РАЗВИТИЕ НА ДИНАМИЧНО ПАРАЛЕЛНИ ПРИЛОЖЕНИЯ

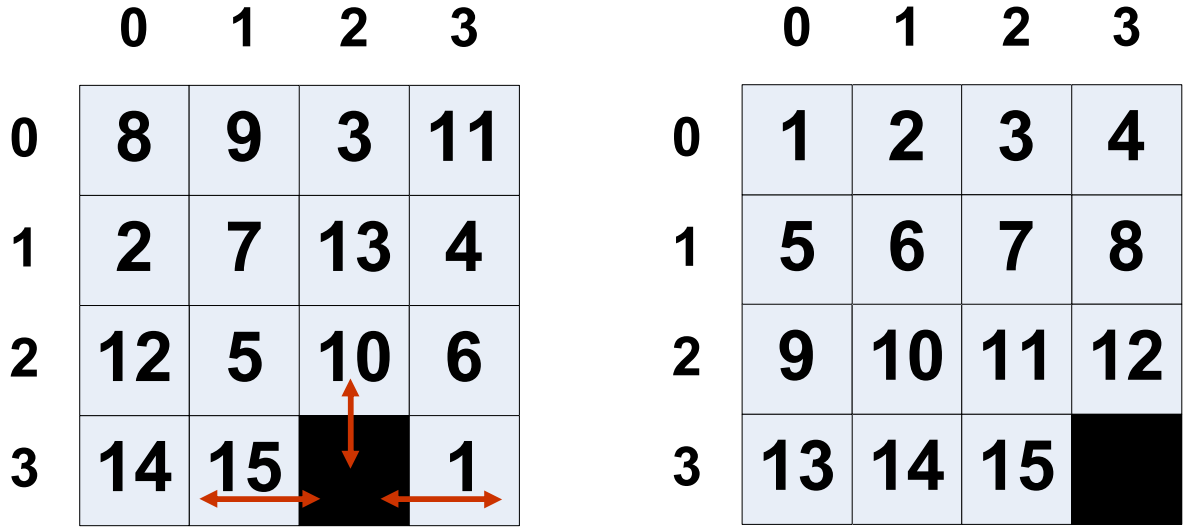
ЧАСТ 2

КОМБИНАТОРНО ТЪРСЕНЕ

Комбинаторно търсене по метода на клоните и границите

Branch-and-bound combinatorial search

- Вариант на търсенето с обратен ход
- Анализира се информацията за оптималността на частичните решения, като се избягват решенията, които не могат да бъдат оптимални
- Пример: решаване на пъзела на Сам Лойд с минимален брой ходове – оптимизационен проблем



Пъзел на Сам Лойд

- За представяне на позициите, които могат да бъдат получени от първоначалното разположение на плочките върху дъската използваме дърво на пространството на търсене
- Използваме метода “breadth-first search”
- Целта е да се разгледат минимален на брой ходове
- За всяка конфигурация определяме минималният брой ходове, които са необходими за решаване на пъзела

- ***Разстояние по Манхатан*** – между текущата позиция и желаната позиция върху дъската
- Две точки с координати

(x_1, y_1) (x_2, y_2)

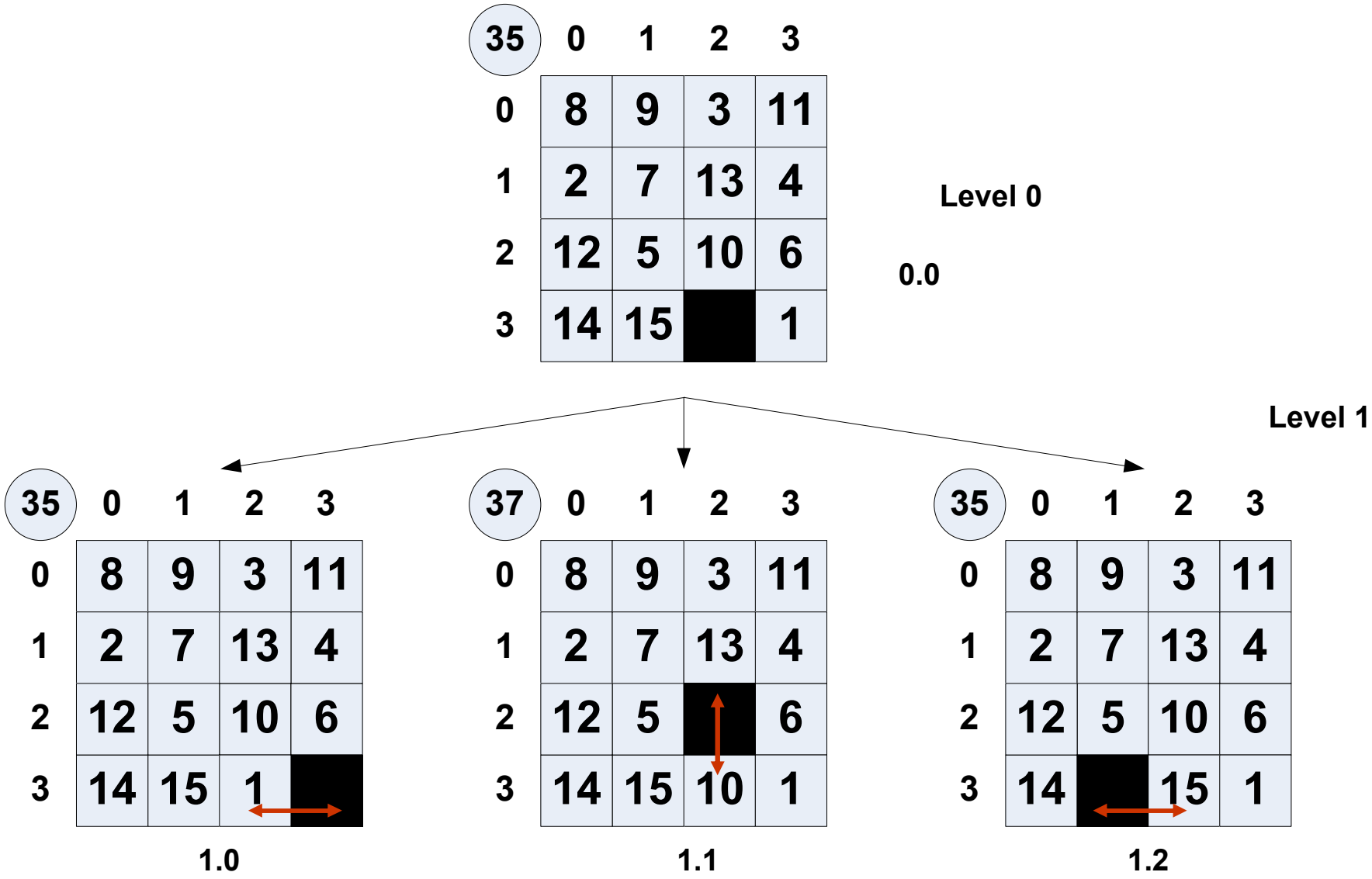
6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	0	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6

