Algorytmy dokładne, rozgałęzianie

Autorzy:
Bartłomiej Chechliński
Mateusz Chiliński
Kazimierz Wojciechowski

Branching (Backtracking)

Branching

Reguły Branchingu

“A branching rule is used to solve a problem instance by recursively solving smaller instances of the problem. “

Reguły Redukcji

“A reduction rule is used to simplify a problem instance or to halt the algorithm. “

Złożoność

Let $T(n)$ be the maximum number of leaves in any search tree of an input of size n when executing a certain branching algorithm. The general approach is to analyse each branching rule separately and finally to use the worst-case time over all branching rules as an upper bound on the running time of the algorithm.

Let b be any branching rule of the algorithm to be analysed. Consider an application of b to any instance of size n. Let $r \geq 2$, and $t_i > 0$ for all $i \in \{1,2,\ldots,r\}$. Suppose rule b branches the current instance into $r \geq 2$ instances of size at most $n-t_1, n-t_2, \ldots, n-t_r$, for all instances of size $n \geq \max\{t_i : i = 1,2,\ldots r\}$. Then we call $\mathbf{b} = (t_1,t_2,\ldots t_r)$ the *branching vector* of branching rule b. This implies the linear recurrence

$$T(n) \leq T(n-t_1) + T(n-t_2) + \cdots + T(n-t_r). \tag{2.1}$$

Złożoności branchingu(problem podzielony na podproblemy max. n-x, n-y)

	1	2	3	4	5	6
1	2.0000	1.6181	1.4656	1.3803	1.3248	1.2852
2	1.6181	1.4143	1.3248	1.2721	1.2366	1.2107
3	1.4656	1.3248	1.2560	1.2208	1.1939	1.1740
4	1.3803	1.2721	1.2208	1.1893	1.1674	1.1510
5	1.3248	1.2366	1.1939	1.1674	1.1487	1.1348
6	1.2852	1.2107	1.1740	1.1510	1.1348	1.1225

Table 2.1 A table of branching factors (rounded up)

Zastosowanie

Rozgałęzianie jest jedną z podstawowych technik do projektowania szybkich algorytmów wykładniczych. Można śmiało powiedzieć, że co najmniej połowa opublikowanych algorytmów wykładniczych jest algorytmami rozgałęziającymi. Ponadto, dla wielu problemów NP-trudnych najszybszym znanym algorytmem dokładnym jest algorytm rozgałęzienia. Wiele z nich zostało opracowanych w ciągu ostatnich dziesięciu lat poprzez zastosowanie technik takich jak Dziel i Zwyciężaj itd.

W porównaniu do niektórych innych technik tworzenia algorytmów wykładniczych, algorytmy rozgałęziające mają pewne ważne właściwości. Zazwyczaj algorytmy te potrzebują pamięci wielomianowej (lub liniowej). Czas pracy na niektórych danych wejściowych może być znacznie lepszy niż najgorszy czas pracy. Pozwalają na różne naturalne ulepszenia, które nie zmieniają pesymistycznej złożoności, ale znacznie przyspiesza czas pracy w wielu przypadkach.

k-SAT

SAT

Problem spełnialności – zagadnienie rachunku zdań, określające czy dla danej formuły logicznej istnieje takie podstawienie (wartościowanie) zmiennych zdaniowych, żeby formuła była prawdziwa. Jest ono równoważne negacji odpowiedzi na pytanie czy „negacja tej formuły jest tautologią”.

k-SAT

Istnieje też problem spełnialności formuł w koniunkcyjnej postaci normalnej (ang. CNF - conjunctive normal form), których klauzule mają nie więcej niż k literałów (k-SAT). Literałem nazywamy zmienną lub zmienną zanegowaną, klauzulą nazywamy alternatywę literałów, natomiast formuła to koniunkcja klauzul. 1-SAT i 2-SAT mają rozwiązania w P, czyli w deterministycznym czasie wielomianowym, natomiast już 3-SAT jest NP-zupełny, czyli taki, że każdy problem z klasy NP jest do niego redukowalny przy pomocy redukcji w czasie wielomianowym.

