Algorytmy Geometryczne - Sprawozdanie z laboratorium 2

Szymon Tyburczy 420081 Wtorek 15:00-16:30 B Grupa 7

1.Użyte narzędzia oraz krótki opis.

Rysowanie odcinków które odbywa się przy użyciu myszki, korzysta z funkcji z biblioteki matplotlib.
Wykresy przedstawiające przykładowe i „wygenerowane" odcinki oraz
algorytm zamiatania wraz z wizualizacją poszczególnych jego kroków, powstały przy użyciu biblioteki matplotlib oraz dzięki narzędziu przygotowanemu przez koło naukowe BIT.
Szczegółowy opis algorytmy wraz z użytymi bibliotekami dostępny jest w dalszej części sprawozdania.

2. Specyfikacja sprzętu na którym dokonywane było ćwiczenie:

Komputer z systemem windows 11x64
Procesor: 13th Gen Intel(R) Core(TM) i7-10700KF 3.8GHz
RAM: 16GB DDR4 3600Mhz
Procesor Graficzny: NVIDIA RTX 3080

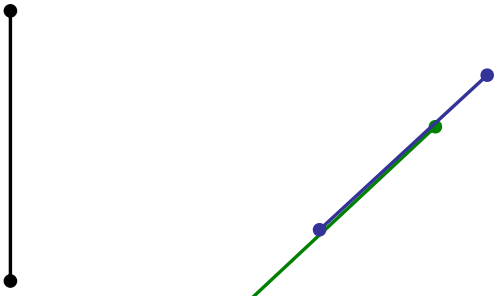
3. Cel ćwiczenia oraz teoria stojąca za implementacją

Celem ćwiczenia jest zrozumienie oraz implementacja algorytmu zamiatania, który wyznacza przecięcia odcinków.
Implementujemy dwa rodzaje: sprawdzanie, czy jakakolwiek para odcinków się przecina, oraz obliczanie wszystkich punktów przecięć odcinków, a także wizualizację wyników.

Oczywistym algorytmem sprawdzającym przecięcia odcinków jest algorytm brutalny o złożoności $O(n^2)$. Dla n odcinków może być w najgorszym przypadku $n(n-1)/2=O(n^2)$ punktów przecięcia. Wtedy prosty algorytm siłowy może zbadać każdą parę odcinków i wyznaczyć punkt przecięcia w czasie $O(n^2)$. Jeżeli punktów przecięcia jest rzędu O(n), chcielibyśmy mieć szybszy algorytm. Istnieje oczywiście alternatywa w postaci algorytmu zwanego sweeping algorithm, który jest "wrażliwy na przecięcia".
Algorytm zamiatania polega na wykorzystaniu "miotły" oraz odpowiedniej struktury stanu i struktury zdarzeń. W naszym przypadku miotła jest daną prostą równoległą do osi Y.
Aby implementacja algorytmu omówionego na wykładzie była możliwa, należy poczynić odpowiednie założenia.

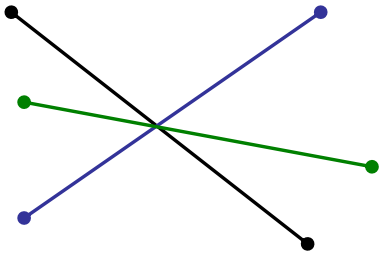
- Nie może istnieć punkt, w którym przecinają się trzy lub więcej odcinki.
- Dwa losowe odcinki mogą się przecinać dokładnie raz albo wcale – oznacza to, że nie ma możliwości, aby odcinki w jakikolwiek sposób na siebie nachodziły, niezależnie od tego, czy w pełni, czy częściowo.
- Nie istnieje odcinek całkowicie pionowy względem osi X, czyli taki, który jest równoległy zarówno do miotły, jak i do osi Y.
- Nie istnieje odcinek, którego końcowa współrzędna x pokrywa się z końcem innego odcinka.

Sytuacje zabronione widoczne na rys 3.1, 3.2 oraz 3.3



rys 3.1

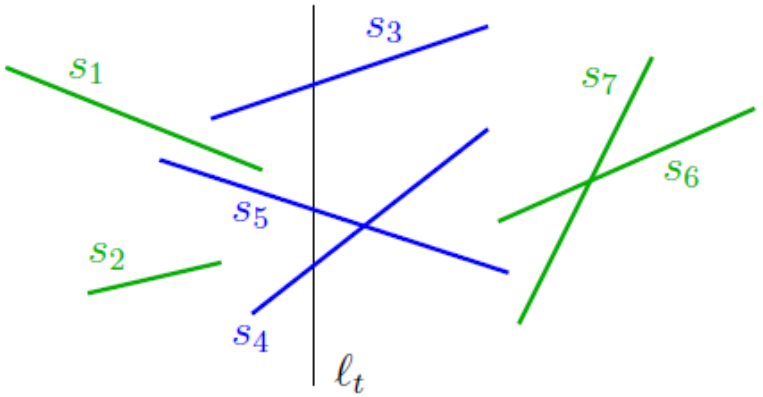
rys 3.2



rys 3.3

Zamiatanie odbywa się w prawą stronę układu współrzędnych, wzdłuż osi X, czyli w stronę rosnących wartości x.

