

Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

**METODY WIZUALIZACJI DLA ZMIENIAJĄCYCH SIĘ W CZASIE ZBIORÓW DANYCH
O STRUKTURZE HIERARCHICZNEJ**

Mateusz Matela

Promotor
dr inż. Dawid Weiss

Poznań, 2010

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

1 Wstęp	1
1.1 Cel i zakres pracy	5
2 Przegląd literatury	7
2.1 Ogólne spojrzenie na metody wizualizacji zmian danych w czasie	7
2.2 Ogólne spojrzenie na wizualizację struktur hierarchicznych	8
2.2.1 Wizualizacja poprzez węzły	8
2.2.2 Rozwój metody <i>treemap</i>	9
2.2.3 Inne metody podobne do <i>treemap</i>	12
2.2.4 Pozostałe metody	15
2.3 Wizualizacja struktur hierarchicznych z uwzględnieniem zmian w czasie	17
2.4 Podsumowanie	19
3 Foamtree — wizualizacja inspirowana bąbelkami w pianie	21
3.1 Wizualizacja statycznej hierarchii	22
3.1.1 Ogólna struktura układu	22
3.1.2 Siły działające w układzie	23
3.1.3 Zmiany topologiczne	29
3.1.4 Naprawa niepoprawnego stanu bąbelków	30
3.1.5 Generowanie stanu początkowego	33
3.1.6 <i>Pseudo-biegunowy</i> układ współrzędnych	35
3.1.7 Uwzględnianie grubości ścianek bąbelków	35
3.1.8 Wizualizacja bąbelków	37
4 Badanie własności i zbieżności modelu	39
4.1 Badane parametry modelu	39
4.2 Konfiguracja mechanizmu redukcji ścian	40
4.2.1 <i>Squeeze threshold</i>	40
4.2.2 <i>Squeeze angle</i>	45
4.2.3 <i>Squeeze candidate times</i>	48
4.3 Równowaga sił w modelu fizycznym	53
4.3.1 <i>Pressure factor</i>	53
4.3.2 <i>Spring factor</i>	56
4.3.3 <i>Angle factor</i>	56
4.4 Stabilizacja i kontrola szybkości wierzchołków	61
4.4.1 <i>Speed limit</i>	61
4.4.2 <i>Minimum speed</i>	61
4.4.3 <i>Slowdown period</i>	66

4.5 Podsumowanie	66
5 Wizualizacja zmian przy użyciu animacji	69
5.1 Animacja „rozrywająca” pianę	70
5.2 Animacja „ciągła” — odwarzanie przebiegu relaksacji	72
5.3 Animacja „rozrywająca” ze stanem końcowym zależnym od początkowego	76
5.4 Zastosowanie animacji w problemach praktycznych	77
5.5 Podsumowanie	79
6 Zakończenie	83
6.1 Uwagi na temat implementacji	83
6.2 Dalsze perspektywy rozwoju	84
Literatura	85

Rozdział 1

Wstęp

Dzięki wykorzystaniu komputerów zbieranie i przechowywanie danych jest proste. Odkrywanie wiedzy i czerpanie wniosków z takich surowych danych jest jednak procesem trudnym. Prócz statystyki, która leży u podstaw wszystkich innych dziedzin analizy danych, w dziedzinie informatyki przetwarzaniem danych zajmują się również eksploracja danych (ang. *data mining*), uczenie maszynowe (ang. *machine learning*), czy też wyszukiwanie wzorców i informacji (ang. *pattern discovery, information retrieval*). W wymienionych dziedzinach, prócz metod czysto obliczeniowych, równie ważnym zagadnieniem jest wizualizacja wyników (i danych), czyli poszukiwanie sposobów przedstawienia danych w taki sposób, aby użytkownik mógł je jak najszybciej i najłatwiej zrozumieć i wyciągnąć z nich interesujące wnioski. Właśnie temat wizualizacji danych będzie poruszony w niniejszej pracy.

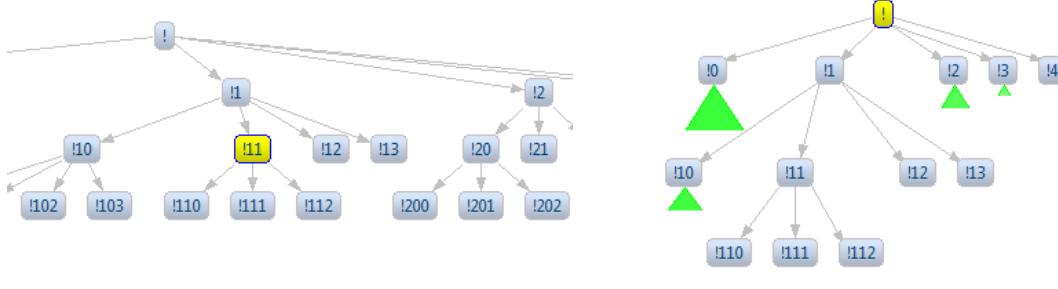
Sposób przedstawienia konkretnych danych zależy przede wszystkim od ich charakteru. Najczęściej spotykane są tabele z liczbami, którym często towarzyszą statystyki opisowe. Takie dane najłatwiej przedstawić w formie wykresów kolumnowych lub słupkowych. Wizualizacja tabelarycznych danych ma długą historię, a powszechnie dostępne oprogramowanie (na przykład arkusze kalkulacyjne) udostępnia bogate możliwości jej stosowania. Stosunkowo rzadziej spotykane w formie wizualizacji, choć równie istotne, są dane o strukturze grafowej, w szczególności hierarchicznej. Strukturę hierarchiczną warto zastosować zawsze wtedy, gdy chcemy przedstawić podział jakiegoś obiektu lub zbioru na części, które znów składają się z jeszcze mniejszych części, i tak dalej. Podobnie w drugą stronę — mając wiele elementów, chcemy je przyporządkować do pewnych zbiorów, a te zbiory dalej łączyć w coraz większe jednostki.

Struktury hierarchiczne najczęściej przedstawiane są w formie graficznej w postaci *drzew*, czyli grafów o jednym węźle początkowym, który rozbija się na podwzły, zwykle połączone liniami prostymi. Taki sposób pozwala najlepiej uchwycić zależności między elementami, jednak staje się bardzo kłopotliwy w przypadku dużej liczby elementów. W przeciętnym drzewie liczba elementów na kolejnych poziomach rośnie wykładniczo, przez co niezależnie od sposobu rozłożenia poszczególnych poziomów, szybko zaczyna brakować miejsca. Istnieją różne podejścia mające na celu rozwiązanie tego problemu (stosuje się również łączenie różnych technik wymienionych poniżej).

Ograniczony widok pełnego drzewa. Metoda ta polega na rozmieszczeniu pełnej struktury drzewa na powierzchni większej niż powierzchnia prezentacji (zwykle ekranu komputera) i umożliwieniu użytkownikowi przesuwania fragmentarnego „widoku” nad pełnym obszarem prezentacji (zob. rys. 1.1a). Rozwiążanie to utrudnia całosciowe spojrzenie na strukturę drzewa i wymaga od użytkownika pamiętań położenia aktualnie oglądanego obszaru drzewa względem innych elementów.

Ukrywanie fragmentów drzewa. Innym rozwiązaniem jest ukrywanie (zwijanie) fragmentów drzewa.

Jest to bardzo dobre rozwiązanie, jeśli użytkownik jest zainteresowany konkretną gałęzią — wtedy pozostałe gałęzie mogą być ukryte i nie rozpraszały uwagi (zob. rys. 1.1b). Jednak, podobnie jak wcześniej, uchwycenie ogólnej struktury drzewa staje się niemożliwe. Połączenie ukrywania fragmentów drzewa z ograniczonym widokiem jego struktury jest stosowane w programach do zarządzania systemem plików (np. *Internet Explorer* w systemie Windows).



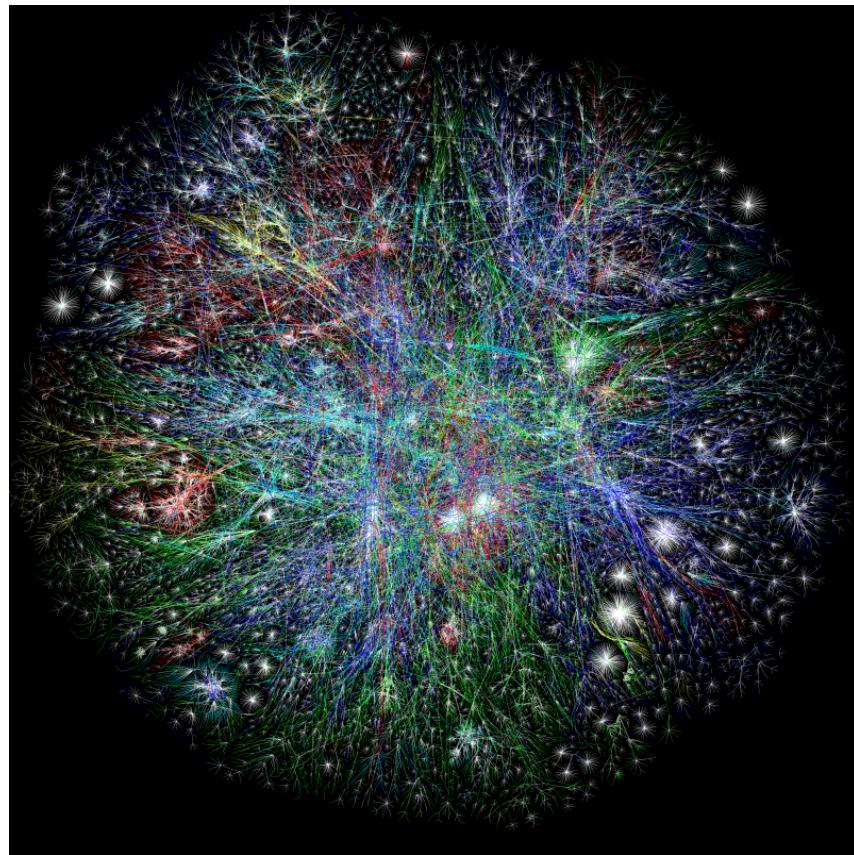
Rysunek 1.1: Podstawowe sposoby prezentacji drzewa.

Ukrywanie informacji. Inna metoda wizualizacji drzew i dużych struktur grafowych polega na ukrywaniu części informacji. Przykładowo, prezentowanie węzłów jako pojedynczych punktów (lub pominięcie ich wizualizacji w ogóle) pozwala na pokazanie dużo większego zbioru danych (zob. rys. 1.2). W takim przypadku niezbędne są dodatkowe mechanizmy przedstawiania ukrytych danych na życzenie użytkownika. Zwykle polegają one na wyświetlaniu informacji w panelu bocznym lub dodatkowym okienku, pojawiającym się po najechaniu kursem lub kliknięciu na wybranym elemencie.

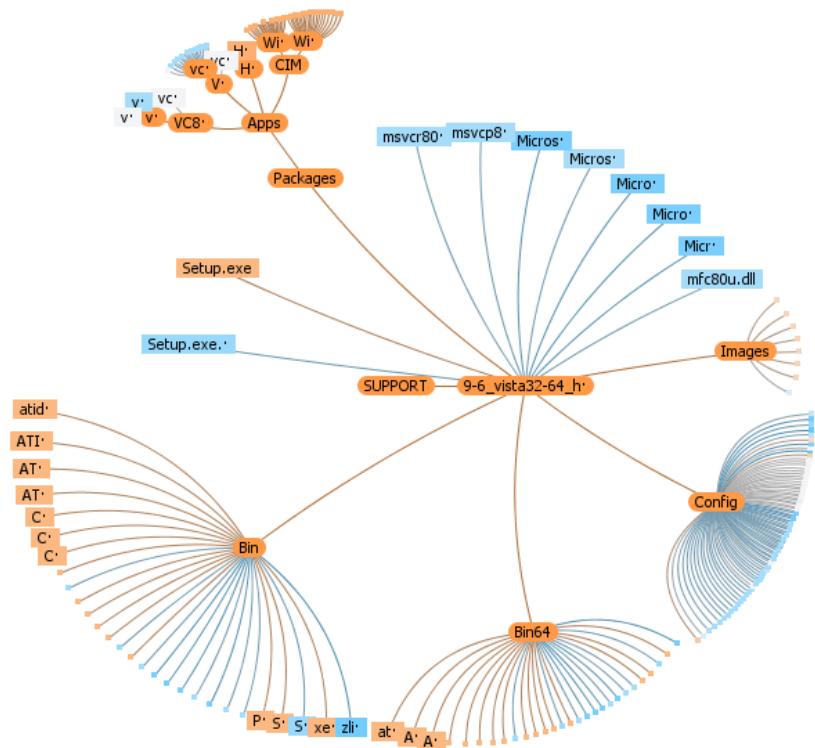
Zmiana odwzorowania przestrzeni. Do rzadziej spotykanych podejść należy wykorzystanie przekształceń geometrycznych przedstawianej powierzchni. Najważniejszym przykładem są tutaj drzewa hiperboliczne, w których elementy rozmieszczane są w przestrzeni nieskończonej, a następnie odwzorowywane na figurę o ograniczonej powierzchni (zwykle okręgu). Daje to efekt „rybiego oka”, czyli elementy w środkowej części są duże i dobrze widoczne, a im bliżej znajdują się krawędzi, tym stają się mniejsze i bardziej zniekształcone (zob. rys. 1.3). Oczywiście, widok można dowolnie przesuwać, aby skupić się na innych fragmentach zbioru danych.

Zupełnie inną metodą wizualizacji struktur drzewiastych są odwzorowania powierzchniowe, czyli *treemaps* (w związku z brakiem polskiego odpowiednika, zdecydowano się używać terminu oryginalnego). Podstawowa wersja tej metody przedstawia całe drzewo w formie prostokąta podzielonego na mniejsze prostokąty, odpowiadające poszczególnym gałęziom. Prostokąty te są rekurencyjnie dzielone na coraz mniejsze, aż do najmniejszych jednostek, odpowiadających liściom (zob. rys. 1.4a). Pole powierzchni danego prostokąta wyraża informację o jego wadze w stosunku do innych węzłów (zwykle liczby podwęzłów lub liści). Na przykład, w przypadku struktury katalogów na dysku, wielkości prostokątów mogą odzwierciedlać sumaryczną wielkość plików w folderze. Koncepcja *treemap* zdobyła popularność i zaproponowano dla niej wiele rozszerzeń, polegających zarówno na optymalizacji ułożenia elementów odwzorowania, jak i na zastosowaniu innych figur geometrycznych niż prostokąty (zob. rys. 1.4c i 1.4b).

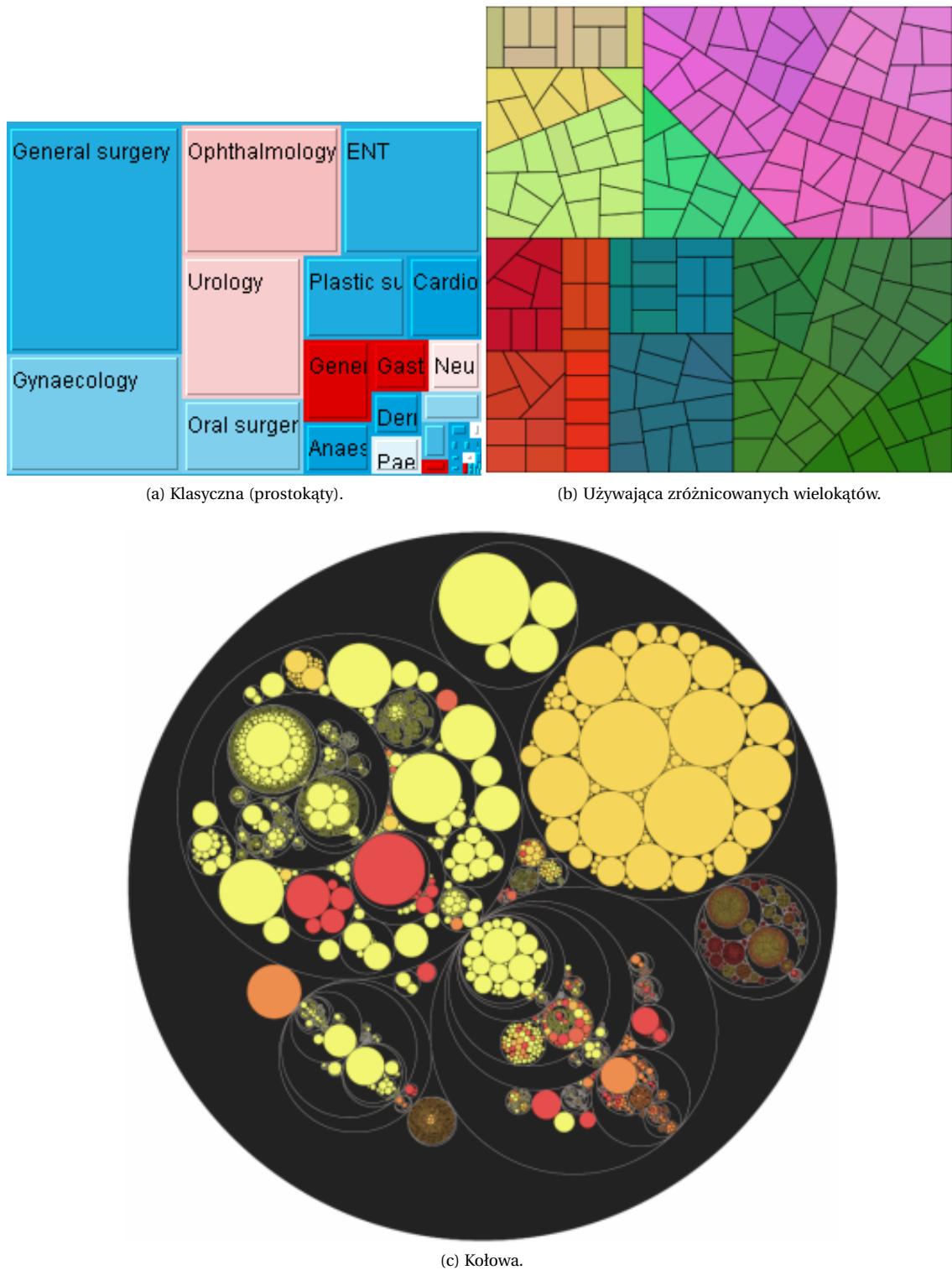
Jak widać ze wstępnej analizy, temat wizualizacji struktur hierarchicznych jest bogaty. Powyższy przegląd dotyczy jedynie części istniejących metod wizualizacji. Istnieje jednak aspekt, na którym do



Rysunek 1.2: Graf przedstawiający strukturę internetu — ekstremalny przykład wizualizacji dużej struktury z ukrywaniem szczegółów. Źródło: <http://matthewgress.com/>.



Rysunek 1.3: Zmiana odwzorowania przestrzeni na przykładzie drzew hiperbolicznych. Źródło: <http://www.randelschofer.ch/treeviz/>.



Rysunek 1.4: Różne sposoby wizualizacji metodą *treemap*. Źródła: <http://en.wikipedia.org/>, <http://people.csail.mit.edu/konak>, <http://lip.sourceforge.net/ctreemap.html>.

tej pory się nie koncentrowano, a który może przyczynić się do odkrycia zupełnie nowych możliwości wykorzystania wizualizacji. Aspektem tym jest przedstawienie nie tylko struktury drzewa i pewnych właściwości jego elementów, ale też *jego zmian w czasie* (tzn. zmian struktury hierarchicznej, która podlega modyfikacjom). Tradycyjne sposoby pokazania upływu czasu opierają się na statycznych, dwuwymiarowych wykresach, w których jedna z osi reprezentuje czas. W przypadku złożonych struktur nie można sobie na to pozwolić, ponieważ dwa wymiary są niezbędne do przedstawienia pojedynczego stanu. Niezbędne wydaje się więc zastosowanie animacji, która pokaże przejścia pomiędzy stanami całego drzewa w czasie. Z wielu możliwych zastosowań takiego algorytmu wizualizacji warto wymienić poniższe:

- przedstawienie zmienności projektów programistycznych w repozytoriach plików takich jak Subversion,
- przedstawienie zmienności skupień (*clusters*) w artykułach prasowych publikowanych w serwisach RSS,
- przedstawienie zmienności skupień dokumentów w odpowiedzi na modyfikację parametrów algorytmów grupujących.

1.1 Cel i zakres pracy

Celem pracy jest **opracowanie metod wizualizacji dla zmiennych w czasie struktur hierarchicznych**. Algorytmy wizualizacji powinny uwzględniać zmiany zarówno właściwości poszczególnych elementów (wagi), jak i ich położenia w strukturze danych wejściowych (dodawanie i usuwanie elementów). Animacje powinny umożliwiać wizualizację zmian w czasie zarówno w przód, jak i w tył.

Zadania szczegółowe to:

- przegląd literatury związanej z tematem oraz dostępnego oprogramowania implementującego *treemaps* lub podobne techniki wizualizacji,
- analiza wybranych metod wizualizacji pod kątem wykorzystania ich do animacji przedstawiających zmiany w czasie, ewentualnie zaproponowanie modyfikacji ułatwiających to zadanie,
- implementacja wybranych metod w formie biblioteki ogólnego przeznaczenia,
- implementacja wybranego zastosowania dla zaproponowanej metody.

Rozdział 2

Przegląd literatury

2.1 Ogólne spojrzenie na metody wizualizacji zmian danych w czasie

W pracy [MS03] Müller i Schumann dokonują przeglądu istniejących metod wizualizacji danych zmieniających się w czasie, wraz z zarysem historycznym. Przegląd ten jest bogaty i pokazuje jak wiele dokonano w tej dziedzinie. Wskazuje też, że każda metoda sprawdza się dobrze tylko w specyficznych warunkach, a gdy te nie są spełnione, prezentacja staje się trudna do zrozumienia — dochodzi do natłoku informacji, zmniejsza się czytelność. Odpowiedni dobór metody wizualizacji do charakteru danych i zadań, jakie należy na nich wykonać, nadal jest więc zadaniem trudnym, wymagającym analizy każdego przypadku z osobna. Mimo że przegląd obejmuje metody przeznaczone dla różnorodnych danych (wartości nominalne, skalarne, jedno- i wielowymiarowe), żadna z nich nie nadaje się do prezentacji zależności hierarchicznych — ten aspekt jest więc stosunkowo mało rozwinięty.

Aigner i inni w artykule [AMM⁺07] przyglądarki się sposobom prezentacji danych związanych z czasem i proponują dla nich podział na kategorie. Podział ten jest bardzo ogólny i opiera się na podstawowych cechach wyróżniających konkretne podejścia. Opierając się na tej propozycji, metody wizualizacji, które zostaną zaproponowane w niniejszej pracy można sklasyfikować następująco.

- Ze względu na sposób traktowania wymiaru czasu:
 - dzielenie czasu na dyskretne punkty (w odróżnieniu od dłuższych, ciągłych odcinków),
 - rozpatrywane upływającego czasu jako linii prostej (inne możliwości to cykl lub rozgałęziające się ścieżki).
- Ze względu na rodzaj obsługiwanych danych:
 - wiele wymiarów zmieniających się w czasie (wielkość elementów, kolor, zależności hierarchiczne).
- Ze względu na sposób prezentacji danych:
 - przedstawienie upływu czasu w formie animacji (w odróżnieniu od statycznych obrazów, np. wykresów),
 - grafika dwuwymiarowa.

W powyższym artykule znajdują się również przydatne spostrzeżenia co do tego, jakie wymagania powinny spełniać narzędzia do wizualizacji danych, w szczególności te koncentrujące się na zależnościach czasowych.

Udostępnienie wielu różnorodnych metod wizualizacji. Wynika to stąd, że potrzeby użytkowników są różne, w zależności od rodzaju posiadanych przez nich danych oraz zadań, których wykonanie ma ułatwić prezentację.

Interaktywność. Użytkownik powinien mieć możliwość swobodnej zmiany pewnych aspektów wizualizacji, na przykład jaki przedział czasu jest wizualizowany, jak szybko działa animacja, i tak dalej.

Metody analizy danych. Jest to rozwinięcie myśli z poprzedniego punktu. Wizualizacja powinna umożliwiać przeszukiwanie i filtrowanie oraz zmianę zakresu przedstawianych danych i poziomu szczegółowości — operacje takie są niezbędne w przypadku dużych zbiorów danych.

2.2 Ogólne spojrzenie na wizualizację struktur hierarchicznych

Większość metod wizualizacji struktur hierarchicznych można przydzielić do jednej z dwóch kategorii: rozłożenie węzłów lub wypełnianie przestrzeni. W podpunktach poniżej zostaną szczegółowo omówione obie kategorie, a także podane zostaną przykłady metod nie mieszczących się w tej klasyfikacji.

2.2.1 Wizualizacja poprzez węzły

Jak wspomniano we wstępie, najpopularniejsze metody wizualizacji hierarchii polegają na przedstawieniu jej w postaci *drzewa*, czyli zbioru węzłów i łuków łączących dzieci z ich rodzicami. Można je często spotkać w podręcznikach i artykułach prasowych, gdzie wyjaśniają budowę systemów lub klasyfikację pojęć. Każdy użytkownik komputerów z pewnością widział kontrolkę służącą do przeglądania zawartości drzewa, w której węzły ułożone są jeden pod drugim, a każdy kolejny poziom przesunięty jest nieco w prawo. Kontrolka taka daje możliwość swobodnego zwijania i rozwijania wybranej gałęzi, czyli ukrycia wszystkich potomków danego elementu. O popularności tej metody świadczy fakt, że obecnie niemal wszystkie systemy operacyjne i biblioteki do tworzenia interfejsów użytkownika udostępniają taką formę zarządzania plikami.

Istnieje wiele modyfikacji tej prostej koncepcji, jednak są one mniej popularne i znajdują zastosowanie jedynie w specjalistycznych aplikacjach. Ciekawym przykładem jest *cone tree* (przedstawiony w [RMC91]), który rozmieszcza węzły w przestrzeni trójwymiarowej. Elementy mające wspólnego przodka umieszczane są tuż pod nim w kształcie okręgu, tak że łuki łączące je z przodkiem tworzą stożek. Dzięki wykorzystaniu trójwymiarowej przestrzeni metoda ta pozwala na przedstawienie struktur o większej liczbie węzłów niż tradycyjne wizualizacje 2D, jednak rodzi to dodatkowe problemy związane z wzajemnym przykrywaniem się węzłów i trudniejszą obsługą — potrzebne są dodatkowe mechanizmy kontrolujące to, które elementy są widoczne na pierwszym planie (animowane obroty stożków).

Inna technika, nazwana w [PGB02] *space tree*, łączy klasyczne ułożenie węzłów (kolejne poziomy drzewa ułożone pionowo lub poziomo) z zaawansowanymi mechanizmami ukrywania poddrzew. Po wybraniu interesującego elementu przez użytkownika, system automatycznie wyświetla jego potomków (do tyłu poziomów, ile zmieści się na ekranie), jednocześnie *zwijając* niezwiązane z nim gałęzie. Aby użytkownik nie stracił orientacji w położeniu i widoczności elementów, wszelkie zmiany przedstawiane są w postaci odrębnych animacji. Każda zmiana węzła centralnego składa się więc z trzech animowanych kroków: ukrycia nieinteresujących węzłów, przesunięcia pozostałych węzłów (w celu dopasowania do ułożenia rozwijanych fragmentów) i odkrycia nowych elementów. Dodatkowo, w miej-

scu ukrytego poddrzewa wyświetlany jest piktogram dostarczający o nim dodatkowych informacji, na przykład trójkąt, którego wysokość jest proporcjonalna do wysokości poddrzewa, a szerokość — do liczby liści. Dzięki temu, mimo że użytkownik nie widzi całego drzewa, ma częściowe pojęcie o jego ogólnej strukturze.

Kolejnym wartym uwagi typem wizualizacji są techniki inspirowane obrazami uzyskiwanymi w obiektywach typu rybie oko (ang. *fisheye*). W metodach tych wyróżnione jest ognisko (ang. *focus*), które użytkownik może zmieniać w czasie rzeczywistym. Obiekty leżące najbliżej tego punktu są przedstawiane najdokładniej i zajmują najwięcej dostępnego miejsca, a im dalej od niego — tym są mniejsze i gęściej ułożone. Jedna z pierwszych implementacji takiej metody, opisana w [SB92], powiększała wybrany fragment obrazu poprzez liniowe przekształcenie współrzędnych, niezależne dla obu osi. Autorzy koncentrowali się na wizualizacji grafów w ogólności, a więc zastosowanie jej do struktur drzewiastych jest również możliwe. Bardziej zaawansowana technika — przedstawiona w [LR95] *hiperbolic tree* — wykorzystuje model dysku Poincaré do odwzorowania przestrzeni hiperbolicznej na powierzchnię koła. Węzły drzewa ułożone są radialnie — korzeń drzewa znajduje się na środku, a gałęzie rozchodzą się we wszystkich kierunkach (możliwe jest też rozchodzenie w jednym kierunku, co prowadzi do bardziej klasycznego układu). Podobną techniką jest *magic eye view*, przedstawiony w *Kreuseler:MEV*, który rozkłada węzły drzewa na powierzchni półkuli, a następnie rzutuje je na powierzchnię koła, również z uwzględnieniem ogniska (punktu skupienia).

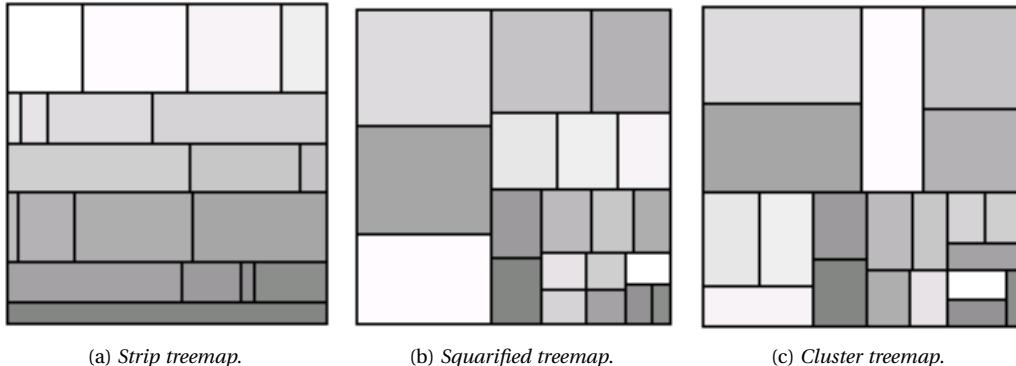
2.2.2 Rozwój metody *treemap*

Pomysł na odwzorowanie struktury hierarchicznej poprzez podział powierzchni prostokąta na coraz mniejsze figury został po raz pierwszy przedstawiony w artykule [Shn92]. Zastosowana w nim metoda ułożenia elementów (znana jako *slice and dice*) polegała na podzieleniu dostępnego prostokąta na równoległe pasy, o grubościach dobranych tak, by łącznie zajmowały całą dostępną przestrzeń, a ich wielkości były proporcjonalne do wizualizowanych wartości atrybutów. Na pierwszym poziomie prostokąt jest dzielony w pionie, tak otrzymane prostokąty dzielone są w poziomie, następnie znowu w pionie, i tak dalej. Ta prosta metoda jest rzadko wykorzystywana w praktyce, ponieważ często prowadzi do powstania długich i wąskich prostokątów — zwłaszcza gdy elementów jest dużo, lub wartości atrybutów znacznie się różnią. Jest to niepożądany efekt, który utrudnia przeglądanie i wybór prostokątów, porównywanie ich wielkości oraz umieszczanie na nich opisów.

Powyższy problem został rozwiązyany dzięki algorytmom, w których kształty elementów są zbliżone do kwadratu. Należą do nich między innymi.

- *Strip treemap* (temu algorytmowi nie został poświęcony osobny artykuł, jego implementację i porównanie z innymi metodami można znaleźć na stronie [WB09]) — dzieli dostępną przestrzeń na równoległe pasy i do każdego z nich przydziela po kilka kolejnych elementów. Do każdego kolejnego pasa przydzielane jest tyle elementów, aby ich średnia proporcja rozmiarów była jak najbliższa jedności (algorytm działa zachłannie). Algorytm ten jest stosunkowo mało efektywny pod względem proporcji uzyskanych prostokątów, ale jego zaletą jest zachowanie kolejności elementów (zob. rys. 2.1a).
- *Squarefied treemap*, przedstawione w [BHvW99] — działa podobnie do algorytmu *strip treemap*, z tym że elementy układane są zawsze w kolejności od największego, a tworzone pasy nie muszą być równoległe (kierunek jest dobierany tak, aby kształt pozostałej części dostępnej przestrzeni był również zbliżone do kwadratu). Ta metoda układają prostokąty o najlepszych proporcjach (zob. rys. 2.1b).

- *Cluster treemap*, wykorzystany do wizualizacji notowań giełdowych ([Sma09]) i opisany w [Wat99] — działa podobnie do *squarified treemap* (zob. rys. 2.1c), ale dodatkowo optymalizuje ułożenie tak, by elementy o podobnej historii zmian leżały blisko siebie.



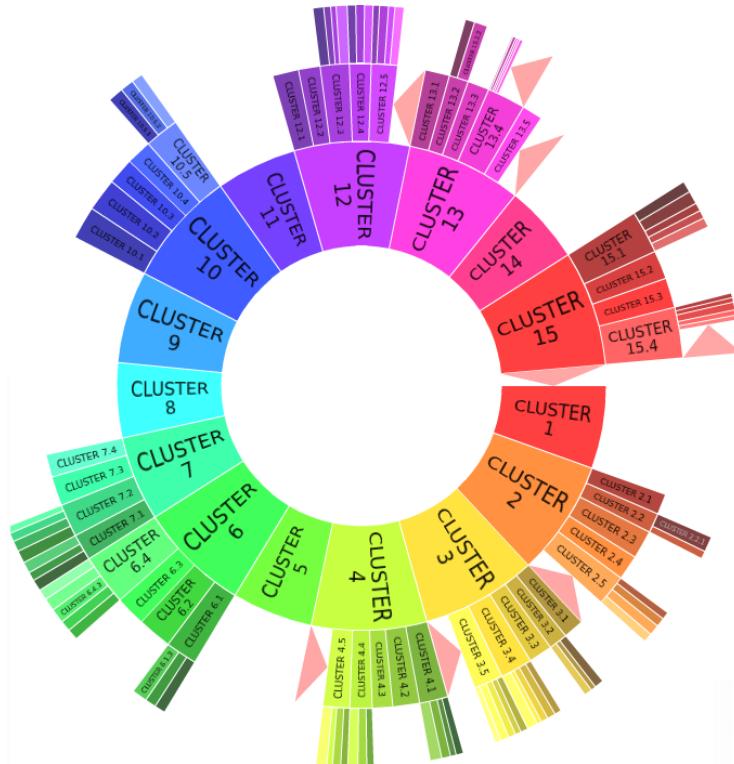
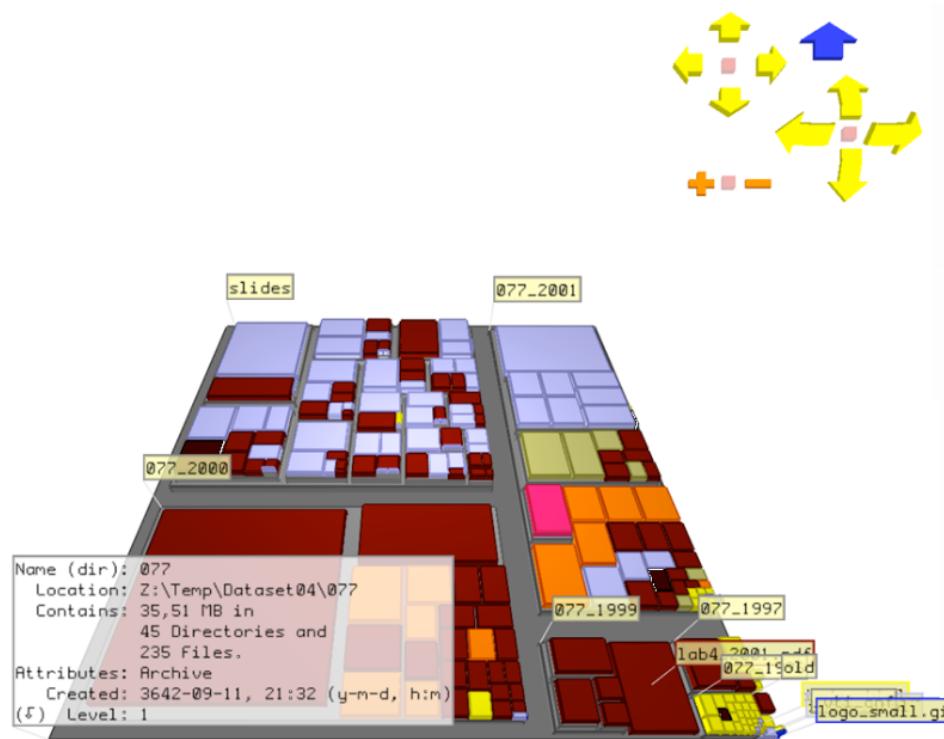
Rysunek 2.1: Przykłady różnych algorytmów podziału powierzchni w metodzie *treemap*. Źródło: [Shn92].

Innym problemem, z którym muszą się zmierzyć osoby projektujące wizualizacje typu *treemap*, jest czytelność przedstawianych zależności hierarchicznych. W tradycyjnym podejściu, gdy rysowane są tylko prostokąty odpowiadające liściom przedstawianej struktury, użytkownik może mieć problemy z rozróżnieniem które elementy są bezpośrednim rodzeństwem, a które mają dalszego wspólnego przodka. Stosunkowo proste rozwiązanie wykorzystuje pogrubianie krawędzi: im wyżej w hierarchii znajdują się elementy, tym grubsza linia oddzielająca odpowiadające im prostokąty. Idąc dalej w tym kierunku, można w każdym prostokącie dodać ramkę, która obejmuje wszystkie wewnętrzne prostokąty. Jeśli na ramkę zarezerwuje się więcej miejsca, daje to możliwość umieszczenia na niej opisu elementu, dzięki czemu będą opisane węzły na wszystkich poziomach, a nie tylko liście. Wadą stosowania grubych krawędzi lub obramowań jest to, że zajmują dodatkową przestrzeń, a przez to proporcje wielkości pomiędzy elementami zostają zachwiane. Na przykład, mimo że dwa prostokąty mają taką samą wielkość, mogą wizualizować różne wartości atrybutów, ponieważ jeden z nich jest położony nisko w hierarchii i dużą część „przydzielonej” mu powierzchni zajmuje szereg pogrubionych obramowań.

Istnieje jeszcze inne podejście do omawianego problemu, które nie ma powyższej wady — podkreślenie zależności hierarchicznych przy pomocy kolorów. Elementy będące liśćmi w tej samej gałęzi mają ten sam kolor lub różnią się tylko odcieniem, a im wyżej w hierarchii, tym różnice pomiędzy sąsiednimi zbiorami są większe. Za przykład wizualizacji działającej w ten sposób może posłużyć metoda wykorzystana w Carrot² [OW] (zob. rys. 2.2), choć akurat ona nie opiera się na technice *treemap* (zob. rozdz. 2.2.4). Oczywiście rozwiązanie to ma również słabą stronę — możliwość wizualizowania dodatkowej informacji przy pomocy koloru zostaje utracona lub mocno ograniczona.

Kolejny pomysł na zwiększenie czytelności przedstawionej hierarchii zaproponowano w artykule [WvdW99]. Opiera się on na dodaniu efektu wypukłości. Dzięki zastosowaniu odpowiedniego cieniowania, prostokąty zaczynają przypominać poduszki (stąd nazwa metody — *cushion treemap*). Wypukłość jest ustalana dla prostokątów na wszystkich poziomach hierarchii (z tym większą wagą, im bliżej korzenia leży dany węzeł), a następnie sumowana. Ostatecznie uzyskuje się złudzenie trójwymiarowej powierzchni, z której łatwo można wywnioskować, jak dzielą się kolejne gałęzie wizualizowanej struktury. Efekt cieniowania pozwala rozróżnić nawet prostokąty o jednolitej barwie bez rysowania ich krawędzi.

Po *cushion treemap*, kolejnym logicznym krokiem jest zastosowanie grafiki w pełni trójwymiarowej.

Rysunek 2.2: Carrot² — przykład wykorzystania kolorów do podkreślenia zależności hierarchicznych.

Rysunek 2.3: StepTree — trójwymiarowa wersja treemap. Źródło: [Bla05].

