Curs 10:

Tehnica alegerii local optimale (“greedy”)

Motivație

Problema portofoliului de acțiuni.

Intrare:

- Suma de investit (capital disponibil): C
- Set de acțiuni $A=\{a_1,a_2,\ldots,a_n\}$; fiecare acțiune este caracterizată prin
  - Cost $(c_1,c_2,\ldots,c_n)$
  - Profit $(p_1,p_2,\ldots,p_n)$

Ieșire:

- Subset de acțiuni S având proprietatea că:
  - Suma costurilor acțiunilor din S nu depășește C
  - Profitul total al acțiunilor din S este maxim

Care ar putea fi criteriul de selecție a acțiunilor?

Motivație

Problema selecției activităților.

Se consideră un set A de activități (de exemplu examene) care partajează o resursă comună (de exemplu o sală de examen). Fiecare activitate se caracterizează printr-un interval de desfășurare (ora de start și ora de final). Se pune problema selectării unui subset maxim de activități care pot fi derulate folosind o singură resursă.

Intrare:

- Momente de start: $b_1,b_2,\ldots,b_n$
- Momente de final: $e_1,e_2,\ldots,e_n$

Ieșire:

- Subset de activități S din A având proprietatea că:
  - Nu există conflicte între activități: pentru oricare două activități i și j din S, intervalele lor de derulare ($[b_i,e_i)$ si $[b_j,e_j)$ sunt disjuncte)
  - Numărul de elemente din S este maxim

Care ar putea fi criteriul de selecție a activităților?

Cuprins

- Probleme de optimizare cu restricții
- Ideea de bază a tehnicii alegerii local optimale
- Exemple
- Verificarea corectitudinii și analiza eficienței
- Aplicații

Probleme de optimizare cu restricții

Structura generală a unei probleme de optimizare cu restricții este:

Să se gasească x în X (X este spațiul de căutare sau spațiul soluțiilor) astfel încât:

(i) x satisface anumite restricții

(ii) x optimizează (minimizează sau maximizează) un criteriu de optim

Caz particular:

X este o mulțime finită – problema face parte din domeniul optimizării discrete (sau combinatoriale)

La prima vedere o astfel de problemă pare simplu de rezolvat. Totuși când spațiul de căutare are foarte multe elemente analiza exhaustivă este impracticabilă astfel că problema poate deveni dificilă

Probleme de optimizare cu restricții

Exemplu 1. Considerăm un caz particular al problemei portofoliului de acțiuni când fiecare acțiune are un cost egal cu 1

Problema este echivalentă cu cea a determinării unei submulțimi de cardinal dat și sumă maximă

Fie $A=\{a_1,\ldots,a_n\}$ și $m<n$
Să se determine o submulțime S a lui A care satisface:

(i) Numărul de elemente din S este m (restricția problemei)
(ii) Suma elementelor din S este maximă (criteriul de optim)

Obs.

1. Spațiul de căutare este X = mulțimea tuturor celor 2^n submulțimi ale lui A
2. O abordare prin tehnica forței brute bazată pe generarea tuturor submulțimilor și calculul sumei elementelor acestora ar avea o complexitate de ordinul $O(n2^n)$.

Exemplul 1

Problema submulțimii de cardinal dat și sumă maximă

Să se determine submulțimea S a mulțimii finite A care are proprietatea că:

(i) S are m<=card A elemente

(ii) Suma elementelor din S este maximă

Exemplu:

Fie A={5,1,7,5,4} si m=3.

Care este soluția? In ce ordine ar trebui selectate elementele?

