Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

VLAK – Bezstratny Kodek Dźwięku

Wstęp

Na wstępie należy postawić sobie pytanie o celowość tworzenia tzw. bezstratnego kodeka dźwięku. Istnieje bowiem bardzo wiele kodeków, które nie mają tej cechy, a są bardzo popularne (np. MP3), natomiast o kodekach dźwięku „bezstratnych" wiele osób nigdy nie słyszało!

Bezstratność rozumiana jest tutaj jako całkowita zgodność bit po bicie sygnału oryginalnego i zdekompresowanego.[1] Na drugim biegunie znajdują się popularne kodeki, jak MP3, OGG, MPC, które zapewniają tzw. bezstratność percepcyjną, czyli mimo iż sygnał zdekompresowany różni się (często nawet znacznie) od oryginalnego to jednak słuchając go ma się wrażenie iż jest to sygnał oryginalny. Oczywiście jakość sygnału zależy od tego ile bitów zostanie przeznaczone na taką reprezentację. W takim podejściu wykorzystuje się właściwości ucha ludzkiego i usuwa z sygnału informacje, których człowiek nie słyszy. Niestety nie wszyscy ludzie słyszą tak samo, tak więc być może większości z nas dźwięk w MP3 wystarcza, są jednak jednostki które oprócz świetnego słuchu mają także świetne zestawy HiFi, dla nich MP3 jest nie do przyjęcia.

Zastosowania bezstratnego kodeka dźwięku to między innymi:

- archiwizacja plików wav (przede wszystkim różnego rodzaju muzyki w jakości „cd-audio")
- profesjonalne przetwarzanie dźwięku (jeśli mamy zamiar poddać dźwięk obróbce cyfrowej, np. redukcji szumów, to kodeki stratne nie nadają się. Zazwyczaj w takich sytuacjach stosowało się po prostu pliki wav, jednak dzięki bezstratnemu kodekowi dźwięku możemy zmniejszyć zapotrzebowanie na miejsce na dysku wymagane przez programy do edycji dźwięku)
- „audiofilskie" (dźwięk w jakości identycznej jak cd-audio)
- bootlegi (większość nagrań koncertowych jest dystrybuowana przez fanów w formacie bezstratnym, wynika to z tego iż dla nagrań na żywo kodeki stratne nie działają zbyt dobrze, a poza tym pozwala to na nagranie na CD w jakości cd-audio a nie „fake cd-audio" ;-))
- odtwarzacze przenośne (kilka nowszych modeli odtwarzaczy przenośnych potrafi już odtwarzać pliki w bezstratnych formatach audio, przykładowo iPod używa formatu opracowanego przez Apple, a inne firmy próbują wdrożyć obsługę formatu FLAC)

Można więc zaobserwować że formaty bezstratne wykorzystywane są tam gdzie wymagany jest dźwięk najlepszej jakości, dlatego też tworząc kodek VLAK przyjęliśmy następujące założenia:

- Zajmujemy się tylko dźwiękiem wysokiej jakości w formacie WAVE PCM (16bit, stereo, o częstotliwości 44100 Hz)
- bezstratność numeryczna (czyli „bit po bicie")
- przenośność i związana z nią dostępność kodu źródłowego (ANSI C++). Dla bardziej zaawansowanych metod predykcji wymagany jest jednak koprocesor matematyczny

1 jest to tzw. bezstratność numeryczna

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

(FPU), co może zawęzić możliwość zastosowania w odtwarzaczach przenośnych.

Wzorowaliśmy się na istniejących już kodekach dźwięku, jednak zazwyczaj ich kod źródłowy był albo niedostępny, albo bardzo nieprzystępnie napisany (zupełny brak komentarzy, wstawki optymalizacyjne w asemblerze, itp. itd.). Dlatego też VLAKa należy uznać za projekt całkowicie wykonany przez nasz zespół.[2]

Poniżej przedstawiamy przegląd interesujących kodeków dźwięku, które zainspirowały nas do zajęcia się tym zagadnieniem.

Kodek FLAC[3] (Free Lossless Audio Codec) jest dynamicznie rozwijany, a jego kod źródłowy udostępniono publicznie (zgodnie z zasadami licencji GPL). Jest on wolny od jakichkolwiek ograniczeń patentowych. Stał się swego rodzaju standardem w środowisku Open Source. Dorównuje efektywnością kompresji najnowszym kodekom i jest przystosowany do wygodnego odtwarzania (np. nieduże wymagania do odtwarzania w czasie rzeczywistym, szybka operacja poszukiwania danej pozycji w strumieniu, wspieranie tzw. tag'ów). Dostępnych jest stosunkowo dużo wtyczek do obsługi tego formatu do wielu z programów odtwarzających muzykę na niemałej liczbie platform sprzętowych. Format ten używa albo predyktorów liniowych z ustalonymi współczynnikami albo metody LPC według algorytmu Levinsona-Durbina. Ciekawostką jest wewnętrzna kontrola integralności poprzez zliczanie wartości sum CRC i MD5 na kilku poziomach. Kodek ten posłużył nam jako wzór do budowy własnego kodeka. W szczególności architektura VLAKa oraz FLACa są bardzo zbliżone (chociaż oczywiście VLAK jest znacznie prostszy), także zestaw opracowanych przez nas predyktorów jest bliski temu co jest dostępne we FLACu.

Kodek SHORTEN, napisany przez Tony'ego Robinsona, był jednym z pierwszych kodeków bezstratnych. Początkowo był to projekt studencki, jednak później stał się produktem komercyjnym. W internecie znaleźć można dosyć dobry opis teoretyczny, zawierający wiele ciekawych informacji o predyktorach FIR oraz LPC[4]. Z całą pewnością FLAC (a także VLAK) jest w dużej mierze oparty na informacjach zawartych w tym dokumencie.

Kodek Monkey's Audio ma nieco większą efektywność kompresji niż FLAC i nie ustępuje mu w większości dziedzin. Jego źródło jest także publicznie dostępne. Początkowo pisany był jedynie pod system Windows. Stosunkowo niedawno powstały implementacje także pod system Linux, w tym wtyczki do kilku popularnych programów odtwarzających muzykę. Jednak ciągle liczba platform umożliwiających odtwarzanie plików w tym formacie jest dość mała. Prawdopodobnie[5] zastosowano w nim predyktor oparty o sieć neuronową. Kodek ten jest popularny w sieciach wymiany plików - wykorzystywany jest przy umieszczaniu w nich obrazów płyt cd-audio.

Kodek La jest swego rodzaju ciekawostką. W testach efektywności na stronie:

http://flac.sourceforge.net/comparison.html

jako jedyny wykazuje średni współczynnik kompresji poniżej 50% (dokładnie 0.4986). Jednak

2 są od tego stwierdzenia dwa wyjątki: funkcja wyliczająca współczynniki w metodzie LPC – zaczerpnięta z libFLAC, ponieważ napisana „od zera” w oparciu o algorytm Levinsona-Durbina nie działała poprawnie oraz kod dla predyktora Wavelet, który został przez nas tylko doprowadzony do stanu zgodności z ANSI C++, co wcale nie było zadaniem łatwym!

3 http://flac.sourceforge.net/

4 http://mi.eng.cam.ac.uk/reports/ajr/TR156/tr156.html

5 kod źródłowy tego kodeka, jest całkowicie pozbawiony komentarzy. Klasa implementująca predyktor nazywa się NNFilter, co najprawdopodobniej jest skrótem od Neural Network Filter

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

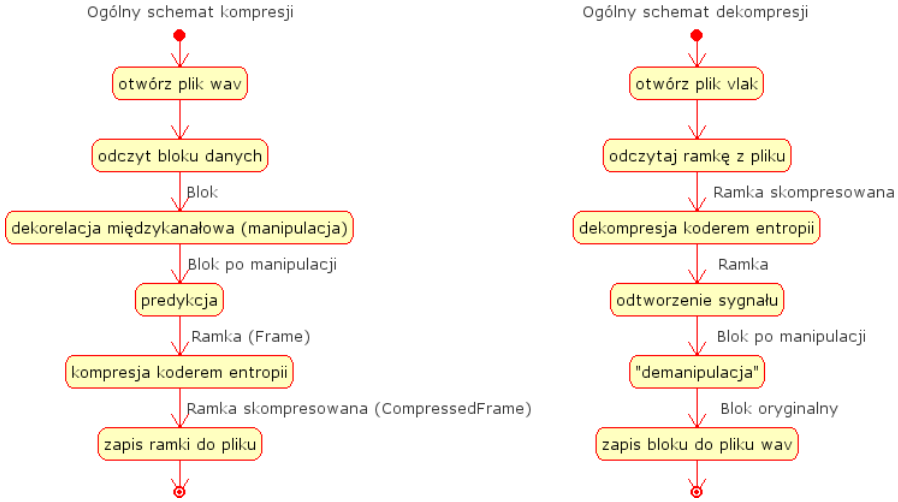
kosztem tego jest nawet dziesięciokrotnie dłuższy czas kompresji oraz dekompresji, co może uniemożliwiać jego zastosowanie do odtwarzania w czasie rzeczywistym. Opis teoretyczny tego kodeka można znaleźć na stronie:

http://www.lossless-audio.com/theory.htm

Według niego dobre rezultaty kompresji La zawdzięcza innowacyjnemu sposobowi kodowania sygnału rezydualnego.

Stosunkowo niedawno powstały także kodeki bezstratne opracowane przez duże i znane firmy. Microsoft dodał możliwość bezstratnej kompresji do WMA (Windows Media Audio), a Apple utworzył format ALAC (Apple Lossless Audio Codec)[6].

Struktura kodeka



Każdemu etapowi kompresji towarzyszą odpowiednie klasy lub interfejsy. Najważniejsze pojęcia to:

blok — fragment pliku wav, zawierający N próbek dla każdego z kanałów. Jest implementowany przez klasę `CBlock`

predyktor — klasa odpowiedzialna za stworzenie reprezentacji pośredniej bloku, która byłaby bardziej podatna na kompresję. Jej zachowanie definiuje interfejs `IPredictor`. VLAK dostarcza kilku predyktorów, które charakteryzują się różną efektywnością

ramka — reprezentacja pośrednia bloku danych. Po poddaniu jej kompresji jest serializowana do pliku vlak. Wynika stąd, że każdemu blokowi odpowiada dokładnie jedna ramka. Przykładowo w ramce zapisane są współczynniki dla predyktora oraz tzw. sygnał rezydualny. Operacje jakie mogą zostać wykonane na ramce definiują interfejsy `IFrame` oraz `ICompressedFrame`

6 http://en.wikipedia.org/wiki/Apple_Lossless_Encoding

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Opis faz kompresji

W tym miejscu chcieliśmy przedstawić dokładniejszy opis każdej z faz kompresji kodeka VLAK.

Podział sygnału na bloki

Strumień próbek dzielony jest na fragmenty o równej długości, czyli na tzw. bloki. Pojęcie bloku odpowiada pojęciu kontekstu w kompresji danych. W każdym bloku znajduje się więc N próbek dla każdego z kanałów. W tej fazie kompresji kanały odzwierciedlają zawartość pliku WAV, a więc będą to 2 kanały: Lewy i Prawy. Za podział na bloki odpowiadają klasy implementujące interfejs `IBlocksProvider`.

Otwartym zagadnieniem jest wybór najlepszego rozmiaru bloku. VLAK pozwala wybrać rozmiar bloku jako parametr kompresora, nie potrafi jednak dobierać go adaptacyjnie. Nie każdy rozmiar bloku jest sensowny, istnieją bowiem wewnętrzne ograniczenia rozmiaru bloku do około 64KB, a dodatkowo niektóre rozmiary mogą być preferowane przez niektóre predyktory.[7] W dalszej części sprawozdania znajdują się porównania jak rozmiar bloku wpływa na efektywność kompresji.

Dekorelacja międzykanałowa sygnału

W wielokanałowych plikach dźwiękowych wartości próbek w sąsiednich kanałach są zwykle silnie skorelowane. Dlatego dobre rezultaty daje zastąpienie kanałów lewego i prawego przez kanały middle i side:

$$middle[n]=\frac{left[n]+right[n]}{2}$$

$$side[n]=left[n]-right[n]$$

Wartość middle[n] jest obcinana do najbliższej liczby całkowitej “w dół”. Taka transformacja pozwala na bezstratne odtworzenie wartości oryginalnych a prowadzi do znacznego zwiększenia efektywności kompresji bardzo niewielkim kosztem.

Tak działa manipulator `CMidSideChannelManipulator`. Jednak powstaje mały problem: różnica dwóch 16-bitowych próbek może mieć długość 17 bitów (dotyczy to tylko kanału side). Taki sygnał zwiększa swoją długość i jest ponadto trudniejszy do obróbki. Dlatego otrzymane różnice są obcinane do 16 bitów. W praktyce dla sygnałów dźwiękowych sytuacja nieodwracalnego przekształcenia następuje przy tym niesłychanie rzadko (w badanych przez nas przypadkach nigdy nie wystąpiła). Jednak teoretycznie może mieć miejsce. Aby wyeliminować tę możliwość wprowadziliśmy `CAdaptiveChannelManipulator`, który najpierw bada daną ramkę, czy przekształcenie z `CMidSideChannelManipulator`'a jest dla niej bezstratne. Jeśli tak, używa go. Jeśli natomiast nie, używa `CSimpleChannelManipulator`'a, czyli przekształcenia tożsamościowego. Informacja o użytym manipulatorze jest przy serializacji zapisywana w nagłówku ramki.

`CSimpleChannelManipulator`'a można także użyć w celach testowych, aby sprawdzić stopień poprawy jakości kompresji przy przekształceniu na sygnał mid-side.

7 przykładowo Wavelet powinien pracować na ramkach o rozmiarach będących potęgami dwójki.

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Predykcja

FIR

Podobny schemat predykcji był po raz pierwszy użyty w kodeku Shorten. Informacje na jego temat można znaleźć pod adresem:

http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf (strona 22)

Jest to bardzo proste koncepcyjnie rozwiązanie, które sprawdza się tam, gdzie zależy nam bardziej na szybkości algorytmu niż na stopniu kompresji. Wartość każdej kolejnej próbki jest tam przewidywana na podstawie trzech poprzednich próbek według zawsze tego samego wzoru:

$$x'[n]=3\mathrm{x}[n-1]-3\mathrm{x}[n-2]+x[n-3]$$

Jedynie trzy pierwsze próbki z każdego bloku wymagają zapisania w ramce przez dokładne ich wartości. Dla pozostałych zapisujemy różnice x[n] - x'[n], czyli sygnał residualny.

LPC

LPC (Linear Predictive Coding) jest stosunkowo popularną techniką kompresji oraz syntezy mowy. Dla przykładu LPC jest wykorzystywane w standardzie GSM. W porównaniu do współczesnych metod kompresji mowy (a tym bardziej muzyki) jest to metoda raczej słaba, dlatego też kompresja (stratna) mowy z jej wykorzystaniem jest już rzadkością.

Predykcja polega na znalezieniu m współczynników liniowego aproksymatora sygnału dla których suma błędów średniokwadratowych reprezentacji będzie minimalna. Wartość sygnału przewidywanego w chwili t jest liniową kombinacją poprzednich m próbek: $\hat{s}(t)=\sum_{i=1}^{m} a_i s(t-i)$. Minimalizacji średniokwadratowej podlega sygnał rezydualny: $e(t)=s(t)-\hat{s}(t)$. Do tego celu potrzebna jest nam macierz autokorelacji sygnału. Do wyliczenia współczynników predyktora wykorzystuje się algorytm Levinsona-Durbina.[8]

Dla metody tej istotnym zagadnieniem jest dobór optymalnego rzędu predyktora. VLAK obsługuje dwa podejścia: adaptacyjne oraz nieadaptacyjne.

Metoda nieadaptacyjna jest bardzo prosta – rząd predyktora jest podawany jako parametr predyktora i nie ulega ona zmianie w trakcie kompresji. Oczywiście takie podejście ma kilka istotnych wad: przede wszystkim bloki sygnału mogą mieć zupełnie inna charakterystykę, być może więc dla niektórych z nich ustalony rząd będzie niepotrzebnie wysoki, co spowoduje wzrost rozmiaru ramki (gdyż za dużo będzie współczynników predyktora a sygnał rezydualny nie będzie bardziej podatny na kompresję) oraz czasu obliczeń (gdyż zależy on od rzędu predyktora). Podany rząd może być także zbyt niski w sytuacji gdy w bloku „dużo się dzieje”.

Rozwiązaniem powyższych problemów jest metoda adaptacyjna. Polega ona na wyborze rzędu predyktora z przedziału $\langle minOrder, maxOrder \rangle$ stosownie do charakteru analizowanego bloku próbek. Czas obliczeń jest określony przez maksymalny rząd predyktora. Koncepcja tej metody jest atrakcyjna, dobre rozwiązanie jest jednak trudne do uzyskania.

