NeuroCuts - Neural Packet Classification
Classificador de Pacotes Neural

Autores: Eric Liang[1], Hang Zhu[2], Xin Jin[2], Ion Stoica[1]
[1]UC Berkeley, [2]JHU

Uma nova abordagem de bits efetivos (BE) para endereços IPv4 como estudo de caso: NeuroCuts.

Discentes:
Eliton Luiz Scardin Perin
Jonathan Aldori Alves de Oliveira

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Objetivo:

- Principal:
  - Melhorar a utilização da memória do NeuroCuts com a utilização de técnicas que façam uso apenas do bits efetivos (BE) necessários para identificar o conjunto de regras.

- Específicos:
  - Sintetizar a árvore gerada pelo NeuroCut usando bits efetivos (BE).
  - Utilizar os BE para otimizar as entradas da regra (Endereços de Origem e Destino).
  - Produzir um autoajuste no aprendizado por reforço no NeuroCuts, afim de encontrar um melhor trade-off entre utilização da memória e desempenho no tempo para classificação do pacote.

ClassBrench utilizados:

- Aplicação de BE:
  - acl1_1k
  - acl1_10k
  - acl1_100k
  - acl2_1k
  - acl2_10k
  - acl2_100k
  - acl3_1k
  - acl3_10k
  - acl3_100k
  - acl4_1k
  - acl4_10k
  - acl4_100k
  - acl5_1k
  - acl5_10k
  - acl5_100k
  - fw1_1k
  - fw1_10k
  - fw1_100k
  - fw2_1k
  - fw2_10k
  - fw2_100k
  - fw3_1k
  - fw3_10k
  - fw3_100k
  - fw4_1k
  - fw4_10k
  - fw4_100k
  - fw5_1k
  - fw5_10k
  - fw5_100k
  - ipc1_1k
  - ipc1_10k
  - ipc1_100k
  - ipc2_1k
  - ipc2_10k
  - ipc2_100k

ClassBrench utilizados:

- NeuroCuts original versus NeuroCutsBE:
  - acl2_1k
  - acl4_1k
  - fw1_1k
  - fw2_1k
  - fw3_1k
  - fw4_1k
  - fw5_1k
  - ipc1_1k
  - ipc2_1k

Tempo e poder computacional foi limitado!

Compreenda endereços IP

- O endereço IP (Internet Protocol) é um endereço usado para identificar de maneira única um dispositivo em uma rede IP. O endereço é composto de 32 bits.
- Os 32 bits são divididos em quatro octetos (1 octeto = 8 bits). Cada octeto é convertido em decimal e separado por um ponto final (ponto).
- Por esse motivo, um endereço IP deve ser expressado no formato decimal pontuado (por exemplo,192.168.0.1). O valor em cada octeto varia de 0 a 255 decimais ou de 00000000 a 11111111 em binário.

Compreenda BE

10.	1.	1.	19	Decimal
00001010	00000001	00000001	00010011	Binário

Endereço IP normal

1	10.	1.	1.	0	Decimal	**EB**
1	00001010	00000001	1		Binário	18

1	10.	1.	0.	0	Decimal	**EB**
1	00001010	1			Binário	10

1	10.	0.	0.	0	Decimal	**EB**
1	1010				Binário	5

1	10.	19.	0.	0	Decimal	**EB**
1	00001010	10011			Binário	14

Um bit com valor 1 no início de cada endereço, se faz necessário pois perdermos os bits zerados mais a esquerda nos casos de conversão para decimal e podemos ainda ter colisão de endereços com a aplicação de BE.

