

# **Wykrywanie i śledzenie cyklu komórkowego prątków w programie ImageJ**

(Mycobacterial cell cycle detection and tracking in ImageJ)

Artur Rosa

Praca magisterska

**Promotor:** dr Andrzej Łukaszewski

Uniwersytet Wrocławski  
Wydział Matematyki i Informatyki  
Instytut Informatyki

14 sierpnia 2020



## **Streszczenie**

Polskie streszczenie

---

English abstract



# Spis treści

<b>1. Wprowadzenie</b>	<b>7</b>
1.1. Wstęp . . . . .	7
1.2. Opis zagadnienia . . . . .	7
1.3. Cel pracy . . . . .	9
<b>2. Wykrywanie i śledzenie komórek</b>	<b>11</b>
2.1. Opis problemu . . . . .	11
2.1.1. Obrazy wejściowe . . . . .	11
2.1.2. Pożądany efekt . . . . .	11
2.2. Powiązane prace . . . . .	12
2.3. Wykrywanie komórek . . . . .	13
2.3.1. Wstęp . . . . .	13
2.3.2. Dane wejściowe . . . . .	13
2.3.3. Wybór kanału i wstępne przetwarzanie obrazu . . . . .	13
2.3.4. Szkieletyzacja i wstępna detekcja komórek . . . . .	15
2.3.5. Wybór krawędzi w węzłach . . . . .	16
2.3.6. Rozwiązywanie konfliktów . . . . .	17
2.3.7. Uzyskiwanie linii łamanej . . . . .	19
2.3.8. Korekta końcówek . . . . .	19
2.4. Śledzenie komórek w czasie . . . . .	20
2.5. Interakcja ze strony użytkownika . . . . .	21
<b>3. Opis implementacji</b>	<b>23</b>

3.1.	Kod źródłowy . . . . .	23
3.2.	Kompilacja i uruchomienie . . . . .	23
3.3.	Obsługa pluginu . . . . .	24
3.3.1.	Lokalizacja i wstępna detekcja komórek . . . . .	25
3.3.2.	Śledzenie komórek w czasie . . . . .	25
3.3.3.	Analiza danych . . . . .	26
3.3.4.	Manualna modyfikacja komórek . . . . .	27
3.3.5.	Nazewnictwo komórek oraz ich orientacja . . . . .	29
3.3.6.	Importowanie i eksportowanie komórek . . . . .	30
3.4.	Dokumentacja techniczna . . . . .	30
3.4.1.	Struktura projektu . . . . .	30
3.4.2.	Główne struktury danych . . . . .	31
3.4.3.	Architektura umożliwiająca dalszy rozwój . . . . .	32
3.4.4.	Wykorzystywane API ImageJ . . . . .	35
3.4.5.	Brakujące lub niespójne API ImageJ . . . . .	36
<b>4.</b>	<b>Zakończenie</b>	<b>39</b>
4.1.	Podsumowanie . . . . .	39
4.2.	Ograniczenia zastosowanych metod . . . . .	40
4.3.	Dalszy rozwój . . . . .	40
<b>Bibliografia</b>		<b>43</b>

## Rozdział 1.

# Wprowadzenie

### 1.1. Wstęp

Prątki, czyli bakterie z rodzaju *Mycobacterium*, znane są przede wszystkim za sprawą patogennych gatunków m.in. *Mycobacterium tuberculosis* i *Mycobacterium leprae*, które powodują odpowiednio gruźlicę i trąd. Choroby te, a w szczególności gruźlica, są wciąż poważnym problemem w wielu miejscach świata. Według światowej organizacji zdrowia, na całym świecie gruźlica jest jedną z 10 głównych przyczyn zgonów, a zarazem główną przyczyną spowodowaną przez pojedynczy czynnik zakaźny[1]. Szacuje się, że w samym 2018 roku na całym świecie zachorowało na gruźlicę około 10 milionów ludzi.

Niniejsza praca powstała przy współpracy z pracownikami naukowymi Zakładu Mikrobiologii Molekularnej na Wydziale Biotechnologii Uniwersytetu Wrocławskiego. Badania przez nich prowadzone, są w dużej mierze tzw. badaniami podstawowymi, a więc ich celem jest przede wszystkim zdobywanie nowej wiedzy i poznawanie biologii prątków. Badania tego typu są często wykorzystywane w dalszych analizach, skierowanych już na konkretne zastosowania praktyczne takie jak np. poszukiwanie nowych leków.

### 1.2. Opis zagadnienia

Chcąc dokładniej poznać naturę mikroorganizmów z rodziny *Mycobacteriaceae*, badacze analizują parametry różnych procesów cyklu komórkowego. Takie analizy pozwalają między innymi na określenie wpływu różnych inhibitorów/antybiotyków na cykl komórkowy[2]. Zebrane dane mogą posłużyć także do określenia funkcji wybranego białka[3]. Badanie to polega na modyfikacji szczepu poprzez np. usunięcie genu, który je koduje i analizę różnych procesów komórkowych w takim szczepie. Następnie sprawdza się między innymi czy cykl komórkowy bądź morfologia pojedynczych komórek nie zostały u nich zaburzone.

Ze względu na zróżnicowanie populacji bakteryjnych, ważne jest, aby analizy przeprowadzane były na poziomie pojedynczych komórek. Podejście takie pozwala na odkrycie zależności, które występują tylko dla części populacji. Przykładowo część komórek z tego samego szczezu może inaczej reagować na podany inhibitor.

Istnieje wiele parametrów poszczególnych procesów cyklu komórkowego, których pomiary mogą być przydatne w badaniach nad biologią prątków. Takimi parametrami mogą być np. czas replikacji czy segregacji chromosomów potomnych (procesy powielania DNA chromosomalnego i segregacji nowo zreplikowanego DNA do komórek potomnych), tempo i charakterystyka zmian długości komórek, czy lokalizacja wewnętrzkomórkowa niektórych białek.

### Komórki modelowe

Jednym z organizmów modelowych w badaniach nad biologią prątków są komórki *Mycobacterium smegmatis*. W przeciwieństwie do ich bardziej znanych krewnych *Mycobacterium tuberculosis*, mikroorganizmy te nie są chorobotwórcze, dzięki czemu badania z ich udziałem są bezpieczniejsze i tym samym łatwiejsze do przeprowadzenia. W przypadku obu tych gatunków podstawowe procesy komórkowe wyglądają podobnie. Kolejną ważną cechą jest ich czas podziału, który jest 8 razy krótszy niż czas podziału komórek wywołujących gruźlicę i wynosi około 3 godzin.

W celu śledzenia różnych procesów cyklu komórkowego wykorzystuje się fluorescencyjne szczepy reporterowe, które produkują białka zaangażowane w procesy replikacji czy segregacji lub białka umożliwiające wizualizację chromosomu tych bakterii w fuzji z białkami fluorescencyjnymi tj. białka fuzyjne.

### Pozyskiwanie danych

Do obserwacji komórek modelowych badacze z Wydziału Biotechnologii Uniwersytetu Wrocławskiego używają mikroskopu wyposażonego w zestaw filtrów umożliwiający wizualizację różnych białek fluorescencyjnych. Podczas eksperymentu obserwowane komórki naświetlane są falami o odpowiedniej długości. Aminokwasy wewnętrz danego białka fluorescencyjnego tworzące fluorofory, absorbują promieniowanie elektromagnetyczne o określonej długości, a następnie emitują promieniowanie o niższej energii. Na zarejestrowanych w ten sposób obrazach widoczne są tzw. „ogniska fluorescencji”. Są to makrokompleksy jakie tworzą białka fuzyjne z DNA chromosomalnym bakterii.

Oprócz fluorescencji, która obrazuje wewnętrzną strukturę komórki, obserwuje się także ich zewnętrzną strukturę oświetloną światłem widzialnym. W celu zwiększenia kontrastu takich obrazów stosuje się układy z kontrastem różnicowo-interferencyjnym Nomarskiego[4] (DIC, ang. differential interference contrast).

Przykładowy eksperyment przeprowadzany jest przez 20h w komorze, która umożliwia utrzymanie optymalnej dla komórek modelowych temperatury ( $37^{\circ}\text{C}$ ). Co 10 min wykonywane jest zdjęcie. Tak pozyskane nagranie badacze obrabiają przy użyciu programu Fiji[5]. Jest to dystrybucja otwartego oprogramowania ImageJ służącego do przetwarzania obrazów. Wyposażona jest w wiele wtyczek ułatwiających naukową analizę obrazu. Przy jej pomocy zbierane są także dane dotyczące poszczególnych komórek na poszczególnych klatkach nagrania, takie jak długość czy umiejscowienie „ognisk fluorescencji”. W tym celu pracownicy naukowi ręcznie oznaczają mierzoną komórkę na obrazie, aby następnie skorzystać z odpowiednich narzędzi programu dokonujących pomiarów. Pomiarów są zapisywane, a proces powtarzany jest dla pozostałych komórek widocznych w danej klatce, a następnie dla pozostałych klatek nagrania. Kolejną częścią procesu jest analiza zbiorczych danych w programie RStudio[6]. Tam dokonywane są analizy statystyczne czy wizualizacje wyników, które następnie wykorzystywane są w pracach naukowych.

Podczas wywiadu z badaczami Wydziału Biotechnologii dowiedziałem się, że istnieją różne narzędzia do automatycznej analizy danych mikroskopowych, ale są one tworzone pod konkretne gatunki bakterii. Populacje prątków są bardzo zróżnicowane, przez co ciężko stworzyć ujednolicony algorytm do ich rozpoznawania. Dodatkowym utrudnieniem jest specyficzna budowa ściany komórkowej bakterii *Mycobacterium smegmatis*, która powoduje, że po podziale, komórki te bardzo długo są do siebie „przyklejone”.

### 1.3. Cel pracy

Aktualny proces zbierania danych z nagrania jest żmudny i dodatkowo wymaga pracy z wieloma narzędziami jednocześnie. Idealnym rozwiązaniem byłaby pełna automatyzacja procesu, ale dużym usprawnieniem mogłoby być również podzielenie tego procesu na dwa etapy:

- zaznaczanie komórek
- dokonywanie pomiarów.

W aktualnym procesie każda komórka zaznaczana jest osobno, dokonywane i zapisywane są dla niej pomiary, a następnie proces powtarzany jest dla kolejnych komórek. Możliwość zaznaczenia wszystkich komórek w pierwszym kroku, a następnie dokonania dla nich pomiarów, mogłaby usprawnić cały proces. Przy takim podejściu możliwy staje się także zapis i odczyt wszystkich zaznaczeń, co dałoby możliwość wielokrotnego wykorzystania raz włożonej pracy.

Celem niniejszej pracy jest stworzenie narzędzia, które ułatwiałoby w przyszłości opisany powyżej proces pozyskiwania danych, a tym samym ułatwiało pracę badaczom. Narzędzie to postanowiłem zaimplementować jako wtyczkę do programu

ImageJ, ze względu na dotychczasowe doświadczenia zespołu i biegłość w obsłudze tego programu.