Solutia este S={7,5,5} - elementele sunt selectate în ordine descrescătoare

Exemplul 1

Abordare greedy: se selectează cele mai mari m elemente din A

```
Subset(A[1..n],m)  //varianta 1
FOR i ← 1,m DO
  k ← i
  FOR j ← i+1,n DO
     IF A[k]<A[j] THEN k ← j  ENDIF
  ENDFOR
   IF k<>i THEN A[k]↔A[i] ENDIF
  S[i] ← A[i]
ENDFOR
RETURN S[1..m]
```

```
Subset(A[1..n],m)  //varianta 2
A[1..n] ← sortare_descr.(A[1..n])
FOR i ← 1,m DO
     S[i] ← A[i]
RETURN S[1..m]

// mai puțin eficientă decât varianta 1
          (daca A nu este deja sortat și
          daca m este mic)
```

Obs. Se poate demonstra că pentru această problemă tehnica “greedy” conduce la soluția optimă

Ideea de bază a tehnicii alegerii local optimale

Reformulăm problema generală de optimizare sub forma:

Fie $A=(a_1,\ldots,a_n)$ un multiset (o colecție de elemente care nu sunt neapărat distincte). Să se găsească $S=(s_1,\ldots,s_k)$, subset al lui A astfel încât S să satisfacă anumite restricții și să optimizeze un criteriu.

Ideea căutarii local optimale (căutare lacomă – greedy):

- S este construită în mod incremental pornind de la primul element
- La fiecare pas un nou element (cel ce pare a fi cel mai promițător la pasul respectiv) este selectat din A și adăugat la S.
- O dată făcută o alegere, aceasta este irevocabilă (nu se poate reveni și înlocui o componentă cu o altă valoare)

Ideea de bază a tehnicii

Structura generală a unui algoritm de tip “greedy”:

```
Greedy(A)
  S ← Ø
  WHILE “S nu este finalizată” AND “există elemente neselectate în A”  DO
        a ←  alege (A)  // “alege cel mai bun element a,  disponibil in A”
        IF    S U {a} satisface restricțiile problemei
              THEN   S ← S U {a}     // “adauga a la S”
        ENDIF
  ENDWHILE
  RETURN S
```

Obs. Etapele principale în construirea soluției: inițializare, selecție, extindere

Ideea de baza a tehnicii

Cel mai important element al algoritmilor de tip “greedy” îl reprezintă selecția elementului care se adaugă la fiecare pas.

Selecția se realizează pe baza unui criteriu care este stabilit în funcție de specificul problemei de rezolvat

Criteriul de selecție se bazează de regulă pe euristici

(euristica = tehnică bazată mai mult pe experiență și intuiție decât pe o analiză profundă/formală a problemei

= arta de a descoperi cunoștințe noi (cf. DEX))

Exemplul 2

Problema monedelor

Presupunem că avem la dispoziție un număr nelimitat de monede având valorile: $\{v_1,v_2,\ldots,v_n\}$. Să se găseasca o modalitate de a acoperi o sumă C astfel încât numărul de monede folosite să fie cât mai mic.

Fie s_i numărul de monede de valoare v_i selectate

Restricție: $s_1v_1+\ldots+s_nv_n=C$
Criteriu de optim: numărul de monede selectate $(s_1+s_2+\ldots+s_n)$ este cât mai mic

Abordare greedy: se pornește de la moneda cu cea mai mare valoare și se acoperă cât mai mult posibil din sumă, pentru restul sumei se încearcă utilizarea următoarei monede (în ordine descrescătoare a valorii) s.a.m.d.

Exemplul 2

```
monede(v[1..n],C)
v[1..n] ← sortare_descrescatoare(v[1..n])
FOR i ← 1,n DO S[i]:=0 ENDFOR
i ← 1
WHILE C>0 and i<=n DO
   S[i] ← C DIV v[i]       // numărul maxim de monede de valoare v[i]
   C ← C MOD v[i]          // restul rămas de acoperit
   i ← i+1
ENDWHILE
IF C=0 THEN  RETURN S[1..n]
           ELSE  RETURN “problema nu are soluție”
ENDIF
```

Exemple

Observații:

1. Uneori problema nu are soluție:

Exemplu: V=(20,10,5) and C=17

Totuși, dacă sunt disponibile monede de valoare 1, atunci problema are întotdeauna o soluție