W każdym położeniu miotły odcinki mogą być w trzech stanach: przetworzone, aktywne lub oczekujące. Stanem miotły jest zbiór odcinków aktualnie przecinających miotłę.



1) Odcinki s1 oraz s2 to odcinki przetworzone.
2) Odcinki s3, s4, s5 są w trakcie przetwarzania.
3) Odcinki s6 oraz s7 są odcinakmi oczekującymi.

Miotła zatrzymuje się w sytuacjach, gdy odcinek kończy się, zaczyna lub przecina z sąsiednim odcinkiem. W takich momentach aktualizujemy stan miotły i przeprowadzamy odpowiednie testy.
Algorytm korzysta z dwóch głównych struktur danych:

- Struktura zdarzeń (Q) – zawiera zdarzenia uporządkowane rosnąco względem współrzędnych x, a także punkty przecięć wszystkich par odcinków aktywnych, które kiedykolwiek były sąsiadami w tej strukturze.
- Struktura stanu (T) – jest to zbiór odcinków aktywnych, uporządkowanych względem współrzędnych y.

Złożoność czasowa algorytmu, przy odpowiednim doborze struktur danych, wynosi O((k+n)logn), gdzie:

- O(nlogn) to czas inicjalizacji struktury Q,
- O((k+n)logn) to czas potrzebny na aktualizację struktury T,
- O(klogn) to czas aktualizacji struktury Q.

W mojej implementacji struktura Q to events, natomiast T w kodzie nazywa się active_segments.

Algorytm oraz jego krótki opis są zawarte w kodzie.
Wykorzystana została biblioteka sortedcontainers, z której zaimportowano klasę SortedSet. SortedSet charakteryzuje się tym, że elementy są zawsze przechowywane w określonej kolejności, zgodnie z naturalnym porządkiem sortowania lub zadanym kluczem. SortedSet jest wydajny, ponieważ operacje takie jak wstawianie, usuwanie i wyszukiwanie elementów mają złożoność czasową O(log n). Dodatkowo, SortedSet umożliwia dostęp do elementów za pomocą indeksów, podobnie jak listy, co pozwala na szybkie pobieranie elementów na podstawie ich pozycji w uporządkowanym zbiorze.
SortedSet automatycznie utrzymuje odcinki w odpowiedniej kolejności, eliminując potrzebę ręcznego sortowania lub utrzymywania dodatkowych struktur danych.

Przy kolorowaniu poszczególne kolory oznaczają różne stany:

- Brązowy - Gdy segment jest usuwany z aktywnej listy segmentów, jego początkowy i końcowy punkt oraz sam segment są oznaczane na brązowo.

- Niebieski - Wszystkie początkowe i końcowe punkty segmentów są zaznaczane na niebiesko. Dzięki temu łatwo zidentyfikować wszystkie segmenty na płaszczyźnie. Segmenty w tym kolorze czekają na przetworzenie.

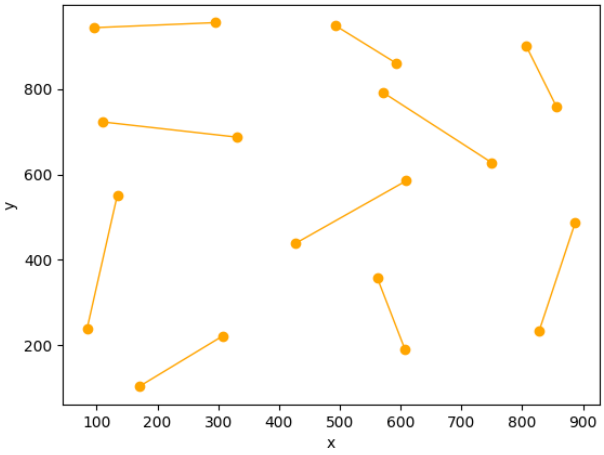
- Żółty - Używany do podświetlania par segmentów, które są aktualnie sprawdzane pod kątem przecięcia.