## Rozdział 2.

# Wykrywanie i śledzenie komórek

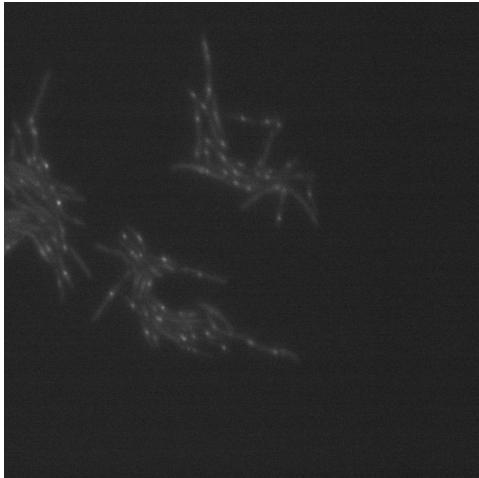
### 2.1. Opis problemu

#### 2.1.1. Obrazy wejściowe

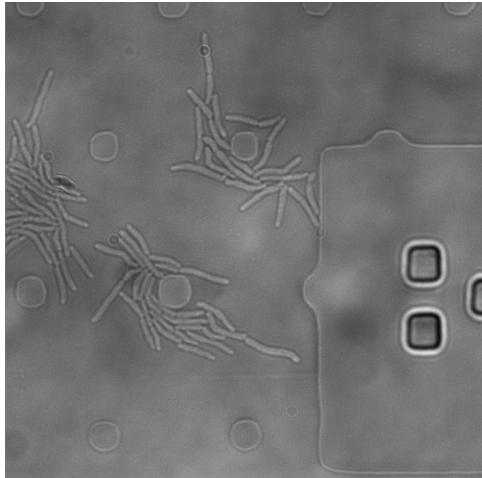
Proces uzyskiwania nagrań przedstawiających cykl komórkowy Mycobacterium smegmatis opisałem w rozdziale 1.2.. Nagrania takie są w istocie stosem obrazów wykonanych w stałych odstępach czasu. Na pojedynczy obraz składają się przynajmniej dwa podstawowe kanały. Pierwszy z nich pozyskany jest za pomocą zjawiska fluorescencji i przedstawia głównie rozmieszczenie białek fuzyjnych. Drugi kanał przedstawia komórki oświetlone światłem widzialnym w kontraste DIC. Pozwala na dokładniejsze zaobserwowanie kształtu i rozmieszczenia poszczególnych komórek. Oba kanały przykładowego kadru nagrania przedstawia rysunek 2.1.

#### 2.1.2. Pożądany efekt

Celem pracy, który został opisany dokładniej w rozdziale 1.3., jest uproszczenie części procesu pozyskiwania danych o cyklu komórkowym prątków z gatunku Mycobacterium smegmatis. Narzędzie stworzone na potrzeby tej pracy ma pomóc badaczom w żmudnym procesie oznaczania komórek na nagraniach. Optymistyczny scenariusz zakłada że program, przy niewielkiej interakcji ze strony użytkownika, oznaczy komórki na wszystkich obrazach, zachowując przy tym informacje na temat pochodzenia każdej z nich. Oznaczenie komórki oznacza odnalezienie krzywej lub linii łamanej przechodzącej przez środek komórki. Taki opis komórki pomoże nie tylko przy analizie zmian długości komórek, ale także pozwoli odczytać profil intensywności takiego oznaczenia z kanału fluorescencji. Dodatkowym atutem byłaby możliwość edycji tego typu zaznaczeń w przypadkach gdy program nie wyznaczył ich poprawnie.



(a) Kanał fluorescencji.



(b) Kanał światła widzialnego w kontraste DIC.

Rysunek 2.1: Dwa kanały tej samej klatki przykładowego nagrania.

Ze względu na słabą jakość nagrań oraz często występujące na obrazach artefakty, automatyczne oznaczanie może nie być możliwe przy niektórych klatkach lub nawet całych nagraniach. W takich przypadkach użytkownik powinien mieć możliwość ręcznego stworzenia oznaczeń.

## 2.2. Powiązane prace

W literaturze opisano wiele metod wykrywania prątków na obrazach wykonanych przy pomocy mikroskopu. W większości dotyczą one jednak wykrywania komórek gruźlicy w celach diagnostycznych. Część takich metod, szczególnie wcześniejszych, opierała się na badaniach z wykorzystaniem fluorescencji[7]. Istnieją także prace wykorzystujące zdjęcia oświetlane światłem widzialnym. Metody te mogą być szczególnie przydatne w miejscach, w których dostęp do mikroskopów fluoresencyjnych jest ograniczony lub wręcz niemożliwy[8]. Celem tego typu metod jest najczęściej segmentacja obrazu na komórki gruźlicy i pozostałe obszary. Efekt takiego działania pozwala na stwierdzenie obecności lub nieobecności komórek na obrazie, a także na określeniu ich zagęszczenia i lokalizacji. O ile metody te niewątpliwie są bardzo ważne w diagnostyce, o tyle niekoniecznie sprawdzają się w przypadku analizy danych na poziomie pojedynczych komórek.

Podobnym zagadnieniem, niekoniecznie w kontekście dziedziny, ale z perspektywy przetwarzania obrazów, jest także wykrywanie nicieni. Jeden z gatunków, *C. elegans*, od wielu lat jest organizmem modelowym w badaniach nauk biologicznych[9]. *C. elegans*, podobnie jak prątki, ma symetryczny tubowy kształt przypominający robaka. Oczywiście jest to zupełnie inny rzęd wielkości. Badania takich bezkręgowców często polega na analizie danych na poziomie pojedynczego

osobnika. W literaturze można znaleźć prace opisujące metody automatycznego oznaczania i opisywania tych nicieni. Ze względu na różnicę w wielkości, dużo łatwiej jest wykonać bardzo dobrej jakości nagranie przedstawiające te żywątko, niż nagranie przedstawiające prątki. Lepsza jakość nagrani znacząco ułatwia ich późniejszą obróbkę. Mimo różnic pomiędzy tymi dziedzinami, praca opisana przez Javiera Fernándeza[10] była w pewnym stopniu inspiracją do zaproponowanej przeze mnie metody.

## 2.3. Wykrywanie komórek

### 2.3.1. Wstęp

Problem opisany w sekcji 2.1. zdefiniowany jest dla nagrani spod mikroskopu. Postanowiłem jednak najpierw rozwiązać podobny problem, ale zdefiniowany dla pojedynczego obrazu. Rozwiązanie tego problemu mogłoby z łatwością zostać uogólnione na stos obrazów (nagranie). W tym rozdziale opiszę rozwiązanie uproszczonego problemu: oznaczanie komórek widocznych na pojedynczym obrazie. Przez „oznaczenie komórki” mam na myśli odnalezienie łamanej przechodzącej przez środek komórki, a więc jej szkieletu.

### 2.3.2. Dane wejściowe

Oznaczenie wszystkich komórek widocznych na obrazie można rozłożyć na dwa osobne problemy:

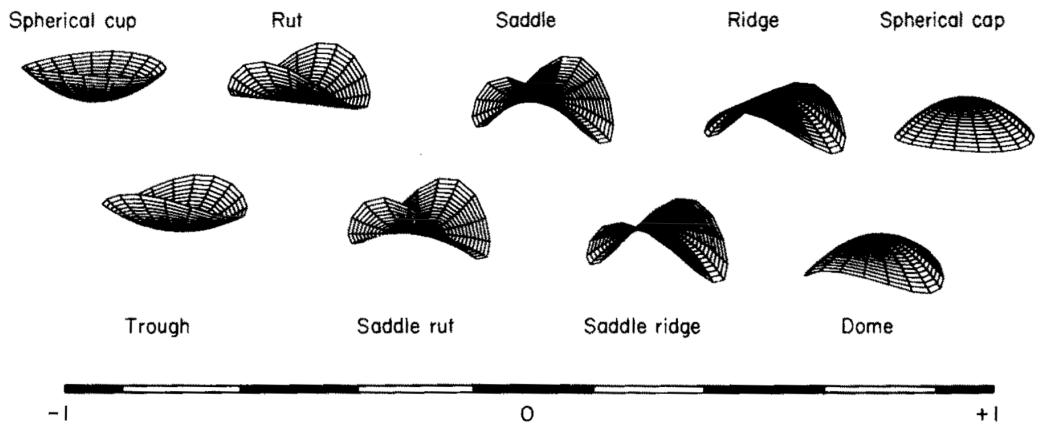
1. Określenie liczby oraz lokalizacji poszczególnych komórek
2. Odnalezienie kształtu poszczególnych komórek.

W niniejszej pracy zdecydowałem się nie rozwiązywać automatycznie pierwszego problemu. Zamiast tego użytkownik zobowiązany jest ręcznie zaznaczyć dokładnie jeden punkt wewnętrzny każdej komórki widocznej na obrazie. Wymóg ten dotyczy tylko pierwszej klatki nagrania, co opiszę dokładniej w dalszej części pracy (2.4.).

Danymi wejściowymi są zatem dwa kanały obrazu **I** oraz zbiór punktów **P** lokalizujących komórki.

### 2.3.3. Wybór kanału i wstępne przetwarzanie obrazu

Obraz wejściowy składa się z dwóch podstawowych kanałów (2.1.1.). Chcąc jak najdokładniej oznaczyć początek i koniec komórki, a także miejsca ich podziału, postanowiłem wybrać kanał, który zawiera wyraźną informację o krawędziach w tych



Rysunek 2.2: Skala indeksu kształtu z podziałem na dziewięć kategorii. [11]

miejscach. O ile kanał z fluorescencją mógłby bardzo dobrze sprawdzić się do określenie liczby oraz lokalizacji poszczególnych komórek (w przypadku ich niewielkiej liczby), o tyle drugi kanał zawiera dużo dokładniejszą informację na temat krawędzi komórek.

Celem wstępnego przetwarzania obrazu wejściowego jest w tym przypadku oddzielenie poszczególnych komórek od otoczenia, zachowując przy tym informację na temat ich krawędzi. W ramach pracy przeprowadziłem wiele testów mających na celu odnalezienie narzędzia spełniającego ten cel. Krawędzie otrzymywane za pomocą filtrów czy algorytmów do ich wyszukiwania często były nieciągłe. Dobierając inne parametry często pojawiały się niechciane zaznaczenia wewnętrz komórek.

Ostatecznie zdecydowałem się użyć narzędzia które nie jest związanego z wykrywaniem krawędzi. Zaobserwowałem, że zdjęcia zrobione tą techniką mają pewne specyficzne właściwości. Komórki są na nich dość równomiernie oświetlone, przez co dobrze widać ich „tubowy” kształt. Każdą komórkę otacza też ciemne obramowanie.

### Mapa indeksów kształtu

Postanowiłem spróbować wyliczyć mapę indeksów kształtu[11] (ang. shape index map) dla obrazu wejściowego, na którym wcześniej zastosowałem rozmycie gaussowskie. Indeks kształtu to liczba z przedziału  $[-1, 1]$  przyporządkowana na podstawie „lokalnego kształtu” powierzchni. Jest to niezmiennicza na skalę (ang. scale invariant) miara, która dzieli powierzchnię na obszary wypukłe, wklęsłe i hiperboliczne (rysunek 2.2). Obraz wejściowy interpretowany jest tutaj jako mapa wysokości. Zastosowanie rozmycia gaussowskiego w pierwszym kroku jest niezbędne w tym przypadku ze względu na nieodporność tej miary na szum obecny na obrazie wejściowym. Do wyliczenia mapy indeksów skorzystałem z wtyczki dla programu ImageJ autorstwa Johanna Schindelina[12].

