Optymalizacja. Przeszukiwanie „tabu”

dr hab. inż. Maciej Komosiński, prof. PP

Instytut Informatyki
Politechnika Poznańska
www.cs.put.poznan.pl/mkomosinski

© Maciej Komosiński, Maciej Hapke

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!
- Poziom aspiracji – akceptacja atrakcyjnych ruchów zakazanych

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!
- Poziom aspiracji – akceptacja atrakcyjnych ruchów zakazanych
  Algorytm przeszukiwania „tabu”

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!
- Poziom aspiracji – akceptacja atrakcyjnych ruchów zakazanych
  Algorytm przeszukiwania „tabu”
- Czasochłonne sprawdzanie całego sąsiedztwa w każdym kroku?

Naturalny sposób powstania algorytmu

- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!
- Poziom aspiracji – akceptacja atrakcyjnych ruchów zakazanych
  Algorytm przeszukiwania „tabu”
- Czasochłonne sprawdzanie całego sąsiedztwa w każdym kroku?
  „Kandydaci” – podzbiór sąsiedztwa $V \subset N$

Naturalny sposób powstania algorytmu

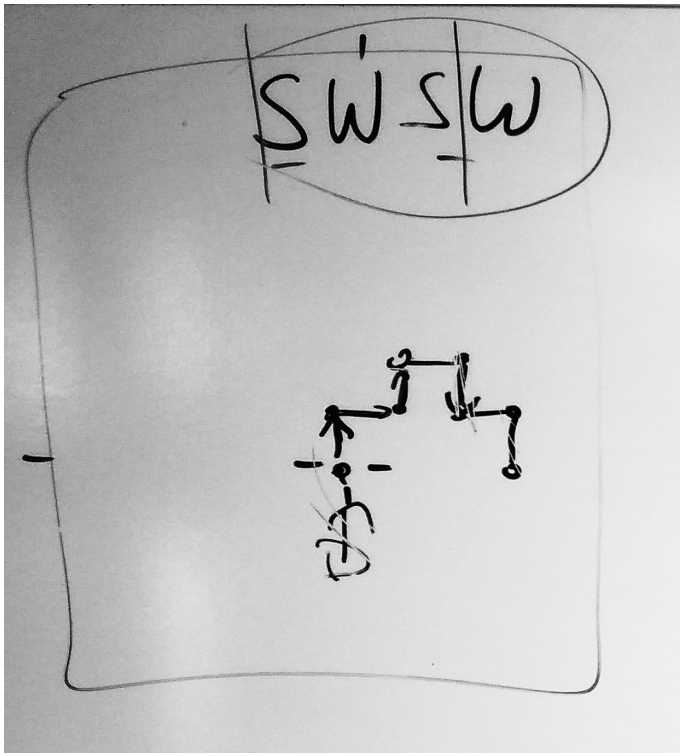
- Algorytm optymalizacji lokalnej
  Niezdolność wyjścia z lokalnych optimów!
- Akceptacja ruchów niepolepszających
  Cykle!
- Lista „tabu”
  Za dużo zakazanych ruchów!
- Poziom aspiracji – akceptacja atrakcyjnych ruchów zakazanych
  Algorytm przeszukiwania „tabu”
- Czasochłonne sprawdzanie całego sąsiedztwa w każdym kroku?
  „Kandydaci” – podzbiór sąsiedztwa $V \subset N$

Przeszukiwanie „tabu”

- główna idea – wykorzystanie pamięci

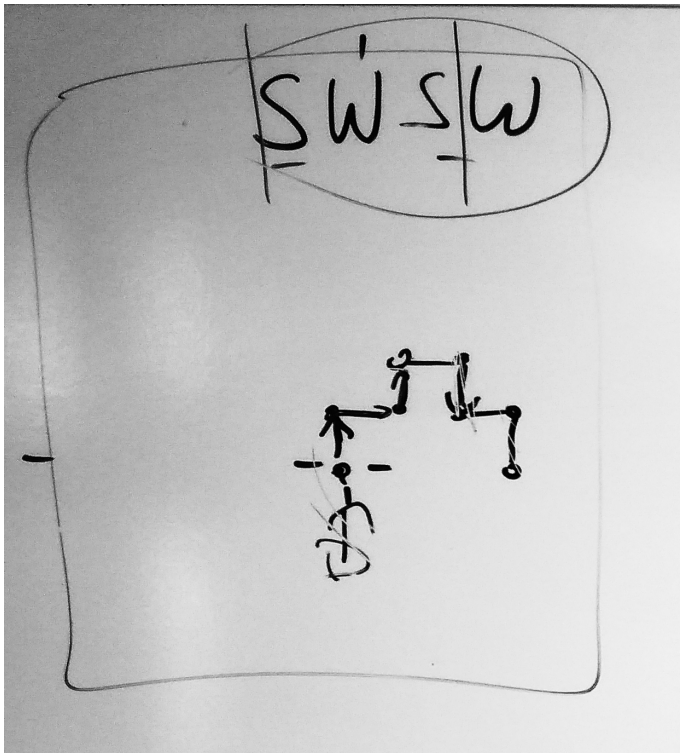
Przeszukiwanie „tabu”