- Zielony - Kiedy segment jest dodawany do aktywnej listy segmentów (ang. active segments), jego początkowy i końcowy punkt oraz sam segment są zaznaczane na zielono.

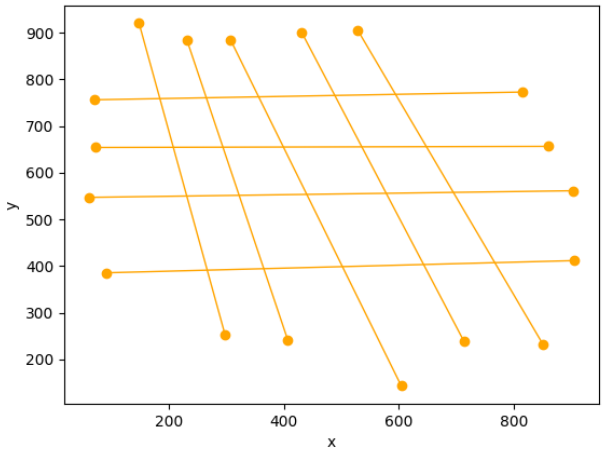
- Czerwony - Punkty przecięcia: Gdy zostaje wykryty punkt przecięcia dwóch segmentów, jest on oznaczany na czerwono, co pozwala na łatwe zlokalizowanie miejsc przecięć. Linia miotły:

- Czerwony kolor jest używany do rysowania pionowej linii miotły, która przesuwa się od lewej do prawej w trakcie działania algorytmu. Linia ta reprezentuje aktualną pozycję skanowania na osi X.

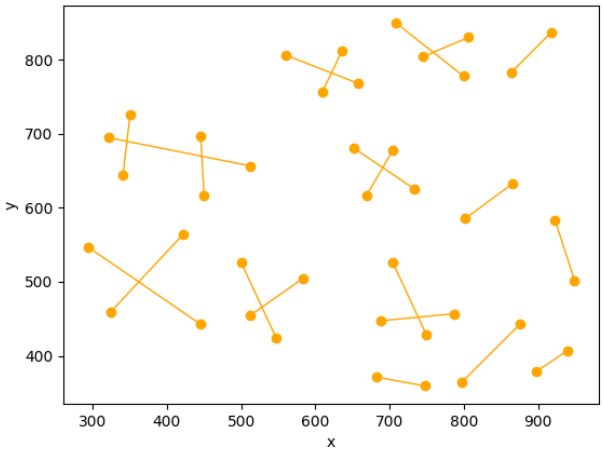
Wygenerowałem 5 zbiorów do testów. Każdy z nich ma nieco inną specyfikację – różnią się odcinkami i ich gęstością, kątami względem osi X, liczbą przecięć oraz rozmieszczeniem w układzie.