Okazało się, że na wynikowej mapie otoczenie komórek interpretowane jest jako obszar wklęsły, w przeciwieństwie do samych komórek, których wnętrze oznaczane jest indeksami kształtów wypukłych. Własność ta zachodzi dla większości testowych obrazów nawet przy stosunkowo dużym zagęszczeniu komórek.

#### 2.3.4. Szkieletyzacja i wstępna detekcja komórek

Zgodnie z powyższą obserwacją, możemy łatwo oddzielić komórkę od jej otoczenia na obrazie ustalając pewien próg  $t \approx 0$  dla indeksu kształtu. Założmy przez chwilę, że binaryzując w ten sposób mapę indeksów kształtu otrzymamy obraz  $\mathbf{I}_{bin}$ , na którym każdy piksel leżący wewnątrz dowolnej komórki będzie miał wartość 1, natomiast każdy piksel należący do zewnętrznego obrysu dowolnej komórki (nie należący do komórki, lecz sąsiadujący z pikselem należącym do niej) będzie miał wartość 0. Przy takim założeniu każda komórka jest niezależną „wyspą” na binarnym obrazie  $\mathbf{I}_{bin}$ . Chcąc odnaleźć lamaną przechodzącą przez środek komórki, chcemy tak naprawdę znaleźć lamaną, która jest równoodległa do jej krawędzi. Analogiczny problem rozwiązuje algorytmy do wyznaczania szkieletu. Ich celem jest odnalezienie dla danego kształtu zbioru punktów równoodległych do co najmniej dwóch brzegów.

Na potrzeby tej pracy do szkieletyzacji użyta została implementacja algorytmu „3D thinning algorithm”[13] w formie pluginu dla programu ImageJ[14]. Mimo to że wtyczka pozwala na szkieletyzację obrazów 3D, w tym przypadku została użyta do przetworzenia pojedynczego obrazu 2D. Wynikiem szkieletyzacji jest binarny obraz o pewnych właściwościach. Każdy aktywny piksel można przyporządkować do trzech grup:

- końcówki – mają mniej niż 2 sąsiadujące aktywne piksele
- węzły – mają więcej niż 2 sąsiadujące aktywne piksele
- połączenia – mają dokładnie 2 sąsiadujące aktywne piksele.

Przedstawiając szkielet jako graf, końcówki tworzyłyby wierzchołki o stopniu równym 1 lub 0, wierzchołki o większych stopniach przedstawiałyby zbiory sąsiadujących ze sobą węzłów, natomiast krawędzie reprezentowałyby zbiory sąsiadujących ze sobą połączeń (zakończonych zbiorem węzłów lub końwką). Taką reprezentację grafową można uzyskać za pomocą kolejnej wtyczki dla programu ImageJ tego samego autora[15]. Poza standardowymi informacjami wierzchołki i krawędzie utworzonego za jej pomocą grafu przechowują zbiory pikseli które reprezentują.

Ze względu na specyficzny kształt komórki można zaobserwować następującą właściwość: po przeprowadzeniu szkieletyzacji obrazu  $\mathbf{I}_{bin}$ , o ile przyjęte wcześniej założenie jest spełnione, szkielet każdej z komórek składa się dokładnie z dwóch końcówek i połączeń między nimi. Graf opisujący komórkę będzie zawierał w takim przypadku dokładnie dwa wierzchołki i jedną krawędź łączącą je ze sobą. Mając do

dyspozycji zbiór punktów  $\mathbf{P}$  lokalizujących komórki, można teraz w łatwy sposób odnaleźć dla każdej z nich graf ją opisujący. Jednym ze sposobów może być wyszukiwanie dla każdego punktu ze zbioru  $\mathbf{P}$  krawędzi która znajduje się najbliżej tego punktu, gdzie odległość między punktem a krawędzią zdefiniowana jest jako odległość między punktem, a najbliższym pikselem, który należy do zbioru opisywanego przez tę krawędź.

### 2.3.5. Wybór krawędzi w węzłach

Niestety przyjęte założenie o tym, że każdy piksel należący do zewnętrznego obrysu dowolnej komórki będzie miał wartość 0, nie zawsze jest spełnione. W realistycznym scenariuszu zdarza się, że jedna z końcówek komórki znajdują się na tyle blisko innej komórki, że wstępne przetwarzanie i progowanie obrazu nie powoduje ich rozdzielenia na obrazie binarnym. Czasem artefakty widoczne na obrazie wejściowym powodują, że wyspa na obrazie binarnym zawiera nie tylko komórkę, ale także fragment innego kształtu. W zdecydowanej większości takich przypadków szkielet komórki można opisać spójnym podgrafem o stopniu 2 grafu reprezentującego szkielet wyspy zawierającej komórkę.

Wstępna detekcja szkieletu danej komórki polega, tak jak w scenariuszu optymistycznym, na odnalezieniu krawędzi  $\{v_0, u_0\}$  w grafie  $\mathbf{G}$  leżącej najbliżej punktu opisującego komórkę, a następnie na stworzeniu z niej i jej wierzchołków nowego grafu  $\mathbf{S}_0$ . Tak utworzony szkielet rozszerzany jest później zgodnie z następującym algorytmem:

```

for each  $v \in \{v_0, u_0\}$  do
  loop
     $E_v \leftarrow$  zbiór krawędzi  $e \in E(\mathbf{G}) - E(\mathbf{S}_n)$  incydentnych do  $v$ ,
    takich że graf  $\mathbf{S}_n + e$  nie posiada cykli
    if  $E_v = \emptyset$  then
      break
    end if
     $\{v, u\} \leftarrow \text{argmax}_{e \in E_v} q_v(e)$ 
     $\mathbf{S}_{n+1} \leftarrow \mathbf{S}_n + \{v, u\}$ 
     $v \leftarrow u$ 
  end loop
end for
```

W powyższym algorytmie oznaczenie  $E(\mathbf{G})$  opisuje zbiór krawędzi grafu  $\mathbf{G}$ . Wyrażenie  $\mathbf{S} + e$  oznacza graf utworzony poprzez dodanie do grafu  $\mathbf{S}$  krawędzi  $e$  oraz jej wierzchołków. Funkcja  $q_v(e)$  jest tutaj funkcją oceny, która służy do wyboru najmocniej związanego krawędzi. Siłą wiążania nazywam najniższą wartość indeksu kształtu (oryginalnego obrazu) dla piksela leżącego na krawędzi  $e$ , w okolicy wierzchołka  $v$  (w

odległości nie większej niż pewna stała  $d$  od środka masy pikseli tworzących węzeł  $v$ ).

Takie rozwiązań sprawdza się dobrze dla komórek, których szkielet znajduje się w grafie  $\mathbf{G}$ , oraz jego zakończenia w tym grafie mają stopień 1. Pomijam na ten moment przypadek, gdy szukany szkielet komórki nie istnieje w grafie  $\mathbf{G}$ . W pozostałych przypadkach przynajmniej jedno z zakończeń szukanego podgrafa  $\mathbf{S}$  ma w grafie  $\mathbf{G}$  stopień większy niż 1. Zatem, o ile funkcja oceny  $q_v(e)$  sprawdzi się dobrze jeśli chodzi o dobór krawędzi należących do szukanego szkieletu komórki, uzyskany na końcu algorytmu graf  $\mathbf{S}_n$  będzie nadgrafenem szukanego szkieletu  $\mathbf{S}$ . W takim przypadku jego „nadmiarowa” część należy do szkieletu innej komórki lub jest wynikiem artefaktu widocznego na oryginalnym obrazie. Drugi przypadek pozostawiam do ręcznego rozwiązań użytkownikowi. Pierwszy natomiast, nachodzące na siebie szkielety komórek, rozwiążę automatycznie.

### 2.3.6. Rozwiązywanie konfliktów

Rozdzielenie nachodzących na siebie komórek nazwałem „rozwiązywaniem konfliktów”. Niech  $\Sigma_0$  będzie zbiorem wszystkich znalezionych szkieletów komórek, natomiast  $X(\Sigma)$  zbiorem par szkieletów nachodzących na siebie, należących do zbioru  $\Sigma$ . Rozwiązywanie konfliktów przebiega w następujący sposób:

```
while  $X(\Sigma_n) \neq \emptyset$  do
     $\{S_x, S_y\} \leftarrow$  dowolna para ze zbioru  $X(\Sigma)$ 
     $\{S'_x, S'_y\} \leftarrow$  wynik rozwiązywania konfliktu pomiędzy  $S_x$  i  $S_y$ 
     $\Sigma_{n+1} \leftarrow (\Sigma_n - \{S_x, S_y\}) \cup \{S'_x, S'_y\}$ 
end while
```

Sposób w jaki rozwiązywany jest konflikt pomiędzy dwoma szkieletami, zależy od tego w jaki sposób nachodzą one na siebie. Kluczową rolę odgrywają tutaj także punkty lokalizujące komórki. W dalszej części pisząc, o punkcie charakterystycznym komórki, będę miał na myśli punkt leżący na szkielecie będący najbliższym punktem lokalizującego komórkę.

Zaimplementowałem dwie metody rozwiązywania konfliktów, które stosuję w zależności od rozmieszczenia punktów charakterystycznych. W obu przypadkach rozwiązywanie konfliktu pomiędzy komórkami polega na skróceniu ich w taki sposób, by nie nachodziły one na siebie.

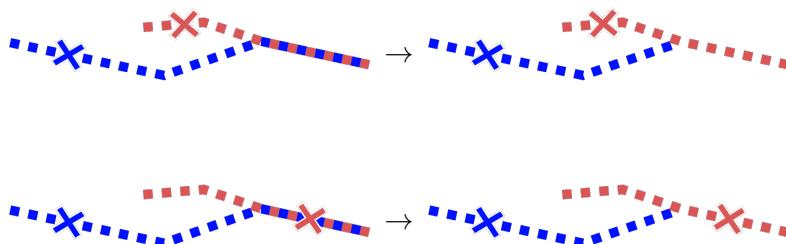
### Przyłączenie części wspólnej do jednej z komórek

Po rozwiązaniu konfliktu za pomocą tej metody, jedna z komórek pozostanie niezmieniona. Druga zostanie skrócona w taki sposób, że jej nową końcówką stanie

się miejsce, w którym obie komórki się spotykały. Rozwiązywanie takie stosuje w dwóch przypadkach:

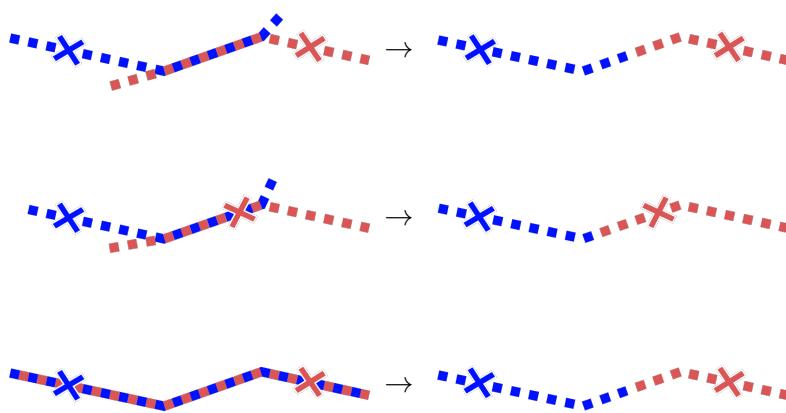
1. Gdy oba punkty charakterystyczne znajdują się poza częścią wspólną, ale po tej samej stronie części wspólnej.
2. Gdy punkt charakterystyczny pierwszej z komórek znajduje się wewnętrzczczęści wspólnej, a drugiej poza nią, oraz pierwsza komórka nie ma żadnej krawędzi poza częścią wspólną, w kierunku przeciwnym do punktu charakterystycznego drugiej komórki.

