# Minimális költségű folyam-algoritmusok összehasonlítása

Király Zoltán[1], Kovács Péter[2]

[1] ELTE TTK Számítógéptudományi Tanszék, ELTE CNL
kiraly@cs.elte.hu

[2] ELTE IK Algoritmusok és Alkalmazásaik Tanszék, ELTE CNL
kpeter@inf.elte.hu

XXVIII. Magyar Operációkutatási Konferencia
Balatonőszöd, 2009. június 8-10.

# Áttekintés


1. A minimális költségű folyam modell
   - Definíció
   - Alkalmazások
2. Célok
3. Implementáció és tesztelés
   - LEMON
   - Algoritmusok
   - Tesztadatok, tesztelés paraméterei
4. Megoldási módszerek
   - Negatív körök kiiktatása
   - Duál algoritmusok
   - Költségskálázó algoritmusok
   - Hálózati szimplex algoritmus
5. Összehasonlítás
6. Összefoglalás

# A minimális költségű folyam modell

A vizsgált probléma:

- Szállítsunk megadott mennyiségű terméket egy hálózat termelő csúcsaiból a fogyasztó csúcsokba.
- A folyam („*szállítási terv*”) összköltsége legyen minimális.

# A minimális költségű folyam modell

Minimális költségű folyam (*minimum cost flow*) feladat:

- Adott egy $G = (V, E)$ irányított gráf.
- Minden $(i, j) \in E$ élhez hozzárendelünk
  - egy $l_{ij} \geq 0$ alsó korlátot,
  - egy $u_{ij} \geq l_{ij}$ felső korlátot és
  - egy $c_{ij}$ költséget (egy egységnyi folyam szállításának díja).

# A minimális költségű folyam modell

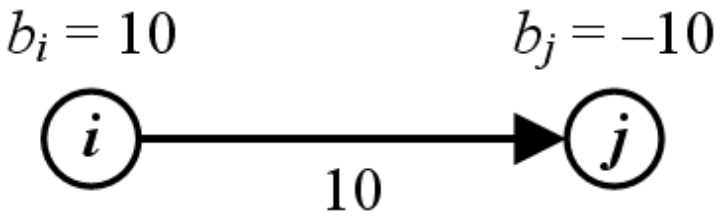
Minimális költségű folyam (*minimum cost flow*) feladat:

- Adott egy $G = (V, E)$ irányított gráf.
- Minden $(i, j) \in E$ élhez hozzárendelünk
  - egy $l_{ij} \geq 0$ alsó korlátot,
  - egy $u_{ij} \geq l_{ij}$ felső korlátot és
  - egy $c_{ij}$ költséget (egy egységnyi folyam szállításának díja).
- Minden $i \in V$ csúcshoz hozzárendelünk egy $b_i$ előjeles termelés/fogyasztás értéket.

# A minimális költségű folyam modell

Minimális költségű folyam (*minimum cost flow*) feladat:

- Adott egy $G = (V, E)$ irányított gráf.
- Minden $(i, j) \in E$ élhez hozzárendelünk
  - egy $l_{ij} \geq 0$ alsó korlátot,
  - egy $u_{ij} \geq l_{ij}$ felső korlátot és
  - egy $c_{ij}$ költséget (egy egységnyi folyam szállításának díja).
- Minden $i \in V$ csúcshoz hozzárendelünk egy $b_i$ előjeles termelés/fogyasztás értéket.




  - Ha $b_i > 0$, akkor $i$ termelő $b_i$ termeléssel.
  - Ha $b_j < 0$, akkor $j$ fogyasztó $-b_j$ fogyasztással.

# A minimális költségű folyam modell

Minimális költségű folyam (*minimum cost flow*) feladat:

- Adott egy $G = (V, E)$ irányított gráf.
- Minden $(i, j) \in E$ élhez hozzárendelünk
  - egy $l_{ij} \geq 0$ alsó korlátot,
  - egy $u_{ij} \geq l_{ij}$ felső korlátot és
  - egy $c_{ij}$ költséget (egy egységnyi folyam szállításának díja).
- Minden $i \in V$ csúcshoz hozzárendelünk egy $b_i$ előjeles termelés/fogyasztás értéket.
- A folyam költségét lineárisan számítjuk.

# A minimális költségű folyam modell

Minimális költségű folyam (*minimum cost flow*) feladat:

- Adott egy $G = (V, E)$ irányított gráf.
- Minden $(i, j) \in E$ élhez hozzárendelünk
  - egy $l_{ij} \geq 0$ alsó korlátot,
  - egy $u_{ij} \geq l_{ij}$ felső korlátot és
  - egy $c_{ij}$ költséget (egy egységnyi folyam szállításának díja).
- Minden $i \in V$ csúcshoz hozzárendelünk egy $b_i$ előjeles termelés/fogyasztás értéket.
- A folyam költségét lineárisan számítjuk.
- **Feladat:** minimális költségű megengedett folyam keresése.

Forrás: R. K. Ahuja – T. L. Magnanti – J. B. Orlin: *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

# A minimális költségű folyam modell

A feladat formálisan (LP):

## Minimális költségű folyam feladat

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\sum_{j\,:\,(i,j)\in E} x_{ij} - \sum_{j\,:\,(j,i)\in E} x_{ji} = b_i \qquad \forall i \in V \tag{2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in E \tag{3}$$

# A minimális költségű folyam modell

A feladat formálisan (LP):

## Minimális költségű folyam feladat

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\sum_{j\,:\,(i,j)\in E} x_{ij} - \sum_{j\,:\,(j,i)\in E} x_{ji} = b_i \qquad \forall i \in V \tag{2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in E \tag{3}$$

*Megjegyzés:* megengedett megoldás létezéséhez $\sum_{i\in V} b_i = 0$ szükséges feltétel.

# A minimális költségű folyam modell

A feladat formálisan (LP):

## Minimális költségű folyam feladat

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\sum_{j\,:\,(i,j)\in E} x_{ij} - \sum_{j\,:\,(j,i)\in E} x_{ji} = b_i \qquad \forall i \in V \tag{2}$$

$$l_{ij} \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in E \tag{3}$$

*Megjegyzés:* megengedett megoldás létezéséhez $\sum_{i\in V} b_i = 0$ szükséges feltétel.

Általában feltesszük, hogy minden mennyiség- és költségérték egész, és a megoldást is egészértékű folyamként keressük.

# A minimális költségű folyam modell

Alkalmazások:

- Általános, sok helyen alkalmazható modell.

# A minimális költségű folyam modell

Alkalmazások:

- Általános, sok helyen alkalmazható modell.
- Közvetlenül felhasználható számos területen:
  - szállítmányozás,
  - logisztika,
  - telekommunikáció,
  - hálózattervezés,
  - erőforrás-elosztás,
  - ütemezés
  - stb.

# A minimális költségű folyam modell

Alkalmazások:

- Általános, sok helyen alkalmazható modell.
- Közvetlenül felhasználható számos területen:
  - szállítmányozás,
  - logisztika,
  - telekommunikáció,
  - hálózattervezés,
  - erőforrás-elosztás,
  - ütemezés
  - stb.
- Gyakran előfordul bonyolultabb optimalizációs problémák részfeladataként is, pl. többtermékes folyam feladatok.

# Célok

Kutatásunk célkitűzései:

- Számos ismert algoritmus hatékony megvalósítása.
- Implementációs részletek, heurisztikák vizsgálata.

# Célok

Kutatásunk célkitűzései:

- Számos ismert algoritmus hatékony megvalósítása.
- Implementációs részletek, heurisztikák vizsgálata.
- A különböző algoritmusok összehasonlító elemzése (azonos keretrendszerben, nagy hálózatokon).

# Célok

Kutatásunk célkitűzései:

- Számos ismert algoritmus hatékony megvalósítása.
- Implementációs részletek, heurisztikák vizsgálata.
- A különböző algoritmusok összehasonlító elemzése (azonos keretrendszerben, nagy hálózatokon).
- Implementációink összehasonlítása hatékony publikus megoldóprogramokkal.

# Célok

Kutatásunk célkitűzései:

- Számos ismert algoritmus hatékony megvalósítása.
- Implementációs részletek, heurisztikák vizsgálata.
- A különböző algoritmusok összehasonlító elemzése (azonos keretrendszerben, nagy hálózatokon).
- Implementációink összehasonlítása hatékony publikus megoldóprogramokkal.
- Nyílt forrású implementációk közzététele a **LEMON** programkönyvtár részeként.

# LEMON

A megvalósított algoritmusok a **LEMON** kombinatorikus optimalizálási programcsomag részét képezik.

# LEMON

A megvalósított algoritmusok a **LEMON** kombinatorikus optimalizálási programcsomag részét képezik.

**LEMON** programkönyvtár:

- **L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks

`http://lemon.cs.elte.hu`

# LEMON

A megvalósított algoritmusok a **LEMON** kombinatorikus optimalizálási programcsomag részét képezik.

**LEMON** programkönyvtár:

- **L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks
- Az ELTE-n fejlesztett nyílt forrású, generikus C++ könyvtár.

`http://lemon.cs.elte.hu`

# LEMON

A megvalósított algoritmusok a **LEMON** kombinatorikus optimalizálási programcsomag részét képezik.

**LEMON** programkönyvtár:

- **L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks
- Az ELTE-n fejlesztett nyílt forrású, generikus C++ könyvtár.
- Gráfokkal, hálózatokkal kapcsolatos optimalizálási problémák megoldására szolgál.
- Rendkívül hatékony és rugalmasan felhasználható adatszerkezeteket, algoritmusokat tartalmaz.

`http://lemon.cs.elte.hu`

# LEMON

A megvalósított algoritmusok a **LEMON** kombinatorikus optimalizálási programcsomag részét képezik.

**LEMON** programkönyvtár:

- **L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks
- Az ELTE-n fejlesztett nyílt forrású, generikus C++ könyvtár.
- Gráfokkal, hálózatokkal kapcsolatos optimalizálási problémák megoldására szolgál.
- Rendkívül hatékony és rugalmasan felhasználható adatszerkezeteket, algoritmusokat tartalmaz.
- Hasonló könyvtárak: BGL (Boost Graph Library), LEDA.

`http://lemon.cs.elte.hu`

# Algoritmusok

Összesen 9 algoritmust valósítottunk meg, amelyek 4 különböző megoldási módszeren alapulnak.

# Algoritmusok

Összesen 9 algoritmust valósítottunk meg, amelyek 4 különböző megoldási módszeren alapulnak.

1. Negatív körök kiiktatása – **primál** megközelítés

# Algoritmusok

Összesen 9 algoritmust valósítottunk meg, amelyek 4 különböző megoldási módszeren alapulnak.

1. Negatív körök kiiktatása – **primál** megközelítés
2. Javítóutas algoritmusok – **duál** megközelítés

# Algoritmusok

Összesen 9 algoritmust valósítottunk meg, amelyek 4 különböző megoldási módszeren alapulnak.

1. Negatív körök kiiktatása – **primál** megközelítés
2. Javítóutas algoritmusok – **duál** megközelítés
3. Költségskálázás – **primál–duál** megközelítés

# Algoritmusok

Összesen 9 algoritmust valósítottunk meg, amelyek 4 különböző megoldási módszeren alapulnak.

1. Negatív körök kiiktatása – **primál** megközelítés
2. Javítóutas algoritmusok – **duál** megközelítés
3. Költségskálázás – **primál–duál** megközelítés
4. Hálózati szimplex módszer

# Tesztadatok, tesztelés paraméterei

Tesztadatok:

- NETGEN programmal generált inputfájlok (DIMACS formátum).

# Tesztadatok, tesztelés paraméterei

Tesztadatok:

- NETGEN programmal generált inputfájlok (DIMACS formátum).
  - Ritka hálózatok: $m \approx n \log_2 n$ ($n \leq 1\,000\,000$, $m \leq 20\,000\,000$).
  - Sűrű hálózatok: $m \approx n\sqrt{n}$ ($n \leq 100\,000$, $m \leq 32\,000\,000$).

  $n$ = csúcsok száma, $m$ = élek száma.

# Tesztadatok, tesztelés paraméterei

Tesztadatok:

- NETGEN programmal generált inputfájlok (DIMACS formátum).
  - Ritka hálózatok: $m \approx n \log_2 n$ ($n \leq 1\,000\,000$, $m \leq 20\,000\,000$).
  - Sűrű hálózatok: $m \approx n\sqrt{n}$ ($n \leq 100\,000$, $m \leq 32\,000\,000$).

  $n$ = csúcsok száma, $m$ = élek száma.
- Élköltségek, kapacitások: az $[1..10\,000]$ intervallumból.

# Tesztadatok, tesztelés paraméterei

Tesztadatok:

- NETGEN programmal generált inputfájlok (DIMACS formátum).
  - Ritka hálózatok: $m \approx n \log_2 n$ ($n \leq 1\,000\,000$, $m \leq 20\,000\,000$).
  - Sűrű hálózatok: $m \approx n\sqrt{n}$ ($n \leq 100\,000$, $m \leq 32\,000\,000$).

  $n$ = csúcsok száma, $m$ = élek száma.
- Élköltségek, kapacitások: az $[1..10\,000]$ intervallumból.
- $\sqrt{n}$ termelő és $\sqrt{n}$ fogyasztó csúcs.
- Össztermelés: $1000\sqrt{n}$.

# Tesztadatok, tesztelés paraméterei

Tesztadatok:

- NETGEN programmal generált inputfájlok (DIMACS formátum).
  - Ritka hálózatok: $m \approx n \log_2 n$ ($n \leq 1\,000\,000$, $m \leq 20\,000\,000$).
  - Sűrű hálózatok: $m \approx n\sqrt{n}$ ($n \leq 100\,000$, $m \leq 32\,000\,000$).

  $n$ = csúcsok száma, $m$ = élek száma.
- Élköltségek, kapacitások: az $[1..10\,000]$ intervallumból.
- $\sqrt{n}$ termelő és $\sqrt{n}$ fogyasztó csúcs.
- Össztermelés: $1000\sqrt{n}$.

Tesztkörnyezet:

- Két Intel Xeon 3.20 GHz processzor, 2 MB cache, 2 GB memória;
- openSUSE 10.2, GCC 4.1.2 fordító, –O3 optimalizáció.

# Negatív körök kiiktatása

Optimalitási kritérium:

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

# Negatív körök kiiktatása

Optimalitási kritérium:

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

*Megjegyzés:* egy megfordított reziduális él költsége az eredeti élköltség ellentettje.

# Negatív körök kiiktatása

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

A tételből egy egyszerű megoldási módszer is adódik:

# Negatív körök kiiktatása

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

A tételből egy egyszerű megoldási módszer is adódik:

1. Keressünk egy megengedett megoldást (visszavezethető maximális folyam feladatra).

# Negatív körök kiiktatása

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

A tételből egy egyszerű megoldási módszer is adódik:

1. Keressünk egy megengedett megoldást (visszavezethető maximális folyam feladatra).
2. Amíg a reziduális hálózat tartalmaz negatív kört, keressünk meg egyet és iktassuk ki: folyassunk rajta annyi folyamot, hogy valamely éle telítődjön.

# Negatív körök kiiktatása

**1. Tétel:** Negatív kör optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha a $G_x$ reziduális hálózat nem tartalmaz negatív összköltségű irányított kört.

A tételből egy egyszerű megoldási módszer is adódik:

1. Keressünk egy megengedett megoldást (visszavezethető maximális folyam feladatra).
2. Amíg a reziduális hálózat tartalmaz negatív kört, keressünk meg egyet és iktassuk ki: folyassunk rajta annyi folyamot, hogy valamely éle telítődjön.

*Primál módszer*: minden lépésben megengedett megoldást kapunk, és a célfüggvény értékét fokozatosan csökkentjük.

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

- Negatív körök keresése Bellman–Ford-algoritmussal.
- Hatékony heurisztikákat dolgoztunk ki.

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

- Minden menetben egy min. átlagú kör keresése és kiiktatása.

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

- Minden menetben egy min. átlagú kör keresése és kiiktatása.
- Egyszerű, közismert erősen polinomiális algoritmus.
- A gyakorlatban viszont rendkívül lassú.

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

- Az MMCC javított változata.

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

- Az MMCC javított változata.
- Valójában *primál–duál* módszert alkalmaz.
- Csúcspotenciálok (duál megoldás) nyilvántartásával átlagosan jóval gyorsabban találhatunk negatív köröket.