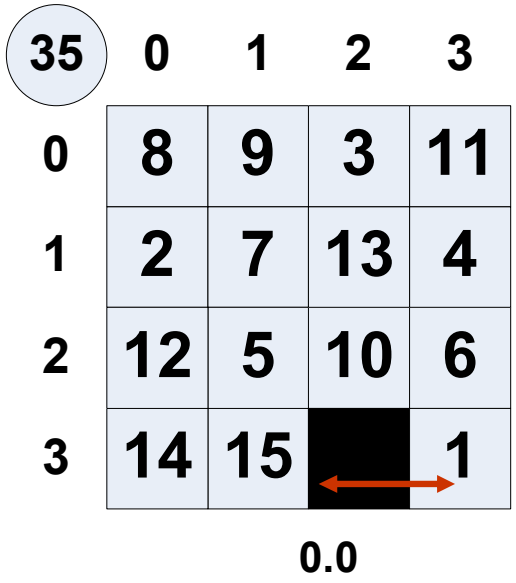
$$MD = |x_1 - x_2| + |y_1 - y_2|$$

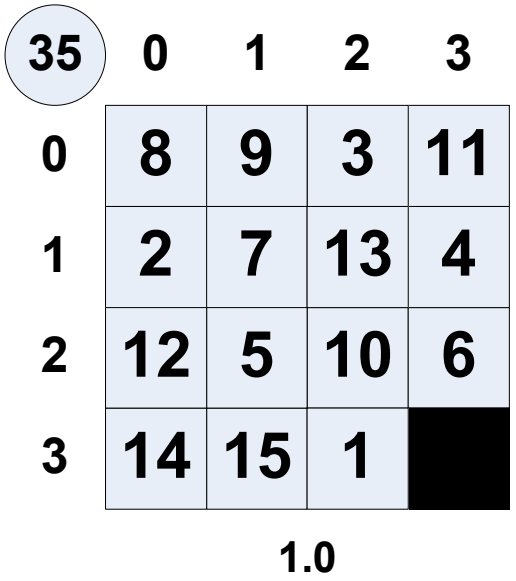
	0	1	2	3
0	8	9	3	11
1	2	7	13	4
2	12	5	10	6
3	14	15		1

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	

- ***Функция g*** – за всяка плочка, която не е на мястото си, прибавяме към Манхатановото разстояние броя на направените ходове
- Можем да се концентрираме върху областите на дървото, съдържащи най-обещаващите ходове
- Продължаваме търсенето от възела с най-малката стойност на функцията
- Ако два възела имат една и съща стойност на функцията, избираме произволно един от възлите

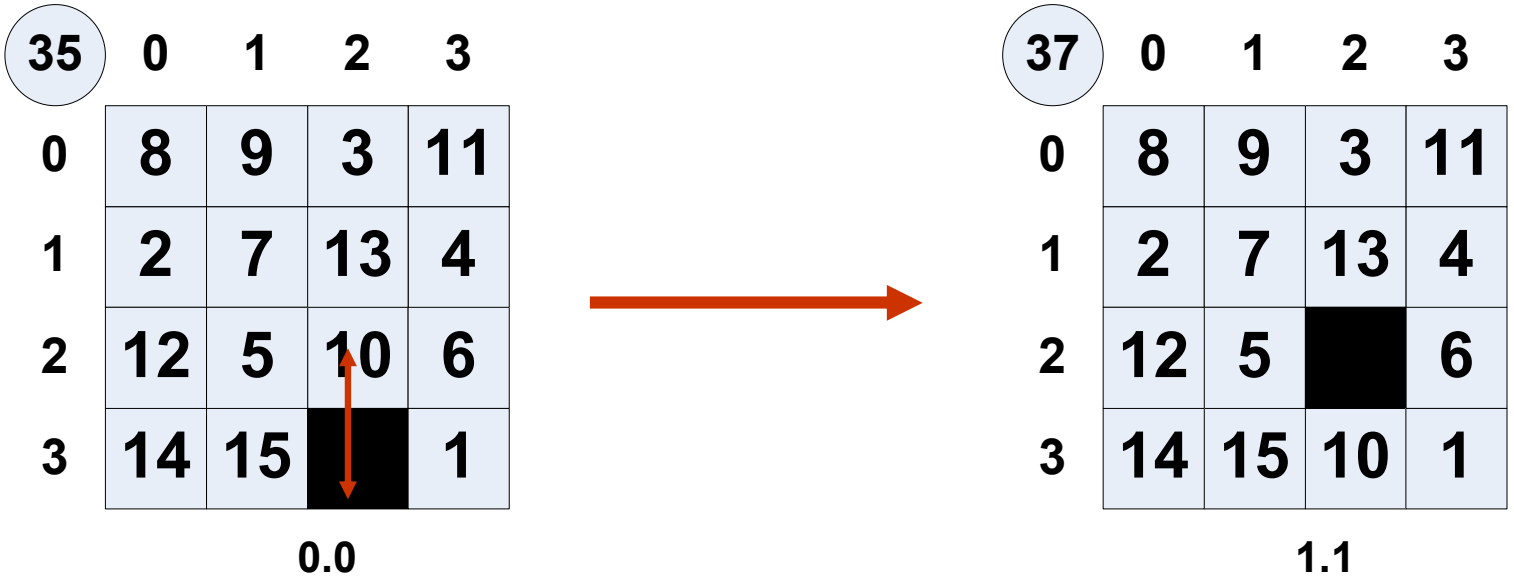

35 0 1 2 3
0 8 9 3 11
1 2 7 13 4
2 12 5 10 6
3 14 15 1
Level 0
0.0
Level 1
35 0 1 2 3
0 8 9 3 11
1 2 7 13 4
2 12 5 10 6
3 14 15 1
1.0
37 0 1 2 3
0 8 9 3 11
1 2 7 13 4
2 12 5 6
3 14 15 10 1
1.1
35 0 1 2 3
0 8 9 3 11
1 2 7 13 4
2 12 5 10 6
3 14 15 1
1.2





