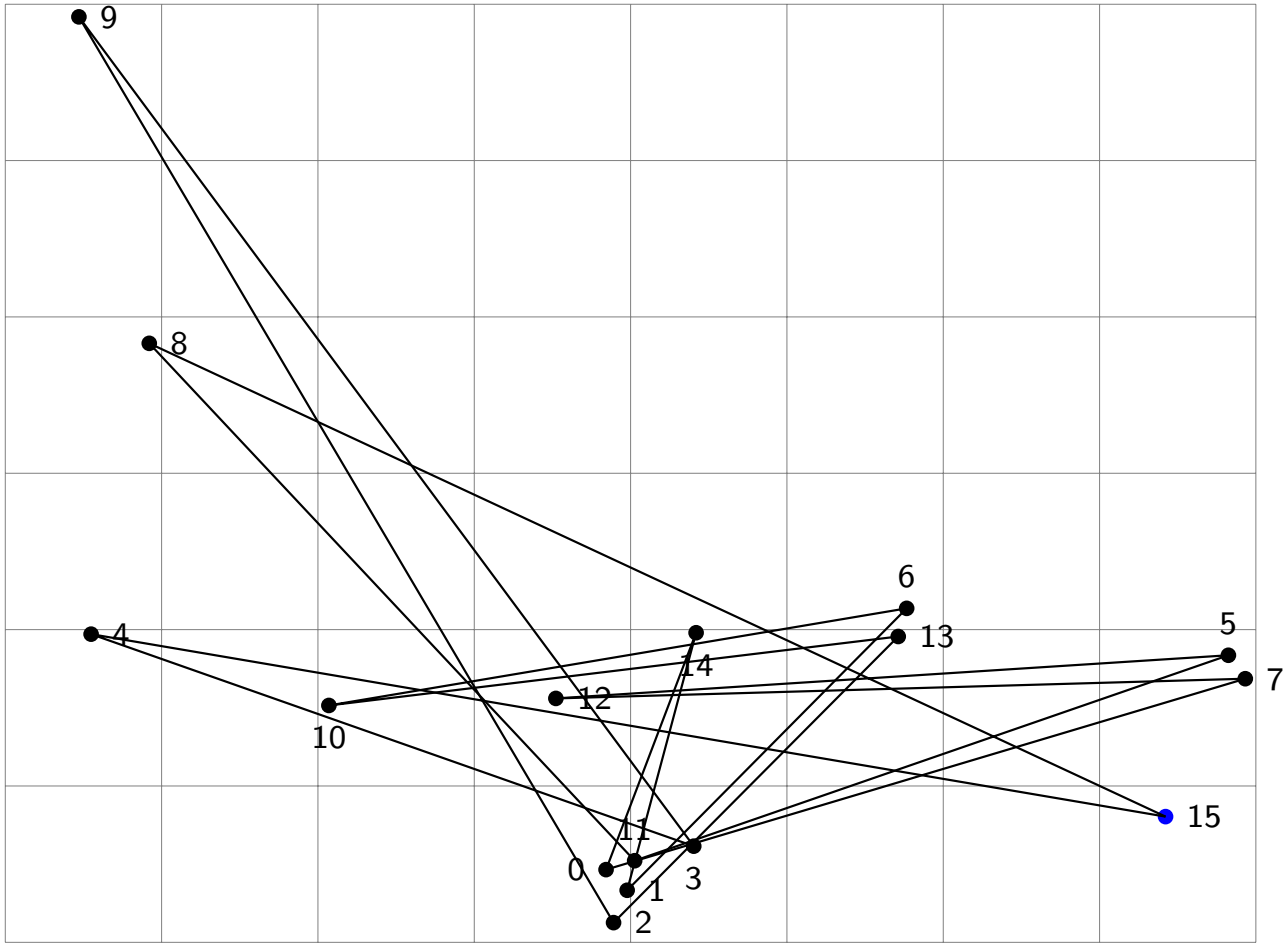
Tabulka četností

Délka trasy [m]	Četnost
10090	2
10110	2
10120	2
10140	2
⋮	⋮
22590	2 493 603 794
22600	2 499 847 424
22610	2 492 130 770
⋮	⋮
28410	132
28420	70
28430	68
28440	14
28450	10