# Negatív kör algoritmusok

Megvalósított algoritmusok:

**SCC**: *simple cycle canceling* (egyszerű negatív kör algoritmus)

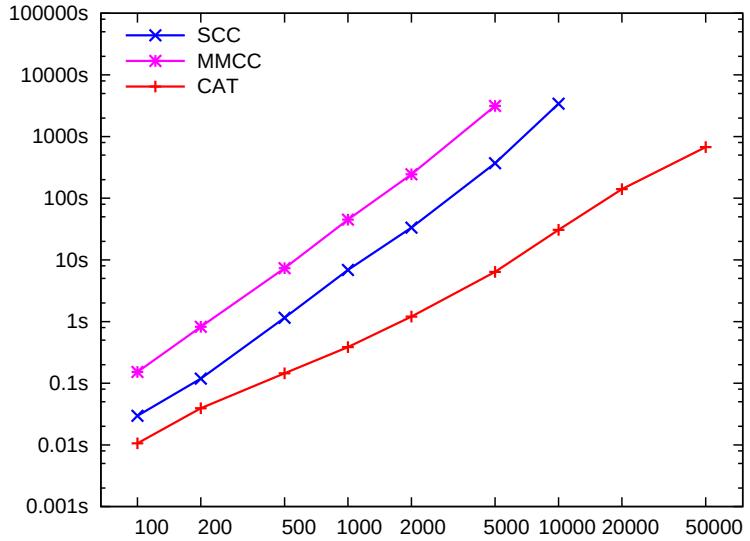
**MMCC**: *minimum mean cycle canceling* (min. átlagú körök kiiktatása)

**CAT**: *cancel and tighten* (kiiktatás és megszorítás)

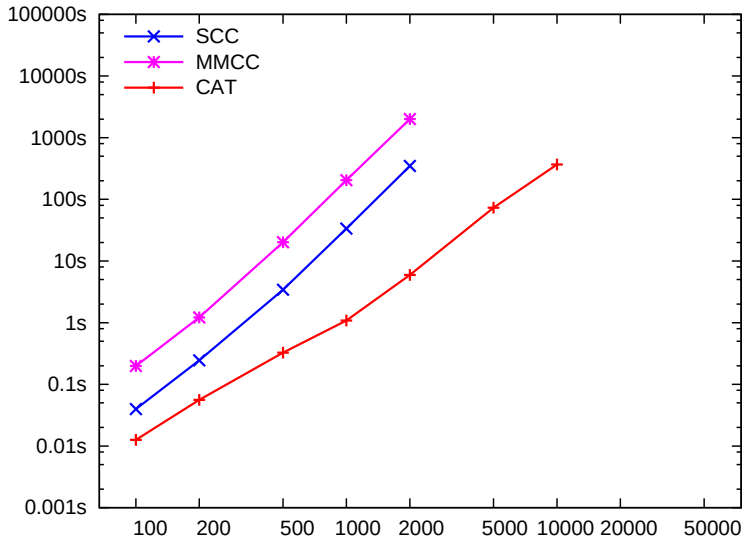
- Az MMCC javított változata.
- Valójában *primál–duál* módszert alkalmaz.
- Csúcspotenciálok (duál megoldás) nyilvántartásával átlagosan jóval gyorsabban találhatunk negatív köröket.
- Ez is erősen polinomiális, de elméletben és gyakorlatban is sokkal hatékonyabb.

# Negatív kör algoritmusok

Az alábbi grafikonokon összehasonlítjuk a negatív kör algoritmusokat. A futásidőket a csúcsok számának függvényében, logaritmikus skálán ábrázoljuk.




Ritka hálózatokon ($m \approx n \log_2 n$)

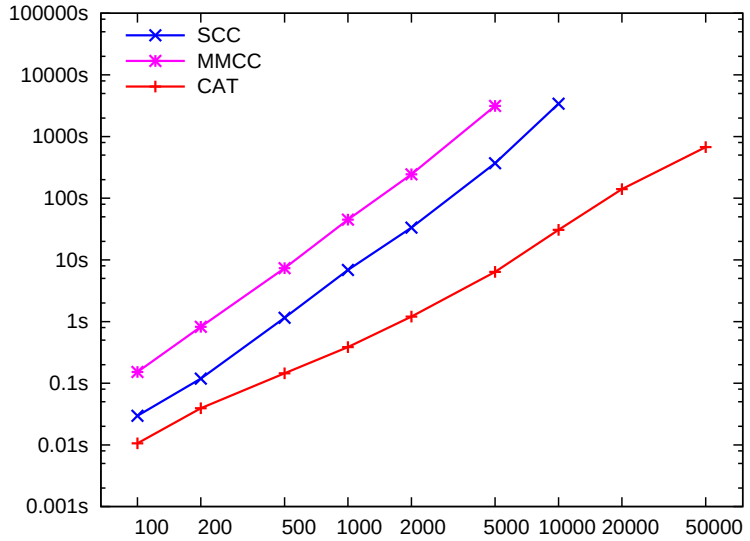



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

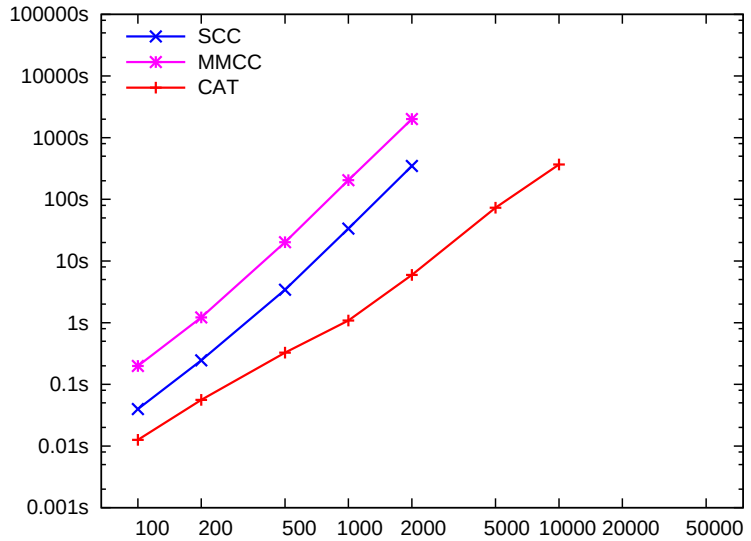
- **SCC**: *simple cycle canceling*
- **MMCC**: *minimum mean cycle canceling*
- **CAT**: *cancel and tighten*

# Negatív kör algoritmusok

Az alábbi grafikonokon összehasonlítjuk a negatív kör algoritmusokat. A futásidőket a csúcsok számának függvényében, logaritmikus skálán ábrázoljuk.




Ritka hálózatokon ($m \approx n \log_2 n$)




Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- Az **SCC** 6-8-szor gyorsabb, mint az **MMCC**.
- A **CAT** viszont nagyságrendekkel gyorsabb mindkettőnél.

# Duál algoritmusok

Duál megoldási módszer:

# Duál algoritmusok

Duál megoldási módszer:

- Megengedett duál megoldást tartunk fent, és a primál megengedettség felé törekszünk.

# Duál algoritmusok

Duál megoldási módszer:

- Megengedett duál megoldást tartunk fent, és a primál megengedettség felé törekszünk.
- Egy folyamot és csúcspotenciálokat tárolunk.
- A folyam *nem feltétlenül megengedett*: teljesülnek a kapacitásfeltételek, de megsértheti a termelés/fogyasztás feltételeket.

# Duál algoritmusok

Duál megoldási módszer:

- Megengedett duál megoldást tartunk fent, és a primál megengedettség felé törekszünk.
- Egy folyamot és csúcspotenciálokat tárolunk.
- A folyam *nem feltétlenül megengedett*: teljesülnek a kapacitásfeltételek, de megsértheti a termelés/fogyasztás feltételeket.
- Minden lépésben egy többlettel rendelkező csúcsból folyamot küldünk egy hiánnyal rendelkező csúcsba.
- Ezt a reziduális hálózat egy, a redukált költségek szerint vett legrövidebb útja mentén tesszük.

# Duál algoritmusok

Duál megoldási módszer:

- Megengedett duál megoldást tartunk fent, és a primál megengedettség felé törekszünk.
- Egy folyamot és csúcspotenciálokat tárolunk.
- A folyam *nem feltétlenül megengedett*: teljesülnek a kapacitásfeltételek, de megsértheti a termelés/fogyasztás feltételeket.
- Minden lépésben egy többlettel rendelkező csúcsból folyamot küldünk egy hiánnyal rendelkező csúcsba.
- Ezt a reziduális hálózat egy, a redukált költségek szerint vett legrövidebb útja mentén tesszük.
- Ha már nincs többletes csúcs, akkor megengedett primál megoldást kapunk.
- Ez optimális is lesz (ui. a duál megengedettséget fenntartjuk).

# Duál algoritmusok

A feladat duálisát csúcspotenciálok segítségével írhatjuk fel.

# Duál algoritmusok

A feladat duálisát csúcspotenciálok segítségével írhatjuk fel.

Az 1. tétellel ekvivalens optimalitási kritérium:

**2. Tétel:** Redukált költség optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha létezik olyan $\pi$ potenciálfüggvény, amellyel a $G_x$ reziduális hálózat minden élének redukált költsége nemnegatív.

# Duál algoritmusok

A feladat duálisát csúcspotenciálok segítségével írhatjuk fel.

Az 1. tétellel ekvivalens optimalitási kritérium:

**2. Tétel:** Redukált költség optimalitási feltétel

A minimális költségű folyam feladat egy $x$ megengedett megoldása akkor és csak akkor optimális, ha létezik olyan $\pi$ potenciálfüggvény, amellyel a $G_x$ reziduális hálózat minden élének redukált költsége nemnegatív.

**Definíció:** Redukált költség

Egy $\pi$ potenciálfüggvényre nézve egy $(i,j)$ él redukált költsége:

$$c_{ij}^{\pi} = c_{ij} + \pi(i) - \pi(j).$$

# Duál algoritmusok

Megvalósított algoritmusok:

**SSP**: *successive shortest path* (ismételt legrövidebb út)

**CAS**: *capacity scaling* (kapacitásskálázás)

# Duál algoritmusok

Megvalósított algoritmusok:

**SSP**: *successive shortest path* (ismételt legrövidebb út)

- Egyszerű változat.

**CAS**: *capacity scaling* (kapacitásskálázás)

# Duál algoritmusok

Megvalósított algoritmusok:

**SSP**: *successive shortest path* (ismételt legrövidebb út)

**CAS**: *capacity scaling* (kapacitásskálázás)

- Az SSP algoritmus egy hatékonyabb (polinomiális) változata.

# Duál algoritmusok

Megvalósított algoritmusok:

**SSP**: *successive shortest path* (ismételt legrövidebb út)

**CAS**: *capacity scaling* (kapacitásskálázás)

- Az SSP algoritmus egy hatékonyabb (polinomiális) változata.
- Minden lépésben olyan legrövidebb utakat keresünk, amelyeken legalább $\Delta$ egységnyi folyamot küldhetünk át.

# Duál algoritmusok

Megvalósított algoritmusok:

**SSP**: *successive shortest path* (ismételt legrövidebb út)

**CAS**: *capacity scaling* (kapacitásskálázás)

- Az SSP algoritmus egy hatékonyabb (polinomiális) változata.
- Minden lépésben olyan legrövidebb utakat keresünk, amelyeken legalább $\Delta$ egységnyi folyamot küldhetünk át.
- Ha ilyet már nem találunk, felezzük $\Delta$ értékét, és végrehajtunk egy újabb fázist.

# Duál algoritmusok

Megvalósított algoritmusok:

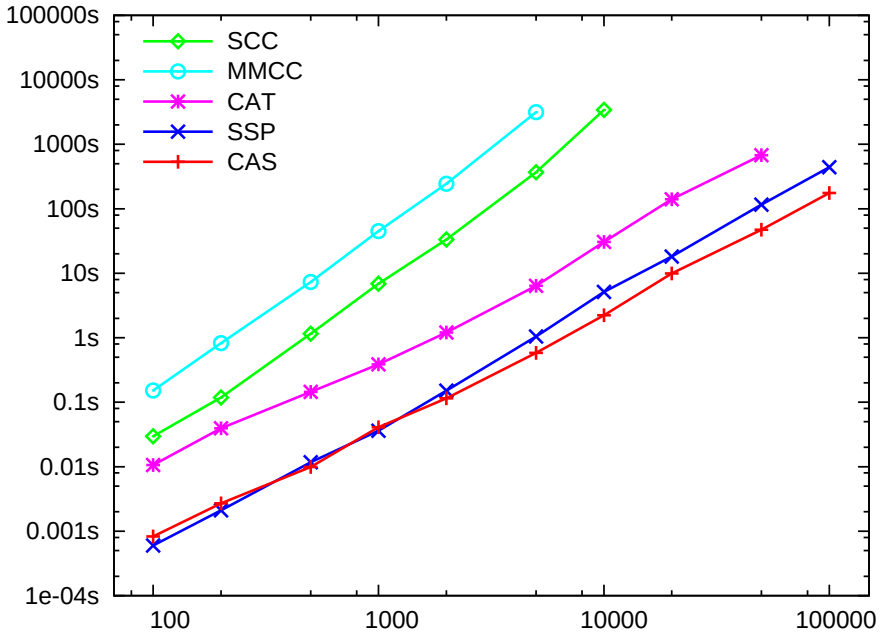
**SSP**: *successive shortest path* (ismételt legrövidebb út)

**CAS**: *capacity scaling* (kapacitásskálázás)

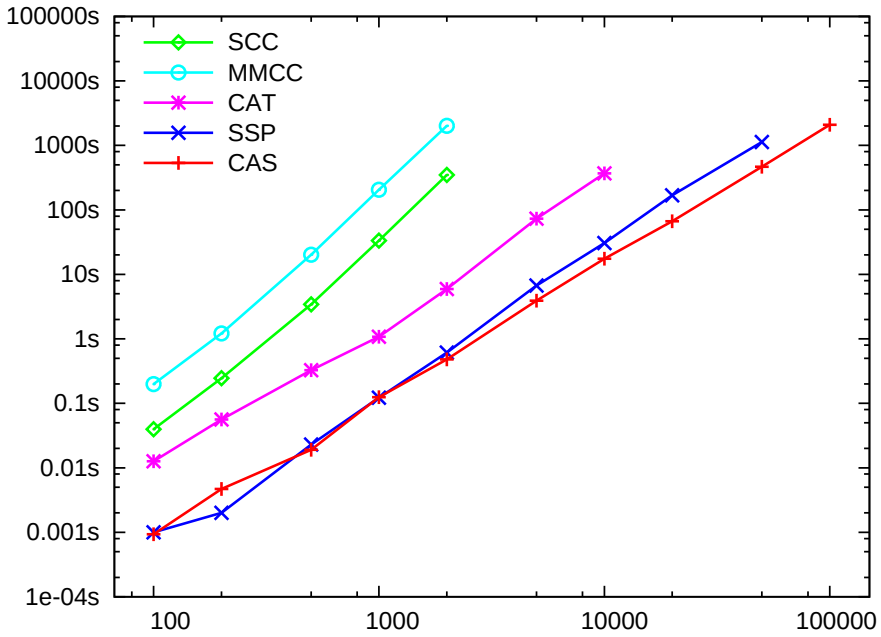
- Az SSP algoritmus egy hatékonyabb (polinomiális) változata.
- Minden lépésben olyan legrövidebb utakat keresünk, amelyeken legalább $\Delta$ egységnyi folyamot küldhetünk át.
- Ha ilyet már nem találunk, felezzük $\Delta$ értékét, és végrehajtunk egy újabb fázist.
- A $\Delta = 1$ fázis végén megengedett és optimális folyamot kapunk.