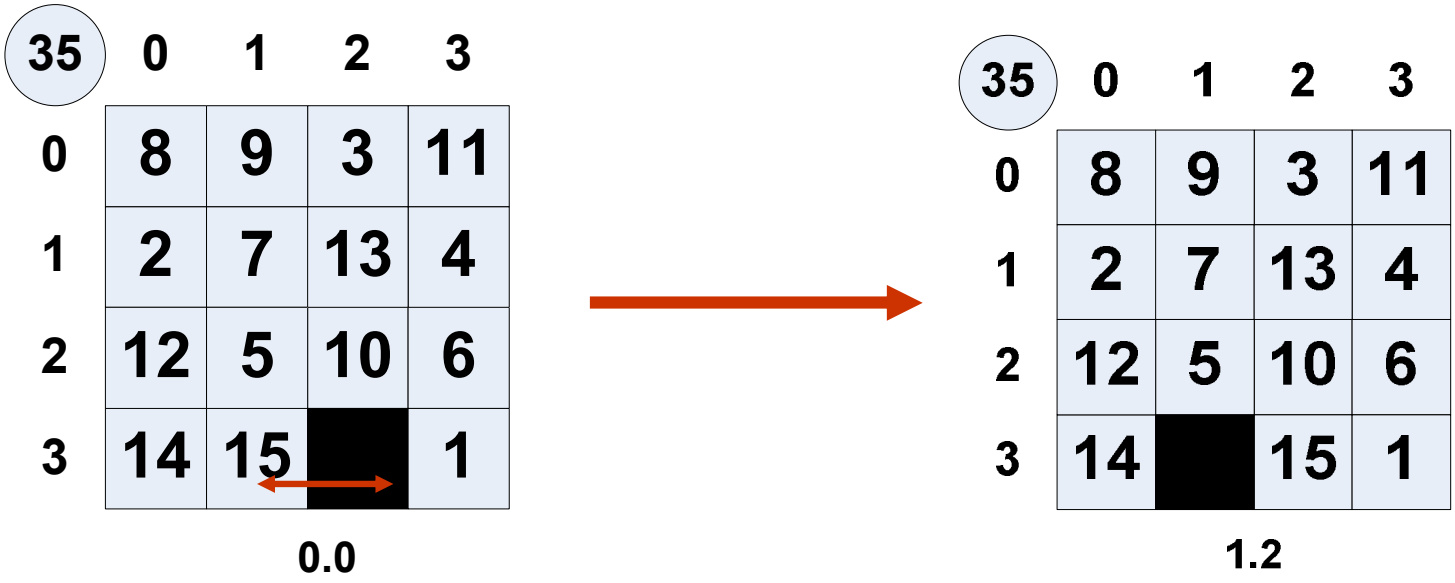
8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1

12 – (2,0)→(2,3) MD=3
5 – (2,1)→(1,0) MD=2
10 – (2,2)→(2,1) MD=1
6 – (2,3)→(1,1) MD=3
14 – (3,0)→(3,1) MD=1
15 – (3,1)→(3,2) MD=1
1 – (3,2)→(0,0) MD=5
MD total = 34



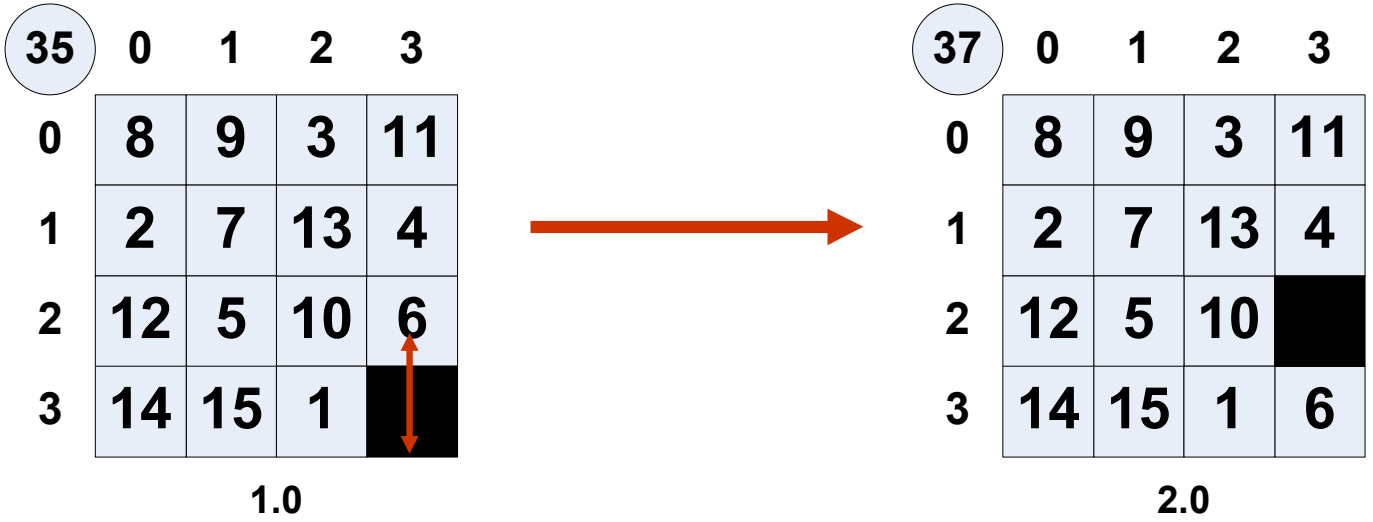
8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1