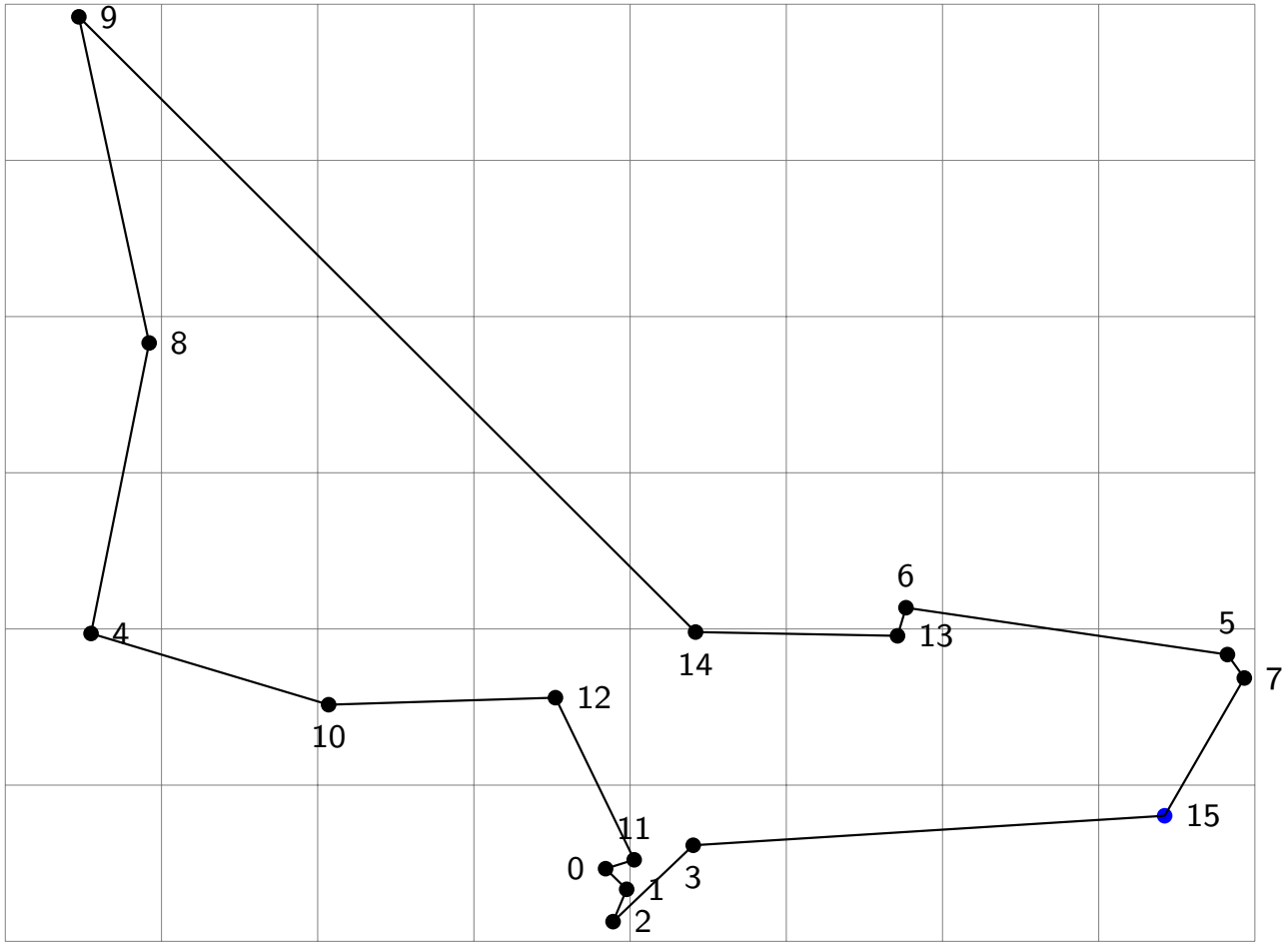
Histogram délek tras



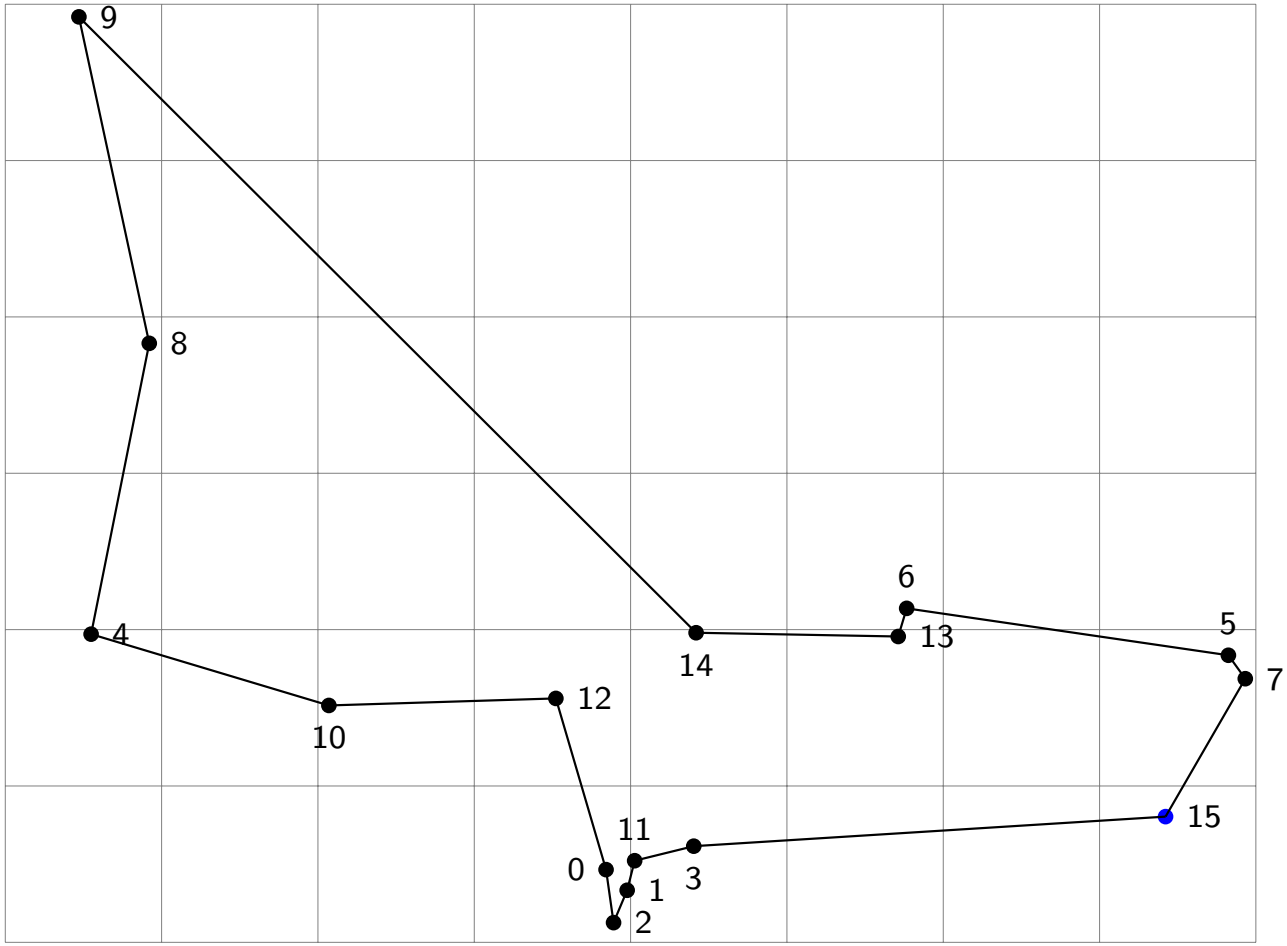