# Összehasonlítás

Primál és duál implementációk összehasonlítása:




Ritka hálózatokon ($m \approx n \log_2 n$)

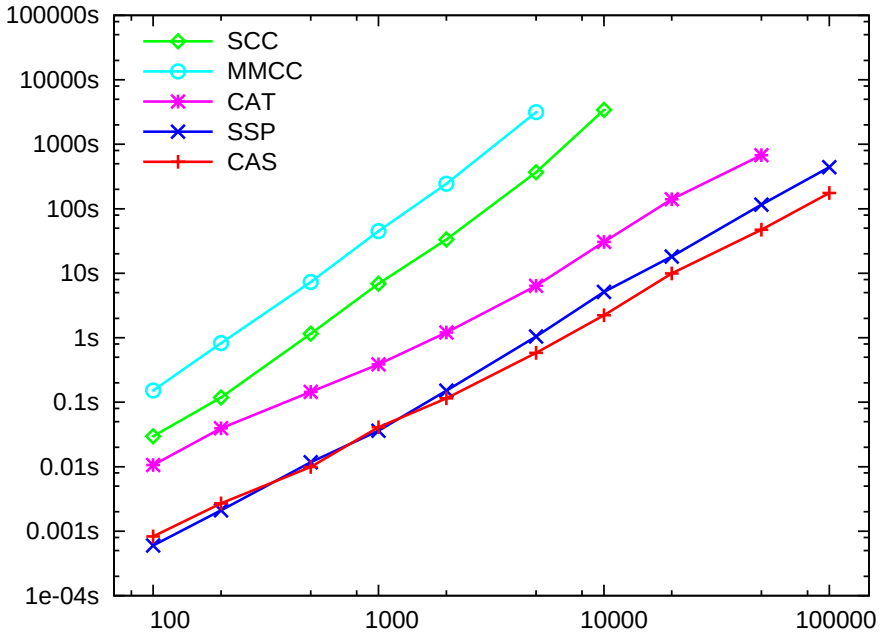



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

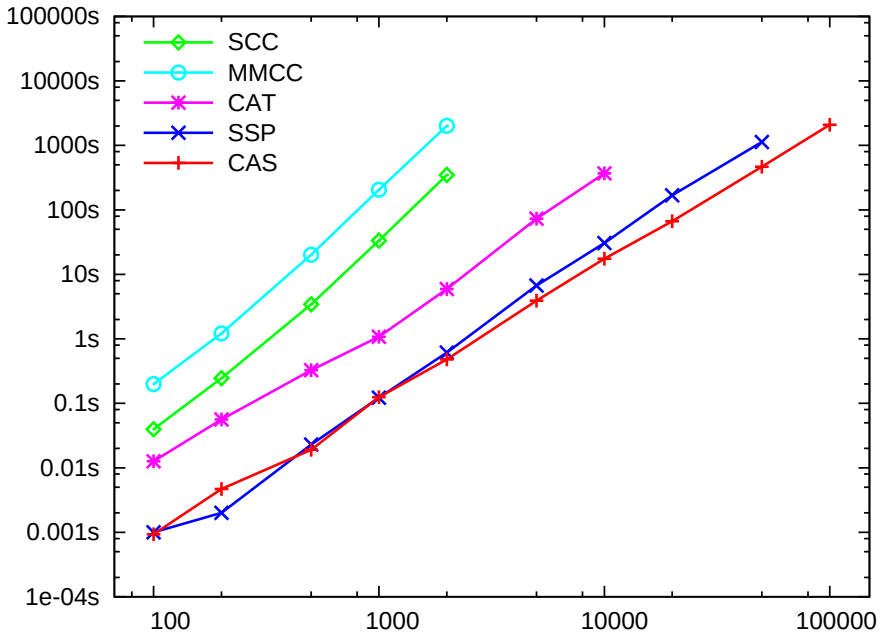
- **SCC**, **MMCC**, **CAT**: primál algoritmusok (negatív körök kiiktatása)
- **SSP**, **CAS**: duál algoritmusok (ismételt legrövidebb út keresés)

# Összehasonlítás

Primál és duál implementációk összehasonlítása:




Ritka hálózatokon ($m \approx n \log_2 n$)




Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- **SCC**, **MMCC**, **CAT**: primál algoritmusok (negatív körök kiiktatása)
- **SSP**, **CAS**: duál algoritmusok (ismételt legrövidebb út keresés)
- A javítóutas módszerek sokkal gyorsabbnak bizonyultak.

# Költségskálázó algoritmusok

Költségskálázó módszer:

# Költségskálázó algoritmusok

Költségskálázó módszer:

- *Primál–duál* megközelítés.
- A max. folyam feladatra adott előfolyam algoritmus (*preflow push-relabel*) egy módosított változata.

# Költségskálázó algoritmusok

Költségskálázó módszer:

- *Primál–duál* megközelítés.
- A max. folyam feladatra adott előfolyam algoritmus (*preflow push-relabel*) egy módosított változata.
- Minden fázisban $\epsilon$-optimális primál–duál megoldáspárt állítunk elő: a reziduális hálózat minden $(i,j)$ élére: $c_{ij}^{\pi} \geq -\epsilon$.

# Költségskálázó algoritmusok

Költségskálázó módszer:

- *Primál–duál* megközelítés.
- A max. folyam feladatra adott előfolyam algoritmus (*preflow push-relabel*) egy módosított változata.
- Minden fázisban $\epsilon$-optimális primál–duál megoldáspárt állítunk elő: a reziduális hálózat minden $(i,j)$ élére: $c_{ij}^{\pi} \geq -\epsilon$.
- Ezután $\epsilon$-t felezzük, és egy újabb menetet hajtunk végre.
- $\epsilon < 1/n$ esetén optimális megoldást kapunk.

# Költségskálázó algoritmusok

Költségskálázó módszer:

- *Primál–duál* megközelítés.
- A max. folyam feladatra adott előfolyam algoritmus (*preflow push-relabel*) egy módosított változata.
- Minden fázisban $\epsilon$-optimális primál–duál megoldáspárt állítunk elő: a reziduális hálózat minden $(i,j)$ élére: $c_{ij}^{\pi} \geq -\epsilon$.
- Ezután $\epsilon$-t felezzük, és egy újabb menetet hajtunk végre.
- $\epsilon < 1/n$ esetén optimális megoldást kapunk.
- Az egyes fázisokban *pumpálás* (*push*) és *átcímkézés* (*relabel*) műveleteket végzünk.

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

- Az eredeti változat lokális pumpálás (*push*) és átcímkézés (*relabel*) műveletekkel.

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

- Pumpálás helyett javítóutakat keresünk többletes csúcsokból hiányos csúcsokba.
- Ezen utak mentén növeljük a folyamot (*augment*).

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

- Goldberg új ötletét átültettük erre a feladatra: a javítóutak hosszát korlátozzuk.

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

- Goldberg új ötletét átültettük erre a feladatra: a javítóutak hosszát korlátozzuk.
- Legfeljebb $k = 4$ élből álló utak mentén növelünk (ekvivalens $k$ pumpálással).

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

- Goldberg új ötletét átültettük erre a feladatra: a javítóutak hosszát korlátozzuk.
- Legfeljebb $k = 4$ élből álló utak mentén növelünk (ekvivalens $k$ pumpálással).
- Forrás: Andrew V. Goldberg: *The partial augment-relabel algorithm for the maximum flow problem*. ESA 2008, 466–477, 2008.

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

- A három változat teljesítménye nagyon hasonló.

# Költségskálázó algoritmusok

Megvalósított algoritmusok:

**COS-PR**: *push-relabel* (pumpálás-átcímkézés)

**COS-AR**: *augment-relabel* (növelés-átcímkézés)

**COS-PAR**: *partial augment-relabel* (részleges növelés-átcímkézés)

- A három változat teljesítménye nagyon hasonló.
- Általában a **COS-PAR** algoritmus bizonyult a leggyorsabbnak.
- A továbbiakban csak ezt vizsgáljuk.

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

- Az LP szimplex módszer specializált változata erre a feladatra.

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

- Az LP szimplex módszer specializált változata erre a feladatra.
- Az LP változóknak a gráf élei felelnek meg.
- A bázismegoldásoknak pedig ún. *feszítőfa-megoldások*: olyan feszítőfák, amelyeken kívül minden él korlátozott (a folyamértéke nulla vagy telített).

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

- Az LP szimplex módszer specializált változata erre a feladatra.
- Az LP változóknak a gráf élei felelnek meg.
- A bázismegoldásoknak pedig ún. *feszítőfa-megoldások*: olyan feszítőfák, amelyeken kívül minden él korlátozott (a folyamértéke nulla vagy telített).
- Egy feszítőfát és egy folyamot (primál megoldás), valamint csúcspotenciálokat (duál megoldás) tartunk nyilván.

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

- Az LP szimplex módszer specializált változata erre a feladatra.
- Az LP változóknak a gráf élei felelnek meg.
- A bázismegoldásoknak pedig ún. *feszítőfa-megoldások*: olyan feszítőfák, amelyeken kívül minden él korlátozott (a folyamértéke nulla vagy telített).
- Egy feszítőfát és egy folyamot (primál megoldás), valamint csúcspotenciálokat (duál megoldás) tartunk nyilván.
- Az egyes iterációk során a célfüggvény értékén próbálunk javítani.

# Hálózati szimplex algoritmus

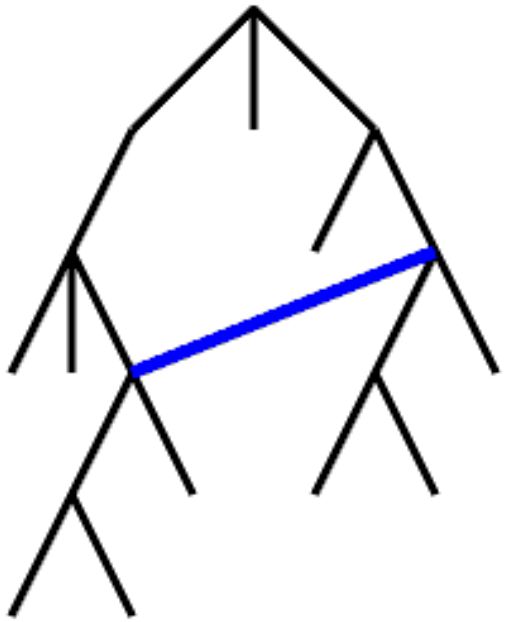
Primál hálózati szimplex módszer:

- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.

# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

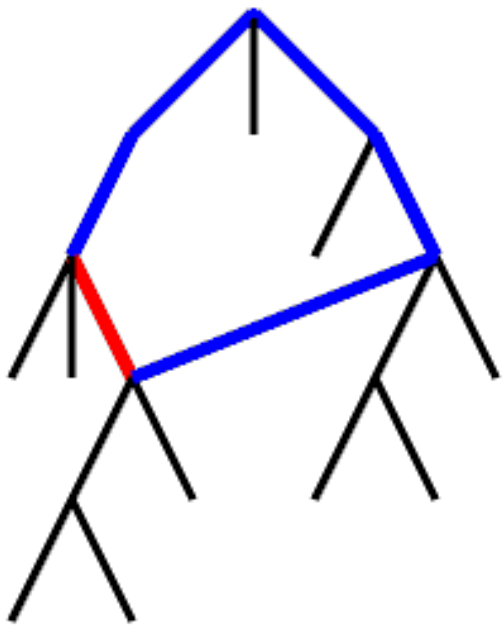
- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).



# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

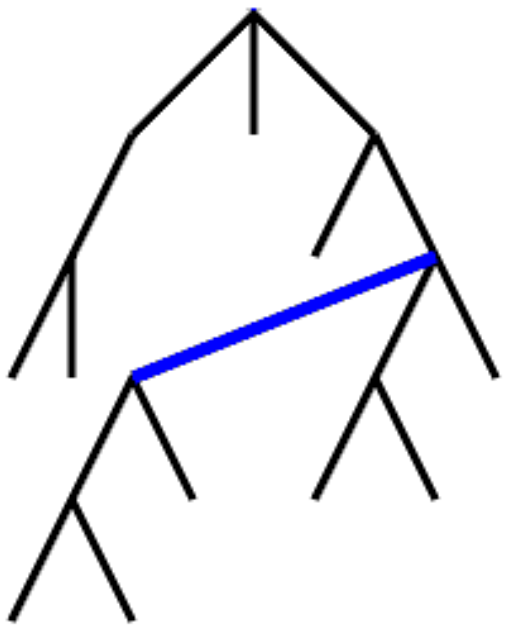
- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).
- Az így kapott negatív kör mentén javítunk: folyamot küldünk rajta és egy telítődő élét kivesszük a bázisból (pivotálás).



# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

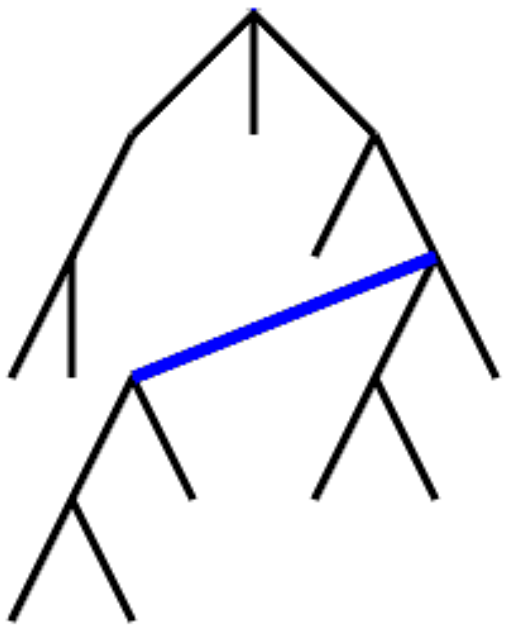
- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).
- Az így kapott negatív kör mentén javítunk: folyamot küldünk rajta és egy telítődő élét kivesszük a bázisból (pivotálás).



# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

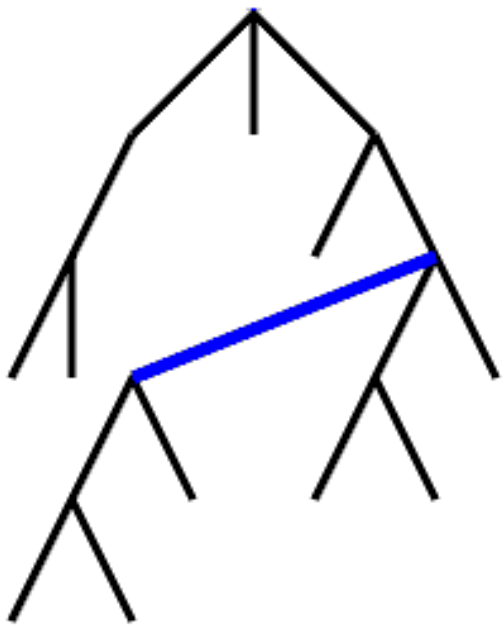
- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).
- Az így kapott negatív kör mentén javítunk: folyamot küldünk rajta és egy telítődő élét kivesszük a bázisból (pivotálás).
- Ha már nincs alkalmas belépő él, akkor a folyam optimális.



# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

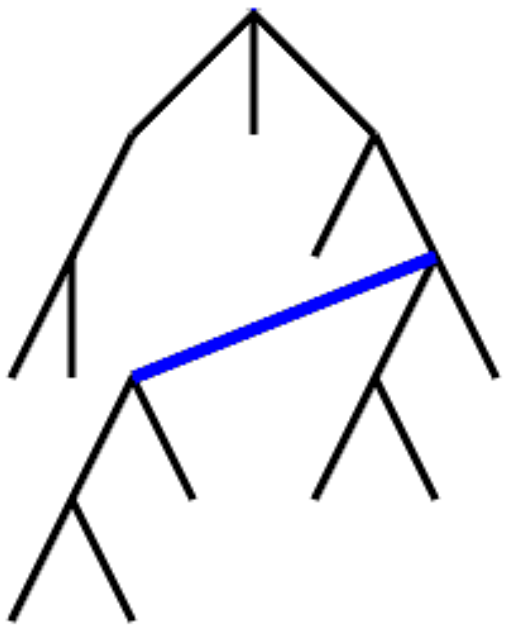
- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).
- Az így kapott negatív kör mentén javítunk: folyamot küldünk rajta és egy telítődő élét kivesszük a bázisból (pivotálás).
- Ha már nincs alkalmas belépő él, akkor a folyam optimális.
- Valójában a primál módszer (*körök kiiktatása*) egy sajátos változata.



# Hálózati szimplex algoritmus

Primál hálózati szimplex módszer:

- Minden lépésben kiválasztunk egy fán kívüli élt, amely megsérti az optimalitási feltételt.
- Ezt hozzávesszük a feszítőfához (bázishoz).
- Az így kapott negatív kör mentén javítunk: folyamot küldünk rajta és egy telítődő élét kivesszük a bázisból (pivotálás).
- Ha már nincs alkalmas belépő él, akkor a folyam optimális.
- Valójában a primál módszer (*körök kiiktatása*) egy sajátos változata.
- Az adatszerkezetnek köszönhetően egy negatív kör megtalálása sokkal gyorsabb ($O(m)$ idejű).