12 – (2,0)→(2,3) MD=3
5 – (2,1)→(1,0) MD=2
10 – (3,2)→(2,1) MD=2
6 – (2,3)→(1,1) MD=3
14 – (3,0)→(3,1) MD=1
15 – (3,1)→(3,2) MD=1
1 – (3,3)→(0,0) MD=6
MD total = 36



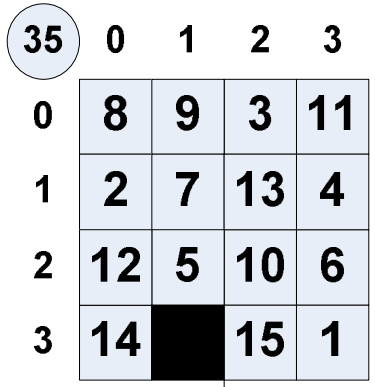
8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1

12 – (2,0)→(2,3) MD=3
5 – (2,1)→(1,0) MD=2
10 – (2,2)→(2,1) MD=1
6 – (2,3)→(1,1) MD=3
14 – (3,0)→(3,1) MD=1
15 – (3,2)→(3,2) MD=0
1 – (3,3)→(0,0) MD=6
MD total = 34



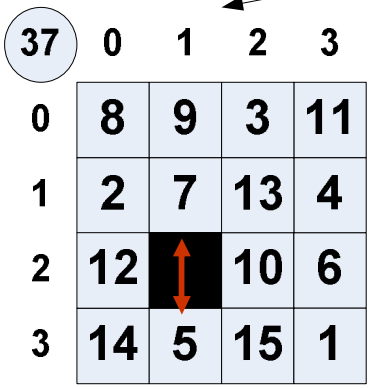
8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1

12 – (2,0)→(2,3) MD=3
5 – (2,1)→(1,0) MD=2
10 – (2,2)→(2,1) MD=1
6 – (3,3)→(1,1) MD=4
14 – (3,0)→(3,1) MD=1
15 – (3,1)→(3,2) MD=1
1 – (3,2)→(0,0) MD=5
MD total = 35



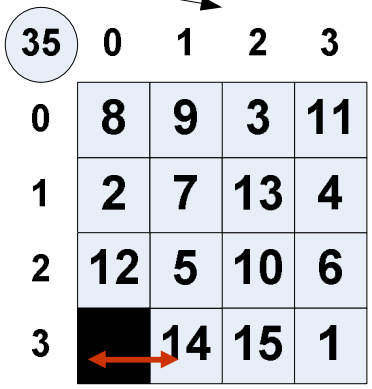
1.2

8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1



2.4

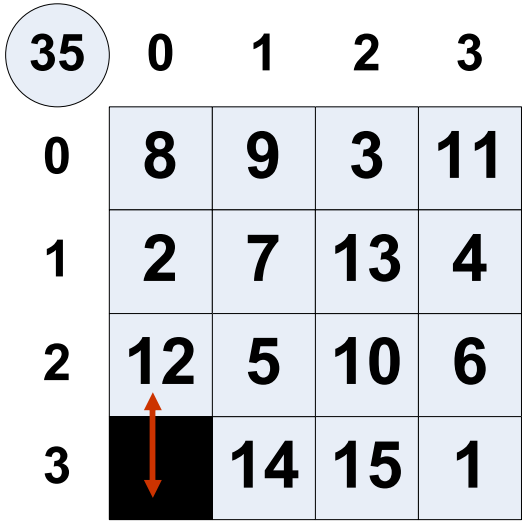
8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1



2.5

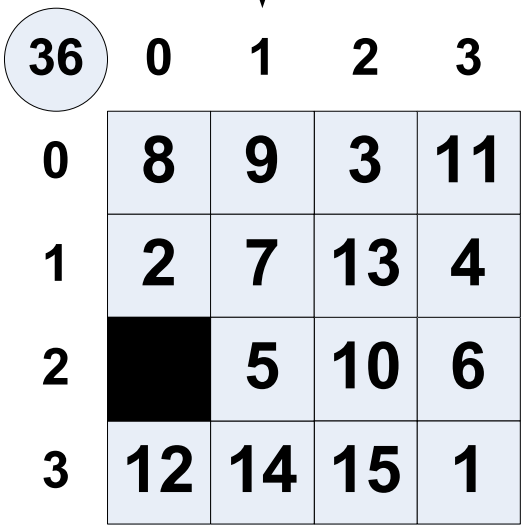
12 – (2,0)→(2,3) MD=3
5 – (3,1)→(1,0) MD=3
10 – (2,2)→(2,1) MD=1
6 – (2,3)→(1,1) MD=3
14 – (3,0)→(3,1) MD=1
15 – (3,2)→(3,2) MD=0
1 – (3,3)→(0,0) MD=6
MD total = 35

12 – (2,0)→(2,3) MD=3
5 – (2,1)→(1,0) MD=2
10 – (2,2)→(2,1) MD=1
6 – (2,3)→(1,1) MD=3
14 – (3,1)→(3,1) MD=0
15 – (3,2)→(3,2) MD=0
1 – (3,3)→(0,0) MD=6
MD total = 33