Algorytm k-sat1

Algorithm `k-sat1`(F).
Input: A CNF formula F.
Output: Return true if F is satisfiable, otherwise return false.

if F *contains an empty clause* **then**
  return false
if F *is an empty formula* **then**
  return true
choose any clause $c = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_q)$ of F
$b_1 = \texttt{k-sat1}(F[\ell_1 = \text{true}])$
$b_2 = \texttt{k-sat1}(F[\ell_1 = \text{false}, \ell_2 = \text{true}])$
$\cdots \qquad\qquad \cdots$
$b_q = \texttt{k-sat1}(F[\ell_1 = \text{false}, \ell_2 = \text{false}, \cdots, \ell_{q-1} = \text{false}, \ell_q = \text{true}])$
return $b_1 \vee b_2 \cdots \vee b_q$

Fig. 2.2 Algorithm `k-sat1` for *k*-`Satisfiability`

Złożoności k-sat1

$$x^{q+1} - 2x^q + 1 = 0.$$

Złożoność czasowa problemu jest rzędu O(βk^n), gdzie β znajdujemy poprzez rozwiązanie równania, poprzez podstawianie k za q.

Kilka przykładowych rozwiązań:

- β2 < 1.6181
- β3 < 1.8393
- β4 < 1.9276
- β5 < 1.9660

Optymalizacje?

Możemy dodać następującą zasadę redukcji, aby uzyskać lepsze wyniki dla niektórych danych:

- Zastosowanie algorytmu liniowego dla k=2.

Jak widać zasada jest trywialna, a dla pewnych danych poprawia wydajność, nie powoduje też zwiększenia złożoności algorytmu.

Autarki

The tool used to achieve this goal is a logical one. Autarkies are partial (truth) assignments satisfying some subset $F' \subseteq F$ (called an autark subset), while not interacting with the clauses in $F \setminus F'$. In other words, a partial truth assignment t of a CNF formula F is called an *autark* if for every clause c of F for which the Boolean value of at least one literal is set by t, there is a literal ℓ_i of c such that $t(\ell_i) = \text{true}$. For example, for the following formula

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_4})$$

the partial assignment $x_1 = \text{true}, x_2 = \text{false}$ is an autark.

Algorytm k-sat2 (Złożoność O(β(k-1)^n)

Algorithm `k-sat2`**(F).**
Input: A CNF formula F.
Output: Return true if F is satisfiable, otherwise return false.

if *F contains an empty clause* **then**
  return false
if *F is an empty formula* **then**
  return true
choose a clause $c = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_q)$ of F of minimum size
let t_1 be the assignment corresponding to $F_1 = F[\ell_1 = \text{true}]$
let t_2 be the assignment corresponding to $F_2 = F[\ell_1 = \text{false}, \ell_2 = \text{true}]$
$\cdots$ $\cdots$
let t_q be the assignment corresponding to
$F_q = F[\ell_1 = \text{false}, \ell_2 = \text{false}, \cdots, \ell_{q-1} = \text{false}, \ell_q = \text{true}]$
if *there is an $i \in \{1, 2, \ldots, q\}$ s.t. t_i is autark for F_i* **then**
  return `k-sat2`(F_i)
else
  $b_1 =$ `k-sat2`$(F[\ell_1 = \text{true}])$
  $b_2 =$ `k-sat2`$(F[\ell_1 = \text{false}, \ell_2 = \text{true}])$
  $\cdots$ $\cdots$
  $b_q =$ `k-sat2`$(F[\ell_1 = \text{false}, \ell_2 = \text{false}, \cdots, \ell_{q-1} = \text{false}, \ell_q = \text{true}])$
  if *at least one b_i is* true **then**
    return true
  else
    return false

Fig. 2.3 Algorithm `k-sat2` for k-SATISFIABILITY

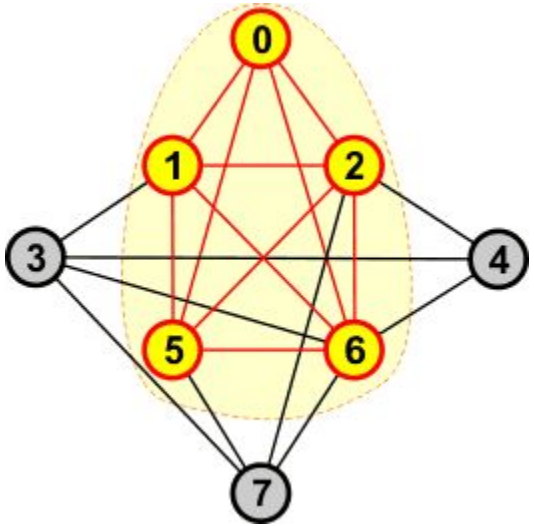
Kliki

Czym jest klika?