# Hálózati szimplex algoritmus

Implementáció:

- Komplex adatszerkezet szükséges: feszítőfa nyilvántartása és hatékony módosítása.

# Hálózati szimplex algoritmus

Implementáció:

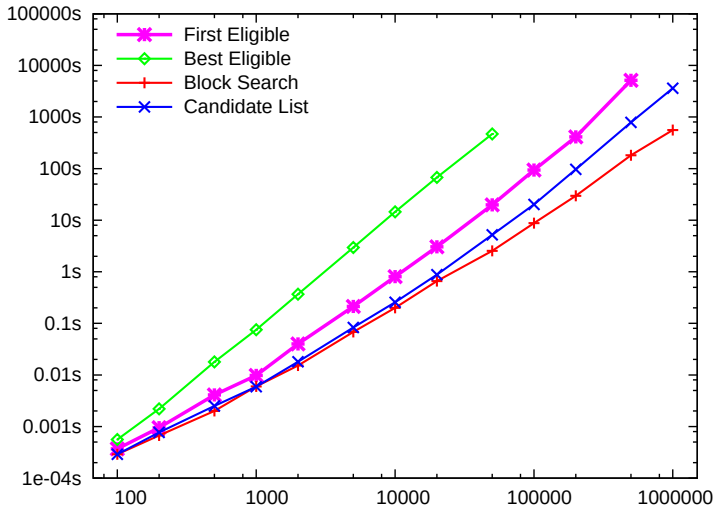
- Komplex adatszerkezet szükséges: feszítőfa nyilvántartása és hatékony módosítása.
- Erre sok különböző módszer ismert: ATI, API, XTI, XPI stb.
- Az egyik leghatékonyabbat, az XTI-t valósítottuk meg.

# Hálózati szimplex algoritmus

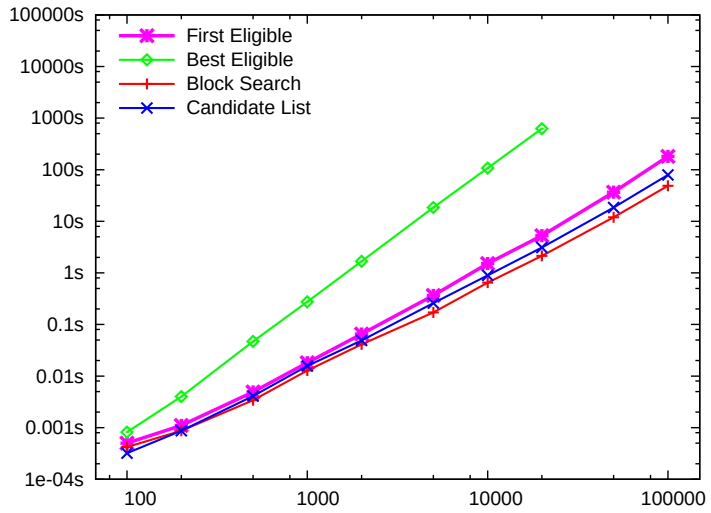
Implementáció:

- Komplex adatszerkezet szükséges: feszítőfa nyilvántartása és hatékony módosítása.
- Erre sok különböző módszer ismert: ATI, API, XTI, XPI stb.
- Az egyik leghatékonyabbat, az XTI-t valósítottuk meg.
- A legkritikusabb művelet a belépő él kiválasztása.
- Erre 5 különböző stratégiát implementáltunk, amelyek hatékonysága nagyon különböző.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

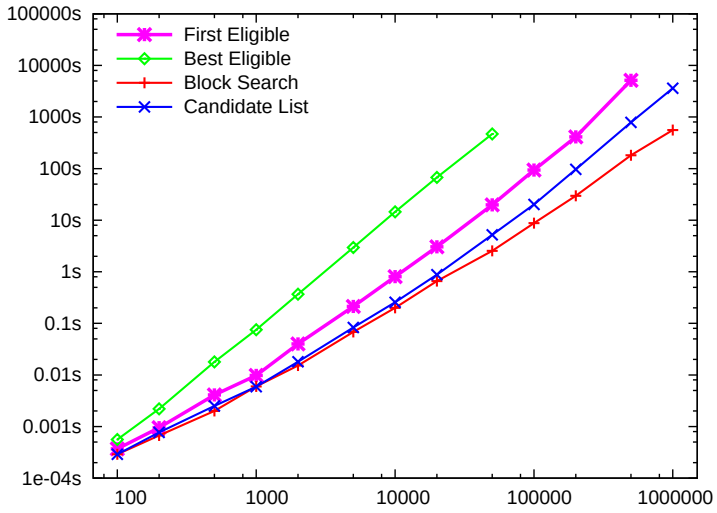



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

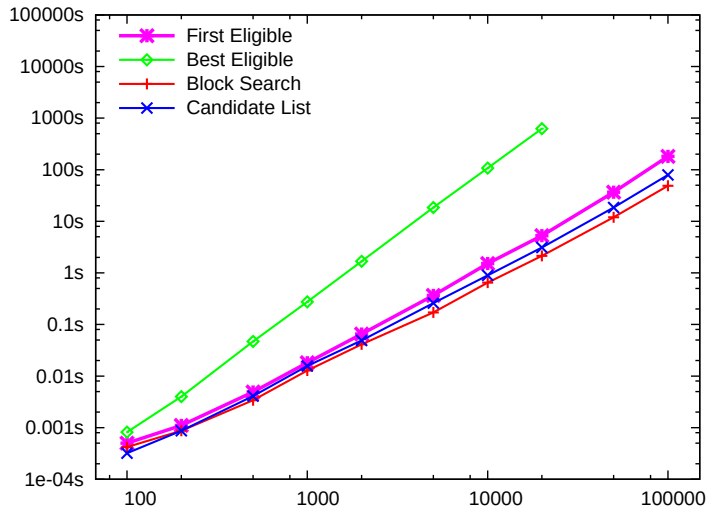
**First Eligible** módszer:

- A legelső élt választjuk, amelyre nem teljesül az optimalitási feltétel.
- Minden keresést az előző lépésben talált él után kezdünk.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

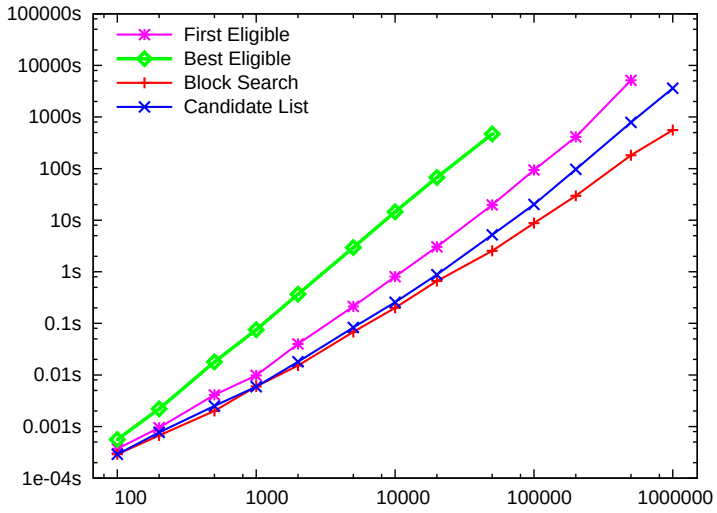



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

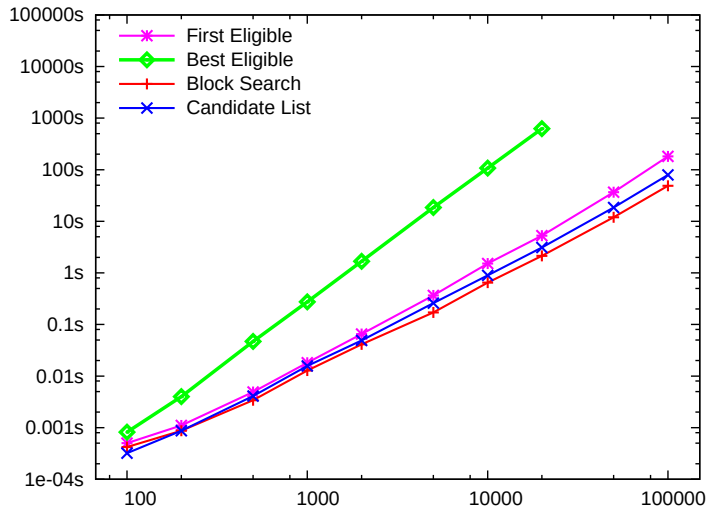
**First Eligible** módszer:

- A legelső élt választjuk, amelyre nem teljesül az optimalitási feltétel.
- Minden keresést az előző lépésben talált él után kezdünk.
- Mindig gyorsan talál belépő élt, de az sokszor nem lesz túl jó.
- Egyszerűsége ellenére elég hatékony módszer.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

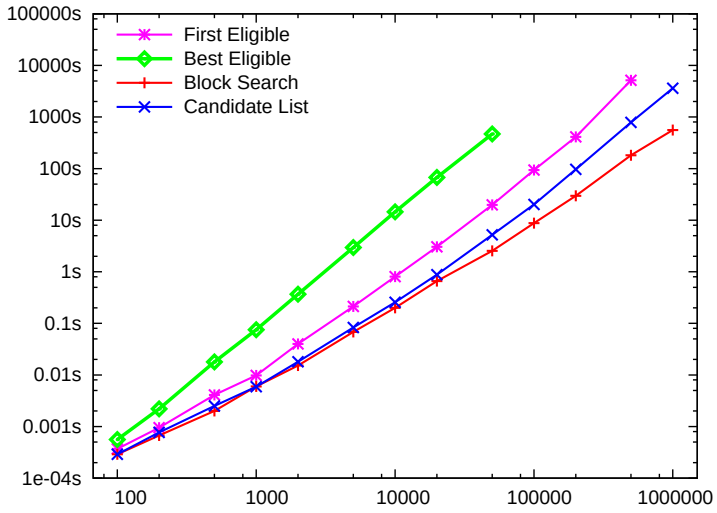



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

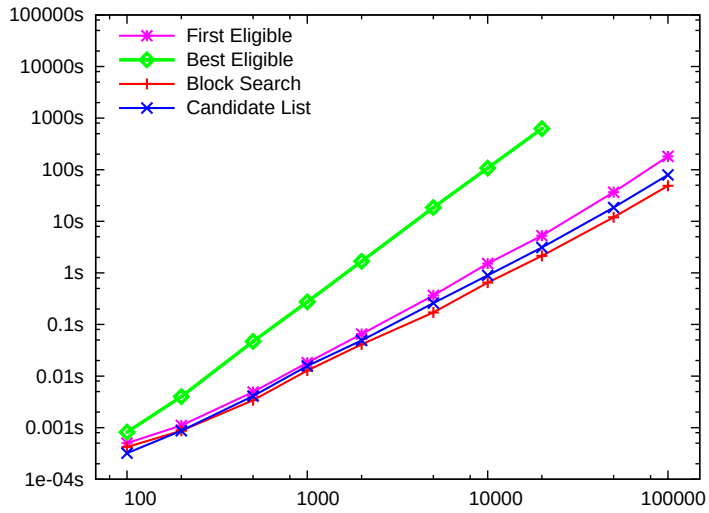
**Best Eligible** módszer:

- Minden lépésben a legjobb él választjuk.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

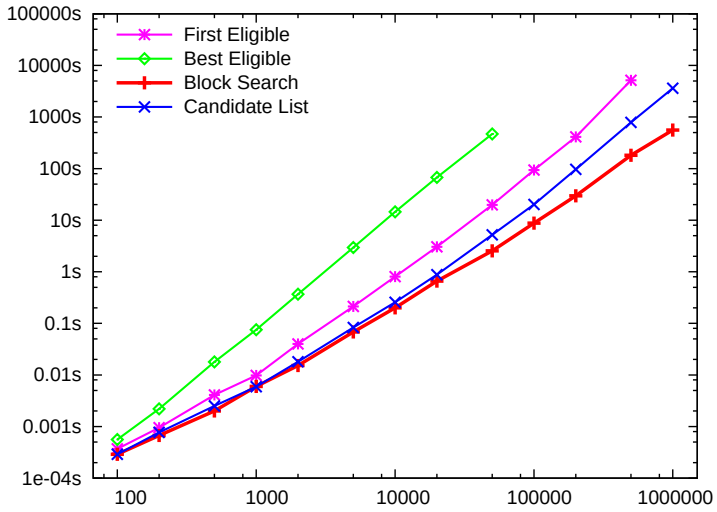



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

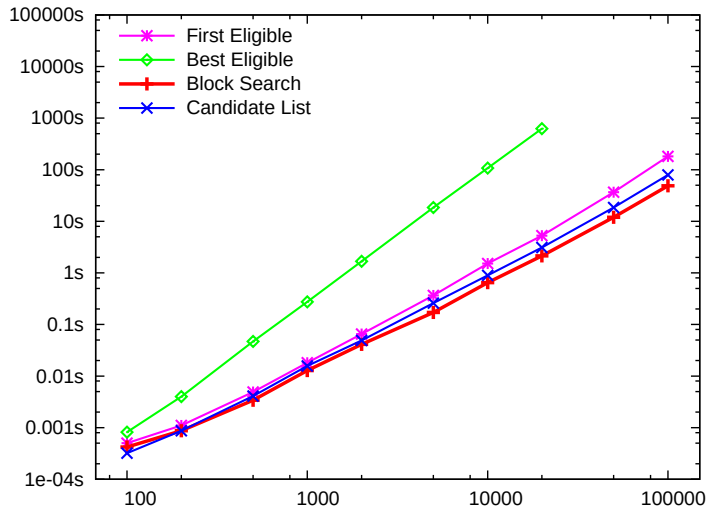
**Best Eligible** módszer:

- Minden lépésben a legjobb él választjuk.
- Ez eredményezi a legkevesebb iterációt.
- Viszont ehhez meg kell vizsgálnunk az összes élt, így a módszer nagyságrendekkel lassabb a többi szabálynál.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

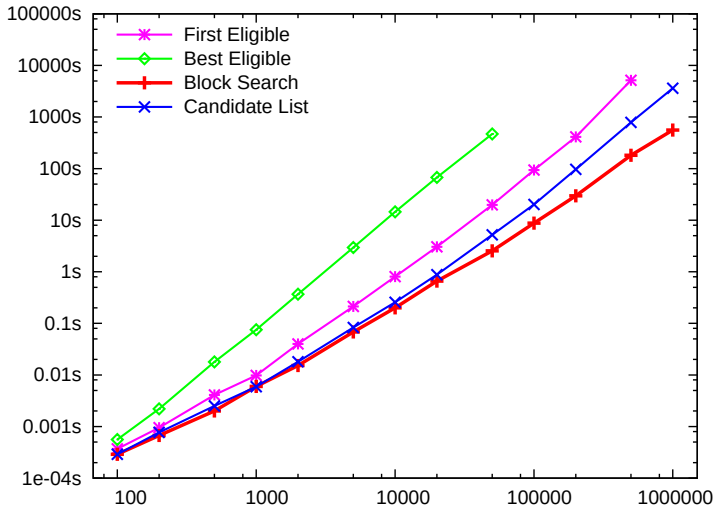



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

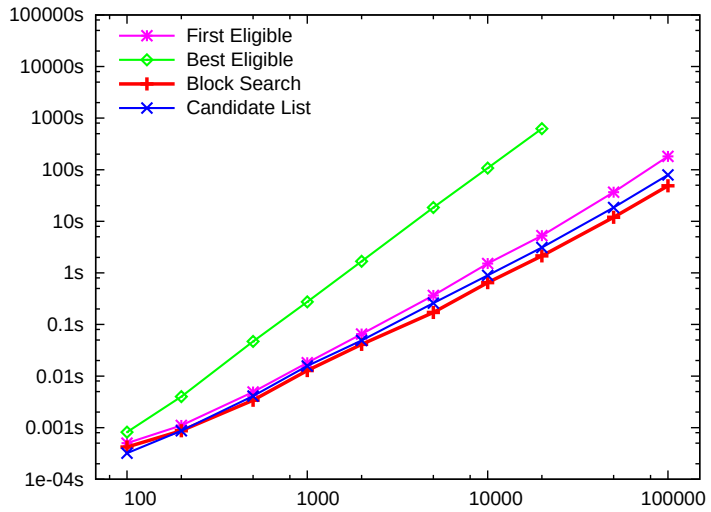
**Block Search** módszer:

- Az előző két stratégia közötti „középút”.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