Algoritmos para seleção do BE

Algoritmo 2: Algoritmo que retorna o bits efetivos do endereço IPv4 (removeNBE)

Entrada: Valor do octeto de bits que sera avaliado
Saída: bits efetivos (BE) do octeto recebido

```
1  valor = 0;
2  comparar = verdadeiro;
3  valorBE = '';
4  foreach caracter in octeto do
5  |   if comparar then
6  |   |   valor = int(caracter);
7  |   |   if valor==1 then
8  |   |   |   comparar = Falso;
9  |   |   |   valorBE += caracter;
10 |   |   else
11 |   |   |   valorBE += caracter;
12 |   |   end
13 end
14 retorna valorBE;
```

Algoritmo 1: Algoritmo que retorna o bits efetivos do endereço IPv4

Entrada: Valores do endereço de origem inicial e final, endereço de destino inicial e final.
Saída: Valor do endereço de origem inicial e final, endereço de destino inicial e final com os BE.

```
1  s = retornaBinario(enderecoIP);
2  s1 = converteDecimal(s[0:8]);
3  s2 = converteDecimal(s[8:16]);
4  s3 = converteDecimal(s[16:24]);
5  s4 = converteDecimal(s[24:32]);
6  enderecoBE = 0;
7  if s4==0 then
8  |   if s4==0 and S3==0 then
9  |   |   if s4 == 0 and s3 == 0 and s2 == 0 then
10 |   |   |   enderecoBE = removeNBE(s[0:8]);
11 |   |   else
12 |   |   |   enderecoBE = s[0:8] + removeNBE(s[8:16]);
13 |   |   end
14 |   else
15 |   |   enderecoBE = s[0:16] + removeNBE(s[16:24]);
16 |   end
17 else
18 |   enderecoBE = s[0:24] + removeNBE(s[24:36]);
19 end
   /* Problema da conversão de decimal com zeros a
      esquerda e colisão de endereços                */
20 enderecoBE = '1' + enderecoBE;
21 retorna converteDecimal(enderecoBE)
```

Algoritmos para retornar os bits não BE

Algoritmo 4: Algoritmo que retorna o bits do octeto original do endereço IPv4 (retornaB)

Entrada: Valor do octeto de bits que sera avaliado
Saída: octeto original

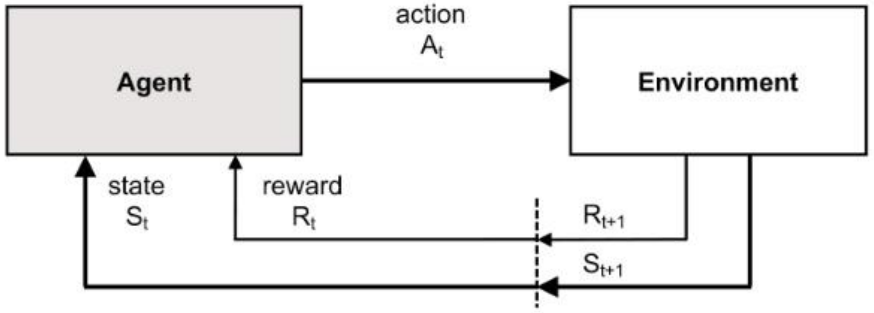
```
1 valor = '00000000' + octeto;
2 valor = valor[len(valor)-8:len(valor)];
3 retorna valor;
```

Algoritmo 3: Algoritmo que retorna O endereço IP v4 original (retornaOriginal)