Klika - podgraf pełny grafu.

Maksymalna klika - dołożenie kolejnego wierzchołka nie utworzy grafu pełnego.

Najliczniejsza klika - klika o największej liczbie wierzchołków pośród klik w grafie.



Jaka jest maksymalna liczba f(n) możliwych klik w grafie o n wierzchołkach i jakie grafy mają taką liczbę klik?

Pytanie zadane przez Paula Erdős i Leo Moser w latach 1950.

Wyznaczanie wartości f(n)

Twierdzenie 1.

$$\text{If } n \geq 2, \text{ then } f(n) = \begin{cases} 3^{n/3}, & \text{if } n \equiv 0 (\bmod 3); \\ 4.3^{[n/3]-1}, & \text{if } n \equiv 1 (\bmod 3); \\ 2.3^{[n/3]}, & \text{if } n \equiv 2 (\bmod 3). \end{cases}$$

Dowód

G - dowolny graf spójny o co najmniej 5 wierzchołkach;
F(x) - zestaw wierzchołków połączonych z wierzchołkiem x;
a(x) - liczba grafów pełnych w grafie F(x), maksymalne względem G - x;
b(x) - liczba grafów pełnych w grafie F(x), maksymalne w F(x), nie maksymalne w G - x;

(1) $p(x) = a(x) + b(x)$

(2) $c(G - x) = c(G) - b(x)$

Dowód cd.

x, y - nie połączone wierzchołki;
G(x; y) - graf bez krawędzi wychodzących z “x”, z dodanymi krawędziami “x” ⇔ F(y);

$$(3) \quad c(G(x; y)) = c(G) + p(y) - p(x) + a(x)$$

b(x) = b(x, y) + b’(x, y), gdzie b(x, y) oznacza liczbę grafów pełnych w F(x) ∩ F(y), które są maksymalne w odniesieniu do grafu G - x - y.

$$\begin{aligned} c(G(x,y)) &= c(G) - b'(x,y) + a(y) + b'(y,x) \\ &= c(G) - b'(x,y) - b(x,y) + p(x) \\ &= c(G) + p(y) - p(x) + a(x) \end{aligned}$$

Dowód cd.

G - dowolny graf o co najmniej 5 wierzchołkach i posiadającym maksymalną liczbę klik:

- musi być spójny,
- nie ma wierzchołka połączonego z wszystkimi innymi wierzchołkami.

Jeśli x i y nie są połączone w G, to p(x) = p(y). W innym wypadku, np p(y) > p(x), to G(y, x) posiada więcej klik niż G.

Z (3) wynika, że a(x) = 0, dla każdego wierzchołka “x” w grafie G.

Dowód cd.

a, b … - wierzchołki z którymi wierzchołek x należący do G nie jest połączony

$G' = G$

$G'' = G'(a; x)$

$G''' = G''(b; x)$

...

Dowód cd.

Kontynuacja dla y należących do F(x).

Otrzymany graf G*:

- tyle samo klik co G,
- wierzchołki G* mogą być podzielone na rozłączne podzbiory, dwa wierzchołki są połączone tylko wtedy gdy nie należą do tego samego podzbioru.

Liczność podzbiorów: $j_1, j_2 \dots j_z$, gdzie: $j_1 + j_2 + \dots + j_z = n$

$$(4) \quad c(G^*) = j_1 * j_2 * \dots * j_z$$

Grafy o największej liczbie klik

Niech H_n oznacza graf posiadający f(n) klik opisany wcześniej jako G*.

Twierdzenie 2.

Jeśli graf G posiada n wierzchołków i f(n) klik, to wtedy $G = H_n$, jeśli $n > 1$.