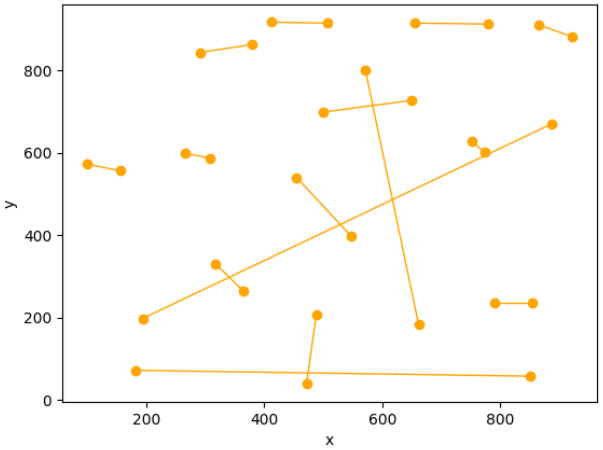
Rys 4.1 Zbiór 1



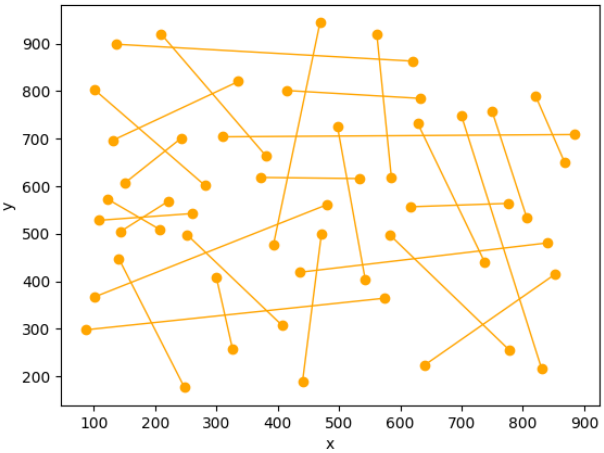
Rys 4.2 Zbiór 2



Rys 4.3 Zbiór 3



Rys 4.4 Zbiór 4



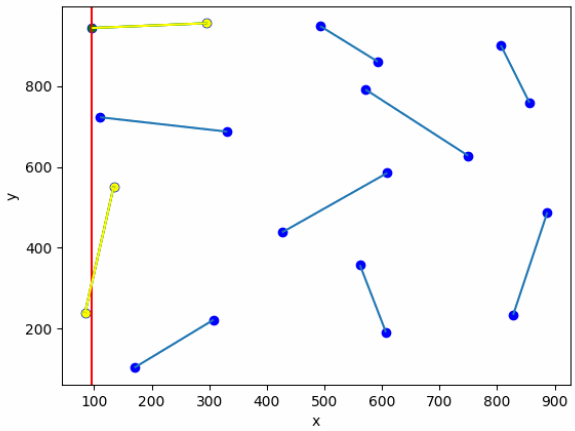
Rys 4.5 Zbiór 5

Algorytm zgodnie z oczekiwaniami poprawnie określa, czy występuje jakiekolwiek przecięcie w danym zbiorze.

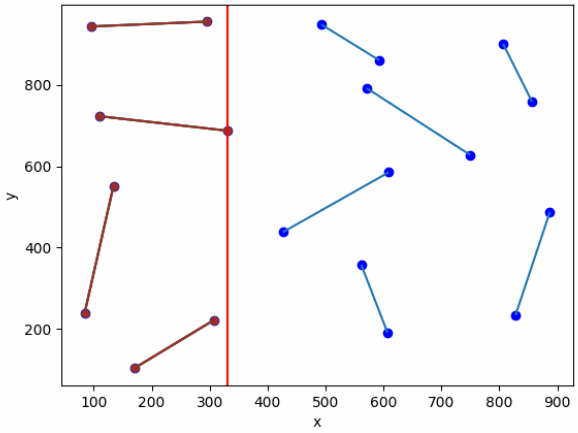
- W zbiorze 1 nie zostało wykryte żadne przecięcie.
- W zbiorze 2 przecięcie zostało poprawnie wykryte.
- W zbiorze 3 przecięcie zostało poprawnie wykryte.
- W zbiorze 4 przecięcie zostało poprawnie wykryte.
- W zbiorze 5 przecięcie zostało poprawnie wykryte.

5. Zbiór 1 - zwizualizowane etapy algorytmu zamiatania

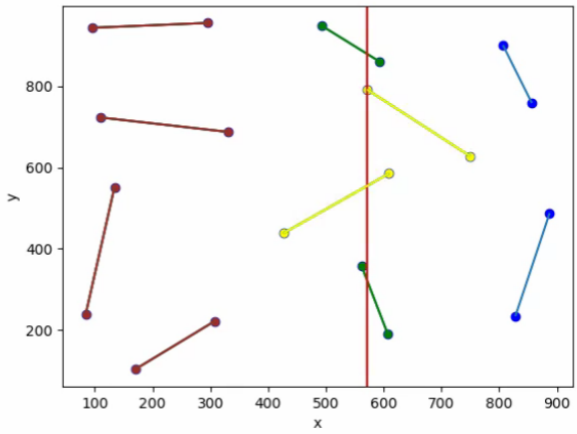
Zbiór 1 charakteryzuje się brakiem jakichkolwiek przecięć oraz średnią gęstością położenia odcinków.