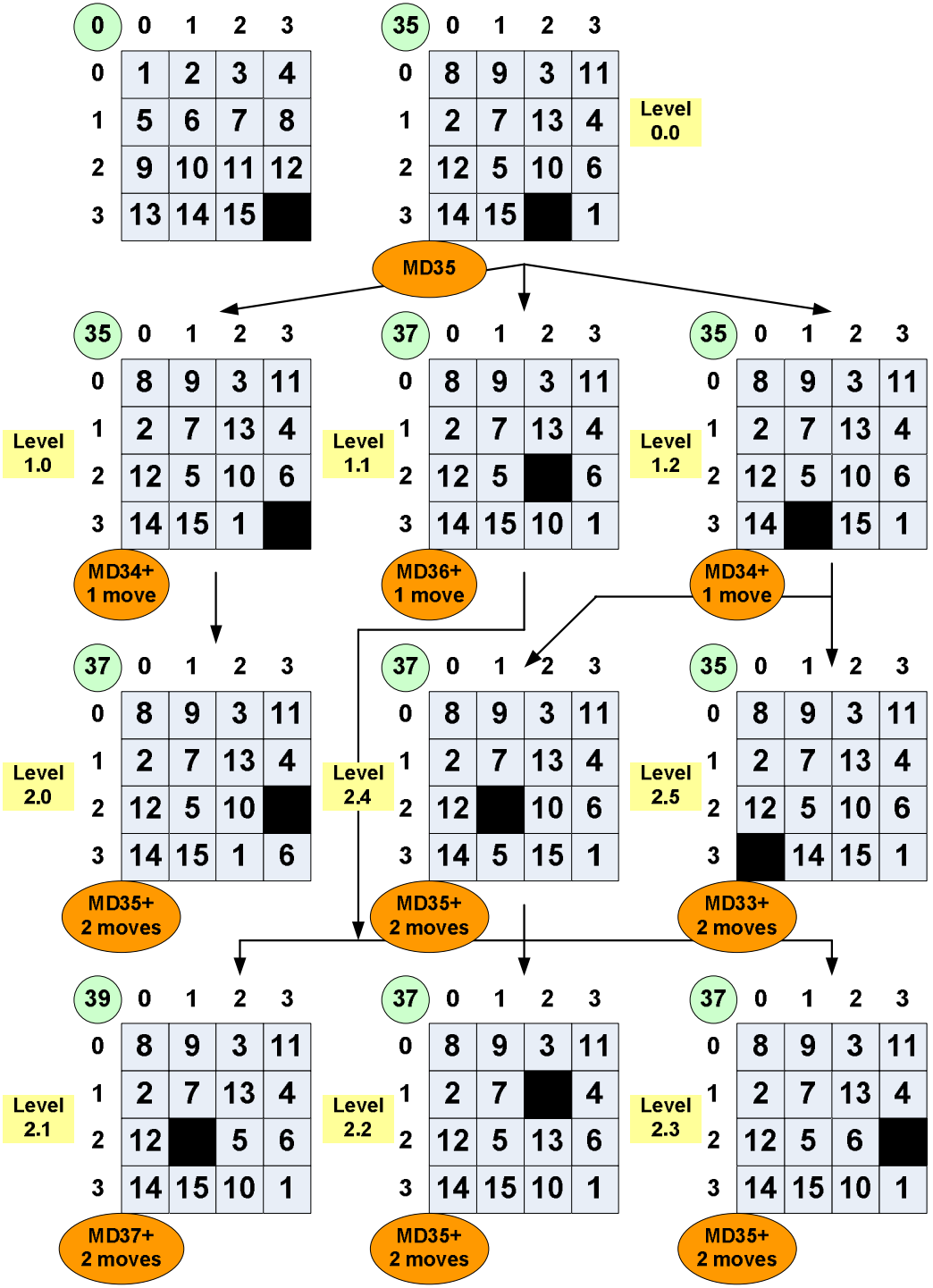


2.5

8 – (0,0)→(1,3) MD= 4
9 – (0,1)→(2,0) MD=3
3 – (0,2)→(0,2) MD=0
11 – (0,3)→(2,2) MD=3
2 – (1,0)→(0,1) MD=2
7 – (1,1)→(1,2) MD=1
13 – (1,2)→(3,0) MD=4
4 – (1,3)→(0,3) MD=1

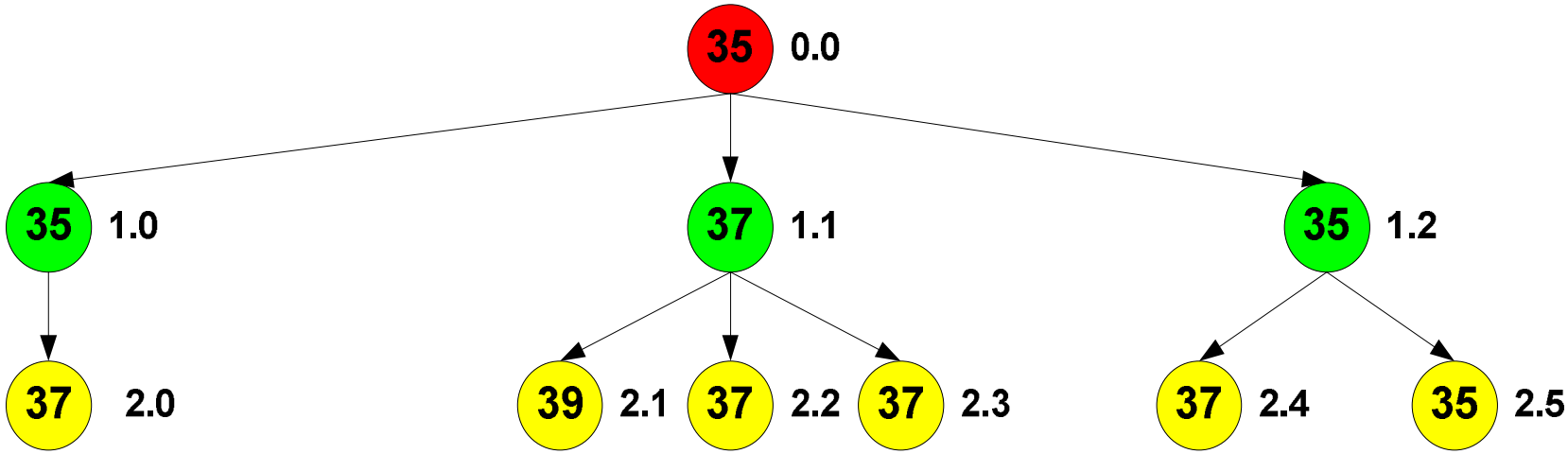


12 – (3,0)→(2,3) MD=4
5 – (2,1)→(1,0) MD=2
10 – (2,2)→(2,1) MD=1
6 – (2,3)→(1,1) MD=3
14 – (3,1)→(3,1) MD=0
15 – (3,2)→(3,2) MD=0
1 – (3,3)→(0,0) MD=6
MD total = 34


0
35
Level 0.0
MD35
35
37
35
Level 1.0
Level 1.1
Level 1.2
MD34+ 1 move
MD36+ 1 move
MD34+ 1 move
37
37
35
Level 2.0
Level 2.4
Level 2.5
MD35+ 2 moves
MD35+ 2 moves
MD33+ 2 moves
39
37
37
Level 2.1
Level 2.2
Level 2.3
MD37+ 2 moves
MD35+ 2 moves
MD35+ 2 moves