Dowód

Niech G będzie grafem o co najmniej 5 wierzchołkach i f(n) klikach, ale nie jest grafem H_n.
Modyfikujemy G aż będzie H_n.
Niech G' - ostatni graf przed otrzymaniem H_n. Posiada f(n) klik i dwa wierzchołki x, y, które nie są ze sobą połączone: $G'(x; y) = H_n$.
Niech $n = 3l$. H_n składa się z trójek wierzchołków.

$$(5) \quad p(x) = t_1 * t_2 * \ldots * t_{l-1}$$

Dowód cd.

$$p(x) = p(y)$$

$$p(y) = 3^{l-1}$$

$t_i \leq 3$ z (5) więc $t_i = 3$

F(x) zawiera się w G’ - x - y - z

$$G' = H_n$$

Dolne ograniczenie liczby klik różnych rozmiarów

$$g(n) \geq [(n+1)/2]$$

$$g(n) \geq n - [\log_2 n] - 2[\log_2 \log_2 n] - 4 \qquad \text{dla } n \geq 26$$

Górne ograniczenie liczby klik różnych rozmiarów

$$g(n) \leq n - [\log_2 n] \qquad \text{dla } n \geq 4$$

Twierdzenie Turan’a

W dowolnym grafie G takim, że G ma co najwyżej n wierzchołków oraz nie zawiera (r + 1) wierzchołkowej kliki jest co najwyżej:

$$\frac{r-1}{r}\frac{n^2}{2} = (1-\frac{1}{r})\frac{n^2}{2}$$

krawędzi.

Podsumowanie

Pomimo poziomu skomplikowania problemu szukania klik w grafie, wyznaczenia ich ilości jest dużo łatwiejsze.

Stosując poznane twierdzenia jesteśmy w stanie sprawdzić czy graf ma szansę być K-kolorowalny, bez szukania konkretnych klik.

3-Kolorowanie w czasie $1.3289...^n$

3-Kolorowanie grafów w czasie $1.3289...^n$

3-Coloring in Time O(1.3289^n),
czerwiec 2000,
Richard Beigel (Univ. of Illinois Chicago),
David Eppstein (Univ. of California, Irvine)

Wysokopoziomowe kroki algorytmu

1. Wyznaczamy w grafie **maksymalny bujny las**
2. Pokrywamy pozostałe wierzchołki drzewami o wysokości 2 tworząc las
3. Kolorujemy na wszystkie możliwości wewnętrzne wierzchołki bujnego lasu oraz niektóre w ramach drzew wysokości 2
4. Sąsiedzi pokolorowanych wierzchołków mogą być pokolorowanie jedynie na 2 możliwości, zatem są skutecznie eliminowani w ramach algorytmu CSP
5. Pozostałe wierzchołki kolorujemy za pomocą skrajnie zoptymalizowanego algorytmu **(3,2)-CSP**

Pesymistyczna złożoność czasowa

Złożoność czasowa jest rzędu $3^p 2^q \Lambda^s (3\Lambda^3)^{t/7}$ gdzie:

- *p* to liczba korzeni w bujnych lasach
- *q* to liczba wierzchołków wewnętrznych niebędących korzeniem w bujnym lesie
- *r* to liczba liści w bujnym lesie
- *s* to liczba sąsiadów liści w bujnym lesie
- *t* to wszystkie pozostałe wierzchołki stopnia 3 w pozostałym lesie wysokości 2
- $\Lambda \approx 1.36443$ to szczególny *work factor* (t.j. współczynnik rozgałęzienia rekurencji, największy pierwiastek funkcji $1-2x^{-4}-2x^{-5}$)

Po uwzględnieniu ograniczeń na wszystkie zmienne wykorzystując:

1. definicję bujnego lasu
2. maksymalności bujnego lasu
3. lemat 22 ograniczający sumę s+t przez ułamek liści bujnych lasów

obliczamy dozwoloną pesymistyczną kombinację współczynników otrzymując złożoność $2^q 3^{4q/3} \Lambda^{8q} \approx 1.3289...^n$

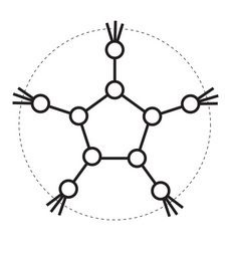
Czym jest maksymalny bujny las?

Bujny las to las bez korzeni, którego wewnętrzne wierzchołki są stopnia co najmniej 4.