W pierwszym przypadku część wspólna przydzielana jest do tego szkieletu z którym jest mocniej związana (siłę wiązania określam w podobny sposób jak w opisanym wcześniej algorytmie konstrukcji szkieletu). W drugim przypadku sprawa jest prosta – część wspólna zostaje przydzielona do szkieletu którego punkt charakterystyczny leży w części wspólnej.



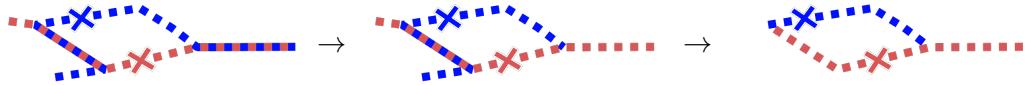
### Podzielenie części wspólnej

We wszystkich pozostałych przypadkach stosuję drugą metodę, polegającą na skróceniu obu komórek. Rozwiązywanie konfliktu polega tutaj na znalezieniu najsłabszego punktu (mającego najniższy indeks kształtu) leżącego w części wspólnej pomiędzy punktami charakterystycznymi, a następnie odpowiednim skróceniu obu szkieletów do tego punktu.



### Nakładanie się wielokrotne

Dwa szkielety mogą mieć więcej niż jeden konflikt równocześnie. Jest to rzadkie, ale realistyczne. W tym przypadku w każdej iteracji rozwiązywany jest jeden z takich konfliktów za pomocą sposobów opisanych powyżej.



### 2.3.7. Uzyskiwanie linii łamanej

Po rozwiązaniu wszystkich konfliktów każdy szkielet opisuje pewną komórkę. Kolejnym krokiem jest konwersja reprezentacji grafowej (która zawiera zbiór pikseli opisujących szkielet) do linii łamanej. W takiej formie użytkownik będzie mógł ręcznie modyfikować zaznaczenia komórek korzystając ze standardowego interfejsu programu ImageJ. Pierwsza wersja linii łamanej powstaje ze wszystkich punktów tworzących krawędzie i końcówki szkieletu, a także ze środków masy jego węzłów. Następnie geometria łamanej jest upraszczana za pomocą algorytmu Ramera–Douglasa–Peuckera[17][18]. Algorytm ten eliminuje punkty, których usunięcie nie wpływa znacząco na kształt łamanej tj. po ich usunięciu odległość uproszczonej łamanej od oryginalnej jest nie większa niż pewna przyjęta stała.

### 2.3.8. Korekta końcówek

Ostatnim krokiem oznaczania komórek jest korekta końcówek. Potrzeba korekty wynika z algorytmu szkieletyzacji. Proces ten polega na erodowaniu binarnego obrazu — z tego powodu końcówki szkieletu komórki często oddalone są od faktycznej krawędzi na oryginalnym obrazie. Dotyczy to tylko zakończeń które reprezentowane są przez wierzchołki mające w oryginalnym szkielecie obrazu stopień równy 1. Nie dzieje się tak w przypadku, gdy binarny obraz komórki łączy się w tym miejscu z inną komórką, a zakończenie powstało na skutek rozwiązania konfliktu.

Właściwej lokalizacji zakończenia komórki szukam na półprostej tworzonej przez ostatni odcinek łamanej, która ją opisuje. Mając do dyspozycji obraz binarny na podstawie którego powstał szkielet, szukam zakończenia wyspy leżącego najbliżej początku półprostej. Jeśli punkt ten leży nie dalej niż pewna przyjęta stała, staje się on nowym zakończeniem łamanej. Stała którą przyjąłem była nie większa niż przewidywana szerokość komórki.

Proces ten powtarzany jest dla każdego zakończenia komórki wymagającego poprawienia.

## 2.4. Śledzenie komórek w czasie

Do tej pory opisywałem sposób na wykrywanie i oznaczanie komórek na pojedynczym obrazie, mając do dyspozycji punkty lokalizujące komórki wprowadzone przez użytkownika. Oryginalny problem dotyczył jednak stosu obrazów przedstawiającego cykl komórkowy bakterii w czasie. Uogólnienie opisanego sposobu polega na automatycznym wyznaczaniu dla każdej komórki kilku potencjalnych punktów ją lokalizujących, na podstawie łamanej opisującej tę komórkę w poprzedniej klatce nagrania naniesionej na aktualną klatkę. Następnie wybierane są te punkty (i stworzone przez nie szkielety), które dały najlepsze efekty względem pewnej funkcji oceny.

Jeśli nowy szkielet komórki jest znacznie krótszy niż łamana opisująca ją na poprzednim obrazie, prawdopodobnie oznacza to, że nastąpił podział komórki. W takim przypadku wyszukiwany jest kolejny szkielet. Tym razem kandydatów na punkty lokalizujące poszukuję tylko po jednej stronie łamanej. Jeśli pierwszy szkielet powstał na podstawie punktu leżącego bliżej końca łamanej, jego bliźniaczy szkielet prawdopodobnie znajduje się bliżej jej początku i vice versa.

Po odnalezieniu wszystkich nowych szkieletów zgodnie z powyższą procedurą, następuje etap rozwiązywania konfliktów, konwersji na linie łamane i ostatecznej korekty końcówek, tak jak to zostało opisane (2.3.6.–2.3.8.).

Niech  $\mathbf{C}_n$  będzie zbiorem łamanych opisujących komórki na  $n$ -tym obrazie stosu. Niech  $\phi_\sigma(d)$  będzie funkcją która liczbą z przedziału  $[0, 1]$  przyporządkowuje punkty leżące na łamanej  $\sigma$ , proporcjonalnie do odległości od jej początku (przyjmijmy, że linia łamana ma zdefiniowany „kierunek”, tzn. ma początek i koniec). Niech  $S_n(p)$  będzie szkieletem komórki wyznaczonym zgodnie z metodą opisaną wcześniej (2.3.2.–2.3.5.) dla punktu lokalizującego  $p$  i  $n$ -tej klatki nagrania. Cały proces można zobrazować następującym algorytmem:

```

for each  $n \in \{0, 1, \dots, n - 1\}$  do
     $\Sigma_{n+1} \leftarrow \emptyset$ 
    for each  $\sigma \in C_n$  do
         $\Sigma \leftarrow \{S_{n+1}(\phi_\sigma(d)) \mid d \in D\}$ 
         $S \leftarrow \text{argmax}_{S \in \Sigma} q_\sigma(S)$ 
         $\Sigma_{n+1} \leftarrow \Sigma_{n+1} \cup \{S\}$ 
        if  $|S| < 0.9 \cdot |\sigma|$  then
             $\Sigma' \leftarrow \{S_{n+1}(\phi_\sigma(d')) \mid d' \in D'\}$ 
             $S' \leftarrow \text{argmax}_{S' \in \Sigma'} q_\sigma(S')$ 
             $\Sigma_{n+1} \leftarrow \Sigma_{n+1} \cup \{S'\}$ 
        end if
    end for
     $\hat{\Sigma}_{n+1} \leftarrow$  wynik rozwiązań konfliktów w zbiorze  $\Sigma_{n+1}$ 
     $C_{n+1} \leftarrow$  wynik konwersji na łamane i korekty końcówek elementów zbioru  $\hat{\Sigma}_{n+1}$ 
end for

```

W algorytmie pojawia się zbiór  $D \subseteq [0, 1]$  którego elementy dobrałem empirycznie jako  $\{0.2, 0.4, 0.6, 0.8\}$ . Zbiór  $D' \subset D$  to zbiór, który zależy od wcześniejszego wyboru najlepszego kandydata na szkielet komórki. Jeśli w danej iteracji szkielet  $S$  powstał na podstawie punktu lokalizującego leżącego na pierwszej połowie łamanej, będzie to zbiór  $\{d \in D \mid d > 0.5\}$  w przeciwnym wypadku będzie to pozostała część zbioru  $D$ .

Funkcja  $q_\sigma(S)$  jest funkcją oceny szkieletu względem krzywej  $\sigma$ . Bazuje ona na założeniu, że komórka w nowej klatce nie powinna zbytnio oddalić się od miejsca w którym znajdowała się poprzednio. Dla wszystkich potencjalnych punktów lokalizujących komórkę, wyliczana jest ich odległość od nowo utworzonego szkieletu  $S$ . Wartość oceny jest odwrotnie proporcjonalna do sumy tych odległości.

## 2.5. Interakcja ze strony użytkownika

Niestety powyższe działania nie sprawdzają się dobrze w każdym przypadku. Jest wiele czynników które utrudniają poprawną detekcję komórek. Większość z nich związana jest ze słabą jakością nagrani. Obrazy są mocno zaszumione, często występują na nich artefakty, czasem płytka na której poruszają się komórki minimalnie się przesuwa, a innym razem mikroskop traci ostrość na kilka klatek nagrania. Zapewne można próbować rozwiązać te problemy automatycznie, ale na pewno pojawią się nowe, nie rozważane wcześniej przypadki. Właśnie z tego powodu postanowiłem zapewnić użytkownikowi możliwość interaktywnego poprawiania automatycznie stworzonych oznaczeń komórek, ale także całkowicie manualne kontynuowanie detekcji, zachowując przy tym informacje o historii komórek.

Komórki przedstawione są w oknie programu jako standardowe zaznaczenia programu ImageJ i można je edytować za pomocą standardowego interfejsu. Ponadto stworzyłem dwa dodatkowe narzędzia dla użytkownika. Pierwsze z nich służy do rozcinania komórek w miejscach, w których się one dzielą. Mimo że algorytm śledzenia ma mechanizm odpowiedzialny zaauważenie takich zmian, zdarza się że dzieje się to zbyt późno (komórki jeszcze przez pewien czas po podziale znajdują się bardzo blisko siebie).

Drugie narzędzie służy do skracania końcówek zaznaczeń. Zauważałem, że częstym problemem jest to, że zaznaczenie wybiega poza krawędzie komórki. Najczęściej dzieje się to ze względu na występujące na obrazie artefakty lub punkty stałe na płytkach znajdujące się blisko zakończeń komórek. Żeby przyspieszyć proces poprawiania takich zaznaczeń można skorzystać z narzędzia do skracania końcówek, które działa podobnie do „gumki” w popularnych programach graficznych.

Chcąc ręcznie oznaczyć komórki w kolejnej klatce (zamiast wyliczać je automatycznie na podstawie poprzedniej), użytkownik może skorzystać z opcji duplikowania klatki. Dzięki temu w aktualnej klatce pojawią się wszystkie zaznaczenia z klatki poprzedniej, które następnie można ręcznie dopasować do ich nowej pozycji.