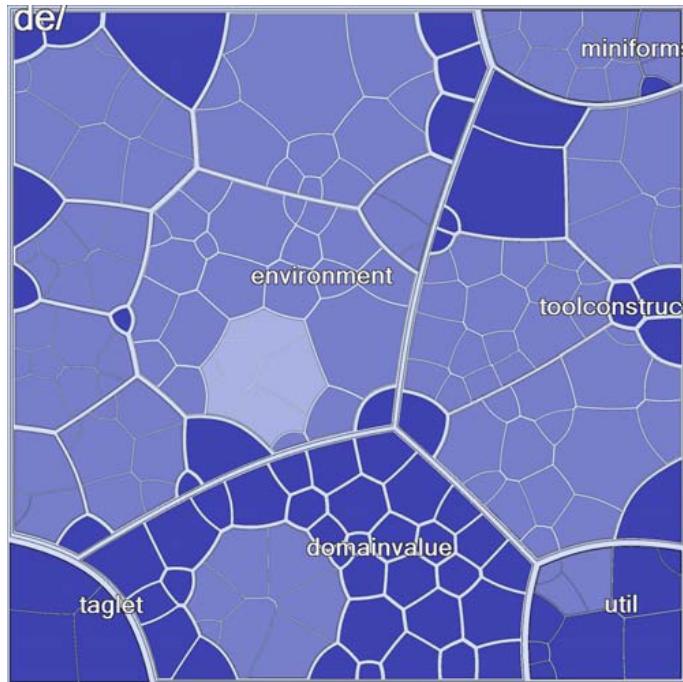
Podejście takie stosuje na przykład program *StepTree* (zob. rys. 2.3), opisany w [BCS04]. Zamiast prostokątów zawartych jeden w drugim, węzły przedstawiane są jako prostopadłościany leżące jeden na drugim. Wysokości wszystkich prostopadłościanów są równe, zatem poddrzewa o największej głębokości „odstają” najwyżej. Aby elementy potomne nie zakrywały w całości swojego przodka, zostają one lekko zmniejszone (jak bardzo, to zależy od zadanego parametru). Dzieje się tak na każdym poziomie, stąd kolejne poziomy przypominają stopnie schodów. Operacja ta jest analogiczna do pogrubiania krawędzi w tradycyjnej metodzie *treemap*, a więc prowadzi do zaburzeń utrudniających porównywanie wielkości elementów nie będących rodzeństwem. Zaburzenie to jest dodatkowo zwiększone przez operację wyrównywania, polegającą na tym, że największe elementy zostają pomniejszone, a najmniejsze powiększone (suma wielkości pozostaje stała). Mechanizm ten wprowadzono z myślą o wizualizacji systemu plików, aby umożliwić wyświetlenie pustych katalogów i małych plików.

Przeprowadzono eksperyment porównujący *StepTree* i tradycyjny *treemap* pod względem szybkości i poprawności wykonywania podstawowych zadań przez użytkowników (na przykład porównywanie elementów, znajdowanie największych). Nie stwierdzono jednak istotnych różnic. Jednym zadaniem w którym *StepTree* okazał się lepszy było odszukanie katalogu leżącego najbliżej w hierarchii. W drugim eksperymencie, opisanym w [Bla05], badano, jak wyświetlanie płynnej animacji przy przechodzeniu pomiędzy katalogami wpływa na orientację użytkowników w strukturze plików. Zadania polegały na kilkukrotnym przechodzeniu do katalogów o zadanych ścieżkach i z powrotem do katalogu głównego. Również w tym przypadku nie udało się zauważyc różnic w średnich rezultatach użytkowników. Co prawda użytkownicy korzystający z animacji czuli się pewniej i częściej przechodzili do wybranych katalogów „na skróty” (przeskakując kilka katalogów na raz), jednak część z nich „gubiła się” i musiała szukać drogi na nowo.

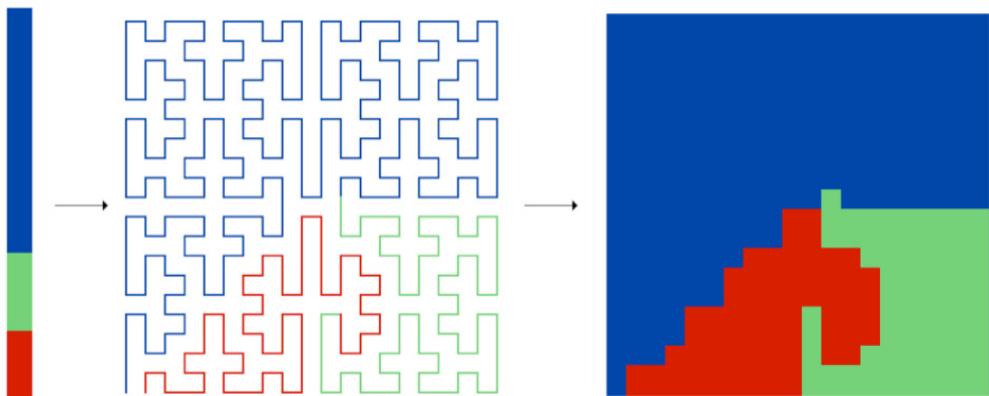
2.2.3 Inne metody podobne do *treemap*

W miarę wzrostu popularności metody *treemap*, zaczęły się pojawiać inne wizualizacje działające podobnie do niej, ale różniące się kształtem przedstawianych elementów. Rezygnacja z prostokątów otwiera szerokie możliwości, jednak stworzenie czytelnej wizualizacji tego typu nie jest proste. Niezależnie od zastosowanej formy odwzorowania przestrzeni, jednym z najważniejszych kryteriów dobrej wizualizacji tego typu są proporcje elementów. Figury powinny być zbliżone do koła lub kwadratu. Zagadnienie to bada Krzysztof Onak — w artykule [OS08] przedstawia formalny dowód, że dla dowolnego drzewa można zbudować taki podział kwadratu na wielokąty wypukłe, w którym proporcje każdego wielokąta spełniają określone ograniczenie (dokładniej, stosunek przekątnej do szerokości jest ograniczony wielomianem od wysokości drzewa i logarytmu z liczby liści). Został też stworzony przykładowy algorytm podziału oparty na tym dowodzie.

Ciekawa wizualizacja została przedstawiona w [BDL05] i [BD05] — *Voronoi treemap*. Opiera się na diagramie Woronoja, czyli takim podziale przestrzeni, w którym wyróżnia się n punktów początkowych (centroidów), po czym każdy punkt przestrzeni przypisuje się do zbioru najbliższego centroidu. Otrzymane zbiory to wielokąty wypukłe. Zauważono, że wielokąty te mają właściwości korzystne dla wizualizacji typu *treemap* (są regularne i mają proporcje zbliżone do wielokątów foremnych), gdy ich centra pokrywają się z ich środkami masy. Rozkład taki można otrzymać poprzez wielokrotne generowanie diagramu Woronoja i ustalanie nowych punktów początkowych tam, gdzie znajdują się centroidy wielokątów. Taki proces pozwala jednak uzyskać jedynie wielokąty o równych lub zbliżonych wielkościach. Wprowadzono więc wagi przypisane do każdego punktu początkowego i zmodyfikowano funkcję odległości wykorzystywaną przy podziale w taki sposób, aby do punktów z większą wagą przydzielone były wielokąty o większej powierzchni. Opracowano dwie takie funkcje, przy czym dla jednej z nich ścianki oddzielające wielokąty są odcinkami prostymi, a dla drugiej — krzywymi hiper-



Rysunek 2.4: Przykładowa wizualizacja typu *Voronoi treemap*. Źródło: [BDL05].



Rysunek 2.5: Przykładowa krzywa Peano i oparta na niej wizualizacja typu *jigsaw map*. Źródło: [Wat05].

bolicznymi. Wagi punktów początkowych są dostrajane w kolejnych iteracjach na zasadzie ujemnego sprzężenia zwrotnego, tak aby powierzchnia wielokąta była jak najbardziej zbliżona do zadanej wielkości. Dzięki korzystnym właściwościom zastosowanego podziału powierzchni, *Voronoi treemap* jest wizualizacją czytelną a zarazem atrakcyjną wizualnie (zob. rys. 2.4). Iteracyjne obliczanie rozkładu wielokątów jest jednak złożonym procesem, przez co wygenerowanie wizualizacji dla dużych zbiorów danych trwa bardzo długo (przy kilku tysiącach węzłów czas liczony jest w minutach).

Innym przykładem modyfikacji koncepcji *treemap* jest *jigsaw map* (zob. rys. 2.5) zaproponowany przez Wattenberga w [Wat05]. Podstawą działania *jigsaw map* jest krzywa Peano, czyli krzywa wypełniająca kwadrat (a właściwie jej uogólniona wersja, działająca dla dowolnych prostokątów). Wizualizacja polega na przeskalowaniu wielkości liści drzewa w taki sposób, by wszystkie były liczbami całkowitymi, a ich suma była równa liczbie punktów w prostokącie przechowującym wizualizację. Następnie, liście drzewa odwiedzane są w kolejności w głąb i odkładane na krzywej Peano po tyle punktów, ile wynosi ich przeskalowana wielkość. Ostatecznie, każdy liść ma przypisany spójny obszar, choć czę-

sto pojawiają się charakterystyczne „wypustki”, stąd porównanie tej wizualizacji do puzzli (ang. *jigsaw puzzle*). Wattenberg wyraża cztery ważne cechy dobrego odwzorowania typu *treemap* w sposób formalny.

Stabilność pod względem wielkości: Najmniejsza możliwa zmiana w rozmiarach elementów w drzewie wejściowym prowadzi do najmniejszej możliwej zmiany w wyglądzie wizualizacji.

Stabilność pod względem struktury: Rozbitie jednego liścia drzewa na dwa elementy o takiej samej łącznej wielkości sprawia, że w wizualizacji jedna figura zostanie zastąpiona przez dwie, a reszta figur pozostanie niezmieniona.

Zachowanie kolejności elementów: Jeśli dwa węzły występują obok siebie w drzewie (dotyczy to drzew ze zdefiniowaną kolejnością potomków), sąsiadują ze sobą również w wizualizacji.

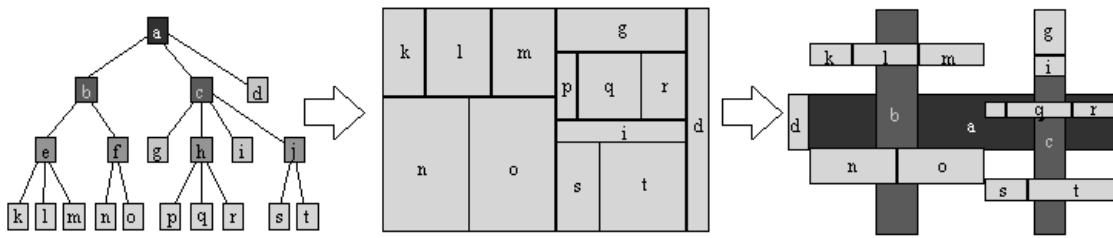
Ograniczenie proporcji figur: Przekątna każdej figury w wizualizacji jest ograniczona przez pierwioszek kwadratowy jej pola powierzchni pomnożony przez niewielką stałą.

Następnie, autor przedstawia dowód, że *jigsaw map* jest doskonałą wizualizacją w tym sensie, że spełnia wszystkie powyższe wymagania. Dowodzi również twierdzenie w drugą stronę: każdy podział przestrzeni, który spełnia te kryteria, jest podziałem opartym na pewnej krzywej Peano. Jednak, mimo teoretycznej „doskonałości” *jigsaw map*, można wskazać jej istotne wady. Po pierwsze, przestrzeń nie jest podzielona na figury wypukłe, co może utrudniać wzrokowe porównywanie ich wielkości. Po drugie, wizualizacja nie daje żadnych wskazówek dotyczących struktury drzewa, a na dodatek dodanie takiej informacji poprzez pogrubianie krawędzi lub efekty cieniowania wydaje się trudne (jedynie zastosowanie w tym celu kolorów nie stanowi problemu).

Kolejny pomysł to zrezygnowanie z wielokątów i wykorzystanie kół. Przykładowy program przedstawiający w ten sposób strukturę plików w systemie Linux jest dostępny na stronie [Wet]. Temat został szerzej zbadany w pracy [WWDW06]. Najważniejsza różnica pomiędzy tą metodą a tradycyjnym *treemap* polega na tym, że figury odpowiadające dzieciom nie stanowią podziału figury odpowiadającej rodzicowi — w przypadku kół zawsze muszą pozostać puste miejsca. Tak więc, mimo że koła są stosunkowo łatwe do wzrokowego porównywania pod względem wielkości, w tej metodzie nie ma sensu porównywać elementów niebędących bezpośrednim rodzeństwem — proporcje wielkości nie zostają zachowane. Z drugiej strony, dzięki pustym przestrzeniom i regularnym kształtom, znacznie łatwiej jest prześledzić budowę drzewa, zatem metoda kół może znaleźć zastosowanie właśnie tam, gdzie najważniejsza jest struktura, a rozmiary elementów mają znaczenie drugorzędne.

Dobrym przykładem tego jest program *CropCircles*, zaprezentowany w [WP06]. Jest to wyspecjalizowany program służący do wizualizacji ontologii zapisanych w języku OWL. W tym przypadku obiekty nie mają konkretnych atrybutów liczbowych, które należałyby przedstawić, dlatego promień koła odpowiadającego danemu elementowi zależy od liczby węzłów w jego poddrzewie. Zastosowano też inny algorytm rozkładu — wcześniejsze implementacje starały się wypełnić kołami jak największą powierzchnię wewnątrz koła-rodzica, ta zaś zostawia dużo wolnej przestrzeni, rozkładając elementy w równej odległości od krawędzi, w kolejności od największego. Dokładny sposób ułożenia zmienia się w zależności od względnych różnic w wielkościach elementów, tak aby dodatkowo ułatwić użytkownikowi szybką ocenę budowy struktury.

W następnej technice, przedstawionej w [vHvW03] i nazwanej *beamtree*, każdy węzeł jest reprezentowany w formie belki. Ułożenie belek jest ustalane przez algorytm oparty o *slice and dice*, metodzie zastosowanej w pierwszej implementacji *treemap* (zob. punkt 2.2.2). Podobnie jak tam, prostokąty były tworzone przez podział prostokąta-rodzica na równoległe części, w kolejnych poziomach na przemian poziomo i pionowo, tak tutaj beleki układane są na belce-rodzicu, na każdym poziomie prostopadle do poprzedniego. Kształt belek pochodzi od kształtu odpowiadającego mu prostokąta w *treemap*,



Rysunek 2.6: Przykładowe drzewo (po lewej) przedstawione za pomocą *treemap* (na środku) i *beamtree* (po prawej). Źródło: [vHvW03].

z tym że zostaje odpowiednio przeskalowany — belki na kolejnych poziomach muszą zostać zmniejszone, aby nie zasłaniały całkowicie belek leżących niżej, ale też nie mogą być zbyt krótkie, aby układ pozostał spójny. Jedynie liście drzewa nie są przedstawiane jako osobne belki, tylko jako wyróżnione kolorem fragmenty belek-rodziców (zob. rys. 2.6). Wizualizacja została wzbogacona o efekty wypukłości podkreślające kierunek ułożenia belek oraz cienie dodające głębi kolejnym poziomom. Zaimplementowano również trójwymiarową wersję wizualizacji. Eksperymentalne porównanie *beamtree* i *treemap* pokazało, że nowa metoda pozwala użytkownikom łatwiej zrozumieć strukturę drzewa i ustalić, na których poziomach znajdują się poszczególne elementy. Z drugiej strony, porównywanie wielkości elementów jest znacznie utrudnione — po pierwsze z powodu zmniejszania się skali na kolejnych poziomach hierarchii, a po drugie ze względu na podłużny kształt belek (proporcje elementów są zwykle dużo gorsze niż w metodzie *treemap*).

Ostatnia wizualizacja została ujęta w tym przeglądzie bardziej jako ciekawostka, niż ze względu na konkretne zalety. *Information cube*, zaprezentowana w [RG93] wykorzystuje trójwymiarową grafikę do przedstawienia hierarchii w formie zagnieżdżonych półprzezroczystych prostopadłościanów. Opisy elementów umieszczane są jako tekstury na ścianach brył. Podobnie jak w metodzie z kołami, wnętrze prostopadłościanu nie jest całkowicie wypełnione przez bryły dzieci, ponieważ byłoby to trudne do przedstawienia w czytelny sposób — zamiast tego elementy są rozmieszczone dość luźno. *Information cube* zostało zaprojektowane do pracy ze zwykłymi monitorami oraz wyświetlaczami 3D zakładanymi na głowę. Do kontrolowania pozycji kamery i wybierania elementów służy specjalna rękawica (*data glove*).

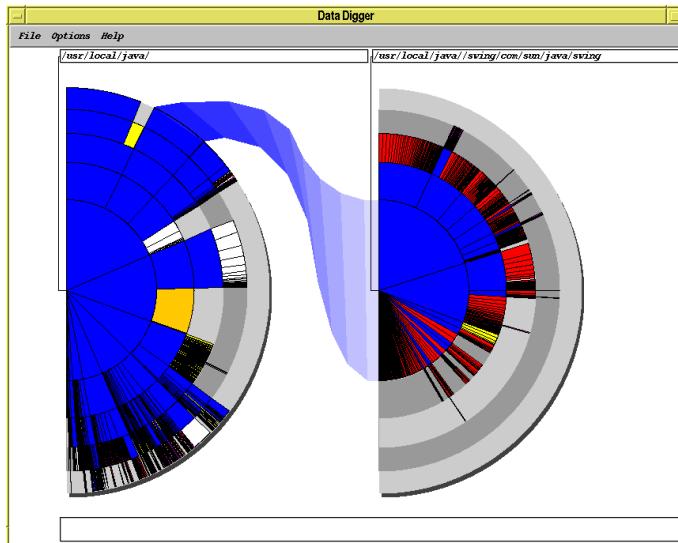
W artykule [WC99] dokonano doświadcjalnego porównania trzech metod wizualizacji struktur hierarchicznych opartych na grafice trójwymiarowej:

- *information cube* (implementację niewykorzystującą rękawicy),
- *cam tree* (metoda bardzo podobna do *cone tree*, zob. punkt 2.2.1), oraz
- *information landscape* (metoda przedstawiająca drzewo na dwuwymiarowej powierzchni).

Grupa użytkowników wykonywała zadania polegające na wyszukiwaniu elementów, ich zliczaniu i porównywaniu liczb dzieci. Metoda *information cube* wypadła zdecydowanie najgorzej, zarówno pod względem szybkości i poprawności wykonywania zadań, jak i subiektywnych ocen użytkowników. Rezultat ten można traktować jako przestrogi, aby przy poszukiwaniu nowych rozwiązań, które będą zaawansowane technologicznie i efektowne, nie zapomnieć o użyteczności.

2.2.4 Pozostałe metody

Warto wspomnieć o jeszcze innym podejściu do tematu wizualizacji hierarchii, który łączy zalety metody tradycyjnej (rysowanie wierzchołków na każdym poziomie) i bezpośrednie przedstawienie ich



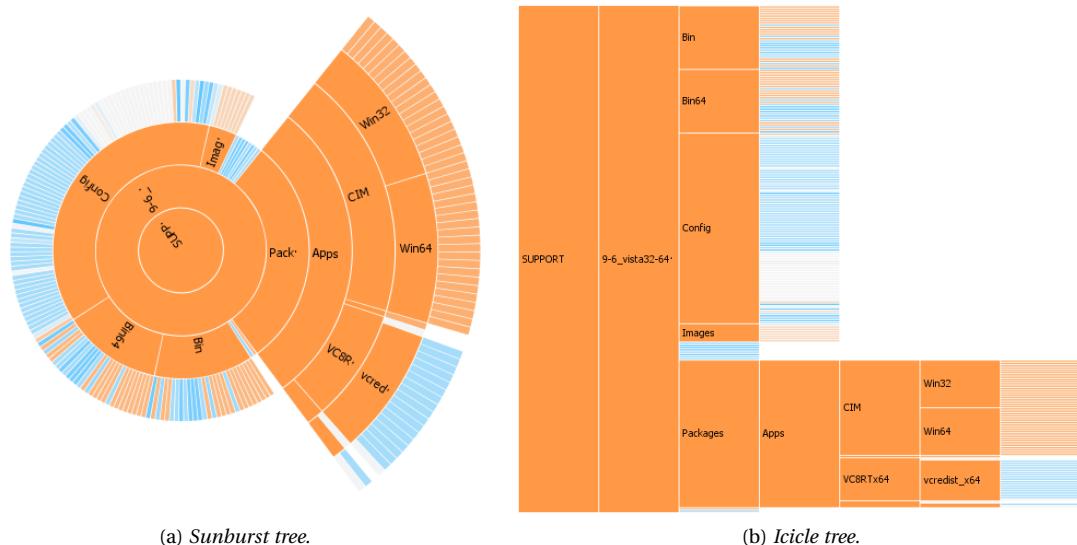
Rysunek 2.7: Przykład wizualizacji typu *infomation slices*. Źródło: [AH01].

struktury) i *treemap* (zakodowanie informacji w postaci wielkości elementu). W podejściu tym korzeniowi drzewa odpowiada koło w środku zajmowanej powierzchni, a kolejnym poziomom wierzchołków — coraz większe pierścienie otaczające całą strukturę. Na każdy wierzchołek przypada część pierścienia o wielkości kątowej proporcjonalnej do wizualizowanej „wielkości”, przy czym wierzchołki potomne muszą mieścić się w przedziale kątowym swojego przodka. Metoda ta, nazwana *sunburst tree* (zob. rys. 2.8a), dobrze oddaje ogólną strukturę elementów, jednak kiepsko gospodaruje dostępną powierzchnią, dlatego już dla niewielkiej liczby elementów pojawia się konieczność takiego ich zmniejszenia, że wizualizacja przestaje być czytelna.

Zaproponowano kilka modyfikacji mających na celu rozwiązywanie powyższego problemu. W pracy [AH01] przedstawiono program do wizualizacji systemu plików, w którym zamiast pełnego koła, widoczne są dwa półkola. W lewym półkolu wyświetlane są elementy drzewa w sposób analogiczny do podstawowej wizualizacji, ale tylko do określonej liczby poziomów (ustalanej przez użytkownika). Po wybraniu jednego z tych elementów przez użytkownika, wszystkie wychodzące z niego poddrzewa zostają przedstawione w prawym półkolu. Aby dodatkowo uwidocznić zależność pomiędzy półkolami, narysowana zostaje ścieżka od wybranego elementu w lewym półkolu do centrum drugiego półkola (zob. rys. 2.7). Ta metoda wizualizacji (nazwana *information slices*) częściowo rozwiązuje problem wyświetlania dużych drzew, jednak nie skaluje się — jeśli tylko drzewo będzie o kilka poziomów większe, problem znów się pojawi.

Inne podejście opisano w [SZ00]. Na początku wizualizacja przedstawia pełne drzewo *sunburst*, zajmujące całą dostępną przestrzeń. Gdy użytkownik wybierze interesujący go element, wyświetlony zostaje szczegółowy widok jego potomków, przy czym reszta drzewa pozostaje na ekranie w zmniejszonej formie. Zaproponowano trzy sposoby wyświetlania szczegółów:

- zmniejszenie pełnego drzewa i przesunięcie go do krawędzi ekranu (w kierunku przeciwnym do wybranego elementu) oraz wyświetlenie poddrzew wybranego elementu w centrum w formie powiększonych fragmentów pierścieni,
- zmniejszenie pełnego drzewa i wyświetlenie poddrzew wybranego elementu w formie pełnych pierścieni otaczających zmniejszone drzewo,
- powiększenie i zwężenie pierścieni pełnego drzewa oraz wyświetlenie poddrzew wybranego elementu w formie nowego *sunburst tree* w powiększonym centrum.



Rysunek 2.8: Wizualizacje oparte na polu powierzchni z uwzględnieniem wszystkich poziomów drzewa. Źródło: <http://www.randelshofer.ch/treeviz/>.

Użytkownik może później dowolnie wybierać kolejne elementy, zarówno z pełnego drzewa jak i z powiększonej części. Wszelkie zmiany widoku wprowadzane są jako płynna animacja.

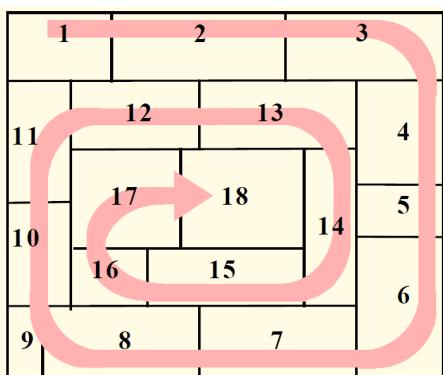
Podobne rozwiązanie zostało wykorzystane do wizualizacji wyników w wyszukiwarce Carrot² [OW]. Tam jednak wybrane elementy zostają powiększone w ramach podstawowego diagramu w taki sposób, że zajmuje prawie cały zakres 360 stopni, podczas gdy pozostałe elementy zostają zmniejszone lub całkowicie ukryte. To rozwiązanie jest prostsze w odbiorze, ponieważ użytkownik zawsze operuje na jednej strukturze. Dodatkowo każdy element drzewa można niezależnie przełączać pomiędzy trybem „normalnym” a „zminimalizowanym”, dzięki czemu, nie tracąc obrazu całości, można łatwo przejrzeć i porównać dowolnie wybrane fragmenty struktury.

Istnieje również uproszczona wersja metody *sunburst tree*, w której zrezygnowano z kołowego układu — poziomy drzewa nie są pierścieniami, tylko sąsiadującymi paskami (zob. rys. 2.8b). Otrzymany w ten sposób obraz może przypominać zwisający sople lodu, stąd w [KL83] metodę nazwano *icicle plot* (spotyka się też nazwę *icicle tree*). Prostota tego rozwiązania sprawia, że wykorzystanie dostępnej przestrzeni jest jeszcze gorsze niż w przypadku *sunburst tree*. Pojedynczy wierzchołek, odpowiadający korzeniowi drzewa, zajmuje tyle samo miejsca, co wszystkie liście razem wzięte.

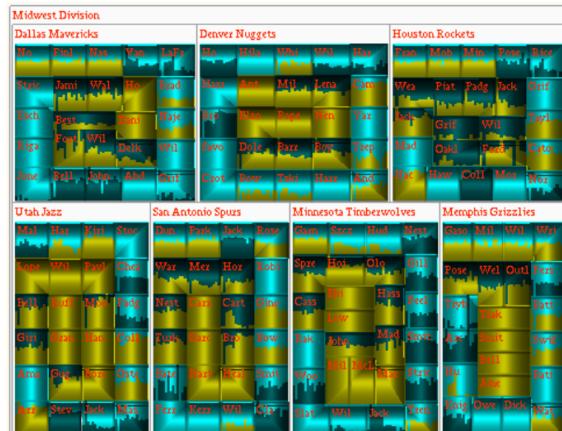
2.3 Wizualizacja struktur hierarchicznych z uwzględnieniem zmian w czasie

Pierwszą pracą związaną z metodą *treemap*, w której zwraca się uwagę na możliwość wprowadzania zmian w wizualizowanych danych w czasie rzeczywistym, jest [SW01]. Autor zauważa, że w przypadku dotychczasowych algorytmów ułożenia prostokątów, zmiany takie powodują zwykle gwałtowne przesunięcia poszczególnych elementów, co zakłóca odbiór prezentacji oraz uniemożliwia oszacowanie, które elementy zmieniły swoją wielkość i o ile. Proponuje więc nowe algorytmy ułożenia, nazwane *ordered treemap layouts*, które częściowo rozwiązują ten problem, a jednocześnie nie pogarszają znacząco proporcji boków prostokątów ani kolejności ich ułożenia. Rozwiązanie to nie jest jednak idealne — niepożądane efekty przeskóków, mimo że rzadziej, nadal występują.

Dalszy rozwój tego tematu nastąpił dzięki pracy Ying Tu i Han-Wei Shen przedstawionej w [TS07]. Pierwsza część pracy poświęcona jest nowemu algorytmowi ułożenia elementów — *spiral treemap*, w którym prostokąty układane są w kształt spirali obiegającej dostępną przestrzeń wzdłuż ścian, a na-



(a) Spiral treemap - algorytm rozłożenia elementów



(b) Wizualizacja wielu atrybutów - każdy element jest „wykresem”

Rysunek 2.9: Wybrane modyfikacje metody *treemap* zaproponowane przez Ying Tu i Han-Wei Shen. Źródło: http://www.vgtc.org/PDF/slides/2007/infovis/info_s08p01.pdf.

stępnie schodzącej w kierunku środka (zob. rys. 2.9a). Głównym celem przyświecającym projektowaniu algorytmu było ułatwienie użytkownikowi śledzenia zmian wielkości elementów. I rzeczywiście — wykazano, że w przypadku *spiral treemap* zakłócenia spowodowane zmianami wielkości są najmniejsze w porównaniu z innymi znanyymi algorytmami. *Spiral treemap* dobrze dobiera też proporcje prostokątów — dzięki temu wizualizacja jest czytelna, a porównanie wielkości elementów stosunkowo łatwe. Dodatkowo, charakterystyczny układ elementów sprawia, że łatwo je przejrzeć wzrokiem po kolej lub wyszukać dowolny element.

W drugiej części wspomnianej wyżej pracy zaproponowano kilka technik nazwanych wspólnie *contrast treemap*, które dodatkowo ułatwiają porównywanie dwóch struktur hierarchicznych, tym razem w sposób statyczny (informacje o obu grupach umieszczone na jednym obrazie).

Podział prostokąta na dwie części. W tej technice w każdym prostokącie lewy górny róg odpowiada stanowi elementu we wcześniejszym momencie, a prawy dolny róg — w późniejszym. Do zakodowania informacji można wykorzystać zarówno różnice w kolorach, jak i wielkości obu części. Jeśli różnica w wielkości nie musi być wyraźnie zaznaczona, przejście pomiędzy kolorami może być łagodne, aby nie zakłócać wyglądu obrazu silnymi kontrastami. Jeśli element zmienił położenie, informacja o tym jest zakodowana specjalnym kolorem w odpowiedniej części prostokąta (w takim przypadku jeden element może mieć przypisane dwa prostokąty odpowiadające dwóm pozycjom w hierarchii).

Znieksztalconie obrazu. Najpierw tradycyjny *treemap* stworzony dla pierwszego zbioru danych nakładany jest na powierzchnię jakiejś tekstyury. Następnie tworzony jest *treemap* dla drugiego zbioru, ale każdy prostokąt zostaje pokryty odpowiednio „rozciągniętą” częścią tekstyury odpowiadającą pierwszemu zbiorowi. Otrzymany w ten sposób obraz jest znieksztalconą formą pierwotnej tekstyury, w której zakłocenia zwracają uwagę użytkownika na różnice pomiędzy zbiorami danych.

Wizualizacja szybkości zmian. Wykorzystanie skali kolorów do wyrażenia stosunku wartości atrybutu danego elementu w pierwszym i drugim zbiorze danych. Do pokazania siły zmian można również zastosować wspomniany w części 2.2.2 efekt wypukłości — im większa zmiana, tym silniejszy efekt.

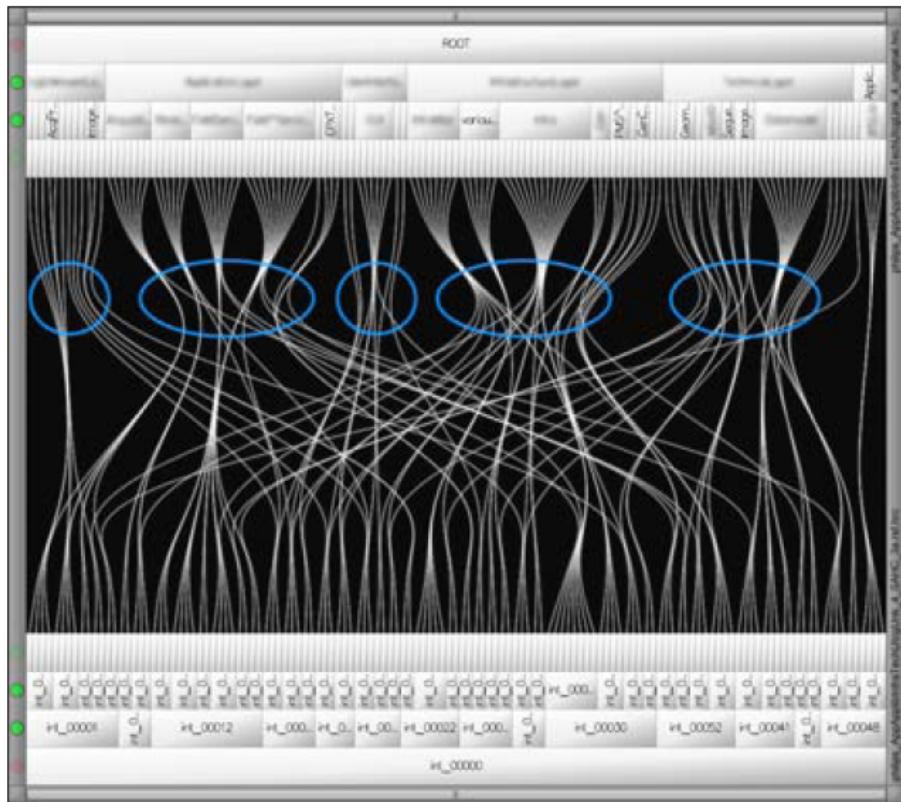
Wizualizacja wielu atrybutów. W tym podejściu każdy prostokąt staje się osobnym diagramem (wykresem słupkowym) przedstawiającym różnice wartości kilku atrybutów (zob. rys. 2.9b). Ze względu na małą ilość miejsca diagram taki jest bardzo uproszczony i nieopisany, a przez to raczej nieczytelny. Pozwala jedynie zauważać ogólne trendy, gdy na przykład dla danego elementu wartości wszystkich atrybutów w rozważanym przedziale czasu wzrosły.

Więcej prac koncentrujących się na metodzie *treemap* w kontekście wizualizacji zmian w czasie nie znaleziono. Istnieją za to przykłady wykorzystania w tym celu innych metod. Jednym z nich jest wizualizacja danych rynkowych przedstawiona w [TS08]. Opiera się ona na metodach *sunburst tree* i *icicle plot* (zob. podrozdział 2.2.4). Metody te zostały rozszerzone do wersji trójwymiarowej — elementy są przedstawiane nie jako płaskie figury, ale bryły, dzięki czemu dodatkowa informacja może zostać zakodowana w formie wysokości elementu. Zmiany w czasie przedstawiane są w formie animacji obejmującej wybrany przez użytkownika odstęp czasu. Animacja zakłada liniową zmianę wartości w czasie i obejmuje szerokość i wysokość elementów, a także ich kolor (o ile kolor wyraża wartość liczbową). Wizualizacja nie przewiduje zmian w strukturze hierarchii, ponieważ nie zdarzają się one w analizach, dla których program był projektowany (np. kategorie produktów są niezmienne). Jako pomocniczą formę wizualizacji zmian wartości wybranego przez użytkownika elementu zastosowano tradycyjne wykresy liniowe. Warto wspomnieć też o bogatych funkcjach ułatwiających analizę danych, takich jak ograniczenie wyświetlanych poziomów hierarchii, wyświetlenie tylko wybranego elementu i jego potomków, przedstawienie wybranego poziomu hierarchii w formie diagramu kołowego oraz wyróżnienie elementów przekraczających określona wysokość. Włączaniu i wyłączaniu tych funkcji również towarzyszy animacja, która ułatwia śledzenie zmian.

Kolejne ciekawe rozwiązanie zostało zaprezentowane w [HvW08] i służy do wizualizacji zmian w strukturze kodu źródłowego oprogramowania. W przeciwieństwie do poprzedniego przykładu, tutaj autorzy koncentrowali się na zmianach w samej strukturze hierarchii, a nie na wartościach atrybutów (wszystkie liście mają równą wielkość). Nie stosuje się też animacji, tylko pozwala porównać dwie hierarchie na statycznej grafice. Wizualizacja opiera się o metodę *icicle plot*, przy czym oba drzewa „wyrastają” z przeciwnych stron obrazu (zob. rys. 2.10). W centralnej części pomiędzy nimi przebiegają linie łączące liście, które odpowiadają sobie w obu strukturach. Aby linie te przecinały się jak najrzadziej, a łączone liście jak najczęściej występowały naprzeciw siebie, liście w obu drzewach (a pośrednio również węzły na wyższych poziomach) są odpowiednio sortowane. Połączenie pomiędzy parą liści w drzewach nie jest linią prostą, ale krzywą odzwierciedlającą kształt ścieżek idących od tych liści w kierunku korzenia. Dzięki temu linie odpowiadające liściom, które są blisko spokrewnione, zostają zgrupowane. Dodatkowo, jeśli linia jest symetryczna, oznacza to, że w obu drzewach liście „zawieszone” są w ten sam sposób, natomiast zaburzenia symetrii oznaczają zmiany strukturalne. Użytkownik może ustalić, ile poziomów drzewa będzie uwzględnione przez linie łączące, aby zobaczyć na których poziomach pojawiły się zmiany. Analizę zmian ułatwia mechanizm podświetlania odpowiadających sobie liści, które można włączyć przez klikanie w węzły lub zakreślanie wybranych linii łączących. Można również powiększyć wybrany zbiór elementów, ukrywając pozostałą część drzewa (operacja ta jest animowana).

2.4 Podsumowanie

Tematy wizualizacji danych zmieniających się w czasie oraz wizualizacji struktur hierarchicznych są rozpatrywane już od długiego czasu, dzięki czemu opracowano wiele nowatorskich metod prezentacji, a także modyfikowano już istniejące. Rzadko jednak można spotkać prace łączące te dwa tematy, choć wydaje się, że powinien to być ważny aspekt, dla którego znajdują się liczne zastosowania prak-



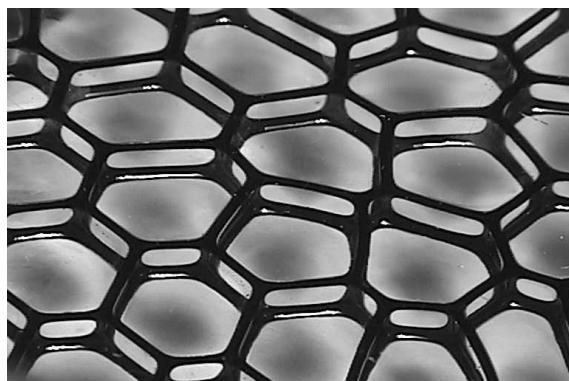
Rysunek 2.10: Wizualizacja zmian w strukturze kodu źródłowego. Źródło: [HvW08].

tyczne. Kilka interesujących propozycji pojawiło się w ostatnich latach, ale wciąż istnieją obszary praktycznie niezbadane. Jednym z takich obszarów, zgodnie z wiedzą autora, jest przedstawianie zmian w strukturze hierarchicznej przy pomocy animacji.

Rozdział 3

Foamtree — wizualizacja inspirowana bąbelkami w pianie

Koncepcja *foamtree* czerpie inspirację ze zjawiska występującego w przyrodzie — układaniu się bąbelków w pianie. Ograniczono się do przestrzeni dwuwymiarowej, ponieważ trójwymiarowe bąbelki byłyby trudniejsze w implementacji i wizualizacji. Pianę dwuwymiarową można sobie wyobrazić jako warstwę bąbelków umieszczoną w wąskiej szczelinie pomiędzy dwiema równoległymi taflami szkła (grubość szczeliny musi być znaczco mniejsza od długości średnic bąbelków) — zob. rys. 3.1. Taki układ szkieł jest często wykorzystywany do obserwacji zjawisk związanych z przepływem cieczy. Został on nazwany komorą Hele-Shaw, na cześć brytyjskiego badacza [Bat67].



Rysunek 3.1: Warstwa piany pomiędzy dwiema taflami szkła — odpowiednik piany dwuwymiarowej.
Źródło: <http://www.maths.tcd.ie/~foams/gallery.htm>.

Zachowanie piany zostało już dość dobrze zbadane i ujęte w teoretyczne modele. Obszerne omówienie tej wiedzy można znaleźć w książce [WH01]. Został tam opisany model, który pozwala na bardzo dokładną symulację piany, zarówno trój- jak i dwuwymiarowej. Postanowiono jednak zaprojektować własny, uproszczony model, który nie będzie zupełnie zgodny z rzeczywistą fizyką, ale będzie łatwiejszy w implementacji i mniej wymagający pod względem mocy obliczeniowej i zajętości pamięci.

W pierwszej kolejności przedstawione zostaną podstawowe właściwości stanu stabilnego piany, którymi kierowano się, projektując model. Prawa te zostały ujęte w XIX wieku przez belgijskiego fizyka Josepha Plateau. Trzy ostatnie reguły wynikają z pierwszej, co zostało formalnie udowodnione przez Jean Taylor [Tay76].

Ściany bąbelków mają minimalną powierzchnię. Ponieważ piana jest układem, w którym energia jest wprost proporcjonalna do pola powierzchni ścianek, bąbelki „dążą” do takiego ułożenia,

w którym ta powierzchnia jest najmniejsza (przy ograniczeniu wynikającym ze stałych objętości bąbelków). Nie zawsze zostaje osiągnięte globalnie najlepsze ułożenie — w zależności od stanu początkowego, piana może ustabilizować się w jednym z minimów lokalnych. W przypadku piany dwuwymiarowej, ścianki są rozpatrywane jako odcinki (a właściwie krzywe), przy czym minimalizowana jest suma ich długości, a ograniczeniem są stałe pola powierzchni bąbelków.

Każda ściana jest wycinkiem powierzchni kuli. Im mniejsza objętość bąbelka, tym większe ciśnienie gazu wewnętrz, stąd ścianki są wypukłe w kierunku większego bąbelka. Ściana oddzielająca dwa bąbelki o takim samym ciśnieniu jest płaska (co można traktować jako powierzchnię kuli o nieskończonym promieniu). W ujęciu dwuwymiarowym ściany są więc łukami.

Ściany bąbelków stykają się po trzy. W przestrzeni trójwymiarowej ścianki stykają się we wspólnej krawędzi, nazwanej krawędzią Plateau (ang. *Plateau border*). W wersji dwuwymiarowej jest to wspólny punkt. W przestrzeni trójwymiarowej ścianki stykają się też po cztery w jednym punkcie (jest to punkt styczności czterech krawędzi Plateau).