Maksymalność lasu formalnie oznacza, że:

1. Sąsiedzi wierzchołków wewnętrznych również należą do lasu
2. Liście lasu mają co najwyżej 2 sąsiadów spoza lasu
3. Wierzchołki spoza lasu posiadają co najwyżej 3 sąsiadów spoza lasu

Lemat 22 mówi że w grafie którego najmniejszy stopień to co najmniej 3 w którym nie ma cyklu wierzchołków stopni 3 (przypadek ten rozpatruje poprzedni lemat 20) liczba wierzchołków spoza bujnego lasu jest ograniczona przez 20/3 liczby liści bujnego lasu



Czym jest problem (a,b)-CSP?

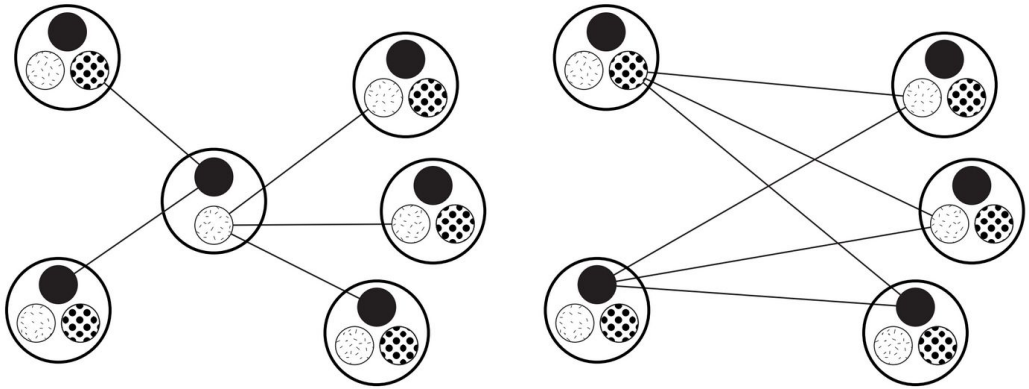
CSP w skrócie *Constraint Satisfaction Problem* jest definiowany jako trójka:

1. zbioru zmiennych
2. rodziny zbiorów możliwych wartości dla zmiennych
3. Zbioru ograniczeń
    a. Ograniczenie to para której pierwszy element to podzbiór zmiennych, a drugi to pewna relacja nad zmiennymi

(a,b)-CSP to problem spełnialności gdzie każda zmienna może przybrać *a* wartości, natomiast ograniczenia dotyczą co najwyżej *b* zmiennych.

Przegląd wybranych lematów usprawniających złożoność algorytmu CSP

Lemat 2: Istnieje zmienna dla której dozwolone są tylko 2 wartości, możemy je rozszerzyć na sąsiadów i usunąć dany wierzchołek



Przegląd wybranych lematów usprawniających złożoność algorytmu CSP

Lemat 3: (a, 2)-CSP: ograniczenia ((v, X), (w, Z)), ((v, Z), (w, Y)) można pokolorować na (v, X) oraz (w, Y)

Lemat 4: (a, 2)-CSP: każde wystąpienie ((v, R), (w, X)) implikuje ((v, B), (w, X)) - stąd można usunąć możliwość kolorowania v na B i możliwe jest zastosowanie lematu 2

Lemat 6: (a, 2)-CSP: wystąpienie ograniczenia (v, R) dla każdego z 3 możliwych kolorów w implikuje niemożność pokolorowania v na kolor R, zatem stosujemy lemat 2 w celu usunięcia wierzchołka

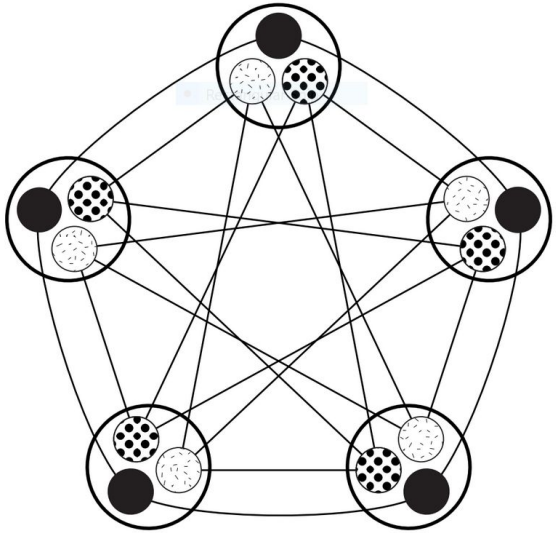
Złożoność czasowa

Po zastosowaniu lematów i redukcji problemu w czasie **wielomianowym** algorytmem backtracking złożoność czasowa jest rzędu $1.36443...^n = \Lambda^n$