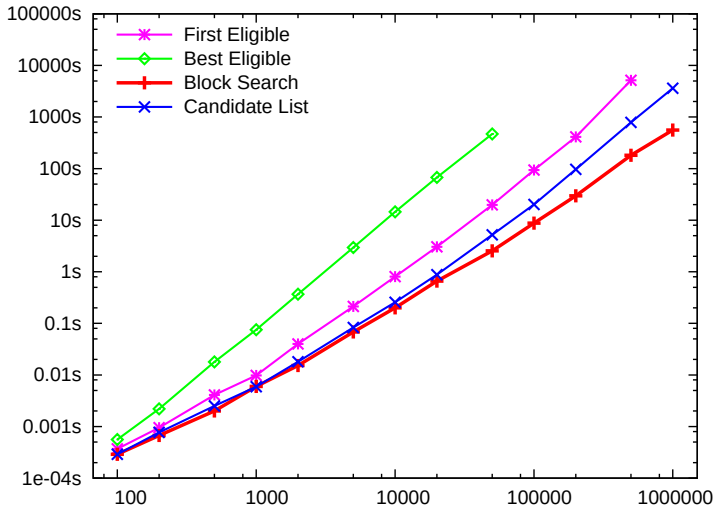



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

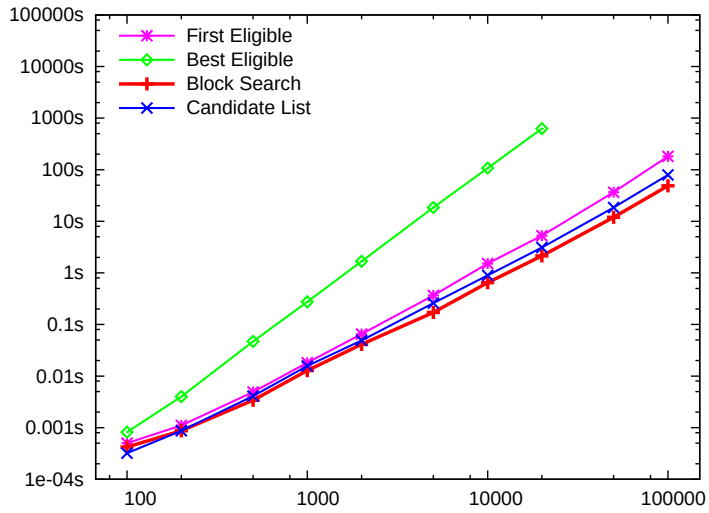
**Block Search** módszer:

- Az előző két stratégia közötti „középút”.
- Minden lépésben rögzített mennyiségű élt nézzük át, és azokból választjuk ki a legjobbat.
- Ha egy blokkban nincs alkalmas él, akkor újabb blokkokkal folytatjuk.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

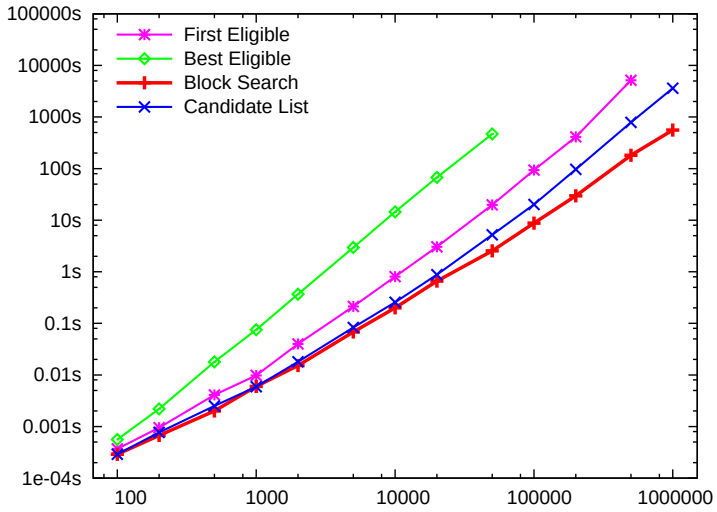



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

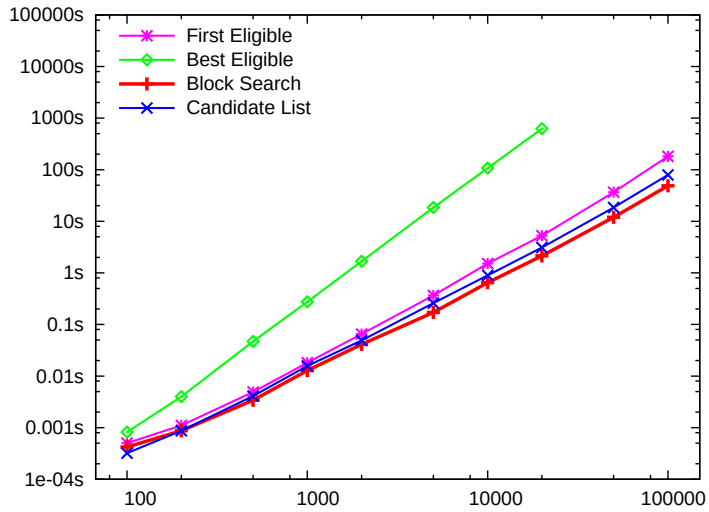
**Block Search** módszer:

- Az előző két stratégia közötti „középút”.
- Minden lépésben rögzített mennyiségű élt nézzük át, és azokból választjuk ki a legjobbat.
- Ha egy blokkban nincs alkalmas él, akkor újabb blokkokkal folytatjuk.
- Fontos paraméter a blokkméret: egy kompromisszumot határoz meg.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

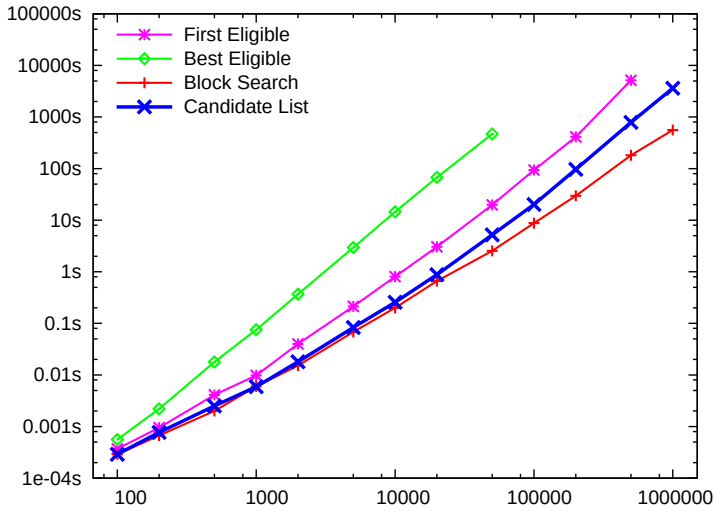



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

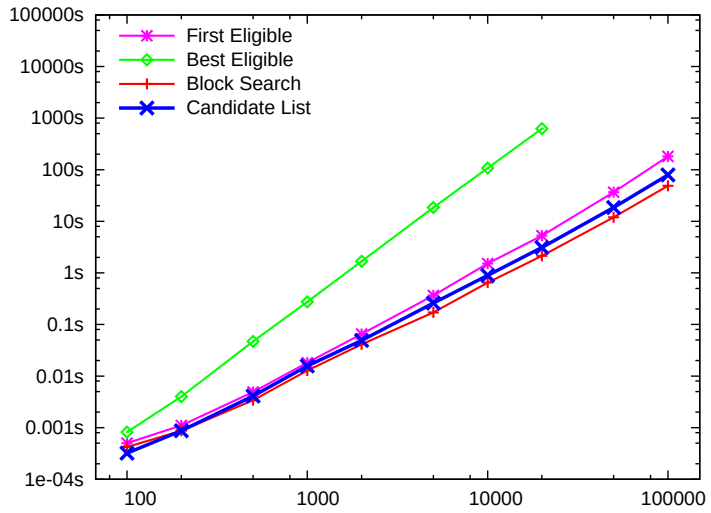
**Block Search** módszer:

- Az előző két stratégia közötti „középút”.
- Minden lépésben rögzített mennyiségű élt nézzük át, és azokból választjuk ki a legjobbat.
- Ha egy blokkban nincs alkalmas él, akkor újabb blokkokkal folytatjuk.
- Fontos paraméter a blokkméret: egy kompromisszumot határoz meg.
- Ez a szabály bizonyult a leghatékonyabbnak és legstabilabbnak.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

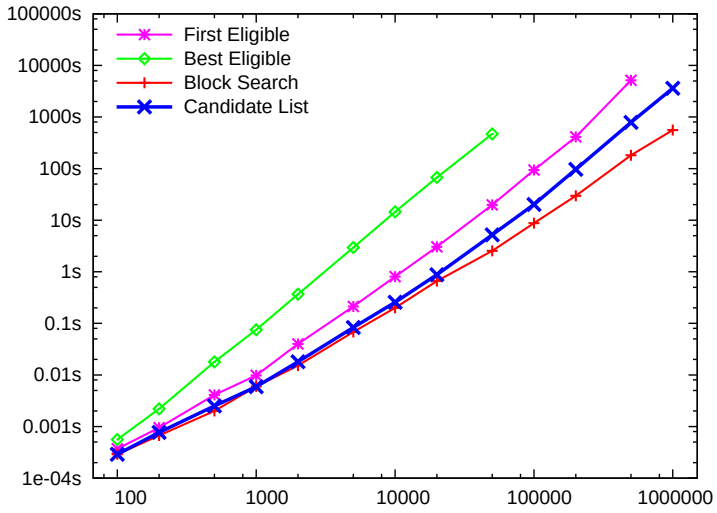



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

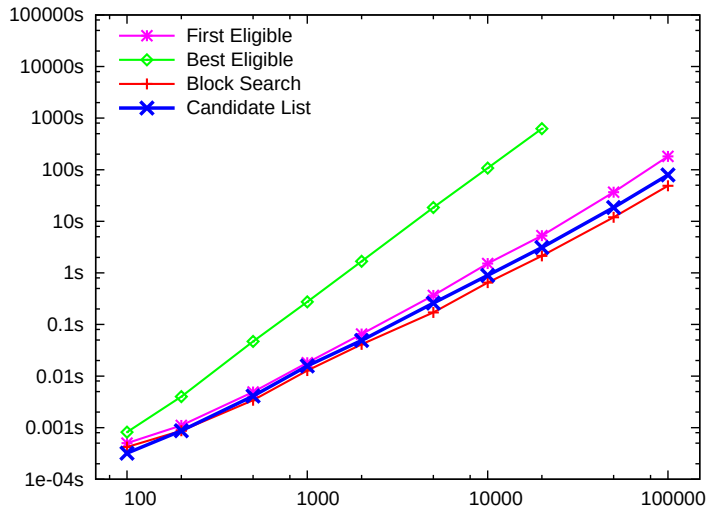
**Candidate List** módszer:

- A választható élekből bizonyos időközönként listát építünk.
- A következő pivotálások során ebből választjuk ki a legjobbat.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

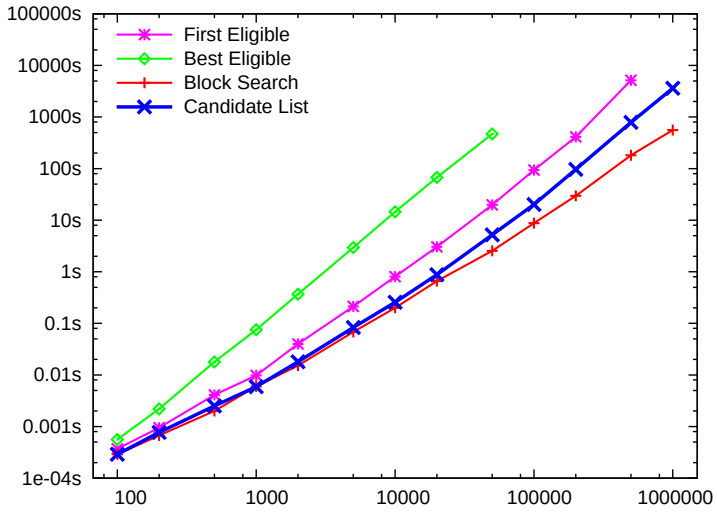



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

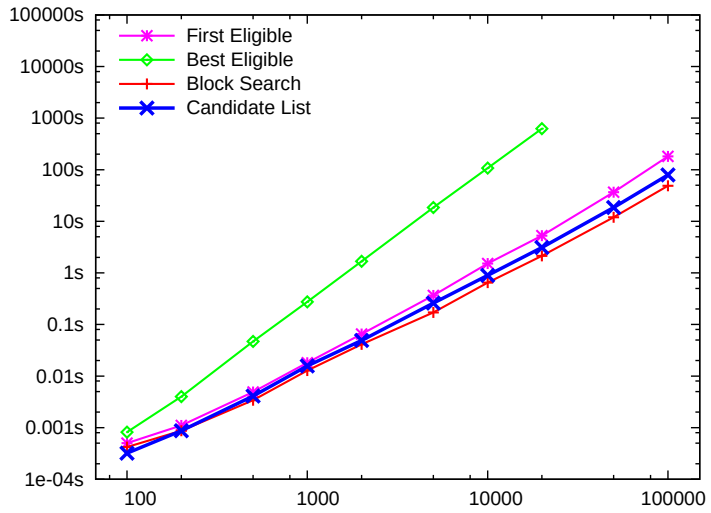
**Candidate List** módszer:

- A választható élekből bizonyos időközönként listát építünk.
- A következő pivotálások során ebből választjuk ki a legjobbat.
- Fontos paraméter a jelöltlisták mérete.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)

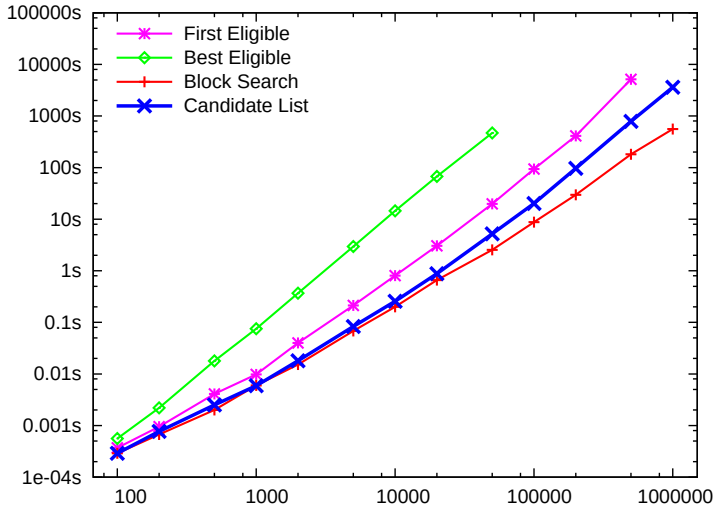



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

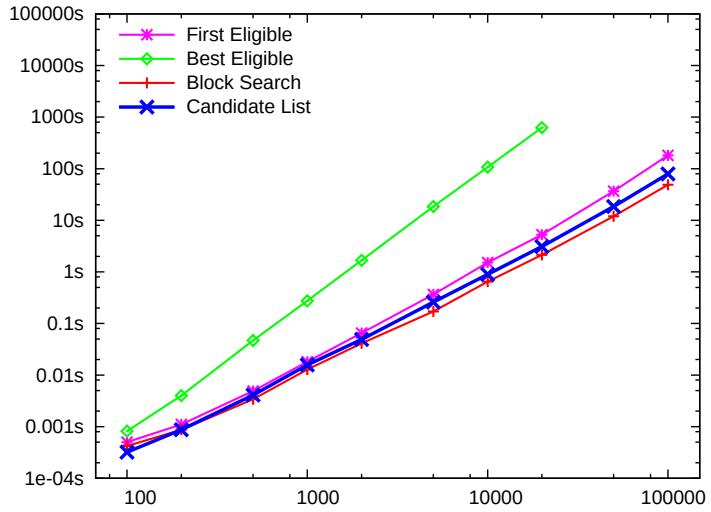
**Candidate List** módszer:

- A választható élekből bizonyos időközönként listát építünk.
- A következő pivotálások során ebből választjuk ki a legjobbat.
- Fontos paraméter a jelöltlisták mérete.
- Elég hatékony módszer, de általában lassabb, mint a **Block Search**.