Kąt pomiędzy dowolnymi dwiema stykającymi się ścianami wynosi 120° . Tam, gdzie piana styka się z ciałem stałym (na przykład naczyniem), ściany są prostopadłe do jego powierzchni.

Uproszczenie w implementowanym modelu polega na przyjęciu założenia, że każda ściana jest odcinkiem. Skutkiem tego jest fakt, że jeśli bąbelki mają różne pola powierzchni, to ostatni warunek stanu stabilnego — dotyczący kątów pomiędzy ściankami — nie może zostać spełniony. Można jednak wykorzystać ten warunek, by poprawić jakość symulacji — poprzez dodanie siły, która „dąży” do utrzymania odpowiednich kątów.

Kształty bąbelków w dwuwymiarowej pianie do złudzenia przypominają figury uzyskane w *Voronoi treemap* (zob. rozdz. 2.2.3). Ciekawym zadaniem może być sprawdzenie, w jakim stopniu stan równowagi zaimplementowanego tam układu posiada wymienione powyżej właściwości. Kwestia ta wykracza jednak poza ramy niniejszej pracy. W założeniu, najważniejszym aspektem odróżniającym *foamtree* od *Voronoi treemap* ma być szybkość generowania wizualizacji.

3.1 Wizualizacja statycznej hierarchii

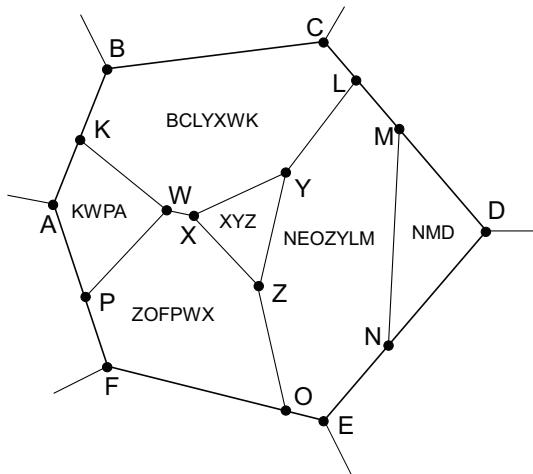
W pierwszym etapie prac skupiono się na wizualizacji pojedynczej struktury hierarchicznej, jeszcze bez uwzględniania zmian w czasie. Wygenerowanie wizualizacji rozpoczyna się od stworzenia początkowego układu bąbelków, w którym każdy bąbelek odpowiada jednemu węzlowi wizualizowanej struktury. Następnie wykonywany jest proces relaksacji, w którym oddziałujące siły doprowadzają układ do pewnego (lokalnego) stanu równowagi. Wszystkie te aspekty zostaną dokładniej omówione poniżej.

3.1.1 Ogólna struktura układu

Każdemu elementowi w wizualizowanej strukturze odpowiada jeden bąbelek w pianie. Przykładowy fragment układu został przedstawiony na rys. 3.2. Bąbelki są wielokątami, a informacja o ich kształcie jest przechowywana w postaci listy wierzchołków w kolejności zgodnej z ruchem wskazówek zegara. Bąbelki, które się ze sobą stykają, mają więc dwa wierzchołki wspólne, z tym że przechowują je w odwrotnej kolejności.

Hierarchia bąbelków polega na tym, że bąbelki potomne znajdują się wewnątrz rodzica i w całości wypełniają jego powierzchnię. Niektóre bąbelki, a zatem i wierzchołki, będą więc przylegały do ścianek bąbelka nadzakresnego. Wierzchołki takie są oznaczone jako leżące *na krawędzi*. Przechowują one

informację o tym, na której krawędzi leżą, ponieważ mogą poruszać się wyłącznie wzdłuż niej. Wierzchołki takie zawsze należą do dwóch bąbelków, w przeciwieństwie do pozostałych (mogących przemieszczać się w dowolnym kierunku) wierzchołków, które zawsze należą do trzech bąbelków (z wyjątkiem wierzchołków tworzących bąbelek-korzeń). Mowa tu o bąbelkach na tym samym poziomie hierarchii — biorąc pod uwagę wszystkie poziomy, można powiedzieć, że każdy wierzchołek należący do danego bąbelka, należy też do jednego z jego dzieci (o ile takowe istnieją), a także — rekurencyjnie — do jednego z dzieci tego dziecka i tak dalej.



Rysunek 3.2: Bąbelek podzielony na sześć mniejszych bąbelków. Punkty A–F należą do bąbelka nadrzędnego, punkty K–P leżą na jego krawędzi, a W–Z mogą poruszać się swobodnie.

3.1.2 Siły działające w układzie

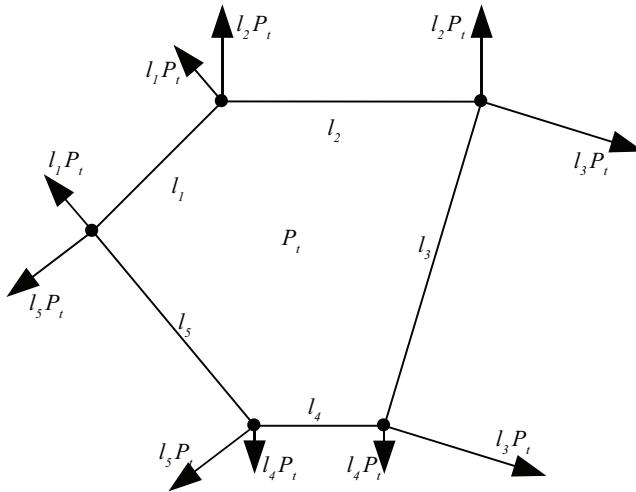
Relaksacja polega na iteracyjnym i dyskretnym przesuwaniu wierzchołków zgodnie z siłami oddziaływanymi w aktualnym stanie układu. Ważnym aspektem branym pod uwagę podczas projektowania mechanizmów sił było to, by oddziaływanie na wierzchołki było proporcjonalne do wielkości bąbelków, do których one należą. Dzięki temu, piana zachowuje się podobnie w różnych skalach wielkości (można umieścić wszystkie bąbelki w kwadracie o rozmiarze 1 na 1 lub 1000 na 1000 — nie powinno to wpływać na przebieg relaksacji). Istnieją następujące rodzaje sił.

Ciśnienie. Ta siła sprawia, że każdy bąbelek „rozpycha” swoje ścianki na zewnątrz. Nie jest obliczana dla bąbelka-korzenia. Ciśnienie wewnętrz bąbelka w momencie t jest wyznaczane w następujący sposób:

$$P_t = f_p m_t \frac{A^*}{A_t}, \quad (3.1)$$

gdzie:

- f_p jest stałą regulującą wpływ ciśnienia na relaksację — im większa ta wartość, tym szybciej zostanie osiągnięty stan równowagi, jednak zbyt duża wartość może spowodować zupełną utratę stabilności układu i drganie ścian,
- A^* jest stosunkiem oczekiwanej powierzchni bąbelka do oczekiwanej powierzchni jego rodzica,



Rysunek 3.3: Sposób oddziaływania ciśnienia na wierzchołki bąbelka.

- A_t jest stosunkiem faktycznej powierzchni bąbelka do sumy faktycznej powierzchni jego rodzeństwa (łącznie z nim) w momencie t ; obliczanie tego stosunku w odniesieniu do faktycznej powierzchni rodzica byłoby mniej dokładne, ponieważ ze względu na grubość ścianek (zob. część 3.1.7) powierzchnia rodzica jest większa niż suma powierzchni jego dzieci,
- m_t jest modyfikatorem ciśnienia zależnym od powierzchni bąbelka we wcześniejszych stanzach.

Wartość m_t w każdym kroku obliczana jest następująco:

$$m_t = m_{t-1} \frac{A^*}{A_t}. \quad (3.2)$$

Gdy powierzchnia bąbelka jest więc odpowiednia, modyfikator pozostaje stały. Gdy powierzchnia jest zbyt mała, wartość ułamka jest większa od 1 i modyfikator stopniowo się zwiększa, zwiększając wewnętrzne ciśnienie. Podobnie gdy powierzchnia jest zbyt duża, ciśnienie staje się coraz mniejsze. Modyfikator ten wprowadza bezwładność ruchu ścianek. Przykładowo można rozważyć bąblek, który przez dłuższy czas miał zbyt małą powierzchnię i jego modyfikator ciśnienia wzrósł do dużej wartości. Gdy w końcu bąblek osiągnie oczekiwanyą powierzchnię, modyfikator wciąż powoduje silne rozpychanie ścian, przez co powierzchnia dalej będzie się zwiększać, aż modyfikator nie ulegnie zmniejszeniu. Wtedy w analogiczny sposób może dojść do zbytniego zmniejszenia bąbelka. Przy dużych różnicach pomiędzy oczekiwanyimi a faktycznymi powierzchniami bąbelków stan równowagi może być osiągnięty dopiero po kilku oscylacjach. Aby zmniejszyć ten efekt, wprowadzono ograniczenie $0.5 \leq m_t \leq 2$.

Gdy znane jest ciśnienie, do każdej ściany bąbelka przykładana jest siła odpychająca. Wartość siły jest równa wartości P_t pomnożonej przez długość ściany. Siła ta zostaje dodana do obydwu wierzchołków na końcach ściany, prostopadle do niej, w stronę do zewnątrz (zob. rys. 3.3). Siła ta nie jest przykładana do ścian, które leżą na krawędzi bąbelka nadzawanego.

Napięcie powierzchniowe. Dzięki tej sile wszystkie ścianki działają jak sprężyny przyciągające do siebie wierzchołki końcowe. Siła przyciągania ma wartość równą stałej kontrolującej znaczenie napięcia, podzielonej przez długość ścianki. Siła ta zostaje dodana do obu punktów końcowych

w przeciwnych kierunkach wzdłuż ścianki. Ścianki, które leżą na krawędzi bąbelka nadzędnego, nie mają napięcia powierzchniowego — odpowiada to miejscu styczności piany z ciałem stałym (naczyniem).

Wyrównywanie kątów pomiędzy ścianami. Ta siła nie ma swojego odpowiednika w rzeczywistości.

Ciśnienie i napięcie powierzchniowe powinny wystarczać, żeby ściany układły się pod kątem 120° do siebie i 90° do powierzchni zewnętrznej. Jednak, ze względu na uproszczenie modelu, osiągnięcie stanu, w którym wartości kątów były choćby zbliżone do optymalnych wartości, zajmowało zbyt wiele iteracji, lub (często) wcale nie było możliwe. Dlatego postanowiono wprowadzić siłę, która będzie dodatkowo wymuszać odpowiednie kąty.

Siły związane z kątami są obliczane w każdym bąbelku, dla każdego kąta wewnętrznego (choć wierzchołki należące do bąbelka nadzędnego nie podlegają oddziaływaniu). Sposób działania sił dla przykładowego kąta przedstawia rys. 3.4. Najpierw wyznaczane są kierunki, w jakich powinny być położone ściany, aby kąt miał odpowiednią wartość (w tym przypadku 120° ; gdyby wierzchołek kąta leżał na krawędzi zewnętrznej, byłoby to 90°). Kierunki te są zaznaczone linią przerywaną. Następnie, dla każdego z ramion kąta wyznaczany jest punkt, w którym leżałby jego końcowy wierzchołek, gdyby ściana leżała w wyznaczonym kierunku (zob. rys. 3.4b — wierzchołek B powinien leżeć w punkcie B'). Siła popychająca końcowy wierzchołek w stronę tego punktu wynosi

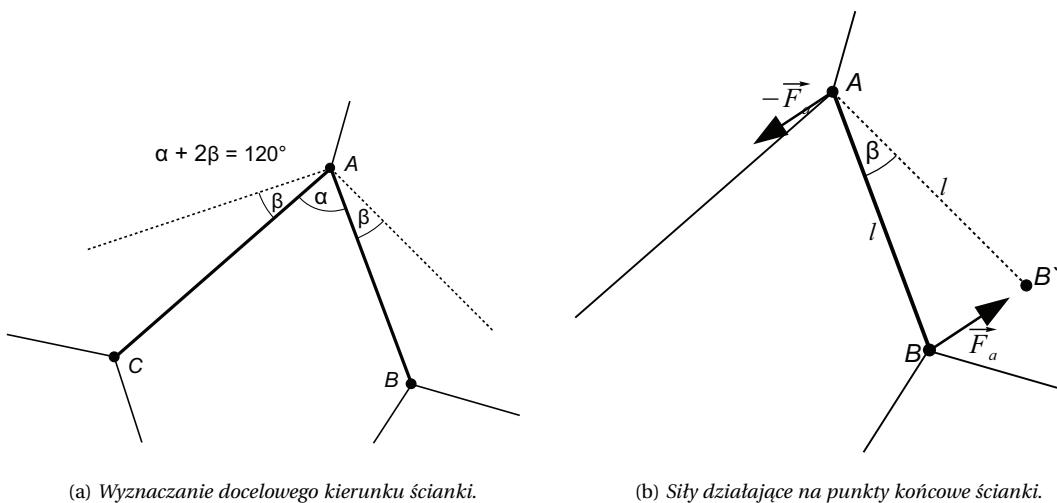
$$\vec{F}_a = f_a \vec{B} \vec{B}', \quad (3.3)$$

gdzie f_a jest stałą określającą, jak bardzo kąty mają być wyrównywane. Aby zachować wzajemność oddziaływań, taka sama siła z przeciwnym zwrotem zostaje przyłożona w wierzchołku A .

Początkowo, wartość siły we wzorze 3.3 była jeszcze dzielona przez długość ściany l . Wydawało się to dobrym rozwiązaniem, ponieważ było zgodne z fizycznymi własnościami momentu siły — im dłuższe ciało, do którego przykładany jest moment siły, tym dalej od punktu przyłożenia znajdują się jego końce, przez co siły działające w tych końcach są mniejsze. Poza tym, z obserwacji działania piany we wstępnych fazach implementacji wynikało, że krótkie ścianki znacznie częściej tworzą nieprawidłowe kąty z pozostałymi ściankami, więc ich silniejsza korekcja byłaby uzasadniona. Okazało się jednak, że powodowało to utratę dość istotnej własności całego modelu piany — niezależności od skali. Bąbelki powinny zachowywać się tak samo bez względu na to, czy ich wielkości liczone są w ułamkach, czy w milionach jednostek. Pierwsza postać powyższej formuły powodowała, że przy zbyt małym obszarze dostępnym dla piany (lub zbyt „gęstym” podziale na coraz mniejsze bąbelki) siła wyrównująca kąty działała zbyt gwałtownie i uniemożliwiała dojście układu do stanu równowagi.

Redukcja ścian pogarszających kształty bąbelków. Podczas pracy z pierwszymi wersjami modelu fizycznego zauważono, że często w minimach lokalnych znajdują się ścianki, których „redukcja” (rozumiana jako złączenie ze sobą jej dwóch końców i przeprowadzenie jednej z transformacji przedstawionych w rozdz. 3.1.3) powoduje znaczne polepszenie kształtów bąbelków. Polepszenie to oznacza nie tylko poprawienie subiektywnej oceny ich proporcji, ale wyraża się formalnie przez zmniejszenie łącznej długości ścianek w układzie. Ścianki, o których mowa, zazwyczaj są bardzo krótkie, a bąbelki z nich zbudowane są mocno rozcięgięte. Ponieważ zdefiniowane wcześniej siły nie radzą sobie z takimi ścianami, postanowiono zaimplementować specjalny mechanizm.

Trudno z góry określić, które ścianki warto zredukować, bez przeprowadzenia symulacji ich redukcji i sprawdzenia, czy nowe optimum lokalne będzie lepsze od pierwotnego. Można opracować różne przybliżone metody oparte na spostrzeżeniu, że ścianki takie są zwykle dużo krótsze



Rysunek 3.4: Sposób wyrównywania kątów pomiędzy ścianami.

od sąsiednich ścianek i leżą daleko od środków bąbelków. Szukając kompromisu pomiędzy dokładnością oceny i prostotą obliczeń, zdecydowano się sprawdzać stosunek długości ścianek, które się ze sobą stykają.

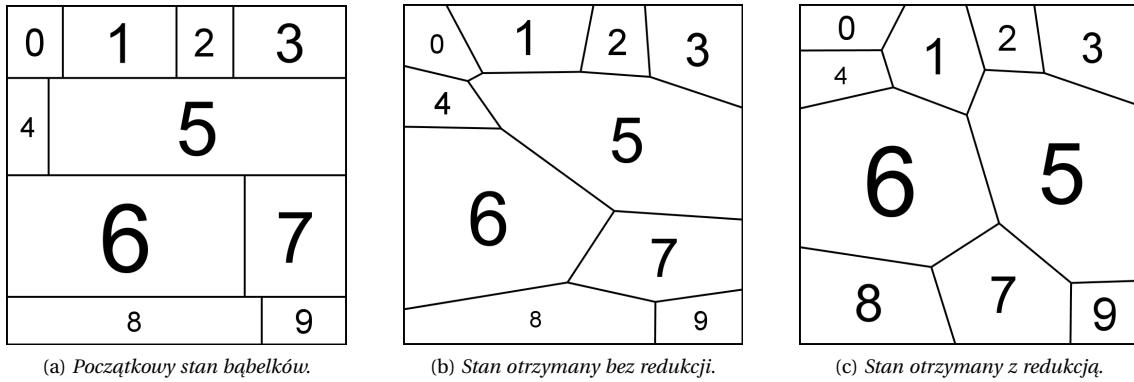
Analiza ścianek pod tym kątem odbywa się w każdym kroku relaksacji. W każdym bąbelku wyszukiwana jest ściana, która może być zredukowana (to znaczy, nie jest równocześnie ścianą bąbelka nadrzednego, ani nie jest rozpięta bezpośrednio pomiędzy dwiema ścianami bąbelka nadrzednego) i dla której stosunek długości do poprzedniej lub następnej ścianki jest najmniejszy. Jeśli stosunek ten jest mniejszy od pewnego parametru, ściana ta zostaje oznaczona jako potencjalny kandydat do zredukowania.

Redukcja nie następuje jednak od razu, ponieważ istnieje możliwość, że pozostałe siły w modelu oddziałują na tę ścianę na tyle mocno, że jej stan sam się wkrótce zmieni — zostanie zredukowana lub też dopasuje się do kształtów sąsiednich bąbelków i przestanie być dobrym kandydatem do redukcji. Dlatego ściana zostaje zapamiętana wraz z jej początkową długością. Jeśli w kolejnych iteracjach ta sama ściana stanie się kandydatem do redukcji określona liczbę razy, rozpoczyna się jej redukcja. Jeśli jednak przed tym zdarzeniem ściana wydłuży się o więcej niż określony procent, oznacza to, że jej stan jeszcze nie jest stabilny i przestaje być kandydatem. Ściany, które się skracają, pozostają kandydatami do redukcji pomimo braku stabilności, ponieważ wiadomo, że ich ewentualna redukcja będzie zgodna ze zmianami zachodzącymi w układzie.

Sama redukcja ściany polega na tym, że w każdym kroku relaksacji do obu jej końców zostaje przyłożona bardzo duża siła ściągająca je ku sobie. Dzięki temu, końce zbliżają się do siebie z maksymalną możliwą szybkością (ograniczenie szybkości zostało omówione poniżej). Jednocześnie inne siły, które mogą oddziaływać na te punkty, zostaną praktycznie zignorowane, ponieważ ich wpływ na wypadkowy kierunek ruchu będzie bardzo mały w porównaniu z „siłą redukującą”. W pewnym momencie punkty zetkną się i nastąpi wymiana ścianek, co kończy proces redukcji.

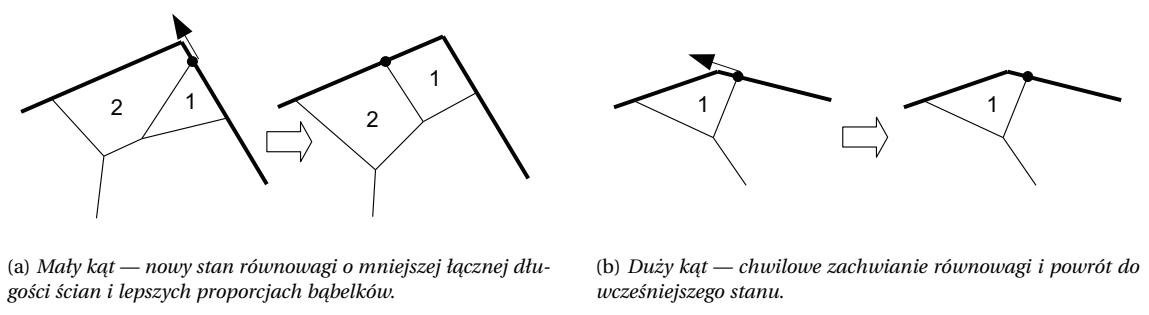
Podczas implementacji powyższego mechanizmu należało pamiętać, żeby zapobiegać redukcji ścianek, które należą do bąbelka mającego tylko trzy ścianki — doprowadziłoby to do powstania bąbelka o dwóch ścianach, który jest niepoprawny, ponieważ jego pole powierzchni wynosi zero.

Aby uniknąć pojawiania się wielu zmian obok siebie, zadbano również, aby w żadnym bąbelku nie dochodziło do redukcji więcej niż jednej ścianki w tym samym czasie. Kolejnym problemem, który należało rozwiązać, jest fakt, że często po wykonaniu redukcji na pewnej ściance, ta sama lub inną ściankę znów staje się dobrym kandydatem do redukcji — może to prowadzić do cyklicznego powtarzania tych samych redukcji. Dlatego każda zredukowana ściana zostaje zapamiętana wraz z jej początkową długością. Proces redukcji na danej ścianie może być powtórzony tylko wtedy, gdy jej aktualna długość jest mniejsza od jej długości przed ostatnią redukcją. Przykład tego, jak redukcja ścianek wpływa na układ bąbelków, można zobaczyć na rys. 3.5.



Rysunek 3.5: Wpływ redukcji ścianek na znajdowane minimum lokalne.

Po zaimplementowaniu powyższego mechanizmu okazało się, że dość często dochodzi do redukcji, które nie przynoszą korzyści i ich efekt zostaje szybko wycofany przez pozostałe siły w układzie. Sytuacje takie mają miejsce zwykle na styku bąbelka z krawędzią bąbelka nadzędnego, gdy wierzchołek leżący na krawędzi zostanie przesunięty w kierunku wierzchołka należącego do rodzica. To, czy taka operacja jest korzystna, zależy zazwyczaj od kąta pomiędzy ściankami stykającymi się w wierzchołku rodzica (zob. rys. 3.6). Wykrywanie takich sytuacji i powalanie na redukcję tylko wtedy, gdy ten kąt jest odpowiednio mały, pozwoliło uniknąć wielu bezsensownych prób redukcji, dzięki czemu średnia liczba kroków potrzebnych do zakończenia relaksacji znaczowo zmalała.



Rysunek 3.6: Prawdopodobne skutki redukcji ściany należącej do bąbelka nadzędnego w zależności od kąta przy wierzchołku należącym do rodzica.

Po wyznaczeniu wszystkich sił oddziałujących na wierzchołki, zostają one przesunięte zgodnie z siłą wypadkową. Maksymalny dystans, jaki może pokonać wierzchołek w pojedynczej iteracji został ograniczony, aby zmniejszyć prawdopodobieństwo sytuacji, w której siły skumulują się do dużych

wartości i wierzchołek zostanie wyrzucony na błędную pozycję. Ograniczenie to zależy od średniej przewidywanej średnicy bąbelka zawierającego dany wierzchołek. Jeśli jako p oznaczymy bąbelek będący rodzicem wszystkich bąbelków zawierających pewien wierzchołek, to maksymalny dystans tego wierzchołka wynosi

$$D_{\max} = d_{\max} \sqrt{\frac{A_p}{C_p}}, \quad (3.4)$$

gdzie d_{\max} jest parametrem, A_p jest polem powierzchni bąbelka p , a C_p jest liczbą dzieci bąbelka p .

W analogiczny sposób jest obliczany próg dystansu, poniżej którego wierzchołek zostaje uznany za nieruchomy (ustabilizowany). Oczywiście, parametr jest wtedy inny (jego wartość powinna być mniejsza niż w przypadku maksymalnego dystansu). Próg stabilizacji nie jest jednak porównywany jedynie z dystansem przebytym w aktualnej iteracji, ale ze średnią ważoną zmian położenia we wcześniejszych iteracjach, wyliczaną następująco:

$$\overline{D}_t = \sqrt{\overline{\Delta x_t}^2 + \overline{\Delta y_t}^2}. \quad (3.5)$$

Średnie przesunięcie wzdłuż osi X w iteracji t jest wyznaczane z poniższego wzoru (dla osi Y jest on analogiczny):

$$\overline{\Delta x}_t = (1 - w)\overline{\Delta x}_{t-1} + w \Delta x_t, \quad (3.6)$$

gdzie w jest współczynnikiem o wartości z przedziału $[0, 1]$ — im wyższa jego wartość, tym większe znaczenie ma dystans przebyty w ostatnim kroku w stosunku do wcześniejszych kroków. Powyższe podejście pozwala unikać oznaczania jako ustabilizowane wierzchołków, które są w ruchu, ale przypadkowo w jednej iteracji przebyły mały dystans. Z drugiej strony, wierzchołki, które praktycznie są nieruchome, ale na przykład oscylują blisko pewnego punktu, mogą zostać łatwiej uznane za stabilne. Gdy wszystkie wierzchołki w pianie są ustabilizowane, proces relaksacji zostaje zakończony. Informacja o tym, które wierzchołki są nieruchome, pozwala też na optymalizację procesu relaksacji, polegającą na nieprzeliczaniu stabilnych obszarów piany.

Ze względu na dyskretny charakter relaksacji, nie ma gwarancji, że uda się ustabilizować wszystkie wierzchołki w skończonej liczbie iteracji. Może się zdarzyć na przykład, że dwa lub więcej bąbelków będzie w nieskończoność „przepychało” pomiędzy sobą ściankę w tą i z powrotem, za każdym razem w takim samym stopniu odchylając ją od stanu równowagi. Aby rozwiązać ten problem, postanowiono stopniowo zmniejszać wpływ sił na przesunięcie wierzchołków, na które oddziałują. W każdej iteracji, przed przesunięciem wierzchołka zgodnie z wypadkową siłą, wypadkowa ta zostaje pomnożona przez współczynnik równy

$$q_t = 0.5^{\frac{t}{\lambda}} \quad (3.7)$$

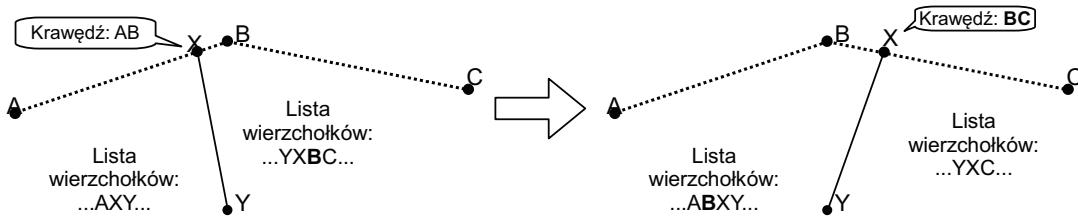
gdzie t jest numerem iteracji, a λ parametrem określającym, po ilu krokach przesunięcie bąbelków ma spaść o połowę. Wykładnicze osłabianie sił z czasem sprawia, że ewentualne wierzchołki krążące wokół stanu równowagi w nieskończonym cyklu, po pewnej liczbie iteracji zbliżą się do stanu równowagi na tyle, że cykl zostanie przerwany (a nawet gdyby tak się nie stało, w końcu siły zostaną stłumione na tyle, że nawet „aktywne” wierzchołki będą miały mniejszą prędkość niż próg stabilizacji).

Współczynnik q_t zastosowano również do stopniowego zmniejszania maksymalnego dystansu, jaki może pokonać wierzchołek w trakcie jednej iteracji. Jest to korzystne w sytuacji, gdy siły oddziałujące na wierzchołek przez długi czas przekraczają ustalony limit — wtedy efekt spowalniania może zadziałać od razu, a nie dopiero wtedy, gdy współczynnik q_t stanie się wystarczająco mały by stłumić wspomniane siły poniżej limitu. Zadbano jednak, by maksymalny dystans nigdy nie stał się mniejszy niż próg stabilności, ponieważ to by automatycznie doprowadziło do uznania wszystkich (nawet aktywnych) punktów za stabilne i zakończenia relaksacji.

3.1.3 Zmiany topologiczne

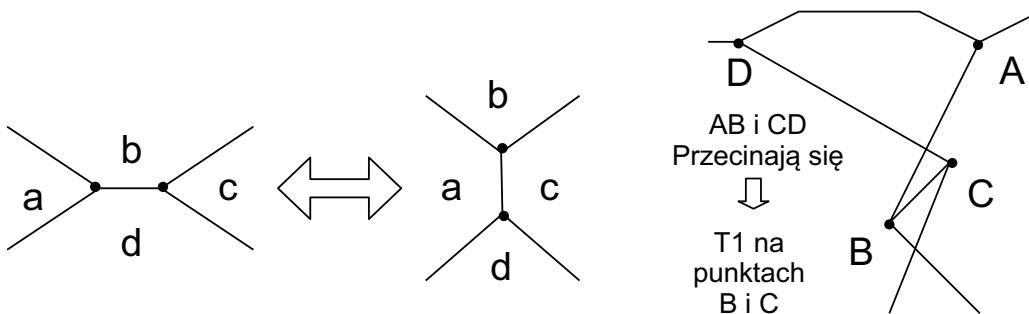
To, z których wierzchołków składają się poszczególne bąbelki, może się zmieniać w kolejnych krokach relaksacji. Ścianki mogą się ze sobą stykać na różne sposoby, dlatego niezbędne jest wykrywanie takich sytuacji i odpowiednie reagowanie.

Najprostsza zmiana zachodzi wtedy, gdy wierzchołek leżący na krawędzi zewnętrznej zostanie przesunięty do jej końca i musi „przeskoczyć” na następną krawędź. Wykrycie takiej sytuacji jest łatwe, ponieważ zakres ruchu podczas przesuwania wierzchołka jest ograniczony do prostej zawierającej krawędź. Wystarczy więc sprawdzić, czy wierzchołek pozostaje na odcinku pomiędzy dwoma końcami krawędzi. Operacja przeniesienia wierzchołka na sąsiednią krawędź została przedstawiona na rys. 3.7. Po pierwsze, krawędź pamięta w przenoszonym wierzchołku X zostaje zastąpiona przez nową. Zostają też zmodyfikowane oba bąbelki zawierające wierzchołek X. Z jednego z nich zostaje usunięty „przeskakiwany” wierzchołek (oznaczony jako B), a do drugiego dodany.



Rysunek 3.7: Przeniesienie wierzchołka z jednej krawędzi zewnętrznej na drugą.

Kolejna transformacja polega na tym, że w skutek zmiany połączeń pomiędzy wierzchołkami dwa bąbelki, które miały wspólną ściankę, zostają od siebie oddzielone, a inne dwa, które leżały daleko od siebie, stają się bąbelkami sąsiednimi (zob. rys. 3.8). W literaturze taka zmiana stanu piany została nazwana T1. T1 musi zajść wtedy i tylko wtedy, gdy przetną się dwie ścianki tego samego bąbelka połączone dokładnie jedną inną ścianką (zob. rys. 3.9). Sprawdzanie tego warunku sprowadza się więc do przejrzenia we wszystkich bąbelkach wszystkich trójk kolejnych ścian — jeśli pierwsza i trzecia z trójki ścianek się przecinają, to druga ściana musi zostać poddana transformacji.

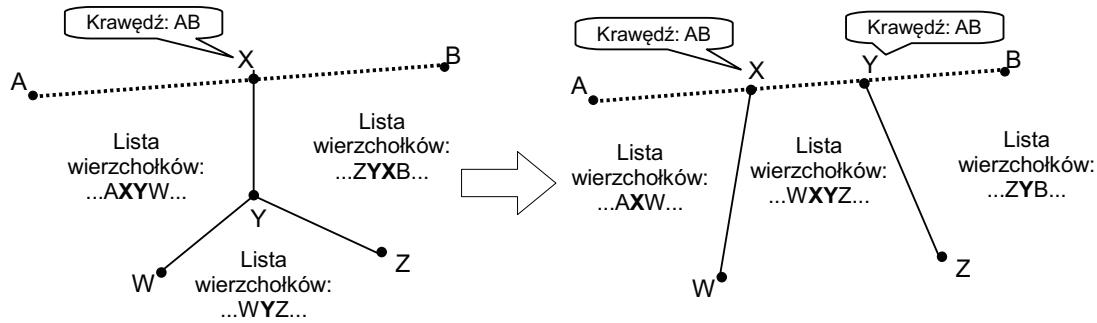


Rysunek 3.8: Transformacja T1.

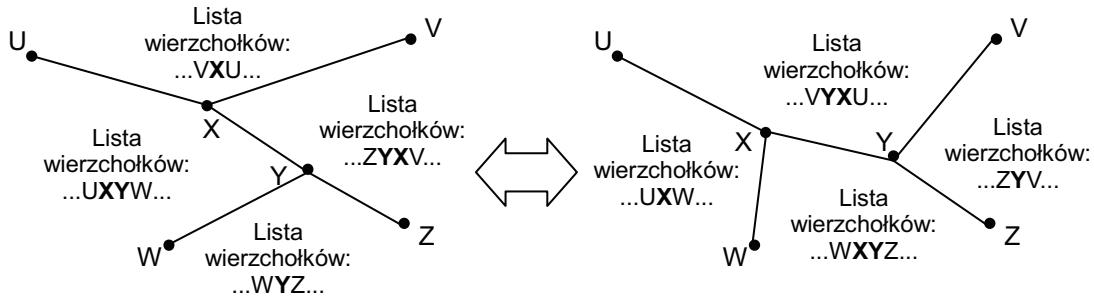
Rysunek 3.9: Warunek zajścia transformacji T1.

Przebieg transformacji T1 jest różny w zależności od tego, czy zmieniane wierzchołki leżą na krawędzi zewnętrznej. Sytuacja, gdy jeden wierzchołek leży na krawędzi, a drugi nie, została przedstawiona na rys. 3.10 (ze względu na czytelność, ścianki na rysunkach się nie przecinają — nie wpływa

to na charakter przedstawianych zmian strukturalnych). Po zmianie, oba wierzchołki znajdują się na krawędzi. Zatem, w sytuacji gdy na początku oba wierzchołki leżą na krawędzi (musi to być ta sama krawędź), należy wykonać operację odwrotną do tej przedstawionej na rys. 3.10. Trochę inne przekształcenia są potrzebne, gdy żaden z punktów nie leży na krawędzi — przedstawi to rys. 3.11. W tym przypadku przeniesione zostały ścianki połączone z wierzchołkami V i W. Poprawna byłaby też transformacja, w której sąsiadów zmieniają wierzchołki U i Z — wtedy wierzchołek Y należałby do lewego bąbelka, a X do prawego. Decyzję o tym, które połączenia zmienić, można oprzeć na odległościach wierzchołków X i Y od poszczególnych bąbelków, ale oba rozwiązań będą poprawne (w gorszym przypadku ułożenie będzie trochę dłużej korygowane przez siły w układzie).



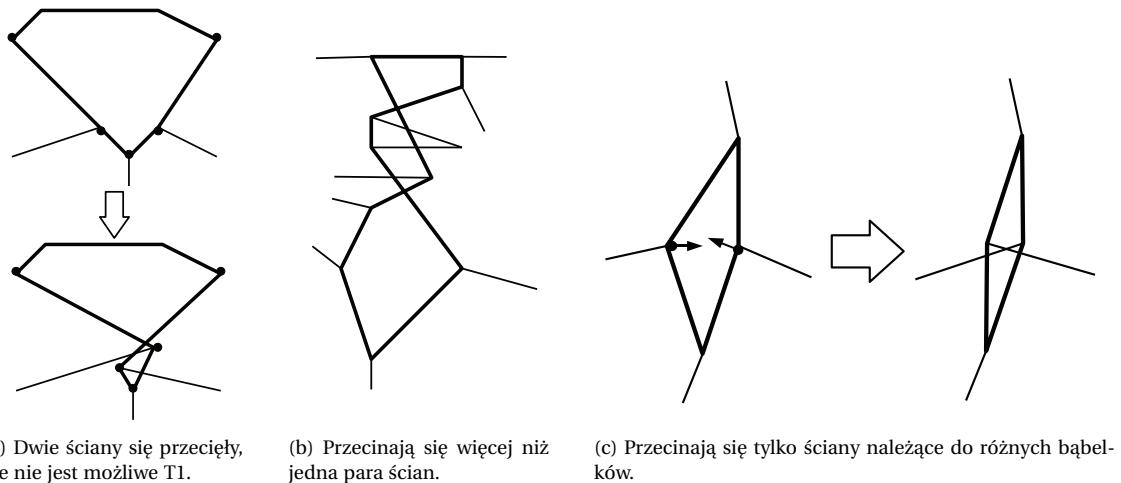
Rysunek 3.10: Zmiany wprowadzone przez T1 przy krawędzi zewnętrznej.



Rysunek 3.11: Zmiany wprowadzone przez T1 gdy modyfikowana ściana nie dotyczy krawędzi.

3.1.4 Naprawa niepoprawnego stanu bąbelków

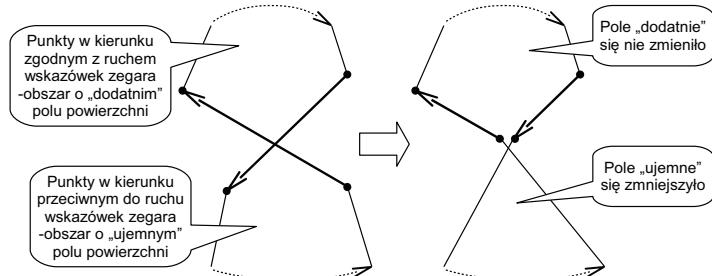
Ze względu na skomplikowany ruch wierzchołków, po każdym kroku relaksacji stan piany musi zostać sprawdzony i ewentualnie poprawiony. Przedstawione w poprzednim podrozdziale transformacje nie wystarczają, by naprawić wszystkie możliwe niepoprawne sytuacje. Może się na przykład zdarzyć, że ściany jednego bąbelka będą przecinać się tak, że nie uda się wykonać transformacji T1 (zob. rys. 3.12a i 3.12b). Jest też możliwe, że wierzchołki w bąbelku ułożą się w odwrotnej kolejności, przez co obliczenie pola powierzchni takiego bąbelka da ujemny wynik, mimo to w żadnym bąbelku ściany nie będą się przecinały (zob. rys. 3.12c).



Rysunek 3.12: Możliwe niepożądane stany bąbelków.

Wykrycie wszystkich takich sytuacji wymagałoby sprawdzenia wszystkich par ścian, czy się nie przecinają — złożoność obliczeniowa takiej operacji byłaby zbyt duża. Zauważono, że w praktyce wystarczy sprawdzić przecięcia wszystkich par ścian w ramach pojedynczych bąbelków (oraz osobno rozpatrzyć bąbelki o zerowym i ujemnym polu powierzchni, co zostanie omówione później). Koszt obliczeniowy sprawdzenia jednego bąbelka zależy wtedy od kwadratu liczby ścianek w tym bąbelku, a ponieważ średnio w bąbelku jest około sześciu ścian, można założyć koszt stały. Koszt sprawdzenia całej piany jest więc w przybliżeniu liniowy względem liczby bąbelków.

Po wykryciu przecięcia ścianek, następuje próba „naprawienia” go przy pomocy transformacji T1. Jeśli okaże się to niemożliwe (gdyż ścianki nie są odpowiednio połączone lub transformacja spowodowałaby powstanie niepoprawnego stanu, na przykład bąbelka o dwóch wierzchołkach), wierzchołki zostają przesunięte tak, aby ścianki przestały się przecinać. Przesunięcia tego można dokonać na cztery sposoby — w każdej ze ścianek można przenieść jeden z jej końców do punktu przecięcia (a właściwie trochę dalej — tak, aby ścianki nie miały żadnego punktu wspólnego). Należy wybrać ten sposób, który zmaksymalizuje końcowe pole powierzchni bąbelka. Co ciekawe, przyjmując, że wierzchołki ułożone są w kierunku zgodnym z ruchem wskazówek zegara, można łatwo stwierdzić, które przesunięcia spełniają ten warunek (zob. rys. 3.13).



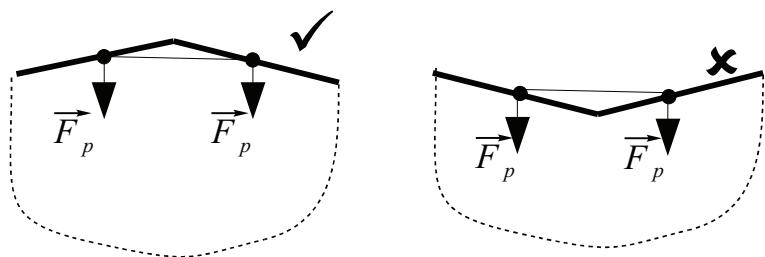
Rysunek 3.13: Sposób „naprawiania” przecinających się ścianek. Ścianki są przedstawione jako strzałki, których groty wskazują kolejność wierzchołków w bąbelku. W ścianie skierowanej bardziej na prawo przesunięty zostaje wierzchołek początkowy, a w ścianie skierowanej na lewo — końcowy. Powstałe nowe przecięcia będą naprawione w kolejnych próbach, aż do całkowitego wyeliminowania obszaru o „ujemnym” polu powierzchni.