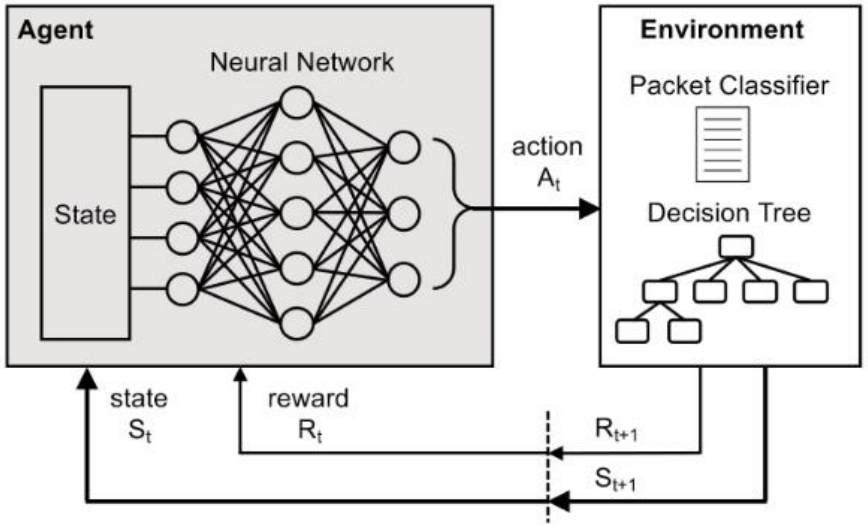
Entrada: Valores do endereço com os EB
Saída: Valores do endereço orginal sem os EB

```
1  tccRemove o 1 inserido para resolver o problema da
     conversão de decimal com zeros a esquerda e colisão de
     endereços s = retornaBinario(enderecoIPBE);
2  s = s[1:];
3  if len(s) <= 8 then
4  |   s = retornaB(s) + '00000000000000000000000';
5  else
6  |   if len(s) > 8 and len(s) <= 16 then
7  |   |   s = s[0:8] + retornaB(s[8:len(s)]) +
   |   |     '0000000000000000';
8  |   else
9  |   |   if len(s) > 16 and len(s) <= 24 then
10 |   |   |   s = s[0:16] + retornaB(s[16:len(s)]) + '00000000';
11 |   |   else
12 |   |   |   s = s[0:24] + retornaB(s[24:len(s)]);
13 |   |   end
14 |   end
15 end
16 retorna converteDecimal(s)
```

Diagrama do funcionamento do algoritmo de aprendizado por reforço.

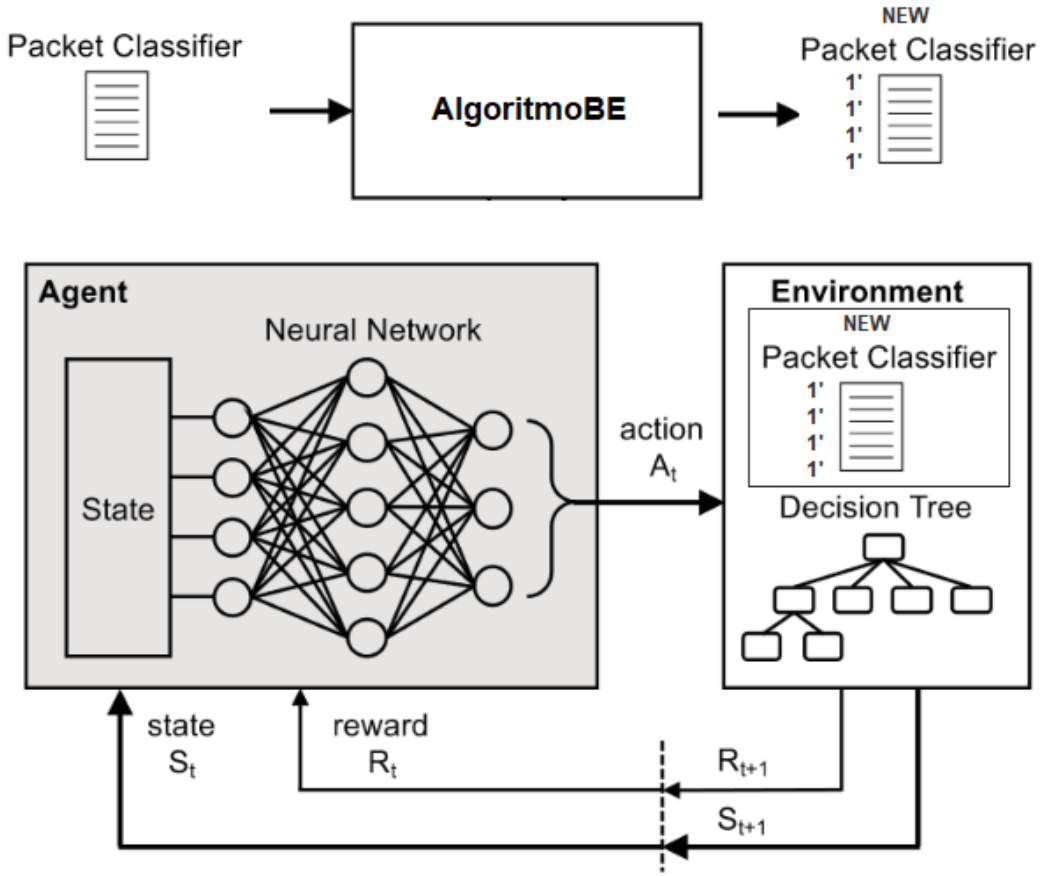


(a)