# Hálózati szimplex – pivotálási szabályok




Ritka hálózatokon ($m \approx n \log_2 n$)



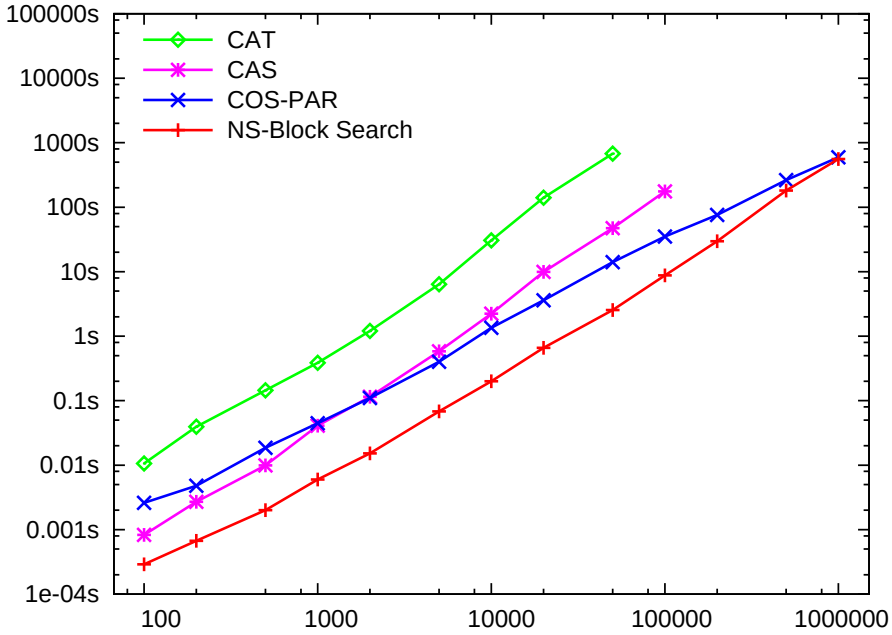

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

**Candidate List** módszer:

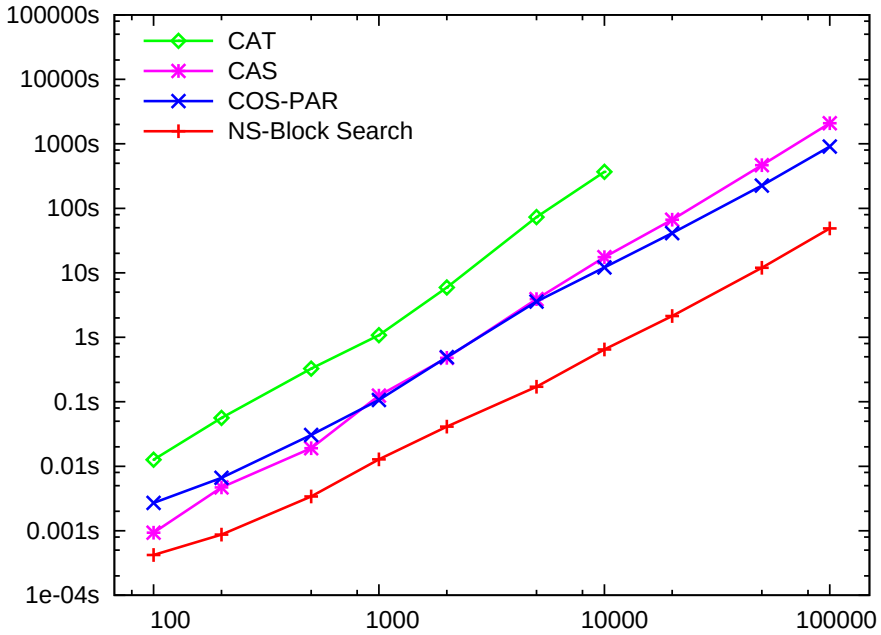
- A választható élekből bizonyos időközönként listát építünk.
- A következő pivotálások során ebből választjuk ki a legjobbat.
- Fontos paraméter a jelöltlisták mérete.
- Elég hatékony módszer, de általában lassabb, mint a **Block Search**.
- Kidolgoztuk egy valamivel hatékonyabb változatát is.

# Összehasonlítás

Az alábbi grafikonokon a 4 megközelítési mód leghatékonyabb implementációit hasonlítjuk össze.

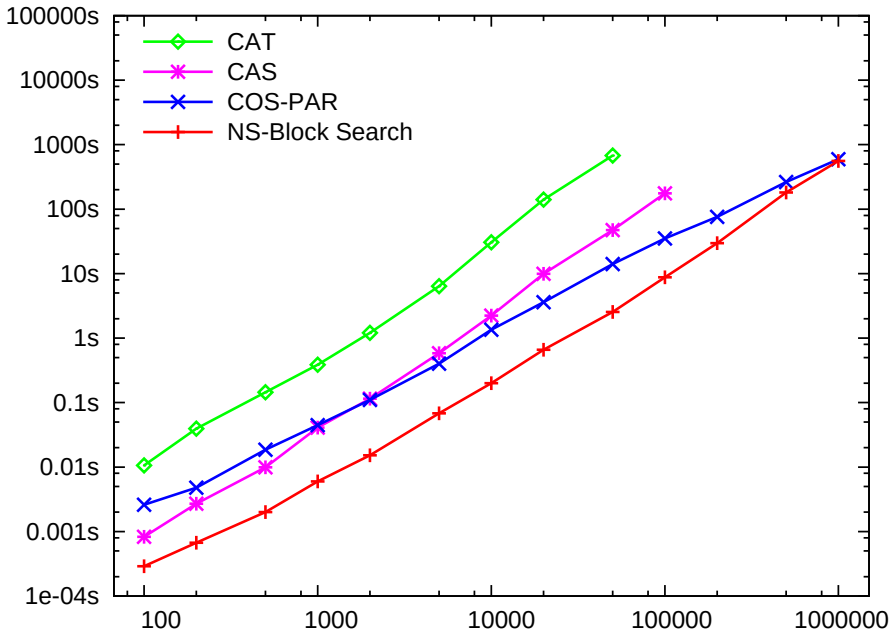



Ritka hálózatokon ($m \approx n \log_2 n$)

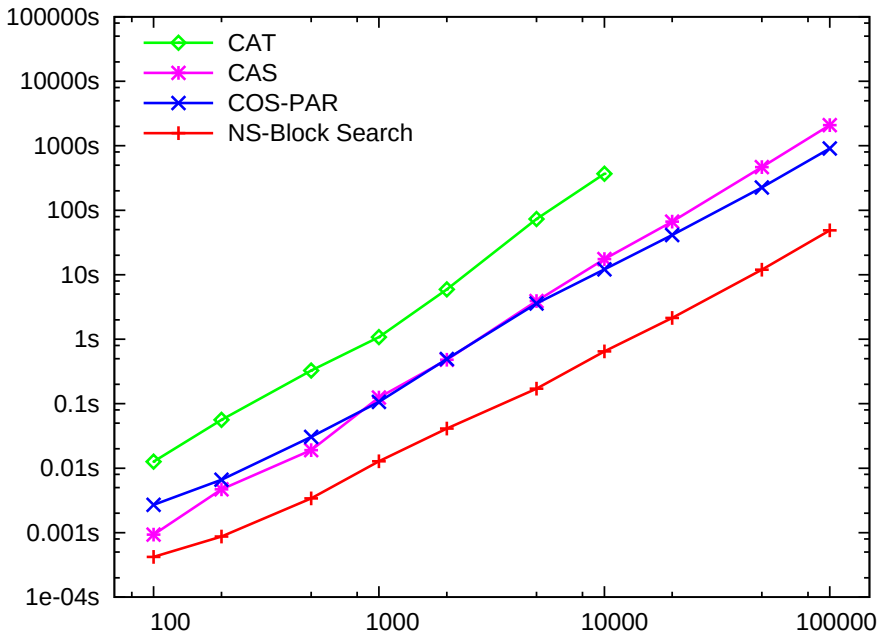



Sűrű hálózatokon ($m \approx n\sqrt{n}$)

# Összehasonlítás



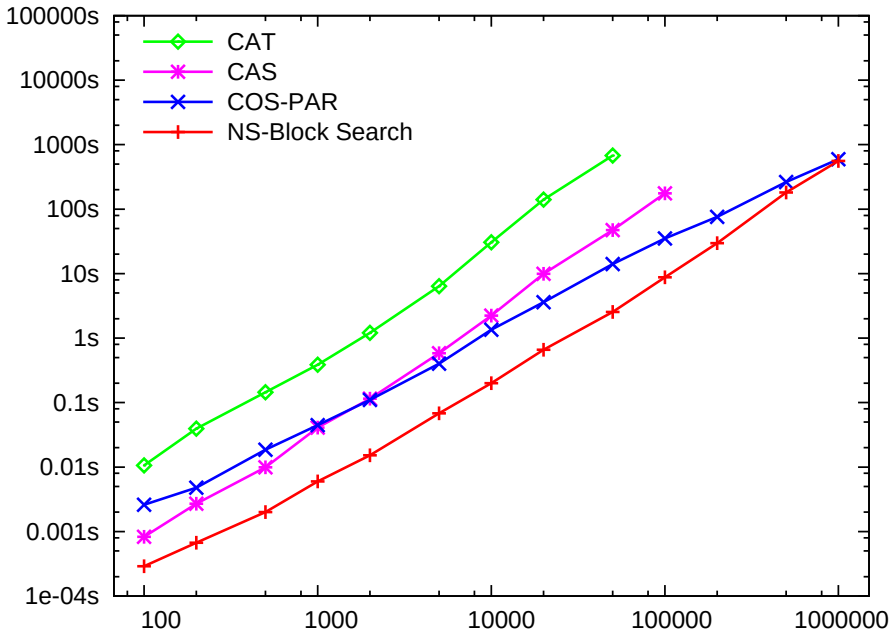

Ritka hálózatokon ($m \approx n \log_2 n$)



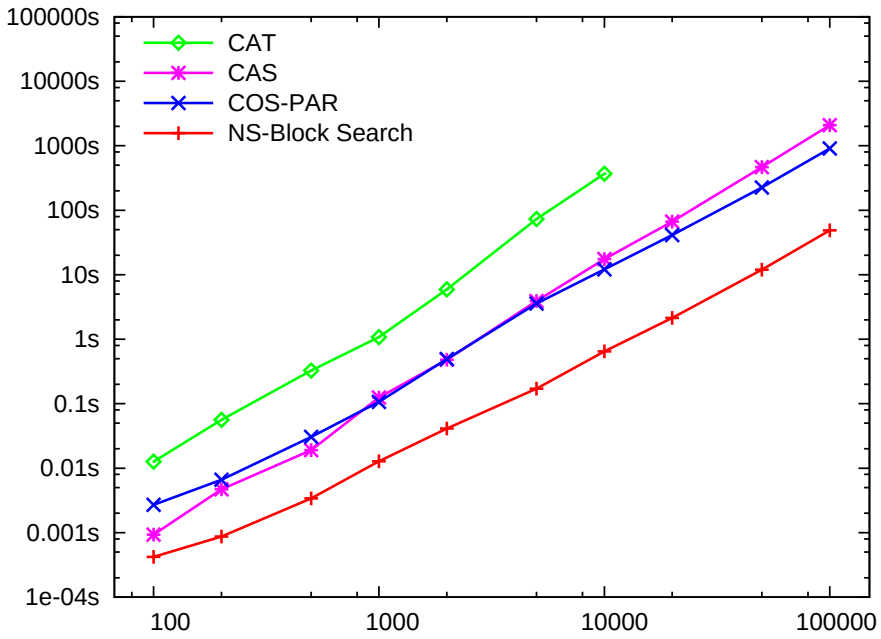

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- A **negatív kör algoritmus (CAT)** sokkal lassabb a többinél.

# Összehasonlítás



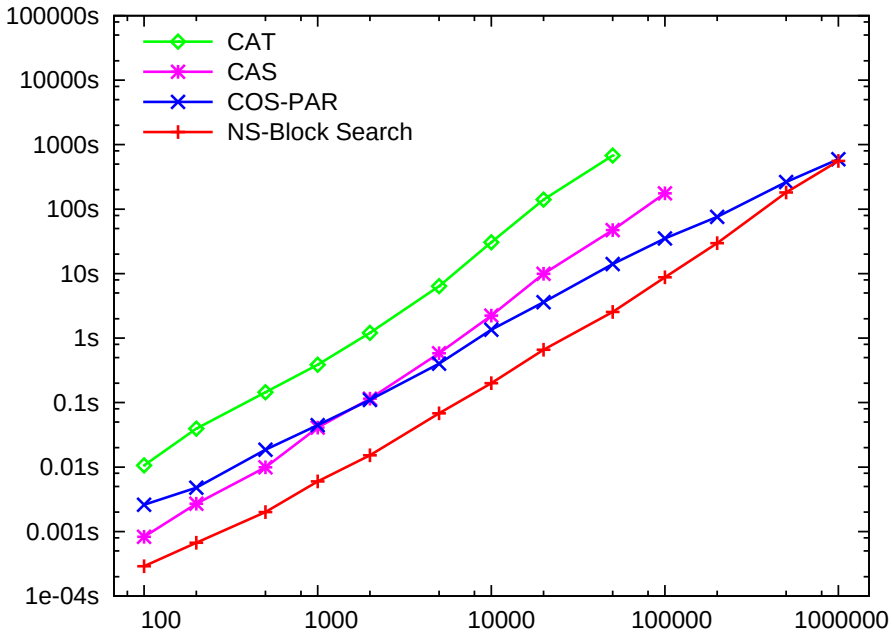

Ritka hálózatokon ($m \approx n \log_2 n$)



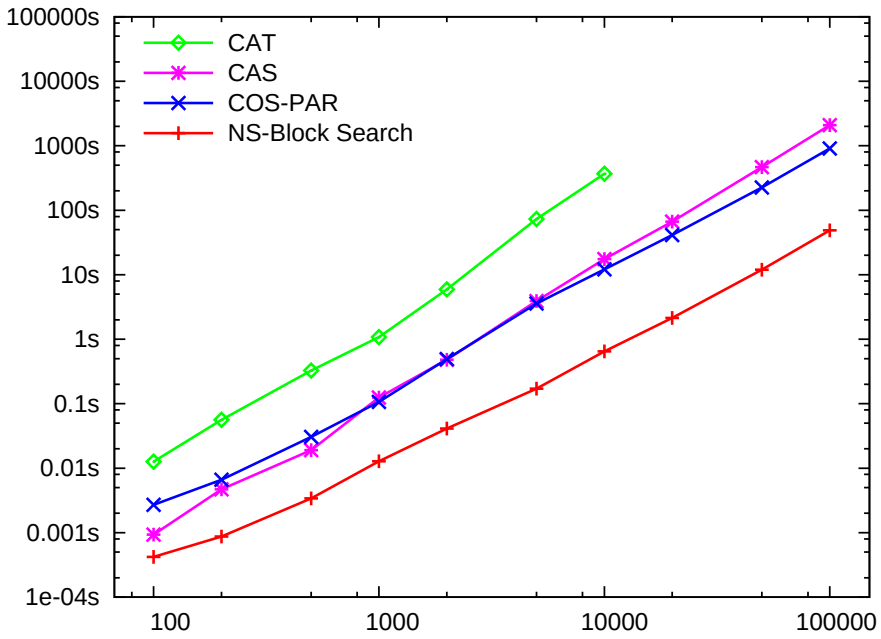

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- A **negatív kör algoritmus (CAT)** sokkal lassabb a többinél.
- A **költségskálázó módszer (COS)** aszimptotikusan gyorsabbnak tűnik a **kapacitásskálázásnál (CAS)**; nagy hálózatokon lényegesen jobbnak bizonyult.