Jest jeszcze jeden rodzaj sytuacji, które występowały stosunkowo często podczas testów i powo-

dowały błędy w programie: zamiana miejscami dwóch wierzchołków leżących na tej samej krawędzi. Zazwyczaj problem ten jest natychmiast wykrywany i zostaje wykonana transformacja T1, jednak nie zawsze jest to możliwe (jeśli na przykład oba wierzchołki należą do trójkątnego bąbelka, lub cały bąbelek znajduje się poza krawędzią zewnętrzną rodzica i odpowiednie ścianki nie będą się przecinały). Dlatego, dla każdej ściany leżącej na krawędzi, następuje sprawdzenie, czy jej punkty końcowe leżą na krawędzi w odpowiedniej kolejności. Jeśli nie, oba wierzchołki zostają przesunięte do tego samego punktu — pośrodku między ich dotychczasowymi pozycjami. Narzut czasowy tego sprawdzenia jest niewielki (sprawdzana jest stosunkowo mała liczba ścian, a test jednej ściany odbywa się w czasie stałym).

Jak już wcześniej wspomniano, bąbelki, w których obliczone pole powierzchni jest ujemne, wymagają osobnej metody naprawy, ponieważ ich ścianki nie muszą się ze sobą przecinać. Zauważono, że w takim przypadku zwykle przecinają się one ze ścianami „odchodzącymi” z wierzchołków bąbelka (to znaczy ścianami, które stykają się ze ścianami bąbelka, ale same do niego nie należą). Jest to widoczne na przykładzie z rys. 3.12c. Sposobem na przywrócenie poprawnego stanu jest w takiej sytuacji przesunięcie wierzchołka, z którego „odchodzi” przecięta ściana, na drugą stronę względem punktu przecięcia. Sprawdzenie, czy dowolna ściana bąbelka przecina dowolną ścianę „odchodzącą” ma złożoność kwadratową względem liczby ścian, podobnie jak wyszukiwanie przecięć wewnętrz bąbelka.

Specjalnej uwagi wymagają również bąbelki o zerowym polu powierzchni. Zazwyczaj ciśnienie w takim bąbelku powoduje powstanie bardzo dużych sił, które rozepchną ścianki do poprawnych rozmiarów, ale nie zawsze jest to możliwe. Jedna z takich wyjątkowych sytuacji polega na tym, że wszystkie wierzchołki bąbelka leżą w tym samym punkcie. Ponieważ ciśnienie nie działa na ścianki, które mają zerową długość, taki bąbelek mógłby pozostać „ściśnięty” w nieskończoność. Druga możliwość to trójkątny bąbelek leżący w narożniku bąbelka nadzewnętrznego, przy czym kąt tego narożnika jest wklesty (zob. rys. 3.14). W obu przypadkach poprawnym działaniem jest przesunięcie jednego z wierzchołków bąbelka do następnej ściany, aby doprowadzić do wykonania jednej z transformacji.



Rysunek 3.14: Standardowy mechanizm „rozpychania” bąbelka przez ciśnienie nie działa poprawnie, gdy bąbelek leży w narożniku bąbelka nadzewnętrznego przy kącie wkleśnym. Bąbelek zostanie „ściśnięty” do zerowego pola powierzchni i zablokowany w tym stanie (w sytuacji na rysunku bąbelek ma „ujemną” powierzchnię).

Powyzsze procedury są przybliżone i nie gwarantują usunięcia wszystkich błędów. Z tego względu, po wykryciu jednego błędu i naprawieniu go, sprawdzanie danego bąbelka rozpoczyna się od nowa (maksymalna liczba powtórzeń jest ograniczona przez parametr). Mimo to, mogą się zdarzyć niepoprawne stany, które nie zostaną wykryte, lub nie będą mogły być naprawione. Testy praktyczne wykazały jednak, że takie stany nie są stabilne i po kilku iteracjach układ wraca do stanu poprawnego lub takiego, który uda się naprawić. Mechanizmy zdefiniowane powyżej wystarczają więc do zachowania stabilności symulacji piany i poprawnego zakończenia relaksacji.

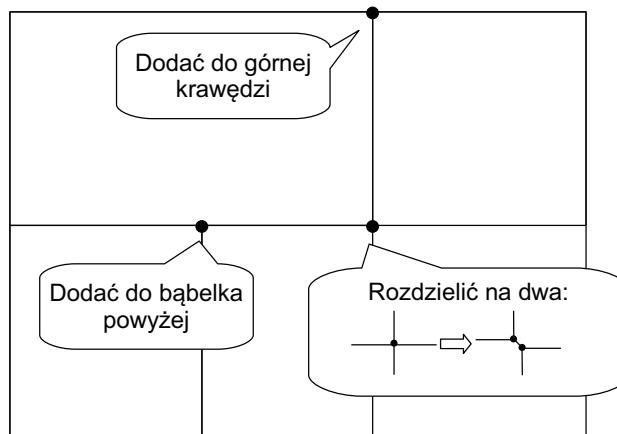
3.1.5 Generowanie stanu początkowego

Od stanu początkowego zależy, po jakim czasie i w jakim minimum lokalnym zakończy się proces relaksacji. Ważne jest, aby stan początkowy w dobrym stopniu spełniał ograniczenia nałożone na powierzchnie bąbelków. Pozwala to zakończyć relaksację w mniejszej liczbie iteracji, a także minimalizuje ryzyko sytuacji, w której duże siły ciśnienia spowodują utratę stabilności układu (czego konsekwencją mogą być drgania lub nawet wyrzucenie wierzchołka poza dostępny obszar). Korzystną cechą jest również determinizm — wizualizacja tego samego zestawu danych powinna zawsze dawać taki sam rezultat.

Zdecydowano się budować stan początkowy w oparciu o *treemap* (zob. rozdz. 2.2.2). Skorzystano z gotowej implementacji udostępnionej jako otwarte oprogramowanie przez autorów wielu algorytmów [WB01]. Dzięki temu podejęciu, można wypróbować różne algorytmy tworzenia *treemap* — są duże szanse na to, że końcowa wizualizacja częściowo zachowią cechy ułożenia początkowego (na przykład kolejność elementów).

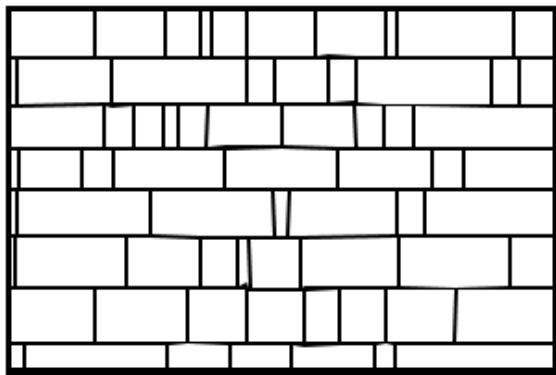
Procedura tworzenia bąbelków na podstawie wyniku działania algorytmu *treemap* jest stosunkowo prosta. Każdemu bąbelkowi zostaje przypisany pewien prostokąt. Wierzchołki prostokąta wyznaczają pozycje wierzchołków w bąbelku. Należy zadbać o to, aby różne bąbelki, które mają wierzchołek w tym samym miejscu (z pewną dokładnością — algorytm *treemap* nie musi zwracać tych samych współrzędnych), otrzymały tę samą instancję wierzchołka. Na końcu należy wprowadzić poprawki do struktury bąbelków (zob. rys. 3.15):

- wierzchołki, które należą do czterech bąbelków należą rozdzielić na dwa wierzchołki,
- wierzchołki, które należą do dwóch bąbelków, należy dodać do trzeciego bąbelka lub oznaczyć jako leżące na krawędzi zewnętrznej.

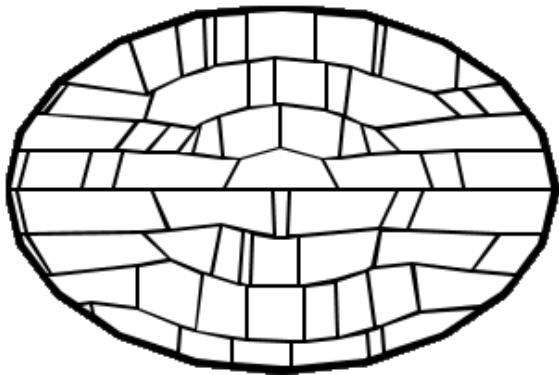


Rysunek 3.15: Rodzaje korekt niezbędnych przy generowaniu bąbelków na podstawie rozłożenia prostokątów w *treemap*.

Opisany powyżej sposób generowania stanu początkowego pozwala na stworzenie układu w kształcie prostokąta. Można jednak przekształcić ten układ tak, aby bąbelek-korzeń miał inny kształt (najlepiej by był to wielokąt wypukły). Jako przykład zaimplementowano moduł, który dopasowuje kształt wygenerowanej piani do wielokąta o wybranej liczbie wierzchołków, przybliżającego kształt elipsy. Poniżej opisano metodę zmiany kształtu bąbelka-korzenia, która powoduje stosunkowo niewielkie zniekształcenia jego bąbelków wewnętrznych. Metoda ta opiera się na przekształceniu współrzędnych z układu prostokątnego do układu *pseudo-biegunkowego*, opisanego dokładniej w podrozdziale 3.1.6.



(a) Bąbelek-korzeń w kształcie prostokąta.



(b) Bąbelek-korzeń w kształcie przybliżonej elipsy.

Rysunek 3.16: Przykład zmiany kształtu bąbelka wraz z jego bąbelkami potomnymi.

Aby zmienić kształt bąbelka, należy zapisać współrzędne wszystkich wierzchołków należących do bąbelków potomnych w układzie *pseudo-biegunkowym*, za *kontur zewnętrzny* przyjmując dla każdego wierzchołka kształt jego bąbelka nadzewnętrznego. Po zmianie kształtu bąbelka korzenia, nowe pozycje wierzchołków w bąbelkach wewnętrznych ustalane są poprzez przekształcenie zapisanych współrzędnych z powrotem do układu prostokątnego, jednak z *konturem zewnętrznym* zgodnym z nowym kształtem bąbelka nadzewnętrznego. Proces ten rekurencyjnie zmienia kształty bąbelków na kolejnych poziomach hierarchii. Na każdym etapie niezbędne jest ponowne przeliczenie informacji przechowywanych w bąbelkach leżących przy krawędzi zewnętrznej, ponieważ to, z którymi wierzchołkami bąbelka nadzewnętrznego się stykają oraz na których krawędziach leżą pozostałe wierzchołki, mogło się zmienić. Rysunek 3.16 przedstawia przykład zmiany kształtu bąbelków z układu prostokątnego do przybliżonej elipsy.

Początkowo wykorzystywano powyższą operację, aby zmieniać początkowy stan piany z prostokąta na inne kształty. Zauważono jednak, że zastosowanie tej metody może być znacznie szersze. Wcześniej, aby wygenerować bąbelki potomne w bąbelku-rodzicu przy pomocy algorytmów typu *treemap*, bąbelek-rodzic musiał mieć kształt prostokąta. Dlatego przy generowaniu stanu początkowego tworzone prostokątne bąbelki od razu na wszystkich poziomach, a następnie uruchamiano proces relaksacji, aby znaleźć optymalne kształty. Ograniczanie się do prostokątnego bąbelka-rodzica nie jest jednak konieczne. Można wykorzystać algorytm *treemap* do utworzenia dzieci w pewnym bąbelku prostokątnym (najlepiej o proporcjach boków zbliżonych do bąbelka-rodzica), a następnie dopasować jego kształt do kształtu docelowego bąbelka, co umożliwia łatwe przeniesienie utworzonych dzieci.

Zgodnie z powyższymi wnioskami, zmieniono metodę generowania stanu początkowego na metodę przyrostową. Najpierw tworzony jest bąbelek-korzeń oraz jego bezpośrednie dzieci i od razu uruchamiany jest proces relaksacji. Dopiero po jego zakończeniu, w każdym z dzieci tworzony jest następny poziom bąbelków potomnych i ponownie zostaje przeprowadzona relaksacja, ale tylko na najniższym poziomie — w każdym dziecku niezależnie. Proces ten jest rekurencyjnie powtarzany aż do utworzenia wszystkich bąbelków.

W porównaniu z pierwszym podejściem do generowania piany, nowa metoda jest znacznie korzystniejsza. Gdy proces relaksacji zostaje przeprowadzony na wszystkich poziomach jednocześnie, bąbelki na niższych poziomach nie mogą się ustabilizować, dopóki nie ustabilizuje się najwyższy poziom. Każda zmiana kształtów na wyższym poziomie sprawia, że na wszystkich niższych poziomach energetyczne minima lokalne zmieniają się i muszą być wyszukiwane od nowa. Tak więc, mimo że w procesie relaksacji obliczenia są od początku przeprowadzane na całej pianie, w rzeczywistości stabilizuje się tylko pierwszy poziom, a dopiero później następne, po kolei. Przyrostowe tworzenie bą-

belków i natychmiastowa relaksacja pozwala uniknąć tych niepotrzebnych obliczeń. Wniosek ten potwierdził się w praktyce — zaimplementowanie powyższego mechanizmu spowodowało, że dla przypadków testowych czas wygenerowania piany w stanie energetycznego minimum lokalnego zmniejszył się około trzykrotnie. Dodatkowo, w wielu zastosowaniach dobrym rozwiązaniem może być początkowe pokazanie użytkownikowi tylko pierwszego (lub kilku pierwszych) poziomu hierarchii i umożliwienie „rozwinięcia” wybranych bąbelków. W takim przypadku program może znacznie szybciej wygenerować i wyświetlić wstępную wizualizację, a następnie przeprowadzić relaksację pozostałych poziomów hierarchii w tle lub na żądanie użytkownika.

3.1.6 Pseudo-biegunowy układ współrzędnych

Pseudo-biegunowy układ współrzędnych pozwala zapisać położenie wierzchołka w odniesieniu do pewnego *konturu zewnętrznego*. Zazwyczaj konturem tym jest kształt bąbelka nadziednego. Wewnątrz tego konturu musi być wyznaczony *punkt centralny* — w zwykłych zastosowaniach jest to środek masy *konturu zewnętrznego*, jednak może być to też inny punkt. Podobnie jak w przypadku tradycyjnego układu biegunkowego, w układzie *pseudo-biegunkowym* położenie punktu P jest określone przez dwie współrzędne, oznaczające:

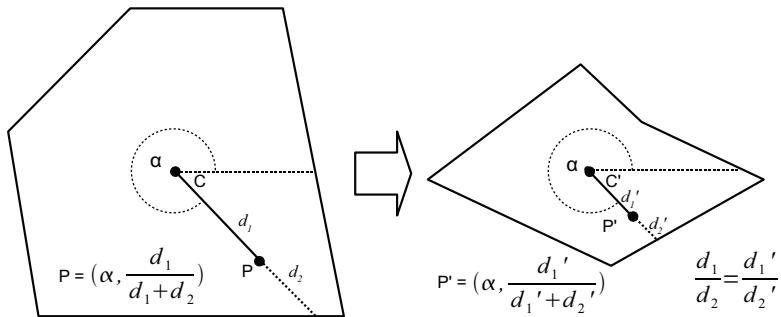
- kierunek prostej łączącej punkt P z *punktem centralnym* (a właściwie kąt skierowany pomiędzy tą prostą a osią poziomą),
- odległość punktu P od *punktu centralnego*. Odległość ta zostaje jednak przeskalowana w taki sposób, aby wartość 1 odpowiadała punktowi leżącemu na *konturze zewnętrzny*, w tym samym kierunku względem *punktu centralnego*, co punkt P .

Wyznaczenie współrzędnych *pseudo-biegunkowych* sprowadza się więc do wyszukania punktu przecięcia *konturu zewnętrznego*, prostą łączącą dany punkt z *punktem centralnym* i obliczenia odpowiednich odległości. W analogiczny sposób, na podstawie współrzędnych *pseudo-biegunkowych* można ustalić pozycję wierzchołka we współrzędnych prostokątnych. Obie te operacje mają złożoność liniową ze względu na liczbę punktów tworzących *kontur zewnętrzny*. Aby zachować jednoznaczność, *kontur zewnętrzny* powinien spełniać następujący warunek: dla każdej półprostej wychodzącej z *punktu centralnego* istnieje dokładnie jeden punkt przecięcia tej półprostej z *konturem zewnętrzny*. Warto zauważyć, że warunek ten spełniają wszystkie figury wypukłe, ale nie tylko. Na przykład, kształt popularnie nazywany gwiazdą może być poprawnym *konturem zewnętrzny*, ale nie jest figurą wypukłą — odcinek łączący punkty leżące na sąsiednich ramionach gwiazdy może przecinać jej krawędzie.

Pseudo-biegunkowy układ współrzędnych pozwala stworzyć odwzorowanie z jednego obszaru na drugi. Polega to na wyznaczeniu współrzędnych *pseudo-biegunkowych* danego punktu z wykorzystaniem konturu pierwszego obszaru i przekształceniu ich z powrotem do współrzędnych prostokątnych z wykorzystaniem konturu drugiego obszaru (zob. rys. 3.17). Odwzorowanie to nie gwarantuje zachowania kątów lub proporcji odległości, jednak pozwala na dość wierne przenoszenie kształtów oraz zmianę wielkości. Mechanizm ten jest wykorzystywany zarówno przy generowaniu piany do wizualizacji statycznej struktury (zob. rozdz. 3.1.5), jak i animacji (zob. rozdz. 5).

3.1.7 Uwzględnianie grubości ścianek bąbelków

Nieregularne kształty bąbelków często dostarczają więcej informacji o strukturze hierarchicznej niż prostokąty w metodzie *treemap*. Po przyjrzeniu się ściankom — ich długościom i sposobowi połączenia, można zazwyczaj wywnioskować, które bąbelki tworzą wspólną całość, a które leżą dalej od siebie



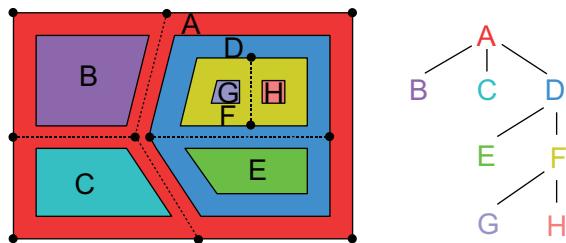
Rysunek 3.17: Znajdowanie współrzędnych w układzie *pseudo-biegunkowym* i odtwarzanie pozycji punktu w innym konturze.

w hierarchii. Na pierwszy rzut oka może być jednak trudno zauważyc te zależności. Aby zmniejszyć ten problem, postanowiono zastosować wskazówki wizualne: obramowanie i kolor.

Zastosowanie obramowania polega na tym, że kształt bąbelka nie jest bezpośrednio określony przez pozycje tworzących go wierzchołków. Zamiast tego, obliczane są pozycje punktów wewnętrznych, położone w pewnej odległości od krawędzi zewnętrznej. Punkty wewnętrzne bąbelków potomnych zależą też od obramowania przodków, więc grubość obramowania na kolejnych poziomach hierarchii kumuluje się. Zostały wyróżnione dwa rodzaje obramowania wewnętrz bąbelka, różniące się grubością (ich grubość można definiować niezależnie):

- *obramowanie bazowe* — dokładane do ścianek, które oddzielają bąbelek od innych bąbelków mających wspólnego rodzica. Grubość tego obramowania jest zazwyczaj mała i wpływa na grubość linii oddzielających bąbelki na najniższym poziomie hierarchii.
- *obramowanie zewnętrzne* — dokładane do ścianek, które leżą na ścianie bąbelka nadziednego. Im większa grubość tego obramowania, tym szybciej wzrasta grubość widocznych linii oddzielających bąbelki na coraz wyższych poziomach hierarchii.

Dzięki zastosowanemu podejściu, ścianki pomiędzy dwoma bąbelkami, które mają wspólnego rodzica, będą miały zawsze minimalną grubość, niezależnie od tego, jak głęboko w całej hierarchii się znajdują. Im dalsze „pokrewieństwo” pomiędzy bąbelkami, tym więcej obramowań będzie je oddzielało. Przykładowy układ uwzględniający obramowania bąbelków został przedstawiony na rys. 3.18. *Obramowanie bazowe* ma tam o połowę mniejszą grubość niż *obramowanie zewnętrzne*, ale ponieważ jest dodawane po obu stronach ścianek, wszystkie ramki wyglądają, jakby miały taką samą grubość.



Rysunek 3.18: Przykład działania obramowań w bąbelkach (po prawej: wizualizowana struktura drzewiasta).

Grubość obramowania jest brana pod uwagę nie tylko w trakcie rysowania bąbelków na ekranie, ale wpływa na cały model fizyczny. Po pierwsze, wierzchołki leżące na krawędzi bąbelka nadzied-

nego są przesuwane wzduż obramowania po wewnętrznej stronie. Dzięki temu, nawet gdy bąbelek jest bardzo głęboko w hierarchii i jest otoczony przez grube obramowania, siły oddziałujące na wierzchołki w jego wnętrzu działają tak samo jak siły na poziomie korzenia. Do obliczania ciśnienia w bąbelku w trakcie relaksacji również wykorzystywane jest wewnętrzne pole powierzchni (powierzchnia obramowania zostaje odjęta). Pozwala to na dokładniejsze wyrównanie proporcji w widocznych powierzchniach bąbelków i zmniejsza prawdopodobieństwo sytuacji, w której obramowanie zajmuje całą powierzchnię przeznaczoną na bąbelek. Takich sytuacji nie da się jednak całkowicie uniknąć. Dlatego, jeśli powierzchnia bąbelka staje się tak mała, że obramowanie zajmuje ją w całości, bąbelek zostaje oznaczony jako *stłoczony*.

Jeśli w pewnym bąbelku choć jeden bezpośredni potomek jest stłoczony, we wszystkich jego bąbelkach potomnych obramowanie zostaje tymczasowo wyłączone (przyjmowana jest zerowa grubość ramki). W ten sposób wszystkie bąbelki mające wspólnego rodzica są traktowane jednakowo i mają szansę się ustabilizować. Jeśli zostanie osiągnięty stan, w którym wszystkie bąbelki mają wystarczająco dużo miejsca na obramowanie, zostaje ono przywrócone. W przeciwnym razie, podczas wizualizacji piony należy podjąć decyzję, jak reagować na stłoczone bąbelki. Można je przedstawić bez obramowania lub też zrezygnować z ich wizualizacji i zamiast tego pokazać ich rodzica.

3.1.8 Wizualizacja bąbelków

W wizualizacji widoczne są wszystkie bąbelki odpowiadające liściom przedstawianej hierarchii. Można również włączyć rysowanie bąbelków na wszystkich poziomach, dzięki czemu można wyróżnić kolorami poszczególne obramowania. Kolory bąbelków mogą być zdefiniowane razem z innymi parametrami wizualizowanej struktury, ale jest też możliwość automatycznego przypisania kolorów na podstawie pozycji bąbelka w hierarchii i zdefiniowanego spektrum kolorów (odwzorowania liczb z przedziału $[0, 1]$ na kolory). Algorytm przypisuje korzeniowi pełen zakres spektrum, a każdemu z dzieci pewną jego część — wszystkim taką samą albo proporcjonalną do wielkości węzłów, w zależności od parametru. Rozdzielenie zakresu spektrum powtarza się rekurencyjnie we wszystkich wierzchołkach, aż do liści, którym zostaje przypisany kolor odpowiadający punktowi środkowemu otrzymanego zakresu.

Każdy węzeł ma przypisany pewien łańcuch tekstowy, który może być wyświetlony na bąbelku jako jego opis. Zaprojektowano algorytm, który stara się jak najlepiej dopasować wielkość czcionki opisu i sposób zawijania wierszy do różnych kształtów i wielkości bąbelków. Pierwszym krokiem algorytmu jest znalezienie prostokąta o największej powierzchni, który mieści się wewnątrz danego bąbelka. Dla uproszczenia założono, że środek tego prostokąta leży w środku ciężkości bąbelka. Poszukiwanie największego prostokąta polega na sprawdzeniu kilku różnych proporcji wysokości do szerokości w pewnym przedziale. Dla każdej proporcji, metodą binarnego przeszukiwania znajdowany największy prostokąt, który mieści się w bąbelku i spełnia tę proporcję. Po wybraniu największego prostokąta spośród wszystkich proporcji, następuje dopasowanie do niego wielkości czcionki, tak aby napis zmieścił się w prostokącie. Tu również zastosowano przeszukiwanie binarne. Do obliczania szerokości napisu na ekranie oraz określania, w których miejscach można złamać wiersze, wykorzystano mechanizmy dostępne w bibliotece standardowej.

Rozdział 4

Badanie własności i zbieżności modelu

Wykorzystanie przygotowanego modelu piany w praktycznych zastosowaniach wymaga poznania jego właściwości. Należy sprawdzić, jak układ zachowuje się w różnorakich warunkach: w zależności od dobranych wartości parametrów, charakteru wizualizowanej struktury danych oraz stanu początkowego. Najważniejszym aspektem wymagającym zbadania jest zbieżność procesu relaksacji do lokalnego minimum energetycznego oraz jak szybko zostaje on osiągnięty. Postanowiono zatem przeprowadzić serię eksperymentów.

Ze względu na dużą liczbę mierzonych wartości (zob. poniżej) oraz parametrów, które można modyfikować, nie jest możliwe systematyczne sprawdzenie zachowania modelu przy zmianie wielu parametrów jednocześnie w celu wykrycia ciekawych zależności między nimi — kombinacji jest zbyt wiele, szczególnie jeśli przyjmie się kilka różnych zestawów danych do wizualizacji. Postanowiono więc badać każdy parametr niezależnie, ustawiając pozostałe parametry na „standardowe” wartości (dobrane metodą prób i błędów w czasie implementacji badanego modelu).

Każdy badany parametr zostanie krótko opisany, zanim podane zostaną wyniki związań z nim eksperymentów i wnioski. Bardziej szczegółowe informacje na temat parametrów można znaleźć w rozdziale 3.1.2.

4.1 Badane parametry modelu

Poniżej omówione zostaną cechy modelu, które były mierzone podczas eksperymentów w każdym kroku relaksacji.

Błąd pola powierzchni. Wartość ta jest sumą wartości obliczonych dla każdego bąbelka według wzoru (por. wzór 3.1):

$$E_t = \left(1 - \frac{A^*}{A_t}\right)^2, \quad (4.1)$$

gdzie A^* jest stosunkiem oczekiwanej powierzchni bąbelka do oczekiwanej powierzchni jego rodzica, a A_t jest stosunkiem faktycznej powierzchni bąbelka do sumy faktycznej powierzchni jego rodzeństwa w chwili t . Wartość tej funkcji powinna dążyć do zera, ponieważ celem wizualizacji jest takie dobranie powierzchni bąbelków, aby ich proporcje były równe proporcjom oczekiwaniom. Ze względu na dyskretny charakter procesu mogą pojawić się pewne odchylenia od wartości zero, jednak duże wartości tej miary w stanie końcowym są niedopuszczalne i oznaczają, że niektóre wielkości w modelu nie zostały poprawnie odwzorowane.

W początkowej fazie relaksacji, gdy ruchy wierzchołków są gwałtowne, wartość błędu może osiągać dość duże wartości (w dużej mierze zależy to od jakości stanu początkowego). Szczególnie gdy jeden lub więcej bąbelków jest chwilowo „ścisnięty” na zerowej powierzchni — w tym przy-

padku do obliczeń za pole powierzchni przyjmowana jest stała bliska零, przez co wynikowy błąd rośnie do astronomicznych wielkości. Jednocześnie w końcowych fazach błąd zazwyczaj jest bardzo bliski零. Aby wygodnie porównywać wartości błędów na wykresie całego procesu relaksacji, zdecydowano się przedstawiać je na skali logarytmicznej oraz dodatkowo ograniczać maksymalną wielkość (żadna wartość na wykresie nie przekroczy 10^6).

Suma długości ścian. Ta wartość jest bardzo ważna, ponieważ założono, że mała suma długości ścian w pionie jest wyznacznikiem jakości końcowej wizualizacji pod względem ogólnego kształtu i proporcji elementów. Można więc przyjąć, że im mniejsza wartość tej cechy w stanie końcowym, tym lepiej (o ile błąd pola powierzchni pozostanie odpowiednio małe).

Przebyte odległości. Jest to suma odległości euklidesowej pomiędzy pozycją wierzchołka w danej iteracji a jego pozycją w poprzedniej iteracji, liczona po wszystkich wierzchołkach. Cechą ta pozwala ocenić gwałtowność zmian w pionie i dąży do zera w końcowych stanach.

Liczba niestabilnych punktów. Liczba wierzchołków, które poruszają się na tyle szybko, że nie mogą być uznane za stabilne. Ta cecha pozwala ocenić, jak szybko grupy wierzchołków się stabilizują w energetycznym minimum lokalnym. Jeśli większość punktów stabilizuje się stosunkowo szybko, to nawet w przypadku dużej liczby iteracji można to uznać za korzystną sytuację, ponieważ relaksacja tylko pojedynczych grup wierzchołków w końcowych fazach pozwala na znaczne przyspieszenie obliczeń.

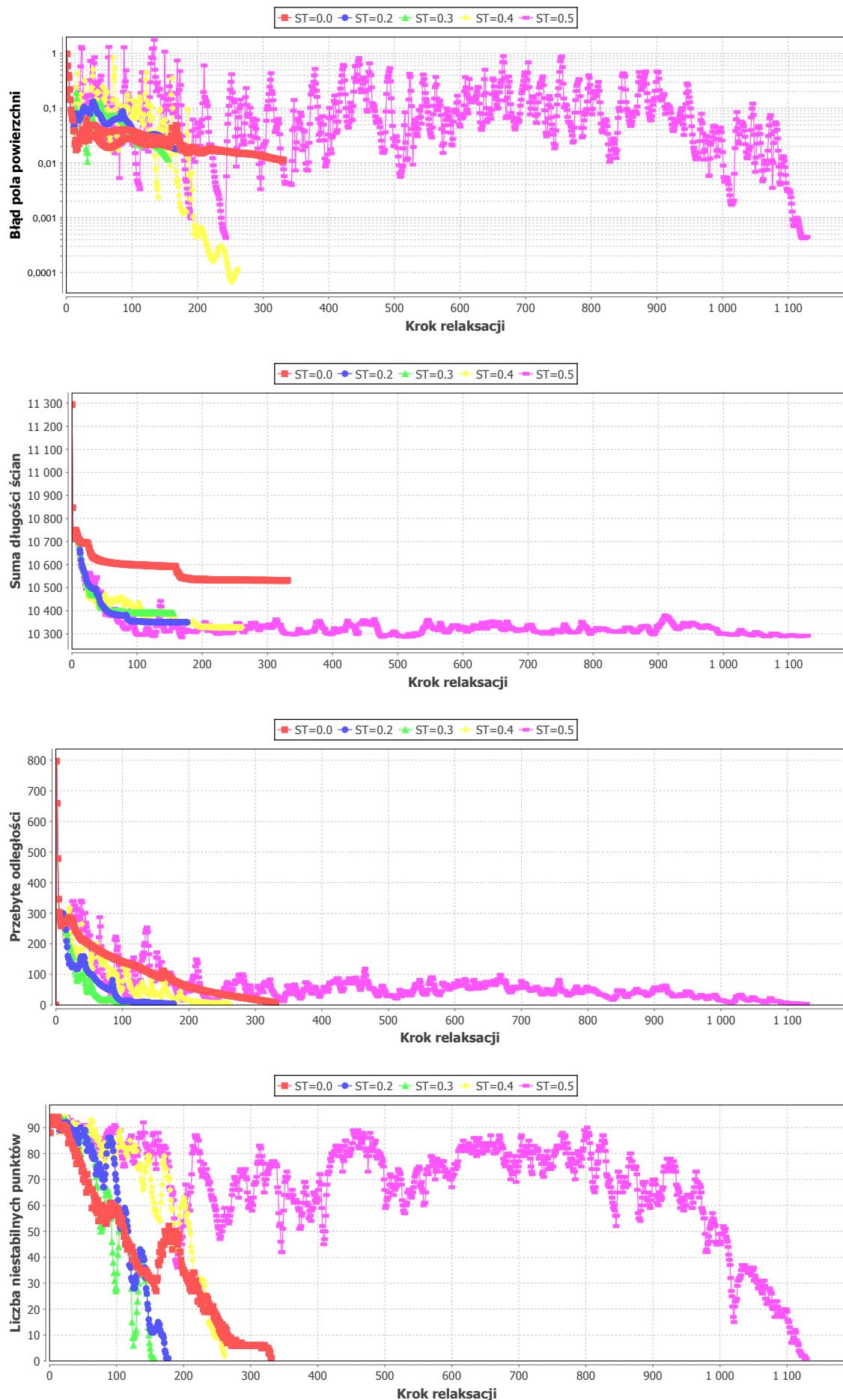
4.2 Konfiguracja mechanizmu redukcji ścian

Pierwsze eksperymenty będą dotyczyć najbardziej nieprzewidywalnego elementu modelu — mechanizmu redukcji ścian. Aby efekty redukcji pojedynczych ścian był widoczne na wykresach badanych własności, liczba bąbelków musi być stosunkowo mała. Stworzono losowe struktury danych zawierające po 48 liści, o wielkościach będących liczbami całkowitymi z przedziału [1, 10]. W jednej z nich wszystkie liście należą bezpośrednio do korzenia, w drugiej hierarchia obejmuje trzy poziomy (liczba potomków w elemencie zależy od poziomu, na którym się znajduje: korzeń ma czterech potomków, każdy z nich znów po czterech, a na ostatnim poziomie — po trzech). W przypadku hierarchii wielopoziomowej cała struktura bąbelków jest generowana przed rozpoczęciem relaksacji, ponieważ przy zastosowaniu metody przyrostowej taki test byłby równoważny wielokrotnemu wykonaniu niezależnych prób na małych, jednopoziomowych zbiorach.

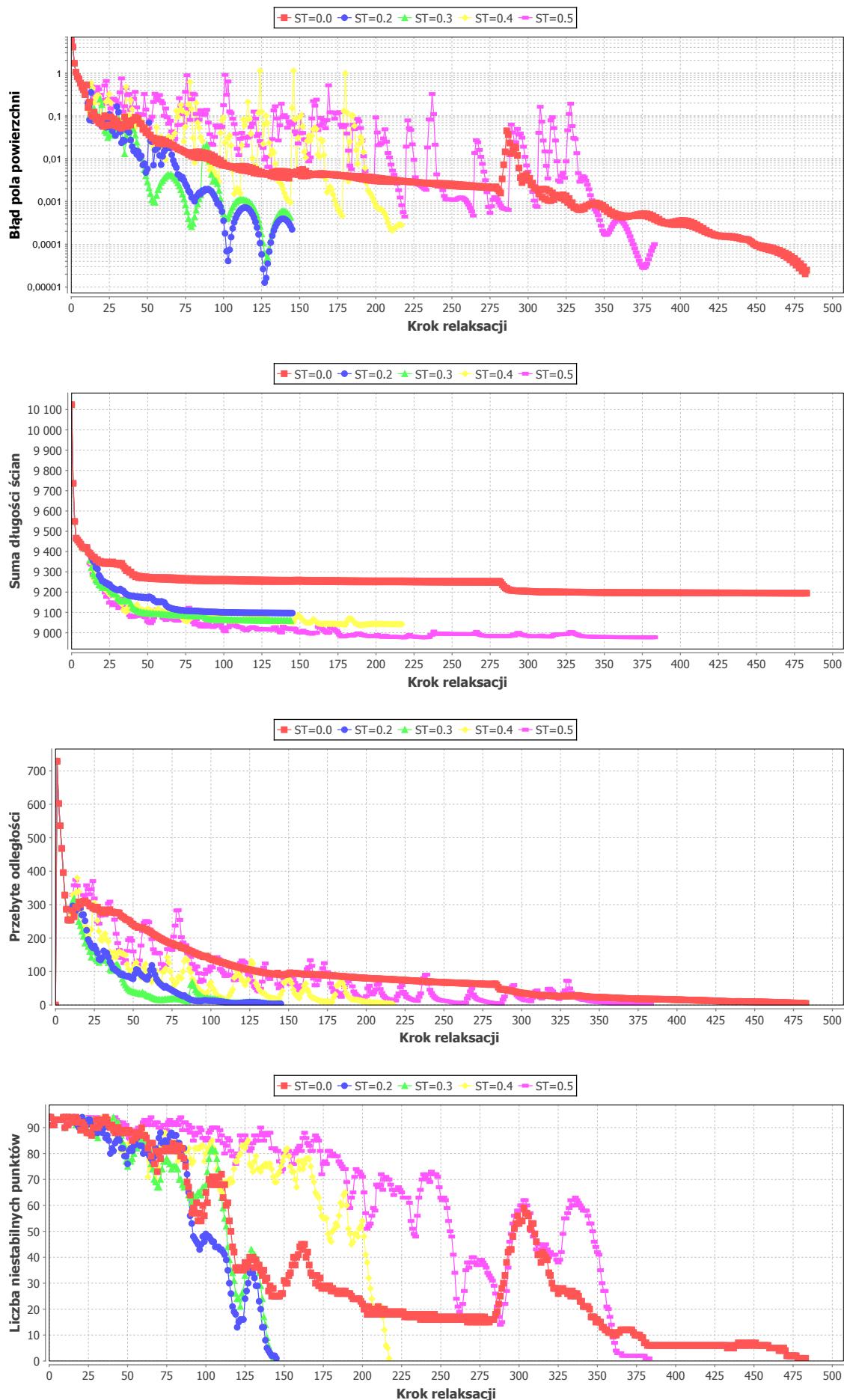
4.2.1 Squeeze threshold

Pierwszy badany parametr to *squeeze threshold* (ST). Jest to minimalny tolerowany stosunek długości dwóch sąsiadujących ścianek w bąbelku. Gdy dwie ścianki przekroczą ten próg, krótsza z nich zostanie w miarę możliwości zredukowana. Domyślną wartością jest 0,33. Postanowiono sprawdzić zachowanie modelu dla wartości 0 (która oznacza wyłączenie mechanizmu redukcji) oraz od 0,2 do 0,5 z krokiem co 0,1. Dla struktury o jednym poziomie hierarchii przetestowano dwa stany początkowe, z bąbelkiem-korzeniem w kształcie kwadratu i koła. Obliczenia dla drugiego zestawu danych przeprowadzono tylko dla kwadratowego ułożenia. Wyniki zostały przedstawione na rysunkach 4.1, 4.2 i 4.3.

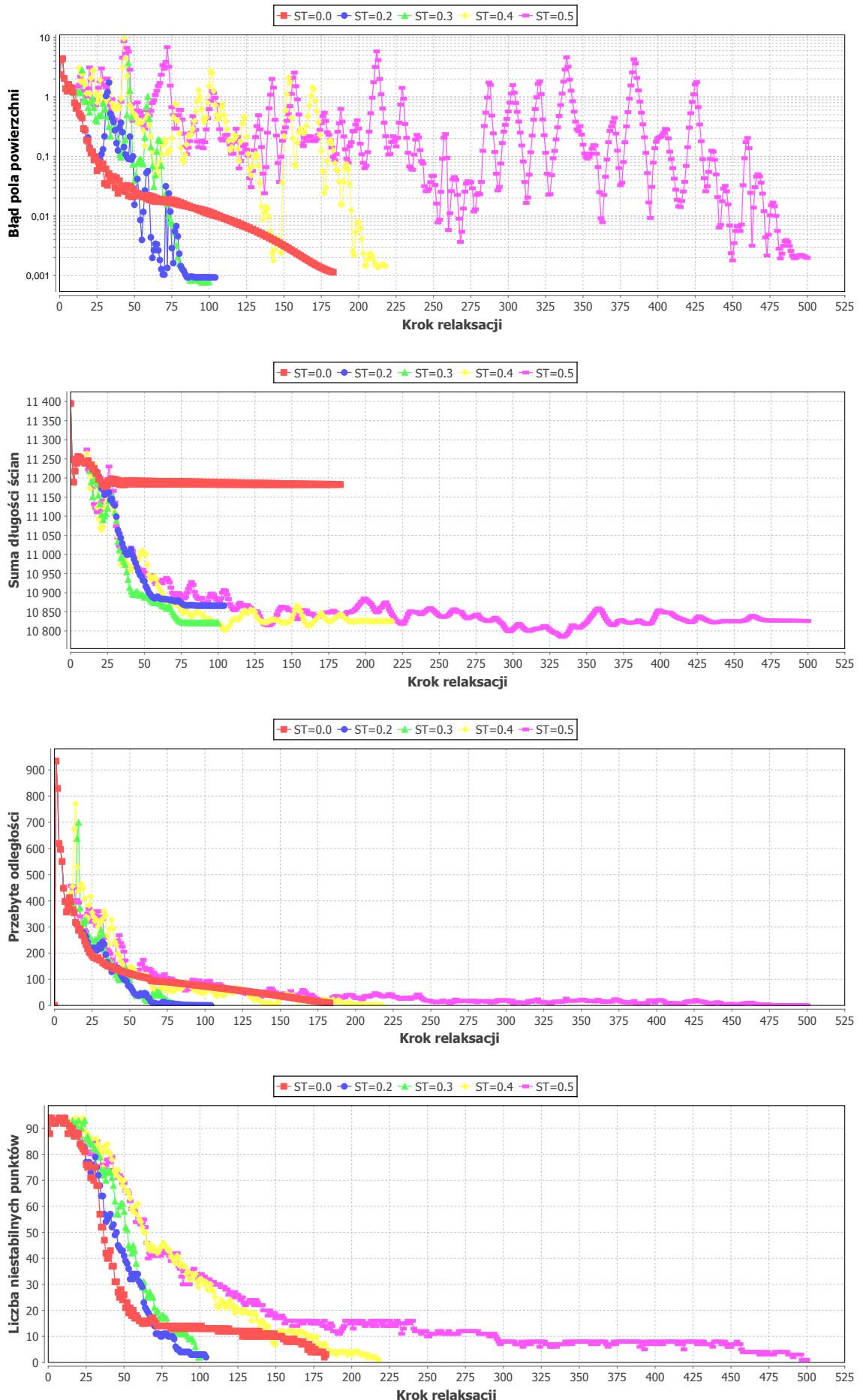
Charakterystycznym zjawiskiem widocznym na niemal wszystkich wykresach jest to, że im większa wartość parametru *squeeze threshold*, tym bardziej „rozchowane” są linie na wykresach. Linie przedstawiające zachowanie modelu przy parametrze równym zero są w dużych przedziałach monotoniczne i stosunkowo rzadko zmieniają swój kierunek, podczas gdy dla coraz większych wartości pojawiają się



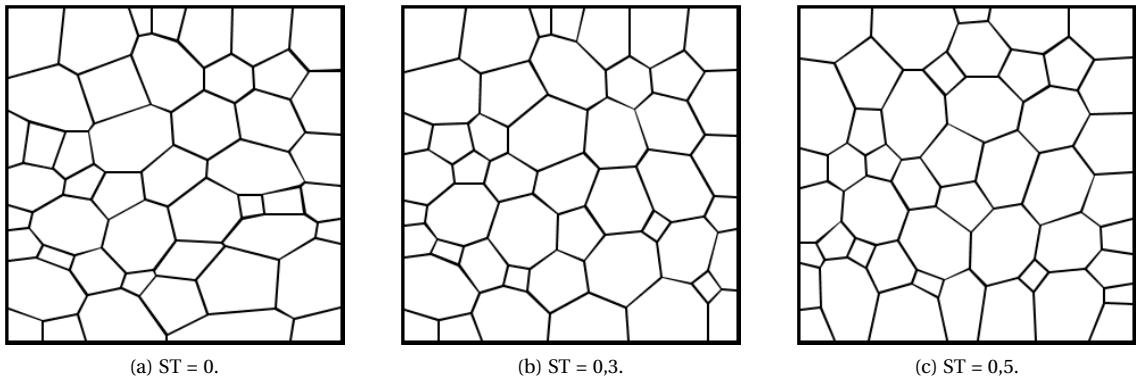
Rysunek 4.1: Wpływ parametru *squeeze threshold* na relaksację dla struktury jednopoziomowej o 48 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.2: Wpływ parametru *squeeze threshold* na relaksację dla struktury jednopoziomowej o 48 liściach, dla bąbelka-korzenia w kształcie koła.