- główna idea – wykorzystanie pamięci



Przeszukiwanie „tabu”

- główna idea – wykorzystanie pamięci



- zapamiętywanie rozwiązań lub ruchów (zmian)

Przykład zapamiętywania ruchów „tabu” (0)

Struktura listy tabu:

	2	3	4	5	6	7
1						
2						
3						
4						
5						
6						

Wewnątrz numer kadencji (ilość iteracji tabu do końca).

Przykład zapamiętywania ruchów „tabu” (0)

Struktura listy tabu:

	2	3	4	5	6	7
1						
2						
3						
4						
5						
6						

Wewnątrz numer kadencji (ilość iteracji tabu do końca).

Iteracja 0 (punkt startowy, zadanie maksymalizacji)

2	5	7	3	4	6	1

Rozwiązanie bieżące – wartość=10

	2	3	4	5	6	7
1						
2						
3						
4						
5						
6						

ruch		Δ
5	4	6
7	4	4
3	6	2
2	3	0
4	1	−1

Przykład zapamiętywania ruchów „tabu” (1, 2)

Iteracja 1

2	4	7	3	5	6	1

Rozwiązanie bieżące – wartość=16

	2	3	4	5	6	7
1						
2						
3						
4				3		
5						
6						

ruch		Δ
3	1	2
2	3	1
3	6	−1
7	1	−2
6	1	−4

Przykład zapamiętywania ruchów „tabu” (1, 2)

Iteracja 1

2	4	7	3	5	6	1

Rozwiązanie bieżące – wartość=16

	2	3	4	5	6	7
1						
2						
3						
4			3			
5						
6						

ruch		Δ
3	1	2
2	3	1
3	6	−1
7	1	−2
6	1	−4

Iteracja 2

2	4	7	1	5	6	3

Rozwiązanie bieżące – wartość=18

	2	3	4	5	6	7
1		3				
2						
3						
4			2			
5						
6						

	ruch		Δ
T	1	3	−2
	2	4	−4
	7	6	−6
T	4	5	−7
	5	3	−9

Przykład zapamiętywania ruchów „tabu” (3, 4)

Iteracja 3

4	2	7	1	5	6	3

Rozwiązanie bieżące – wartość=14

	2	3	4	5	6	7
1		2				
	2		3			
		3				
			4	1		
				5		
					6	

	ruch		Δ
T	4	5	6
	5	3	2
	7	1	0
	1	3	−3
	2	6	−6

Przykład zapamiętywania ruchów „tabu” (3, 4)

Super!

Iteracja 3

4	2	7	1	5	6	3

Rozwiązanie bieżące – wartość=14

	2	3	4	5	6	7
1		2				
	2		3			
		3				
			4	1		
				5		
					6	

	ruch		Δ
T	4	5	6
	5	3	2
	7	1	0
	1	3	−3
	2	6	−6

Przykład zapamiętywania ruchów „tabu” (3, 4)

Super!

Iteracja 3

4	2	7	1	5	6	3

Rozwiązanie bieżące – wartość=14

	2	3	4	5	6	7
1		2				
2			3			
3						
4				1		
5						
6						

	ruch		Δ
T	4	5	6
	5	3	2
	7	1	0
	1	3	−3
	2	6	−6

Iteracja 4

5	2	7	1	4	6	3

Rozwiązanie bieżące – wartość=20

	2	3	4	5	6	7
1		1				
2			2			
3						
4				3		
5						
6						

ruch		Δ
7	1	0
4	3	−3
6	3	−5
5	4	−6
2	6	−8

Recency-based memory vs. frequency-based memory

Częstość wystąpień poszczególnych ruchów może być dodatkowo wykorzystana do większego rozproszenia przeszukiwania w przestrzeni rozwiązań dopuszczalnych (dywersyfikacja). Na przykład: ruchy otrzymują proporcjonalną do ich częstości karę w przypadku, gdy nie polepszają wartości rozwiązania.

Dywersyfikacja jest przydatna tylko w określonych warunkach (np. gdy nie ma żadnych polepszeń).