# Összehasonlítás



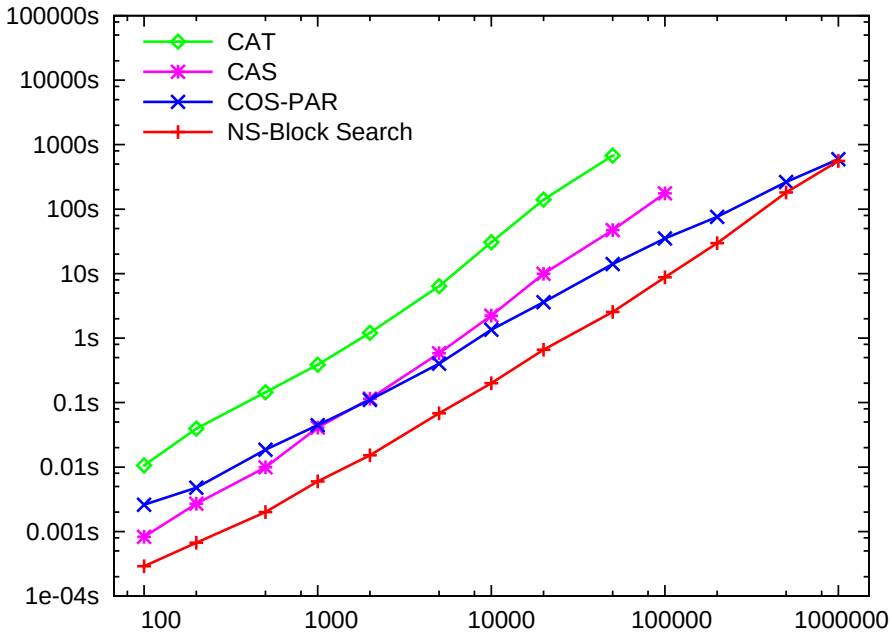

Ritka hálózatokon ($m \approx n \log_2 n$)



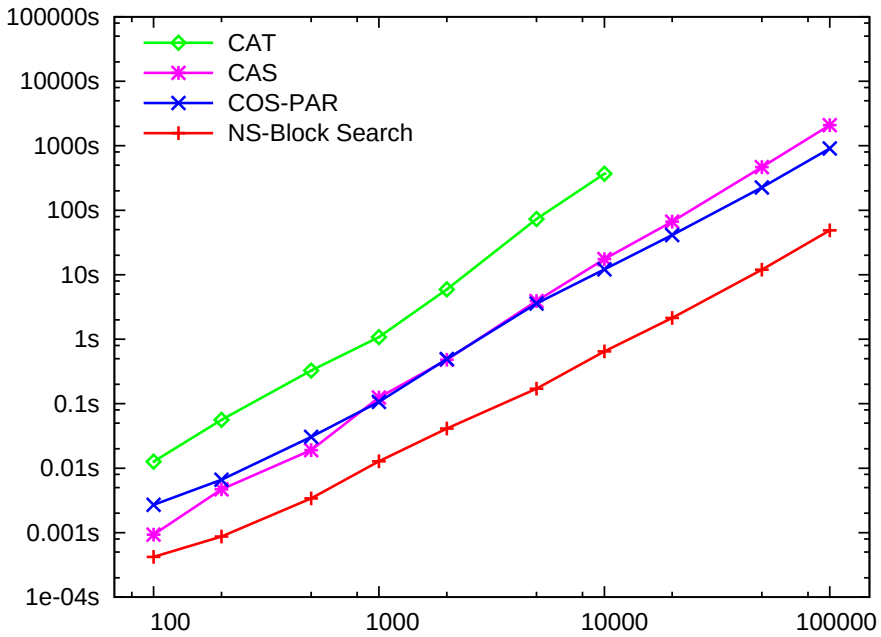

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- Sűrű gráfokon egyértelműen a **hálózati szimplex (NS)** a leghatékonyabb, kb. 20-szor gyorsabb a **költségskálázó algoritmusnál (COS)**.

# Összehasonlítás



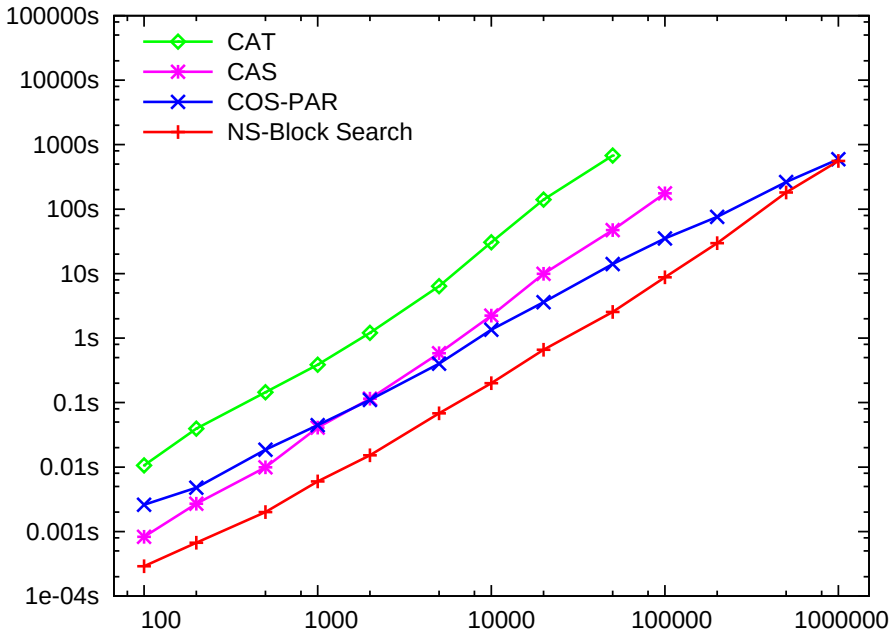

Ritka hálózatokon ($m \approx n \log_2 n$)



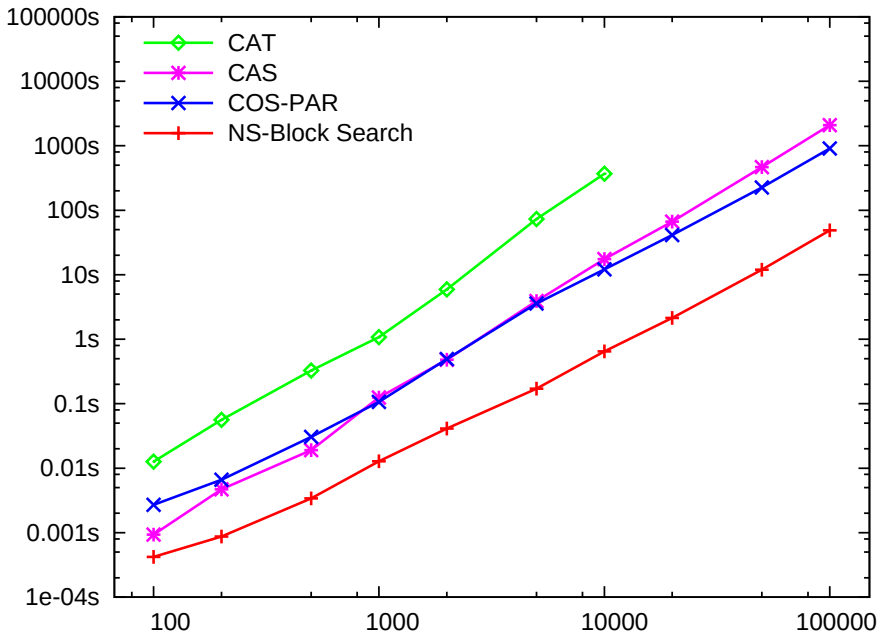

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- Sűrű gráfokon egyértelműen a **hálózati szimplex (NS)** a leghatékonyabb, kb. 20-szor gyorsabb a **költségskálázó algoritmusnál (COS)**.
- Ritka gráfokon viszont aszimptotikusan lassabb a **COS**-nál.

# Összehasonlítás



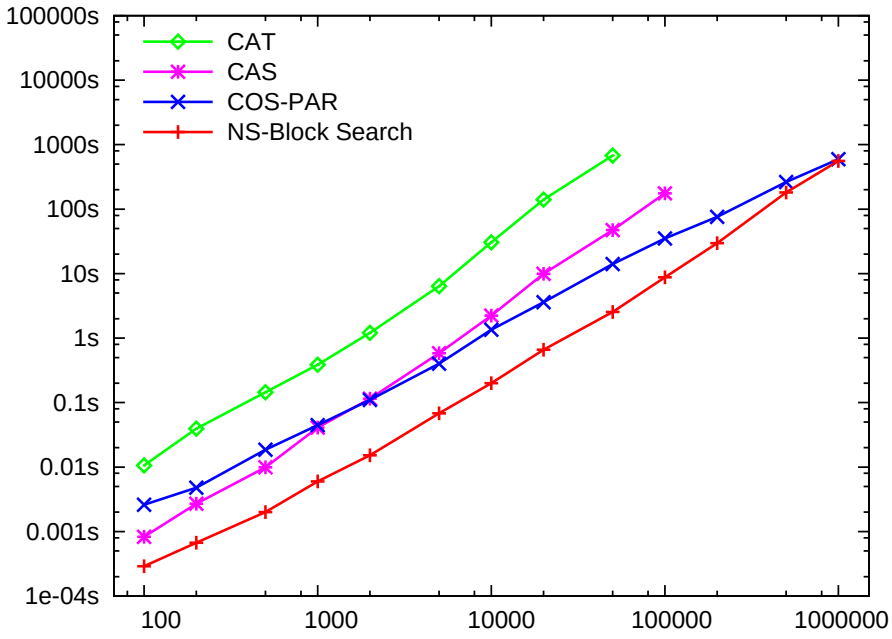

Ritka hálózatokon ($m \approx n \log_2 n$)



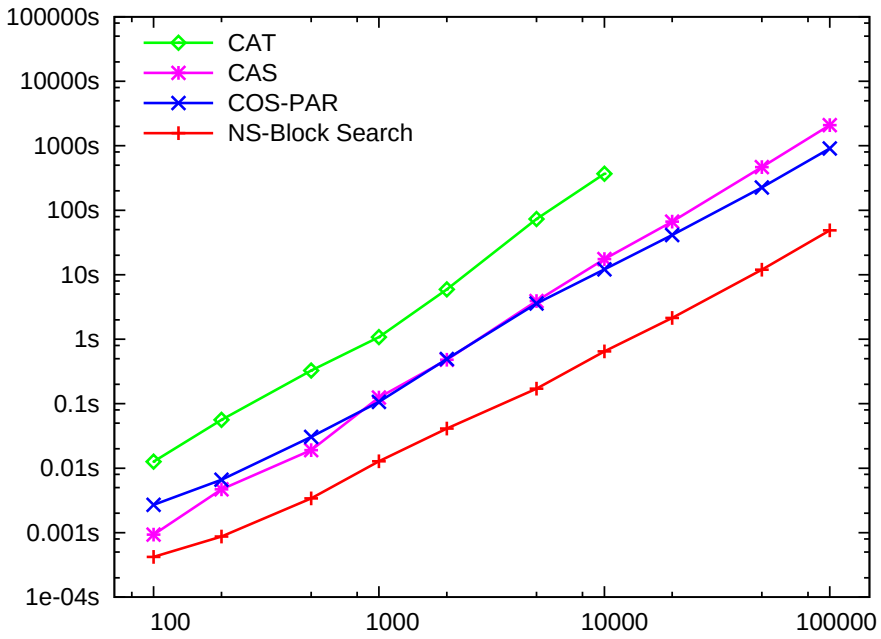

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- Sűrű gráfokon egyértelműen a **hálózati szimplex (NS)** a leghatékonyabb, kb. 20-szor gyorsabb a **költségskálázó algoritmusnál (COS)**.
- Ritka gráfokon viszont aszimptotikusan lassabb a **COS**-nál.
- A **CAS** és **COS** skálázó algoritmusok futásideje erősen függ az élek számától.

# Összehasonlítás




Ritka hálózatokon ($m \approx n \log_2 n$)



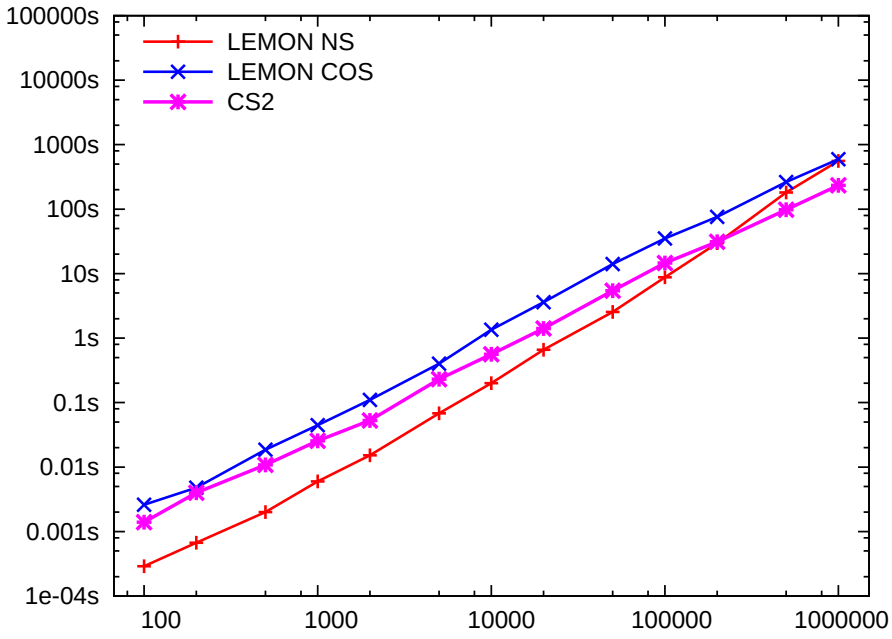

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- Sűrű gráfokon egyértelműen a **hálózati szimplex (NS)** a leghatékonyabb, kb. 20-szor gyorsabb a **költségskálázó algoritmusnál (COS)**.
- Ritka gráfokon viszont aszimptotikusan lassabb a **COS**-nál.
- A **CAS** és **COS** skálázó algoritmusok futásideje erősen függ az élek számától.
- Az **NS** algoritmus pedig a használt komplex adatszerkezet miatt sokkal erősebben függ a csúcsok számától.

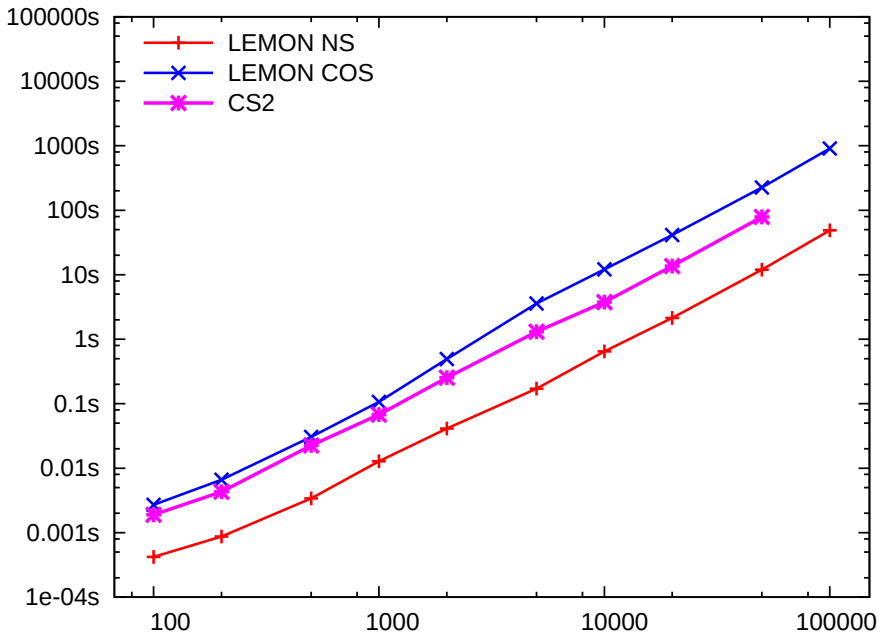
# Összehasonlítás

A továbbiakban a **LEMON** könyvtárba bekerült két leghatékonyabb implementációt (**COS** és **NS**) hasonlítjuk össze más publikus megoldóprogramokkal.