## Rozdział 3.

# Opis implementacji

### 3.1. Kod źródłowy

Kod źródłowy projektu zamieszczony został w publicznie dostępnym repozytorium w serwisie GitHub na licencji MIT. Repozytorium można znaleźć pod adresem <https://github.com/rossinek/cell-detector-imagej-plugin>.

Program zaimplementowany jest jako wtyczka do programu ImageJ, napisana w języku Java. Do automatyzacji procesu budowy wykorzystany został Apache Maven. Wszystkie zależności zostały uwzględnione w pliku konfiguracyjnym dla Mavena, który odpowiada również za ich pobranie.

Plugin był rozwijany głównie na systemie macOS Catalina, ale został również przetestowany na systemie Windows 10.

### 3.2. Kompilacja i uruchomienie

Przed zbudowaniem pluginu z kodu źródłowego należy upewnić się, że zainstalowane są wymagane zależności. Poniższa instrukcja zakłada, że system wyposażony jest w:

- Apache Maven w wersji większej lub równej 3.3.9
- Java w wersji 8.

Aby zbudować projekt należy skorzystać z komendy `mvn`, która spowoduje pobranie wszystkich zależności, komplikację kodu źródłowego i wywołanie testów automatycznych.

Zbudowany plugin dla programu ImageJ można znaleźć w folderze `target` pod nazwą `Mtbt_Plugin-{wersja}.jar`. Tak spakowany plugin można zainstalować w



Rysunek 3.1: Menu programu ImageJ rozszerzone o wtyczki zaimplementowane na potrzeby tej pracy.

programie ImageJ (lub Fiji) wklejając go do folderu `jars` w miejscu gdzie zainstalowany jest program.

Aby uruchomić plugin w ramach własnej niezależnej instancji ImageJ, po zbudowaniu projektu można uruchomić główną metodę programu komendą `mvn exec:java -Dexec.mainClass="dev.mtbt.Main"`.

### 3.3. Obsługa pluginu

Po zainstalowaniu pluginu w programie ImageJ pojawi się nowe menu `Mycobacterium` (rysunek 3.1) w którym znajdują się narzędzia zaimplementowane na potrzeby tej pracy. Poza głównym pluginem (`Cell detector`) znajdują się tam jeszcze dodatkowe wtyczki służące do importu oraz eksportu komórek, a także narzędzia pomocne w rozwoju programu. Główny plugin podzielony jest na trzy kroki:

1. Lokalizacja i wstępna detekcja komórek
2. Śledzenie komórek w czasie
3. Analiza danych

Przed uruchomieniem pluginu należy otworzyć nagranie które chcemy analizować. Po jego uruchomieniu zobaczymy okno pluginu przedstawiające pierwszy krok. Interfejs pluginu na każdym etapie składa się z kilku stałych i kilku zmiennych (w zależności od aktualnego kroku) części składowych. Po prawej stronie znajduje się zawartość standardowego okna wielokanałowego stosu obrazów ImageJ. Dwa suwaki w dolnej części służą do wyboru kanału oraz klatki nagrania. Wybór kanału ma wpływ na to, na którym z nich odbywa się detekcja. Pierwszym wyborem powinien być kanał przedstawiający komórki oświetlone światłem widzialnym. Obok suwaka do wyboru klatki znajduje się przycisk start/stop służący do odtwarzania nagrania. W górnej części panelu znajdującego się po lewej stronie znajdują się kontrolki specyficzne dla danego kroku pluginu. W jego dolnej części znajdują się narzędzia przydatne na każdym etapie działania pluginu, a także przyciski do zmiany kroku.



Rysunek 3.2: Pierwszy krok pluginu służący do wstępnej detekcji komórek.

### 3.3.1. Lokalizacja i wstępna detekcja komórek

Pierwszy etap polega na zaznaczeniu przez użytkownika punktów lokalizujących komórki, a następnie uruchomieniu wstępnej detekcji (rysunek 3.2). W tym celu należy wcisnąć przycisk **Select cells**, zaznaczyć każdą komórkę widoczną na pierwszej klatce nagrania klikając w nią kursem, a następnie wcisnąć przycisk **Run detection**. W celu poprawienia wstępnej selekcji można skorzystać z przycisku **Reset**, który usuwa cały uzyskany do tej pory efekt.

W dodatkowym panelu **advanced settings** ukryte zostały dodatkowe parametry sterujące procesem detekcji komórek (2.3.), takie jak siła rozmycia gaussowskiego czy wartość progowa użyte na etapie wstępnego przetwarzania obrazu. Można tam również włączyć podgląd mapy indeksów kształtu, a także szkieletu na podstawie których przebiega proces detekcji.

### 3.3.2. Śledzenie komórek w czasie

Kolejnym krokiem jest śledzenie komórek w czasie (rysunek 3.3). Użytkownik na tym etapie może zdecydować w jaki sposób chce dalej pracować. Może skorzystać z opcji automatycznego wyliczenia jednej bądź kilku kolejnych klatek (przycisk **calculate next**) lub zduplikować aktualnie zaznaczone komórki do kolejnej klatki,



Rysunek 3.3: Drugi krok pluginu służący do wstępnej śledzenia komórek w czasie.

a następnie skorzystać z dostępnych narzędzi i manualnie je dostosować (`duplicate`). Cofając się w historii do poprzednich klatek można poprawiać zaznaczenia. Można też powtórnie wyliczyć zaznaczenia na danym obrazie, jednak należy wtedy pamiętać, że zaznaczenia w kolejnych klatkach zostaną usunięte.

### 3.3.3. Analiza danych

Ostatni krok umożliwia analizę danych wyliczonych na podstawie zaznaczonych komórek (rysunek 3.4). W celu zapewnienia poprawności danych należy najpierw wskazać numer kanału z fluorescencją. Kolejnym krokiem jest wybór komórki do analizy. Umieszczona niżej sekcja przycisków pozwala na wyświetlenie kilku rodzajów wykresów (rysunek 3.5):

- wykresu przedstawiającego zmianę długości w czasie
- stosu profili intensywności pod lamaną opisującą komórkę na kanale z fluorescencją (dla każdej klatki życia komórki)
- średniego profilu intensywności pod lamaną opisującą komórkę na kanale z fluorescencją (przed uśrednieniem oś X każdego profilu jest normalizowana do przedziału  $[0, 1]$ )
- umiejscowienia „ognisk” na kanale z fluorescencją w czasie.



Rysunek 3.4: Trzeci krok pluginu służący do analizy zaznaczonych komórek.

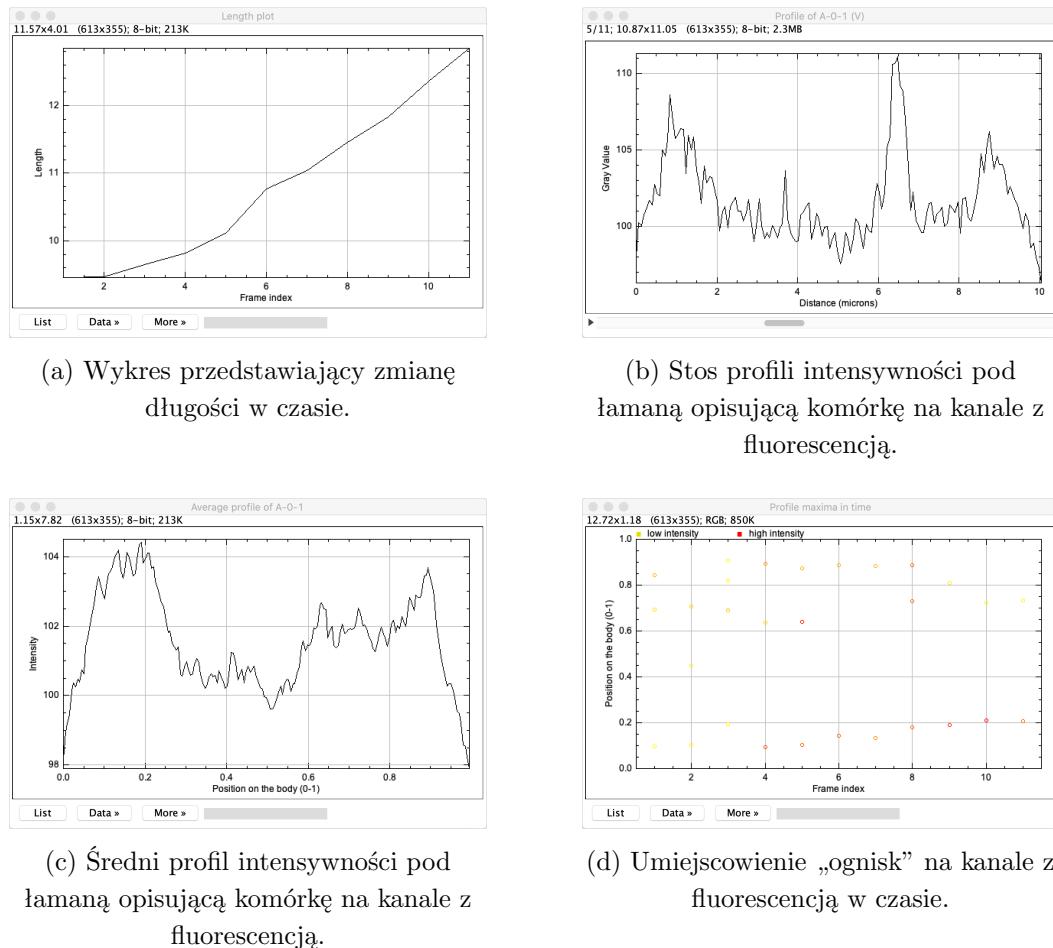
W celu analizy profili i długości komórek poza programem można, za pomocą kolejnych przycisków, wyeksportować dane pojedynczej lub wszystkich komórek do formatu CSV (ang. comma-separated values).

#### 3.3.4. Manualna modyfikacja komórek

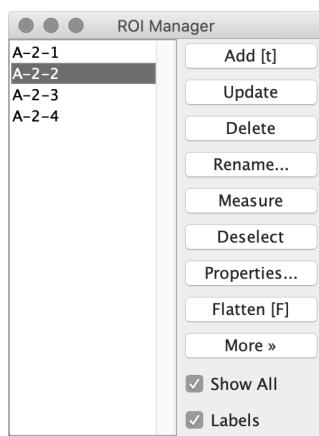
W celu manualnej modyfikacji komórki należy wybrać ją w oknie **ROI Manager** (rysunek 3.6), które otwierane jest automatycznie podczas pracy z komórkami. Komórka na obrazie będzie wyświetlona w postaci standardowego zaznaczenia wykonanego narzędziem do zaznaczania linii łamanej (ang. Segmented Line Selection Tool)[19]. Przesuwanie, usuwanie lub dodawanie wierzchołków takiego zaznaczenia zapisywane jest także dla komórki. Komórki usunięte z listy okna **ROI Manager** zostaną usunięte również z kolekcji wraz z zaznaczeniami w kolejnych klatkach, jak i całym potomstwem. Kolekcja zaznaczeń dla poprzednich klatek pozostanie bez zmian.

Ponadto okno pluginu wyposażone jest w dwa dodatkowe narzędzia mające ułatwić manualną edycję komórek. Narzędzia można włączyć za pomocą przycisków widocznych w lewym dolnym rogu okna pluginu.