Rysunek 4.3: Wpływ parametru *squeeze threshold* na relaksację dla struktury trójpoziomowej o 48 уровнях, для бабелька-корzenia в квадрате.



Rysunek 4.4: Wygląd stanu końcowego piany dla struktury jednopoziomowej o 48 liściach dla różnych wartości parametru ST.

coraz częstsze i większe odchylenia. Szczególnie zwracają uwagę charakterystyczne „pagórki” na wykresach odległości przebytych przez wierzchołki. Każdy taki nagły, chwilowy wzrost wartości tej cechy jest prawdopodobnie związany z rozpoczęciem redukcji pewnej ściany. Na wykresie przedstawiającym błędy pola powierzchni zazwyczaj w tym samym momencie również pojawia się gwałtowny, chwilowy wzrost, ponieważ bąbelki sąsiadujące z redukowaną ścianą zostają znieształcone i ich powierzchnie muszą zostać na nowo wyrównane. Podobnie zachowuje się wykres opisujący sumę długości ścian — w czasie redukcji wartość tej cechy zazwyczaj wzrasta, choć już nie tak wyraźnie. Najczęściej bardziej widoczne jest obniżenie długości ścian tuż po zakończeniu redukcji — oznacza to, że redukcja pozwoliła osiągnąć lepszy stan piany. Nie widać jedynie synchronizacji wału liczby niestabilnych wierzchołków z redukcjami ścian — te aspekty nie są ze sobą bezpośrednio związane, ponieważ redukowane są tylko ściany, których wierzchołki nie są jeszcze ustabilizowane.

Brak widocznej zależności pomiędzy wartością błędu w stanie końcowym a wartością parametru *squeeze threshold*. Najważniejsze jednak, że zawsze udaje się osiągnąć stan o akceptowalnie małym błędzie (poniżej 0,1).

We wszystkich analizowanych przypadkach wyłączenie redukcji ścian powoduje wyraźne zwiększenie długości ścianek w stanie końcowym. Można zauważyc, że im większy współczynnik *squeeze threshold*, tym lepszy stan końcowy udaje się uzyskać, choć występują wyjątki od tej reguły. Rysunek 4.4 pozwala porównać wzrokowo końcowe efekty otrzymywane dla różnych wartości parametru ST. O ile dla wartości 0 i 0,3 różnice w wyglądzie piany są dość istotne, o tyle porównanie 0,3 i 0,5 jest już trudniejsze — poprawa wyglądu jest raczej dyskusyjna i dotyczy tylko kilku bąbelków.

Można by przypuszczać, że zwiększenie parametru ST powinno powodować zwiększenie liczby kroków potrzebnych do zakończenia relaksacji. Większa liczba redukcji ścian oznacza więcej odchylen od stanu równowagi, który musi zostać przywrócony. Okazuje się jednak, że zależność nie jest tak prosta. Redukcja z parametrem ST równym 0,2 i 0,3 skutkuje dwu- lub nawet trzykrotnym skróceniem czasu relaksacji w porównaniu z wyłączoną redukcją. Największa różnica jest widoczna w przypadku testu z bąbelkiem-korzeniem w kształcie koła — można więc przypuszczać, że przy takim kształcie utworzony stan początkowy jest mniej korzystny niż w przypadku kwadratu, przez co ustabilizowanie piany bez mechanizmu redukcji ścian staje się znacznie trudniejsze. Może to jednak być przypadek, stąd potrzeba przeprowadzenia dodatkowych testów.

Dalsze zwiększanie wartości parametru powyżej 0,3 powoduje znaczne zwiększenie liczby kroków, zgodnie z przewidywaniami. Dla największej z badanych wartości (0,5), relaksacja trwa kilkukrotnie dłużej niż w najkrótszych eksperymentach, a końcowy rezultat jest niewiele lepszy lub czasem nawet gorszy. Oznacza to, że zbyt często dochodzi do redukcji, które nie poprawiają stanu piany. Współczyn-

nik o wartości 0,3 wydaje się dawać stosunkowo najlepsze rezultaty — we wszystkich testach pozwolił znaleźć stan końcowy porównywalny z wynikami uzyskanymi przy większych wartościach ST, a dla struktury trój-poziomowej był nawet najlepszy.

Aby dokładniej zbadać zależność liczby kroków potrzebnych do relaksacji od wartości parametru ST, postanowiono przeprowadzić kolejne testy, tym razem na znacznie większych zestawach danych (w ten sposób zjawiska przypadkowe powinny mieć mniejszy wpływ na ogólny stan modelu). Stworzono hierarchię o czterech poziomach, w której liczba potomków w każdym elemencie na kolejnych poziomach kształtuje się następująco: 8, 7, 6, 5 (łącznie powstało więc 1680 liści). Podobnie jak wcześniej, przetestowano układ w kształcie kwadratu oraz koła. Wyniki zostały przedstawione na rysunkach 4.5 i 4.6.

Zgodnie z przewidywaniami, linie na wykresach są znacznie bardziej wygładzone w porównaniu z pierwszymi testami. Duże wahania widać jedynie na wykresie błędów powierzchni, ale i tak są one wyraźnie mniejsze. Warto zwrócić uwagę na fakt, że w początkowych fazach wartość błędu często przekraczała ustalony pułap, ponieważ dla tak dużej liczby bąbelków często zdarza się „ścisnięcie” kilku z nich do zerowej powierzchni. Błąd w końcowym stanie wydaje się dostatecznie mały jak na tak dużą liczbę bąbelków, choć przy wyłączonej redukcji można zaobserwować jego kilkukrotny wzrost.

Potwierdziło się spostrzeżenie, że długość ścianek w stanie końcowym maleje wraz ze wzrostem parametru ST, oraz że nie warto zwiększać go powyżej 0,3, ponieważ różnice stają się pomijalnie małe w stosunku do drastycznego wzrostu czasu obliczeń.

Tym razem wzrost liczby kroków potrzebnych do zakończenia relaksacji spowodowany wyłączeniem mechanizmu redukcji ścian nie jest tak wielki jak w pierwszych testach, jednak nadal występuje. Najszybciej udało się otrzymać stan końcowy dla parametru ustawionego na 0,2 w pierwszym teście i 0,3 w drugim. Można to wytlumaczyć tym, że dzięki redukcji niepoprawnych ścian relaksacja trafia do stanów, w których jest mniejsze prawdopodobieństwo „zablokowania” się niektórych wierzchołków w cyklicznym ruchu wokół punktu lokalnego minimum energetycznego. Cykl taki może zostać przerwany dopiero przez mechanizm stopniowego „osłabiania” sił w układzie. Dlatego wyłączenie lub ograniczenie redukcji ścian może skutkować zwiększeniem liczby kroków relaksacji.

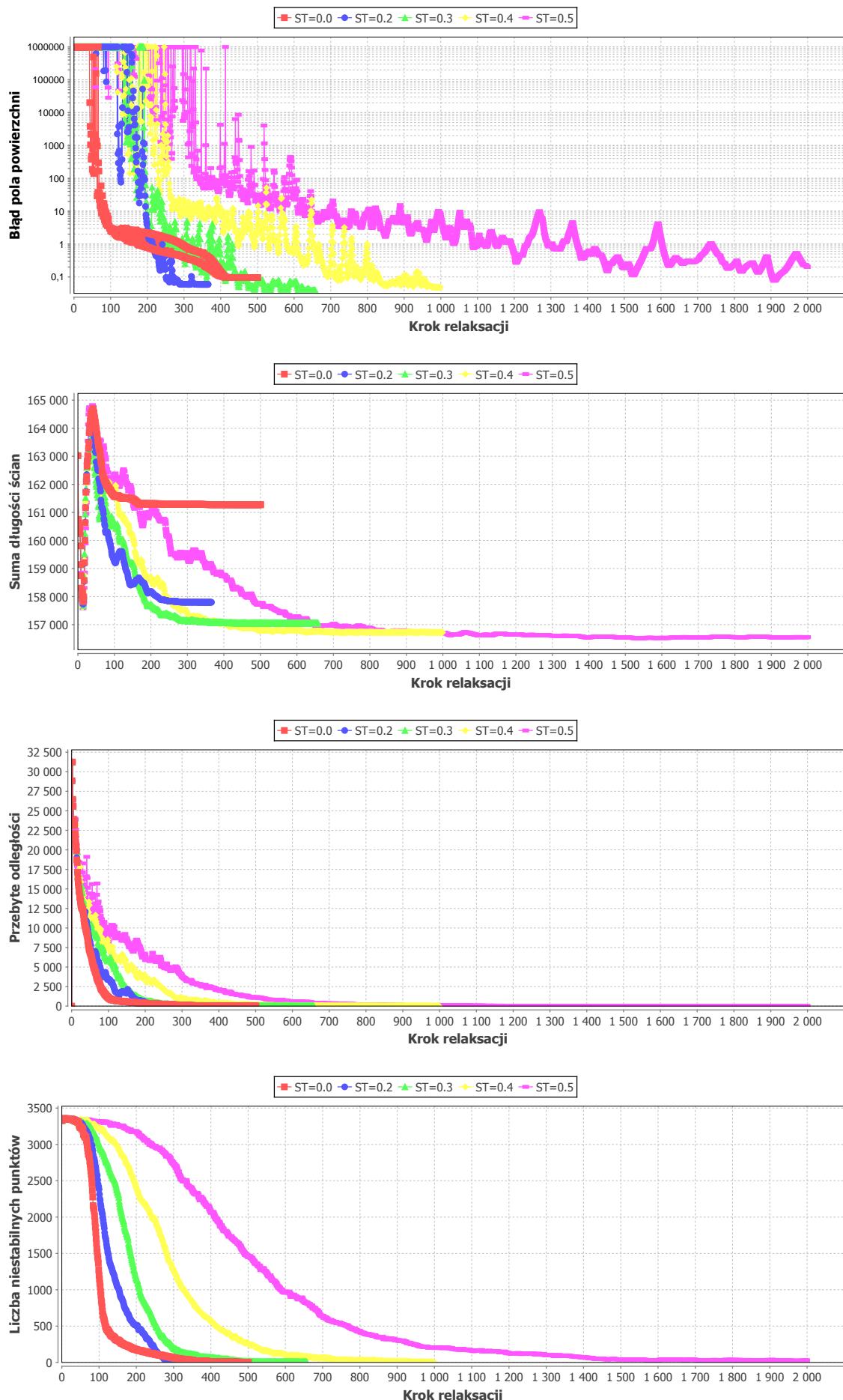
„Blokowanie” wierzchołków w cyklach nie dotyczy jednak całej piny, ale tylko pojedynczych bąbelków. Jak widać na wykresach przedstawiających liczbę niestabilnych punktów, reszta pinii zachowuje się zgodnie z intuicją — im mniej redukcji ścian, tym szybciej wierzchołki osiągają stan stabilności. Tak więc przy wyłączonej redukcji większość wierzchołków stabilizuje się bardzo szybko, ale wiele iteracji musi być przeznaczonych na stabilizację kilku ostatnich punktów (ta ostatnia faza nie wydłuża jednak znacząco czasu obliczeń).

Z badania parametru ST można wyciągnąć wniosek, że jest on bardzo ważny dla jakości znajdowanego minimum lokalnego, a najlepszą empirycznie wartością wydaje się być 0,3. Poza tym, aby zmniejszyć wpływ gwałtowności i przypadkowości procesu redukcji ścian na przebieg kolejnych testów, warto będzie korzystać ze stosunkowo dużych zestawów danych wejściowych.

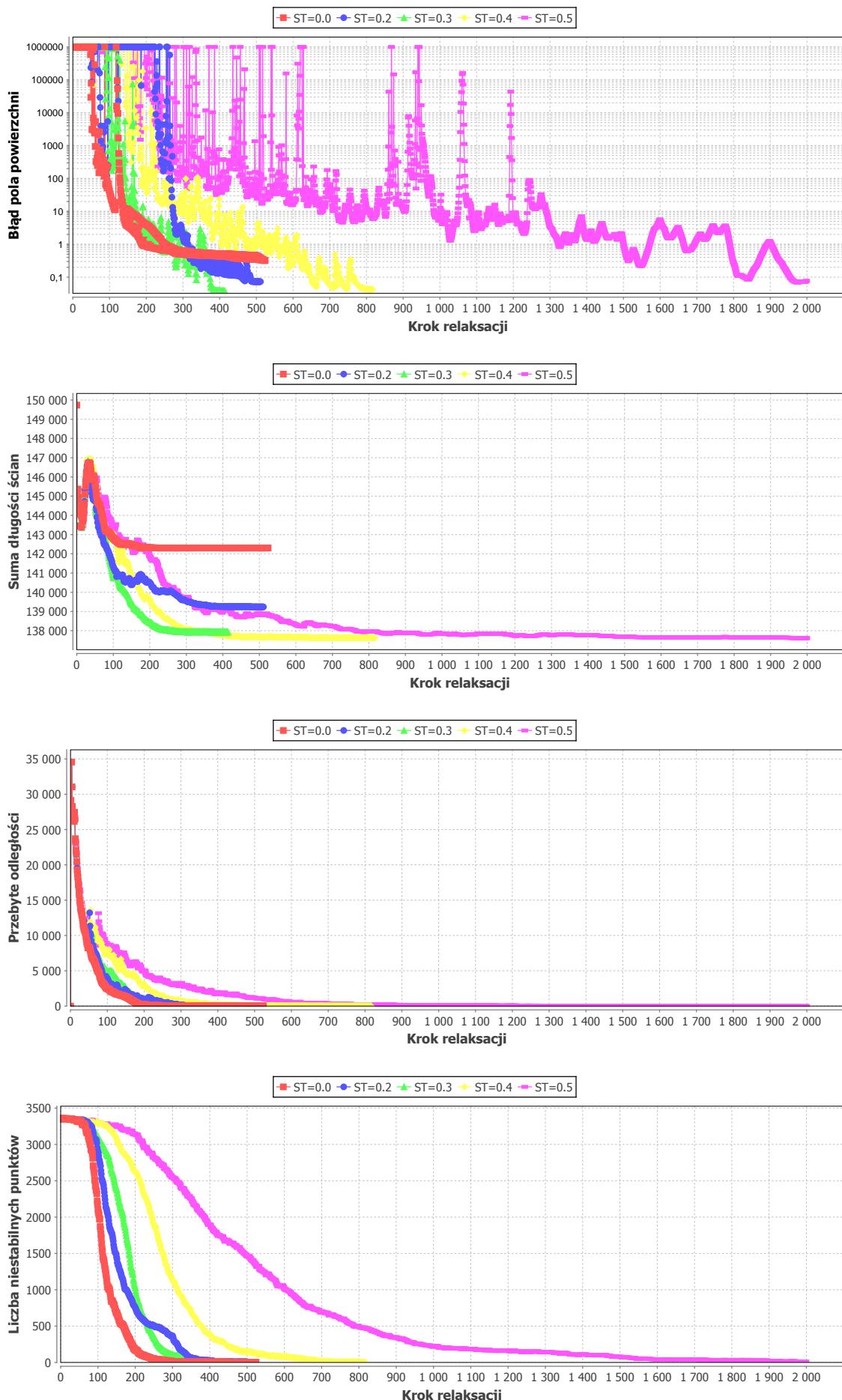
4.2.2 Squeeze angle

Drugi parametr związany z mechanizmem redukcji ścian to *squeeze angle* (SA). Gdy redukowana jest ściana leżąca na krawędzi bąbelka nadzewnętrznego, ten parametr określa maksymalny dopuszczalny kąt pomiędzy tą ścianą, a ścianą leżącą na kolejnej krawędzi. Gdy ten kąt będzie większy, ściana nie zostanie zredukowana. Standardową wartością jest $0,75\pi (\approx 2,36)$, co oznacza, że wierzchołek leżący na krawędzi bąbelka nadzewnętrznego może być dosunięty do jednego z jej końców tylko wtedy, gdy kąt przy tym wierzchołku jest mniejszy niż 135° .

W eksperymencie sprawdzono też wartość 0 (całkowite zablokowanie redukcji ścian leżących na



Rysunek 4.5: Wpływ parametru *squeeze threshold* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.6: Wpływ parametru *squeeze threshold* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie koła.

krawędzi bąbelka nadzewnętrznego), π (pozwolenie na redukcję dla wszystkich kątów wypukłych) oraz kilka wartości pośrednich. Wykorzystano ten sam zestaw danych zawierający 1680 liści, co przy badaniu parametru *squeeze threshold*. Badanie parametru SA na małej strukturze danych nie dało ciekawych rezultatów — przebiegi wyglądały bardzo podobnie dla różnych wartości parametru, ponieważ rzadko trafiały się sytuacje, w których parametr ten miał wpływ na zachowanie modelu. Wyniki zostały przedstawione na rysunkach 4.7 i 4.8.

Z wykresów widać, że im większa wartość parametru SA, tym lepsze minimum lokalne udaje się znaleźć, jednak stabilizacja wierzchołków trwa dłużej (co, podobnie jak w przypadku parametru ST, nie musi oznaczać większej liczby kroków potrzebnych do zakończenia relaksacji). Najlepszym kompromisem wydaje się być wartość 1,96 lub 2,36 — obie pozwalają uzyskać dość dobre wyniki przy stosunkowo niewielkim wzroście czasu obliczeń.

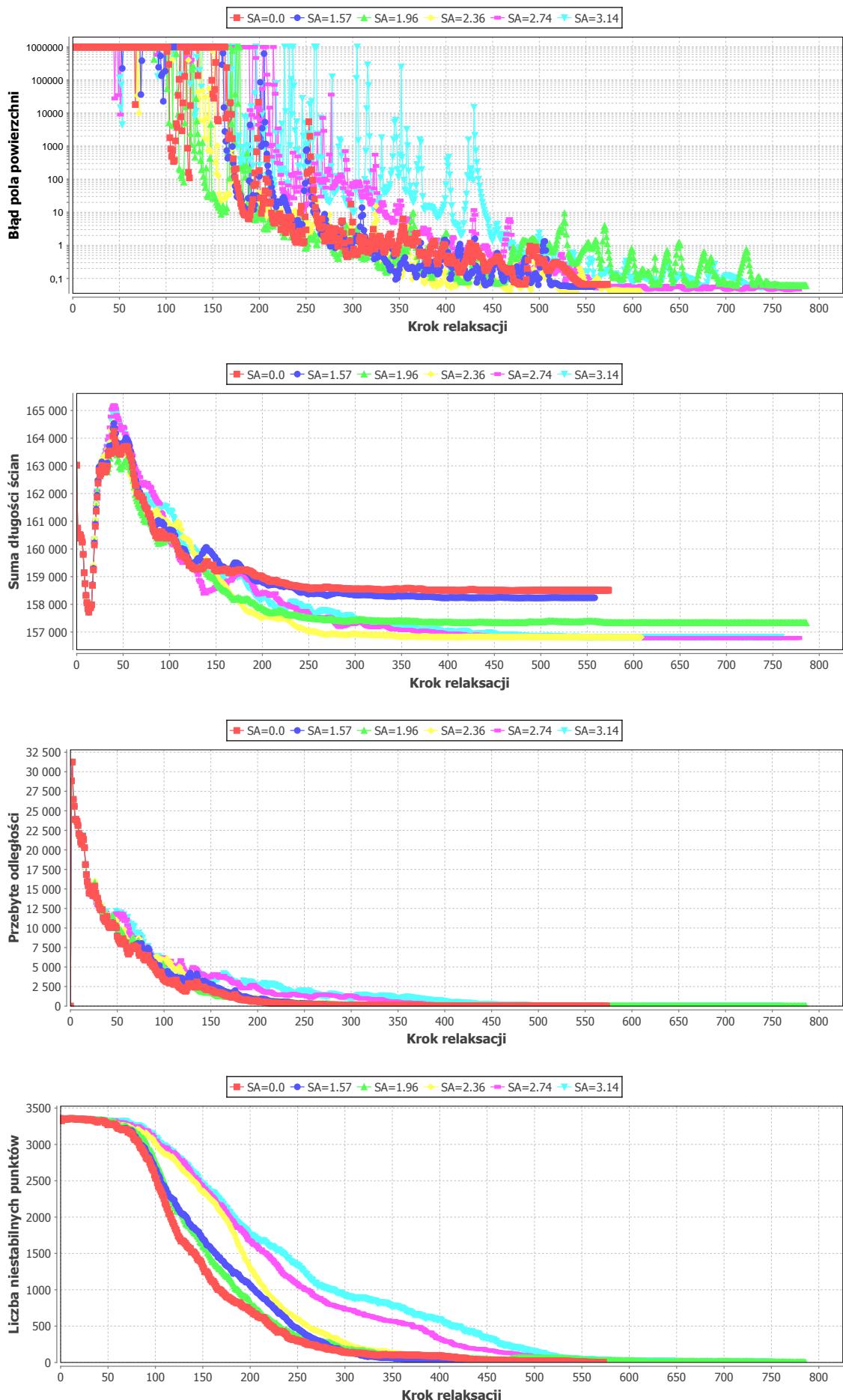
Największa różnica pomiędzy dwoma przeprowadzonymi testami jest widoczna dla wartości 3,14. Gdy bąbelek-korzeń ma kształt koła, wartość ta powoduje drastyczne wydłużenie czasu stabilizacji wierzchołków (liczba kroków przekroczyła 1500, jednak zdecydowano się ją ograniczyć ze względu na czytelność wykresów). Wynika to stąd, że w rzeczywistości bąbelek-korzeń ma kształt wielokąta o dużej liczbie wierzchołków i kątach o miarach zbliżonych do 180° . Zatem wierzchołek leżący na krawędzi bąbelka-korzenia bardzo często znajduje się blisko jednego z wierzchołków korzenia i tworzy z nim ściankę, która ze względu na swoją małą długość będzie uznana za dobrego kandydata do redukcji. W ten sposób dochodzi do bardzo wielu redukcji, które nie powodują istotnych zmian w ułożeniu piany, a jedynie oddalają układ od stanu równowagi. To właśnie obserwacja zachowania piany o „kolistym” kształcie bąbelka-korzenia była inspiracją do wprowadzenia parametru SA i, jak widać, w tej sytuacji przynosi on największą korzyści.

4.2.3 *Squeeze candidate times*

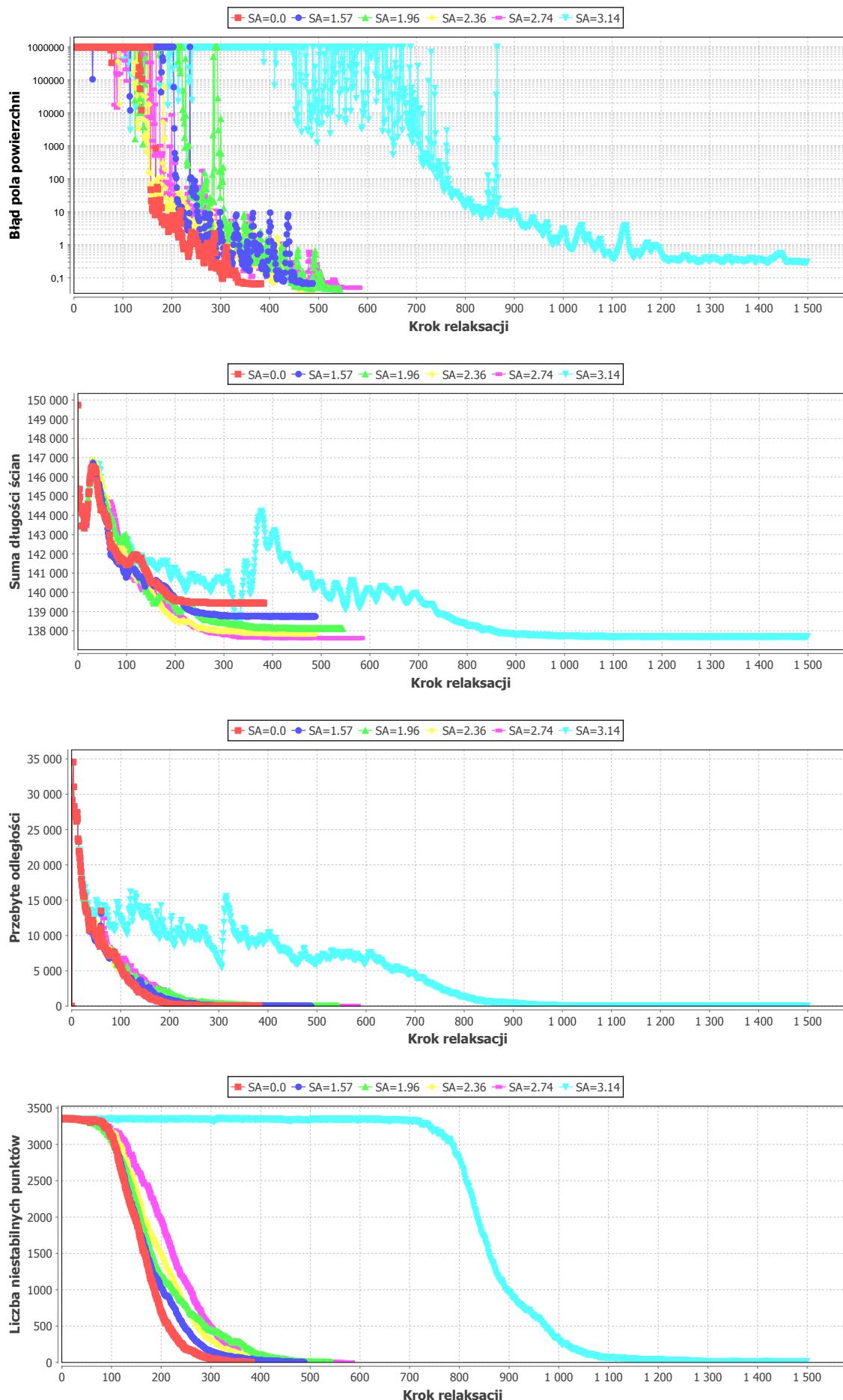
Mechanizm redukcji ścian jest jeszcze kontrolowany przez trzeci parametr: *squeeze candidate times* (SC). Wartość tego parametru to liczba razy, jaką ścianą musi zostać uznana za dobrego kandydata do redukcji, zanim rozpoczęnie się faktyczna redukcja. Domyślną wartością jest 15. Tym razem do badań wybrano instancje o dwóch rozmiarach (48 i 1680 liści), ale tylko w wersji wielopoziomowej i z układem w kształcie kwadratu. Dla zestawów jednopoziomowych różnice w wynikach dla różnych wartości parametrów były mniej widoczne, a ustalenie kolistego kształtu bąbelka-korzenia nie wpływało w istotny sposób na rezultaty. Wyniki eksperymentów zostały przedstawione na rysunkach 4.9 i 4.10.

Ogólny wniosek z eksperymentu jest taki sam dla obu badanych zestawów danych, mimo dużej różnicy w ich rozmiarach: zarówno zbyt mała (a konkretnie — zerowa) jak i zbyt duża wartość parametru SC wpływa negatywnie na jakość znajdowanego minimum lokalnego. Gdy wartość jest równa 0, wiele ścian jest niepotrzebnie redukowanych już na początku relaksacji, nawet gdy w tym momencie nie poprawi to stanu piny. Dodatkowo, tuż po redukcji, zanim ściana zdąży się wydłużyć, zostaje zredukowana ponownie, ponieważ jest krótsza niż na początku. Z kolei w późniejszej fazie, gdy rzeczywiście zajdzie potrzeba zredukowania takiej ściany, będzie to niemożliwe, ponieważ będzie ona dłuższa niż przed ostatnią redukcją. Na rys. 4.9 można zauważyć, że w pierwszych kilkunastu krokach to właśnie dla SC=0 długości ścianek i przebyte odległości są zdecydowanie największe, co wskazuje na dużą liczbę redukcji ścian.

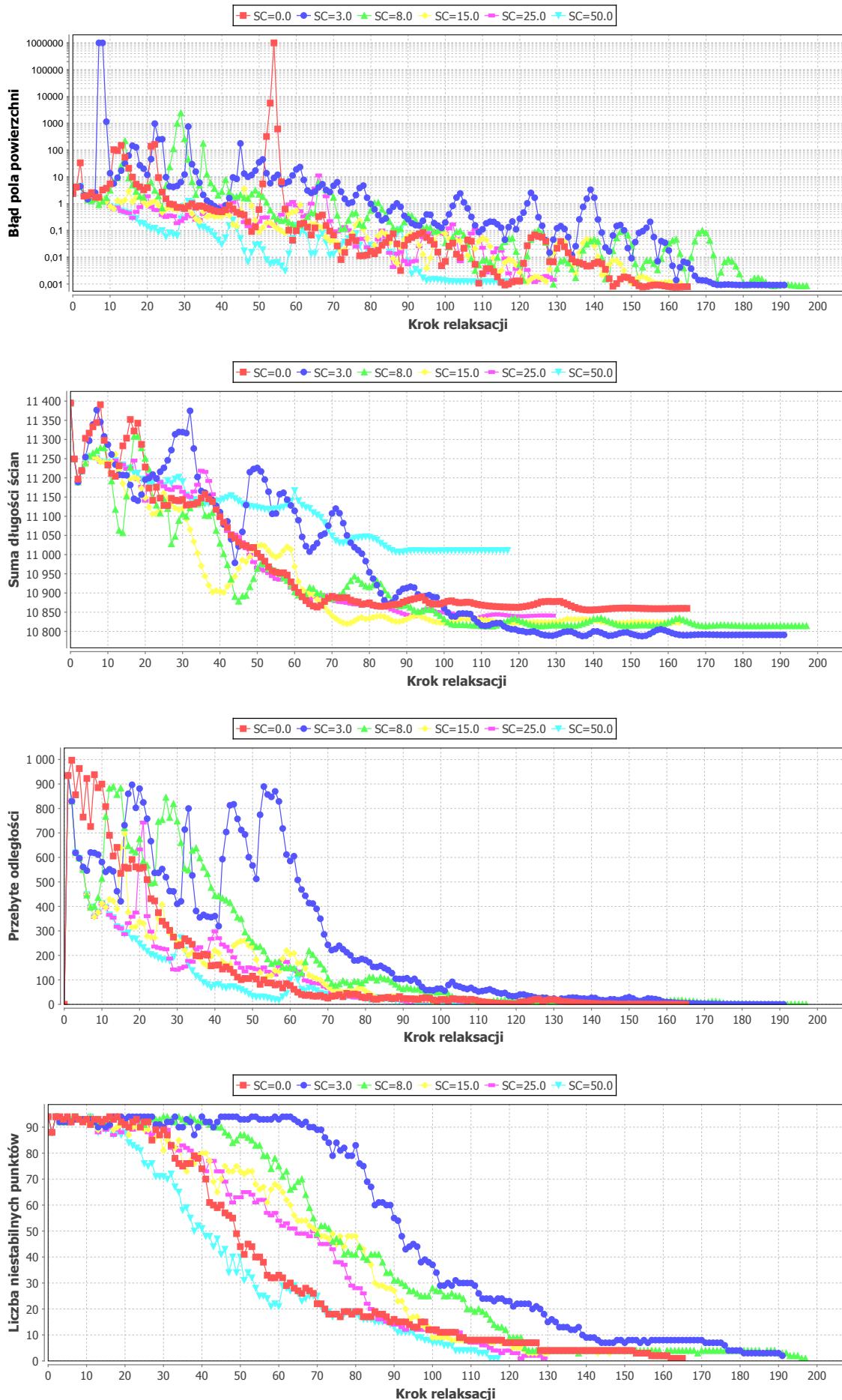
W przypadku zbyt dużej wartości SC może się zdarzyć, że zanim dojdzie do redukcji odpowiedniej ściany, zawierającą ją grupa bąbelków ustabilizuje się i obliczenia na niej nie będą kontynuowane. Taki fragment piany pozostanie więc do końca relaksacji w takim stanie, jakby redukcja była wyłączona. Na podstawie kształtu wykresów na rys. 4.9 można zauważyć, że dla wartości SC=50 pierwsze



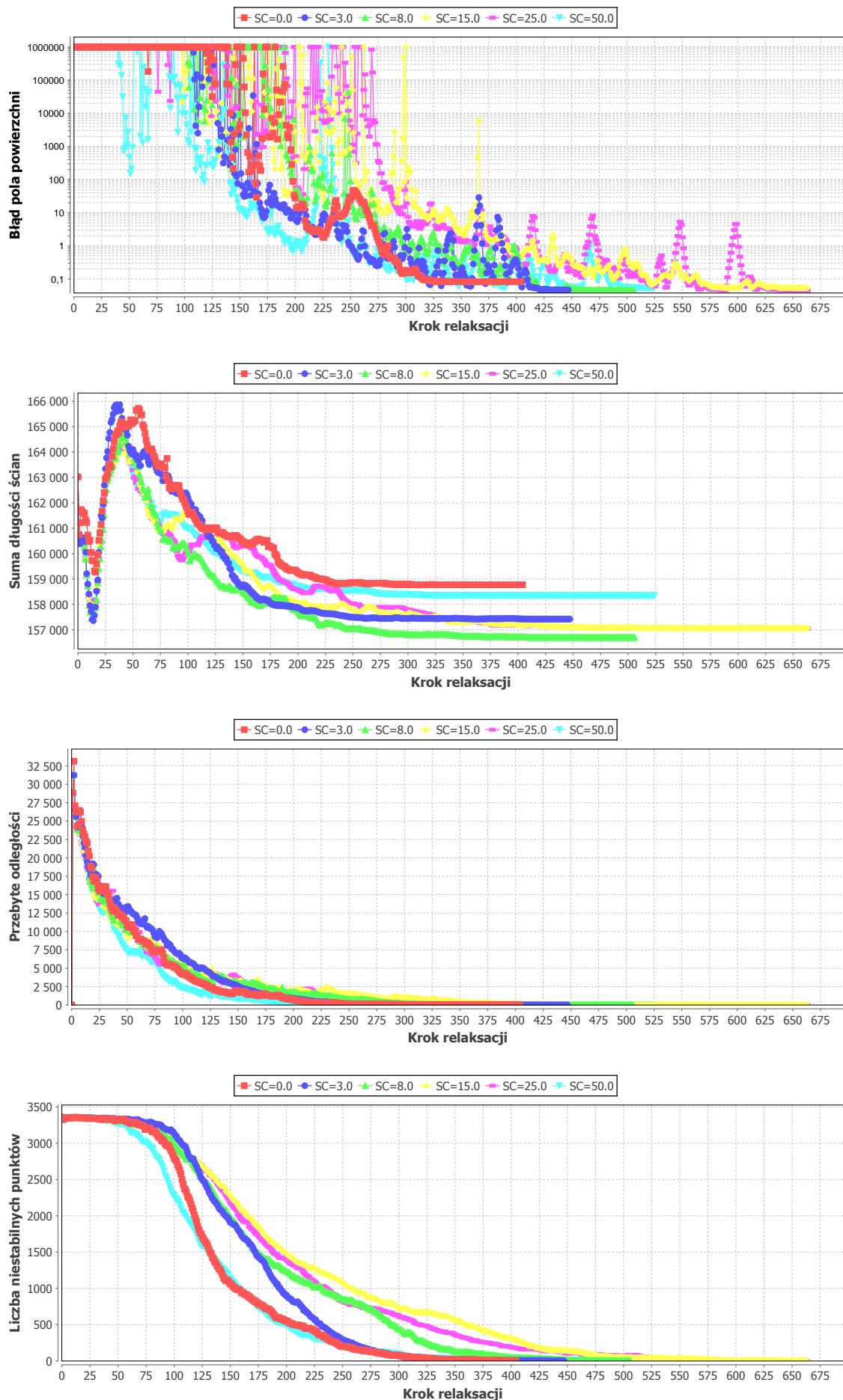
Rysunek 4.7: Wpływ parametru *squeeze angle* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.8: Wpływ parametru *squeeze angle* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kola.



Rysunek 4.9: Wpływ parametru *squeeze candidate times* na relaksację dla struktury trójpoziomowej o 48 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.10: Wpływ parametru *squeeze candidate times* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.

redukcje rozpoczęły się około kroku 30. Oczekiwanie na redukcję nie trwało pełnych 50 kroków, ponieważ ściana może zostać uznana za dobrego kandydata do redukcji dwa razy w ciągu jednej iteracji, jeśli jest najkrótszą ścianą w obu bąbelkach, do których należy.

Pozostałe, pośrednie wartości parametru SC skutkują dość podobnymi przebiegami relaksacji i trudno je między sobą porównywać. Zarówno szybkość stabilizacji piany, jak i jakość stanu końcowego wydają się zależeć bardziej od przypadku niż od wartości parametru SC. Trudno więc oszacować, na jaką wartość najlepiej ustawić ten parametr.

4.3 Równowaga sił w modelu fizycznym

W tej serii eksperymentów zbadany zostanie wpływ parametrów kontrolujących podstawowe siły w modelu: ciśnienie, sprężanie ścian (napięcie powierzchniowe) oraz wyrównywanie kątów.