Дърво на търсене за решаване на пъзела на Сам Лойд

По всеки път в дървото стойностите на функциите *g* са ненамаляващи



- Стратегията за търсене определя реда на разглеждане на възлите
- Стратегията “първо най-добрият път” (best-first, best bound) избира неразгледаните проблеми с най-малката долна граница
- При стратегията “първо най-добрият път” се използват абстрактни типове данни – приоритетни опашки, които поддържат операциите *Delete_Min* или *Delete_Max*

Best-First Branch and Bound (minimization)

initial – първичен проблем
q – проритетна опашка с оператори Initialize, Insert и Delete_min
s – най-доброто решение, открито до сега
u – възел на дървото на търсене
v – нов възел с допълнителни ограничения

```
Initialize (q)
Insert (q, initial)
repeat
  u ← Delete_Min(q)
  if u е решение then
     Print_Solution (u)
     Halt
  else
     for i ←1 to Possible_Constraints(u) do
        Add constraint i to u, creating v
        Insert (q,v)
     endfor
  endif
forever
```

Производителност

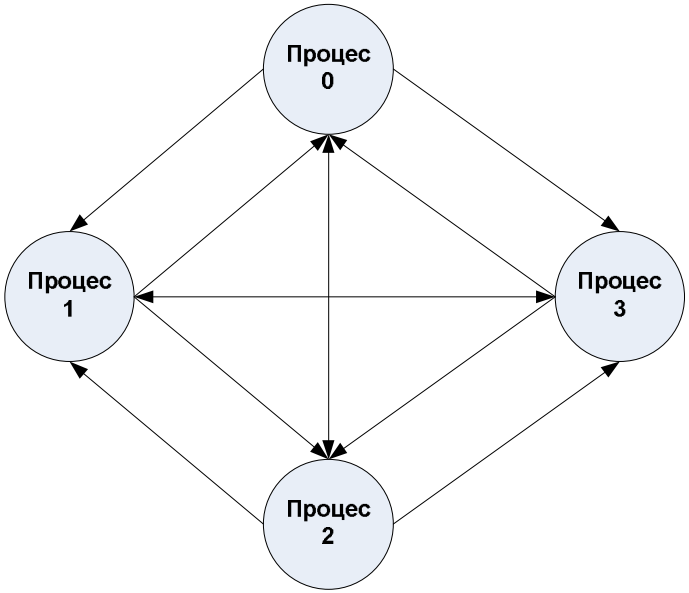
- Целта на търсенето по метода на клоните и границите е да се намали броят на възлите, които трябва да се разгледат като се използва функция на долната граница за елиминиране на възлите, които не могат да доведат до оптимални решения
- В най-лошия случай функцията определя обхождане на дървото в ширина без никакво “отрязване” на части на дървото
- Ако оптималното решение се намира на ниво k в дървото при среден фактор на разклонение b, в най-лошия случай сложността е $\sim\Theta(b^k)$
- Ограниченията от паметта поставят горна граница на размера на решавания проблем

Паралелно търсене по метода на клоните и границите

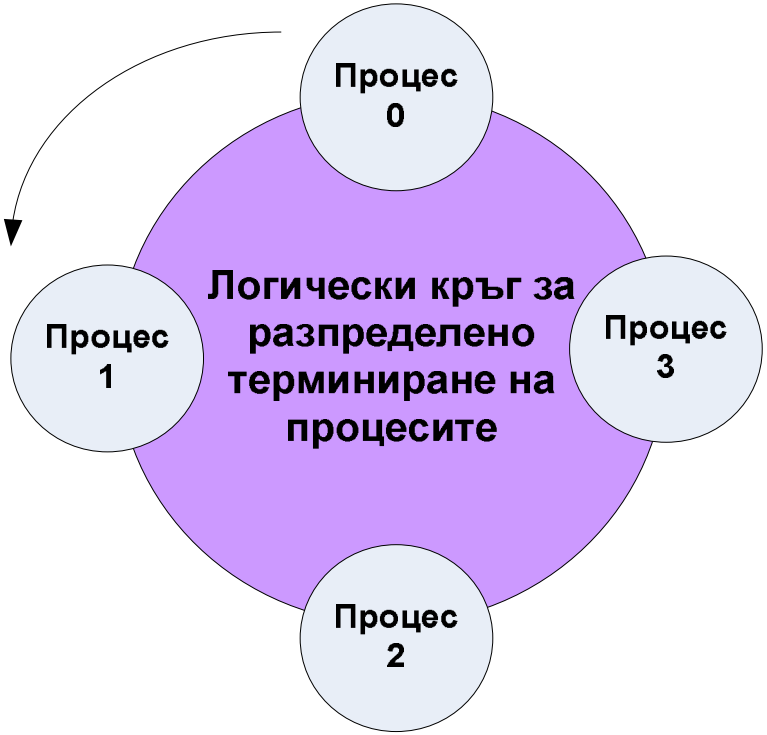
- Последователният алгоритъм предполага изграждането на приоритетна опашка за неразгледаните проблеми
- За мултикомпютърните платформи централизираната приоритетна опашка е неефективна
- Ако един процесор осъществява операциите над приоритетната опашка – тясно място
- Една приоритетна опашка не дава възможност за мащабиране