Tryb rozcinania komórek można aktywować za pomocą przycisku z ikoną w



Rysunek 3.5: Różne rodzaje wykresów.



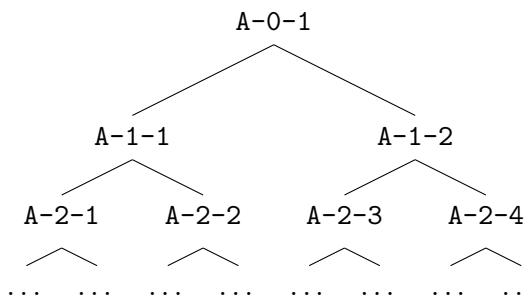
Rysunek 3.6: Okno ROI Manager pozwalające na wybór komórki.

ksztalcie nożyczek (☒). Służy on do ręcznego oznaczania punktów podziału komórek. Po jego uruchomieniu użytkownik może „przeciąć” komórkę rysując symboliczną linię na obrazie. Przecięta komórka zostanie podzielona na dwie komórki, które zostaną dodane do kolekcji jako „dzieci” komórki sprzed podziału (jeśli występuje ona w poprzedniej klatce). Ze względu na specyfikę cyklu komórkowego, nie jest możliwe doprowadzenie do stanu w którym jedna komórka dzieli się na więcej niż dwie części. Należy pamiętać, że rozcięcie komórki, która została oznaczona w późniejszych klatkach, wiąże się z utratą jej zaznaczeń w tych klatkach oraz usunięciem jej potomstwa.

Drugim dodatkowym narzędziem jest tryb skracania końcówek, uruchamiany za pomocą przycisku z ikoną w kształcie gumki do mazania (◐). Działa on podobnie do narzędzia „gumki” obecnego w popularnych programach graficznych. Za jego pomocą nie można podzielić komórkę, ale można ją skrócić lub całkowicie usunąć. Jeśli cała komórka zostanie „wymazana” w całości to, analogicznie jak przy usuwaniu komórki z poziomu okna ROI Manager, usunięte zostaną również zaznaczenia w kolejnych klatkach oraz jej potomstwo.

### 3.3.5. Nazewnictwo komórek oraz ich orientacja

Aby ułatwić identyfikację komórek, przyjąłem stałą konwencję ich nazewnictwa. Nazwa składa się z trzech członów: identyfikatora komórki źródłowej, indeksu pokolenia oraz indeksu w ramach pokolenia. Komórkom źródłowym (nie mającym rodzica) zostaje przydzielony unikalny identyfikator w postaci wielkiej litery alfabetu. Indeks pokolenia jak i indeks w ramach pokolenia wynikają z natury kolekcji komórek. Każda komórka dzieli się w pewnym momencie na dwie nowe komórki (dzieci) nowego pokolenia. „Drzewo genealogiczne” komórek będzie w takim przypadku drzewem binarnym, a indeksy pokolenia i w ramach pokolenia będą odpowiednio odległością od korzenia i liczbą porządkową w ramach tego samego indeksu pokolenia. Przykładowe drzewo pokrewieństwa opisane nazwami komórek przedstawia rysunek 3.7.



Rysunek 3.7: Drzewo genealogiczne opisane zgodnie z konwencją nazewnictwa.

Dzięki tak skonstruowanej konwencji nazewnictwa można łatwo zidentyfikować pochodzenie komórki, a także stopień pokrewieństwa z innymi komórkami.



Rysunek 3.8: Komórki wraz ze znacznikami identyfikującymi końcówki.

W celu zapewnienia spójności analizowanych na końcu danych, ustaliłem również stałą orientację komórek. Aby móc rozróżnić końcówki komórek, wystarczy zaznaczyć opcję `show endpoints` w dolnej części okna pluginu. Kolorem czerwonym oznaczone zostaną końcówki powstałe w wyniku podziału komórki, a zielonym te które były wcześniej końówkami komórki rodzica (rysunek 3.8). W przypadku komórki źródłowej końówkami oznaczone są losowo.

### 3.3.6. Importowanie i eksportowanie komórek

Manualne oznaczanie i poprawianie komórek może być czasochłonne i nie zawsze można zakończyć pracę w ciągu jednej sesji. Aby nie utracić efektów swojej pracy, użytkownik może wyeksportować oznaczone komórki za pomocą menu `Mycobacterium > Export cells`. Stworzony w ten sposób plik XML jest wynikiem serializacji kolekcji komórek. Podczas kolejnej sesji, po otwarciu tego samego nagrania, zapisane komórki można wczytać korzystając z menu `Mycobacterium > Import cells`.

## 3.4. Dokumentacja techniczna

### 3.4.1. Struktura projektu

Główna struktura plików projektu podzielona jest zgodnie z konwencją zaproponowaną przez zespół Apache Maven[20]. Główne foldery to:

- `src/main/java` – kod źródłowy aplikacji
- `src/main/resources` – zasoby aplikacji np. obrazki
- `src/test/java` – kod źródłowy testów automatycznych.

Kod źródłowy aplikacji podzielony jest na kilka folderów, ze względu na aspekt którego dotyczy:

- **cells** – kod zawierający logikę dotyczącą domeny pracy, m.in. kod pluginu, interfejsy poszczególnych kroków, główne algorytmy itp.; wewnątrz wydzielono dodatkowo foldery:
  - **serialization** – kod dotyczący importu i eksportu komórek
  - **skeleton** – implementacja króków detekcji i śledzenia komórek opisana w tej pracy (oparta na szkieletyzacji)
  - **measurements** – implementacja etapu analizy danych
- **graph** – implementacja grafu
- **gui** – implementacje generycznych okien i komponentów interfejsu
- **imagej** – klasy ułatwiające prace i komunikację z programem–hostem ImageJ
- **util** – inne pomocnicze klasy
- **vendor** – publicznie dostępne wtyczki do programu ImageJ lub opakowania ułatwiające korzystanie z zależności zadeklarowanych w **pom.xml**.

### Konwencja nazewnictwa

Nazwy interfejsów rozpoczynają się od wielkiej litery „I” np. **ICellsPluginStep**, natomiast nazwy klas abstrakcyjnych zaczynają się od prefiku „Abstract” np. **AbstractCellCollection**.

#### 3.4.2. Główne struktury danych

##### **AbstractCellCollection**

Ta abstrakcyjna klasa służy do opisywania drzewiastej struktury komórek. Dziedziczą po niej dwie klasy: **CellCollection** oraz **Cell**. Ta pierwsza służy do przechowywania kolekcji komórek reprezentowanych przez klasę **Cell**. Druga natomiast oprócz reprezentowania komórki, może zawierać także referencje do dwóch komórek potomnych, tworząc w ten sposób binarne „drzewo genealogiczne” komórki.

Klasa deklaruje m.in. metody dodawania i usuwania komórek z kolekcji, ale także metody do wydobywania wszystkich komórek „żyjących” w danej klatce nagrania.

Klasy dziedziczące po **AbstractCellCollection** muszą być serializowalne. Własność ta jest wykorzystywana podczas importowania i eksportowania komórek do pliku XML.

### **Cell**

Jest to klasa opisująca komórkę i cały jej cykl życia – od pojawienia się, do momentu podziału na dwie nowe komórki (może zawierać także referencje do tych komórek potomnych). Obiekt tej klasy zawiera informacje o każdej klatce w której występuje komórka w postaci referencji do instancji **AbstractCellFrame**. Pozwala też określić i odczytać poszczególne części nazwy komórki takie jak identyfikator komórki źródłowej, indeks pokolenia czy indeks w ramach pokolenia (3.3.5.).

Poza wyżej wymienionymi oraz dziedzicznymi po klasie **AbstractCellCollection** funkcjonalnościami, obiekt tej klasy zawiera również metody potrzebne do uzyskania reprezentacji komórki w postaci standardowego zaznaczenia w programie ImageJ.

### **AbstractCellFrame**

Klasa ta deklaruje metody służące do uzyskania zaznaczenia pojedynczej komórki w konkretnej klatce nagrania, a także do modyfikacji tego zaznaczenia.

Na potrzeby tej pracy powstała jej implementacja **PolylineCellFrame**, która reprezentuje zaznaczenie komórki w postaci linii łamanej.

### **Graph, Edge, Vertex, Point**

Klasy te składają się na implementację grafu, który zawiera dodatkowe informacje o punktach reprezentowanych przez krawędzie i wierzchołki.

#### **3.4.3. Architektura umożliwiająca dalszy rozwój**

Główną klasą projektu jest **CellsPlugin**, która instancjonowana jest przez program ImageJ po otwarciu wtyczki z poziomu menu **Mycobacterium > Cell Detector**. Podczas uruchomienia tworzona jest kopia aktualnie otwartego stosu obrazów, która wyświetlona zostaje w specjalnym oknie pluginu (jego opis znajduje się w rozdziale 3.3.), a także utworzona zostaje pusta kolekcja komórek.

Podczas inicjalizacji uruchomiony zostaje również nasłuch na różnego rodzaju zdarzenia programu–hosta ImageJ m.in. na:

- zmiany podglądu obrazu – umożliwia to wyświetlenie odpowiednich komórek dla aktualnie oglądanej klatki
- zmiany zaznaczeń na obrazie – dzięki temu zmiany mogą być zapisywane dla komórek opisywanych przez zmodyfikowane zaznaczenia
- zmianę aktywnego narzędzia – może mieć wpływ na zachowanie wybranego narzędzia specjalnego (3.3.4.)

- zmiany na liście okna **Roi Manager** – umożliwia to usuwanie komórek z poziomu tego okna.

Każdy z trzech etapów pluginu (wstępna detekcja, śledzenie komórek i analiza danych) jest obiektem klasy implementującej interfejs **ICellsPluginStep**. Interfejs ten deklaruje zaledwie trzy publiczne metody:

- JComponent init(ImagePlus imp, CellCollection cells)**

Metoda ta wywoływana jest podczas inicjalizacji danego kroku np. przy starcie pluginu (dla kroku wstępnej detekcji) lub przy wcisnięciu przycisku **next** (dla kolejnego kroku). Argumentami są obraz źródłowy (nagranie) oraz referencje do kolekcji komórek. Metoda odpowiedzialna jest również za stworzenie i zwrócenie komponentu interfejsu danego kroku, który zostanie automatycznie umieszczony w odpowiednim miejscu okna wtyczki.

- void imageUpdated()**

Ta metoda służy do nasłuchiwanego zmiany obrazu i wyświetlanych komórek. Uruchamiana jest przez plugin dla aktywnego kroku po każdej aktualizacji podglądu obrazu i komórek na nim wyświetlonych.

- void cleanup()**

Metoda ta wywoywana jest podczas zmiany aktualnego kroku lub przy zamknięciu pluginu.

Instancja klasy implementującej interfejs **ICellsPluginStep** tworzona jest za pomocą metody **Class.newInstance**. Do jej stworzenia użyty zostanie konstruktor niewymagający podania argumentów.

Dzięki tak zaprojektowanemu systemowi każdy z etapów programu jest niezależny i może zostać z łatwością wymieniony na inną implementację tego interfejsu. Klasy które mają zostać użyte są zadeklarowane jako statyczne stałe **CellsPlugin.StepDetectorClass**, **CellsPlugin.StepLifeTrackerClass** oraz **CellsPlugin.StepMeasurementsClass**.