We VLAKu rząd predyktora ustalany jest na podstawie oszacowania rozmiaru sygnału rezydualnego po kompresji. Dla każdego rzędu predyktora z ustalonego przedziału estymowany jest parametr k kodera Rice'a, czyli minimalny rozmiar w bitach każdej zapisywanej próbki i na

8 więcej informacji można znaleźć na stronach: http://en.wikipedia.org/wiki/Linear_prediction oraz http://cnx.rice.edu/content/m10482/latest/

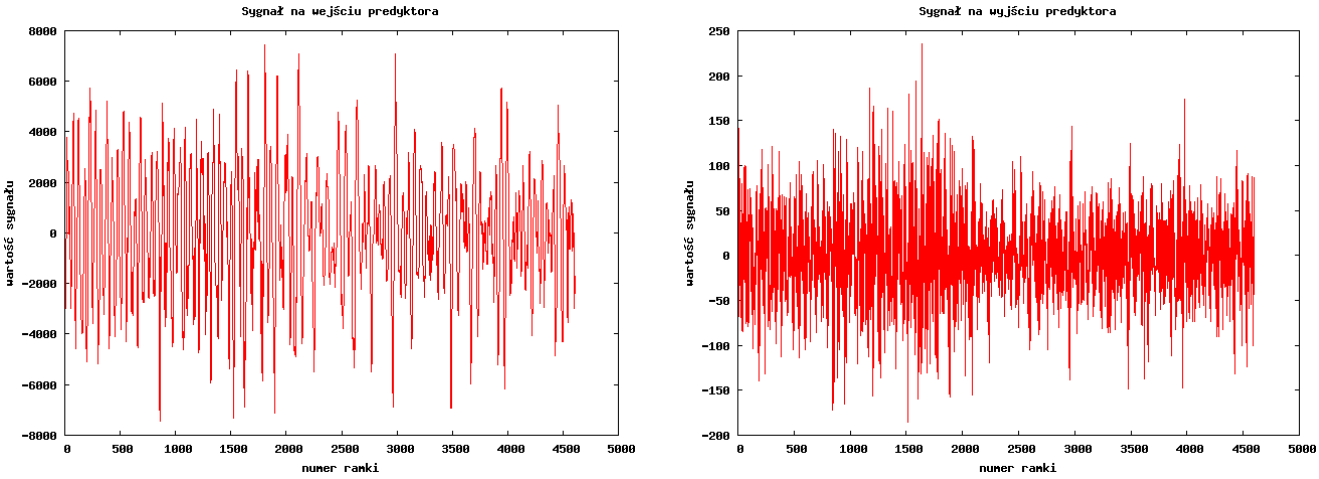
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

podstawie tej informacji wyliczany jest przewidywany rozmiar skompresowanej ramki. Następnie wybierany jest rząd dla którego rozmiar ten jest najmniejszy.

Można wyobrazić sobie także inne schematy:

- w koderze SHORTEN jeśli 2 kolejne rzędy nie poprawiają stopnia kompresji to wyszukiwanie optymalnego rzędu jest kończone.
- w koderze FLAC (jeśli korzystamy z przełącznika -e, stosowanego domyślnie w trybie -8) rozmiary ramek po kompresji są „fizycznie” wyliczane, więc nie ma mowy o estymacji – daje to oczywiście optymalne wyniki jest to jednak czasochłonne.

Poniższy wykres przedstawia przykładowy sygnał na wejściu predyktora oraz sygnał rezydualny na jego wyjściu:



FFT (metoda autorska)

Podstawową ideą tej metody jest użycie do predykcji widma częstotliwościowego kompresowanego sygnału. Często jest tak, że wiele informacji o sygnale jest zawarte w stosunkowo niewielkiej ilości głównych częstotliwości o najwyższej amplitudzie, a pozostałe mało wnoszą do ogólnego jego przebiegu.

Z teorii sygnałów wiadomo, że dany sygnał ciągły może zostać dokładnie odtworzony, jeśli znamy jego widmo do częstotliwości $2f_{max}$, gdzie f_{max} to maksymalna częstotliwość występująca w sygnale. Jeśli natomiast mamy spróbkowany w n punktach sygnał, to dokładnie opisuje go jego dyskretna transformata Fouriera o n współczynnikach zespolonych. Przy przetwarzaniu sygnału rzeczywistego (z takim przypadkiem mamy do czynienia w kompresji dźwięku) widmo jest w pewien sposób symetryczne. Na pozycji 0 jest składowa stała, natomiast pozostałe amplitudy są symetryczne względem pozycji n/2 (która jest pojedyncza), przy czym odpowiadające sobie liczby zespolone są ze sobą sprzężone. W efekcie widmo można opisać n+2 liczbami rzeczywistymi. Dodatkowo uwzględniając to, że składowa stała oraz składowa na pozycji n/2 są rzeczywiste, można by zmniejszyć tą ilość do n, ale nie zostało to zaimplementowane w ten sposób.

Zwykłe kompresowanie widma, które w składa się z liczb rzeczywistych prawdopodobnie byłoby mniej efektywne niż kompresja samego sygnału. Dlatego w tej metodzie widmo jest „kwantyzowane” (opis niżej). W wyniku tego w widmie pojawia się wiele zer, a niezerowe

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

współczynniki można łatwo zapisać jako niewielkie liczby całkowite. Jednak takie widmo nie opisuje już dokładnie oryginalnego sygnału. Dlatego obliczana jest różnica między sygnałem otrzymanym na podstawie skwantowanego widma oraz oryginalnym. Tak powstały sygnał residualny musi zostać dołączony do skompresowanych danych aby umożliwić wierne odtworzenie oryginalnego sygnału.

Szczegóły implementacji

Kompresja:

- Zespolona dyskretna transformata Fouriera sygnału jest obliczana z użyciem algorytmu FFT.
- Kwantyzacja widma:
  - wszystkie współczynniki są zaokrąglane do najbliższej wielokrotności pewnej liczby (jeden z parametrów predyktora - Qfactor), małe liczby stają się zerami
  - pomijane są współczynniki odpowiadające częstotliwościom o amplitudzie mniejszej niż ustalone minimum, które jest pewnym ułamkiem największej amplitudy (ułamek ten jest również parametrem predyktora - MinRelAmpl)
- Skwantowane widmo po podzieleniu przez Qfactor jest pakowane prostym koderem RLE nastawionym na długie sekwencje zer. Najpierw wyznaczane i zapisywane do strumienia wynikowego są długości leżących na przemian ciągów zer i elementów różnych od zera (przy czym brane są pod uwagę łącznie część rzeczywista i urojona). Następnie zapisywane są kolejne niezerowe elementy (na przemian część rzeczywista i urojona).
- Na podstawie skwantowanego widma wyznaczany jest przewidywany sygnał przy użyciu odwrotnej FFT i obliczany jest sygnał rezydualny zapisywany do strumienia wyjściowego.
- Wszystkie powyższe dane są kompresowane koderem Rice'a
- W każdej ramce dodatkowo przechowywane są: rozmiar oryginalnej ramki oraz wartości parametrów Qfactor i MinRelAmpl

Dekompresja:

- Dekompresowane jest skwantowane widmo sygnału.
- Na podstawie widma z użyciem odwrotnej FFT obliczany jest przewidywany sygnał, do którego dodawany jest następnie sygnał rezydualny.

Do obliczania FFT użyłem kodu znalezionego w sieci Internet.

Parametry

- Rozmiar bloku – ze względu na użytą implementację FFT powinien być potęgą 2, gdyż w przeciwnym razie zostanie wyrównany do najwyższej większej potęgi 2. Ostatnia ramka zazwyczaj nie ma takiej długości, więc tracimy w niej na wyrównanie.

  Blok nie powinien być bardzo duży, gdyż wtedy widmo będzie wyznaczane z długiego odcinka czasu, a wtedy jest większa szansa, że sygnał zmieni charakter – w efekcie otrzymamy wymieszane widma. Z drugiej strony za mały blok spowoduje duży narzut.

  Proponowana wartość: 512 – 2048, domyślnie: 1024
- Qfactor – stopień skwantowania widma. Części rzeczywista i urojona widma po kwantyzacji są wielokrotnościami Qfactor. Decyduje on zatem o ziarnistości podziału

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

skali amplitud.

Proponowana wartość: 8 – 20, domyślnie: 8

- MinRelAmpl – używane do usunięcia częstotliwości o względnie niskiej amplitudzie, które nie wpływają znacząco na sygnał, natomiast bez tego zabiegu musiałyby zostać zapamiętane.

  Proponowana wartość: 0,005 – 0,02, domyślnie: 0,01

Możliwe ulepszenia

- Adaptacyjny dobór rozmiaru bloku oraz parametrów Qfactor i MinRelAmpl
- Użycie innego kodera entropii do kompresji widma, np. kodera Huffmana
- Nierównomierna kwantyzacja częstotliwości faworyzująca pewne pasma. W zależności od kompresowanych danych mogłyby to być np.: średnie częstotliwości dla mowy, basy dla muzyki, itp.

Wavelet

Użyliśmy jako bazy gotowej implementacji odwracalnej transformaty falkowej (w oparciu o algorytm interpolacji liniowej) dostępnej na stronie:

http://www.bearcave.com/misl/misl_tech/wavelets/packet/index.html

Jedyne modyfikacje oryginalnego kodu, których dokonaliśmy, miały na celu umożliwienie jego kompilacji w środowisku systemu Linux i kompilatora gcc.

Wybraliśmy ten konkretny algorytm, gdyż spośród branych przez nas pod uwagę (rozważaliśmy jeszcze algorytm Haara i transformatę TS) dawał najlepszą efektywność kompresji.

Wszystkie algorytmy, które braliśmy pod uwagę do poprawnego działania wymagały podawania im na wejście ciągów o długości równej potędze dwójki. Gdy dana ramka nie ma takiej długości (zwykle ostatnia ramka jej nie ma) jest sztucznie wydłużona zerami do najbliższej dozwolonej wartości. Aby odtworzyć po dekompresji oryginalny plik dźwiękowy konieczne jest oczywiście zapamiętanie prawdziwej długości takiej ramki.

Kompresja koderem entropii

Idea kodera Rice'a

Przy bezstratnym kodowaniu dźwięku bardzo ważnym zagadnieniem jest kompresja sygnału rezydualnego, który powstaje na wyjściu predyktora. Zadanie polega na takim zapisaniu 32 bitowych próbek dźwiękowych, aby zajmowały jak najmniej miejsca. Idealnie właśnie pasuje do tego zadania koder Rice'a, który jest szczególnym przypadkiem kodera Golomba.

Na wyjściu predyktora otrzymujemy wartości, które są relatywnie małe (do ich reprezentacji potrzebujemy kilku bitów). Koder Rice'a dzieli wejściową liczbę na dwie części (LSB – least significant bits, MSB – most significant bits) na podstawie parametru 'k' (k – liczba LSB), po czym część MSB zapisuje unarnie, a LSB zostawia nienaruszoną. Następnie na wyjście wypisuje zakodowaną część MSB i po niej LSB.
Np. Załóżmy, że chcemy zakodować binarną liczbę 1010110, mając dany parametr 'k' = 5.

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Część LSB, to 10110, część MSB to 10. Po zapisaniu MSB unarnie 110, otrzymujemy na wyjściu liczbę 11010110.

Podstawowymi problemami przy kodowaniu Rice jest zakodowanie dużej liczby przy małym 'k' oraz optymalne dobranie parametru 'k'.

Kodowanie dużych liczb.

Przy relatywnie małym 'k' i dużej liczbie, może dojść do sytuacji, w której będziemy musieli zakodować unarnie dużą liczbę, np. 159, co spowoduje, że zamiast 32 bitów wejściowych, otrzymamy ponad 159 bitową liczbę. Aby uniknąć takich sytuacji stosujemy następującą metodę:
Gdy liczba wejściowa ('X') jest większa niż ustalona granica('T'), zapisujemy T bitów '1'. Następnie zapisujemy floor($\log_2$(X-T)) bitów '1' (jest to liczba bitów, jaką potrzebujemy, żeby zapisać liczbę X-T w normalnym kodzie dwójkowym). Następnie wypisujemy na wyjście liczbę X-T dwójkowo.
W ten sposób ustrzegamy się przed dużymi skokami wejściowych wartości, które 'psują' jakość kompresji.

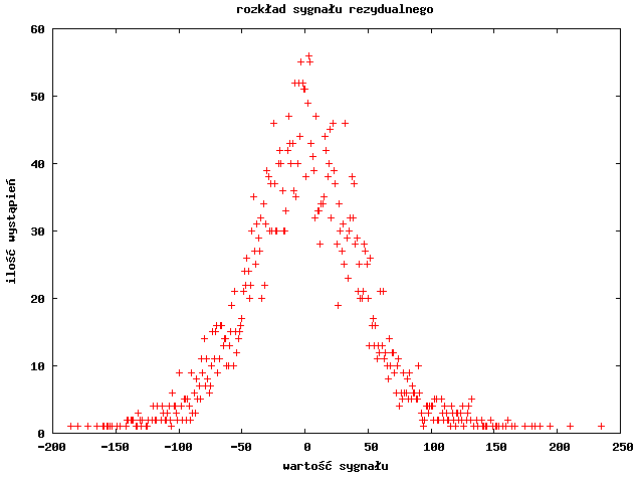
Optymalny dobór parametru 'k'

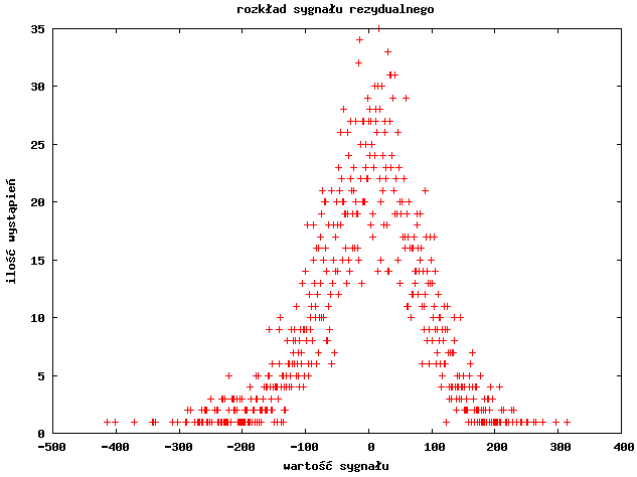
Ze względu na to, że dane wejściowe są silnie zróżnicowane, bardzo trudno jest przyjąć jakąś skuteczną metodę adaptacji parametru 'k'. Przyjęliśmy metodę stosowaną w kodeku *Monkey's Audio*, gdzie analizujemy ostatnie 32 wartości i na ich podstawie przewidujemy optymalny parametr k.

Zalety kodera Rice'a

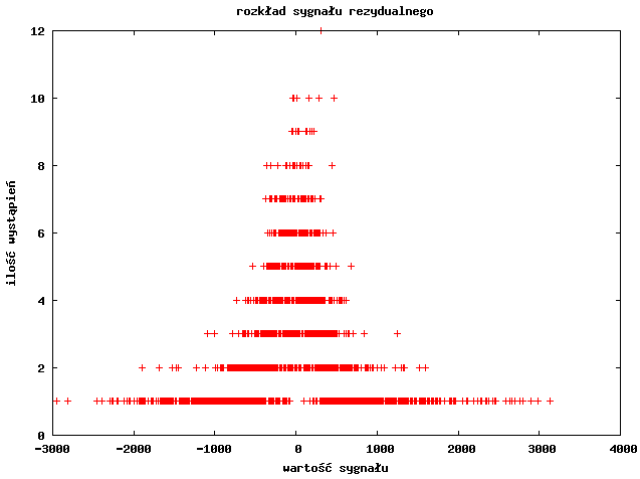
Jak już wcześniej wspominaliśmy, koder Rice'a najlepiej sprawuje się w sytuacjach, w których dane na wejściu, są jak najmniejsze. Jak to widać z poniższych przykładowych wykresów sygnału rezydualnego, wartości skupiają się wokół zera, dzięki czemu ten sposób kodowania jest bardzo wydajny. Duże usprawnienie kodowania daje również adaptacyjność zastosowana w naszej implementacji, gdyż daje nam to bardziej optymalne kodowanie zgrupowań o podobnych wartościach, niż sztywne przyjęcie parametru 'k'

Przykładowe rzeczywiste rozkłady (funkcje gęstości) sygnału rezydualnego





Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>



Zapis do pliku .vlak

Do zapisu skompresowanych danych wykorzystujemy koncepcję serializacji, dzięki czemu format pliku .vlak jest wyjątkowo prosty, a jednocześnie łatwy w rozbudowie. Plik składa się z nagłówka (serializowany obiekt klasy `CVLAKHeader`) oraz z pewnej ilości skompresowanych ramek (serializowane klasy implementujące interfejs `ICompressedFrame`).

Niestety C++ nie dostarcza mechanizmów serializacji, należało więc napisać je samemu. Serializacja ramek, polega na zapisaniu: długości ramki oraz jej dokładnego typu (jest to tzw. `GUID`), tak aby wiadomo było, jaką metodę deserializacji wywołać przy odczycie. Później zapisuje się już zawartość ramki, a więc sygnał rezydualny oraz parametry predyktora. Takie podejście pozwala przykładowo na stosowanie dla każdego bloku danych dowolnego predyktora, a nawet dowolnego kodera entropii.

Odczyt pliku .vlak polega zatem na deserializacji obiektów z pliku.