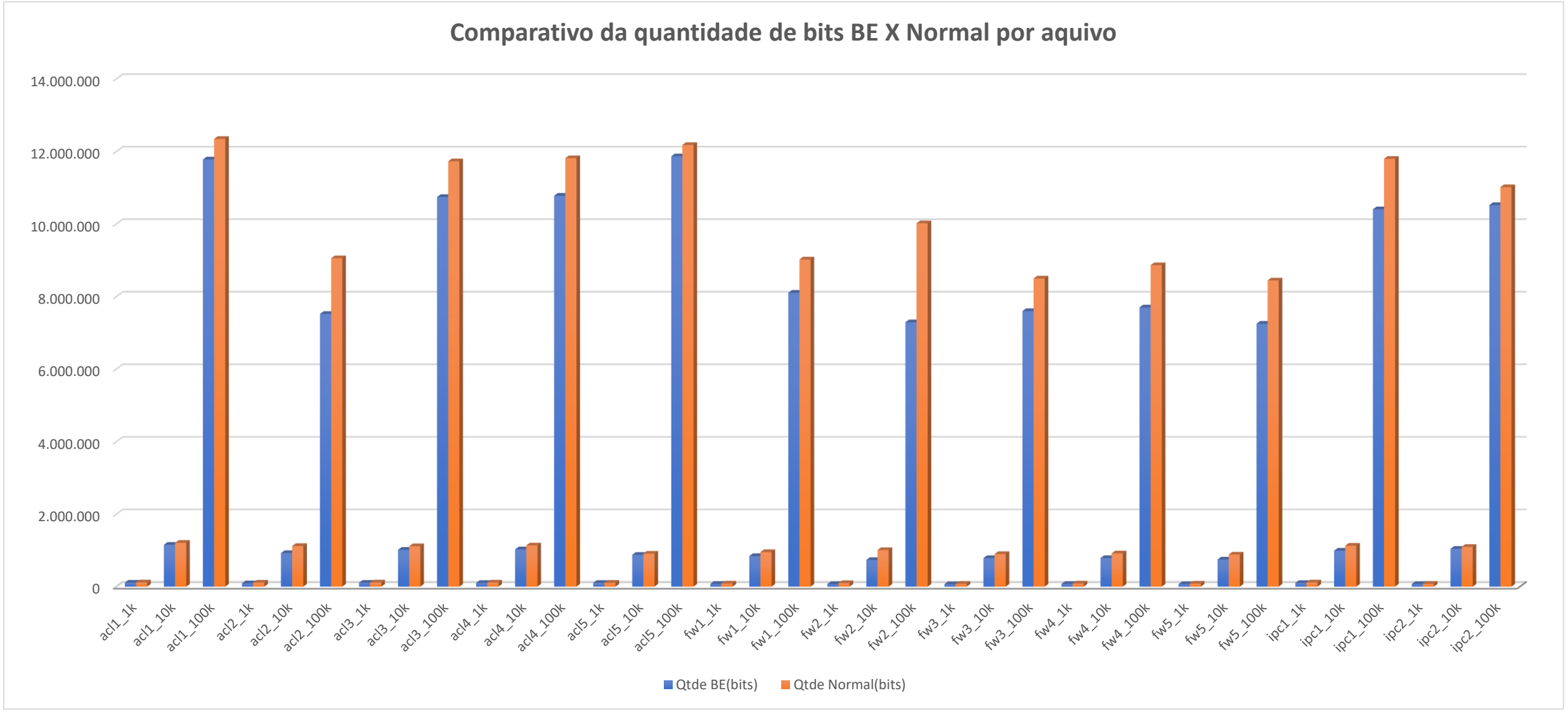
(b)