W aktualnej wersji pluginu implementacjami poszczególnych kroków są klasy **SkeletonCellDetector**, **SkeletonCellLifeTracker** oraz **StepMeasurements**. Przykładowo chcąc użyć klasy **ExampleLifeTracker** jako innej implementacji kroku śledzenia komórek, należy upewnić się że implementuje ona interfejs opisany powyżej, a następnie zmienić wartość stałej **CellsPlugin.StepLifeTrackerClass** na **ExampleLifeTracker.class**.

Kod pluginu zawiera wszystkie główne struktury danych (lub ich abstrakcyjne wersje) i dzięki temu jest zupełnie niezależny od implementacji poszczególnych kroków – komunikuje się z nimi jedynie za pomocą opisanych wyżej interfejsów. Funkcjonalności niezależne od implementacji poszczególnych etapów takie jak wyświetlanie,

usuwanie i ręczna modyfikacja komórek, a także ich eksportowanie i importowanie, bazują wyłącznie na metodach deklarowanych przez odpowiednie interfejsy lub klasy abstrakcyjne.

### Przykładowa implementacja pojedynczego etapu pluginu

Poniższy fragment kodu przedstawia przykładową implementację kroku śledzenia komórek. Jest to uproszczona implementacja wyświetlająca jedynie przycisk `duplicate`, który umożliwia skopiowanie aktualnych zaznaczeń do kolejnej klatki.

```
public class ExampleLifeTracker implements ICellsPluginStep {
    ImagePlus imp;
    CellCollection cellCollection;

    @Override
    public JComponent init(ImagePlus imp, CellCollection cellCollection) {
        this.imp = imp;
        this.cellCollection = cellCollection;

        JPanel component = new JPanel();
        component.setLayout(new BoxLayout(component, BoxLayout.Y_AXIS));
        component.add(new RunnableButton("duplicate", this::onDuplicateClick));
        return component;
    }

    @Override
    public void imageUpdated() {
        // ignore
    }

    @Override
    public void cleanup() {
        // ignore
    }

    void onDuplicateClick() {
        int currentFrameIndex = imp.getT();
        // get list of cells visible at current frame
        List<Cell> cells = cellCollection.getCells(currentFrameIndex);
        int nextFrameIndex = currentFrameIndex + 1;
        if (cells.size() < 1 || nextFrameIndex > imp.getNFrames()) {
            return;
        }
        // clear previous selections for next frames
        cells.forEach(cell -> cell.clearFuture(nextFrameIndex));
        cells.forEach(cell -> {
            // clone current selection
            AbstractCellFrame duplicate = cell.getFrame(currentFrameIndex).clone();
            // set selection for next frame to cloned value
            cell.setFrame(nextFrameIndex, duplicate);
        });
        // show next frame
        imp.setT(nextFrameIndex);
        // update image and cells preview
        imp.updateAndDraw();
    }
}
```

Podczas inicjalizacji zapisywane są referencje do obrazu wejściowego `imp` oraz do kolekcji komórek `cellCollection`, a następnie tworzony jest komponent zawierający jeden przycisk. Wciśnięcie przycisku `duplicate` powoduje wywołanie metody `onDuplicateClick`, która duplikuje zaznaczenia widoczne w aktualnej klatce do kolejnej klatki. Jeśli w aktualnej klatce nie ma żadnych zaznaczeń lub jest to ostatnia klatka nagrania, metoda nie powoduje żadnego efektu. Zwrócony przez metodę `init` komponent zostanie automatycznie umieszczony w oknie pluginu.

Załóżmy że chcemy jeszcze dodatkowo zmienić podgląd obrazu, tak aby zamiast oryginalnego obrazu użytkownik widział mapę indeksów kształtu. Można to zrobić nakładając dodatkową warstwę `Overlay` na podgląd obrazu.

```
public class ExampleLifeTracker implements ICellsPluginStep {

    // ...

    @Override
    public void imageUpdated() {
        ImagePlus frame = HyperstackHelper.extractFrame(imp);
        ImagePlus shapeIndexMap = ShapeIndexMap.getShapeIndexMap(frame, 4.0);
        ImageRoi roi = new ImageRoi(0, 0, shapeIndexMap.getProcessor());
        Overlay overlay = new Overlay();
        overlay.add(roi);
        imp.setOverlay(overlay);
    }

    @Override
    public void cleanup() {
        imp.setOverlay(null);
    }

    // ...
}
```

Metoda `imageUpdated` zostanie wywołana przy każdej aktualizacji oryginalnego obrazu np. gdy zmieni się wyświetlana klatka lub kanał. Po przejściu do kolejnego etapu pluginu zostanie wywołana metoda `cleanup`, która w powyższej wersji usunie dodatkową warstwę przywracając tym samym podgląd do oryginalnej wersji.

Analogicznego mechanizmu używają aktualne implementacje kroku detekcji oraz śledzenia komórek do wyświetlenia podglądu mapy indeksów kształtu lub szkieletu (`advanced settings`).

#### 3.4.4. Wykorzystywane API ImageJ

ImageJ jest częścią większego ekosystemu SciJava. Projekty tworzone w ramach tego ekosystemu są wytwarzane jako otwarte oprogramowanie z publicznie dostępnym kodem. ImageJ jest nie tylko programem, ale rozszerzalną platformą zbudowaną na architekturze opartej o mechanizm pluginów. Stanowi to jeden z największych atutów tej platformy[21].

Plugin został napisany przy wykorzystaniu nowszego API ImageJ2, jednak korzysta również z warstwy zapewniającej kompatybilność z ImageJ 1.x. Spowodowane jest to głównie brakiem dobrej dokumentacji istotnych funkcjonalności dla nowszej wersji API, dostępnej w trakcie implementacji projektu.

W moim programie zdecydowałem się użyć natywnych dla ImageJ narzędzi do tworzenia zaznaczeń (ROIs, ang. regions of interest), w celu wyświetlania i modyfikacji komórek. Dzięki temu nie musiałem tworzyć od nowa mechanizmów służących do wyświetlania wielu linii łamanych na obrazie, a także do ich modyfikacji. Również narzędzia do przecinania i skracania końcówek są tak naprawdę nakładką na standardowe narzędzia do zaznaczania linii i zaznaczania za pomocą pędzla.

### 3.4.5. Brakujące lub niespójne API ImageJ

W tak dużym otwartym oprogramowaniu tworzonym przez dużą społeczność nie sposób uniknąć niespójności czy obsłużyć wszystkie warunki brzegowe. Implementując niektóre funkcjonalności zdecydowałem się użyć standardowych narzędzi ImageJ, w celu nieco innym niż ich oryginalne przeznaczenie. Ze względu na to, potrzebowałem niskopoziomowego dostępu do różnych mechanizmów za tym stojących, aby wstrzyknąć tam własną warstwę pośredniczącą. Twórcy ImageJ, mając na uwadze potrzebę rozszerzalności, wyposażyci tą platformę w szereg narzędzi mających w tym pomóc. Mimo wszystko kilka elementów systemu okazało się działać nie do końca zgodnie z intuicją i wymagało nie do końca eleganckiego rozszerzenia.

#### Nasłuchiwanie na zdarzenia modyfikacji ROI

Chcąc rejestrować wydarzenia na temat modyfikacji wyświetlanych aktualnie zaznaczeń, należy zaimplementować interfejs `RoiListener`, a następnie tę implementację zarejestrować przekazując ją do statycznej metody `Roi.addRoiListener`. Słuchacz będzie otrzymywać zgłoszenia na temat różnych wydarzeń związanych z zaznaczeniami m.in. na temat ich utworzenia, przesunięcia, modyfikacji, rozszerzenia, ukończenia czy usunięcia. W moim programie nasłuchiwanie na te wydarzenia odgrywa kluczową rolę. Zdiagnozowałem kilka miejsc w których system notyfikacji działa, moim zdaniem, niezgodnie z intuicją:

- Tworzenie zaznaczenia w kształcie linii polega na oznaczeniu kursorem myszy punktu początkowego, wcisnięciu głównego klawisza myszy, przeciągnięciu kurSORA do miejsca docelowego, a następnie zwolnieniu klawisza myszy. Konstruując zaznaczenie w kształcie linii, słuchacz dostaje informacje na temat wydarzenia związanego z modyfikacją, natomiast po zwolnieniu przycisku myszy nie jest zgłaszone żadne wydarzenie. Intuicja podpowiada mi, że powinno zostać zgłoszone wydarzenie związane z ukończeniem konstrukcji zaznaczenia.

- Analogiczny problem występuje w przypadku zaznaczania za pomocą pędzla. Tutaj proces zaznaczania podobny jest do „malowania” narzędziem pędzla w popularnych programach graficznych. Również w tym przypadku po zwolnieniu przycisku myszy nie jest zgłaszane żadne wydarzenie.
- Zaznaczenie w postaci lini łamanej może być modyfikowane na różne sposoby. Można przeciągać całą linię łamanej lub modyfikować jej wierzchołki poprzez ich przeciąganie, usuwanie lub dodawanie. W przypadku przesunięć całego zaznaczenia jak i w przypadku zmiany pozycji poszczególnych wierzchołków zgłaszone są poprawne wydarzenia (odpowiednio o przesunięciu i modyfikacji). Jednak w przypadku usuwania i dodawania wierzchołków żadne wydarzenie nie jest zgłaszane.

W celu rozwiązania pierwszych dwóch problemów postanowiłem stworzyć własną warstwę aplikacji (**RoiObserver**) odpowiedzialną za obserwowanie zmian zaznaczeń. Nasłuchuje ona na zmiany zaznaczeń poprzez interfejs **RoiListener**, ale także nasłuchuje na zdarzenia związane z interakcją między kursorem myszy, a otwartymi oknami obrazów. Podczas konstrukcji problematycznych zaznaczeń zapisywane są ich referencje, aby następnie po otrzymaniu informacji o zwolnieniu przycisku myszy wyemitować brakujące wydarzenie.

Trzeci problem został rozwiązany poprzez utworzenie i korzystanie z nowej klasy rozszerzającej zaznaczenie w kształcie lini łamanej. Nadpisuje ona tylko jedną metodę **PolygonRoi.mouseDownInHandle**, która używana jest do tworzenia i usuwania dodatkowych wierzchołków – odbywa się to przez kliknięcie kursorem myszy w wierzchołek, mając przy tym wcisnięty klawisz **alt** lub **control**. Nowa wersja metody wywołuje oryginalną, zachowując przy tym pierwotny sposób działania, a następnie zgłasza wydarzenie związane z modyfikacją.

### Usuwanie ROI z poziomu okna ROI Manager

Najbardziej naturalnym sposobem na usunięcie komórki (np. w przypadku gdy została niepoprawnie podzielona), wydaje się zaznaczenie jej na liście okna **ROI Manager** i wcisnięcie przycisku **Delete** na klawiaturze lub z poziomu interfejsu. Bez specjalnej obsługi takiego zachowania, zaznaczenie komórki zniknęłoby, mimo że komórka ciągle pozostała w kolekcji i po odświeżeniu podglądu znów pojawiłaby się na ekranie. Takie zachowanie byłoby nieintuicyjne z perspektywy użytkownika. Postanowiłem więc stworzyć odpowiednią obsługę takiej sytuacji. Okazało się jednak, że nie istnieje żadne narzędzie pozwalające otrzymywać powiadomienia na temat usuniętego z poziomu tego okna zaznaczenia.