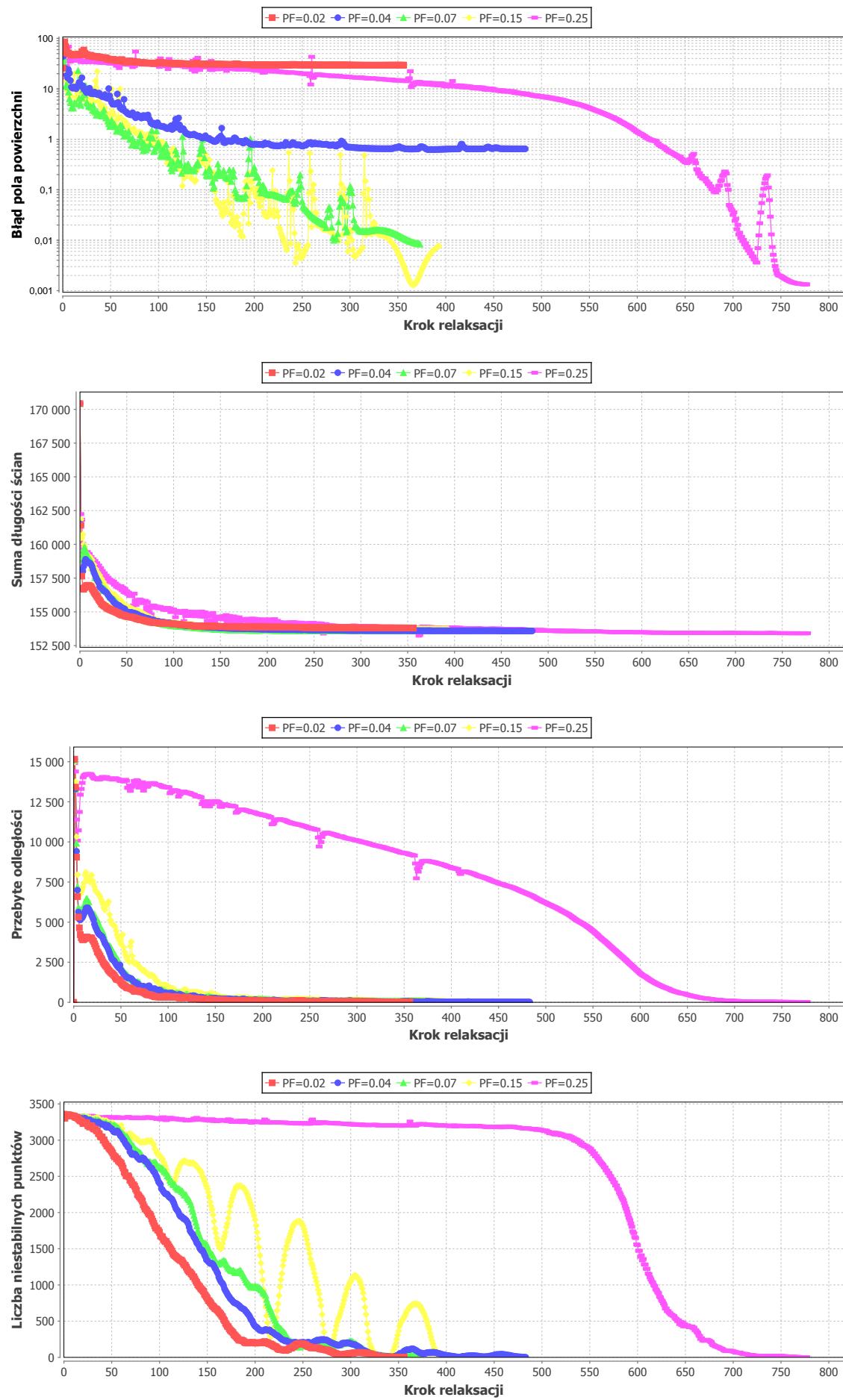
4.3.1 Pressure factor

Parametr kontrolujący ciśnienie został nazwany *pressure factor* (PF). Wartość domyślna to 0,07. Testy przeprowadzono tylko na dużych zbiorach danych (1680 liści), zgodnie z wnioskami wyciągniętymi z badania parametrów mechanizmu redukcji ścian. Przygotowano również strukturę jednopoziomową, aby sprawdzić, czy charakter podziału struktury wpływa na wyniki. Rezultaty zostały przedstawione na rys. 4.11 i 4.12.

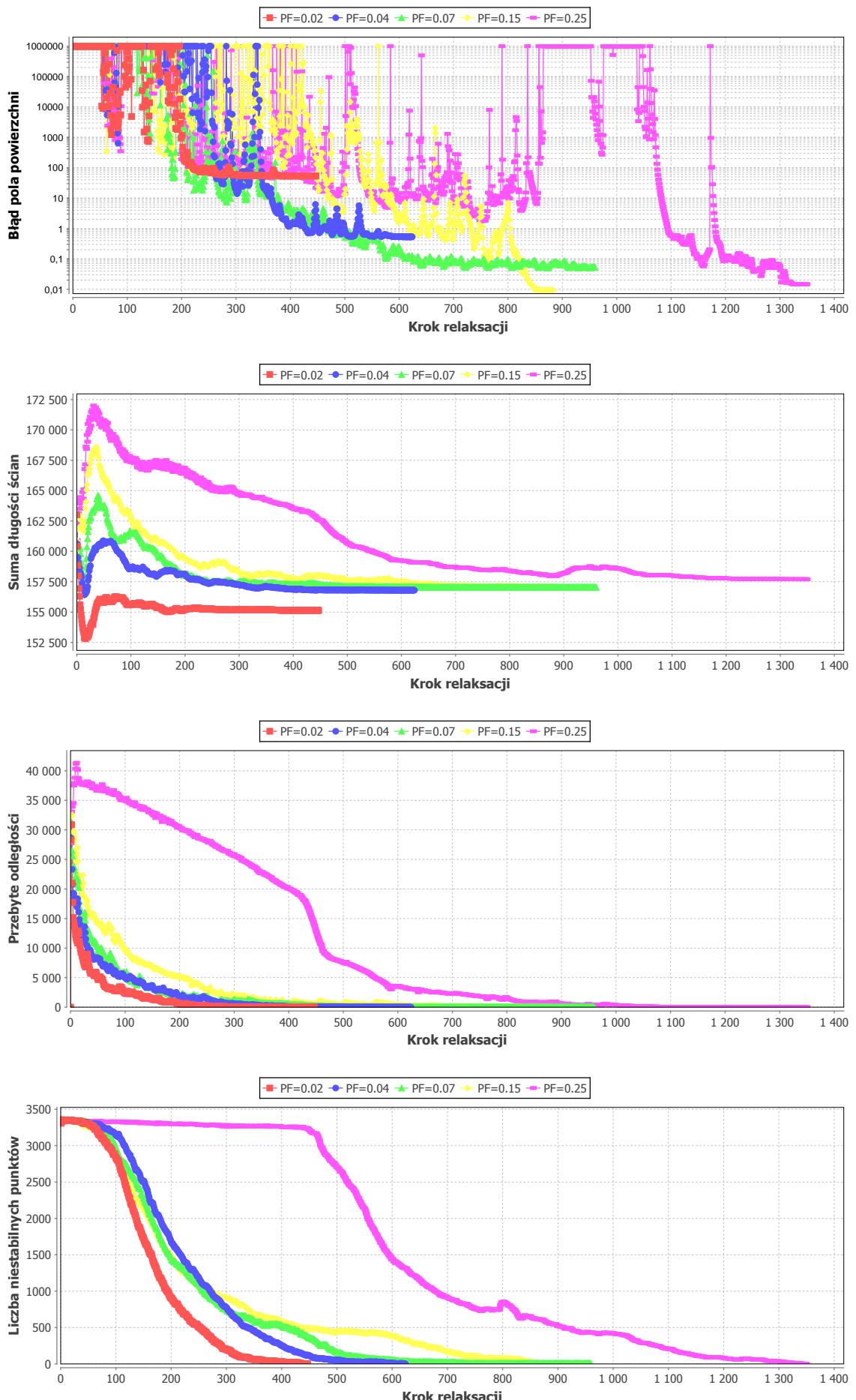
Pierwsze, co należy zauważyć, to fakt, że przy zbyt małych wartościach parametru PF (0,02, w mniejszym stopniu 0,04) błędy pól powierzchni są zdecydowanie zbyt duże, by zaakceptować taką wizualizację. Wynika to stąd, że ciśnienie nie jest wystarczająco silne, by zrównoważyć pozostałe siły (zwłaszcza napięcie powierzchniowe), przez co niektóre bąbelki „zapadają się” pod ich wpływem. Dla większych wartości parametru błędy są już wystarczająco małe, jednak zależność błędu od siły ciśnienia nie jest monotoniczna — dla największej wartości 0,25 uzyskano w drugim teście większy błąd niż dla 0,15. Prawdopodobnie jest to związane z większą gwałtownością ruchu wierzchołków, rzucającą się w oczy na wykresach przedstawiających liczbę niestabilnych punktów i przebyte odległości. Niemal cała piana pozostaje niestabilna na długo po tym, jak relaksacje dla mniejszych wartości PF są prawie zakończone. Powolne, stabilne zmniejszanie się przebytych odległości w początkowej części wykresu wskazuje na to, że praktycznie wszystkie wierzchołki są „zablokowane” w cyklicznym ruchu wokół swoich punktów energetycznego minimum i poruszają się z maksymalną dopuszczalną szybkością, która jest stopniowo zmniejszana.

Charakterystyczną cechą relaksacji w strukturze jednopoziomowej jest oscylacja liczby niestabilnych punktów, widoczna dla wartości 0,15. Bierze się ona stąd, że wszystkie bąbelki ze sobą oddziałują — gdy w jednym miejscu zajdzie gwałtowniejsza zmiana, zmieniająca pole powierzchni bąbelka (na przykład redukcja ściany lub transformacja), zaburzenie to rozpropaguje się stopniowo po niemal całej pianie, wprawiając w ruch również wierzchołki, które były już ustabilizowane. Większość bąbelków szybko udaje się znaleźć nowy stan równowagi i liczba aktywnych wierzchołków znów spada. Dla mniejszych wartości parametru oscylacja nie jest widoczna, ponieważ zaburzenia nie roznodzą się tak dobrze (niewielkie zmiany ciśnienia w bąbelku nie powodują gwałtownej reakcji wszystkich jego wierzchołków).

Ogólnie można powiedzieć, że wartość 0,07 daje najlepsze rezultaty spośród badanych możliwości. Wartość 0,15 również pozwala uzyskać poprawny stan końcowy w sensownym czasie, jednak powoduje wolniejszą stabilizację wierzchołków.



Rysunek 4.11: Wpływ parametru *pressure factor* na relaksację dla struktury jednopoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.12: Wpływ parametru *pressure factor* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.

4.3.2 Spring factor

Drugi badany parametr odpowiada za napięcie powierzchniowe, które sprawia, że ścianki bąbelków „próbuja” się kurczyć niczym sprężyny: *spring factor* (SF). Wartością standardową jest 0,09. Warunki eksperymentu są takie same jak w przypadku parametru *pressure factor*. Rezultaty zostały przedstawione na rys. 4.13 i 4.14.

Wyniki potwierdzają spostrzeżenie dokonane w trakcie analizy parametru PF: zbyt duży wpływ napięcia powierzchniowego w porównaniu z ciśnieniem prowadzi do dużych błędów w polach powierzchni bąbelków. Zaskakujące jest jednak to, że parametr SF nie wpływa w istotny sposób na sumę długości ścianek w stanie końcowym. Mogłoby się wydawać, że większa siła sprężająca ścianki powinna zmniejszać ich końcową długość. Zależność taką widać jedynie w przypadku struktury czteropoziomowej, przy czym różnice są znaczące jedynie na początku relaksacji, później niemal zanikają. Dla struktury jednopozyciowej długości ścianek kształtoły się prawie tak samo niezależnie od wartości parametru.

Wartość parametru ma jednak wyraźny wpływ na szybkość stabilizacji piany. Szczególnie zwraca uwagę zachowanie układu przy zbyt dużej wartości SF. Podobnie jak w przypadku zbyt dużego ciśnienia, wiele punktów „oscyluje” wokół punktów równowagi z maksymalną dopuszczalną amplitudą, co objawia się dużymi wartościami na wykresach przebytych odległości. Z kolei, gdy wartość parametru jest za mała, stosunkowo dużo wierzchołków pozostaje niestabilnych w końcowej fazie relaksacji, przez co ogólna liczba kroków się zwiększa. Domyślna wartość 0,09 okazuje się być tutaj dobrym kompromisem.

4.3.3 Angle factor

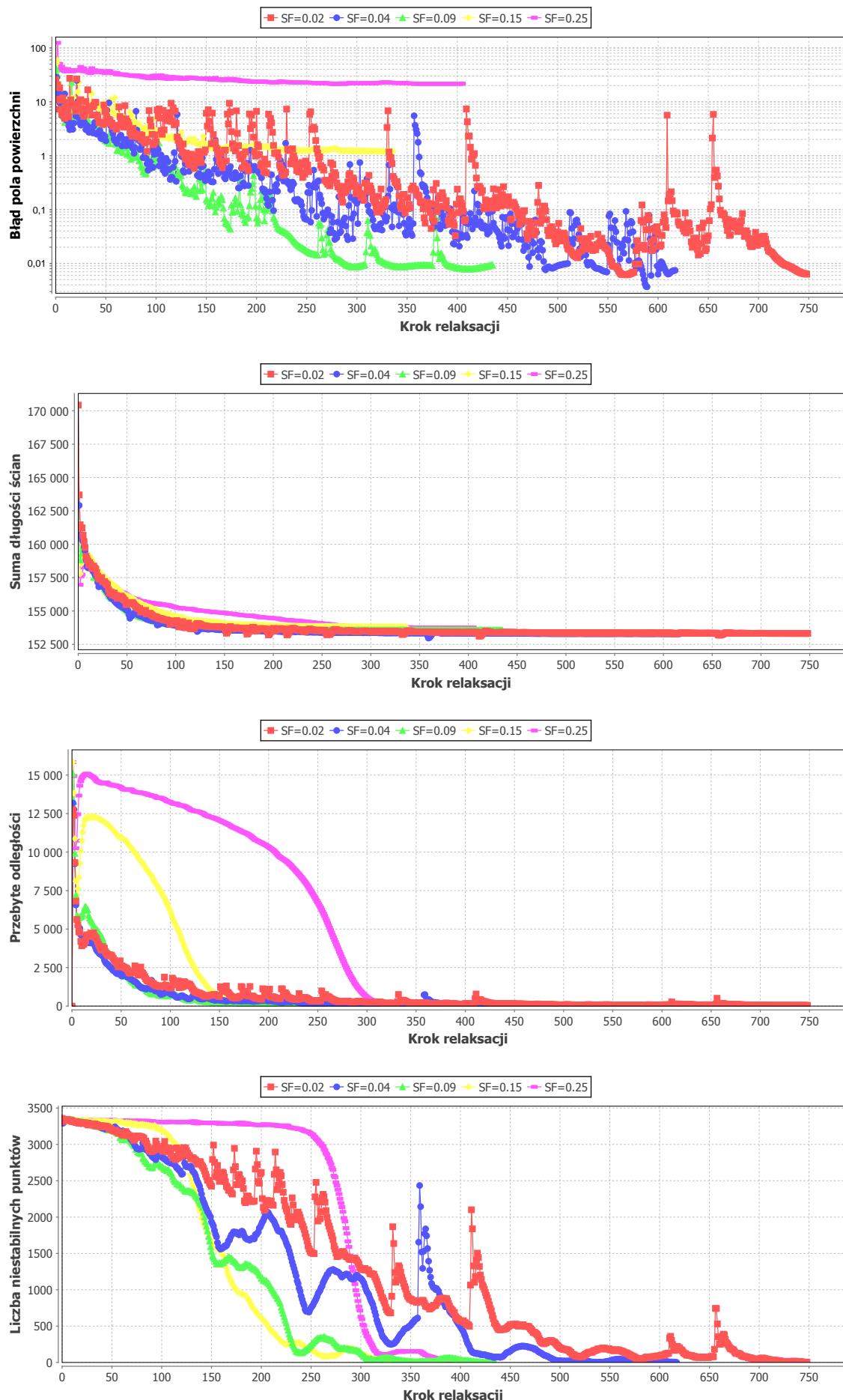
Ostatni parametr w tej serii pomiarów, *angle factor* (AF), kontroluje siłę, która dąży do wyrównania kątów pomiędzy stykającymi się ścianami. Domyślnie parametr ten ma wartość 0,15. Wyniki eksperymentów zostały pokazane na rysunkach 4.15 i 4.16.

Na podstawie wykresów przedstawiających sumy długości ścian można stwierdzić, że im większa wartość parametru AF, tym lepsza jakość uzyskiwanego stanu końcowego. Zbyt duża wartość parametru AF jest jednak niekorzystna, ponieważ opóźnia stabilizację wierzchołków i wiele z nich wprawia w drgania (co widać w postaci dużych wartości na wykresie przebytych odległości w początkowej fazie relaksacji).

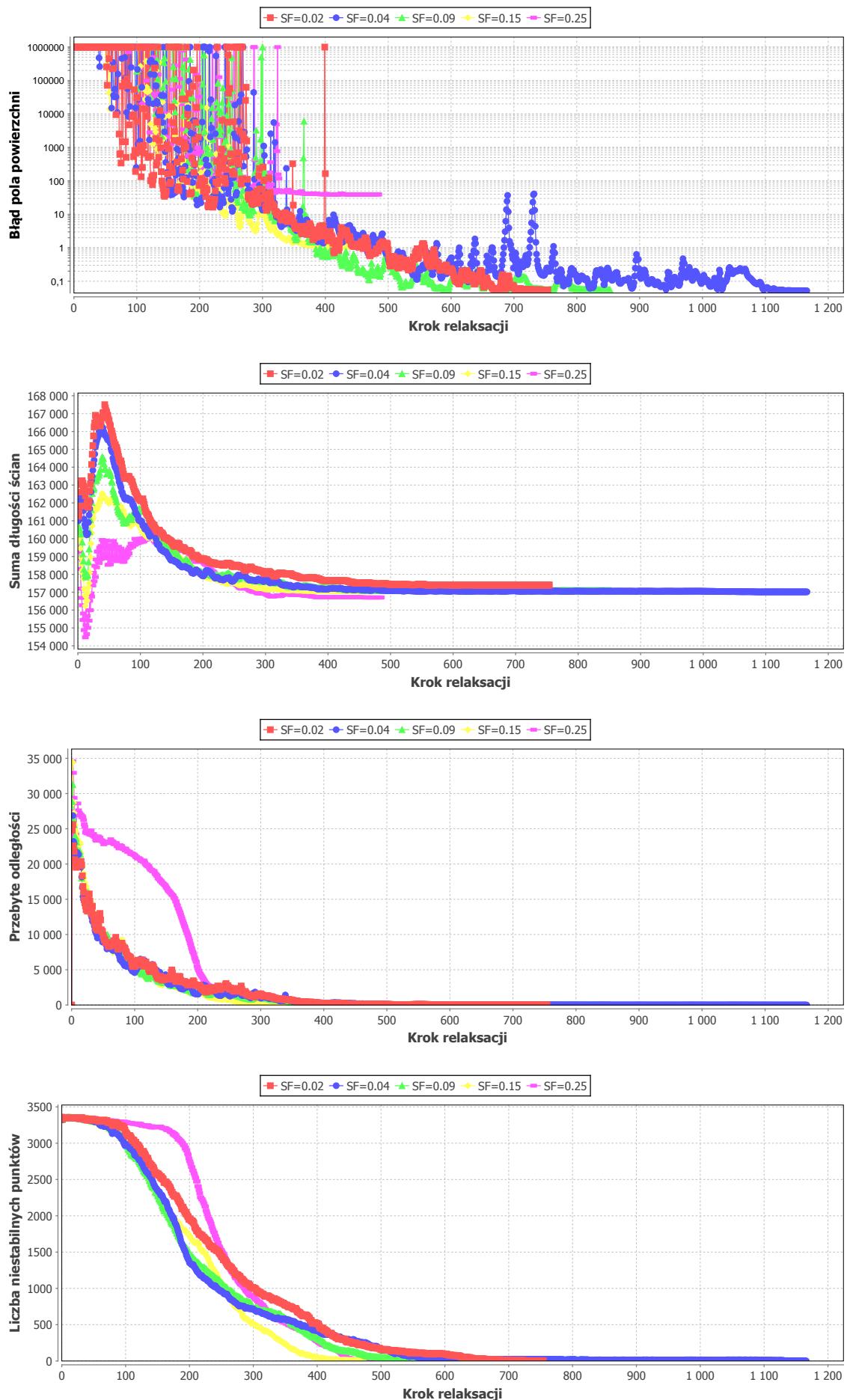
Przy niższych wartościach parametru AF nie wpływa znacząco na szybkość ruchu i czas stabilizacji wierzchołków — przebiegi wykresów niemal się pokrywają dla różnych wartości. Mimo to, można zauważać, że najczęściej zwiększenie wartości parametru powoduje zwiększenie liczby kroków potrzebnych do zakończenia relaksacji. Oznacza to, że duża siła wyrównująca kąty zwiększa prawdopodobieństwo „zablokowania” niektórych wierzchołków w cyklicznym ruchu i utrudnia przerwanie takich oscylacji.

Błędy pola powierzchni są wystarczająco małe niezależnie od wartości parametru AF. Można zauważać, że przy większych wartościach tego parametru wykresy przedstawiające błędy stają się coraz bardziej niestabilne, jednak wahania nie osiągają dużych amplitud i nie wpływają na błąd w stanie końcowym.

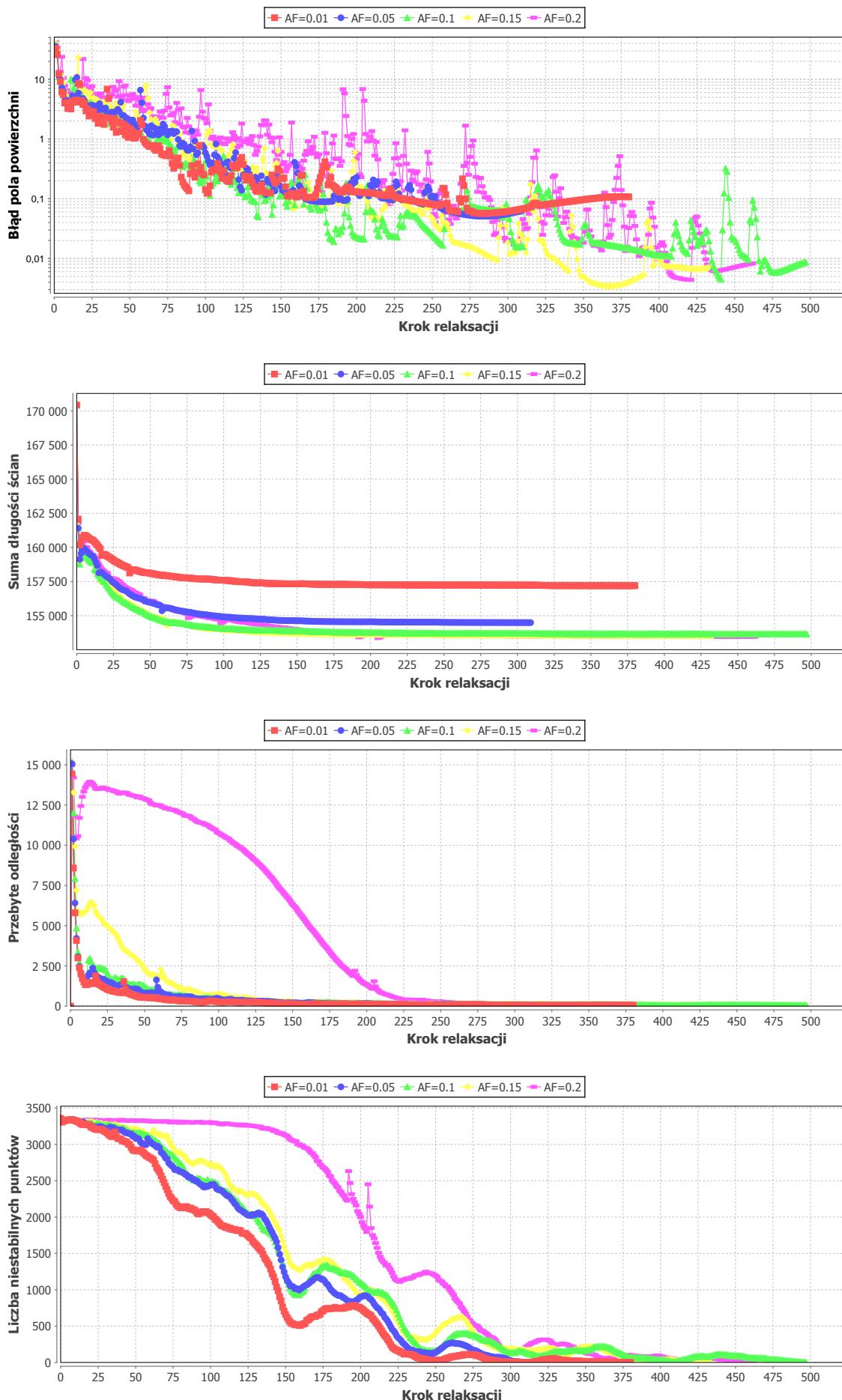
Z badania wynika, że dobrana standardowa wartość parametru *angle factor* jest trochę za duża. W teście na jednowarstwowym zbiorze danych wartość 0,15 powoduje stosunkowo gwałtowne ruchy wierzchołków na początku relaksacji, a na strukturze wielopoziomowej znacznie zwiększa liczbę iteracji, nie poprawiając jakości stanu końcowego (choć akurat w tym teście największa z badanych wartości, 0,2, skutkuje o wiele mniejszą liczbą kroków — świadczy to o tym, że duże znaczenie ma przypadek i nawet przy nieoptimalnych wartościach parametrów relaksacja może natrafić na stan, z którego



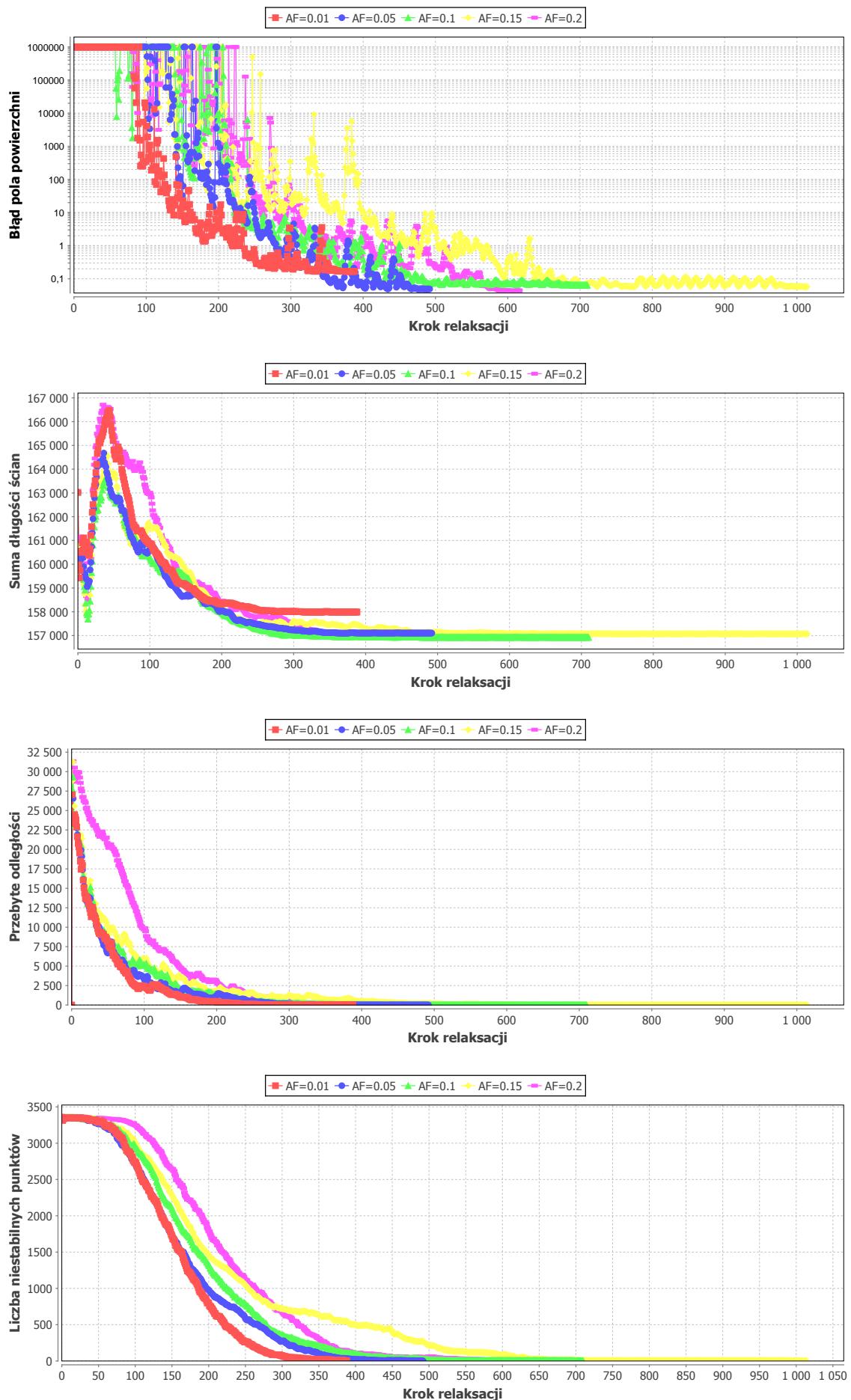
Rysunek 4.13: Wpływ parametru *spring factor* na relaksację dla struktury jednopoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.14: Wpływ parametru *spring factor* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.15: Wpływ parametru *angle factor* na relaksację dla struktury jednopoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.16: Wpływ parametru *angle factor* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.

można szybko otrzymać stan stabilny).

4.4 Stabilizacja i kontrola szybkości wierzchołków

Ostatnia seria badań dotyczy parametrów związanych z ograniczaniem zakresu ruchu wierzchołków i oceną ich stabilności.

4.4.1 Speed limit

Wykresy na rys. 4.17 i 4.18 przedstawiają rezultaty badania parametru *speed limit* (SL). Parametr ten, o domyślnej wartości 0,1, wpływa na maksymalną szybkość, z jaką mogą poruszać się wierzchołki bąbelków. Bezwzględny dystans, jaki może przebyć wierzchołek w każdym kroku relaksacji, jest proporcjonalny do średniej długości średnic bąbelków zawierających ten wierzchołek oraz ich rodzeństwa. Dystans ten jest stopniowo zmniejszany z czasem.

W przypadku struktury jednopoziomowej, zmiana parametru SL wpływa na przebieg relaksacji, jednak nie zmienia jego zasadniczych własności. Przy najmniejszych wartościach parametru, w początkowej fazie odległości pokonywane przez wierzchołki są mniejsze, a długości ścian wolniej zbliżają się do minimum lokalnego. Różnice te jednak szybko się zacierają. Szybkość stabilizacji piany, a także długości ścian w stanie końcowym wyglądają bardzo podobnie bez względu na wartość parametru.

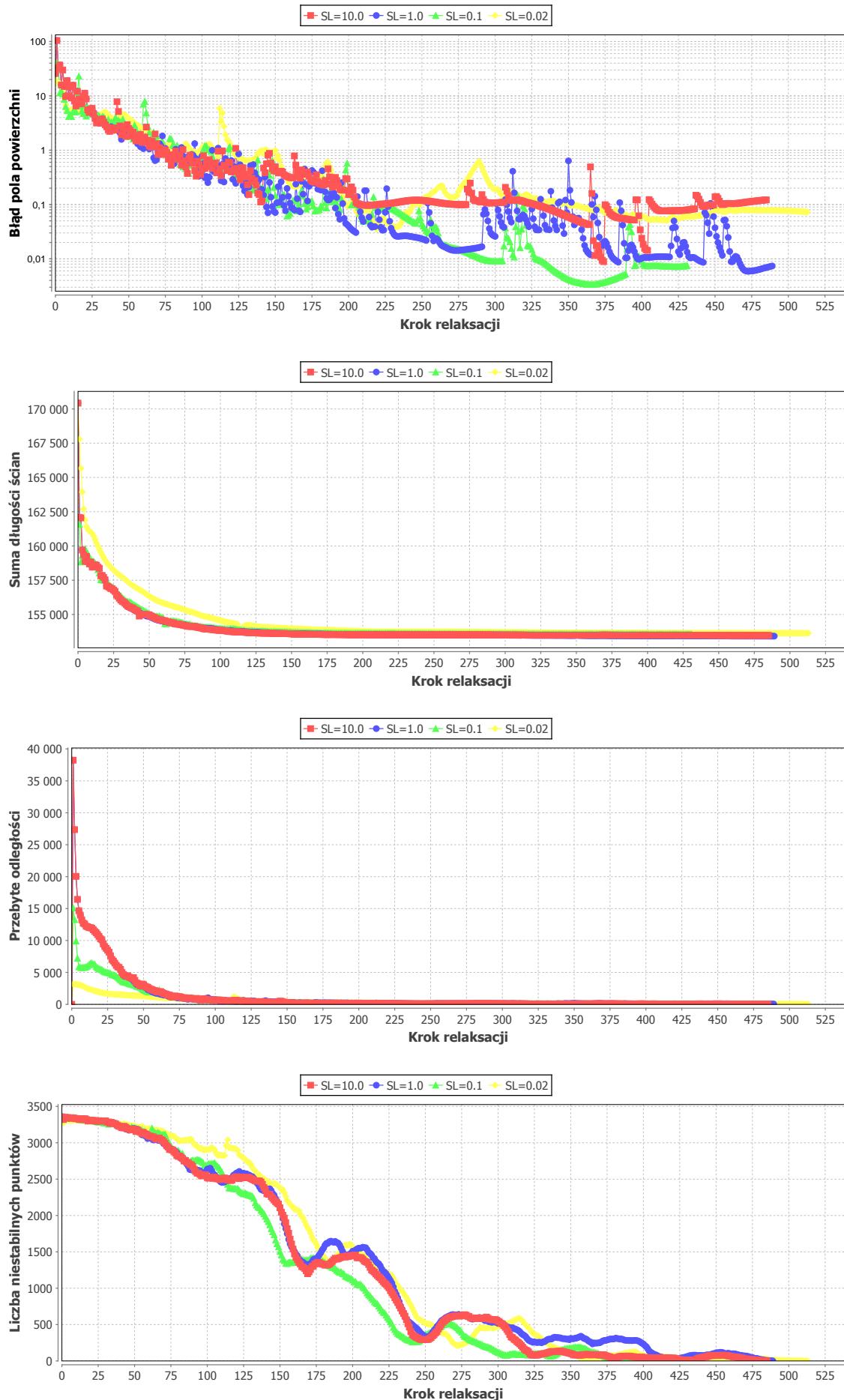
Bardziej znaczące różnice można zauważać w przypadku struktury wielopoziomowej. Zwiększenie limitu szybkości do wartości 1 i więcej powoduje coraz bardziej gwałtowne ruchy wierzchołków i większe zmiany długości ścian w początkowej fazie relaksacji (wartości na wykresie przedstawiającym długości ścian wielokrotnie przekraczały wartość 200 000, ale zostały ograniczone, aby poprawić czytelność). Jednak, podobnie jak wcześniej, po kilkuset krokach różnice te zanikają, zatem jakość uzyskanego stanu końcowego pozostaje na tym samym poziomie, natomiast liczba kroków potrzebnych do zakończenia relaksacji wręcz się poprawia. Z kolei zmniejszenie limitu szybkości prowadzi do wolniejszej stabilizacji wierzchołków oraz nawet kilkukrotnego wzrostu liczby kroków w relaksacji. Testy wskazują więc, że prawdopodobnie warto zwiększyć standardową wartość parametru SL, jednak nie przekraczając wartości 1.

4.4.2 Minimum speed

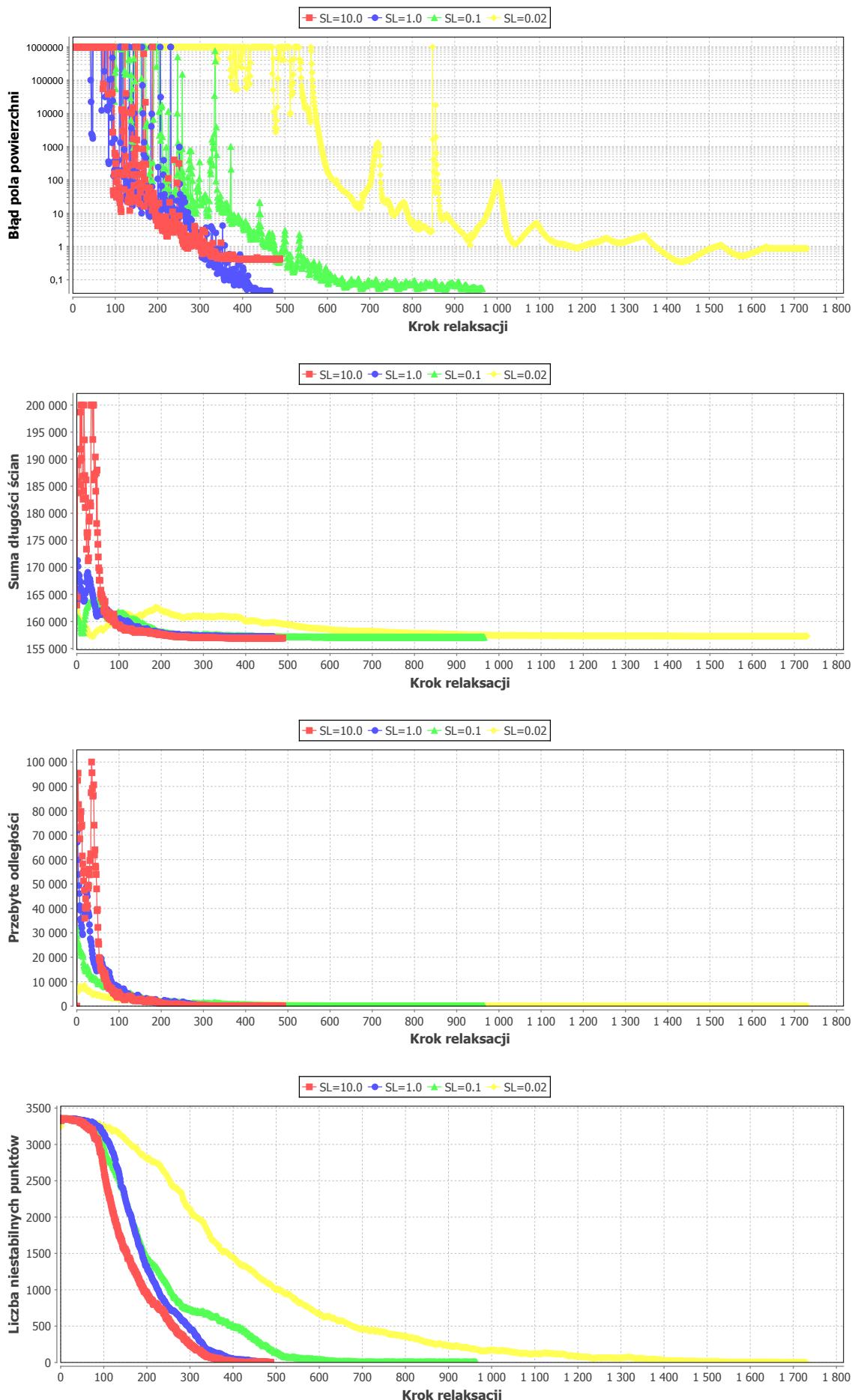
Parametrem w pewnym sensie komplementarnym do *speed limit* jest *minimum speed* (MS). Wpływa on na minimalną szybkość, z jaką muszą poruszać się wierzchołki bąbelków, aby nie zostały uznane za ustabilizowane. Domyślna wartość parametru to 0,001. Rys. 4.19 i 4.20 przedstawiają przebieg relaksacji dla różnych badanych wartości.

W przypadku jednopoziomowego zbioru danych zmiana wartości MS silnie wpływa na moment zakończenia relaksacji, ale nie jej przebieg — wykresy błędów, długości ścian i przebytych odległości niemal się pokrywają. Największa wartość parametru (0,02) powoduje zakończenie relaksacji już po pierwszym kroku, podczas gdy najmniejsza (10^{-4}) — wydłuża ją do ponad 1600 kroków, przy czym duża część wierzchołków pozostaje aktywna prawie do samego końca (dla pozostałych wartości liczba niestabilnych wierzchołków bardzo szybko spada, a końcowa setka kroków relaksacji działa już na pojedynczych wierzchołkach).