NeuroCutsBE



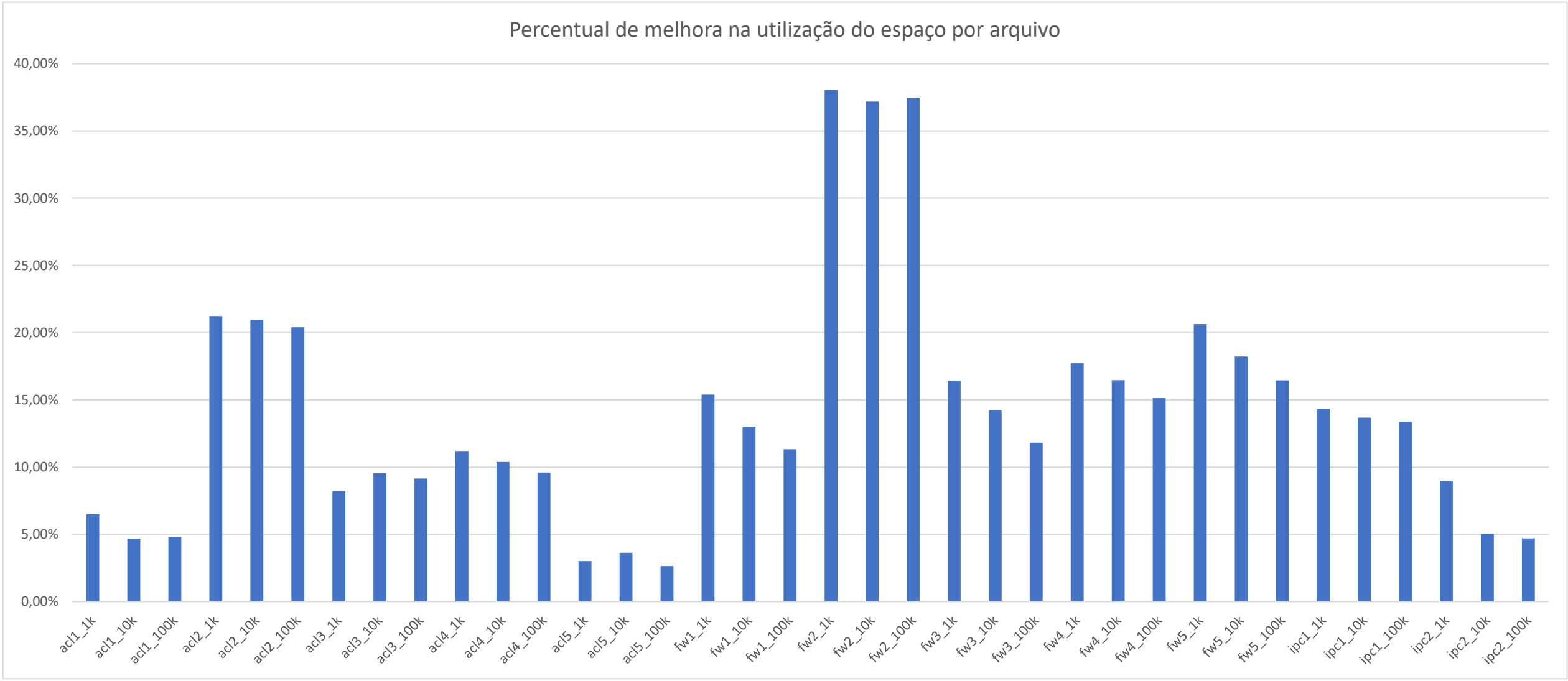
UFMS
UNIVERSIDADE FEDERAL
DE MATO GROSSO DO SUL

Comparativo da quantidade de bits BE X Normal por arquivo.