Problem ten rozwiązałem poprzez odnalezienie referencji do jednego z komponentów okna **ROI Manager**, służącego do wyświetlania listy komórek. Pozwala on na nasłuchiwanie na zmianę zawartości, w szczególności usunięcie jednego lub wielu

elementów. Po otrzymaniu takiego powiadomienia, stan listy porównywany jest z kolekcją komórek, które powinny aktualnie być wyświetlane – te których brakuje na liście usuwane są również z kolekcji. Rozwiążanie to nie jest wyjątkowo stabilne, ponieważ dokonuje konwersji typów bazując na strukturze komponentów z jakich składa się okno **ROI Manager**. Wymaga również każdorazowej ingerencji w to okno podczas jego uruchamiania.

## Rozdział 4.

# Zakończenie

### 4.1. Podsumowanie

Efektem niniejszej pracy jest narzędzie, mające usprawnić proces pozyskiwania danych na temat cyklu komórkowego prątków z gatunku *Mycobacterium smegmatis*. Narzędzie to zostało stworzone jako plugin dla programu ImageJ, powszechnie używanego przez naukowców zajmujących się mikrobiologią. Wtyczka pozwala na oznaczanie komórek na kolejnych klatkach nagrania przedstawiającego cykl komórkowy, tworząc w ten sposób strukturę danych która w późniejszych etapach służy do dokładnej analizy obrazów. Oznaczanie komórek może odbywać się w sposób w pełni automatyczny, może być wspierana przez użytkownika poprzez wprowadzanie poprawek lub może odbywać się w pełni manualnie. Narzędzie zapewnia możliwość zapisu i odczytu zaznaczeń z pliku. Wtyczka wyposażona jest dodatkowo w narzędzia pomagające w analizie cyklu komórkowego poprzez dokonywanie pomiarów, a także tworzenie profili intensywności pod oznaczeniami komórek. Dane można wyświetlić w formie wykresów lub wyeksportować w surowej formie w celu analizy poza programem ImageJ (np. w RStudio).

W pełni automatyczne oznaczanie i śledzenie komórek nie zawsze jest możliwe. Efekty tego podejścia są zadowalające tylko w przypadku dobrych jakościowo nagrań. Mimo to, nawet w trybie w pełni manualnym narzędzie może usprawnić pracę naukowców. Plugin pozwala na rozdzielenie pracy nad zaznaczaniem komórek, od wykonywania dalszych pomiarów. Dzięki temu poszczególne kroki mogą być wykonywane przez inne osoby. Tworzona w pierwszych krokach struktura zawiera informacje o rozwoju całej populacji komórek. Może zostać zapisana i wczytywana z pliku, dzięki czemu użytkownik może podzielić swoją pracę w czasie, wykorzystywać wielokrotnie raz utworzone zaznaczenia, a także przesyłać je do innych użytkowników analizujących to samo nagranie. Sam proces manualnego tworzenia zaznaczeń również został usprawniony poprzez możliwość skopiowania zaznaczeń z poprzedniej klatki oraz stworzenie specjalnych narzędzi do wykonywania najczęściej powtarzających się czynności.

## 4.2. Ograniczenia zastosowanych metod

Przyjęta metoda automatycznego oznaczania i śledzenia komórek nie jest wolna od ograniczeń. Zakłada ona, że na obrazie binarnym, tworzonym na etapie szkieletyzacji, komórki (lub grupy komórek) są oddzielone od otoczenia. Założenie to jest prawie zawsze spełnione w miejscach, w których w okolicy grup komórek nie są widoczne żadne artefakty lub stałe elementy płytek po których poruszają się komórki. Im lepsza była jakość nagrania (mniej szumów, mniej artefaktów, ostrzejsze krawędzie), tym lepiej radził sobie algorytm automatycznego śledzenia komórek. Warto więc w miarę możliwości zadbać o wysoką jakość nagrania.

Przeciętne obrazy spod mikroskopu przedstawiające analizowane eksperymenty, zawierały wiele artefaktów. Wady obrazów nakładające się z komórkami często powodują błędne ich oznaczanie lub przedłużanie ich końcówek. Zdarzało się również, że nagranie traciło ostrość w niektórych momentach lub cała scena przesuwała się nieznacznie w trakcie nagrania. W pierwszym przypadku powodowało to całkowicie błędne oznaczenie komórek. W drugim przypadku, odpowiednio duże przesunięcie również uniemożliwia poprawne odnalezienie lokalizacji komórek na podstawie oznaczenia z poprzedniej klatki.

Jeśli krawędzie pomiędzy komórkami nie są wystarczająco ostre na nagraniu, to w przypadku gdy pojawia się większa ilość komórek na raz, przestają one być oddzielone od siebie na obrazie binarnym. W efekcie szkielet całego obrazu nie zawiera szkieletów poszczególnych komórek, co w przyjętej metodzie uniemożliwia oznaczenie tych komórek.

## 4.3. Dalszy rozwój

Wymienione w poprzednim rozdziale problemy zastosowanej metody automatycznego śledzenia komórek można spróbować rozwiązać korzystając ze znanych algorytmów z dziedziny przetwarzania obrazów. Aby zniwelować wpływ artefaktów na efekt końcowy, można by spróbować zastosować na etapie wstępnego przetwarzania metody automatycznego ich usuwania[22]. Rozwiązaniem problemu związanego z przesuwającą się nieznacznie sceną w trakcie nagrania, mogłoby być wyliczenie przesunięcia (translacji) poszczególnych klatek względem dowolnej klatki referencyjnej, na podstawie stałych elementów tła. Takie dane mogłyby następnie zostać wykorzystane przy próbach określenia poprawnej lokalizacji komórek na podstawie poprzedniej klatki.

Tak jak wspomniałem w rozdziale 2.3.2., w niniejszej pracy nie zajmowałem się problemem określenia liczby i lokalizacji poszczególnych komórek bez wcześniejszego udziału użytkownika. Nawet w trybie automatycznym użytkownik zobowiązany jest do zaznaczenia wszystkich komórek obecnych na pierwsiowej klatce nagrania. Nie jest

to duży wymóg, ponieważ komórek tych jest wtedy zazwyczaj niewiele. Zaimplementowanie automatycznej detekcji komórek dla obrazów z niewielką ich ilością mogłoby być jednak usprawnieniem aktualnego programu.

Podczas implementacji pluginu starałem się podejmować decyzje, które pozwoliłyby w łatwy sposób rozwijać dalej ten projekt. W rozdziale 3.4.3. opisałem w jaki sposób można w łatwy sposób „wymienić” dowolny etap pluginu, tj. lokalizację i wstępna detekcję komórek, automatyczne śledzenie komórek, dokonywanie pomiarów i wizualizacji na podstawie zaznaczeń. Dzięki zastosowanemu podejściu, kod pluginu może zostać wykorzystany jako baza dla kolejnych prób znalezienia zupełnie nowej metody automatycznego oznaczania i śledzenia komórek.



# Bibliografia

- [1] World Health Organization, *Tuberculosis*, <https://www.who.int/news-room/fact-sheets/detail/tuberculosis>, 2020.
- [2] Damian Trojanowski, Marta Kołodziej, Joanna Hołówka, Rolf Müller, Jolanta Zakrzewska-Czerwińska, *Watching DNA Replication Inhibitors in Action: Exploiting Time-Lapse Microfluidic Microscopy as a Tool for Target-Drug Interaction Studies in Mycobacterium*, Antimicrobial Agents and Chemotherapy, American Society for Microbiology Journals, 63(10), 2019.
- [3] Joanna Hołówka, Damian Trojanowski, Katarzyna Ginda, Bartosz Wojtaś, Bartłomiej Gielniewski, Dagmara Jakimowicz, Jolanta Zakrzewska-Czerwińska, *HupB Is a Bacterial Nucleoid-Associated Protein with an Indispensable Eukaryotic-Like Tail*, mBio, American Society for Microbiology, 8(6), 2017.
- [4] Walter Lang, *Nomarski differential interference-contrast microscopy*, ZEISS Information, 70:114–120, 1968.
- [5] ImageJ User Guide, *Fiji*, <https://imagej.net/Fiji>.
- [6] RStudio, *RStudio IDE*, <https://rstudio.com/products/rstudio>.
- [7] H. B. Rachna, M. S. Mallikarjuna Swamy, *Detection of Tuberculosis Bacilli using Image Processing Techniques*, International Journal of Soft Computing and Engineering, 3:47-51, 2013.
- [8] P. Sadaphal, J. Rao, G. W. Comstock, M. F. Beg, *Image processing techniques for identifying Mycobacterium tuberculosis in Ziehl-Neelsen stains*, The international journal of tuberculosis and lung disease : the official journal of the International Union against Tuberculosis and Lung Disease, 12(5):579–582, 2008.
- [9] Wikipedia, *Caenorhabditis elegans*, [https://pl.wikipedia.org/wiki/Caenorhabditis\\_elegans](https://pl.wikipedia.org/wiki/Caenorhabditis_elegans).
- [10] Javier Fernández, *Image Processing to Detect Worms*, Uppsala Universitet, Uppsala, Sweden, 2010.
- [11] Jan J Koenderink, Andrea J van Doorn, *Surface shape and curvature scales*, Image and Vision Computing, 10:557–565, 1992.

- [12] Johannes Schindelin, *Shape Index Map*, 2010, [https://imagej.net/Shape\\_Index\\_Map](https://imagej.net/Shape_Index_Map).
- [13] Ta-Chih Lee, Rangasami L. Kashyap, Chong-Nam Chu, *Building skeleton models via 3-D medial surface/axis thinning algorithms*, Computer Vision, Graphics, and Image Processing, 56(6):462–478, 1994.
- [14] Ignacio Arganda-Carreras, *Skeletonize3D*, 2.1.1, 2017, <https://imagej.net/Skeletonize3D>.
- [15] Ignacio Arganda-Carreras, *AnalyzeSkeleton*, 3.3.0, 2018, <https://imagej.net/AnalyzeSkeleton>.
- [16] Wikipedia, *Ramer–Douglas–Peucker algorithm*, [https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker\\_algorithm](https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm).
- [17] Urs Ramer, *An iterative procedure for the polygonal approximation of plane curves*, Computer Graphics and Image Processing, 1(3):224–256, 1972.
- [18] David Douglas, Thomas Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer, 10(2):112–122, 1973.
- [19] ImageJ User Guide, *Segmented Line Selection Tool*, <https://imagej.nih.gov/ij/docs/guide/146-19.html#sub:Segmented-Line-Selection>.
- [20] Apache Maven Project, *Introduction to the Standard Directory Layout*, <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>.
- [21] ImageJ User Guide, *Philosophy*, <https://imagej.net/Philosophy>.
- [22] Jinwei Gu, Ravi Ramamoorthi, Peter Belhumeur, Shree Nayar, *Removing Image Artifacts Due to Dirty Camera Lenses and Thin Occluders*, ACM Trans. Graph., 28(12), 2009.