Opis użycia programu vlak

Nasz kodek posiada dosyć dużą ilość opcji, które mają często znaczący wpływ na efektywność kompresji. Aby zobaczyć co vlak ma nam do zaoferowania wystarczy wywołać go w następujący sposób:

```
vlak
```

```
vlak --help
```

Spowoduje to wyświetlenie możliwych do ustawienia opcji oraz poinformuje o akceptowanych trybach pracy. Jak łatwo zauważyć opcje należy podawać w postaci akceptowalnej przez standardową funkcję `getopts_long()`.

VLAK może pracować w następujących trybach:

```
Encoding: ./vlak [encoding-options] INPUTWAVFILE [OUTPUTVLAKFILE]
```

```
Decoding: ./vlak -d [decoding-options] VLAKFILE [OUTPUTWAVFILE]
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
Analyzing: ./vlak -a [analysis-options] VLAKFILE
```

Tryby pracy

`-d, --decode`	dekodowanie podanego pliku .vlak
`-h, --help`	wyświetlenie pomocy
`-a, --analyze`	analizowanie podanego pliku .vlak

Opcje enkodera

`-p, --predictor=#`	wybór typu predyktora
`-b, --blocksize=#`	wybór rozmiaru bloku w próbkach na kanał
`-m, --manipulator=#`	wybór typu manipulatora
`-l, --lpcorder=#,#`	wybór minimalnego i maksymalnego rzędu dla predyktora LPC

Opcje analizatora

Każda z opcji powoduje utworzenie plików zawierających interesujące nas informacje. UWAGA: naraz można podać tylko jedną opcję.

`--residual-distrib`	wypisuje rozkład sygnału rezydualnego dla każdej ramki w osobnym pliku
`--residual-signal`	wypisuje sygnał rezydualny dla każdej ramki dla każdej ramki w osobnym pliku
`--original-signal`	wypisuje oryginalny sygnał (na wejściu kodeka) dla każdej ramki w osobnym pliku
`--manipulated-signal`	wypisuje sygnał poddany dekorelacji międzykanałowej dla każdej ramki w osobnym pliku
`--predicted-signal`	wypisuje sygnał będący wynikiem predykcji dla każdej ramki w osobnym pliku
`--predictor-order`	wypisuje informacje o rzędzie predyktora. Tworzony jest jeden plik zawierający rzędy zastosowanych predyktorów. Opcja ta ma zastosowanie w adaptacyjnych metodach predykcyjnych
`--info`	wypisuje podstawowe informacje o pliku vlak, np. ilość ramek, właściwości sygnału PCM
`--compression-ratio`	wypisuje stopień kompresji sygnału rezydualnego dla każdej ramki osobno. Stopień ten pokazuje TYLKO efektywność kodera entropii, nie uwzględnia narzutów związanych z dodatkowym informacjami zapisywanymi w ramce

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Dostępne typy predyktorów

```
0   Simple
1   LPC
2   FIR
3   Wavelet
4   FFT
```

Dostępne typy manipulatorów

```
0   CSimpleChannelManipulator
1   CMidSideChannelManipulator
2   CAdaptiveChannelManipulator
```

Pomiary

Opis plików testowych

Poniższe utwory pochodzą z oryginalnych płyt, które oczywiście polecamy. Zostały one zgrane z jakością CD-AUDIO a więc 16 bit, 44100 Hz, stereo.

Izrael – See I & I

Utwór ten reprezentuje muzykę reggae. To co wyróżnia ten gatunek, to charakterystyczna rytmiczność, czyli tzw. offbeat, zwany też "akcentem na słabą część taktu". W skład dość bogatego instrumentarium wchodzi między innymi kilka instrumentów dętych oraz pokaźny zestaw tzw. „przeszkadzajek”, co bardzo urozmaica muzykę i przy okazji rozszerzając jej pasmo częstotliwościowe.

Led Zeppelin – D'yer Mak'er

Jest to dość stary utwór jak na muzykę rockową, bo stworzono go około trzydzieści pięć lat temu, kiedy wzmacniacze muzyczne były na dużo niższym poziomie jakości, co wyraźnie słychać chociażby w brzmieniu gitar. I właśnie dlatego wybraliśmy go do testowania naszego kodeka.

Gladiator OST – Now We Are Free

Utwór ten jest zbliżony stylistycznie do muzyki poważnej, symfonicznej. Jest to bardzo wartościowy materiał do testów, gdyż na tego rodzaju plikach dźwiękowych niektóre standardy kompresji stratnej okazywały się szczególnie nieefektywne. Być może wyjątkowo liczny zestaw instrumentów i skrajna wielogłosowość utrudni zadanie także naszemu kodekowi.

Sweet Noise – 9/1

Jest to reprezentant "ostrzejszych" odmian muzyki metalowej. Bardzo charakterystyczne dla tego gatunku brzmienie charakteryzuje duży udział fal o niskiej częstotliwości. Wysokie częstotliwości reprezentuje między innymi wokaliza samej Anny Marii Jopek. Można powiedzieć, że w tego muzyce większość instrumentów jednoczy siły by razem stworzyć ścianę dźwięku - nie występuje specjalna wielość brzmień i melodii.

Kult – Baranek

Powszechnie znany utwór zespołu Kult.

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Myslovitz – Zgon (live)

Nagranie z koncertu Myslovitz wykonane prostym dyktafonem kasetowym z wbudowanym mikrofonem. Sygnał jest więc zaszumiony, powinny występować także silne korelacje międzykanałowe. Odzwierciedla to w przybliżeniu charakterystykę bootlegów o słabej jakości (nagrywanych z widowni a nie z konsolety).

Vypsana Fixa – Palenie Titonia

Kolejny utwór rockowy, tym razem śpiewany po czesku. Nagrany został dosyć głośno, a sygnał w każdym z kanałów jest zróżnicowany.

Turbo – Szalony Ikar

Utwór ten, popularnego niegdyś polskiego zespołu Turbo, reprezentuję muzykę metalową. Jest wyjątkowo głośny i grany w szybkim tempie. Nagranie pochodzi ze „zremasterowanej" płyty CD.

Iron Maiden – Sign Of The Cross

Tym razem jest to długi (ponad 11 minutowy) utwór zespołu Iron Maiden. Jest to jednak w miarę spokojny utwór ze zróżnicowanymi fragmentami, cichego, nastrojowego grania oraz „ostrzejszymi" fragmentami solowymi.

Zastosowane miary efektywności kompresji

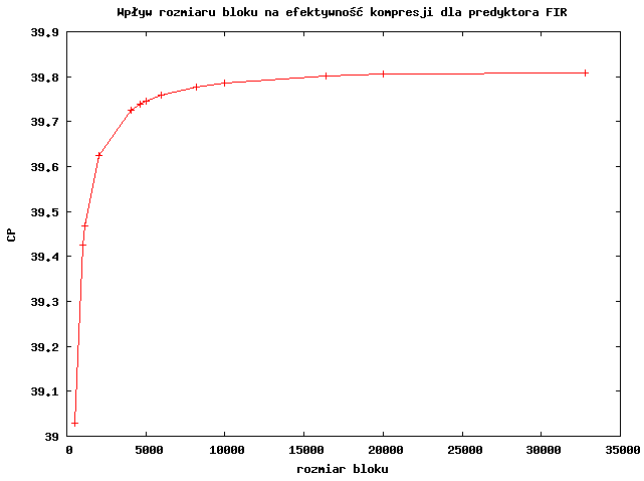
$$CR = \frac{rozmiar\ pliku\ oryginalnego}{rozmiar\ pliku\ skompresowanego}$$

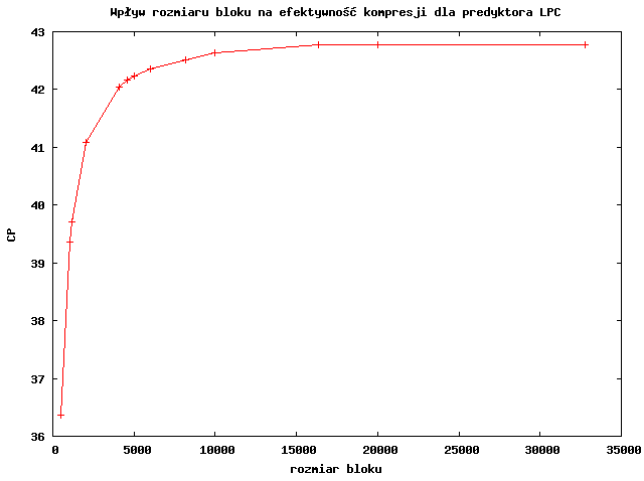
$$CP = (1 - \frac{1}{CR}) \cdot 100 \quad (\%)$$

Rozmiar bloku a efektywność kompresji

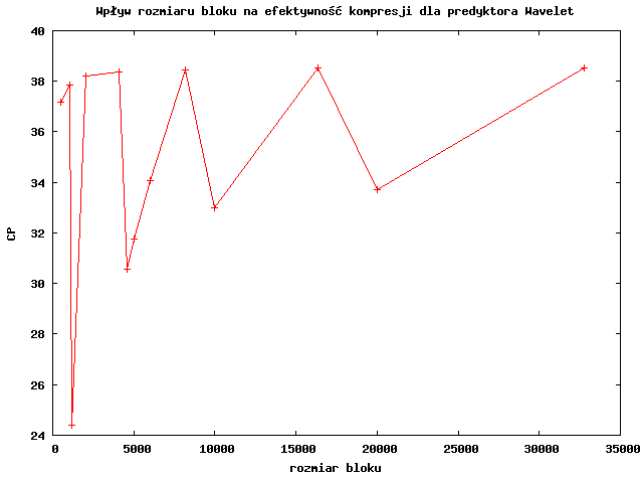
Test ten ma pokazać jak duży wpływ na efektywność kompresji ma rozmiar bloku. Zostanie on przeprowadzony dla różnych typów predyktorów.

Izrael – See I & I

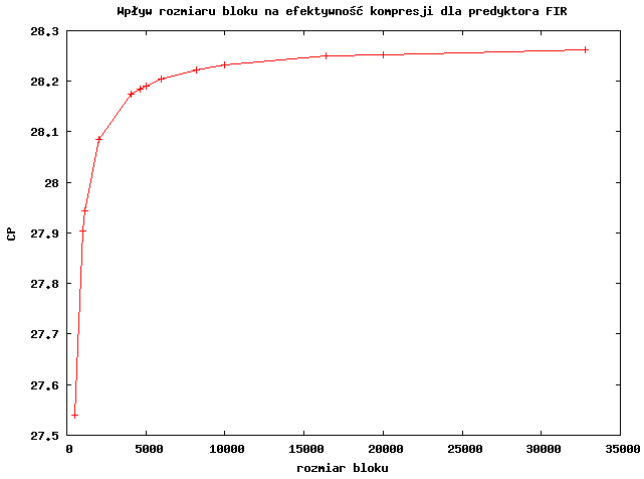


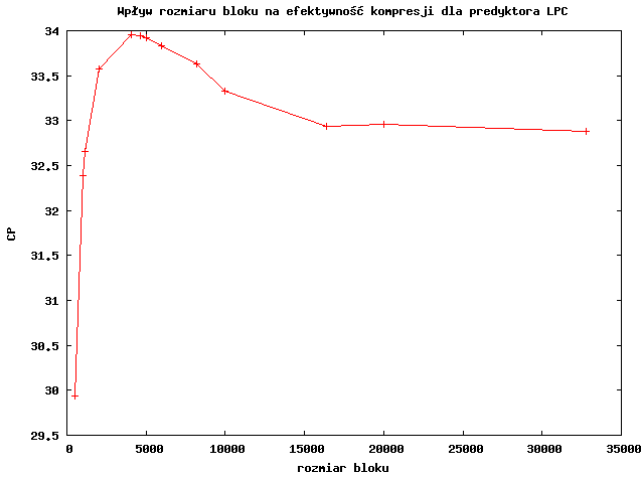


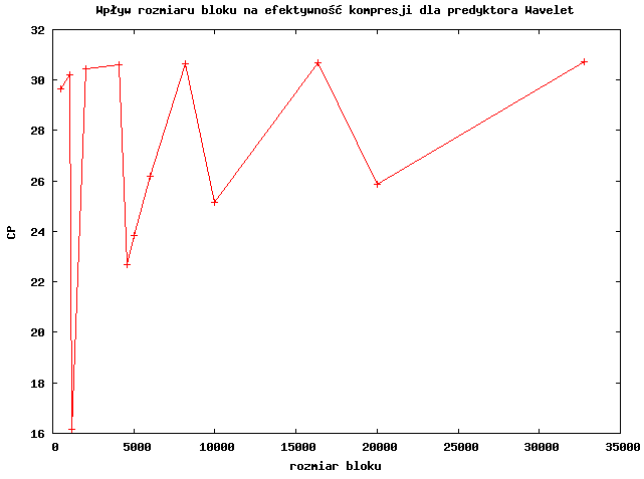
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>