Nejhorší trasa, 28450m



Nejlepší trasa, 10090m

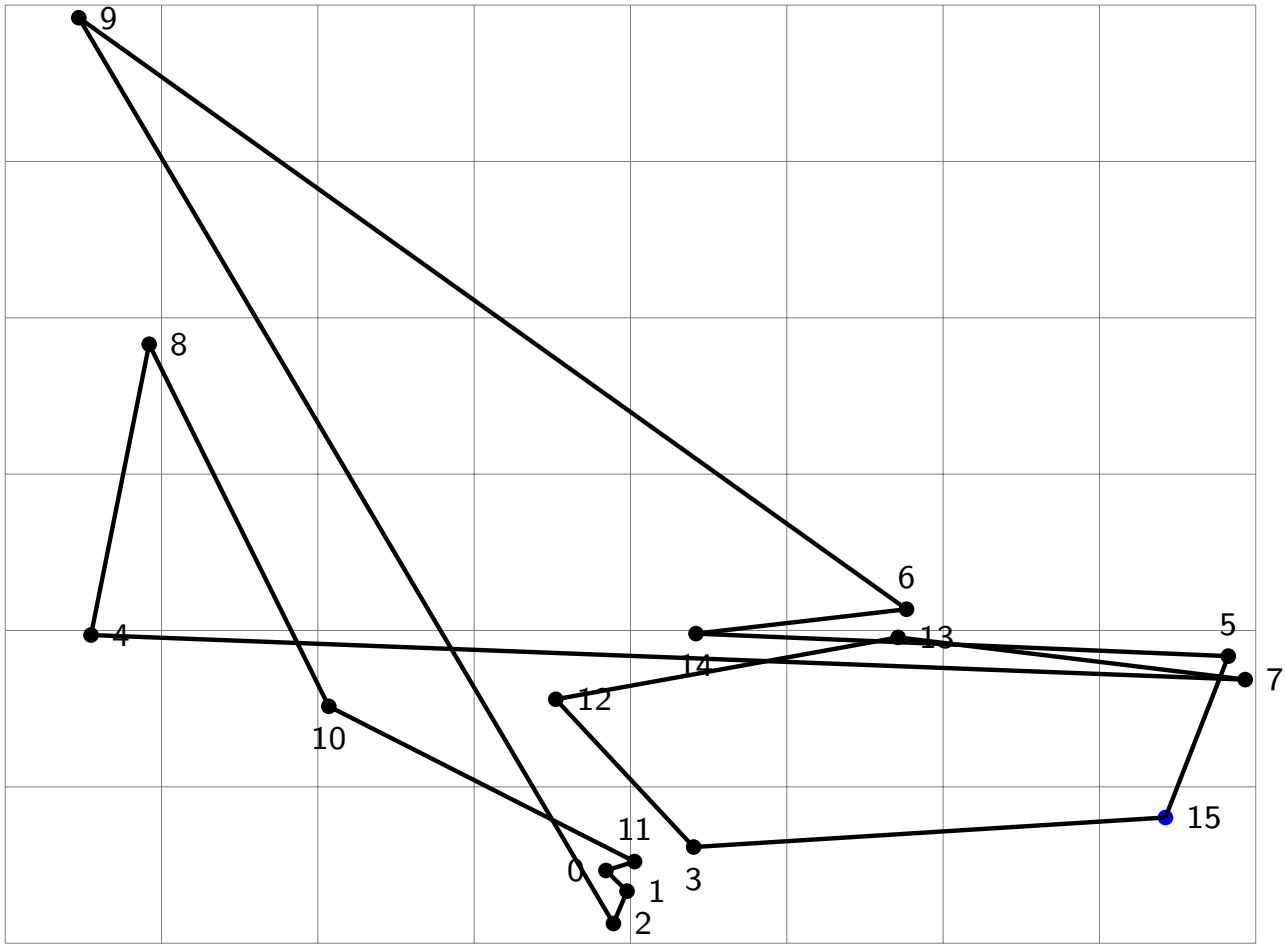