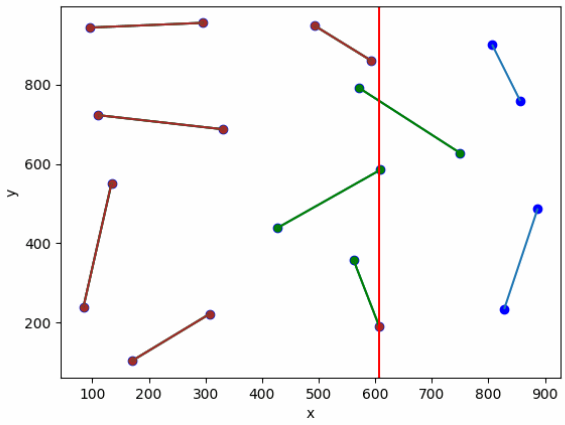
Rys 5.1



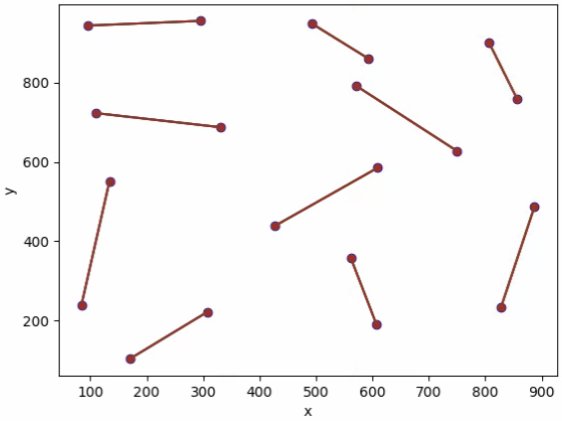
Rys 5.2



Rys 5.3



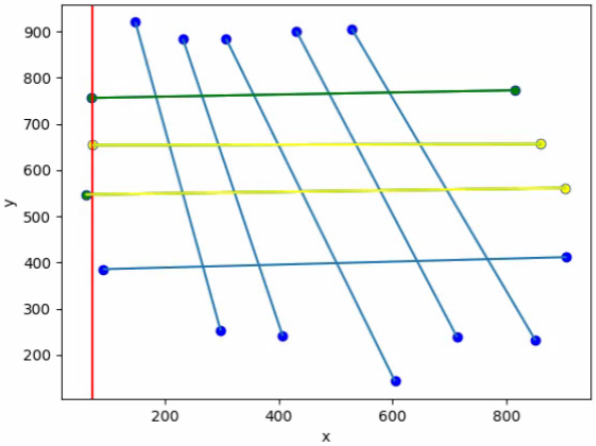
Rys 5.4



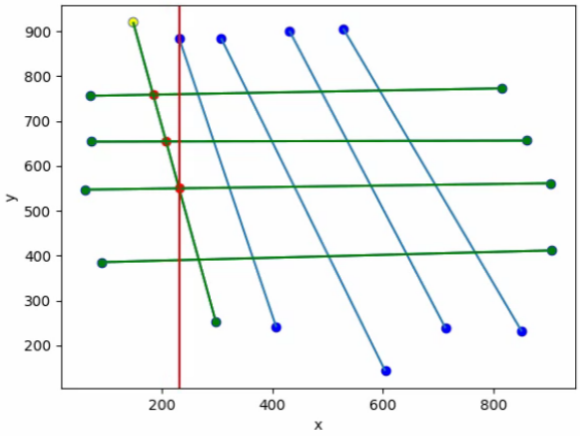
Rys 5.5 - Zbiór 1 całkowicie zamieciony

6. Zbiór 2 - zwizualizowane etapy algorytmu zamiatania

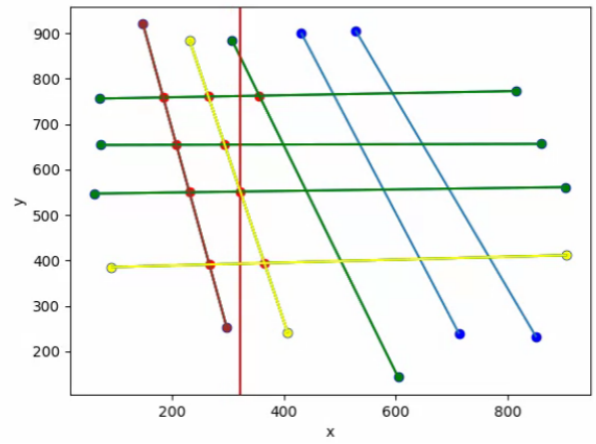
Zbiór 2 charakteryzuje się systematycznie pojawiającymi się przecięciami oraz średnią gęstością położenia odcinków.