2. Uneori tehnica “greedy” nu conduce la o soluție optimă

Exemplu: V=(25,20,10,5,1), C=40

Abordare “greedy”: (1,0,1,1,0)

Soluția optimă: (0,2,0,0,0)

O condiție suficientă pentru optimalitate: $v_1>v_2>\ldots>v_n = 1$ si $v_{i-1}=d_{i-1}v_i$ (i=2 ..n)

Caracteristici ale tehnicii

Tehnica alegerii local optimale conduce la :

- algoritmi simpli și intuitivi
- algoritmi eficienți

dar

- nu conduce întotdeauna la o soluție optimă (alegerea local optimală poate avea efecte negative la nivel global; ceea ce pare promițător la un anumit pas se poate dovedi a nu fi optim la nivel global)
- uneori soluțiile obținute de tehnica greedy sunt sub-optimale adică valoarea criteriului este "suficient" de apropiată de cea optimă (un algoritm care conduce la soluții sub-optimale este denumit algoritm de aproximare a soluției)

Intrucât tehnica "greedy" nu garantează optimalitatea soluției, pentru fiecare caz în parte trebuie verificat dacă se obține soluția optimă sau nu

Verificarea corectitudinii

Multe dintre problemele pentru care soluția greedy este optimă sunt caracterizate prin următoarele proprietăți:

- Proprietatea substructurii optime
  - Orice soluție optimă a problemei inițiale conține o soluție optimă a unui subprobleme (problemă de același tip dar de dimensiune mai mică)
- Proprietatea alegerii greedy
  - Componentele unei soluții optime au fost alese folosind criteriul greedy de selecție sau pot fi înlocuite cu elemente alese folosind acest citeriu fără a altera proprietatea de optimalitate

Proprietatea de substructură optimă

Când se poate spune despre o problemă că are proprietatea de substructură optimă?

Atunci când pentru o soluție optimă $S=(s_1,\ldots,s_k)$ a problemei de dimensiune n, subsetul $S_{(2)}=(s_2,\ldots,s_k)$ este o soluție optimă a unei subprobleme de dimensiune (n-1).

Cum se poate verifica dacă o problemă are această proprietate ?

Folosind demonstrarea prin reducere la absurd

Proprietatea de alegere “greedy”

Când se poate spune că o problemă are proprietatea de alegere “greedy”?

Atunci când soluția optimă a problemei fie este construită printr-o strategie “greedy” fie poate fi transformată într-o altă soluție optimă construită pe baza strategiei “greedy”

Cum se poate verifica dacă o problemă posedă sau nu această proprietate?

Se demonstrează că înlocuind primul element al unei soluții optime cu un element selectat prin tehnica “greedy”, soluția rămâne optimă. Apoi se demonstrează același lucru pentru celelalte componente fie folosind metoda inducției matematice fie folosind proprietatea de substructură optimă

Verificarea corectitudinii

Exemplu: (submulțime de cardinal fixat și sumă maximă)

Fie $A=(a_1>=a_2>=\ldots>=a_n)$. Soluția greedy este $(a_1,\ldots,a_m)$.

Fie $O=(o_1,\ldots,o_m)$ o soluție optimă (presupunem că elementele lui O sunt ordonate descrescător: $o_1>=\ldots>=o_m$)

a) Proprietatea de alegere "greedy". Presupunem că $o_1!=a_1$. Aceasta înseamnă că $o_1<a_1$ Atunci $O'=(a_1,o_2,\ldots,o_m)$ are proprietatea:

$$a_1+o_2+\ldots+o_m> o_1+o_2+\ldots+o_m$$

Adică O' este mai bună decât O. Acest lucru este însă în contradicție cu faptul că O este optimă. Deci o_1 trebuie să fie egal cu a_1

Verificarea corectitudinii

Exemplu: (submulțime de cardinal fixat și sumă maximă)

Fie $A=(a_1>=a_2>=\ldots>=a_n)$. Soluția greedy este $(a_1,\ldots,a_m)$.