Led Zeppelin – D'yer Mak'er

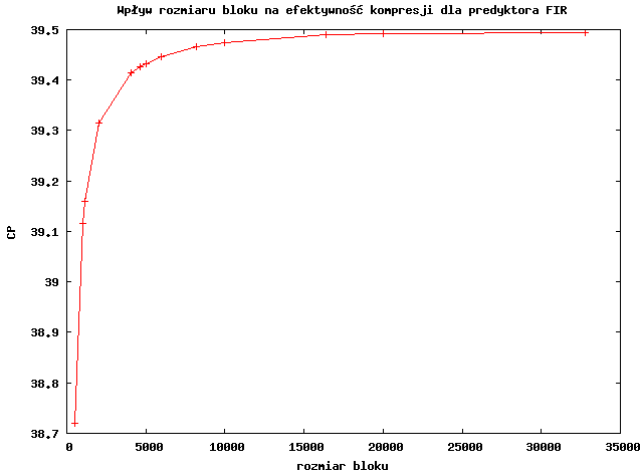


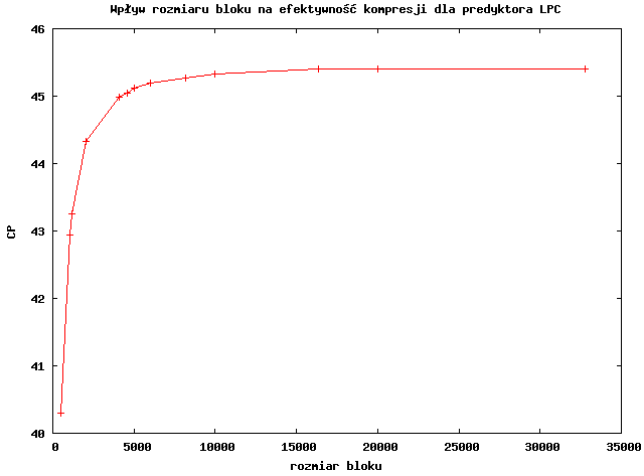


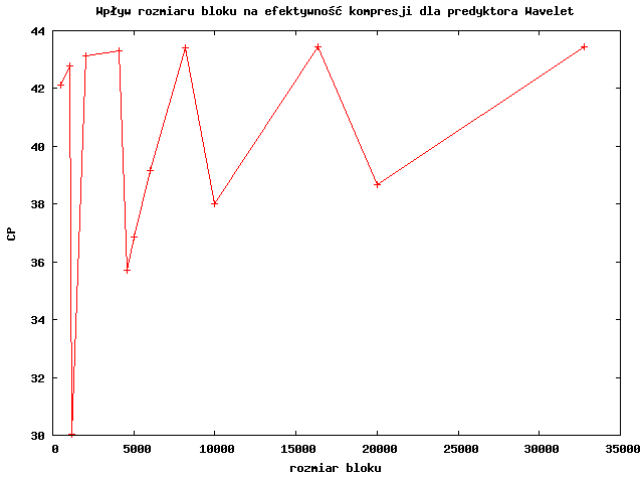


Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

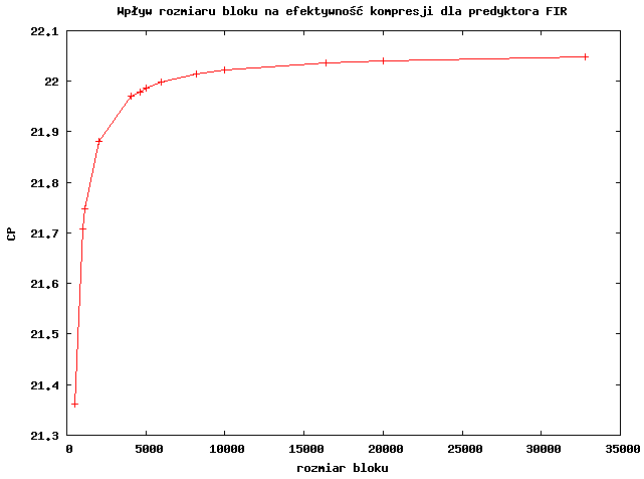
Gladiator OST – Now We Are Free

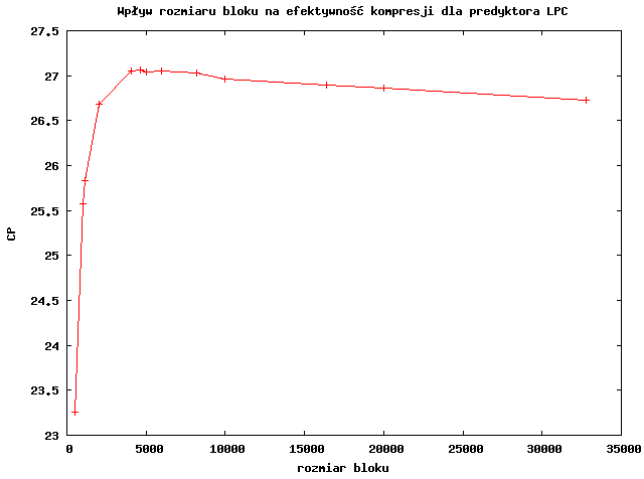




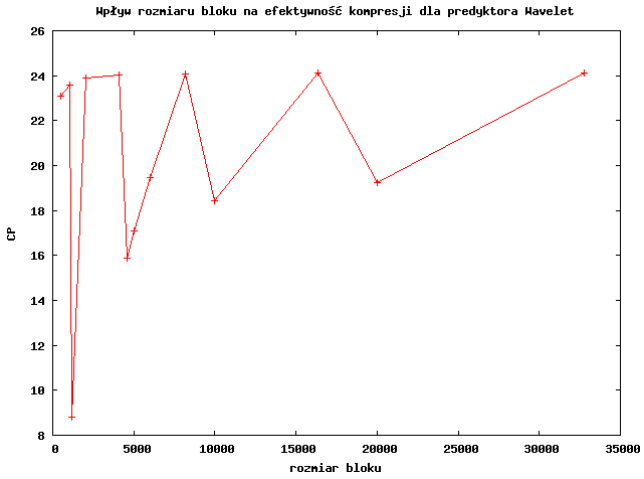


Sweet Noise – 9/1

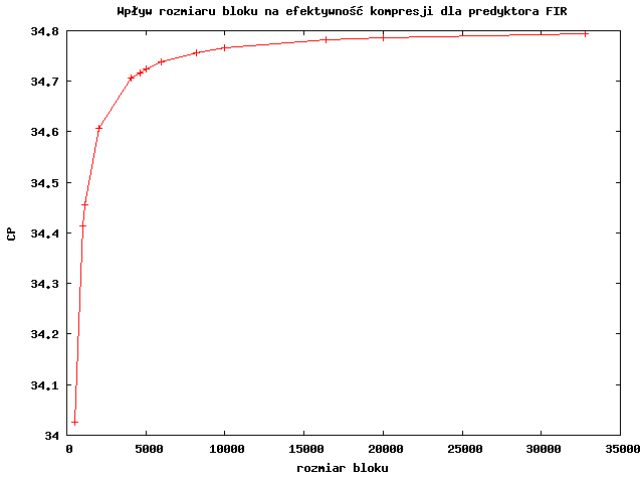


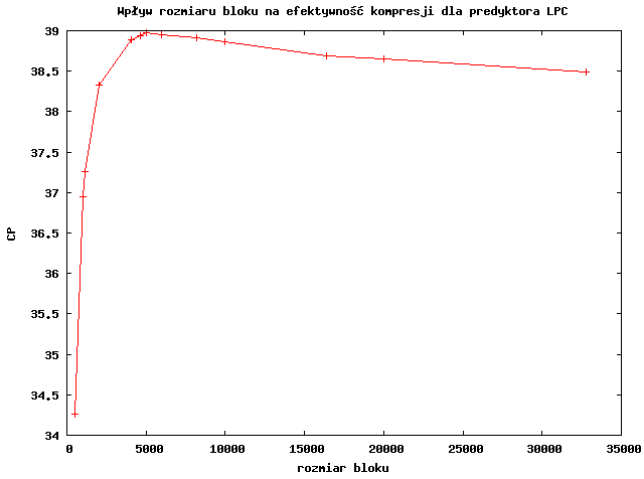


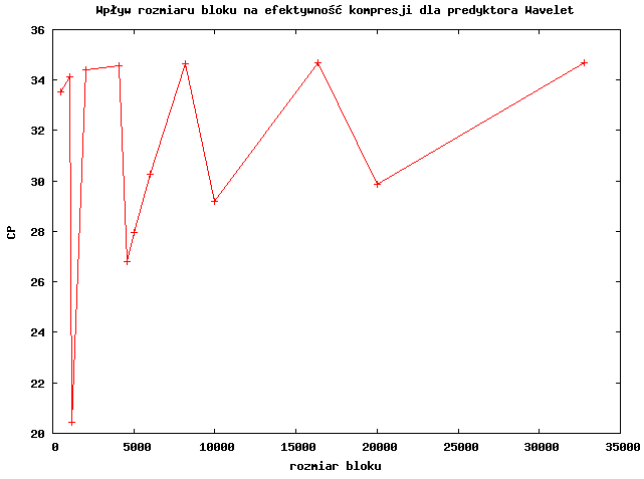
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>



Kult – Baranek

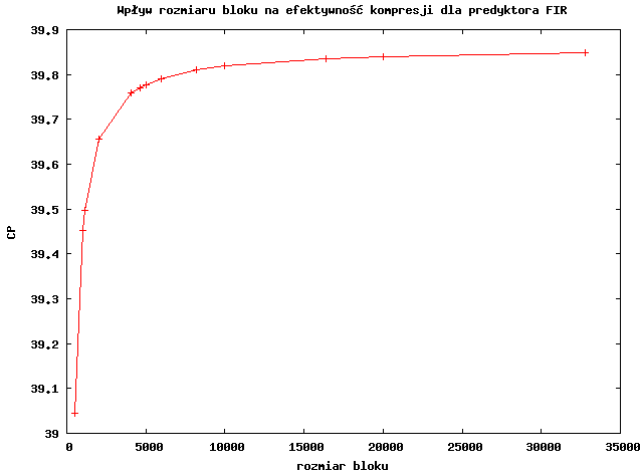


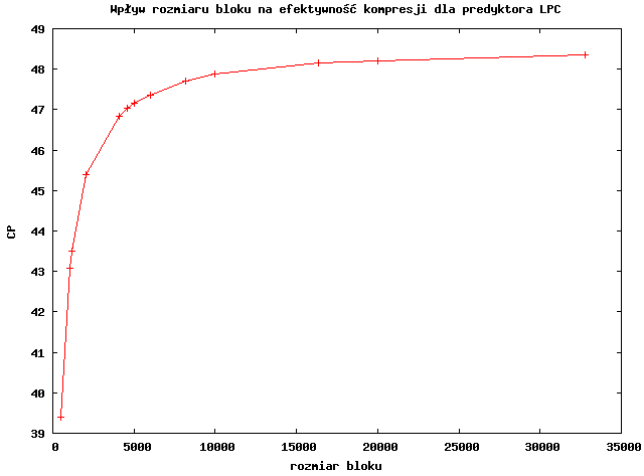


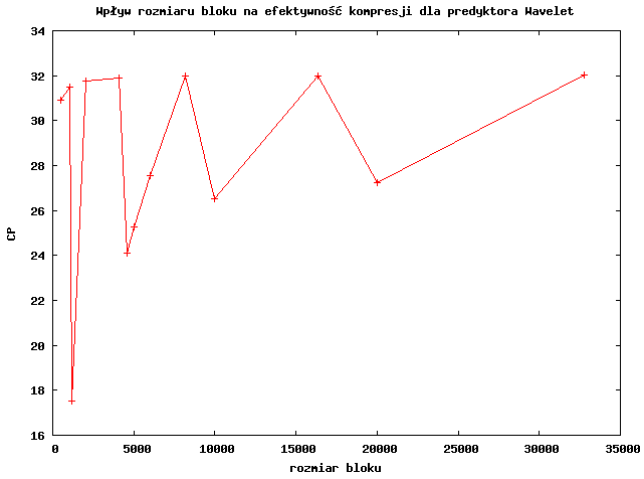


Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

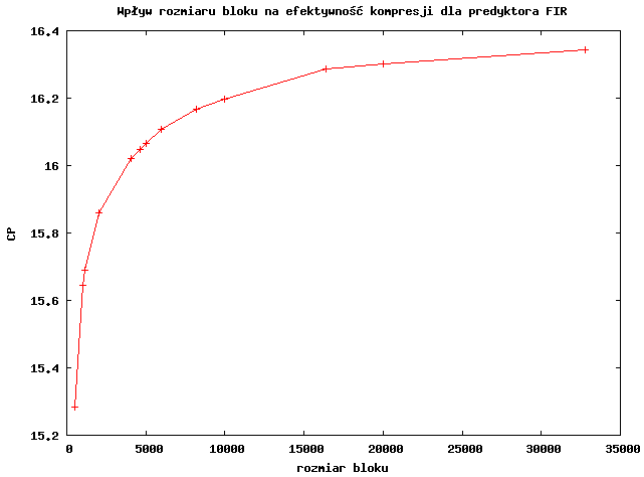
Myslovitz – Zgon (live)

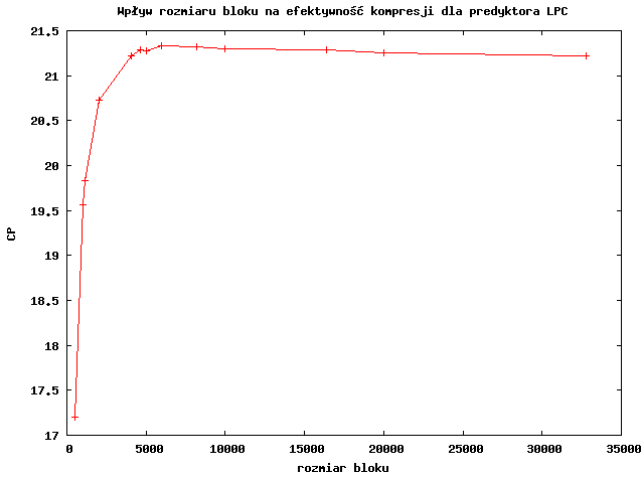




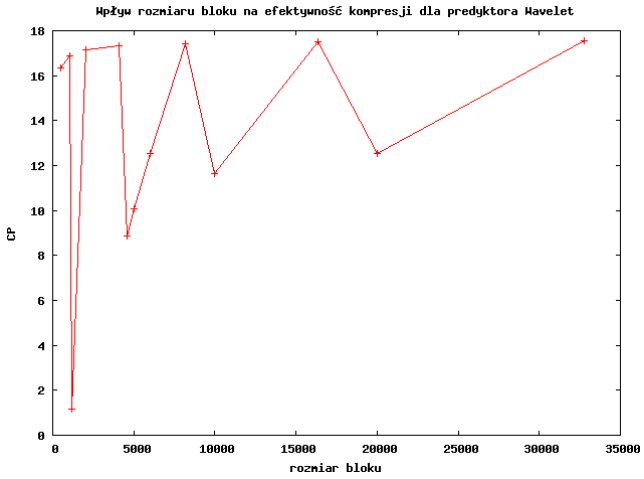


Vypsana Fixa – Palenie Titonia

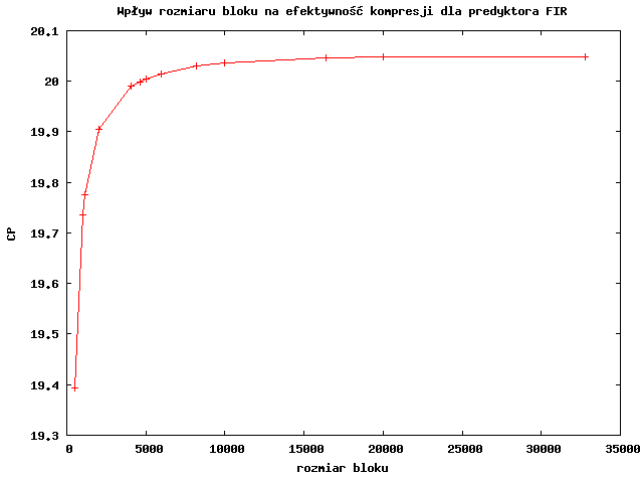


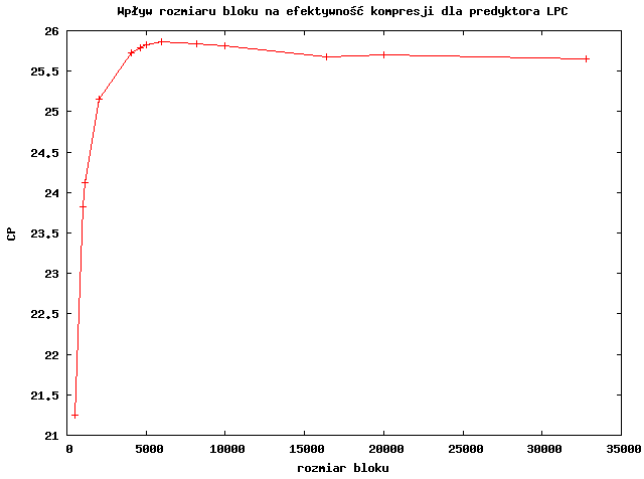


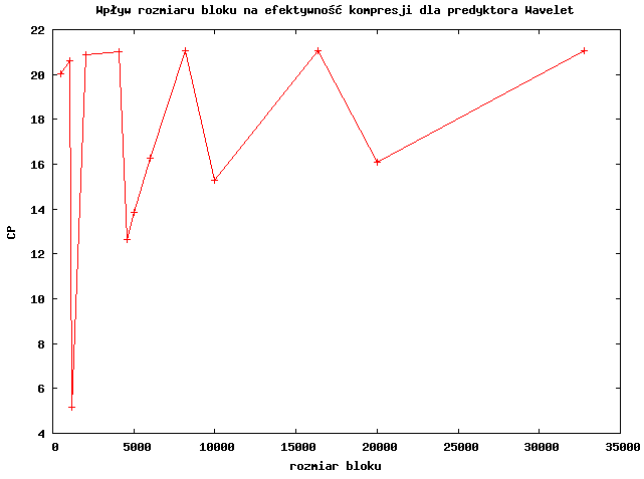
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>



Turbo – Szalony Ikar

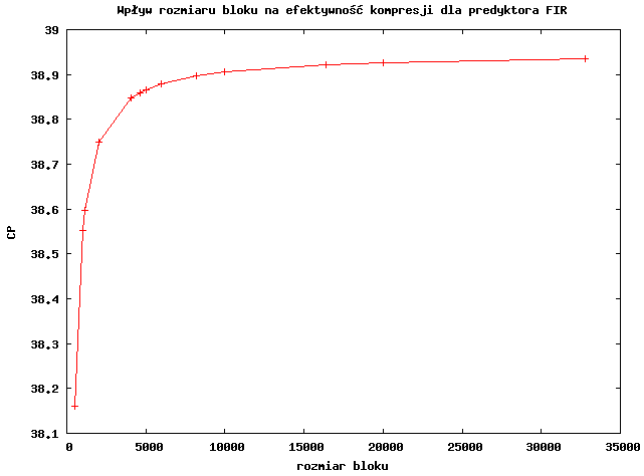


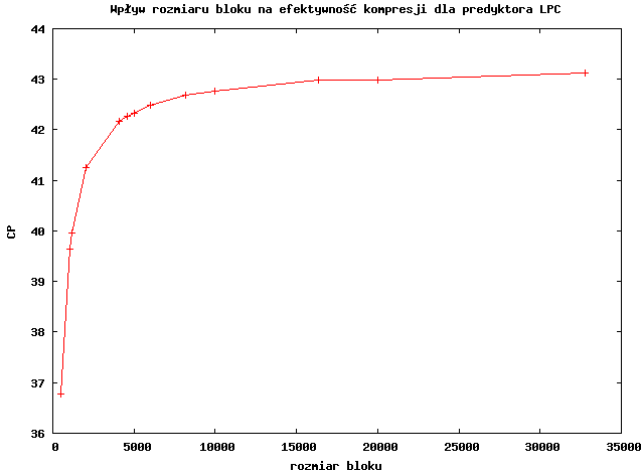


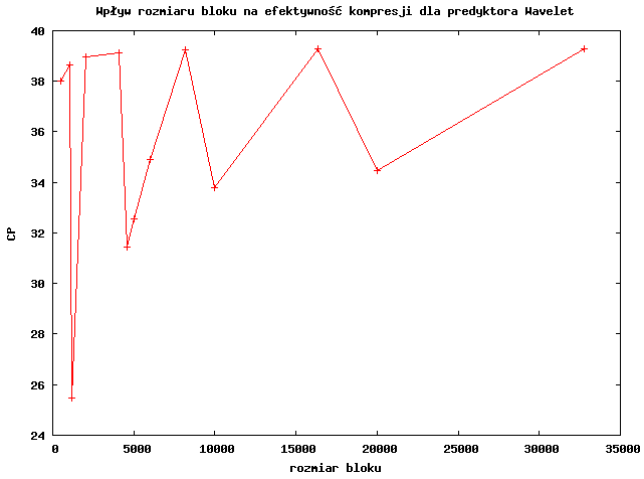


Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Iron Maiden – Sign Of The Cross







Wnioski

- FIR

Predyktor ten przewiduje wartości kolejnych próbek, przy użyciu cały czas tego samego wzoru, biorąc pod uwagę 3 ostatnie próbki z danego kanału. Dlatego wraz ze wzrostem długości ramki efektywność kompresji może się tylko polepszać. Jest tak ponieważ wraz ze wzrostem długości bloku nagłówki ramki coraz rzadziej występują w pliku wynikowym. W zasadzie najlepszym rozwiązaniem byłoby gdyby plik vlak składał się tylko z jednej ramki. Niestety jest to uniemożliwione przez górne ograniczenie długości bloku. Nie ma to jednak zbyt wielkiego znaczenia, gdyż zwykle dla długości bloku powyżej około 10000-15000 próbek wzrost efektywności jest już bardzo powolny (w przypadku CP zmiany na 5000 bajtów są rzędu jednej lub kilku dziesiątych procenta).