Iteracja 26. $\Delta' = \Delta - \text{kara_za_częstość}$

5	2	7	1	4	6	3

	1	2	3	4	5	6	7
1	·			3			
2		·					1
3	3		·				
4	2	5		·	2		
5		4		4	·		
6					1	·	
7	2			3			·

	ruch		Δ	Δ'
T	1	4	3	3
	2	4	-1	-6
	3	7	-3	-3
	1	6	-5	-5
	6	5	-6	-7

Przeszukiwanie tabu

```
procedure PRZESZUKIWANIE_TABU
begin
    INICJALIZUJ(xstart, xbest, T)
    x := xstart
    repeat
        GENERUJ(V ⊂ N(x))
        WYBIERZ(x')      //najlepsze f w V + aspiracja
        UAKTUALNIJ_LISTĘ_TABU(T)
        if f(x') ≤ f(xbest) then xbest := x'
        x := x'
    until WARUNEK_STOPU
end
```

Determinizm

Algorytm jest deterministyczny.

- autor TS: „zły wybór strategiczny jest lepszy, niż dobry wybór losowy” (bo jest pod kontrolą, więc można ocenić strategię i wyciągnąć wnioski)

Nowość: lista „kandydatów” – zbiór V

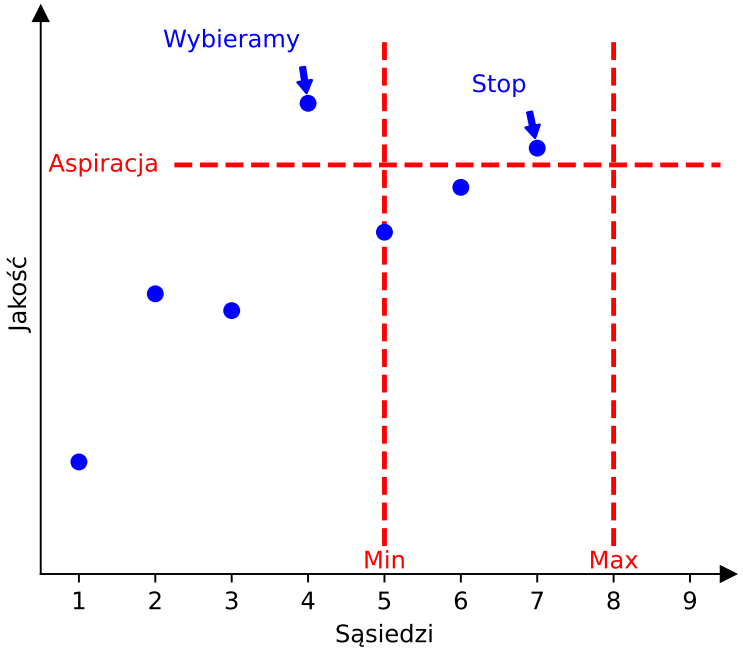
- po co: żeby w każdej iteracji nie generować i nie oceniać całego sąsiedztwa
- dobry ruch, jeśli nie jest wykonany w bieżącej iteracji, będzie wciąż dobry w kilku następnych (?)

Nowość: lista „kandydatów” – zbiór V

- po co: żeby w każdej iteracji nie generować i nie oceniać całego sąsiedztwa
- dobry ruch, jeśli nie jest wykonany w bieżącej iteracji, będzie wciąż dobry w kilku następnych (?)
- jaki podzbiór $V \subset N$ zbioru sąsiadów N ma być kandydatami?
  - kandydaci = dobrzy sąsiedzi
  - należy wybrać takie ruchy, które są korzystne... dla obecnego rozwiązania i dla przyszłych.