Fie $O=(o_1,\ldots,o_m)$ o soluție optimă (presupunem că elementele lui O sunt ordonate descrescător: $o_1>=\ldots>=o_m$)

b) Proprietatea de substructură optimă. Se demonstrează prin reducere la absurd. Presupunem ca $(o_2,\ldots,o_m)$ nu este soluție optimă pentru subproblema corespunzătoare lui $A_{(2)}=(a_2,\ldots,a_n)$.

Considerăm că $O'_{(2)}= (o'_2,\ldots,o'_m)$ este soluție optimă pentru această subproblemă. Atunci $O'=(a_1, o'_2,\ldots,o'_m)$ este o soluție mai bună decât O. Contradicție…deci problema are proprietatea de substructură optimă. Folosind această proprietate se revine la pct (a) și se arată că proprietatea de alegere greedy este satisfăcută pentru fiecare componentă.

Verificarea corectitudinii

Exemplu: problema monedelor

Fie $V=(v_1>v_2>\ldots>v_n=1)$ valorile monedelor și $v_{i-1}=d_{i-1}\, v_i$. Soluția greedy, $(g_1,\ldots,g_m)$, este caracterizată prin $g_1 = C \text{ DIV } v_1$

Fie $O=(o_1,\ldots,o_m)$ o soluție optimă.

a) Proprietatea de alegere "greedy". Presupunem că $o_1<g_1$. Atunci suma $C'=(C \text{ DIV } v_1 - o_1)v_1$ ar fi acoperită prin monede de valoare mai mică. Datorită proprietății satisfăcute de valorile monedelor rezultă că prin înlocuirea lui o_1 cu g_1 se obține o soluție mai bună (aceeași sumă poate fi acoperită cu mai puține monede de valoare mai mare). Deci problema are proprietatea de alegere "greedy".

b) Proprietatea de substructură optimă. Usor de demonstrat.

Analiza eficienței

Algoritmii de tip greedy sunt eficienți

Operația dominantă este cea de selecție a elementelor (în cazul în care e necesară sortarea elementelor mulțimii A, atunci operația de sortare este cea mai costisitoare)

Deci ordinul de complexitate al algoritmilor de tip “greedy” este

$O(n^2)$ sau $O(n\lg n)$ sau $O(n)$

(în funcție de natura elementelor din A și algoritmul de sortare utilizat)

Aplicații

Problema rucsacului (knapsack problem)

Considerăm un set de (n) obiecte și un rucsac de capacitate dată (C). Fiecare obiect este caracterizat prin dimensiune (d) și prin valoare sau profit (p). Se cere selecția unui subset de obiecte astfel încât suma dimensiunilor lor să nu depășească capacitatea rucsacului iar suma valorilor să fie maximă.

Variante:

(i) Varianta continuă (fracționară): pot fi selectate atât obiecte întregi cât și fracțiuni ale obiectelor. Soluția va fi constituită din valori aparținând intervalului [0,1].

(ii) Varianta discretă (0-1): obiectele nu pot fi fragmentate, ele putând fi incluse în rucsac doar în întregime. Soluția va fi constituită din valori aparținând mulțimii {0,1}.

Problema rucsacului

Motivație

Numeroase probleme din practică sunt similare problemei rucsacului

Exemplu. Construirea portofoliului financiar: se consideră un set de "operațiuni financiare"/acțiuni, fiecare fiind caracterizată printr-un cost și un profit; se dorește selecția acțiunilor al căror cost total nu depășește suma disponibilă pentru investiții și pentru care profitul este maxim

Exemplu. Problema rucsacului (varianta discretă) are aplicații în criptografie (a stat la baza dezvoltării unui algoritm de criptare cu chei publice – la ora actuală nu mai este folosit nefiind suficient de sigur).

