A cikk megállapítja, hogy a feladat NP-teljes (nehezen megoldható) sok egyszerű gráf osztály esetében, például teljes gráfok, bipartit gráfok, csillag gráfok és sík gráfok esetén.  
A cikk pozitív eredményeket is bemutat vonalas gráfokra vonatkozóan. A kutatók képesek optimális ütemezéseket találni k robot számára, amelyek m feladatot végeznek el egy n hosszúságú vonalon, O(kmn) idő alatt.

A cikk hangsúlyozza, hogy a robotok ütemezése központilag kontrollált, ami eltér attól, hogy minden robot saját magának kellene meghatároznia a legjobb útvonalat.

**Főbb Eredmények:**

* **Általános grafikonok**: NP-nehéz.
* **Teljes grafikonok** (k ≥ 2): NP-nehéz.
* **Bipartit grafikonok** (k ≥ 2): NP-nehéz.
* **Csillag grafikonok** (és fák) (k ≥ 2): NP-nehéz.
* **Sík grafikonok**: NP-nehéz.
* **Útgrafikonok, m feladat egyenlő időtartammal, k ∈ N**: Optimális O(kmn) időben (18. tétel).
* **Útgrafikonok, k ∈ N**: k-approximation algoritmus (19. tétel)

**1 robot útgráfon:**  
**Bemenet:**  
T = (t1, t2, . . . , tm) feladatok (és hogy hol vannak)  
R robot  
P = {v1, v2, v3 … vn} gráf  
sv: a robot ahol kezd

* is: kezdő pozíció
* i1: első task
* im: utolsó task

**Algo**

Az algoritmus két különböző optimalizált ütemezést határoz meg a robot mozgásához, attól függően, hogy a robot végső pozíciója mennyire közel van az első feladathoz.

1. **Első Ütemezés (C)**:
   * Használatos, ha a robot végső pozíciója (is) és az utolsó feladat pozíciója (im) közötti távolság |is - im| kisebb vagy egyenlő, mint a kezdő pozíció (is) és az első feladat pozíciója (i1) közötti távolság |is – i1|
   * Az ütemezés sorrendje:
     + A robot a kezdőcsúcsról halad a feladatok csúcsai felé.
     + Elvégzi az összes feladatot, majd visszatér a kezdőcsúcsra a feladatok sorrendjében.
2. **Második Ütemezés**:
   * Használatos, ha a robot végső pozíciója és az utolsó feladat pozíciója közötti távolság nagyobb, mint a kezdő pozíció és az első feladat pozíciója közötti távolság.
   * Az ütemezés sorrendje:
     + A robot a kezdőcsúcsról hátrafelé indul a feladatok csúcsai felé.
     + Elvégzi az összes feladatot, majd visszatér a kezdőcsúcsra.

**Példa**

Tegyük fel, hogy van egy path graph, amely 6 csúcsból áll:

v1 -- v2 -- v3 -- v4 -- v5 -- v6

Tegyük fel, hogy a robot a v3 csúcsból indul, azaz sv=v3. Legyenek a feladatok a következőképpen elhelyezve:

* t1​ a v1​ csúcson (itt i1=1)
* t2​ a v4csúcson (itt i2=4)
* t3a v5csúcson (itt i3=5)

Most a feladatok indexei:

* is=3 (robot indulási pozíció)
* i1=1 (első feladat)
* im=5i (utolsó feladat)

Ellenőrzés

Most ellenőrizzük a távolságokat:

1. |i\_s - i\_m|:

∣3−5∣=2

1. |i\_s - i\_1|:

∣3−1∣=2

Mivel ∣is−im∣≤∣is−i1∣, az első ütemezést fogjuk használni.

Első Ütemezés (C)

Az ütemezés a következő lépésekből áll:

1. A robot elindul v3​-tól v4​-re, ahol az t2​ feladat található.
2. Majd a robot továbbhalad v5​-re, ahol az t3​ feladat található.
3. Visszatér v4​-re, majd v3​-ra.
4. Végül elindul v2​-re, ahol t1 található, majd visszatér v1​-re.

**Második Példa**

Most nézzük meg a második ütemezést, ha a robot a v1​ csúcsban indul.

Tegyük fel, hogy a robot a v1csúcsból indul, és a feladatok a következők:

* t1​ a v2​ csúcson (itt i1=2)
* t2 a v4 csúcson (itt i2=4)
* t3​ a v5​ csúcson (itt i3=5)

A robot indulási pozíciója most is=1, és az utolsó feladat pozíciója im=5.

Ellenőrzés

1. |i\_s - i\_m|:

∣1−5∣=4

1. |i\_s - i\_1|:

∣1−2∣=1

Mivel ∣is−im∣>∣is−i1∣ a második ütemezést fogjuk használni.

Második Ütemezés (C)

1. A robot elindul v1​-ről v2-re, ahol t1​ található.
2. Ezután visszatér v2​-re, majd elindul v3​-ra, ahol nem található feladat.
3. Folytatja az utat v4-re, ahol t2 található.
4. Ezután visszatér v3​-ra, majd v2​-re, és végül v1​-re, hogy elérje t1​-et.

**2 robot az útgráfon:**  
**Bemenet:**  
T = (t1, t2, . . . , tm) feladatok (és az hogy hol vannak)  
Rr és Rl robotok  
P = {v1, v2, v3 … vn} gráf  
svr és svl: ahol kezdenek a robotok  
C1​(P,T,sv) egy függvény, amely visszaadja az optimális ütemezést egyetlen robot számára, amely a sv csúcsról indul az P úton a T feladatkészlet teljesítésére.

**Feladatok Particionálása**:

* A feladatokat két halmazra osztjuk:
  + TL=(t1,t2,…,tq) a bal robot Rl számára.
  + TR=(tq+1,tq+2,…,tm) a jobb robot Rr ​ számára.
* A q értéke úgy van meghatározva, hogy minimalizálja mindkét robot befejezési idejének maximális értékét.

max(∣C1​(P1​,max(ℓ,iq​,(t1​,t2​,…,tℓ​),svL​)∣,∣C1​(Pmin(iq+1​,vR​)​,m,(tq+1​,tq+2​,…,tm​),svR​)∣ 🡪 legyen min

**k robot az útgráfon:**  
**Bemenet:**  
T = (t1, t2, . . . , tm) feladatok (és az hogy hol vannak)  
R1, R2, R3 ... Rk robotok  
P = {v1, v2, v3 … vn} gráf  
sv1, sv2, … svk: ahol kezdenek a robotok  
C1​(P,T,sv) egy függvény, amely visszaadja az optimális ütemezést egyetlen robot számára, amely a sv csúcsról indul az P úton a T feladatkészlet teljesítésére.

Készítünk egy k×m táblázatot SSS, ahol S[c,ℓ] tárolja a leggyorsabb ütemezést, amely a t1,t2,…,tℓ​ feladatok elvégzésére vonatkozik R1,R2,…,Rc robotokkal.

**Számítás Iteratív Módon**:

* Ha már kiszámoltuk S[c−1,ℓ] értékeket minden ℓ∈[1,m] esetén, akkor a következő S[c,r] értéket az alábbi képlet szerint számoljuk ki: S[c,r]=min⁡r′∈[1,r]max⁡(∣C1(P,(tr′+1,tr′+2,…,tr))∣,S[c−1,r′])
* Itt r′ azt az indexet jelöli, amely maximalizálja a fenti kifejezést.