Nejlepší trasa nalezená pomocí GA, 10150m

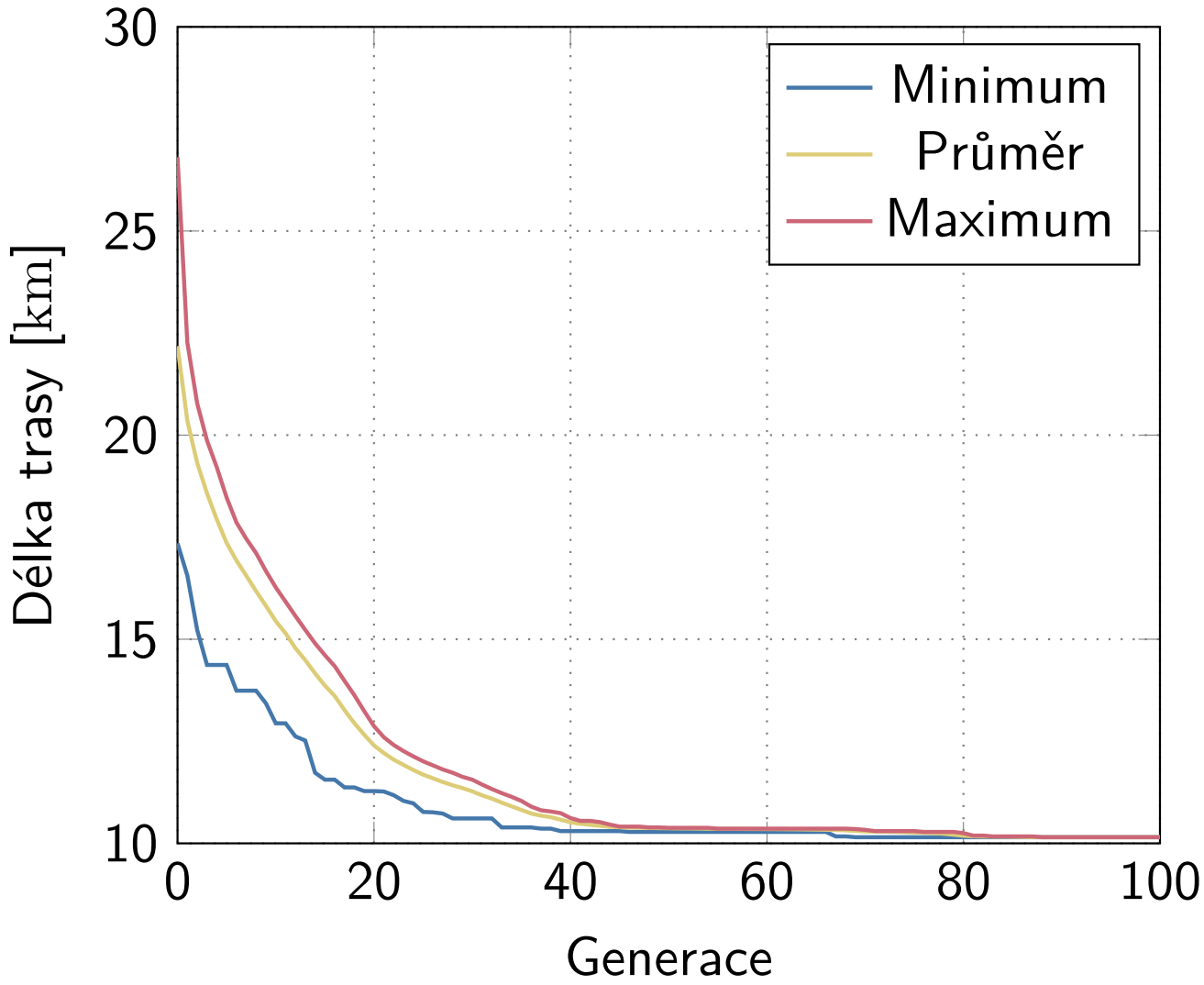