Problema rucsacului

Exemplu:

Val(p)	Dim(d)
6	2
5	1
12	3

C=5

Criteriu de selecție:

In ordinea crescătoare a dimensiunii (selectează cât mai multe obiecte):

5+6+12*2/3=11+8=19

Problema rucsacului

Exemplu:

Val(p)	Dim(d)
6	2
5	1
12	3

C=5

Criteriu de selecție:

In ordinea crescătoare a dimensiunii (selectează cele mai mici obiecte -> cât mai multe obiecte):

5+6+12*2/3=11+8=19

In ordinea descrescătoare a valorii (selectează cele mai valoroase obiecte -> valoare cât mai mare):

12+6=18

Problema rucsacului

Exemplu:

Val(p)	Dim(d)	Profit relativ (Val/Dim)
6	2	3
5	1	5
12	3	4

C=5

Criteriu de selecție:

In ordinea crescătoare a dimensiunii (selectează cât mai multe obiecte): 5+6+12*2/3=11+8=19

In ordinea descrescătoare a valorii (selectează cele mai valoroase obiecte): 12+6=18

In ordinea descrescătoare a profitului relativ (selectează obiectele mici și valoroase):

5+12+6*1/2=17+3=20

Problema rucsacului

```
Knapsack(d[1..n],p[1..n])
 “sorteaza d și p descrescător după valoarea profitului relativ”

  FOR i ← 1,n DO S[i] ← 0 ENDFOR
  i ← 1
  WHILE C>0 AND i<=n DO
       IF C>=d[i] THEN   S[i] ← 1
                         C ← C-d[i]
                ELSE     S[i] ← C/d[i]
                         C ← 0
       ENDIF
       i ← i+1
  ENDWHILE
  RETURN S[1..n]
```

Problema rucsacului

Verificarea corectitudinii:

In cazul variantei continue (fracționare) a problemei, tehnica greedy conduce la soluția optimă

Obs:

- O soluție greedy satisface: S=(1,1,…,1,f,0,..,0)
- $s_1d_1+\ldots+s_nd_n=C$ (restricția poate fi întotdeauna satisfacută cu egalitate)
- Obiectele sunt sortate descrescător după valoarea profitului relativ:

$p_1/d_1>p_2/d_2>\ldots>p_n/d_n$

Dem.

Fie $O=(o_1,o_2,\ldots,o_n)$ o soluție optimă. Demonstrăm prin reducere la absurd că este o soluție greedy. Presupunem că O nu e soluție greedy și considerăm o soluție greedy $O'=(o'_1,o'_2,\ldots,o'_n)$

Problema rucsacului

Fie $B_+ = \{i | o'_i >= o_i\}$ si $B\_ = \{j | o'_j < o_j\}$, k – cel mai mic indice pentru care $o'_k < o_k$.

Datorită structurii unei soluții greedy rezultă că orice indice i din B_+ este mai mic decât orice indice j din B_.

Pe de altă parte ambele soluții trebuie să satisfacă restricția, adică:

$$o_1 d_1 + \dots o_n d_n = o'_1 d_1 + \dots + o'_n d_n$$

$$\sum_{i \in B+} (o'_i - o_i) d_i = \sum_{j \in B\_} (o_j - o'_j) d_j$$

$$P' - P = \sum_{i=1}^{n} (o'_i - o_i) p_i = \sum_{i \in B+} (o'_i - o_i) d_i \frac{p_i}{d_i} - \sum_{j \in B\_} (o_j - o'_j) d_j \frac{p_j}{d_j}$$

$$P' - P \geq \frac{p_k}{d_k} \sum_{i \in B+} (o'_i - o_i) d_i - \frac{p_k}{d_k} \sum_{j \in B\_} (o_j - o'_j) d_j = 0$$

Deci soluția greedy este cel puțin la fel de bună ca O (care este o soluție optimă). Proprietatea de substructură optimă este ușor de demonstrat prin reducere la absurd.

Problema rucsacului

Obs: rezultatul nu este valabil în cazul variantei discrete a problemei

Contraexemplu:

Obiect	Profit	Dim	Profit/Dim
O1	10	6	5/3
O2	7	5	7/5
O3	6	4	3/2
O4	2	1	2

Capacitate (C): 9

(i) Crescător după dimensiune: o3,o4 (valoare totala: 8)
(ii) Descrescător după profit: o1,o4 (valoare totala: 12)
(iii) Descrescător după profit relativ: o4, o1 (valoare totala: 12)

Soluție optimă: o2,o3 (valoare totala: 13).