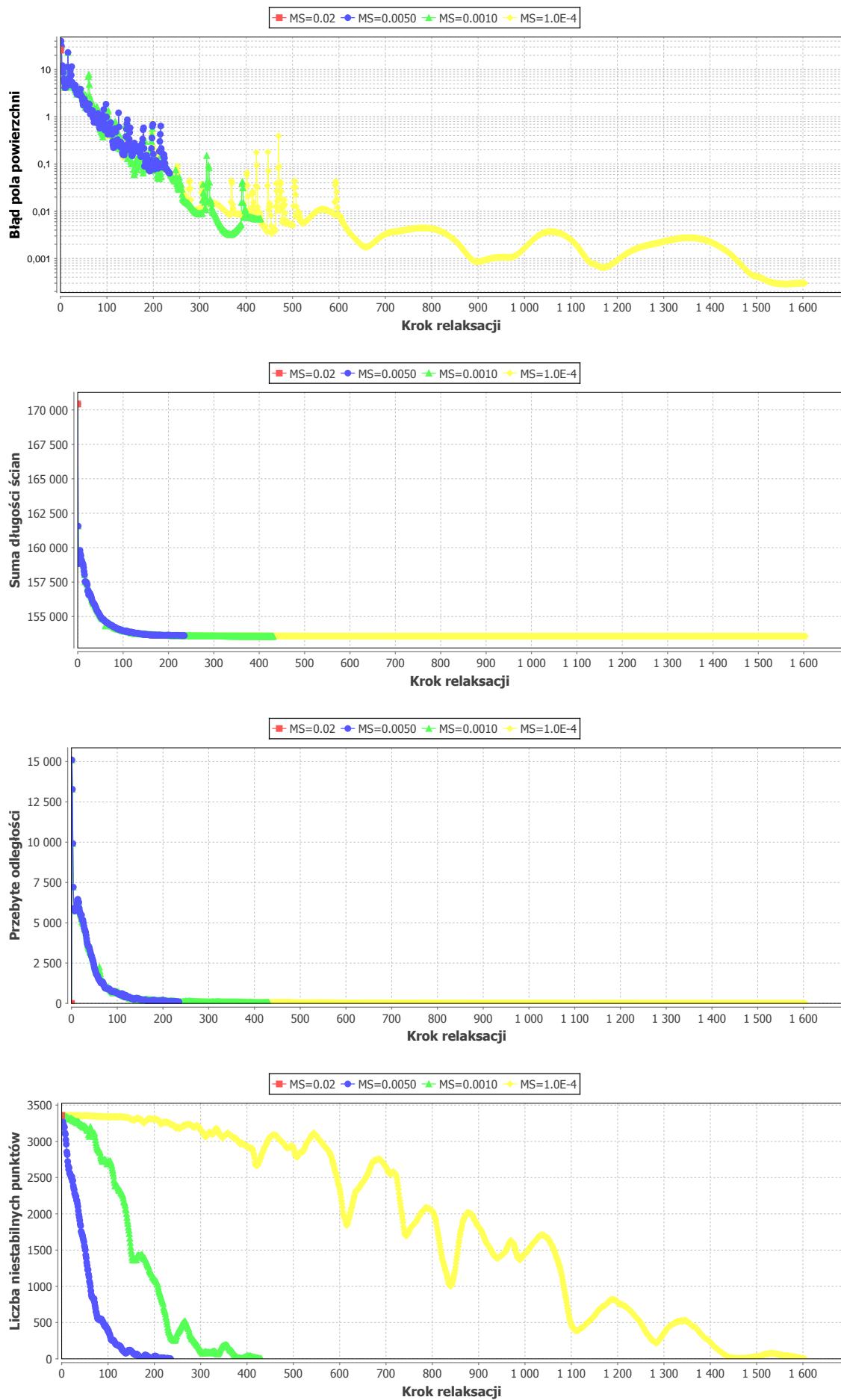
Dla struktury wielopoziomowej sytuacja jest bardziej skomplikowana — parametr *minimum speed* wpływa na wszystkie cechy symulowanej piany. Przyczyną tego jest fakt, że gdy ustabilizują się wszystkie bąbelki należące do wspólnego rodzica, obliczenia nie są dla nich kontynuowane. Zatem, podczas gdy przy dużej wartości MS wiele wierzchołków zostanie bardzo szybko uznanych za ustabilizowane i nie zmieni swojej pozycji aż do końca relaksacji, przy mniejszych wartościach parametru wierzchołki



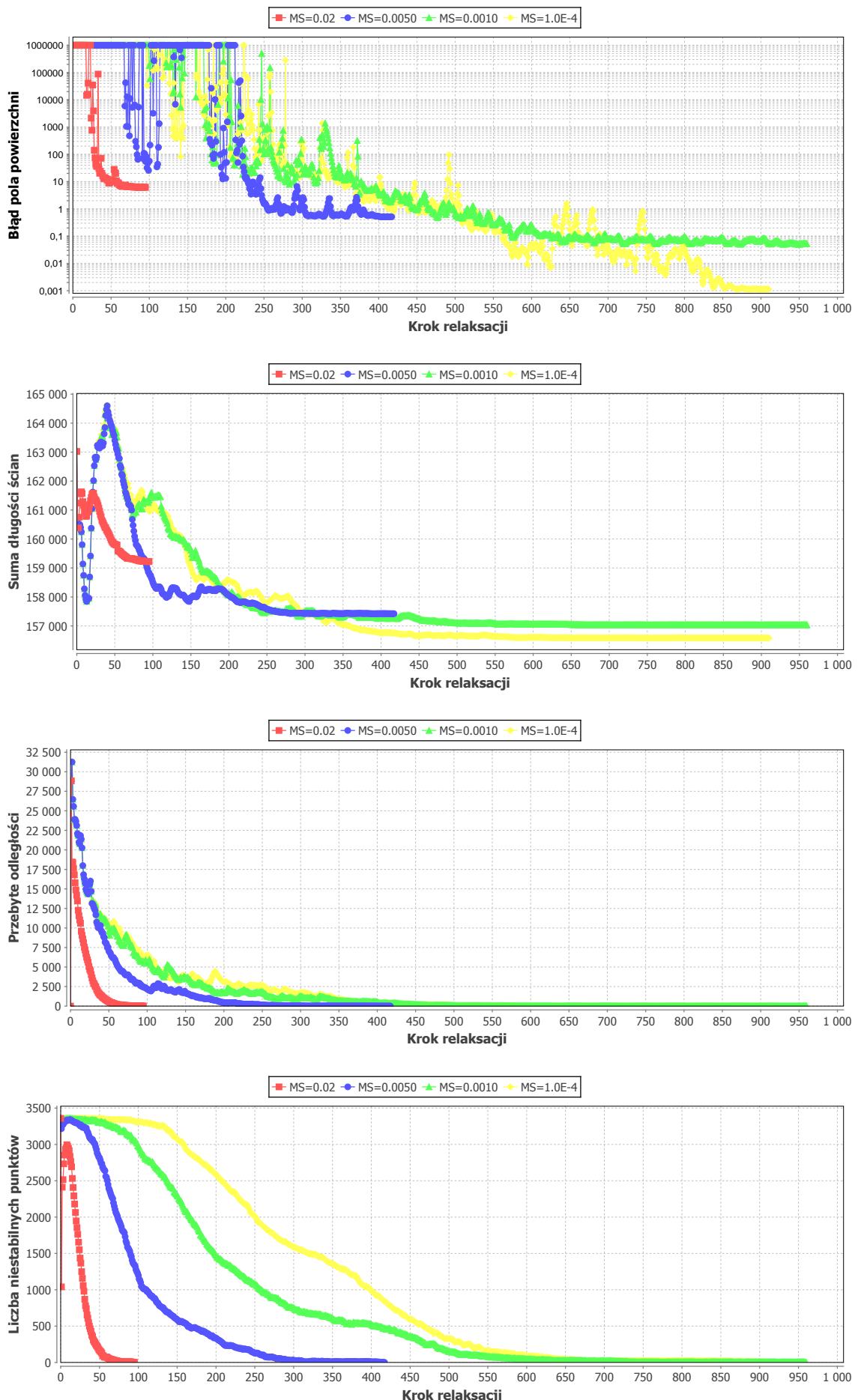
Rysunek 4.17: Wpływ parametru *speed limit* na relaksację dla struktury jednopoziomowej o 1680 liczbach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.18: Wpływ parametru *speed limit* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.19: Wpływ parametru *minimum speed* na relaksację dla struktury jednopoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.20: Wpływ parametru *minimum speed* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.

te będą mogły dłużej brać udział w optymalizacji stanu piany, co doprowadzi do znalezienia lepszego (pod względem błędów powierzchni i długości ścianek) stanu końcowego. I rzeczywiście — im mniejsza wartość parametru MS, tym lepsze własności stanu końcowego. Jednak, podobnie jak w przypadku zbioru jednopoziomowego, zbytnie obniżenie wartości parametru prowadzi do znacznie wolniejszej stabilizacji i kilkukrotnego wzrostu liczby kroków potrzebnych do zakończenia relaksacji. W badanym przypadku standardowa wartość 0,001 okazuje się zbyt mała, ponieważ skutkuje ponad dwukrotnie dłuższą relaksacją w stosunku do wartości 0,005, a polepszenie jakości stanu końcowego jest niewielkie.

4.4.3 Slowdown period

Ostatni z badanych parametrów to *slowdown period* (SP). Reguluje on zmniejszanie szybkości poruszania się wierzchołków (lub inaczej zmniejszanie wpływu sił w układzie na ruch) z upływem czasu. Dokładniej, wartość ta oznacza, po ilu iteracjach szybkość wierzchołków zostanie zmniejszona o połowę (szybkość jest zmniejszana wykładniczo). Standardowo parametr ten ma wartość 500. Wartość zero oznacza, że szybkość nie będzie zmniejszana. Wpływ tego parametru na przebieg relaksacji przedstawiają rys. 4.21 i 4.22.

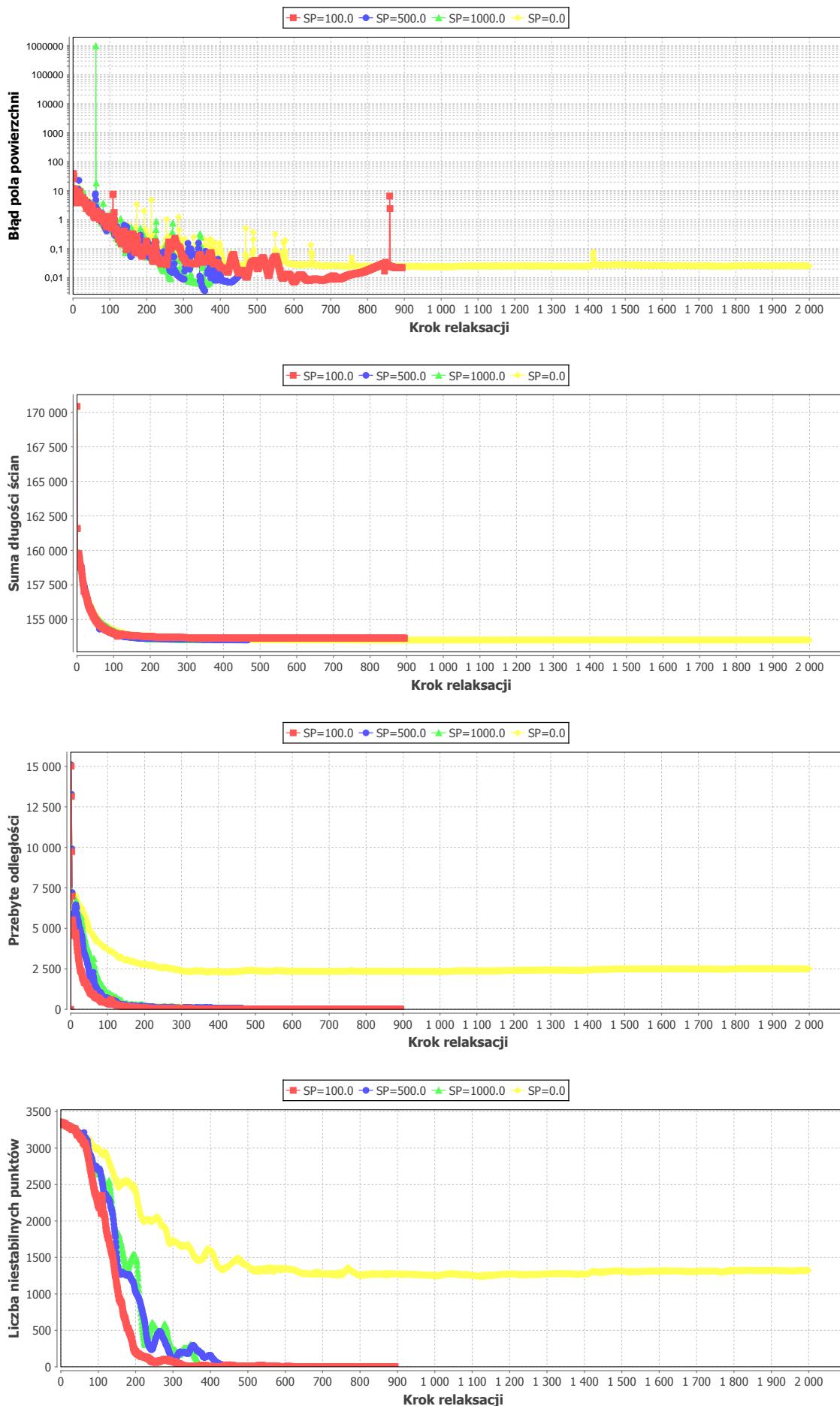
W obydwu przeprowadzonych testach wyłączenie tłumienia (czyli ustawienie parametru SP na zero) powoduje, że relaksacja nigdy się nie kończy (liczbę kroków przedstawionych na wykresach ograniczono do dwóch tysięcy). Wynika to stąd, że wiele wierzchołków zostaje „zablokowanych” w cyklicznym ruchu wokół swoich punktów równowagi. Widać więc, że tłumienie pełni bardzo ważną funkcję w procesie relaksacji.

Zbyt duże tłumienie również powoduje problemy, co widać szczególnie w przypadku testu na strukturze czteropoziomowej — stan końcowy otrzymany przy parametrze SP równym 100 ma bardzo duże błędy w polach powierzchni oraz niekorzystną długość ścianek. Wynika to głównie stąd, że wierzchołki zwalniają do prędkości poniżej progu stabilizacji, jeszcze zanim osiągną pozycje lokalnego minimum energetycznego. Obserwując relaksację w czasie rzeczywistym zauważono również, że przyczyną problemów często jest działanie mechanizmu redukcji ścian. Parametry tego mechanizmu szybko stają się nieadekwatne do spowolnionego tempa symulacji. Pojawiają się problemy podobne do tych, które występowały przy zbyt małej wartości parametru *squeeze candidate times*. Poza tym, redukcja pojedynczej ściany trwa coraz dłużej, więc coraz więcej ścian jest redukowanych jednocześnie, przez co zniekształcenia się kumulują. Biorąc pod uwagę również fakt, że przy dużym tłumieniu sił praktycznie w dowolnym momencie wierzchołki mogą zostać uznane za stabilne (podobny problem był powodowany przez zbyt dużą wartość parametru *minimum speed*), trudno spodziewać się uzyskania korzystnego stanu końcowego.

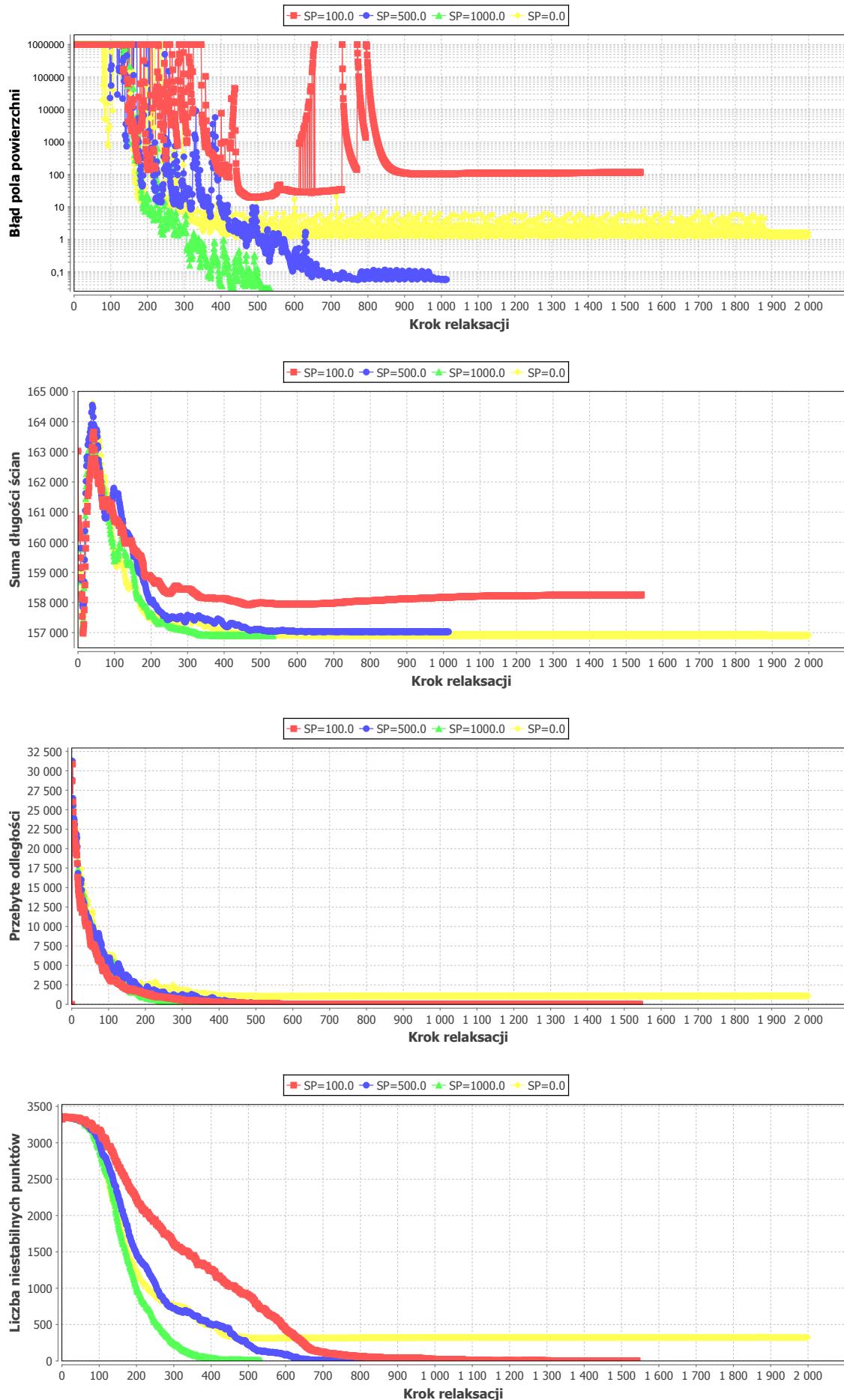
Poza opisanymi powyżej przypadkami ekstremalnymi, zmiana wartości parametru SP nie wpływa w dużym stopniu na przebieg relaksacji i jakość stanu końcowego. W przypadku struktury jednopoziomowej, praktycznie nie widać różnic w zachowaniu piany przy parametrze równym 500 i 1000. Dla struktury wielopoziomowej okazało się, że przy większej wartości badanego parametru relaksacja zakończyła się zdecydowanie szybciej. Warto więc rozważyć zwiększenie jego wartości domyślnej.

4.5 Podsumowanie

Przeprowadzone eksperymenty pozwoliły do pewnego stopnia upewnić się, czy wybrane standardeowe wartości parametrów są korzystne i w jaki sposób można je poprawić. Zaobserwowano też, jakie zjawiska zachodzą przy zmianie wartości parametrów, a wyjaśnienie przyczyn tych zjawisk doprowadziło do lepszego zrozumienia działania zaimplementowanych mechanizmów.



Rysunek 4.21: Wpływ parametru *slowdown period* na relaksację dla struktury jednopoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.



Rysunek 4.22: Wpływ parametru *slowdown period* na relaksację dla struktury czteropoziomowej o 1680 liściach, dla bąbelka-korzenia w kształcie kwadratu.

Rozdział 5

Wizualizacja zmian przy użyciu animacji

Po zaprojektowaniu i eksperymentalnym zbadaniu wizualizacji typu *foamtree* dla pojedynczych, statycznych struktur, rozpoczęto prace nad metodami przedstawiania zmian w czasie. Metody te sprawdzają się do animacji pokazującej przejście pomiędzy statycznymi wizualizacjami dwóch stanów (początkowego i końcowego), dla których zdefiniowano odwzorowanie pomiędzy odpowiadającymi sobie węzłami w pierwszym i drugim stanie. Założeniem projektowym było umożliwienie wizualizacji dowolnych zmian, czyli zarówno w wielkościach wizualizowanych elementów, jak i w strukturze ich hierarchicznych zależności. Zmiany strukturalne można podzielić na następujące kategorie:

- dodanie elementów — gdy w końcowej strukturze pojawiają się węzły, które nie mają swojego odpowiednika w strukturze początkowej. Szczególnym przypadkiem jest dodanie węzłów do rodzica, który w stanie początkowym był liściem. Sytuację taką można interpretować jako podział elementu na mniejsze części i warto ją wizualizować w odrębny sposób.
- usunięcie elementów — sytuacja odwrotna do pierwszego przypadku, węzłom w strukturze początkowej nie odpowiadają żadne węzły w stanie końcowym. Tu również można wyróżnić usunięcie wszystkich potomków pewnego elementu i uczynienie z niego liścia.
- przeniesienie elementów — gdy odpowiadające sobie węzły w strukturze początkowej i końcowej są potomkami węzłów, które sobie nie odpowiadają.

Złożenie wszystkich powyższych zmian pozwala na zdefiniowanie różnic pomiędzy dowolnymi dwiema strukturami (w najgorszym przypadku będzie to usunięcie wszystkich elementów struktury początkowej i zastąpienie ich nowymi elementami).

Istotnym zagadnieniem, które warto mieć na uwadze przy projektowaniu przejścia pomiędzy stanami jest to, czy kształt bąbelków w stanie początkowym powinien wpływać na wizualizację stanu końcowego. Z jednej strony, pożądane jest, aby wizualizacja danego stanu zawsze wyglądała tak samo, bez względu na to, jakie stany były oglądane wcześniej. Jest to szczególnie ważne, gdy użytkownik nie chce tylko przeglądać wszystkich stanów po kolej, ale też cofać się i przeskakiwać o kilka stanów na raz. Aby dobrze orientować się w swojej pozycji w czasie, powinien mieć wtedy możliwość łatwego rozpoznania stanów, które widział wcześniej. Z drugiej strony, jeśli wizualizacje w kolejnych stanach generowane są niezależnie, pozycje i kształty bąbelków mogą się znacznie różnić, nawet gdy zmiany w danych są niewielkie. Animacja przejścia pomiędzy stanami będzie w takim przypadku trudniejsza do wygenerowania i mniej czytelna. Dlatego w niektórych zastosowaniach korzystniejsze może być oparcie wizualizacji stanu końcowego na układzie widocznym w stanie początkowym.

5.1 Animacja „rozrywająca” pianę

Pierwszy z zaimplementowanych sposobów generowania animacji polega na „rozerwaniu” piany na poszczególne bąbelki i animowaniu każdego bąbelka z osobna. Stan początkowy i końcowy są dworzone niezależnie, poprzez ułożenie bąbelków zgodnie z algorytmem *treemap* i przeprowadzenie pełnej relaksacji. Następnie, każdy bąblek w stanie początkowym (na wszystkich poziomach hierarchii) zostaje „oderwany” od piany. Polega to na zastąpieniu wszystkich jego wierzchołków nowymi wierzchołkami, które nie będą jednocześnie należały do sąsiednich ani nadzędnych bąbelków. Później następuje przygotowanie animacji, która w sposób płynny zmieni położenie i kształt bąbelka do formy docelowej.

Przygotowanie animacji polega na zaplanowaniu, które wierzchołki mają się przesunąć na pozycje odpowiednich wierzchołków w bąbelku docelowym, które zostaną usunięte lub czy zostaną dodane nowe. Wykorzystano w tym celu algorytm wzorowany na algorytmie znajdującym odległość Levenshteina [Lev66]. Odległość ta jest oryginalnie zdefiniowana dla ciągów znaków i oznacza najmniejszą liczbę operacji edycji, jakie należy wykonać, aby zamienić jeden ciąg na inny, przy czym możliwe operacje to dodanie, usunięcie lub zamiana pojedynczego znaku. W rozpatrywanym problemie edycje wykonywane są na ciągu wierzchołków należących do bąbelka początkowego, tak aby dopasowały się do wierzchołków bąbelka końcowego. Nadal możliwe są edycje polegające na usunięciu lub dodaniu elementu, natomiast zamiana elementu jest teraz rozumiana jako przesunięcie wierzchołka na pozycję wierzchołka końcowego. Operacjom nadano wagę: dodanie i usunięcie wierzchołka ma wagę 1, a przesunięcie — wartość kąta (w radianach) pomiędzy kierunkiem przesuwanego i docelowego wierzchołka (gdzie kierunek wierzchołka oznacza kierunek od punktu centralnego w bąbelku zawierającym dany wierzchołek do punktu, w którym leży ten wierzchołek).

Dla tak zmodyfikowanego problemu można dostosować znany algorytm programowania dynamicznego, wyznaczający odległość Levenshteina, aby znaleźć transformację o najmniejszej łącznej wadze operacji edycji. Algorytm nie uwzględnia jednak tego, że wierzchołki bąbelka tworzą cykl, a nie ciąg o określonym początku i końcu. Należy więc wykonać go wielokrotnie, za każdym razem zmieniając wierzchołek, od którego zaczyna się opis bąbelka początkowego. Algorytm w podstawowej wersji ma złożoność $O(mn)$, gdzie m i n to liczby wierzchołków w bąbelkach, odpowiednio: początkowym i końcowym. Ostateczna złożoność algorytmu planowania animacji wynosi więc $O(m^2n)$. Nie jest to najszybszy algorytm dla tego problemu (lepszą propozycję można znaleźć między innymi w [PM02]), jednak jego złożoność jest wystarczająca, przy założeniu, że liczba wierzchołków w bąbelkach jest zwykle mała (nie więcej niż 8).

Na podstawie znalezionej planu transformacji bąbelka następuje wyznaczenie pozycji docelowych dla poszczególnych wierzchołków. Dla wierzchołka przesuwanego jest to oczywiście położenie wierzchołka docelowego. W przypadku wierzchołka przeznaczonego do usunięcia, jest to punkt na obwodzie bąbelka docelowego, który leży w tym samym kierunku względem jego środka, co pozycja początkowa względem środka bąbelka początkowego. Analogicznie jest traktowany wierzchołek dodany: na początku ustawiany jest na obwodzie bąbelka startowego, na pozycji o kierunku zgodnym z kierunkiem punktu docelowego.

Pozycje docelowe animowanych wierzchołków nie są pamiętane jako bezwzględne współrzędne w układzie prostokątnym, tylko w układzie *pseudo-biegunowym* (zob. rozdz. 3.1.6), przy *konturze zewnętrznym* zgodnym z kształtem bąbelka nadzędnego w układzie docelowym. Również aktualne pozycje wszystkich punktów po każdym kroku animacji są zapisywane w formie *pseudo-biegunowej* (z *konturem zewnętrznym* zgodnym z kształtem bąbelka-rodzica na danym etapie animacji). Wykorzystanie układu *pseudo-biegunowego* służy temu, by bąbelki należące do przodka, który się przesuwa lub przekształca, dopasowywały się do jego ruchu. Wykonanie kroku animacji dla danego wierzchołka

składa się więc z następujących czynności:

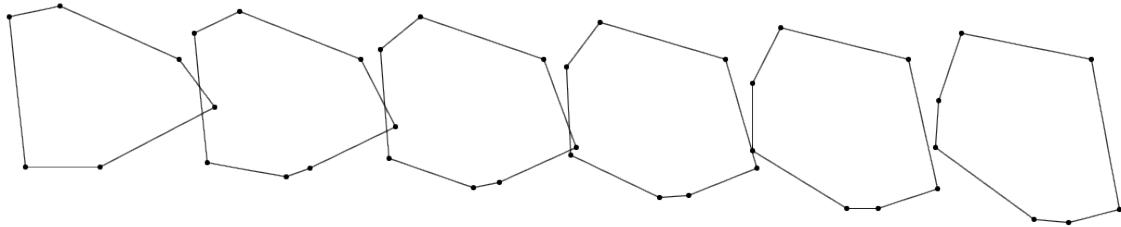
- odtworzenie aktualnej pozycji P_n z zapamiętanych współrzędnych *pseudo-biegunowych* (może być ona inna niż bezwzględna pozycja po wykonaniu poprzedniego kroku animacji, ponieważ kształt bąbelka nadrzednego, wykorzystywanego jako *kontur zewnętrzny*, również może się zmieniać w trakcie animacji),
- odtworzenie pozycji docelowej T_n ze współrzędnych *pseudo-biegunowych* (podobnie jak wcześniej, pozycja ta może być inna w każdym kroku animacji),
- obliczenie nowej pozycji wierzchołka, zgodnie ze wzorem:

$$P_{n+1} = \frac{(k - n)P_n + T_n}{k - n + 1}, \quad (5.1)$$

gdzie n jest aktualnym krokiem animacji, a k — liczbą wszystkich kroków ($k - n$ jest więc równe liczbie kroków pozostałych do wykonania),

- zapamiętanie pozycji P_{n+1} w postaci *pseudo-biegunowej*.

Przykład zmiany kształtu i położenia bąbelka w trakcie animacji został pokazany na rys. 5.1.



Rysunek 5.1: Kolejne kroki animacji zmieniającej kształt bąbelka, z dodaniem jednego wierzchołka.

Przedstawiony powyżej sposób animacji dotyczy bąbelków, które nie uczestniczą w żadnych zmianach strukturalnych, a co najwyżej zmieniają swoją wielkość. W przypadku pozostałych bąbelków należy wprowadzić pewne zmiany:

Bąbelki dodane do rodziców zawierających inne dzieci. Ponieważ nie istnieją one w stanie początkowym, na początku animacji znajdują się one skopiowane jako bąbelki o zerowym polu powierzchni. Każdy taki bąbelek ma tyle samo wierzchołków, co jego bąbelek docelowy. Wierzchołki ustawiane są na pozycji odpowiadającej środkowi bąbelka docelowego (ponownie wykorzystano układ *pseudo-biegunowy*, by dopasować pozycję do różnic w kształcie i położeniu bąbelka-rodzica). W trakcie animacji bąbelek rozszerza się do wymaganego kształtu.

Jeśli w dodawanym bąbelku w stanie docelowym znajdują się bąbelki potomne, znajdują się one również skopiowane do stanu początkowego. Dodawane bąbelki potomne mają po skopiowaniu strukturę bąbelków docelowych (przeskalowaną początkowo do zerowej powierzchni bąbelka-rodzica), a zatem pozostają w pianie jako spójna całość, bez „rozrywania” ich na bąbelki animowane niezależnie. Wierzchołki tych bąbelków w każdym kroku animacji są ustawiane tak, aby ich pozycja w bąbelku nadrzednym odpowiadała pozycji docelowej. Dzięki temu bąbelki są powiększane wraz z rodzicem, w sposób równomierny.

Bąbelki dodane do rodziców niezawierających innych dzieci. Jak już wcześniej wspomniano, ten przypadek powinien być traktowany w odmienny sposób. Zamiast niezależnego rozciągania każdego bąbelka od jednego punktu do docelowego kształtu, bardziej naturalnie wygląda, gdy

bąbelki od początku mają docelową wielkość, ale stają się widoczne stopniowo dzięki efektowi przezroczystości. Bąbelki ze stanu końcowego są przed animacją kopiowane do bąbelka odpowiadającego ich rodzicowi i odpowiednio przekształcane przy pomocy *pseudo-biegunkowego* układu współrzędnych. Na początku animacji są zupełnie przezroczyste, ale w kolejnych krokach coraz bardziej przesłaniają bąbelka-rodzica. Podobnie jak opisane wcześniej bąbelki potomne dodanego bąbelka, nie są one animowane, a jedynie pozycje ich wierzchołków są odświeżane zgodnie z kształtem rodzica.

Bąbelki usunięte z rodziców zawierających inne dzieci. Animacja usuwanych bąbelków polega na ich stopniowym zmniejszaniu poprzez zbliżanie wszystkich wierzchołków do jednego punktu, odpowiadającego punktowi centralnemu bąbelka w stanie początkowym. Zastosowano tu te same mechanizmy, co w przypadku efektu rozciągania dodawanych bąbelków.

Bąbelki usunięte z rodziców, które nie mają więcej dzieci. Tutaj również zastosowano efekt przezroczystości — bąbelki na początku są w pełni widoczne, a wraz z postępem animacji stają się przezroczyste i odsłaniają swojego rodzica.

Bąbelki zmieniające rodzica. Ponieważ bąbelki te nie mają tych samych rodziców w stanie początkowym i końcowym, do określania współrzędnych *pseudo-biegunkowych* ich wierzchołków musi być wykorzystany inny kształt. W tym celu zostaje wyszukany bąbelek położony najniżej w hierarchii, który jest przodkiem przenoszonego bąbelka w stanie początkowym i którego odpowiednik w stanie końcowym jest przodkiem bąbelka docelowego. W ekstremalnym przypadku będzie to korzeń, który z założenia występuje w hierarchii w każdym momencie czasu.

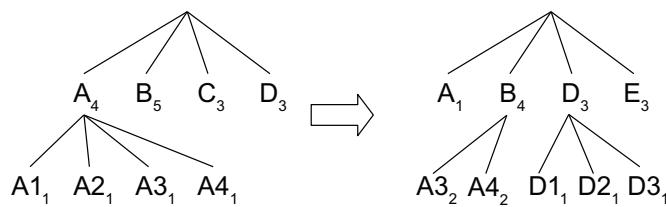
Aby dodatkowo podkreślić w wizualizacji fakt, że dany bąbelek zostaje przeniesiony do innego rodzica, wykorzystano przezroczystość. Na początku animacji przenoszony bąbelek jest całkowicie nieprzezroczysty. W kolejnych krokach przezroczystość zwiększa się, aż osiągnie określony poziom po wykonaniu połowy kroków animacji, po czym zaczyna się z powrotem zmniejszać. Zabieg ten sprawia, że bąbelki przenoszone na większe odległości nie zakrywają całkowicie fragmentów piany, nad którymi się znajdują.

Dodatkowej uwagi wymagają przenoszone bąbelki, które w stanie końcowym są potomkami pewnego dodawanego bąbelka. Zgodnie z wcześniejszym opisem, zostają one skopiowane i wyświetcone w animacji z efektem rozciągania lub stopniowego zmniejszania przezroczystości. Aby uniknąć przedstawiania dwóch wersji tego samego bąbelka, odpowiedni potomek dodawanego bąbelka musi zostać usunięty. W wizualizacji będzie to widoczne jako puste miejsce, do którego trafia właściwy bąbelek. Analogicznie trzeba postąpić w przypadku bąbelków, które w stanie początkowym są potomkami pewnego usuwanego bąbelka — w animacji zmniejszania lub ukrywania należy usunąć odpowiednik przenoszonego bąbelka.

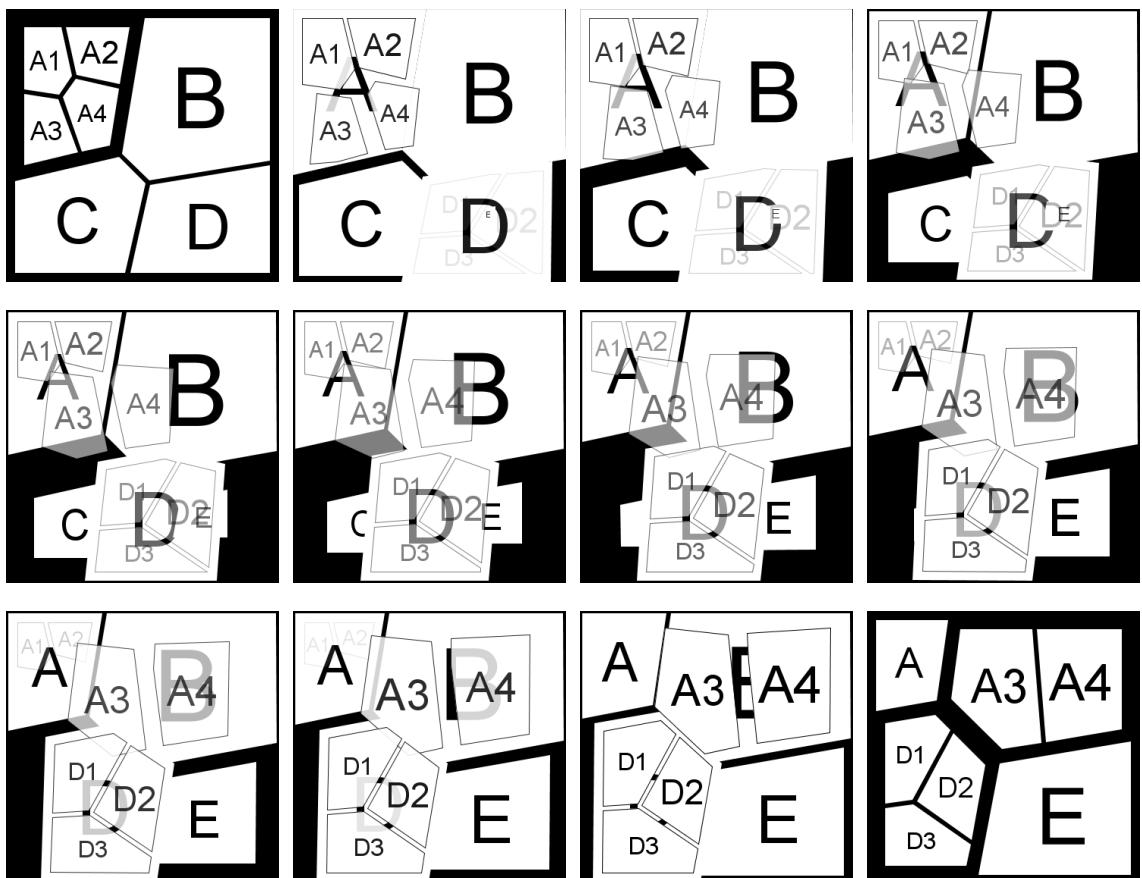
Aby zaprezentować działanie zaprojektowanych animacji, stworzono dwie przykładowe struktury danych, pomiędzy którymi istnieje kilka różnic (zob. rys. 5.2). Wizualizacja tych zmian z wykorzystaniem animacji „rozrywającej” została przedstawiona na rys. 5.3. Aby uprościć przekaz, w pierwszych przykładowych animacjach zrezygnowano z kolorów. Wersje z kolorami zostaną przedstawione w rozdz. 5.4.

5.2 Animacja „ciągła” — odtwarzanie przebiegu relaksacji

W opisany powyżej sposobie przedstawiania zmian w czasie dość istotną wadą jest to, że w trakcie animacji wizualizacja przestaje przypominać pianę, stając się zbiorem niezależnych wielokątów.



Rysunek 5.2: Struktury wykorzystane jako stan początkowy i końcowy w przykładowych animacjach (liczby w indeksach dolnych oznaczają wielkości elementów).



Rysunek 5.3: Kolejne ramki animacji „rozrywającej” — przejście pomiędzy stanami z rys. 5.2.

Postanowiono więc opracować metodę, w której struktura piany zostanie cały czas zachowana. Rozwiązanie to będzie podobne do przedstawienia kolejnych kroków procesu relaksacji. Niestety, przebieg relaksacji nie nadaje się do bezpośredniego zaprezentowania użytkownikowi. Po pierwsze dla tego, że nie jest możliwa kontrola nad tempem relaksacji. Nie tylko nie da się zaplanować ile kroków zajmie relaksacja (ten problem dałoby się rozwiązać przez zapamiętanie wszystkich kroków i odwracanie tylko niektórych z nich), ale różne fragmenty piany stabilizują się w różnym czasie. Po drugie, mechanizm redukcji ścianek pogarszających kształty bąbelków (zob. rozdz. 3.1.2) często powoduje nienaturalnie wyglądające deformacje piany, takie jak wielokrotne skracanie i ponowne wydłużanie tej samej ścianki.

Podejmowano próby implementacji mechanizmu, który pozwalałby tak rozplanować przesunięcia wierzchołków i zmiany strukturalne w pianie, aby bąbelki przyjmowały zadane pozycje i kształty. Nie udało się jednak uzyskać zadowalających rezultatów. Nie zawsze udawało się uzyskać docelowy stan,

a nawet jeśli, to bąbelki przesuwane na większe odległości zwykle były mocno zniekształcone, a cała wizualizacja stawała się nieczytelna.

Zaproponowane rozwiązań powyższych problemów polega na przeprowadzeniu relaksacji i zapisaniu najważniejszych informacji o zmianach, które zaszły. Informacje te są później wykorzystane do pokazania animacji odtwarzającej cały proces w uproszczony sposób. Animacja taka pozwoli wprowadzić zmiany w pianie bez przerwania jej ciągłości. Nie dotyczy to jednak zmian, które w animacji „rozrywającej” zostały przedstawione z wykorzystaniem efektu przezroczystości, a więc przeniesienia bąbelka do innego rodzica, dodania nowych węzłów do liścia oraz usunięcia wszystkich dzieci z węzła. Te zmiany będą przedstawiane w taki sam sposób jak w pierwszej metodzie.

Przed rozpoczęciem relaksacji należy wprowadzić w pianie odpowiednie zmiany (punktrem wyjścia jest piana utworzona na podstawie stanu początkowego, dla której przeprowadzono pełną relaksację):

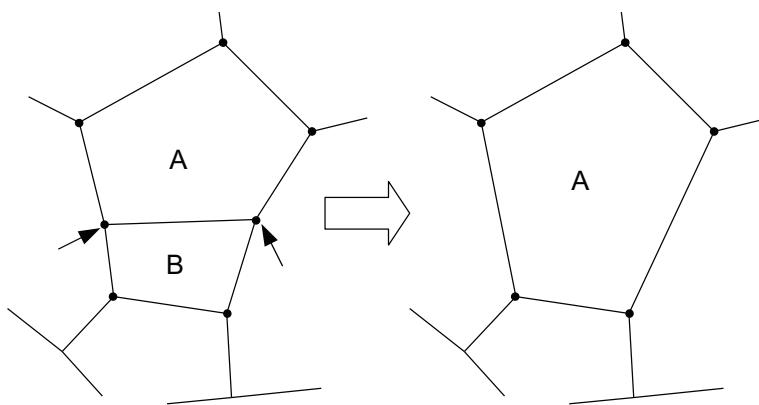
- zmienić oczekiwane pole powierzchni w bąbelkach, które powinny zmienić swój rozmiar,
- w bąbelkach, które mają zostać usunięte lub przeniesione do innego rodzica, ustawić oczekiwane pole powierzchni na zero oraz oznaczyć je jako przeznaczone do usunięcia,
- stworzyć bąbelki, które powinny zostać dodane lub przeniesione z innego rodzica. Bąbelki te powinny mieć na początku zerowe pole powierzchni, ale utrudniałoby to stworzenie dla nich początkowej struktury (nie można wykorzystać algorytmu *treemap*). Dodanie bąbelków zaczyna się więc od stworzenia dodatkowego, trójkątnego bąbelka w jednym z narożników docelowego bąbelka-rodzica. Bąbelek ten ma bardzo małe, ale niezerowe pole powierzchni (jego krawędzie mają około jednej setnej długości krawędzi rodzica). Następnie, wewnątrz tego bąbelka zastają stworzone wraz z potomkami wszystkie bąbelki, które mają być dodane lub przeniesione do rodzica, (jest to ta sama operacja, którą wykorzystuje się do utworzeniu układu początkowego w statycznej wizualizacji, czyli zaplanowanie układu przy pomocy algorytmu *treemap* i dopasowanie go do kształtu docelowego bąbelka — zob. rozdz. 3.1.5). Na końcu, stworzone bąbelki zostają przeniesione poziom wyżej (czyli stają się dziećmi docelowego rodzica), a dodatkowy bąbelek stworzony na początku zostaje usunięty.