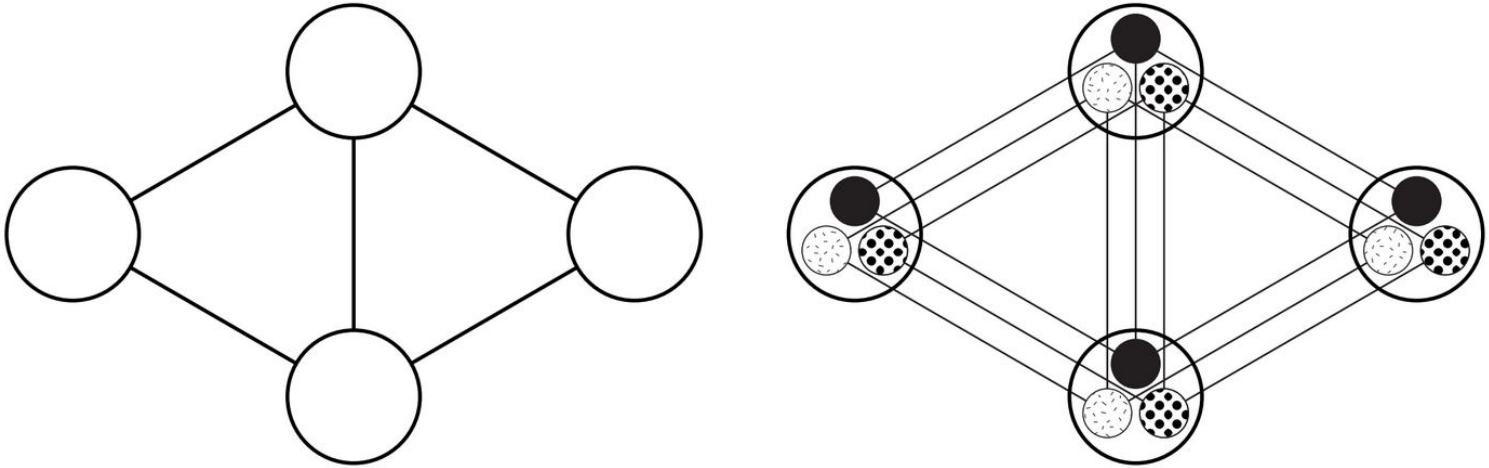
Szerokie zastosowanie algorytmu:

- Problem 3-kolorowanie można rozwiązać w czasie Λ^n
- Problem 3-kolorowanie krawędziowe można rozwiązać w czasie Λ^n
- Problem 3-SAT można rozwiązać w czasie Λ^t
- Problem (d, 2)-CSP dla d>3 można rozwiązać algorytmem randomizowanym w czasie rzędu $(d \times 0.4518\ldots)^n$

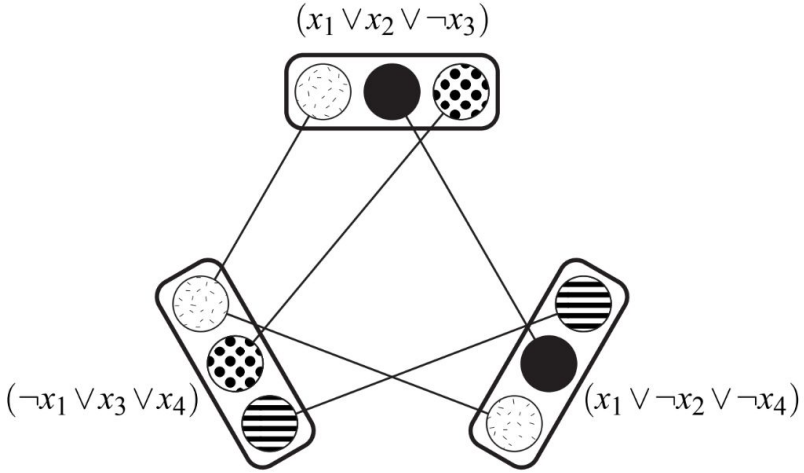
Przykład ogólny (3,2)-CSP



3-kolorowanie jako problem (3,2)-CSP



3-SAT jako problem (3,2)-CSP



Dziękujemy za uwagę