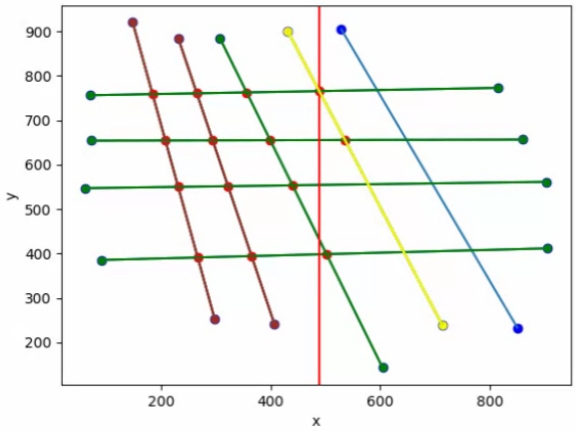
Rys 6.1



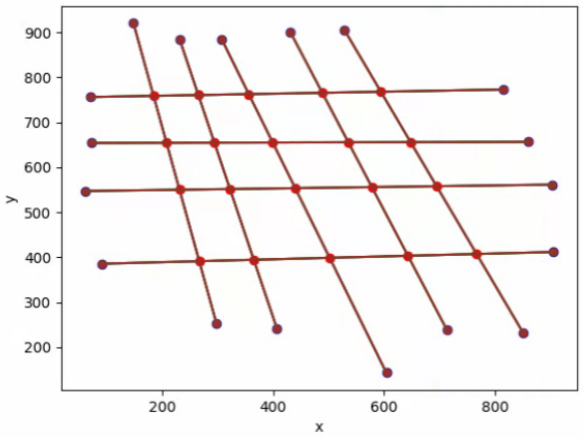
Rys 6.2



Rys 6.3



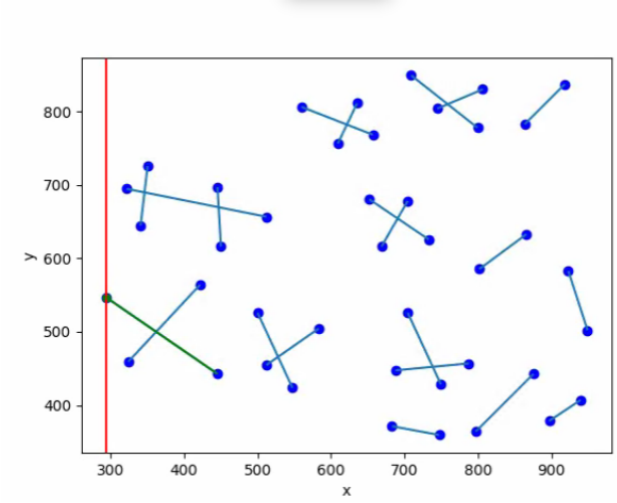
Rys 6.4



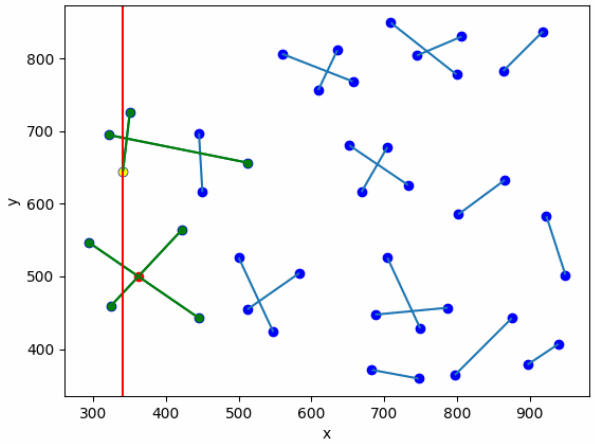
Rys 6.5 - Zbiór 1 całkowicie zamieciony

7. Zbiór 3 - zwizualizowane etapy algorytmu zamiatania

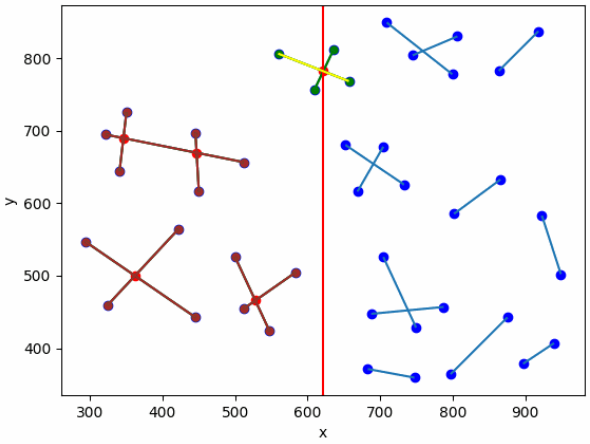
Zbiór 3 charakteryzuje się przecięciami praktytcznie tylko pojedynych przecięć oraz średnią gęstością położenia odcinków.