Rezolvare: tehnica programării dinamice (semestrul 2)

Aplicații

Problema selecției activităților

Fie $A=\{a_1,\ldots,a_n\}$ un set de activități ce partajează aceeași resursă. Fiecare activitate a_i necesită un interval de timp $[b_i,e_i)$ pentru a fi executată. Două activități sunt considerate compatibile dacă intervalele lor de execuție sunt disjuncte și incompatibile în caz contrar.

Problema cere să se selecteze cât mai multe activități compatibile.

Exemplu:

A1: [0,6)

A2: [1,5)

A3: [4,6)

A4: [5,8)

Problema selecției activităților

Fie $A=\{a_1,\ldots,a_n\}$ un set de activități ce partajează aceeași resursă. Fiecare activitate a_i necesită un interval de timp $[b_i,e_i)$ pentru a fi executată. Două activități sunt considerate compatibile dacă intervalele lor de execuție sunt disjuncte și incompatibile în caz contrar.

Problema cere să se selecteze cât mai multe activități compatibile.

Exemplu: Există mai multe criterii de selecție a activităților

a1: [0,6)
a2: [1,5)
a3: [4,6)
a4: [5,8)

a) In ordine crescătoare a momentului de început:

a1

b) In ordinea crescătoare a duratei:

a3

c) In ordinea crescătoare a momentului de finalizare:

a2, a4

Problema selecției activităților

```
// Presupunem că fiecare element a[i] conține două câmpuri
//    a[i].b    -  moment de început (begin)
//    a[i].e     -  moment de finalizare (end)

Selectie_activitati(a[1..n])
a[1..n] ← sortare crescatoare dupa valoarea lui e (a[1..n])
s[1] ← 1  // s va contine indicii elementelor selectate din a
k ← 1
FOR i ← 2,n DO
  IF a[i].b>=a[s[k]].e THEN
                    k ← k+1
                    s[k] ← i
  ENDIF
ENDFOR
RETURN s[1..k]
```

Problema selecției activităților

Verificare corectitudine. Pp că setul de activități este ordonat crescător după momentul de finalizare ($a_1.e< a_2.e<\ldots<a_k.e$).

Proprietatea de alegere “”greedy”: Fie $(o_1,o_2,\ldots,o_k)=(a_{i1},a_{i2},\ldots,a_{ik})$ o soluție optimă (pp ca activitățile selectate sunt specificate în ordinea crescatoare a momentului de finalizare: $i_1<i_2<\ldots<i_k$).

In acest caz activitatea a_{i1} poate fi înlocuită cu a_1 (activitatea care se termină cel mai repede) fără a altera restricția problemei (activitățile selectate sunt toate compatibile) și pastrând același număr (maxim) de activități selectate

Problema selecției activităților

Verificare corectitudine. Pp că setul de activități este ordonat crescător după momentul de finalizare ($a_1.e< a_2.e<\ldots<a_k.e$).

Proprietatea de substructură optimă. Considerăm o soluție optimă: $(a_1,o_2,\ldots,o_k)$ (obs: din propr. de alegere “greedy” rezultă că putem considera $o_1=a_1$).

Pp că $(o_2,\ldots,o_k)$ nu e soluție optimă pt subproblema selecției din $\{a_2,a_3,\ldots,a_n\}$.

Rezultă că există $o'=(o'_2,\ldots,o'_{k'})$ altă soluție cu $k'>k$ și astfel încât o' e compatibilă cu a_1. Acest lucru ar conduce la o soluție $(a_1, o'_2,\ldots,o'_{k'})$ mai bună decât $(a_1,o_2,\ldots,o_k)$. Contradicție.