W tak przygotowanej pianie wykonywany jest proces relaksacji. Po każdym kroku następuje sprawdzenie, czy któreś z bąbelków przeznaczonych do usunięcia zmieniły swój stan na *stłoczony* (zob. rozdz. 3.1.7). Jeśli tak, następuje próba ich usunięcia, ponieważ są już na tyle małe, że ich dalsza relaksacja prawdopodobnie nie ma sensu. Jeśli po ustabilizowaniu piany nadal istnieją bąbelki przeznaczone do usunięcia, które nie zostały *stłoczone*, zostają one również usunięte, a relaksacja jest kontynuowana w poszukiwaniu nowego stanu stabilnego.

Operacja usunięcia bąbelka została przedstawiona na rys. 5.4. Rozpoczyna się od znalezienia sąsiedniego bąbelka, z którym usuwany bąbelek wspólnie dzieli swoją najdłuższą ściankę. Wierzchołki na końcach tej ścianki zostają usunięte z piany, a pozostałe wierzchołki usuwanego bąbelka zostają włączone do bąbelka sąsiedniego.

Podczas relaksacji rejestrowane są następujące rodzaje zdarzeń:

- transformacja T1 (zob. rozdz. 3.1.3): zostaje zapamiętana para punktów, które wzięły udział w transformacji,
- usunięcie bąbelka: zostaje zapamiętane, który bąbelek i które dwa wierzchołki zostały usunięte.
- przeskoczenie wierzchołka leżącego na krawędzi na sąsiednią krawędź: zostaje zapamiętane, który to wierzchołek i w jakim kierunku względem środka bąbelka nadzawanego się znajdował (nie zostają zapamiętane dokładne krawędzie, ponieważ podczas odtwarzania bąbelek nadzawany może mieć inny kształt),



Rysunek 5.4: Przykład usunięcia bąbelka: bąbelek B zostaje usunięty, a jego powierzchnia włączona do bąbelka A.

Dla każdego wierzchołka jest przechowywana lista zdarzeń, w których brał udział. Informacje o zdarzeniach dotyczących dwóch wierzchołków są dodawane do obu powiązanych list.

Zdarzenie związane z transformacją T1 jest o tyle specyficzne, że jeśli transformacja zajdzie dwa razy pod rząd na tej samej parze punktów, struktura piany pozostanie niezmieniona. Dlatego, przed zapamiętaniem informacji o takim zdarzeniu następuje sprawdzenie, czy transformowane wierzchołki nie brały ostatnio udziału w takiej samej transformacji. Jeśli tak, to nowe zdarzenie nie zostaje zarejestrowane, a ostatnie zdarzenie z list obu wierzchołków zostaje usunięte.¹ Dzięki temu, podczas odtwarzania przebiegu relaksacji, transformacje, które nie miały wpływu na ostateczny kształt piany, nie zostają pokazane.

Po zakończeniu relaksacji zostają zapamiętane pozycje docelowe wszystkich wierzchołków, a układ piany zostaje przywrócony do stanu tuż przed jej rozpoczęciem. Zanim będzie możliwe odtworzenie zarejestrowanych zdarzeń w postaci animacji, należy rozplanować dla każdego zdarzenia, w jakim przedziale czasu ma być odtworzone. Przyjęto, że zdarzenia zachodzące na wierzchołkach należących do jednego bąbelka nadzawanego będą stanowiły oddzielne grupy, odtwarzane niezależnie od pozostałych. W każdej grupie zdarzeniom zostają przypisane kroki, w których powinny być odtworzone, tak aby ich kolejność odpowiadała kolejności rejestracji zdarzeń, a odstępy pomiędzy nimi były równe.

Animacja odtwarzająca przebieg relaksacji polega na tym, że w każdym kroku każdy wierzchołek zostaje przesunięty w kierunku określonym na podstawie najbliższego przypisanego mu zdarzenia i tego, ile kroków zostało do jego wykonania. Pozycja, w kierunku której przesuwany jest wierzchołek, ustalana jest następująco:

- dla transformacji T1: punkt leżący pośrodku między dwoma wierzchołkami, na których ma być wykonana transformacja,
- dla usunięcia bąbelka: punkt środkowy usuwanego bąbelka — dzięki temu usuwana ściana będzie przesuwała się w kierunku bąbelka, zmniejszając jego powierzchnię przed usunięciem,
- dla wierzchołka zmieniającego krawędź: punkt leżący na krawędzi bąbelka nadzawanego, leżący zgodnie z zapisanym wcześniej kierunkiem względem punktu centralnego.

Mając określoną pozycję docelową i liczbę pozostałych kroków animacji do wykonania zdarzenia, wierzchołek jest przesuwany zgodnie z metodą opisaną przez wzór 5.1. Wyjątek stanowią wierzchołki

¹W pewnych warunkach może się zdarzyć, że dwukrotne wykonanie T1 na tych samych wierzchołkach powoduje zamianę tych dwóch wierzchołków miejscami — w takim przypadku ostatnie zdarzenie nie zostaje usunięte, tylko odpowiednio oznaczone, aby podczas odtwarzania uzyskać taki sam efekt.

leżące na krawędzi bąbelka nadrzędnego — należy uwzględnić, że mogą poruszać się tylko wzdłuż krawędzi. W tym przypadku, do oszacowania kierunku ruchu oraz dystansu, który muszą przebyć wzdłuż obwodu, porównuje się kierunek wierzchołka i pozycji docelowej względem środka bąbelka nadrzędnego (czyli ich pierwszą wspólną wąską wąską w układzie *pseudo-biegunkowym*).

Gdy animacja dojdzie do kroku, w którym powinno zajść zdarzenie, wykonywana jest odpowiednia operacja — transformacja T1 lub usunięcie bąbelka. Zdarzenie zmiany krawędzi przez wierzchołek nie jest wywoływanie bezpośrednio, ponieważ wierzchołki leżące na krawędzi są przenoszone na kolejne krawędzie automatycznie podczas ruchu. Wierzchołki po kolei „odgrywają” zdarzenia z przypisanej im listy. Po wykonaniu ostatniego zdarzenia, wierzchołek zaczyna być przesuwany w kierunku ostatecznej pozycji, zapamiętanej po zakończeniu relaksacji. Tępo ruchu jest dobrane tak, aby docelowy punkt został osiągnięty w ostatnim kroku animacji.

Podobnie jak w przypadku animacji „rozrywającej”, bieżące pozycje wierzchołków są zapamiętywane w układzie *pseudo-biegunkowym* tak, aby przesunięcia i zmiany kształtów bąbelków nadrzędnych były odwzorowane również na bąbelkach potomnych.

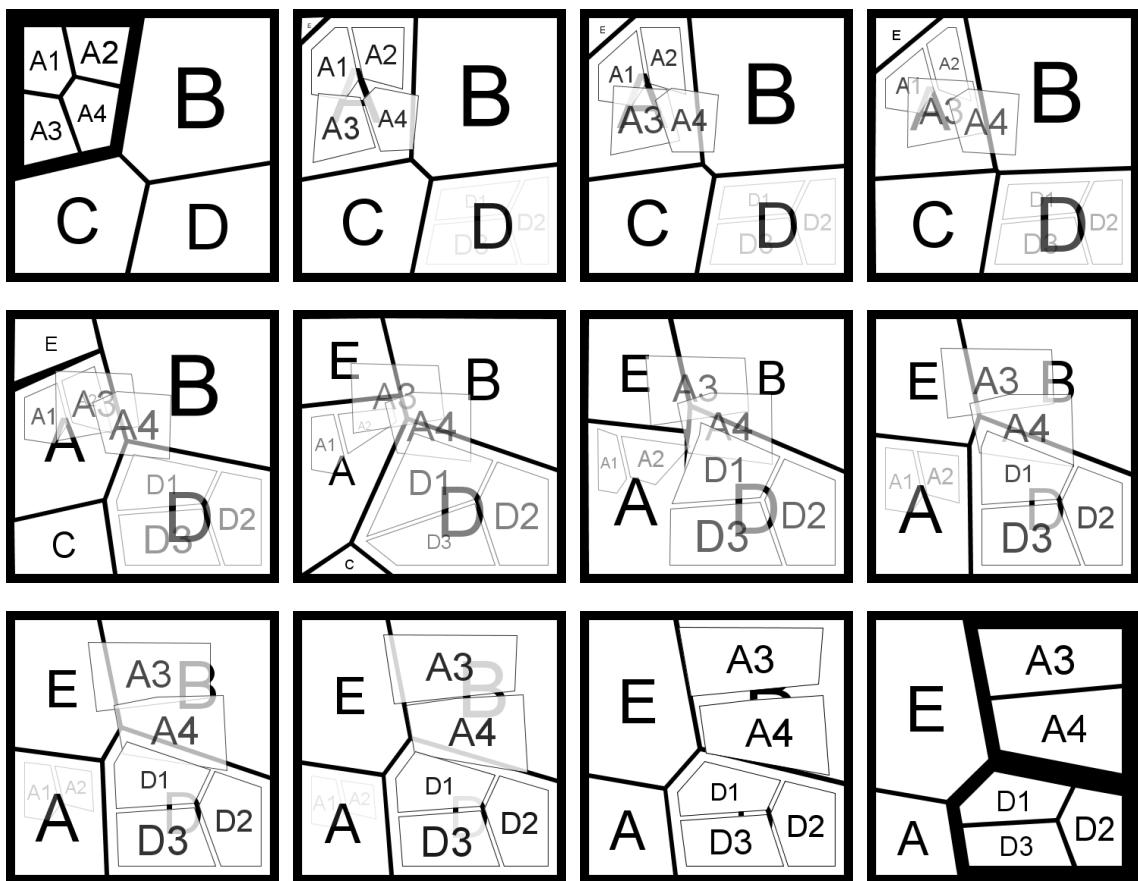
Ostatni z aspektów opisywanej animacji dotyczy bąbelków przenoszonych do innego rodzica. Jak wynika z opisu przygotowania poczatkowej piany do relaksacji, bąbelek taki istnieje w pianie w dwóch wersjach: jeden na pozycji poczatkowej, który jest zmniejszany i w końcu usuwany, oraz drugi na pozycji docelowej, który na początku ma bardzo małą powierzchnię i jest stopniowo powiększany do docelowych rozmiarów. Żadna z tych wersji nie może być wykorzystana do animacji przeniesienia bąbelka z efektem przezroczystości — muszą one pozostać w pianie, aby zachować jej poprawny stan i umożliwić wykonywanie niezbędnych transformacji strukturalnych. Dlatego do animacji zostaje stworzony osobny bąbelek, „oderwany” od reszty piany. Istniejące w pianie bąbelki zostają natomiast oznaczone przed rozpoczęciem animacji jako „atrapy”, dzięki czemu nie są wyświetlane (są widoczne jako puste miejsce).

Opisane powyżej mechanizmy stanowią jedynie podstawę dla projektu animacji i nie rozwiązuje wszystkich problemów, które mogą się pojawić. Jest na przykład możliwa sytuacja, w której zdarzenia zaplanowanego w danym kroku nie można odtworzyć, ponieważ pozycje wierzchołków w bąbelku nadrzędnym są inne niż w czasie rejestracji zdarzenia (wierzchołki, które powinny ulec transformacji T1, mogą sąsiadować z bąbelkiem o trzech wierzchołkach, przez co powtórzenie transformacji doprowadzioby do utworzenia niepoprawnego bąbelka o dwóch wierzchołkach). Również poprawne rozplanowanie zdarzeń w zasie i przesuwanie wierzchołków stanowi pewną trudność — zaproponowane podejścia są bardzo proste, przez co rezultaty bywają niezadowalające. Wszystko to sprawia, że przy próbie wizualizacji dużej liczby zmian zaimplementowany algorytm staje się niestabilny, a używane stany pośrednie często wyglądają na zniekształcone, co będzie można zobaczyć w następnym podrozdziale (rys. 5.8).

Przykład działania animacji „ciąglej” został przedstawiony na rys. 5.5.

5.3 Animacja „rozrywająca” ze stanem końcowym zależnym od poczatkowego

Postanowiono wypróbować jeszcze jeden rodzaj animacji, łączący cechy dwóch poprzednich. Jest to animacja „rozrywająca” pianę na niezależne bąbelki, w której stan końcowy jest wyznaczany w taki sam sposób jak w animacji „ciąglej”, czyli na zasadzie zmodyfikowania stanu poczatkowego i kontynuowania relaksacji. Dodatkowa zmiana polega na tym, że bąbelki, które zostają dodane z animacją rozciągania, nie są na początku ustawiane w punkcie środkowym docelowego kształtu, tylko tam, gdzie zostały dodane w pomniejszonej formie przed relaksacją. Podobnie bąbelki usuwane, zmniejszane w animacji do zerowego pola powierzchni — pozycja, do której zbliżane są wszystkie wierzchołki, to położenie bąbelka tuż przed jego usunięciem w trakcie relaksacji.



Rysunek 5.5: Kolejne ramki animacji „ciąglej” — przejście pomiędzy stanami z rys. 5.2.

Można powiedzieć, że metoda ta łączy jedynie wady dwóch metod, na których bazuje, to znaczy nie zapewnia ani ciągłości piany w trakcie animacji ani niezależności stanu początkowego i końcowego. Implementacja metody była jednak bardzo prosta, a w niektórych zastosowaniach może się okazać przydatna.

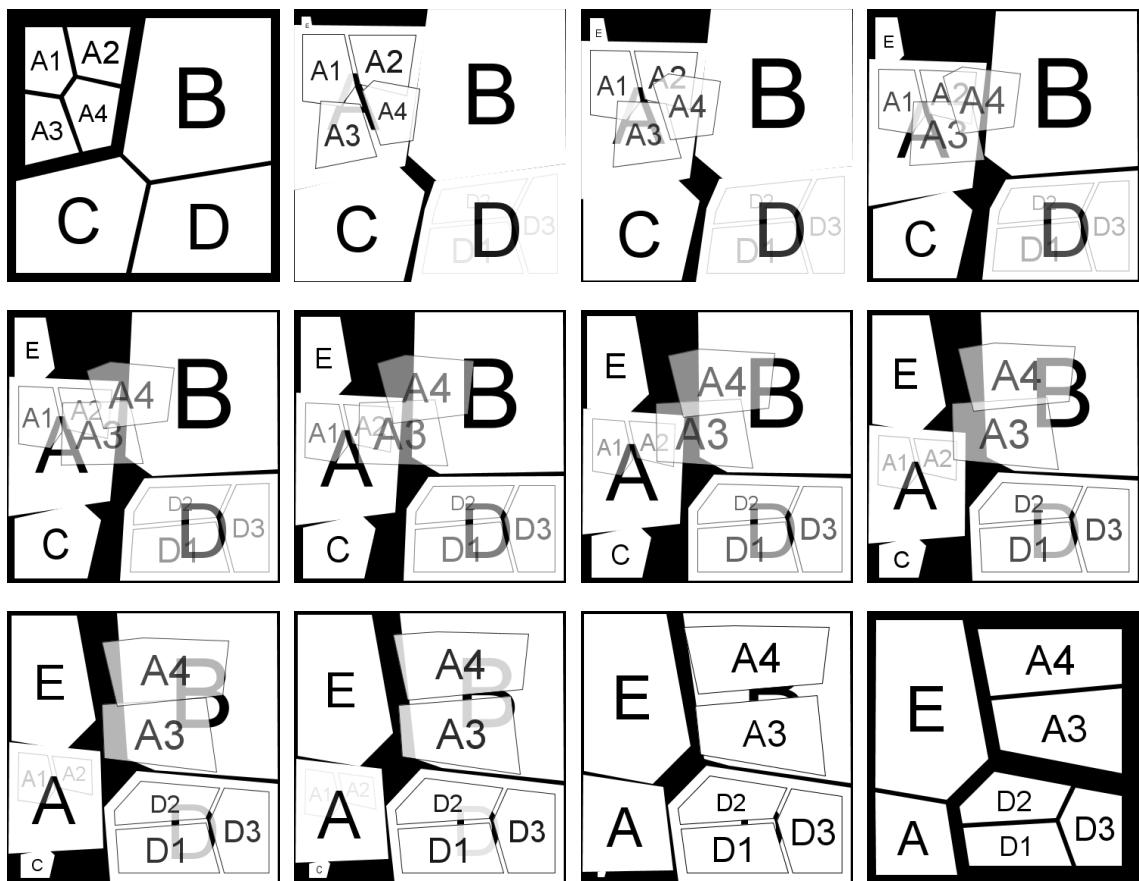
Przykład działania tej animacji został przedstawiony na rys. 5.6.

5.4 Zastosowanie animacji w problemach praktycznych

Działanie każdego z zaprezentowanych typów animacji zostało przedstawione na niewielkim, szczególnie przygotowanym zbiorze danych. Miało to na celu czytelne pokazanie sposobu działania animacji i podkreślenie ich najważniejszych właściwości. Warto jednak sprawdzić ich działanie również dla mniej „przyjaznych” danych, z którymi można się spotkać w rzeczywistych zastosowaniach.

Postanowiono wykorzystać w tym celu wyniki uzyskane z wyszukiwarki Carrot². System ten wyszukuje z różnych źródeł w Internecie dokumenty pasujące do podanego zapytania, a następnie grupuje je według tematów, wykorzystując najczęściej występujące w nich frazy. Dostępne w systemie algorytmy grupujące posiadają po kilka parametrów, które można modyfikować. Można więc przyjąć, że wartość jednego lub więcej parametrów zmienia się w czasie i wizualizować pojawiające się różnice w wynikach za pomocą animacji.

Przed zastosowaniem animacji należy rozwiązać problem, który prawdopodobnie pojawi się również przy próbie wykorzystania animacji w innych dziedzinach — przyporządkowanie do siebie węzłów w strukturach odpowiadających kolejnym momentom. W przypadku pogrupowanych wyników



Rysunek 5.6: Kolejne ramki animacji „rozrywającej”, ze stanem końcowym zależnym od początkowego — przejście pomiędzy stanami z rys. 5.2.

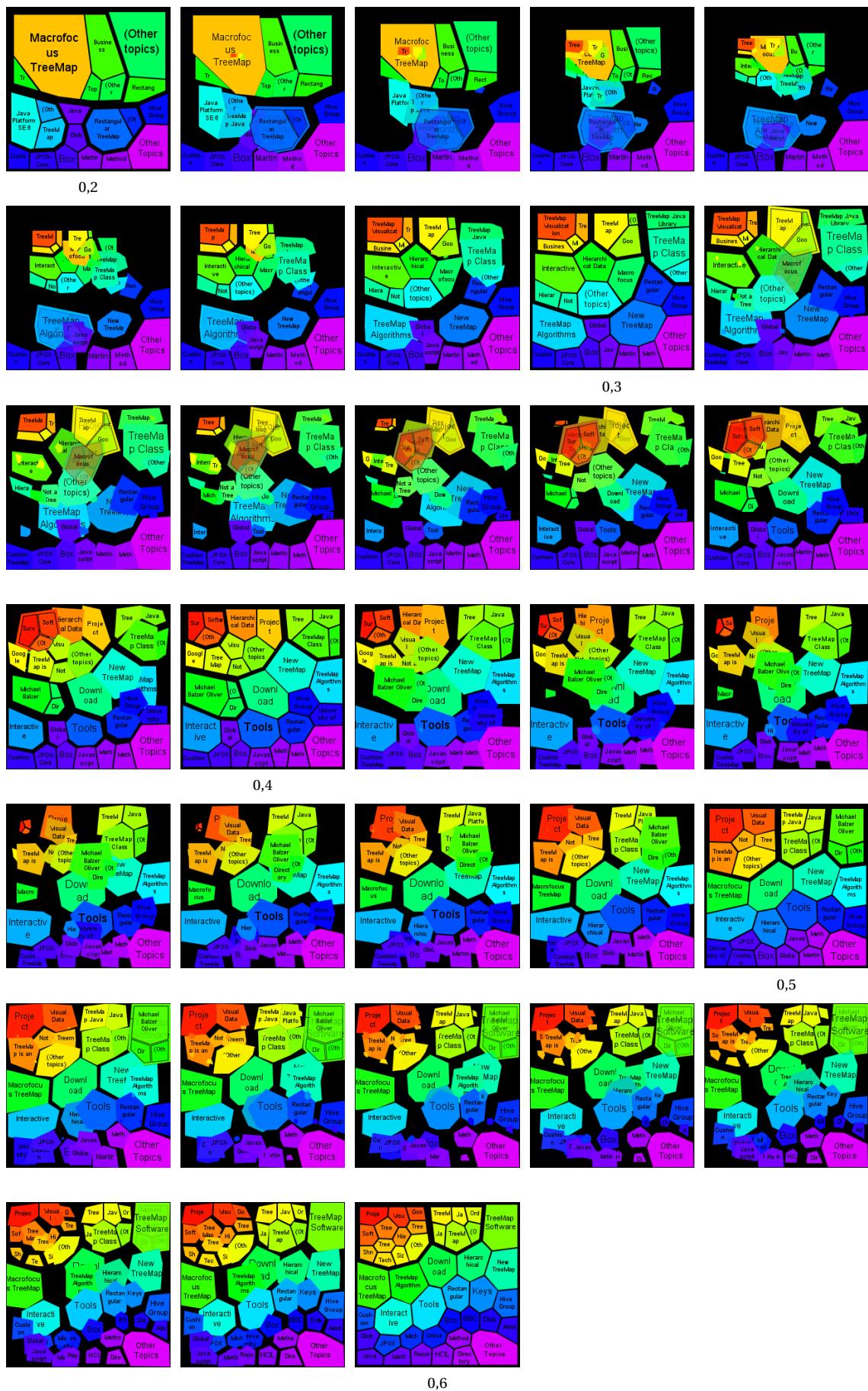
wyszukiwania oznacza to decyzję, które grupy tematyczne w odpowiedziach na zapytanie uzyskanych dla różnych wartości parametrów można uznać za tę samą grupę. Prosta identyfikacja grupy po przypisanej do niej frazie nie wystarczy, ponieważ nawet w pojedynczym wyniku może pojawić się wiele grup o tej samej frazie (w szczególności, w wielu grupach zostaje dodana podgrupa zatytułowana „*Other documents*”, do której trafiają dokumenty nie pasujące do reszty podgrup). Można zaprojektować rozwiązanie polegające na zliczaniu dokumentów współwystępujących w poszczególnych grupach, ale byłoby to podejście skomplikowane i wymagające głębszej analizy. Na potrzeby przetestowania animacji przyjęto, że odpowiadające sobie grupy muszą mieć przypisaną tę samą frazę oraz ich ścieżka do korzenia musi składać się z takich samych grup (fraz).

Wizualizowane dane dotyczą wyników wyszukiwania dla frazy „treemap”. Wyniki pogrupowano przy pomocy algorytmu *Lingo3G*. Kolejne stany uzyskano dzięki modyfikacji parametru *merge threshold*, który kontroluje to, jak bardzo grupy muszą się ze sobą pokrywać, aby zostały scalone do jednej. Im większa wartość tego parametru, tym więcej grup pozostanie nescalonych, co widać na końcowych wizualizacjach. Zastosowano wartości od 0,2 do 0,6 z krokiem co 0,1 — uzyskano więc 5 stanów. Animacje przejścia pomiędzy kolejnymi stanami złożono z siedmiu klatek pośrednich. Rezultat został przedstawiony na rys. 5.7 i 5.8.

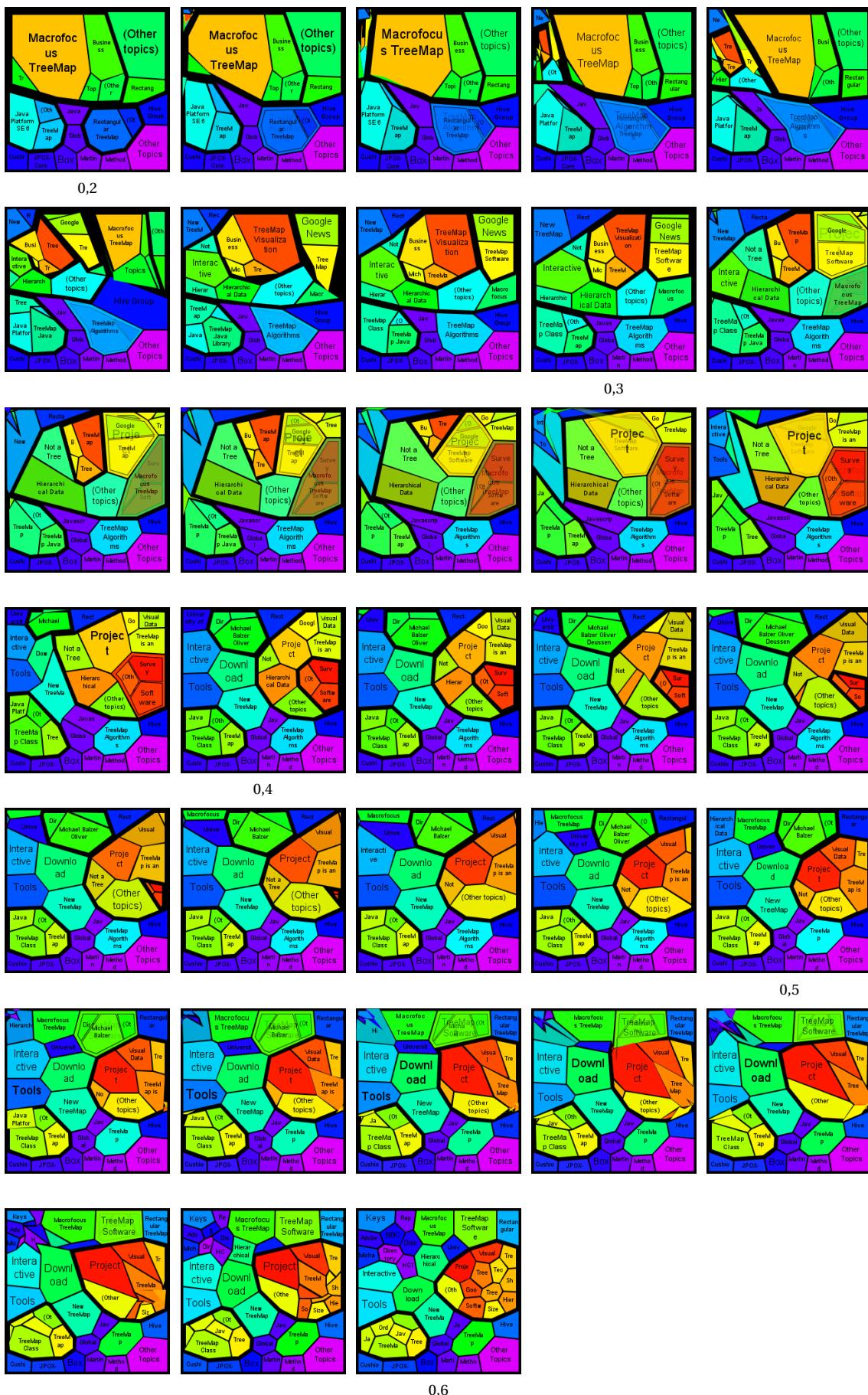
5.5 Podsumowanie

Udało się zaprojektować i zaimplementować trzy metody wizualizacji zmian w czasie. Wybranie najlepszej metody, którą warto wykorzystać w danym przypadku, wymaga rozważenia różnych aspektów. Jeśli najważniejsze jest konsekwentne przedstawianie stanów zawsze w taki sam sposób i zachowanie oczekiwanej kolejności elementów w wizualizacji, należy wybrać animację „rozrywającą” o niezależnych stanach. Jeśli jednak istotniejsze jest, by podczas przejścia pomiędzy stanami bąbelki nie przemieszczały się na duże odległości, wtedy warto zastanowić się nad pozostałymi animacjami. W takim przypadku animacja „ciągła” wydaje się atrakcyjniejsza, ponieważ wizualizacja cały czas przypomina bąbelki w pianie. Z drugiej strony, jest to w pewnym sensie ograniczenie, które sprawia, że animacja może nie być w pełni adekwatna do prezentowanych zmian — na przykład bąbelek, który powinien się zmniejszyć, może przez dłuższy czas pozostawać tej samej wielkości, lub nawet się powiększyć, zanim w końcu osiągnie docelowy kształt.

Stworzone wizualizacje są zdaniem autora atrakcyjne i interesujące wizualnie. Trudno jednak ocenić ich praktyczną przydatność — czy ułatwiają zrozumienie wizualizowanych zmian i pozwalają spostrzec zależności, które byłyby trudniejsze do wyłapania z innymi metodami wizualizacji.



Rysunek 5.7: Kolejne ramki animacji „rozrywającej”, pokazującej zmiany w wynikach wyszukiwania frazy „treemap” w systemie Carrot². Podpisy pod ramkami oznaczają wartość parametru *merge threshold*. Ramki bez podpisu przedstawiają stany przejściowe (w kolejności od lewej do prawej).



Rysunek 5.8: Kolejne ramki animacji „ciągły”, pokazującej zmiany w wynikach wyszukiwania frazy „treemap” w systemie Carrot². Podpisy pod ramkami oznaczają wartość parametru *merge threshold*. Ramki bez podpisu przedstawiają stany przejściowe (w kolejności od lewej do prawej).

Rozdział 6

Zakończenie

W niniejszej pracy dokonano przeglądu istniejących metod wizualizacji hierarchicznych struktur danych i wizualizacji zmian w czasie. Zaprojektowano wizualizację *foamtree* — nowe podejście do przedstawiania struktury drzewiastej, wzorowane na metodzie *treemap* i wykorzystujące uproszczoną heurystykę modelującą zachowanie dwuwymiarowej piany. Następnie zaimplementowano ten model i eksperymentalnie zbadano jego właściwości. Ostatnim etapem było rozwinięcie wizualizacji tak, aby pozwalała na animowane przejście pomiędzy różnymi wersjami hierarchicznych struktur, co umożliwiło wizualizację zmian w czasie. Zaimplementowane mechanizmy wykorzystano do przedstawienia zmian w wynikach wyszukiwania w systemie Carrot² dla różnych wartości parametrów.

6.1 Uwagi na temat implementacji

Cała biblioteka pozwalająca na tworzenie wizualizacji typu *foamtree* oraz przykładowa aplikacja ją wykorzystująca zostały zaimplementowane w języku Java. Stworzony kod znajduje się na płycie CD-ROM dołączonej do pracy. Jest to projekt środowiska Eclipse, z przykładowymi konfiguracjami uruchomieniowymi.

Warto wspomnieć o czasach generowania wizualizacji. Testy przeprowadzono na komputerze wyposażonym w procesor AMD Turion X2 2.3 GHz, w systemie Windows 7 64-bit, na maszynie wirtualnej Sun Java w wersji 1.6.0_16 (64-bitowej). Przygotowanie statycznej wizualizacji jednopoziomowej struktury danych (czyli wygenerowanie stanu początkowego i pełna relaksacja) zajmowała średnio 0,35 sekundy dla stu bąbelków i 3,75 sekundy dla pięciuset bąbelków. Przygotowanie animacji przejścia do innego stanu zajmuje drugie tyle czasu, ponieważ wiąże się z ponowną relaksacją całej piany. W przypadku animacji „ciąglej” czas ten zwykle jest trochę krótszy, o ile stan docelowy nie różni się drastycznie od stanu początkowego.

Jak widać, generowanie wizualizacji w czasie rzeczywistym może sprawiać problemy dla dużej liczby węzłów, zwłaszcza jeśli wiele z nich należy do tego samego rodzica (relaksacja takiej samej liczby węzłów podzielonych na mniejsze podgrupy trwa znacznie krócej). Pewnym rozwiążaniem w takiej sytuacji mogłoby być przeprowadzanie tylko częściowej relaksacji i przerwanie jej po wyznaczonym czasie. Wyniki eksperymentów wskazywały, że większość wierzchołków stabilizuje się dość wcześnie, dlatego są duże szanse na to, że nawet niecałkowita relaksacja da zadowalające wyniki. Inną możliwością przyspieszenia wizualizacji jest rozbicie procesu relaksacji na wiele wątków. Byłoby to stosunkowo łatwe, ponieważ relaksacja bąbelków należących do różnych rodziców może być przeprowadzana niezależnie. W dobie wielordzeniowych procesorów takie usprawnienie byłoby bardzo znaczące.

Moduły związane z wizualizacją statycznej hierarchii zaimplementowano również w technologii Adobe Flash (język ActionScript). Po przeniesieniu kodu okazało się, że generowanie animacji trwa

około sto razy dłużej niż w Javie. Udało się wprowadzić kilka optymalizacji, co poskutkowało około 3-krotnym przyspieszeniem (pełna relaksacja stu bąbelków trwa teraz około 10 sekund).

6.2 Dalsze perspektywy rozwoju

Ciekawym kierunkiem rozwoju przygotowanej biblioteki jest dodanie funkcji zwiększających interakcję z użytkownikiem. Chodzi głównie o umożliwienie kontrolowania prezentowanych informacji i ich poziomu szczegółowości, na przykład:

- możliwość kontrolowania, które warstwy bąbelków mają być widoczne,
- powiększanie wybranych fragmentów piany,
- wyświetlenie tylko wybranego poddrzewa,
- podświetlanie lub odfiltrowanie bąbelków spełniających określone warunki (na przykład, pasujących do wpisanej frazy).

Wymienione zmiany powinny być wprowadzane w wizualizacji na zasadzie płynnego przejścia. Część z tych efektów wydaje się być łatwa do osiągnięcia z wykorzystaniem zaimplementowanych już animacji, inne wymagałyby zaprojektowania nowych rozwiązań.

Duże możliwości rozwoju leżą w modelu fizycznym relaksacji. Jak już wspomniano w poprzednim podrozdziale, warto poszukać możliwości polepszenia wydajności algorytmów. Poza tym, interesującym sposobem na rozwinięcie modelu byłaby symulacja ścianek bąbelków nie jako prostych odcinków, ale jako łuków o promieniu zależnym od różnicy ciśnień w sąsiednich bąbelkach. Pozwoliłoby to na wierniejsze odwzorowanie rzeczywistych zjawisk zachodzących w pianie, a więc i bardziej „naturalną” wizualizację. Mogłoby to też uprościć cały model, gdyby okazało się, że przestaną być potrzebne takie mechanizmy jak redukcja niepoprawnych ścian, czy wyrównywanie kątów pomiędzy ścianami.

Mechanizmy generowania animacji również stanowią jedynie wstępne propozycje i wymagają głębszego zbadania, zwłaszcza animacja „ciągła”.

Ważnym aspektem, który nie został głębiej poruszony w pracy, jest przydatność zaprojektowanych wizualizacji dla użytkowników. Należałoby zbadać, w jakim stopniu *foamtree* ułatwia zrozumienie prezentowanych treści, zarówno w formie statycznej, jak i animowanej. Badania takie są jednak trudne w realizacji, szczególnie jeśli chodzi o znalezienie reprezentatywnej grupy użytkowników oraz opracowanie miarodajnych i w miarę możliwości obiektywnych testów.

Bardzo interesująca byłaby również próba weryfikacji warunków zbieżności zaprezentowanego modelu od strony teoretycznej; jednak czy jest to w ogóle możliwe nie było przedmiotem rozważań.

Literatura

- [AH01] Keith Andrews, Helmut Heidegger. Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs. *October 2001 IEEE Symposium on Information Visualization*, 2001.
- [AMM⁺07] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, Christian Tominski. Visualizing time-oriented data — a systematic view. *Computers & Graphics*, 31(3):401–409, June 2007.
- [Bat67] G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge University Press, 1967.
- [BCS04] Thomas Bladh, David A. Carr, Jeremiah Scholl. Extending tree-maps to three dimensions: A comparative study. *Proceedings of the 6th Asia-Pacific Conference on Computer-Human Interaction*, strony 50–59. Springer Verlag, 2004.
- [BD05] Michael Balzer, Oliver Deussen. Voronoi treemaps. *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, strona 7, Washington, DC, USA, 2005. IEEE Computer Society.
- [BDL05] Michael Balzer, Oliver Deussen, Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, strony 165–172, New York, NY, USA, 2005. ACM.
- [BHvW99] Mark Bruls, Kees Huizing, Jarke van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, strony 33–42. Press, 1999.
- [Bla05] Thomas Bladh. The effect of animated transitions on user navigation in 3d tree-maps. In *Proc. IEEE IV'05*, strony 297–305, 2005.
- [HvW08] Danny Holten, Jarke J. van Wijk. Visual comparison of hierarchically organized data. *Computer Graphics Forum*, 27(3):759–766, May 2008.
- [KL83] J. B. Kruskal, J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, May 1983.
- [Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.
- [LR95] John Lamping, Ramana Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 1995.
- [MS03] Wolfgang Müller, Heidrun Schumann. Visualization for modeling and simulation: Visualization methods for time-dependent data — an overview. *WSC '03: Proceedings of the 35th Conference on Winter Simulation*, strony 737–745. Winter Simulation Conference, 2003.
- [OS08] Krzysztof Onak, Anastasios Sidiropoulos. Circular partitions with applications to visualization and embeddings. *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*, strony 28–37, New York, NY, USA, 2008. ACM.
- [OW] Stanisław Osiński, Dawid Weiss. Carrot2 clustering engine. <http://search.carrot2.org/>.

- [PGB02] Catherine Plaisant, Jesse Grosjean, Benjamin B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. *IEEE Symposium on Information Visualization*, strony 57–64, Boston, MA, USA, 2002. INFOVIS.
- [PM02] Guillermo Peris, Andrés Marzal. Fast cyclic edit distance computation with weighted edit costs in classification. *Pattern Recognition, International Conference on*, 4:40184, 2002.
- [RG93] Jun Rekimoto, Mark Green. The information cube: Using transparency in 3d information visualization. In *Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93)*, strony 125–132, 1993.
- [RMC91] George G. Robertson, Jock D. Mackinlay, Stuart K. Card. Cone trees: animated 3d visualizations of hierarchical information. *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, strony 189–194, New York, NY, USA, 1991. ACM.
- [SB92] Manojit Sarkar, Marc H Brown. Graphical fisheye views of graphs. *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, 1992.
- [Shn92] Ben Shneiderman. Tree visualization with tree-maps: A 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, January 1992.
- [Sma09] SmartMoney. Map of the market at smartmoney.com.
<http://www.smartmoney.com/map-of-the-market/>, 2009.
- [SW01] Ben Shneiderman, Martin Wattenberg. Ordered treemap layouts. *October 2001 IEEE Symposium on Information Visualization*, 2001.
- [SZ00] John Stasko, Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. *Information Visualization, IEEE Symposium on*, 0:57, 2000.
- [Tay76] Jean E. Taylor. The structure of singularities in soap-bubble-like and soap-film-like minimal surfaces. *The Annals of Mathematics*, 103(3):489–539, May 1976.
- [TS07] Ying Tu, Han-Wei Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1286–1293, November/December 2007.
- [TS08] Tatiana Tekusova, Tobias Schreck. Visualizing time-dependent data in multivariate hierachic plots - design and evaluation of an economic application. *IV '08: Proceedings of the 2008 12th International Conference Information Visualisation*, strony 143–150, Washington, DC, USA, 2008. IEEE Computer Society.
- [vHvW03] Frank van Ham, Jarke J. van Wijk. Beamtrees: compact visualization of large hierarchies. *Information Visualization*, 2(1):31–39, 2003.
- [Wat99] Martin Wattenberg. Visualizing the stock market. *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, strony 188–189, New York, NY, USA, 1999. ACM.
- [Wat05] Martin Wattenberg. A note on space-filling visualizations and space-filling curves. *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, strona 24, Washington, DC, USA, 2005. IEEE Computer Society.
- [WB01] Martin Wattenberg, Ben Bederson. Treemaps java algorithms.
<http://www.cs.umd.edu/hcil/treemap-history/Treemaps-Java-Algorithms.zip>, 2001.
- [WB09] Martin Wattenberg, Ben Bederson. Dynamic treemap layout comparison.
http://www.cs.umd.edu/hcil/treemap-history/java_algorithms/LayoutApplet.html, 2009.
- [WC99] Ulrika Wiss, David A. Carr. An empirical study of task support in 3d information visualizations. *Information Visualisation, International Conference on*, 0:392, 1999.

- [Wet] Kai Wetzel. Pebbles — using circular treemaps to visualize disk usage.
<http://lip.sourceforge.net/ctreemap.html>.
- [WH01] Denis Weaire, Stefan Hutzler. *The Physics of Foams*. Oxford University Press, 2001.
- [WP06] Taowei David Wang, Bijan Parsia. Cropcircles: Topology sensitive visualization of owl class hierarchies. *The SemanticWeb — ISWC 2006*, volumen Volume 4273/2006 serii *Lecture Notes in Computer Science*, strony 695–708. Springer Berlin / Heidelberg, 2006.
- [WvdW99] Jarke J. Van Wijk, Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. *Information Visualization, IEEE Symposium on*, 0:73, 1999.
- [WWDW06] Weixin Wang, Hui Wang, Guozhong Dai, Hongan Wang. Visualization of large hierarchical data by circle packing. *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, strony 517–520, New York, NY, USA, 2006. ACM.



© 2010 Mateusz Matela

Instytut Informatyki, Wydział Informatyki i Zarządzania
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX.

BibT_EX:

```
@masterthesis{ key,  
    author = "Mateusz Matela",  
    title = "{Metody wizualizacji dla zmieniających się w czasie zbiorów danych o strukturze  
hierarchicznej}",  
    school = "Poznan University of Technology",  
    address = "Poznań, Poland",  
    year = "2010",  
}
```