- LPC

Dobór kontekstu w przypadku LPC jest bardzo istotny, gdyż założeniem tego predyktora jest działanie na sygnale (prawie) stacjonarnym. Wyniki pomiarów wskazują, że optymalny rozmiar bloku wynosi około 5000 próbek na kanał dla większości rodzajów muzyki. Gdy jednak

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

muzyka jest „spokojniejsza” to kontekst może być znacznie dłuższy, a dzieje się tak np. dla reggae czy dla spokojnej muzyki klasycznej.

Różnice w efektywności LPC dla różnych rozmiarów bloków są znaczne, średnio jest to około 5%, ale bywa to nawet 8% (dla Myslovitz – Zgon). Wskazuje to na duże możliwości optymalizacji kodeka VLAK – jeśli udałoby się adaptacyjnie dobierać rozmiar bloku to można liczyć na znaczną poprawę efektywności.

- Wavelet

Przede wszystkim zastosowana przez nas transformata falkowa przyjmuje tylko ciągi o długości będącej potęgą dwójki. Aby zapewnić bezstratność kodeka także dla serii o innej długości zastosowaliśmy procedurę dopełniania długości serii do najbliżej dozwolonej wartości. Ciąg wyjściowy ma wtedy niestety długość owej potęgi dwójki, a efektywność kompresji jest zaniżona. To powoduje, że stosowanie innych rozmiarów bloku jest nieopłacalne. Stąd wzięły się na wykresie charakterystyczne zęby.

Prawdziwy wykres efektywności kompresji w funkcji długości bloku byłby wtedy zbiorem punktów o współrzędnej x należącej do zbioru potęg dwójki. Zależność tę można sobie uzmysłowić, wyobrażając sobie górną obwiednię powyższych wykresów dla kompresji falkowej. Wydaje się ona być rosnącą, choć od powyżej 8192 próbek zwiększenie rozmiaru o 2 powoduje wzrost CP o kilka dziesiątych części procenta.

Jednak na tych wykresach nie widać pewnego zjawiska (nie pozwalają na to wewnętrzne ograniczenia kodera), które zachodzi przy dzieleniu pliku wav na równe bloki. Zazwyczaj przy dzieleniu sampli tą metodą ostatnia ramka ma długość nie będącą potęgą dwójki i jest kompresowana nieefektywnie. Gdy wybrano bardzo duży rozmiar bloku, ta jedna ramka może popsuć efektywność kompresji całego pliku. Rozwiązaniem byłaby zmiana algorytmu dzielenia na bloki, tak aby ten ostatni blok był dodatkowo podzielony na kilka mniejszych, o np. o długościach będących kolejnymi największymi potęgami dwójki nie większymi niż pozostała do podziału długość. Nie zaimplementowaliśmy tego algorytmu w kodeku, gdyż ograniczenie długości bloku nie pozwala zaobserwować negatywnych objawów jego braku.

Komentarz dotyczący wpływu długości bloku na czas kompresji

W dodatku A znajdują się wyniki pomiaru czasu kompresji względem długości bloku.

- FIR

Algorytm ten przechodzi liniowo przez wartości wszystkich próbek i długość bloku praktycznie nie ma wpływu na jego czas działania (z dokładnością do czasu zapisania nagłówka każdej ramki).

- LPC

Analizując algorytm LPC dochodzi się do wniosku, iż złożoność obliczeniowa algorytmu Levinsona-Durbina zależy od iloczynu rozmiar bloku * rząd predyktora (jest to złożoność wyznaczenia macierzy autokorelacji sygnału) oraz od kwadratu rzędu predyktora (złożoność samego algorytmu).

Wyniki pomiarów pokazują, że dla małych rozmiarów bloków (rzędu 64 KB) czas działania algorytmu jest stały, zależny zapewne od czasu operacji dyskowych, przydziału pamięci a nie od złożoności obliczeniowej algorytmu. Jednak na wolniejszych maszynach

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

zależność od rozmiaru bloku może się ujawnić.

- Wavelet

Widzimy po pierwsze, że dla długości serii nie będących potęgami dwójki czas kompresji jest nienaturalnie długi. Po prostu zera dodawane przez nas sztucznie na koniec ciągu powodują, że dane takie stają się wyjątkowo "niewygodne" dla tej transformaty falkowej. Gdy odrzucimy długości bloku nie będącymi potęgami dwójki (których i tak nie stosuje się, z powodu słabej efektywności kompresji) dostaniemy prawdziwą zależność czasu kompresji od długości bloku. Widzimy, że jest ona rosnąca.

Oznaczmy:

1. n - długość bloku
2. k - liczbę ramek w pliku
3. l - długość pliku (w samplach liczona dla jednego kanału)

Złożoność obliczeniowa algorytmu falkowej interpolacji liniowej dla bloku o długości n wyraża się wzorem:

O(n) = n log (n)

Złożoność dla pliku o k ramkach i długości l (k*n = l):

O(n) = k*nlog(n) = l*log(n)

Widzimy, że czas kompresji rośnie logarytmicznie wraz ze wzrostem długości ramki.

Jak manipulatory wpływają na efektywność kompresji

Autorzy kodeka FLAC uważają, że bardzo istotnym etapem kodeka bezstratnego jest dekorelacja międzykanałowa. Uznali oni, że wystarczą do tego celu bardzo proste środki, przykładowo: konwersja left,right -> mid,side. Test porównuje efektywność kodeka dla przypadku zastosowania takiej konwersji (tzw. manipulacji) oraz bez niej.

Wszystkie wyniki uzyskano wywołując program vlak z następującymi opcjami:

`vlak -p 1 -l 8,8 -b 4608 -m 0` oraz `vlak -p 1 -l 8,8 -b 4608 -m 2`

Izrael – See I & I

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*13*	*1.7163*	*41.7352*
AdaptiveMidSide	*14*	*1.7289*	*42.1605*

Led Zeppelin – D'yer Mak'er

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*12*	*1.5034*	*33.4820*
AdaptiveMidSide	*13*	*1.5138*	*33.9410*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Gladiator OST – Now We Are Free

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*12*	*1.7935*	*44.2441*
AdaptiveMidSide	*12*	*1.8199*	*45.0506*

Sweet Noise – 9/1

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*12*	*1.3580*	*26.3602*
AdaptiveMidSide	*12*	*1.3711*	*27.0642*

Kult – Baranek

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*12*	*1.6305*	*38.6707*
AdaptiveMidSide	*13*	*1.6378*	*38.9420*

Myslovitz – Zgon (live)

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*7*	*1.8497*	*45.9384*
AdaptiveMidSide	*7*	*1.8882*	*47.0386*

Vypsana Fixa – Palenie Titonia

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*9*	*1.2739*	*21.5006*
AdaptiveMidSide	*10*	*1.2704*	*21.2872*

Turbo – Szalony Ikar

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*10*	*1.3461*	*25.7137*
AdaptiveMidSide	*11*	*1.3475*	*25.7906*

Iron Maiden – Sign Of The Cross

manipulator	*czas kompresji*	*CR*	*CP*
bez manipulacji	*33*	*1.7035*	*41.2967*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

manipulator	*czas kompresji*	*CR*	*CP*
AdaptiveMidSide	*38*	*1.7323*	*42.2721*

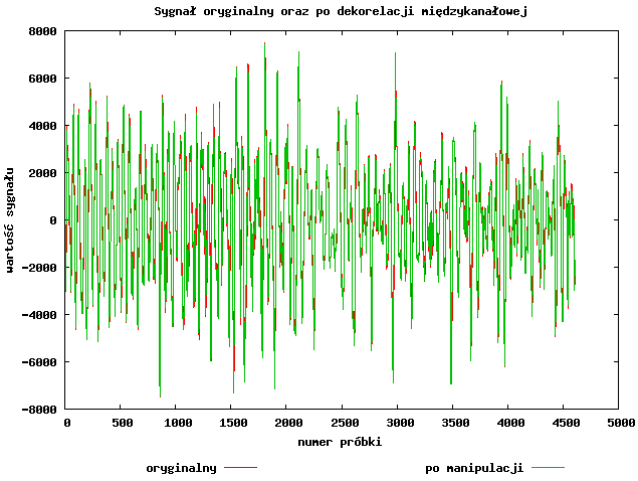
Wnioski

Okazuje się, że taka prosta próba dekorelacji sygnału daje raczej słabe wyniki. Zysk jest w granicach około 1%. Znalazł się nawet przypadek (Vypsana Fixa) dla którego MidSide pogarsza efektywność kompresji! Z drugiej strony – czas potrzebny na kompresję z wykorzystaniem manipulatora MidSide jest niewiele dłuższy niż bez niego, a zysk jest niezerowy. Dlatego też taka manipulacja mimo wszystko może mieć sens.

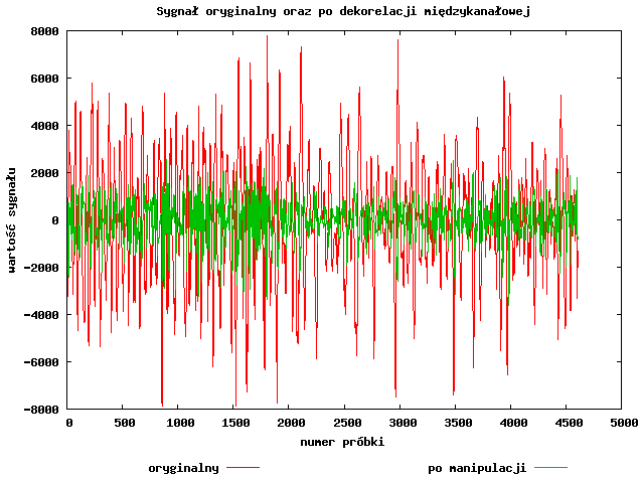
Istnieje jednak sytuacja w której MidSide będzie bardzo dobrym pomysłem. Są to tzw. pliki fake-stereo, czyli takie w których lewy i prawy kanał są identyczne. Zastosowanie manipulatora MidSide spowoduje iż kanał Side będzie miał stale wartość 0, co oczywiście pozwoli na znacznie bardziej efektywną predykcję i kompresję.[9]

Można spróbować usprawnić proces dekorelacji międzykanałowej dodając większą ilość prostych manipulatorów oraz wybierać dla każdego bloku najlepszy z nich. Takie podejście zastosowano w kodeku FLAC.

Przykład skutecznej dekorelacji międzykanałowej sygnału



kanał „mid”



kanał „side”

Efektywność kompresji dla różnych typów predyktorów oraz porównanie z konkurencją

Test ten stanowi porównanie kodeka VLAK w różnych trybach pracy z konkurencyjnymi kodekami. Porównanie wykonano z następującymi kodekami:

- FLAC
- SHORTEN
- Monkey's Audio
- Gzip

9 predyktor LPC potrafi wykryć sygnał zerowy, a wtedy w ramce sygnał rezydualny nie jest zapisywany

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

- Bzip2

Izrael – See I & I

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*13*	*1.6740*	*40.2638*	*9*
vlak -p 2 -m 2 -b 4608	*12*	*1.6594*	*39.7377*	*12*
vlak -p 3 -m 2 -b 4096	*56*	*1.6229*	*38.3804*	*56*
flac -0	*2*	*1.6796*	*40.4635*	*1*
flac -5	*5*	*1.7646*	*43.3289*	*1*
flac -8	*47*	*1.7702*	*43.5104*	*1*
shorten	*2*	*1.6888*	*40.7854*	*1*
shorten -b 4608 -p 8	*4*	*1.6409*	*39.0579*	*1*
mac -c2000	*8*	*1.8518*	*45.9982*	*11*
mac -c3000	*10*	*1.8568*	*46.1450*	*13*
gzip	*5*	*1.0729*	*6.7957*	*1*
bzip2	*35*	*1.1146*	*10.2790*	*14*

Led Zeppelin – D'yer Mak'er

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*12*	*1.4741*	*32.1627*	*9*
vlak -p 2 -m 2 -b 4608	*11*	*1.3925*	*28.1841*	*11*
vlak -p 3 -m 2 -b 4096	*52*	*1.4407*	*30.5904*	*52*
flac -0	*2*	*1.4455*	*30.8181*	*1*
flac -5	*5*	*1.5254*	*34.4453*	*1*
flac -8	*43*	*1.5303*	*34.6528*	*1*
shorten	*2*	*1.4506*	*31.0618*	*1*
shorten -b 4608 -p 8	*4*	*1.4981*	*33.2493*	*1*
mac -c2000	*7*	*1.5814*	*36.7651*	*11*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
mac -c3000	*9*	*1.5869*	*36.9843*	*12*
gzip	*4*	*1.0901*	*8.2646*	*1*
bzip2	*33*	*1.1430*	*12.5104*	*13*

Gladiator OST – Now We Are Free

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*11*	*1.7625*	*43.2635*	*8*
vlak -p 2 -m 2 -b 4608	*11*	*1.6509*	*39.4254*	*10*
vlak -p 3 -m 2 -b 4096	*50*	*1.7639*	*43.3089*	*50*
flac -0	*2*	*1.7372*	*42.4355*	*1*
flac -5	*5*	*1.8396*	*45.6389*	*1*
flac -8	*41*	*1.8464*	*45.8410*	*1*
shorten	*2*	*1.7451*	*42.6975*	*1*
shorten -b 4608 -p 8	*4*	*1.7761*	*43.6984*	*1*
mac -c2000	*7*	*1.9041*	*47.4830*	*10*
mac -c3000	*9*	*1.9170*	*47.8351*	*11*
gzip	*5*	*1.1260*	*11.1908*	*1*
bzip2	*29*	*1.1897*	*15.9479*	*12*

Sweet Noise – 9/1

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*11*	*1.3445*	*25.6234*	*8*
vlak -p 2 -m 2 -b 4608	*10*	*1.2817*	*21.9795*	*18*
vlak -p 3 -m 2 -b 4096	*49*	*1.3161*	*24.0164*	*50*
flac -0	*2*	*1.3180*	*24.1279*	*1*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
flac -5	*5*	*1.3811*	*27.5921*	*1*
flac -8	*41*	*1.3839*	*27.7399*	*1*
shorten	*2*	*1.3221*	*24.3624*	*1*
shorten -b 4608 -p 8	*4*	*1.3594*	*26.4359*	*1*
mac -c2000	*7*	*1.4160*	*29.3768*	*10*
mac -c3000	*8*	*1.4216*	*29.6589*	*11*
gzip	*5*	*1.0668*	*6.2578*	*1*
bzip2	*32*	*1.0903*	*8.2829*	*12*

Kult – Baranek

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*12*	*1.5927*	*37.2145*	*9*
vlak -p 2 -m 2 -b 4608	*11*	*1.5318*	*34.7171*	*11*
vlak -p 3 -m 2 -b 4096	*52*	*1.5282*	*34.5634*	*52*
flac -0	*2*	*1.5652*	*36.1088*	*1*
flac -5	*5*	*1.6493*	*39.3692*	*1*
flac -8	*43*	*1.6568*	*39.6437*	*1*
shorten	*2*	*1.5716*	*36.3715*	*1*
shorten -b 4608 -p 8	*4*	*1.6149*	*38.0776*	*1*
mac -c2000	*7*	*1.7806*	*43.8395*	*11*
mac -c3000	*9*	*1.7980*	*44.3818*	*12*
gzip	*5*	*1.1109*	*9.9813*	*1*
bzip2	*30*	*1.1717*	*14.6553*	*13*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Myslovitz – Zgon (live)

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*7*	*1.8355*	*45.5195*	*5*
vlak -p 2 -m 2 -b 4608	*6*	*1.6603*	*39.7695*	*6*
vlak -p 3 -m 2 -b 4096	*30*	*1.4683*	*31.8943*	*30*
flac -0	*1*	*1.7131*	*41.6272*	*1*
flac -5	*3*	*1.8946*	*47.2181*	*1*
flac -8	*24*	*1.9050*	*47.5056*	*1*
shorten	*1*	*1.6508*	*39.4238*	*0*
shorten -b 4608 -p 8	*2*	*1.7935*	*44.2418*	*1*
mac -c2000	*4*	*2.0855*	*52.0490*	*6*
mac -c3000	*5*	*2.0893*	*52.1366*	*7*
gzip	*2*	*1.0933*	*8.5343*	*0*
bzip2	*17*	*1.2349*	*19.0233*	*7*