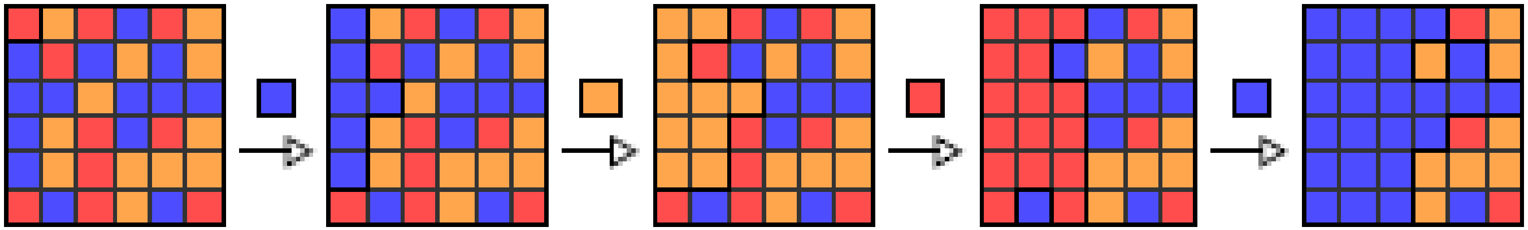
Cursul următor …

… alte tehnici de proiectare a algoritmilor

… și recapitulare

FloodIt Game [2006, LabPixies->Google]

- Se consideră o grilă nxn conținând celule colorate inițial aleator
- Celula din colțul stânga sus este considerată celulă de start
- Toate celulele care au aceeași culoare cu celula de start și care pot fi accesate prin deplasare pe orizontală sau verticală (dar nu diagonală) sunt considerate conectate cu celula de start
- Se pune problema schimbării succesive a culorii celulei de start (și a tuturor celor conectate cu ea) astfel încât grila să fie acoperită complet cu aceeași culoare în cât mai puține etape

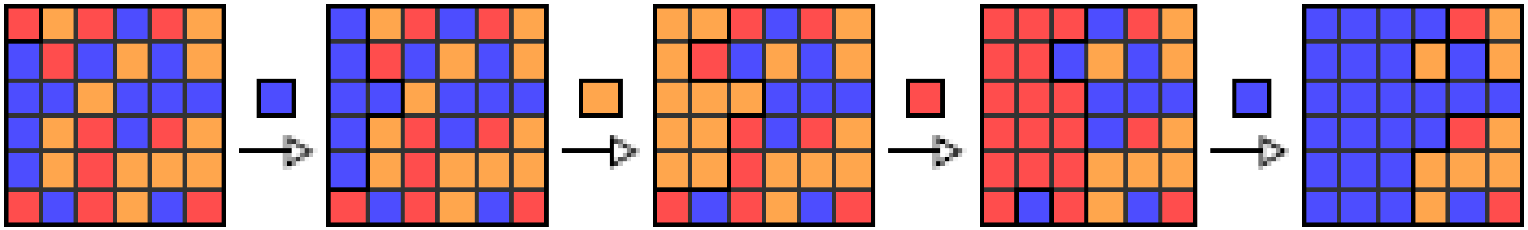


(R. Clifford, The Complexity of Flood Filling Games, 2011)

Exemplu joc online: http://www.kongregate.com/games/xeflor/flood-it, *http:* unixpapa.com/floodit/

FloodIt Game [2006, LabPixies->Google]

- Se consideră o grilă nxn conținând celule colorate inițial aleator
- Celula din colțul stânga sus este considerată celulă de start
- Toate celulele care au aceeași culoare cu celula de start și care pot fi accesate prin deplasare pe orizontală sau verticală (dar nu diagonală) sunt considerate conectate cu celula de start
- Se pune problema schimbării succesive a culorii celulei de start (și a tuturor celor conectate cu ea) astfel încât grila să fie acoperită complet cu aceeași culoare în cât mai puține etape



(R. Clifford, The Complexity of Flood Filling Games, 2011)

Care ar fi o strategie simplă/intuitivă de joc?