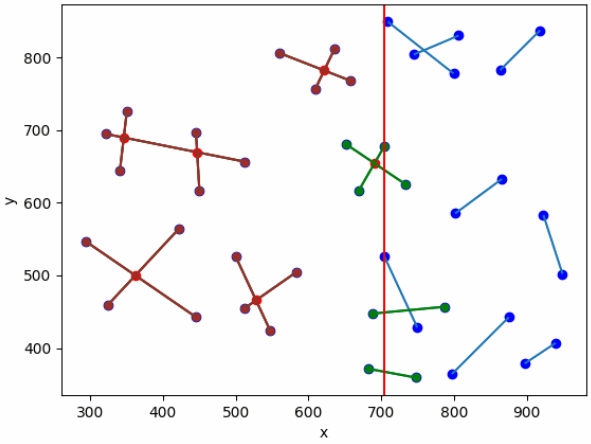
Rys 7.1



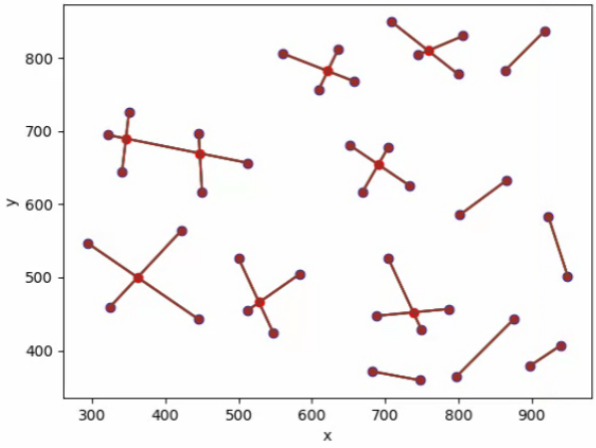
Rys 7.2



Rys 7.3



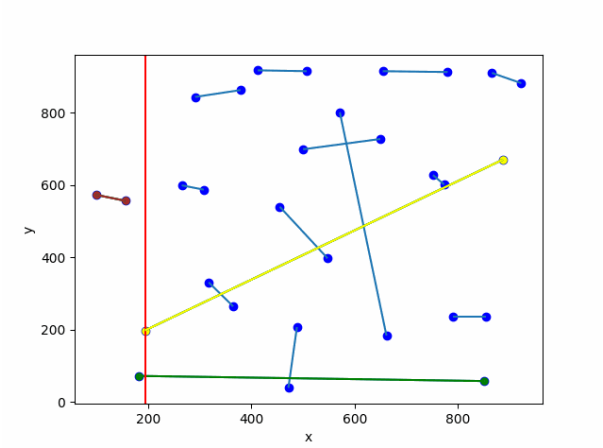
Rys 7.4



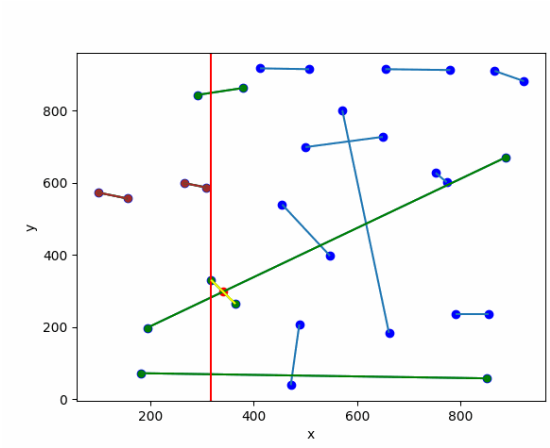
Rys 7.5 - Zbiór 1 całkowicie zamieciony

8. Zbiór 4 - zwizualizowane etapy algorytmu zamiatania

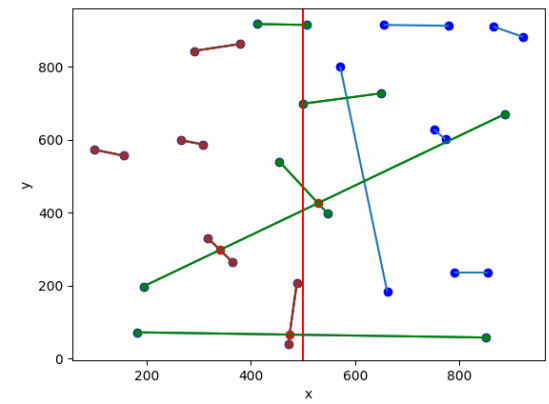
Zbiór 4 charakteryzuje bardzo małą liczbą przecięć wraz ze średnią gęstością położenia odcinków.