Animace hledání řešení pomocí GA

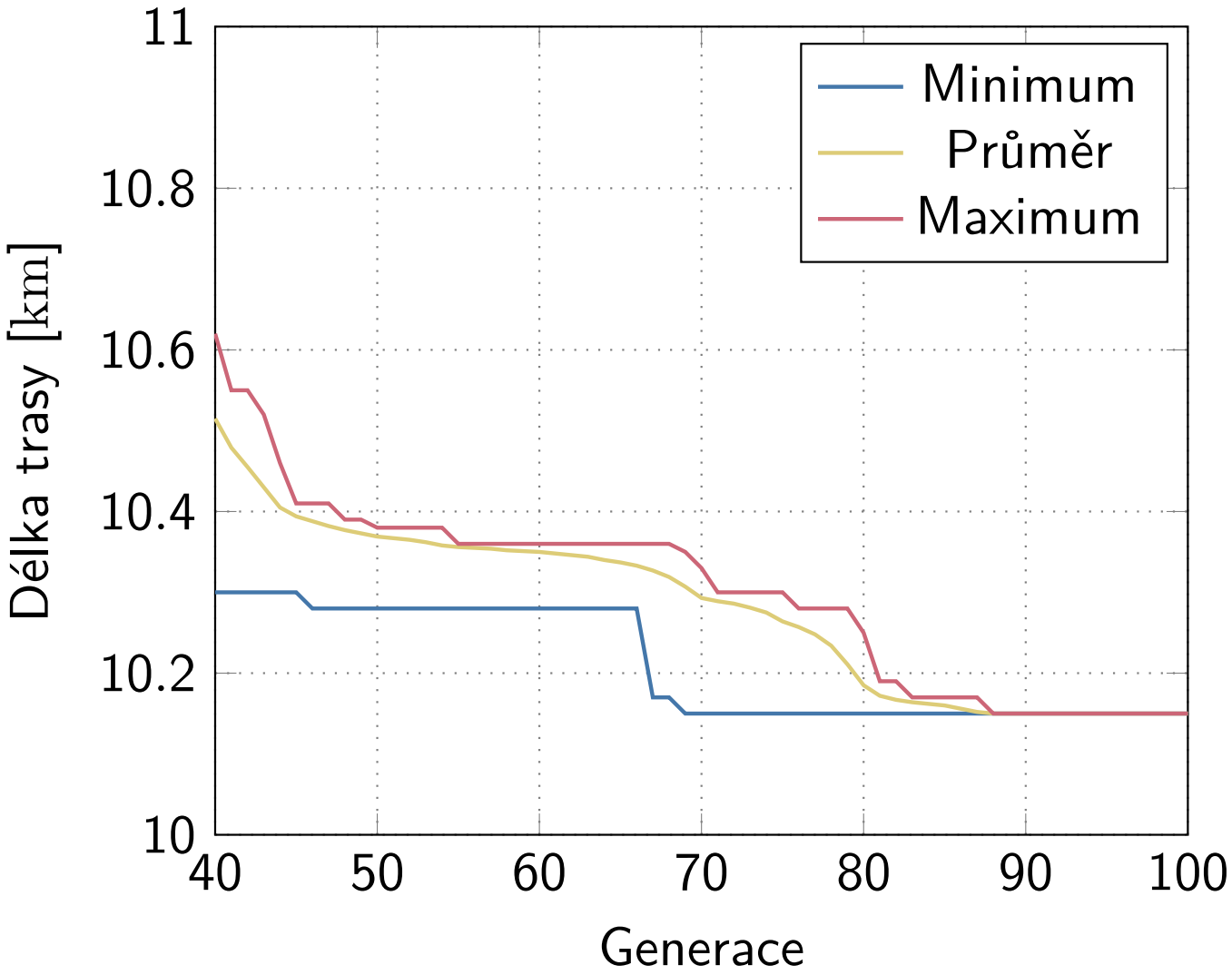