Vypsana Fixa – Palenie Titonia

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*10*	*1.2409*	*19.4149*	*7*
vlak -p 2 -m 2 -b 4608	*8*	*1.1912*	*16.0491*	*9*
vlak -p 3 -m 2 -b 4096	*42*	*1.2094*	*17.3112*	*42*
flac -0	*2*	*1.2270*	*18.4999*	*1*
flac -5	*4*	*1.2848*	*22.1688*	*1*
flac -8	*36*	*1.2889*	*22.4117*	*1*
shorten	*2*	*1.2302*	*18.7095*	*1*
shorten -b 4608 -p 8	*3*	*1.2763*	*21.6496*	*1*
mac -c2000	*6*	*1.3394*	*25.3407*	*9*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
mac -c3000	*8*	*1.3408*	*25.4153*	*10*
gzip	*4*	*1.0208*	*2.0333*	*0*
bzip2	*30*	*1.0184*	*1.8060*	*11*

Turbo – Szalony Ikar

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*9*	*1.3066*	*23.4653*	*7*
vlak -p 2 -m 2 -b 4608	*9*	*1.2500*	*19.9991*	*9*
vlak -p 3 -m 2 -b 4096	*43*	*1.2658*	*20.9992*	*44*
flac -0	*2*	*1.2850*	*22.1808*	*1*
flac -5	*4*	*1.3580*	*26.3599*	*1*
flac -8	*36*	*1.3605*	*26.4963*	*1*
shorten	*2*	*1.2892*	*22.4299*	*1*
shorten -b 4608 -p 8	*3*	*1.3473*	*25.7763*	*1*
mac -c2000	*6*	*1.3955*	*28.3390*	*9*
mac -c3000	*8*	*1.3975*	*28.4444*	*10*
gzip	*4*	*1.0338*	*3.2664*	*0*
bzip2	*30*	*1.0334*	*3.2308*	*11*

Iron Maiden – Sign Of The Cross

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
vlak -p 1 -m 2 -b 4608	*32*	*1.6819*	*40.5449*	*23*
vlak -p 2 -m 2 -b 4608	*29*	*1.6356*	*38.8592*	*29*
vlak -p 3 -m 2 -b 4096	*135*	*1.6431*	*39.1411*	*135*
flac -0	*6*	*1.6770*	*40.3701*	*4*
flac -5	*13*	*1.7604*	*43.1939*	*4*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

parametry wywołania	*czas kompresji*	*CR*	*CP*	*czas dekompresji*
flac -8	*111*	*1.7642*	*43.3180*	*4*
shorten	*6*	*1.6834*	*40.5978*	*3*
shorten -b 4608 -p 8	*11*	*1.6557*	*39.6042*	*4*
mac -c2000	*20*	*1.8210*	*45.0857*	*28*
mac -c3000	*23*	*1.8264*	*45.2483*	*31*
gzip	*13*	*1.1075*	*9.7032*	*3*
bzip2	*84*	*1.1764*	*14.9937*	*34*

Wnioski

Porównanie działania predyktorów wbudowanych we VLAKa jednoznacznie wskazuje, iż najlepszy z nich jest predyktor LPC (-p 1) (mimo zastosowania w powyższych testach metody adaptacyjnej, która – jak pokaże następny test – jest daleka od ideału). Daje on najwyższą efektywność kompresji a także, co ciekawe, najkrótszy czas dekompresji, zachowując czas kompresji na stosunkowo niskim poziomie.

Predyktor Wavelet (-p 3) wyraźnie zawodzi. Nie dość, że kompresja i dekompresja są bardzo wolne to efektywność jest niska – często niższa nawet od efektywności prostego predyktora FIR.

Predyktor FIR (-p 2) uzyskuje umiarkowane rezultaty. Można powiedzieć że jego efektywność jest wystarczająca, szczególnie gdy weźmiemy pod uwagę prostotę tego rozwiązania i minimalne nakłady obliczeniowe. FIR do kompresji nie potrzebuje FPU, tak więc może być używany w prostych urządzeniach przenośnych, np. dyktafonach (np. na kasety DAT).

Porównanie VLAKa z konkurencją nie wypada już tak dobrze. Oczywiście uzyskujemy znacznie lepsze wyniki niż standardowe archiwizery jak gzip czy bzip2 natomiast dedykowane rozwiązania dostępne na rynku uzyskują lepsze rezultaty.

VLAK może dawać minimalnie lepsze rezultaty niż SHORTEN. Jednak SHORTEN jest znacznie szybszy. Nie powinno to nikogo dziwić. VLAK nie został zoptymalizowany pod kątem szybkości działania, chodziło nam raczej o czytelność kodu źródłowego oraz prostotę dodawania nowych predyktorów. Natomiast SHORTEN jest już dosyć wiekowym kodekiem - rozwijany jest od 1992 roku, tak więc na pewno został porządnie zoptymalizowany pod tym kątem.

Kodek FLAC okazuje się nieznacznie (tzn. o ok. 2-3%) bardziej efektywny od naszego kodeka, jest także trochę szybszy. Należy zwrócić jednak uwagę że FLAC jest dużo bardziej rozbudowany – składa się z 2 predyktorów: FIR i LPC, ma większą ilość manipulatorów a przede wszystkim dzieli sygnał wejściowy na mniejsze porcje, zwane subframe'ami. Pozwala to wybrać dla każdego kanału inny predyktor. Dodatkowo FLAC potrafi adaptacyjnie dobierać predyktor oraz jego rząd. Poza tym zastosowany we FLACu koder entropii (oparty na koderze Rice'a) stosuje zaawansowaną metodę adaptacji.

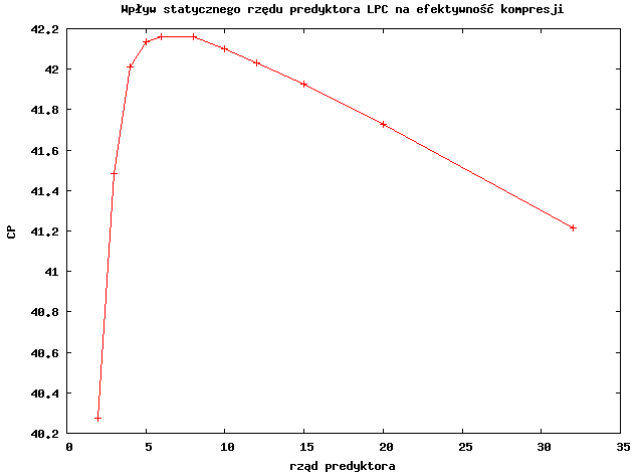
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

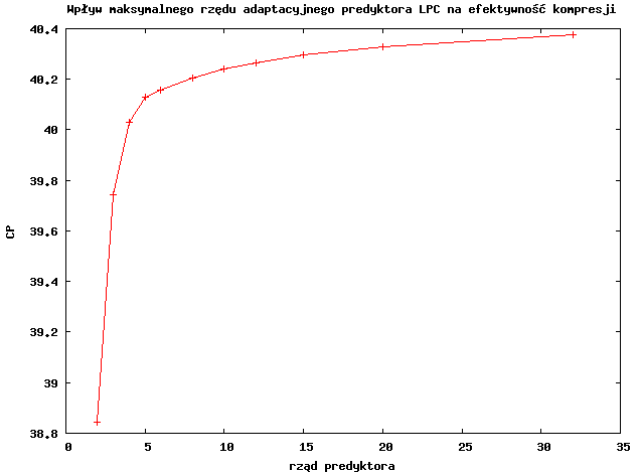
Monkey's Audio jest jednym z najlepszych obecnie kodeków bezstratnych. Zastosowany w nim schemat kompresji znacznie przewyższa wszystkie inne kodeki audio. Jedyną wadą Monkey's Audio jest stosunkowo długi czas dekompresji, dłuższy niż czas kompresji!

Efektywność predyktora LPC dla metody adaptacyjnej i nieadaptacyjnej

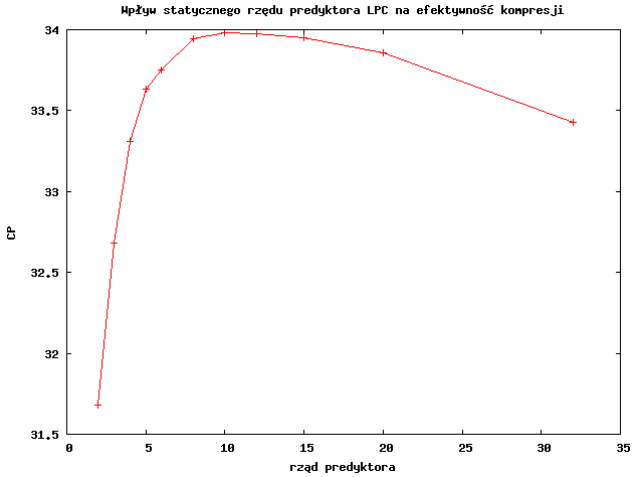
Jest to de facto sprawdzenie jak dobrze radzi sobie metoda adaptacyjna w porównaniu z metodą z wyborem statycznego rzędu predyktora.

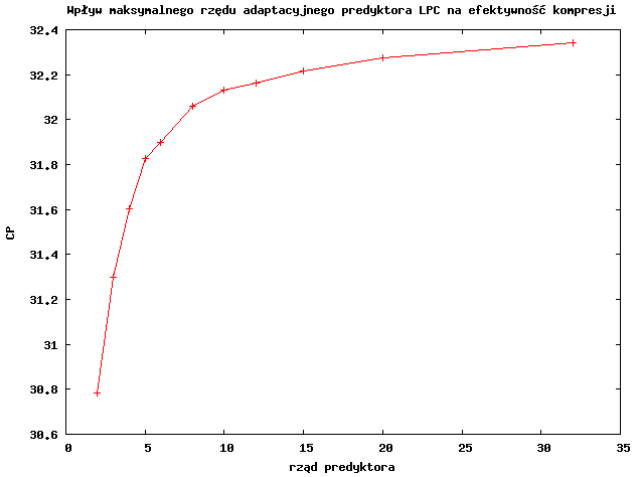
Izrael – See I & I





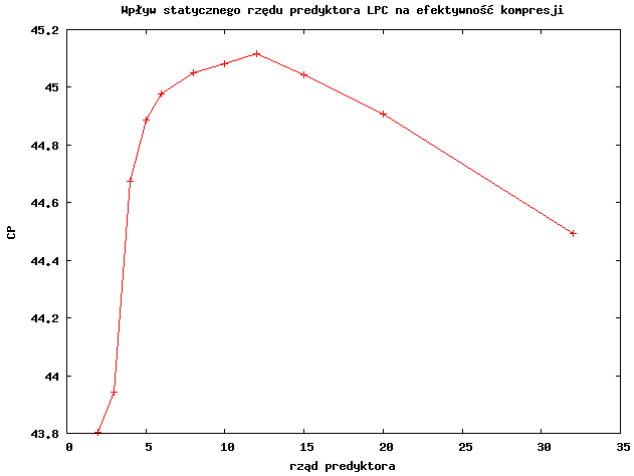
Led Zeppelin – D'yer Mak'er

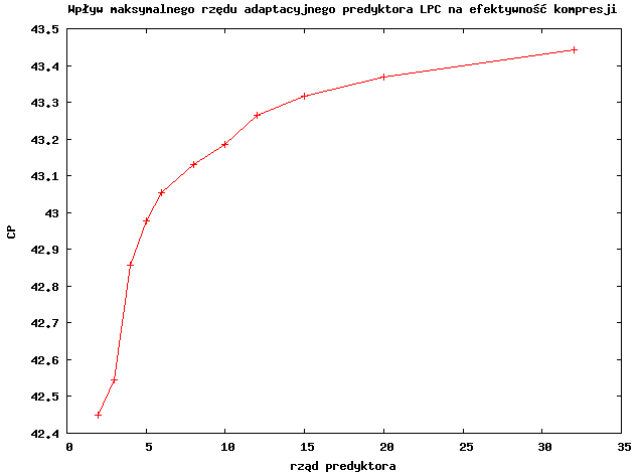




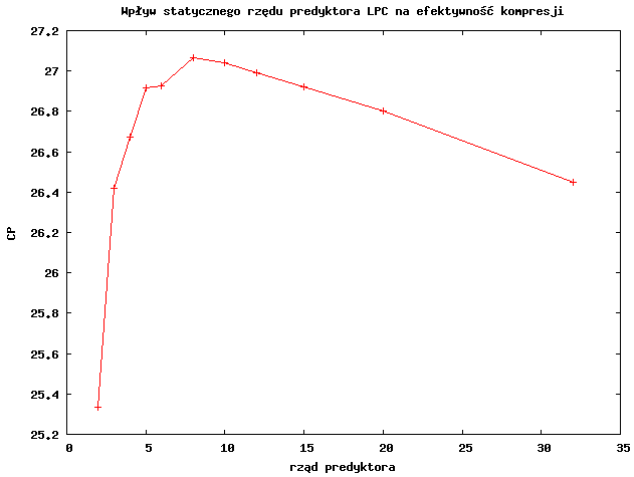
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

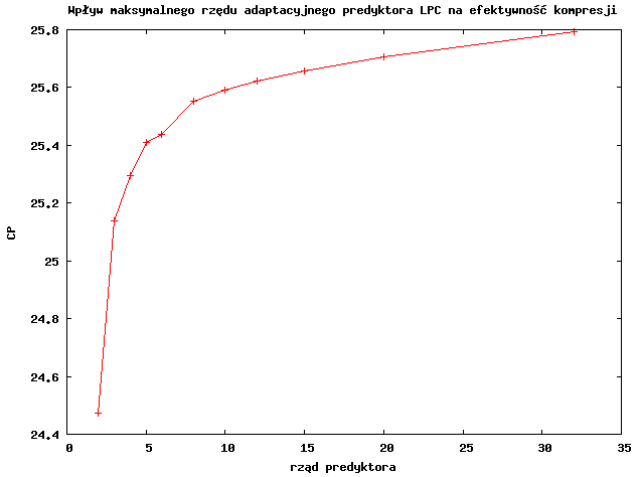
Gladiator OST – Now We Are Free





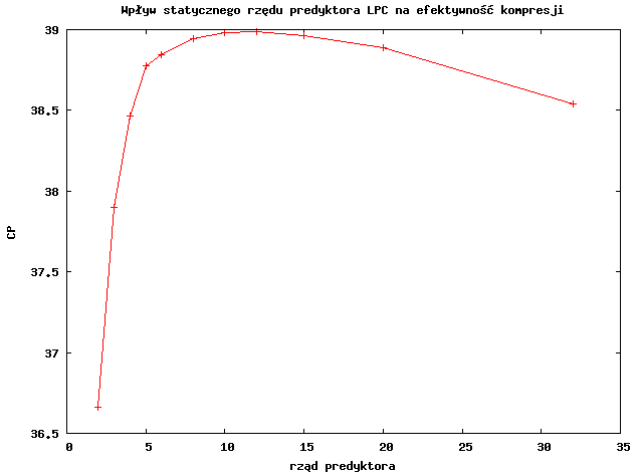
Sweet Noise – 9/1

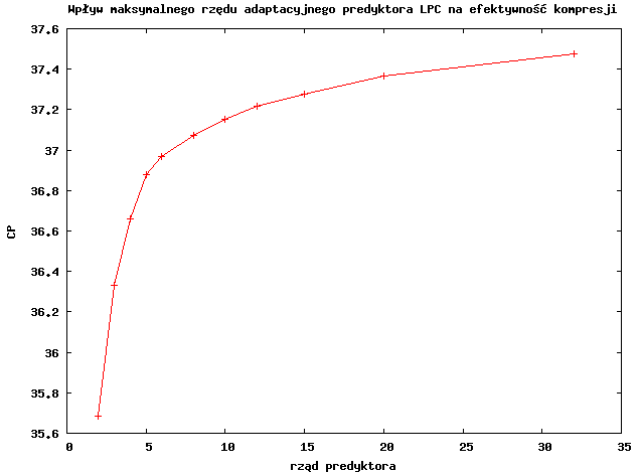




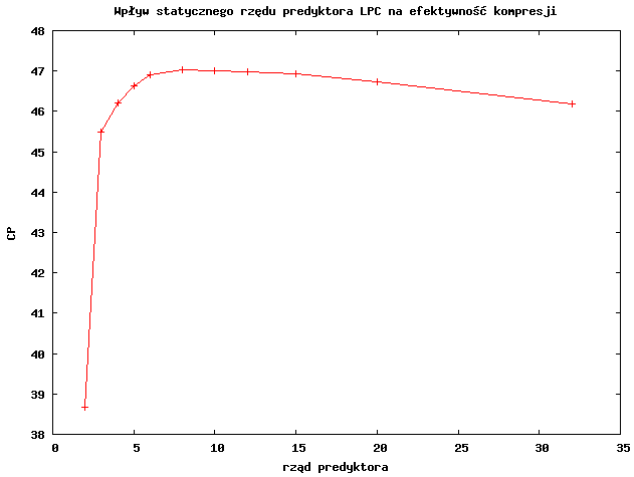
Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