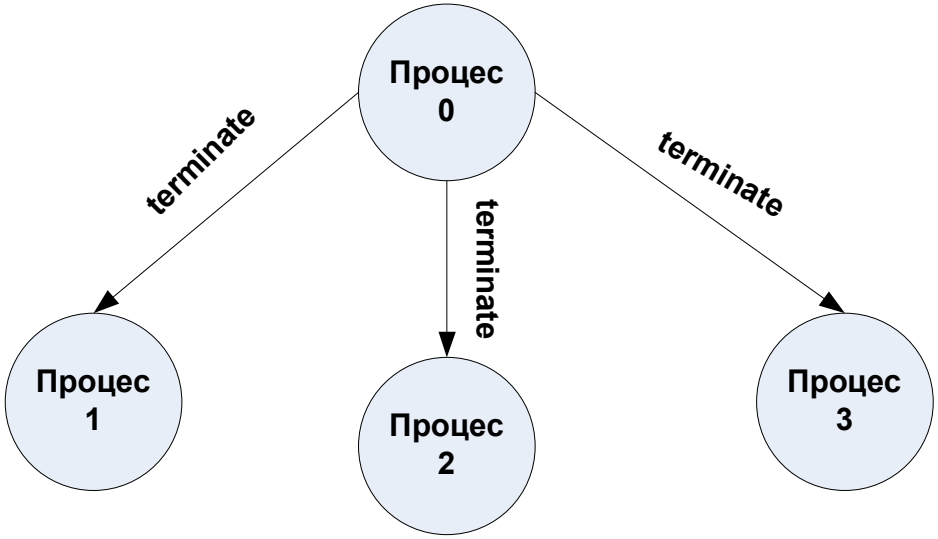
- Всеки процесор трябва да поддържа собствена опашка на неразгледаните подпроблеми
- През всяка итерация всеки процес премахва от приоритетната си опашка подпроблема с най-малка долна граница
- Ако подпроблемът не е възел на решение, то той се разделя на *b* подпроблема, които се вмъкват в приоритетната му опашка
- Не съществува синхронизация между процесите

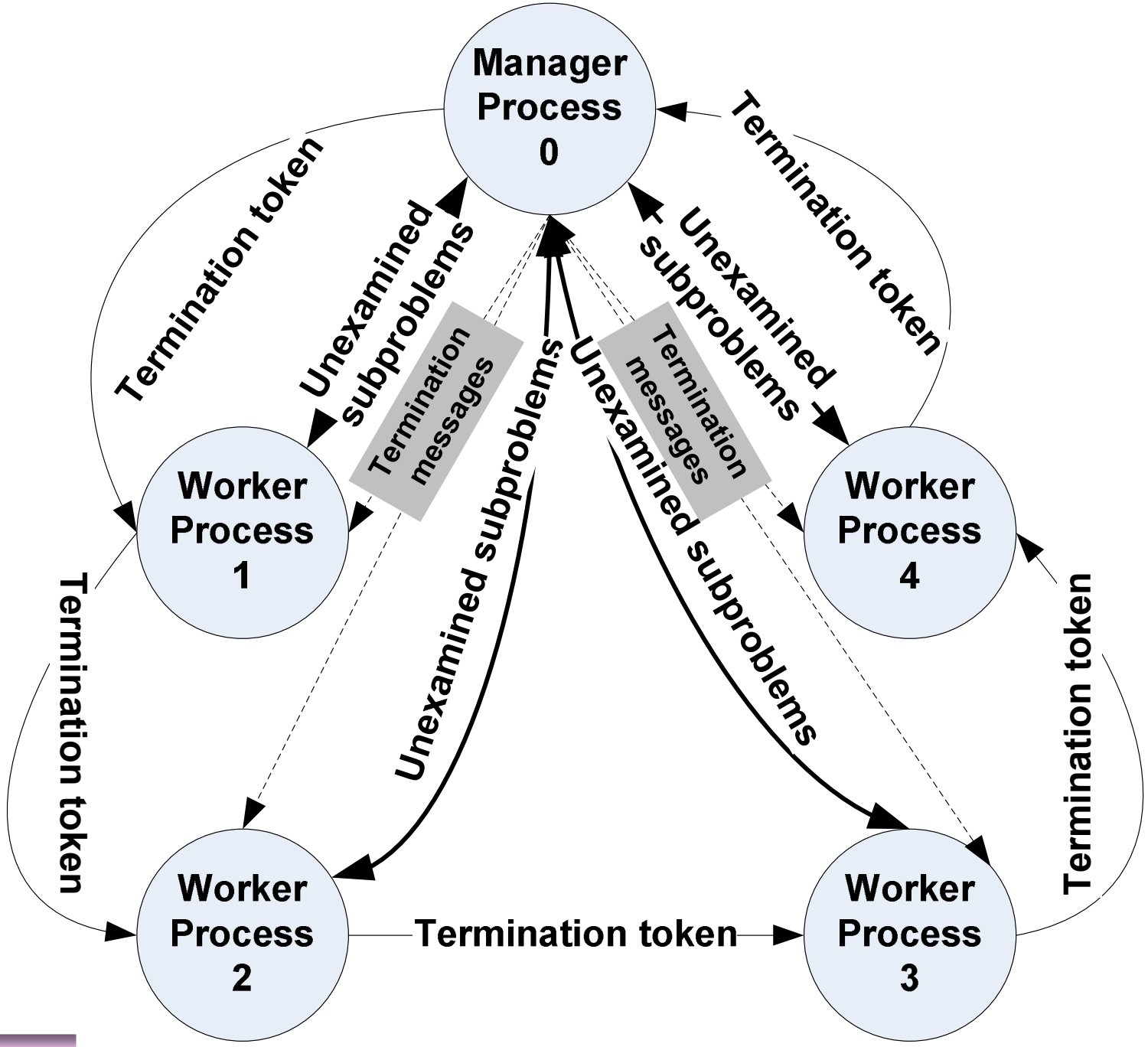
- От време на време даден процес изпраща неразгледани подпроблеми на друг процес
- В началото процес 0 съдържа първо-началния си проблем в приоритетната си опашка
- Приоритетните опашки на другите процеси са празни и те престояват
- Процес 0 разпределя неразгледаните подпроблеми, рекурсивно новите подпроблеми се разпределят, и т.н.