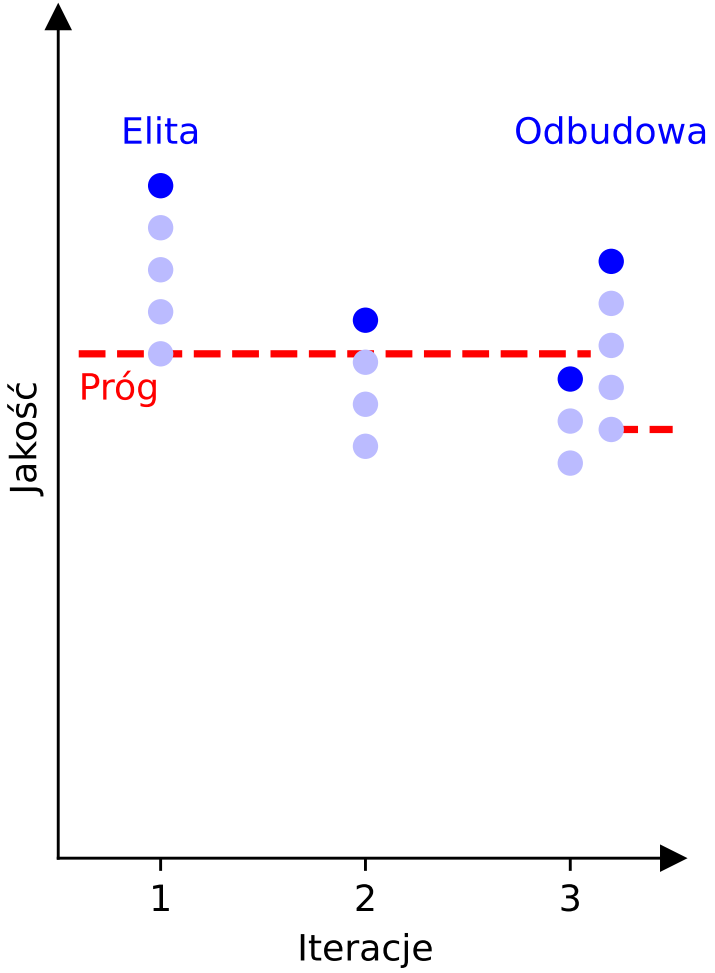
Konstrukcja listy kandydatów. Idea strategii 1: „Aspiracja plus”

- przeszukiwanie sąsiedztwa aż do znalezienia sąsiada lepszego o pewną wartość progową („aspiracja plus”)
- liczba kandydatów rośnie aż do osiągnięcia wartości progowej
- Min $\leq$ liczba sprawdzonych sąsiadów $\leq$ Max
- poziom aspiracji może być zmienny w trakcie przeszukiwania (może zależeć od historii przeszukiwania)
- strategia zwraca 1 lub więcej najlepszych znalezionych sąsiadów
- aż 3 parametry...
- szczegóły: [1]



Konstrukcja listy kandydatów.
Idea strategii 2: „Elitarna lista kandydatów”

- aby stworzyć listę, sprawdź wszystkie lub większość ruchów i wybierz k najlepszych (k jest parametrem)
- w następnych iteracjach aktualnie najlepszy ruch z listy jest wybierany do wykonania, aż jakość ruchu spadnie poniżej danego progu, lub zostanie osiągnięta pewna liczba iteracji
- może być adaptacyjne
- szczegóły: [1]