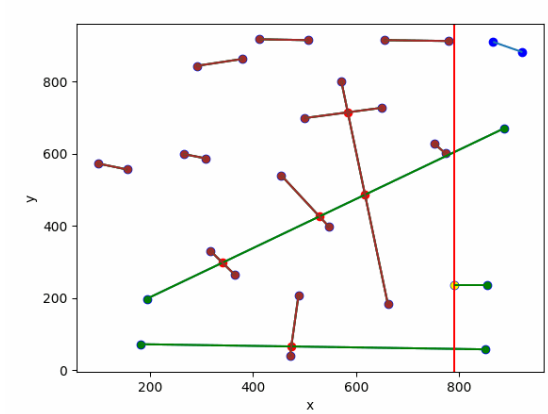
Rys 8.1



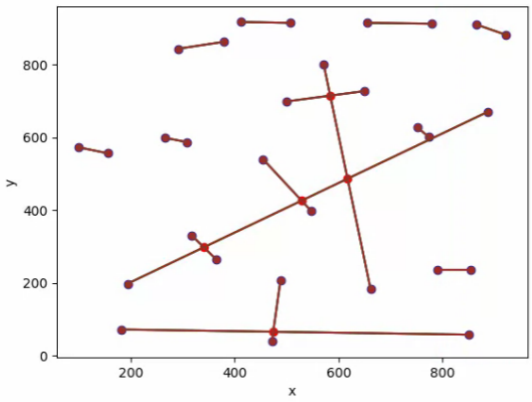
Rys 8.2



Rys 8.3



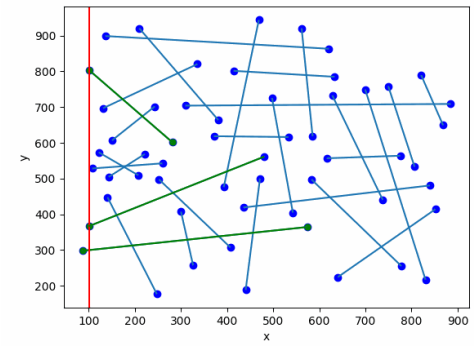
Rys 8.4



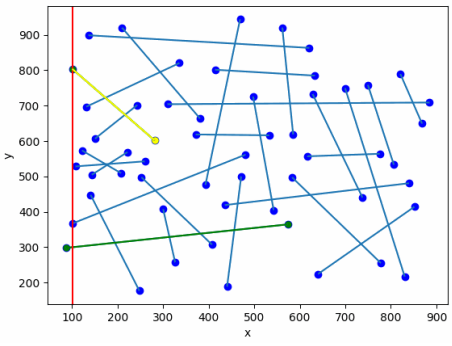
Rys 8.5 - Zbiór 3 całkowicie zamieciony

9. Zbiór 5 - zwizualizowane etapy algorytmu zamiatania

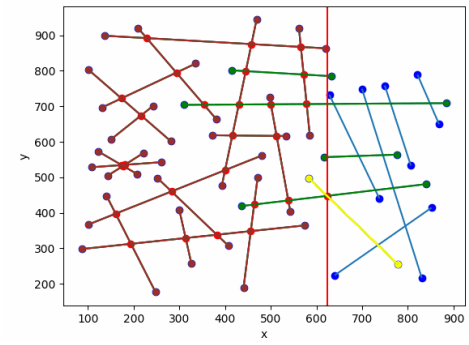
Zbiór 5 charakteryzuje bardzo dużą liczbą przecięć wraz z dużą gęstoścIa położenia odcinków.



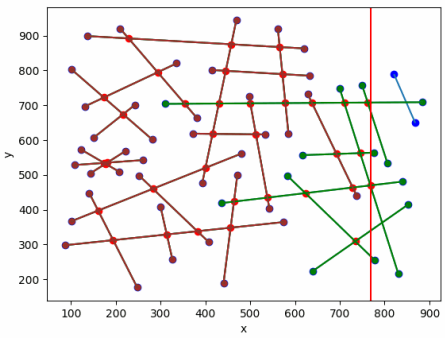
Rys 9.1



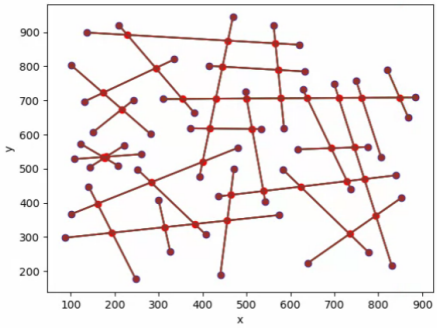
Rys 9.2



Rys 9.3



Rys 9.4



Rys 9.5 - Zbiór 5 całkowicie zamieciony

10. Wnioski

Wszystkie zbiory zostały przygotowane w taki sposób, aby móc przetestować algorytm zamiatania w różnych warunkach. Przygotowane zbiory miały testować poprawność algorytmu dla małej oraz dużej liczby przecięć w różnych konfiguracjach. Algorytm pomyślnie przeszedł testy zamieszczone w templatce stworzonej przez koło naukowe oraz poprawnie przetworzył zbiory, które przygotowałem, co można zobaczyć w zamieszczonych wizualizacjach. Algorytm zamiatania bazuje na tym przedstawionym na wykładzie, co oznacza, że jego złożoność wynosi O((n+k)logn). Jest to istotne, ponieważ dla zbiorów, w których liczba przecięć jest bliska n^2, algorytm ten przestaje być opłacalny i staje się wolniejszy od klasycznego brute-force o złożoności $O(n^2)$.