# Összehasonlítás



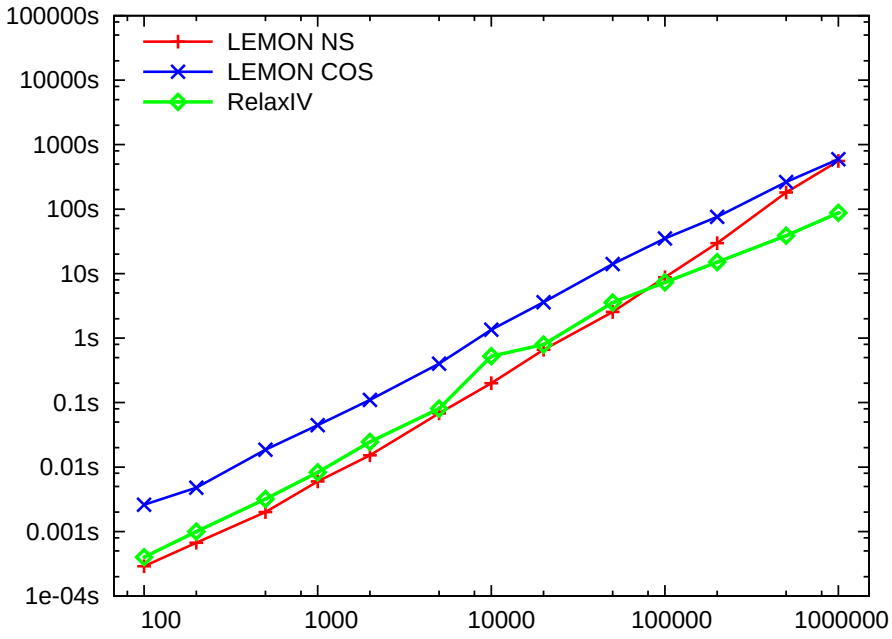

Ritka hálózatokon ($m \approx n \log_2 n$)



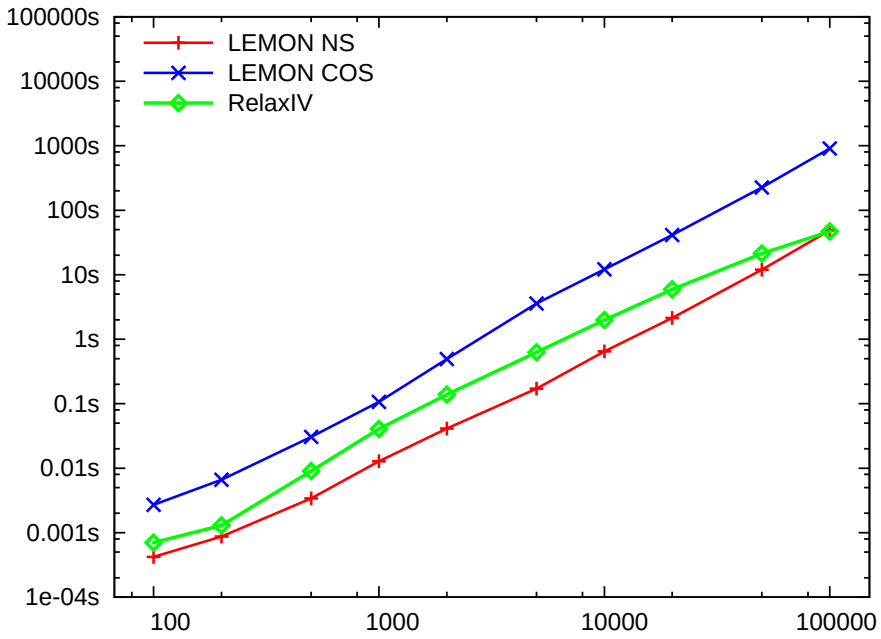

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- **CS2**: Golberg és Cherkassky CS2 3.7 programja.
- A költségskálázó algoritmus hatékony heurisztikákkal (**COS**-hoz hasonlóan).
- Nyílt forrású implementáció.

# Összehasonlítás



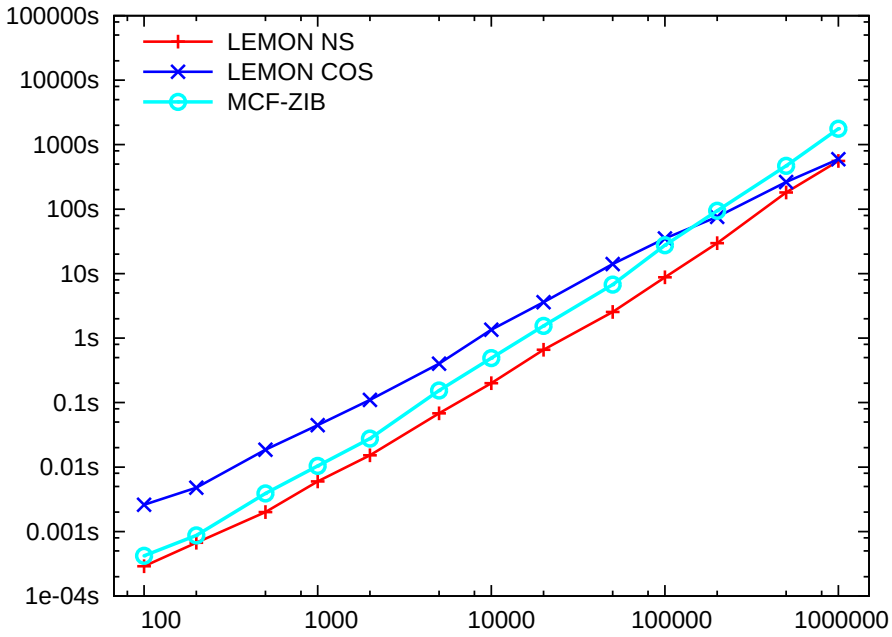

Ritka hálózatokon ($m \approx n \log_2 n$)



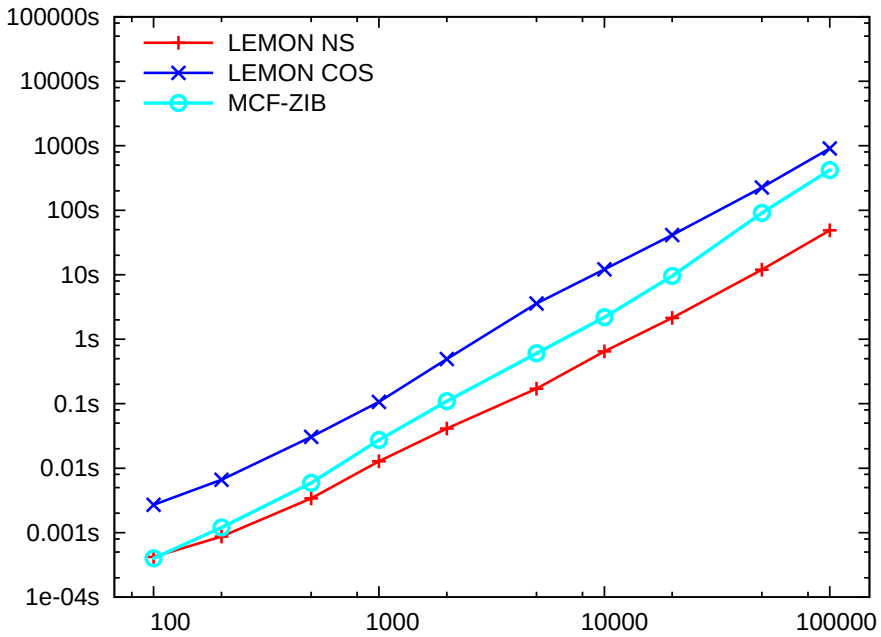

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- **RelaxIV**: Bertsekas és Tseng RelaxIV kódja.
- Relaxációs algoritmus – speciális megközelítés.
- Nyílt forrású implementáció.

# Összehasonlítás



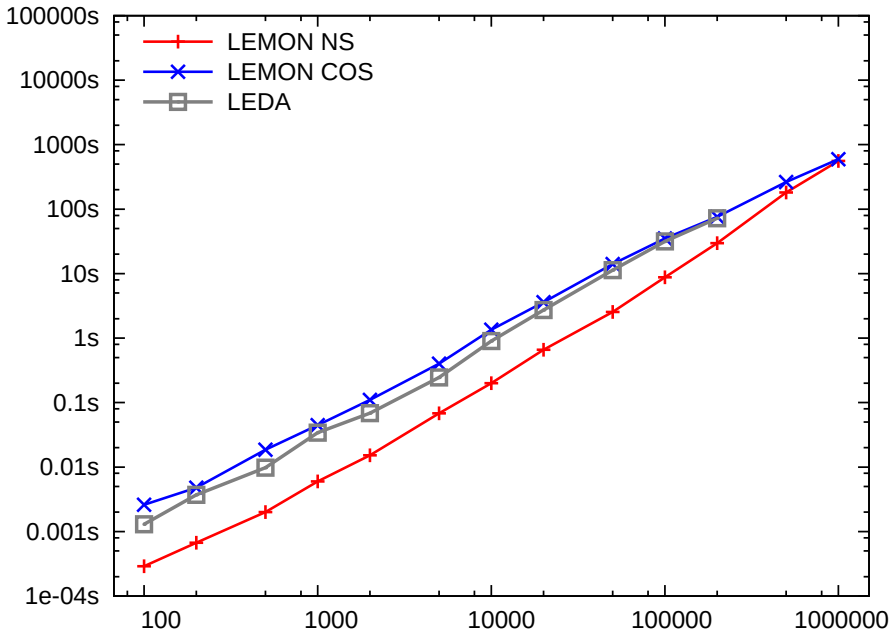

Ritka hálózatokon ($m \approx n \log_2 n$)



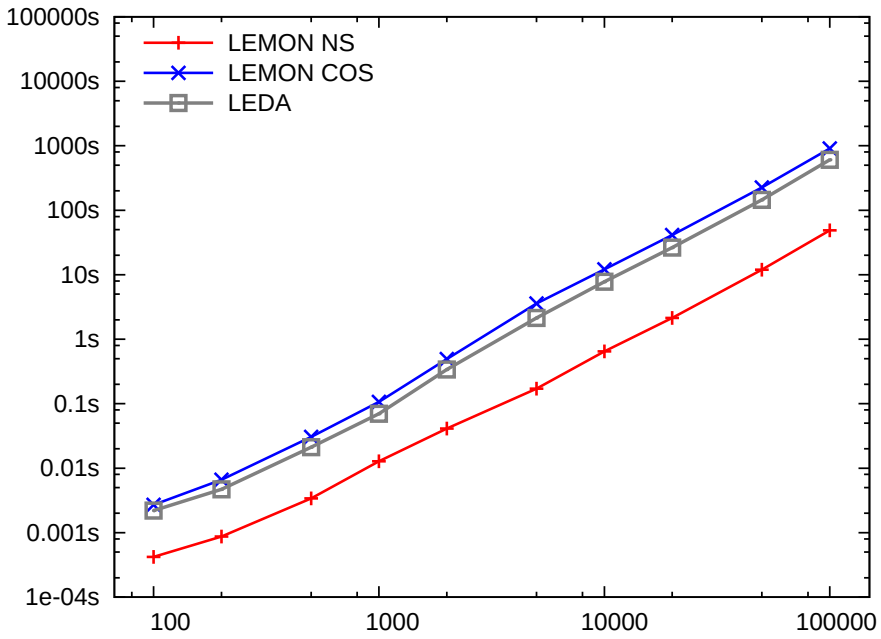

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- **MCF-ZIB**: A. Löbel (ZIB) MCF 1.1 programja.
- Hálózati szimplex algoritmus (**NS**-hez hasonlóan).
- Nyílt forrású implementáció.

# Összehasonlítás



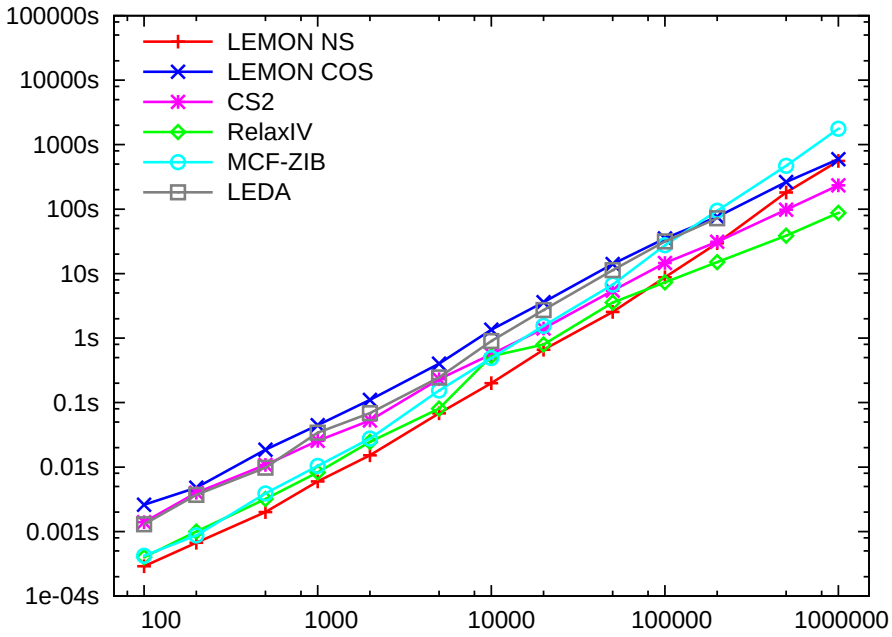

Ritka hálózatokon ($m \approx n \log_2 n$)



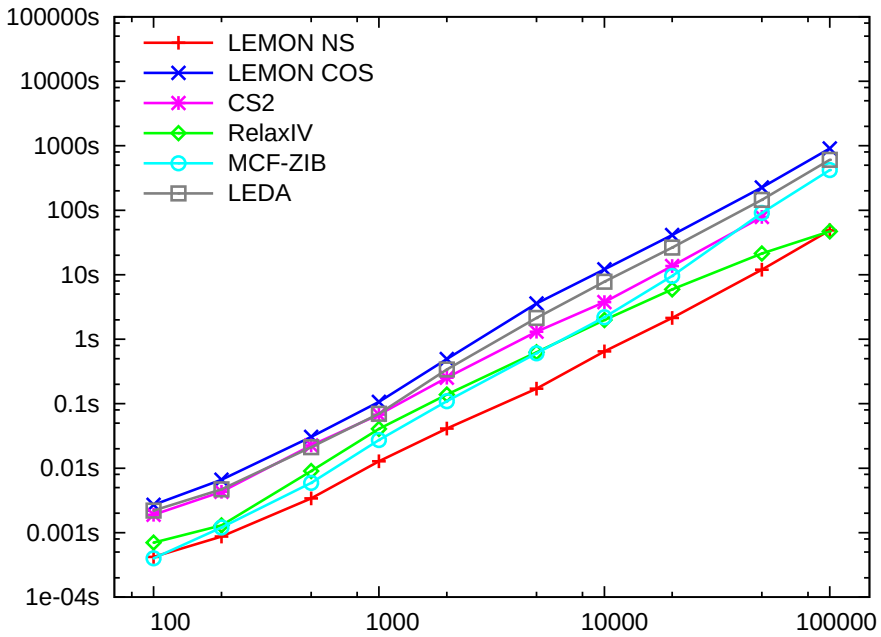

Sűrű hálózatokon ($m \approx n\sqrt{n}$)

- **LEDA**: A LEDA 5.0 programkönyvtár minimális költségű folyam eljárása.
- Feltehetőleg szintén egy költségskálázó algoritmust valósít meg.
- A **LEDA** a **LEMON**-hoz hasonló programkönyvtár, de nem ingyenes.

# Összehasonlítás




Ritka hálózatokon ($m \approx n \log_2 n$)




Sűrű hálózatokon ($m \approx n\sqrt{n}$)

Összefoglalva:

- A nagyon nagy méretű (több százezer csúcsot tartalmazó) ritka hálózatokon a **RelaxIV** és **CS2** kódok gyorsabbak az általunk adott **NS** implementációnál.
- Minden más esetben az **NS** (az egyik) leghatékonyabb.

# Összefoglalás

- Hatékonyan megvalósítottunk 9 algoritmust, illetve ezek számos változatát.
- Az egyes módszereket szisztematikusan összehasonlítottuk igen nagy méretű inputokon.

# Összefoglalás

- Hatékonyan megvalósítottunk 9 algoritmust, illetve ezek számos változatát.
- Az egyes módszereket szisztematikusan összehasonlítottuk igen nagy méretű inputokon.
- Az implementációink hatékonynak bizonyultak, más publikus megoldóprogramokkal összemérhető futási időket kaptunk.
- A hálózati szimplex implementációnk sok esetben hatékonyabb minden más vizsgált kódnál.

# Összefoglalás

- Hatékonyan megvalósítottunk 9 algoritmust, illetve ezek számos változatát.
- Az egyes módszereket szisztematikusan összehasonlítottuk igen nagy méretű inputokon.
- Az implementációink hatékonynak bizonyultak, más publikus megoldóprogramokkal összemérhető futási időket kaptunk.
- A hálózati szimplex implementációnk sok esetben hatékonyabb minden más vizsgált kódnál.
- Az implementációink bekerültek a **LEMON** nyílt forrású optimalizálási programkönyvtárba.

`http://lemon.cs.elte.hu`