Динамично преразпределяне на товара посредством обмен на съобщения






Manager Process 0
Worker Process 1
Worker Process 4
Worker Process 2
Worker Process 3
Termination token
Unexamined subproblems
Termination messages
Termination token
Unexamined subproblems
Termination messages
Unexamined subproblems
Unexamined subproblems
Termination token
Termination token
Termination token

Паралелен алгоритм за търсене по метода на клоните и границите

Константи:

Comm_Interval – времето между две комуникационни транзакции

Temination – тагове на съобщенията за терминиране

Token – тагове на рамковото съобщение

Unexamined_Subproblem – тагове на съобщенията, съдържащи неразгледани подпроблеми

Функции:

Current_Time() – определяне на времето по стенен часовник

Delete_Min – изтриване на подпроблем с най-ниската граница от проритетната опашка

First_element – връща първия елемент от приоритетната опашка без да го изтрива

Initialize() – инициализация на размера на приоритетната опашка на 0

Insert() – вмъква подпроблем в приоритетната опашка

Is_Empty() – връща “истина” ако проритетната опашка е празна

Lower_Bound() – връща стойността на долната граница на неразгледан подпроблем

Променливи:

color – цвят на процеса (за разпределено детектиране на край на обработката)

global_c – цена на тектущото глобално най-добро решение, намерено до този момент

id – ранг на процеса

initial – оригинален проблем за решаване

last_comm – времетраене на последната комуникационна транзакция

local_c – цена на най-доброто решение, намерено до този момент от даден процес (локално най-добро решение)

local_s - най-доброто решение, намерено до този момент от даден процес (локално най-добро решение)

msg_count – брой на изпратените съобщения минус броя на получените съобщения

q – приоритетна опашка

token – рамково съобщение за проба на условията за разпределено терминиране на паралелните изчисления

u – възел на дървото на пространството на търсене

v – нов възел с допълнителни ограничения

Parallel Best-First Branch and Bound (minimization)

Initialize (*q*)
if *id*=0 then
  Insert (*q, initial*)
  token.c ← ∞
  token.color ← WHITE
  token.count ← 0
  изпращане на *token* към следващия процес в логическия ринг
endif
local_c ← ∞
best_soln ← ∞
last_comm ← Current_Time()
msg_count ← 0
color ← WHITE
repeat
  if Is_Empty(*q*) or (Current_Time() - *last_comm*>Comm_Interval) then
    BandB_Communication()
    last_comm ← Current_Time()

```
else if not Is_Empty(q) then
   u ← Delete_Min(q)
   if Lower_Bound(u) < best_c then
      color ← BLACK
      if u е решение then
         if Lower_Bound(u) < global_c then
            local_s ← u
            local_c ← Lower_Bound(local_s)
         endif
      else
          for i ← 1 to Possible_constraints(u) do
             Add constraint i to u, creating v
             if Lower_Bound(v) < global_c then
               Insert(q, v)
             endif
          endfor
      endif
   endif
 endif
forever
```

BandBCommunication():

```
if чакащо съобщение с таг Termination then Halt endif
if чакащо съобщение с таг Токен then
  получи съобщение с таг Токен
  if local_c < token_c then
     token_c ← local_c
     token_s ← local_s
  endif
  if token_c ≤ Lower_Bound(First_Element(q)) then Initialize(q) endif
  global_c ← token.c
  if id=0 then
    if (color = WHITE) and (token_color = WHITE) and
          (token.count + msg_count = 0) then
       изпрати съобщения с таг Termination към всички останали
     процеси
       Halt
    else
       token_color ← WHITE
       token.count ← 0
    endif
  else
```

```
    if color = BLACK then token.color ← BLACK
    token.count ← token.count + msg_count
  endif
  изпращане на token към следващия процес в логическия
     ринг
  color ← WHITE
endif
while има чакащи съобщения с таг Unexamined_Subproblem do
  получи съобщение с неразгледан подпроблем u
  msg_count ← msg_count – 1
  color ← BLACK
  if Lower_Bound(u) < global_c then Insert (q, u)
endwhile
if има повече от един неразгледан подпроблем в q then
  изпрати неразгледан подпроблем към друг процес
  msg_count ← msg_count + 1
  color ← BLACK
endif
return
```

Ефективност

- За да сме сигурни, че ще бъде намерено решение, и то ще бъде оптимално, трябва да са изпълнени две условия:

1. Поне един от възлите на решенията (включително всички негови деца в дървото на търсене) трябва да бъде разгледан – T_{node}
2. Процесите трябва да обходят всички възли на дървото на търсене, чиито долни граници на функцията на цената са по-малки от функцията на цената на оптималното решение – T_{lower_bounds}

- Времето за изпълнение на алгоритъма се определя от T_{node} или T_{lower_bounds} в зависимост от това, кое от двете събития завършва последно
- Кое от двете събития ще завърши последно зависи от броя на процесите и топологията на дървото на търсене

Последователен алгоритъм *best-first branch-and-bound*

- Последователният алгоритъм “обхождане първо на най-доброто решение” при търсене по метода на клоните и границите (*best-first branch-and-bound*) се основава на използването на приоритетна опашка, обхождане на възможния минимален брой възли, при зададена функция на границата *g* – *винаги първо се обхожда възела с най-малка стойност на g*
- Алгоритъмът спира при откриване на решение т.к. по определение не съществува възел с по-малка стойност на *g*

Паралелен алгоритъм
best-first branch-and-bound

- Възможно е да бъдат обходени възли, водещи до неоптимално решение
- Всеки процес обхожда възела, който локално предлага най-доброто решение – възела с минимална стойност на g в локалната приоритетна опашка
- Паралелния алгоритъм насърчава разпределението на подпроблеми с добри стойности на g между всички процеси като по този начин се намалява обема на излишните изчисления
- Стойността на g на текущото най-добро решение се разпространява чрез рамковото съобщение и процесите не обхождат възли със стойност на g, по-голяма от g на текущото най-добро решение
- Динамичното разпределение на товара повишава разходите за комуникация на паралелния алгоритъм

Условия за прекратяване на паралелния алгоритъм за търсене по метода на клоните и границите

- За разлика от търсенето с обратен ход, при което се търси какво да е решение, при търсенето по метода на клоните и границите се търси оптималното решение
- Условията за прекратяване на изпълнението на алгоритъма са две:

1. Намерено е решение
2. Не съществува по-добро решение

- Използва се рамково съобщение, циркулиращо в логически кръг от процеси – концепцията се основава на модифицирания алгоритъм на Дийкстра за разпределено детектиране на края на паралелното търсене
- Предназначението на рамковото съобщение е да се избегнат излишните изчисления и да се проверят условията за край на търсенето – всички процеси са неактивни и обменът на всички съобщения е завършил (за избягване на мъртво блокиране)
- Изпълнението на паралелния алгоритъм се прекратява от процес с ранг 0

К Р А Й