Kryteria aspiracji

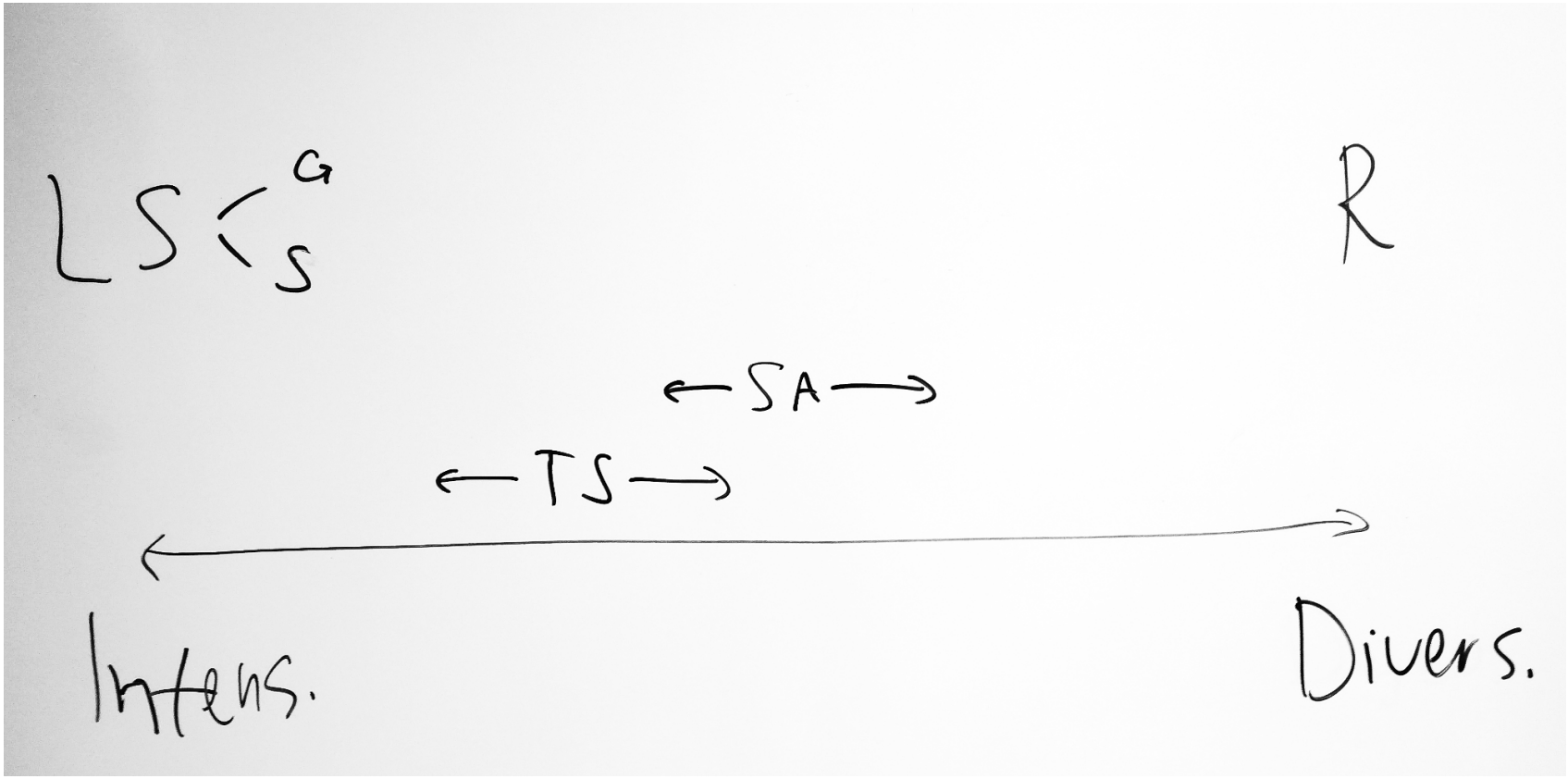
- cel: określić kiedy restrykcje „tabu” mogą zostać pominięte
- proste kryterium aspiracji (uwzględniające funkcję celu, globalne, z przykładu na początku tej prezentacji): usunąć ograniczenie „tabu” wtedy, gdy ruch da rozwiązanie lepsze od najlepszego znalezionego do tej pory
- **aspiracja domyślna**
  - jeżeli wszystkie ruchy są „tabu” i nie są dozwolone przez inne kryteria, to wybierany jest ruch najmniej „tabu”
- **aspiracja uwzględniająca funkcję celu**
  - forma globalna – ruch „tabu” $x \to x'$ jest akceptowany, jeżeli $cost(x') < best_cost$
  - forma regionalna (przestrzeń rozwiązań dzieli się na regiony R) – ruch „tabu” jest akceptowany, jeżeli $cost(x') < best_cost(R)$. R jest regionem, w którym leży x'.

Unifikacja algorytmów optymalizacji

Intensyfikacja i dywersyfikacja (*exploitation and exploration*)

Unifikacja algorytmów optymalizacji

Intensyfikacja i dywersyfikacja (*exploitation and exploration*)



Dyskusja: czy jest możliwa większa intensyfikacja niż LS?
EA? RW? RS z wymuszoną dywersyfikacją?
Czy można obie własności równocześnie polepszyć? 2D?

Intensyfikacja i dywersyfikacja w *Tabu Search*

Intensyfikacja i dywersyfikacja w *Tabu Search*

- intensyfikacja (eksploatacja)
  - w dobrych rejonach
  - powrót do najlepszego rozwiązania znalezionego do tej pory
  - skrócenie listy Tabu *(short term memory)*
  - pamięć długotrwała *(long term memory)*
    - każde rozwiązanie lub ruch to zbiór komponentów
    - zapamiętywanie komponentów dobrych ruchów lub rozwiązań w czasie optymalizacji
    - w czasie intensyfikacji, ruchy lub rozwiązania włączają dobre komponenty
    - pamięć długotrwała daje możliwość „uczenia się"
- dywersyfikacja (eksploracja)
  - dla rzadko odwiedzanych rejonów
  - karanie często wykonywanych ruchów – ucieczka z rejonu
- można postrzegać te mechanizmy jako pewien sposób modyfikacji funkcji celu: $f' = f + Int + Div$

Bibliografia I

Fred Glover, Manuel Laguna, Panos Pardalos, D.-Z. Du, and R. L. Graham.

Tabu search: effective strategies for hard problems in analytics and computational science.

Handbook of Combinatorial Optimization, 2nd ed, vol. XXI, pages 3261–3362, 2013.

doi:10.1007/978-1-4419-7997-1_17.