Generace: 0 **Délka trasy**: 17350 m



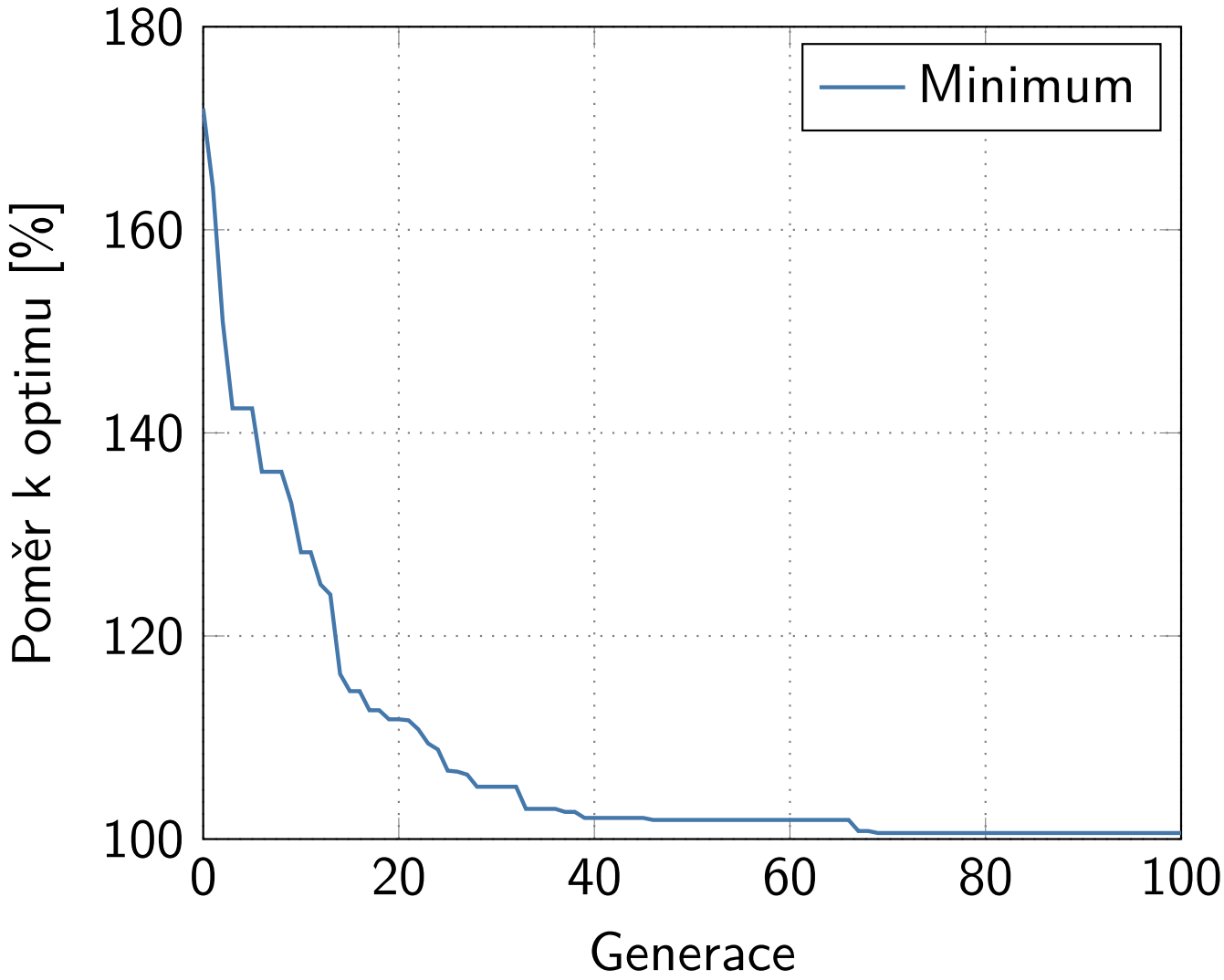
Evoluce řešení v GA



Evoluce řešení v GA – detail



Evoluce řešení v GA



Děkuji za pozornost