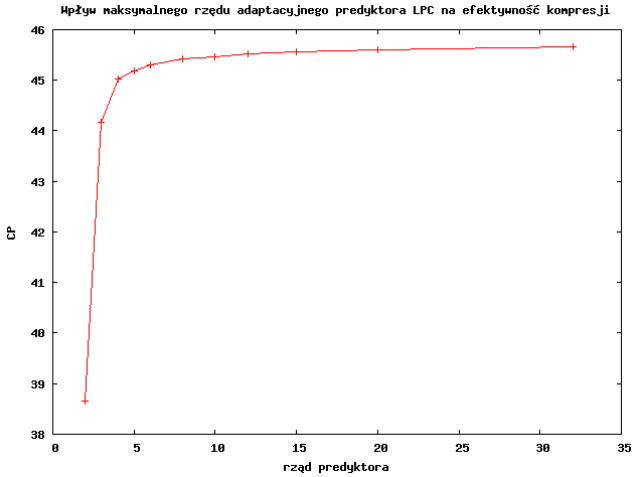
Kult – Baranek

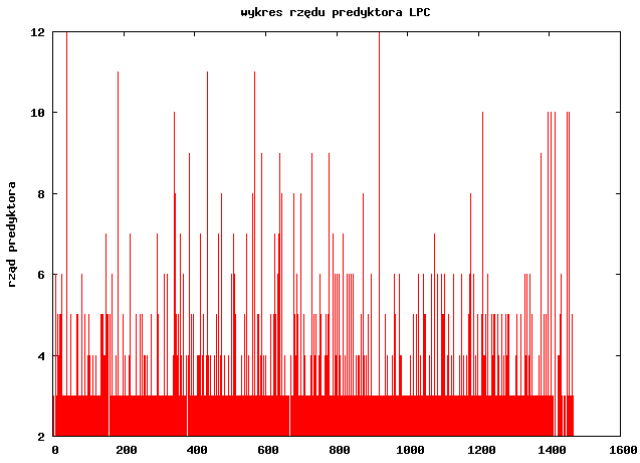


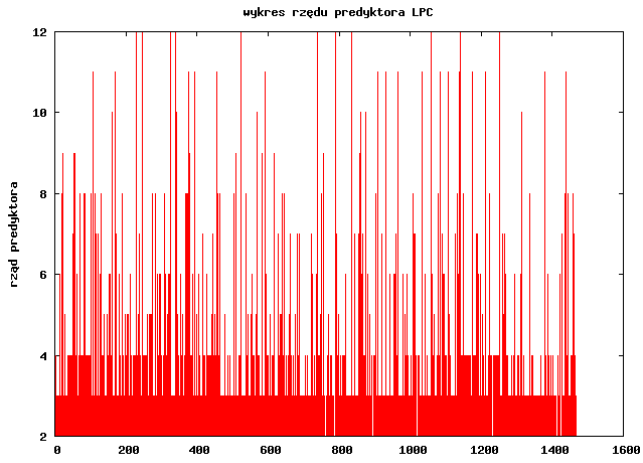


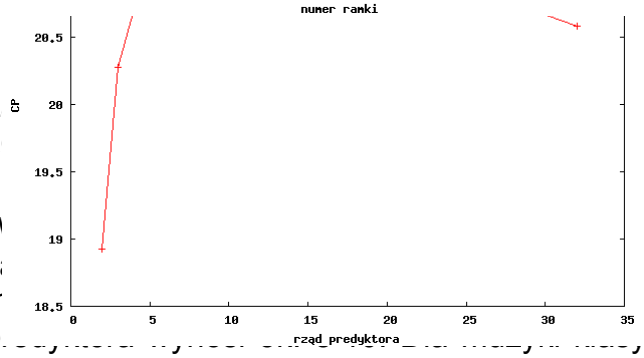
Myslovitz – Zgon (live)

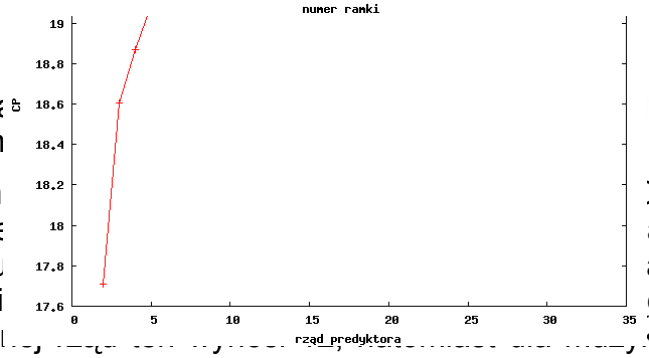






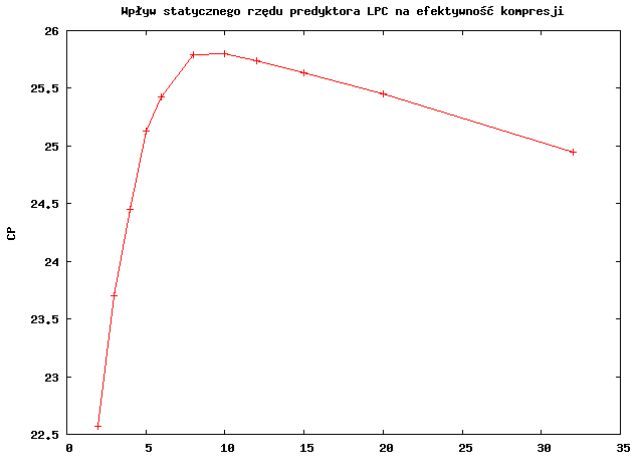


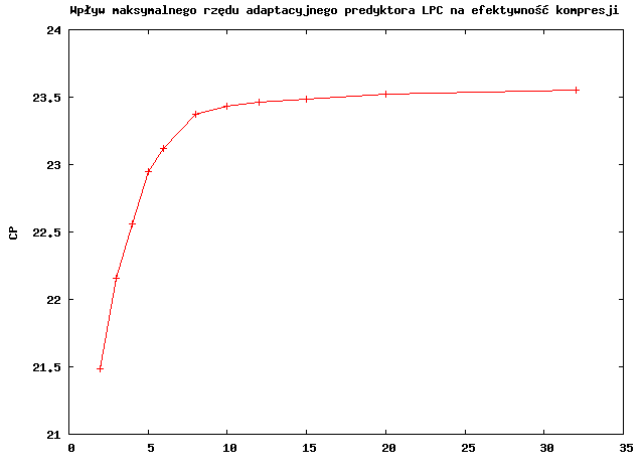




reggae tylko ok. 5. Potwierdza to konieczność stosowania metod adaptacyjnych. Jeśli chcemy stosować rząd statyczny to najlepszy będzie rząd równy 8, chociaż jest to tylko wskazówka.

Turbo – Szalony Ikar





do wniosku, że kodeki te mogą mieć praktyczne zastosowanie, tak jak to zostało wskazane na wstępie niniejszego sprawozdania.

Iron Maiden – Sign Of The Cross

Zastosowane predyktory FIR i LPC są standardem w bezstratnej kompresji dźwięku. Natomiast z metodami bazującymi na FFT oraz falkową nie spotkaliśmy się w żadnym innym kodeku. Wyniki pomiarów pokazują dlaczego tak jest.

Kompresja falkowa jest bardzo wolna w porównaniu z pozostałymi metodami, rezultaty działania są często gorsze niż prostego predyktora FIR. Możliwe, że pozwalałaby osiągnąć najlepszy stopień kompresji, gdyby zastosować kodowanie entropii inne niż algorytm Rice'a. Biorąc jednak pod uwagę niesamowicie długi czas kompresji i dekompresji z wykorzystaniem kompresji falkowej dochodzimy do wniosku, że jej zastosowanie mija się z celem.

Kodek VLAK może zostać znacznie ulepszony, gdyż praktycznie na każdym etapie działania można próbować bardziej wyszukanych pomysłów:

10 Myslovitz – Zgon (live)

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

- na etapie podziału na bloki bardzo dobrym rozwiązaniem byłoby adaptacyjne dobieranie rozmiaru bloku,
- na etapie dekorelacji międzykanałowej – należy dodać większą ilość tzw. manipulatorów oraz wybierać który z nich najlepiej dekoreluje sygnał,
- na etapie predykcji – stosowanie bardziej złożonych predyktorów niż LPC wydaje się nie mieć większego sensu. Proponowane usprawnienie polega na ulepszeniu adaptacyjności doboru rzędu tego predyktora,
- na etapie kompresji sygnału rezydualnego należy spróbować wykorzystać inne algorytmy doboru parametru k dla kodera Rice'a, można też zastosować zupełnie inny koder, np. szybki koder arytmetyczny.

Linki

Poniżej przedstawiamy zbiór interesujących odnośników do zasobów WWW związanych z tematyką bezstratnego kodowania dźwięku.

Testy wydajnościowe

http://flac.sourceforge.net/comparison.html

http://www.monkeysaudio.com/comparison.html

http://web.inter.nl.net/users/hvdh/lossless/lossless.htm

http://foobar2000.net/lossless/

http://members.home.nl/w.speek/comparison.htm

http://www.extremetech.com/article2/0,1558,1560783,00.asp

Teoria

http://www.firstpr.com.au/audiocomp/lossless/#rice

http://www.bearcave.com/misl/misl_tech/wavelets/packet/index.html

http://www.monkeysaudio.com/theory.html

http://en.wikipedia.org/wiki/Linear_prediction

http://cnx.rice.edu/content/m10482/latest/

http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf

http://www.cs.tut.fi/~rezaei/Lossless%20Audio%20Coding.pdf

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Dodatek A: dane wejściowe dla wykresów

rozmiary bloków

Izrael – See I & I

- FIR

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*12*	*1.6401*	*39.0282*
1024	*12*	*1.6509*	*39.4252*
1152	*11*	*1.6520*	*39.4686*
2048	*12*	*1.6563*	*39.6241*
4096	*11*	*1.6591*	*39.7258*
4608	*12*	*1.6594*	*39.7377*
5000	*11*	*1.6596*	*39.7451*
6000	*12*	*1.6600*	*39.7583*
8192	*11*	*1.6605*	*39.7764*
10000	*12*	*1.6608*	*39.7866*
16384	*16*	*1.6612*	*39.8012*
20000	*11*	*1.6613*	*39.8061*
32768	*12*	*1.6613*	*39.8078*

- LPC

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*13*	*1.5714*	*36.3616*
1024	*13*	*1.6493*	*39.3675*
1152	*13*	*1.6584*	*39.7011*
2048	*13*	*1.6973*	*41.0835*
4096	*13*	*1.7253*	*42.0386*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
4608	*13*	*1.7289*	*42.1605*
5000	*13*	*1.7312*	*42.2354*
6000	*13*	*1.7347*	*42.3526*
8192	*13*	*1.7397*	*42.5175*
10000	*14*	*1.7431*	*42.6300*
16384	*15*	*1.7473*	*42.7692*
20000	*13*	*1.7473*	*42.7698*
32768	*14*	*1.7473*	*42.7691*

- Wavelet

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*23*	*1.5915*	*37.1674*
1024	*29*	*1.6089*	*37.8438*
1152	*74*	*1.3225*	*24.3841*
2048	*40*	*1.6182*	*38.2037*
4096	*65*	*1.6229*	*38.3804*
4608	*194*	*1.4406*	*30.5822*
5000	*181*	*1.4651*	*31.7438*
6000	*149*	*1.5171*	*34.0832*
8192	*113*	*1.6251*	*38.4654*
10000	*344*	*1.4921*	*32.9801*
16384	*201*	*1.6262*	*38.5078*
20000	*631*	*1.5082*	*33.6950*
32768	*391*	*1.6266*	*38.5231*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Led Zeppelin – D'yer Mak'er

- FIR

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*11*	*1.3801*	*27.5398*
1024	*10*	*1.3870*	*27.9033*
1152	*11*	*1.3878*	*27.9440*
2048	*10*	*1.3905*	*28.0841*
4096	*12*	*1.3923*	*28.1739*
4608	*11*	*1.3925*	*28.1841*
5000	*11*	*1.3926*	*28.1905*
6000	*10*	*1.3928*	*28.2036*
8192	*11*	*1.3932*	*28.2221*
10000	*10*	*1.3934*	*28.2314*
16384	*12*	*1.3937*	*28.2499*
20000	*10*	*1.3938*	*28.2530*
32768	*12*	*1.3940*	*28.2618*

- LPC

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*12*	*1.4273*	*29.9364*
1024	*12*	*1.4790*	*32.3851*
1152	*13*	*1.4850*	*32.6579*
2048	*13*	*1.5054*	*33.5742*
4096	*12*	*1.5142*	*33.9594*
4608	*12*	*1.5138*	*33.9410*
5000	*12*	*1.5134*	*33.9237*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
6000	*12*	*1.5112*	*33.8284*
8192	*12*	*1.5067*	*33.6277*
10000	*12*	*1.4998*	*33.3228*
16384	*13*	*1.4912*	*32.9382*
20000	*12*	*1.4916*	*32.9585*
32768	*13*	*1.4898*	*32.8763*

- Wavelet

rozmiar bloku	*czas kompresji [s]*	*CR*	*CP*
512	*20*	*1.4218*	*29.6664*
1024	*26*	*1.4326*	*30.1953*
1152	*63*	*1.1927*	*16.1541*
2048	*39*	*1.4381*	*30.4629*
4096	*71*	*1.4407*	*30.5904*
4608	*192*	*1.2936*	*22.6975*
5000	*160*	*1.3133*	*23.8533*
6000	*130*	*1.3549*	*26.1920*
8192	*95*	*1.4420*	*30.6529*
10000	*287*	*1.3359*	*25.1426*
16384	*178*	*1.4428*	*30.6905*
20000	*652*	*1.3490*	*25.8726*
32768	*389*	*1.4432*	*30.7084*

Gladiator OST – Now We Are Free

- FIR

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*11*	*1.6319*	*38.7206*
1024	*11*	*1.6425*	*39.1166*
1152	*10*	*1.6437*	*39.1606*
2048	*10*	*1.6479*	*39.3151*
4096	*10*	*1.6506*	*39.4150*
4608	*10*	*1.6509*	*39.4254*
5000	*11*	*1.6511*	*39.4329*
6000	*10*	*1.6514*	*39.4458*
8192	*10*	*1.6519*	*39.4654*
10000	*10*	*1.6522*	*39.4741*
16384	*12*	*1.6526*	*39.4898*
20000	*10*	*1.6527*	*39.4928*
32768	*11*	*1.6527*	*39.4943*

- LPC

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*12*	*1.6748*	*40.2912*
1024	*12*	*1.7527*	*42.9457*
1152	*12*	*1.7622*	*43.2543*
2048	*12*	*1.7963*	*44.3309*
4096	*12*	*1.8178*	*44.9879*
4608	*12*	*1.8199*	*45.0506*
5000	*12*	*1.8221*	*45.1194*
6000	*12*	*1.8244*	*45.1873*
8192	*12*	*1.8272*	*45.2701*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji*	*CR*	*CP*
10000	*12*	*1.8290*	*45.3266*
16384	*13*	*1.8316*	*45.4036*
20000	*13*	*1.8318*	*45.4092*
32768	*12*	*1.8314*	*45.3978*

- Wavelet

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*19*	*1.7278*	*42.1238*
1024	*22*	*1.7475*	*42.7757*
1152	*55*	*1.4292*	*30.0321*
2048	*31*	*1.7584*	*43.1285*
4096	*52*	*1.7639*	*43.3089*
4608	*161*	*1.5554*	*35.7073*
5000	*149*	*1.5836*	*36.8512*
6000	*124*	*1.6436*	*39.1576*
8192	*91*	*1.7668*	*43.4012*
10000	*275*	*1.6131*	*38.0059*
16384	*180*	*1.7682*	*43.4440*
20000	*577*	*1.6307*	*38.6780*
32768	*362*	*1.7685*	*43.4560*

Sweet Noise – 9/1

- FIR

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*10*	*1.2717*	*21.3623*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji*	*CR*	*CP*
1024	*10*	*1.2773*	*21.7087*
1152	*10*	*1.2779*	*21.7469*
2048	*10*	*1.2801*	*21.8820*
4096	*10*	*1.2816*	*21.9697*
4608	*10*	*1.2817*	*21.9795*
5000	*10*	*1.2818*	*21.9858*
6000	*10*	*1.2820*	*21.9982*
8192	*10*	*1.2823*	*22.0142*
10000	*10*	*1.2824*	*22.0225*
16384	*11*	*1.2827*	*22.0364*
20000	*10*	*1.2827*	*22.0409*
32768	*10*	*1.2828*	*22.0483*

- LPC

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*11*	*1.3030*	*23.2549*
1024	*12*	*1.3436*	*25.5747*
1152	*12*	*1.3483*	*25.8310*
2048	*12*	*1.3639*	*26.6797*
4096	*11*	*1.3708*	*27.0491*
4608	*11*	*1.3711*	*27.0642*
5000	*11*	*1.3707*	*27.0446*
6000	*11*	*1.3708*	*27.0512*
8192	*11*	*1.3703*	*27.0259*
10000	*12*	*1.3692*	*26.9672*

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

rozmiar bloku	*czas kompresji*	*CR*	*CP*
16384	*13*	*1.3679*	*26.8932*
20000	*11*	*1.3673*	*26.8653*
32768	*12*	*1.3649*	*26.7322*

- Wavelet

rozmiar bloku	*czas kompresji*	*CR*	*CP*
512	*18*	*1.2998*	*23.0674*
1024	*22*	*1.3088*	*23.5969*
1152	*56*	*1.0967*	*8.8150*
2048	*35*	*1.3136*	*23.8757*
4096	*55*	*1.3161*	*24.0164*
4608	*163*	*1.1890*	*15.8932*
5000	*145*	*1.2058*	*17.0673*
6000	*121*	*1.2415*	*19.4527*
8192	*89*	*1.3172*	*24.0841*
10000	*269*	*1.2262*	*18.4486*
16384	*172*	*1.3178*	*24.1143*
20000	*535*	*1.2382*	*19.2352*
32768	*334*	*1.3180*	*24.1297*

Kult – Baranek

- FIR

```
# opts: -p 2 -m 2
512     10      1.5157  34.0253
1024    11      1.5247  34.4131
1152    10      1.5257  34.4561
2048    10      1.5292  34.6074
4096    11      1.5315  34.7058
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
4608    10      1.5318  34.7171
5000    10      1.5320  34.7242
6000    10      1.5323  34.7378
8192    11      1.5327  34.7563
10000   10      1.5329  34.7655
16384   12      1.5333  34.7819
20000   10      1.5334  34.7864
32768   11      1.5336  34.7946
```

- LPC

```
# opts: -p 1 -m 2 -l 8,8
512     12      1.5213  34.2662
1024    12      1.5858  36.9416
1152    13      1.5939  37.2618
2048    12      1.6215  38.3297
4096    13      1.6364  38.8892
4608    12      1.6378  38.9420
5000    12      1.6387  38.9761
6000    13      1.6381  38.9525
8192    12      1.6370  38.9120
10000   13      1.6357  38.8634
16384   14      1.6312  38.6951
20000   12      1.6300  38.6486
32768   19      1.6257  38.4879
```

- Wavelet

```
# opts: -p 3 -m 2
512     21      1.5041  33.5140
1024    23      1.5179  34.1216
1152    57      1.2570  20.4454
2048    33      1.5249  34.4208
4096    54      1.5282  34.5634
4608    165     1.3662  26.8053
5000    153     1.3880  27.9549
6000    129     1.4339  30.2625
8192    100     1.5299  34.6342
10000   303     1.4123  29.1911
16384   199     1.5307  34.6696
20000   599     1.4263  29.8893
32768   364     1.5311  34.6871
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Myslovitz – Zgon (live)

- FIR

```
# opts: -p 2 -m 2
512     6       1.6405  39.0438
1024    6       1.6516  39.4524
1152    6       1.6528  39.4972
2048    6       1.6572  39.6561
4096    6       1.6600  39.7586
4608    6       1.6603  39.7695
5000    6       1.6605  39.7767
6000    6       1.6609  39.7909
8192    6       1.6614  39.8098
10000   6       1.6617  39.8190
16384   7       1.6621  39.8359
20000   10      1.6622  39.8404
32768   6       1.6625  39.8486
```

- LPC

```
# opts: -p 1 -m 2 -l 8,8
512     7       1.6503  39.4060
1024    7       1.7570  43.0844
1152    7       1.7697  43.4928
2048    7       1.8315  45.4008
4096    7       1.8806  46.8259
4608    7       1.8882  47.0386
5000    7       1.8926  47.1618
6000    7       1.8992  47.3476
8192    7       1.9124  47.7089
10000   7       1.9186  47.8796
16384   8       1.9288  48.1533
20000   7       1.9305  48.2012
32768   8       1.9359  48.3444
```

- Wavelet

```
# opts: -p 3 -m 2
512     19      1.4475  30.9140
1024    24      1.4594  31.4795
1152    71      1.2124  17.5187
2048    31      1.4653  31.7534
4096    71      1.4683  31.8943
4608    126     1.3180  24.1248
5000    102     1.3382  25.2711
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
6000    83      1.3807  27.5711
8192    56      1.4699  31.9680
10000   173     1.3613  26.5394
16384   107     1.4707  32.0064
20000   329     1.3744  27.2425
32768   207     1.4711  32.0258
```

Vypsana Fixa – Palenie Titonia

- FIR

```
# opts: -p 2 -m 2
512     9       1.1804  15.2833
1024    8       1.1854  15.6434
1152    8       1.1861  15.6883
2048    9       1.1885  15.8599
4096    8       1.1908  16.0222
4608    8       1.1912  16.0491
5000    8       1.1914  16.0651
6000    9       1.1920  16.1077
8192    9       1.1928  16.1667
10000   8       1.1933  16.1970
16384   10      1.1945  16.2854
20000   8       1.1948  16.3011
32768   9       1.1954  16.3425
```

- LPC

```
# opts: -p 1 -m 2 -l 8,8
512     10      1.2077  17.1970
1024    10      1.2433  19.5676
1152    10      1.2474  19.8335
2048    10      1.2615  20.7263
4096    9       1.2694  21.2215
4608    10      1.2704  21.2872
5000    9       1.2702  21.2720
6000    10      1.2711  21.3297
8192    10      1.2710  21.3218
10000   9       1.2707  21.3024
16384   11      1.2705  21.2912
20000   9       1.2700  21.2571
32768   11      1.2694  21.2202
```

- Wavelet

```
# opts: -p 3 -m 2
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
512     15      1.1954  16.3463
1024    20      1.2031  16.8821
1152    48      1.0117  1.1609
2048    27      1.2071  17.1568
4096    46      1.2094  17.3112
4608    138     1.0973  8.8635
5000    129     1.1120  10.0733
6000    108     1.1435  12.5529
8192    80      1.2110  17.4206
10000   248     1.1315  11.6204
16384   165     1.2122  17.5070
20000   481     1.1433  12.5330
32768   274     1.2130  17.5624
```

Turbo – Szalony Ikar

- FIR

```
# opts: -p 2 -m 2
512     9       1.2406  19.3929
1024    9       1.2459  19.7355
1152    9       1.2465  19.7748
2048    9       1.2485  19.9059
4096    9       1.2499  19.9907
4608    9       1.2500  19.9991
5000    9       1.2501  20.0049
6000    9       1.2502  20.0154
8192    9       1.2505  20.0304
10000   9       1.2506  20.0361
16384   10      1.2507  20.0468
20000   9       1.2508  20.0489
32768   9       1.2508  20.0482
```

- LPC

```
# opts: -p 1 -m 2 -l 8,8
512     11      1.2699  21.2539
1024    10      1.3127  23.8183
1152    11      1.3179  24.1189
2048    10      1.3361  25.1566
4096    10      1.3464  25.7293
4608    11      1.3475  25.7906
5000    10      1.3482  25.8245
6000    10      1.3489  25.8661
8192    10      1.3484  25.8372
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
10000   10      1.3481  25.8188
16384   11      1.3454  25.6722
20000   10      1.3460  25.7057
32768   11      1.3450  25.6529
```

- Wavelet

```
# opts: -p 3 -m 2
512     16      1.2502  20.0153
1024    19      1.2594  20.5951
1152    51      1.0546  5.1808
2048    28      1.2638  20.8750
4096    47      1.2658  20.9992
4608    143     1.1447  12.6399
5000    135     1.1606  13.8358
6000    121     1.1943  16.2657
8192    88      1.2666  21.0515
10000   254     1.1804  15.2840
16384   167     1.2668  21.0630
20000   508     1.1919  16.0979
32768   307     1.2667  21.0555
```

Iron Maiden – Sign Of The Cross

- FIR

```
# opts: -p 2 -m 2
512     29      1.6171  38.1597
1024    33      1.6274  38.5530
1152    31      1.6286  38.5967
2048    31      1.6326  38.7498
4096    31      1.6353  38.8482
4608    29      1.6356  38.8592
5000    30      1.6358  38.8661
6000    29      1.6361  38.8797
8192    30      1.6366  38.8979
10000   29      1.6368  38.9069
16384   35      1.6373  38.9227
20000   31      1.6374  38.9272
32768   32      1.6376  38.9349
```

- LPC

```
# opts: -p 1 -m 2 -l 8,8
512     42      1.5819  36.7839
1024    35      1.6565  39.6320
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
1152    37      1.6657  39.9639
2048    35      1.7022  41.2523
4096    34      1.7291  42.1676
4608    34      1.7323  42.2721
5000    36      1.7339  42.3280
6000    34      1.7387  42.4844
8192    34      1.7445  42.6776
10000   34      1.7472  42.7640
16384   38      1.7538  42.9809
20000   34      1.7539  42.9831
32768   36      1.7581  43.1215
```

- Wavelet

```
# opts: -p 3 -m 2
512     53      1.6129  38.0008
1024    63      1.6296  38.6370
1152    150     1.3418  25.4727
2048    89      1.6386  38.9714
4096    143     1.6431  39.1411
4608    433     1.4583  31.4281
5000    400     1.4831  32.5759
6000    335     1.5360  34.8960
8192    249     1.6454  39.2242
10000   745     1.5104  33.7907
16384   480     1.6465  39.2660
20000   1498    1.5263  34.4835
32768   889     1.6471  39.2869
```

rząd LPC

Izrael – See I & I

```
# opts: -p 1 -m 2 -b 4608
2       11      1.6743  40.2725
3       11      1.7089  41.4833
4       12      1.7244  42.0094
5       12      1.7282  42.1378
6       13      1.7289  42.1600
8       14      1.7289  42.1605
10      15      1.7271  42.0987
12      16      1.7251  42.0313
15      17      1.7219  41.9240
20      20      1.7161  41.7280
32      25      1.7012  41.2171
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
# opts: -p 1 -m 2 -b 4608
2       11      1.6351  38.8423
3       12      1.6595  39.7414
4       11      1.6674  40.0279
5       12      1.6702  40.1282
6       12      1.6710  40.1569
8       12      1.6724  40.2048
10      12      1.6733  40.2396
12      13      1.6740  40.2638
15      13      1.6749  40.2964
20      15      1.6758  40.3275
32      17      1.6772  40.3769
```

Led Zeppelin – D'yer Mak'er

```
# opts: -p 1 -m 2 -b 4608
2       11      1.6743  40.2725
3       11      1.7089  41.4833
4       12      1.7244  42.0094
5       12      1.7282  42.1378
6       13      1.7289  42.1600
8       14      1.7289  42.1605
10      15      1.7271  42.0987
12      16      1.7251  42.0313
15      17      1.7219  41.9240
20      20      1.7161  41.7280
32      25      1.7012  41.2171

# opts: -p 1 -m 2 -b 4608
2       11      1.6351  38.8423
3       12      1.6595  39.7414
4       11      1.6674  40.0279
5       12      1.6702  40.1282
6       12      1.6710  40.1569
8       12      1.6724  40.2048
10      12      1.6733  40.2396
12      13      1.6740  40.2638
15      13      1.6749  40.2964
20      15      1.6758  40.3275
32      17      1.6772  40.3769
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

Gladiator OST – Now We Are Free

```
# opts: -p 1 -m 2 -b 4608
2       10      1.7795  43.8046
3       11      1.7839  43.9423
4       11      1.8075  44.6750
5       12      1.8145  44.8879
6       12      1.8174  44.9756
8       13      1.8199  45.0506
10      13      1.8208  45.0804
12      14      1.8220  45.1158
15      15      1.8196  45.0416
20      18      1.8151  44.9065
32      23      1.8016  44.4934

# opts: -p 1 -m 2 -b 4608
2       10      1.7376  42.4499
3       10      1.7405  42.5444
4       11      1.7500  42.8556
5       11      1.7537  42.9774
6       14      1.7561  43.0545
8       11      1.7584  43.1297
10      11      1.7601  43.1858
12      12      1.7625  43.2635
15      12      1.7642  43.3160
20      13      1.7658  43.3678
32      15      1.7681  43.4429
```

Sweet Noise – 9/1

```
# opts: -p 1 -m 2 -b 4608
2       10      1.3393  25.3346
3       11      1.3591  26.4201
4       11      1.3637  26.6727
5       11      1.3683  26.9172
6       12      1.3685  26.9266
8       12      1.3711  27.0642
10      13      1.3706  27.0409
12      14      1.3697  26.9935
15      15      1.3684  26.9202
20      17      1.3662  26.8042
32      23      1.3596  26.4491

# opts: -p 1 -m 2 -b 4608
2       9       1.3240  24.4728
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
3       10      1.3358  25.1380
4       10      1.3386  25.2943
5       10      1.3406  25.4084
6       12      1.3412  25.4384
8       11      1.3433  25.5544
10      11      1.3439  25.5903
12      12      1.3445  25.6234
15      12      1.3451  25.6579
20      12      1.3460  25.7058
32      14      1.3476  25.7944
```

Kult – Baranek

```
# opts: -p 1 -m 2 -b 4608
2       10      1.5788  36.6624
3       11      1.6103  37.9010
4       11      1.6251  38.4651
5       12      1.6333  38.7745
6       13      1.6352  38.8469
8       13      1.6378  38.9420
10      14      1.6389  38.9827
12      15      1.6391  38.9893
15      16      1.6383  38.9598
20      19      1.6364  38.8905
32      24      1.6271  38.5406

# opts: -p 1 -m 2 -b 4608
2       10      1.5548  35.6842
3       11      1.5707  36.3332
4       10      1.5788  36.6597
5       10      1.5843  36.8793
6       11      1.5865  36.9697
8       12      1.5892  37.0737
10      12      1.5912  37.1543
12      13      1.5927  37.2145
15      13      1.5943  37.2773
20      13      1.5966  37.3653
32      16      1.5994  37.4770
```

Myslovitz – Zgon (live)

```
# opts: -p 1 -m 2 -b 4608
2       6       1.6303  38.6630
3       6       1.8349  45.4996
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
4       7       1.8589  46.2040
5       7       1.8738  46.6326
6       7       1.8834  46.9042
8       8       1.8882  47.0386
10      8       1.8872  47.0101
12      9       1.8863  46.9859
15      10      1.8839  46.9187
20      11      1.8769  46.7207
32      16      1.8579  46.1749

# opts: -p 1 -m 2 -b 4608
2       7       1.6303  38.6604
3       7       1.7914  44.1784
4       7       1.8192  45.0302
5       7       1.8244  45.1860
6       7       1.8281  45.2981
8       7       1.8325  45.4309
10      7       1.8339  45.4709
12      7       1.8355  45.5195
15      7       1.8369  45.5617
20      8       1.8384  45.6041
32      10      1.8400  45.6525
```

Vypsana Fixa – Palenie Titonia

```
# opts: -p 1 -m 2 -b 4608
2       8       1.2334  18.9222
3       8       1.2543  20.2756
4       9       1.2612  20.7117
5       10      1.2671  21.0788
6       9       1.2679  21.1297
8       10      1.2704  21.2872
10      11      1.2718  21.3712
12      12      1.2714  21.3490
15      13      1.2702  21.2706
20      14      1.2671  21.0805
32      19      1.2592  20.5839

# opts: -p 1 -m 2 -b 4608
2       8       1.2152  17.7094
3       8       1.2286  18.6061
4       9       1.2326  18.8678
5       8       1.2360  19.0909
6       8       1.2367  19.1420
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
8       11      1.2388  19.2760
10      9       1.2402  19.3696
12      9       1.2409  19.4149
15      10      1.2415  19.4541
20      10      1.2420  19.4819
32      12      1.2426  19.5264
```

Turbo – Szalony Ikar

```
# opts: -p 1 -m 2 -b 4608
2       8       1.2914  22.5665
3       9       1.3106  23.6984
4       9       1.3236  24.4482
5       9       1.3356  25.1269
6       10      1.3409  25.4258
8       11      1.3475  25.7906
10      11      1.3477  25.7999
12      12      1.3466  25.7399
15      13      1.3448  25.6385
20      15      1.3414  25.4493
32      20      1.3324  24.9448

# opts: -p 1 -m 2 -b 4608
2       8       1.2736  21.4820
3       9       1.2846  22.1549
4       9       1.2913  22.5580
5       9       1.2978  22.9477
6       9       1.3007  23.1201
8       10      1.3051  23.3763
10      10      1.3061  23.4360
12      10      1.3066  23.4653
15      10      1.3069  23.4845
20      11      1.3075  23.5201
32      13      1.3081  23.5523
```

Iron Maiden – Sign Of The Cross

```
# opts: -p 1 -m 2 -b 4608
2       29      1.6848  40.6465
3       29      1.7166  41.7458
4       30      1.7260  42.0627
5       31      1.7305  42.2139
6       31      1.7310  42.2291
8       34      1.7323  42.2721
10      36      1.7320  42.2640
```

Mateusz Grzegorzek <armadilo@interia.pl>
Piotr Malinowski <pmalinow@gmail.com>
Krzysztof Jamróz <krzyh000@o2.pl>
Wojciech Wasiak <wwasiak@elka.pw.edu.pl>

```
12      40      1.7307  42.2195
15      44      1.7278  42.1244
20      50      1.7226  41.9474
32      63      1.7070  41.4194

# opts: -p 1 -m 2 -b 4608
2       27      1.6515  39.4488
3       29      1.6690  40.0852
4       30      1.6744  40.2765
5       31      1.6771  40.3732
6       32      1.6782  40.4132
8       31      1.6801  40.4794
10      31      1.6811  40.5145
12      32      1.6819  40.5449
15      33      1.6826  40.5681
20      38      1.6834  40.5969
32      41      1.6843  40.6264
```