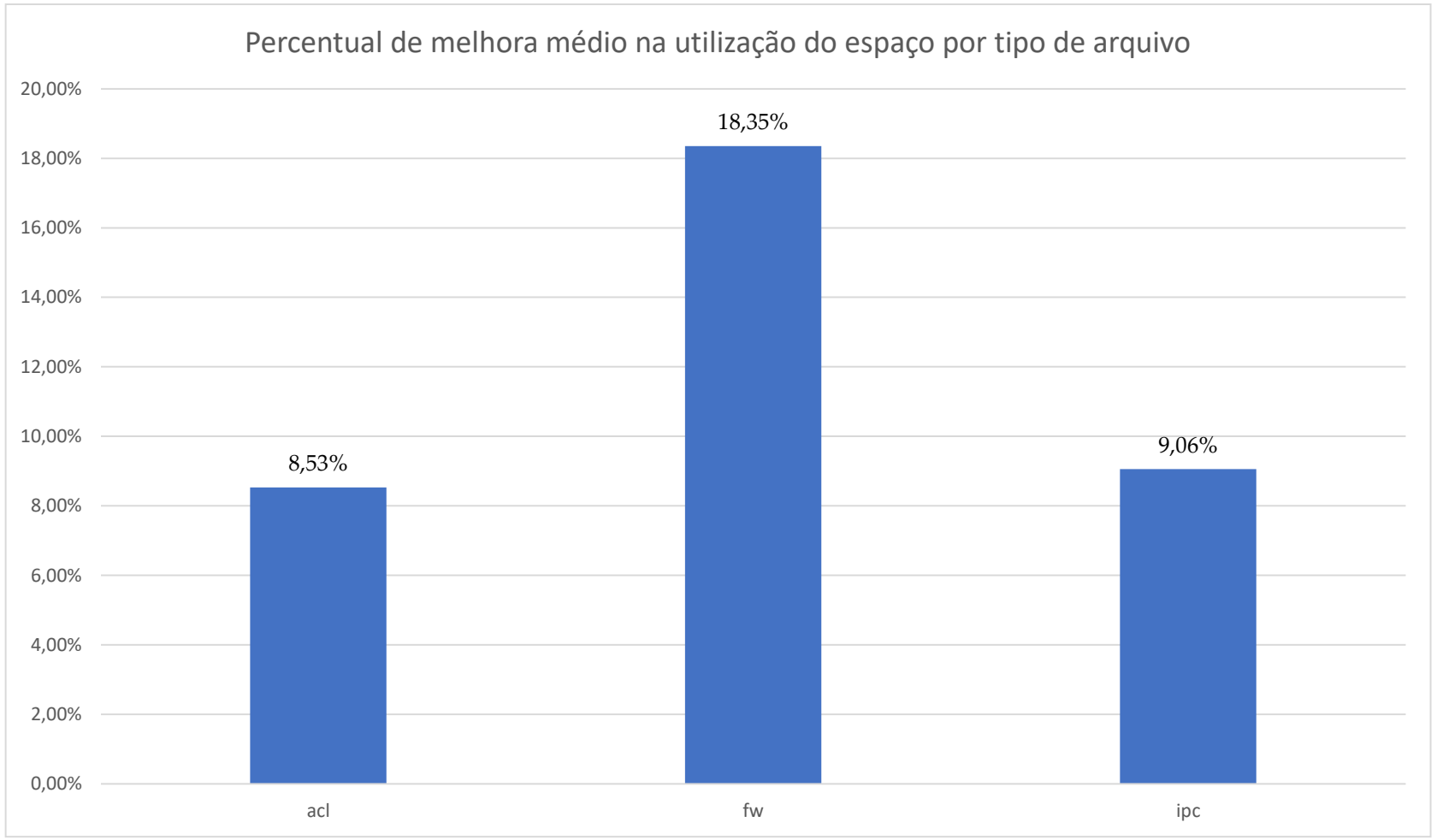
UFMS
UNIVERSIDADE FEDERAL
DE MATO GROSSO DO SUL

Percentual de melhora na utilização do espaço por arquivo

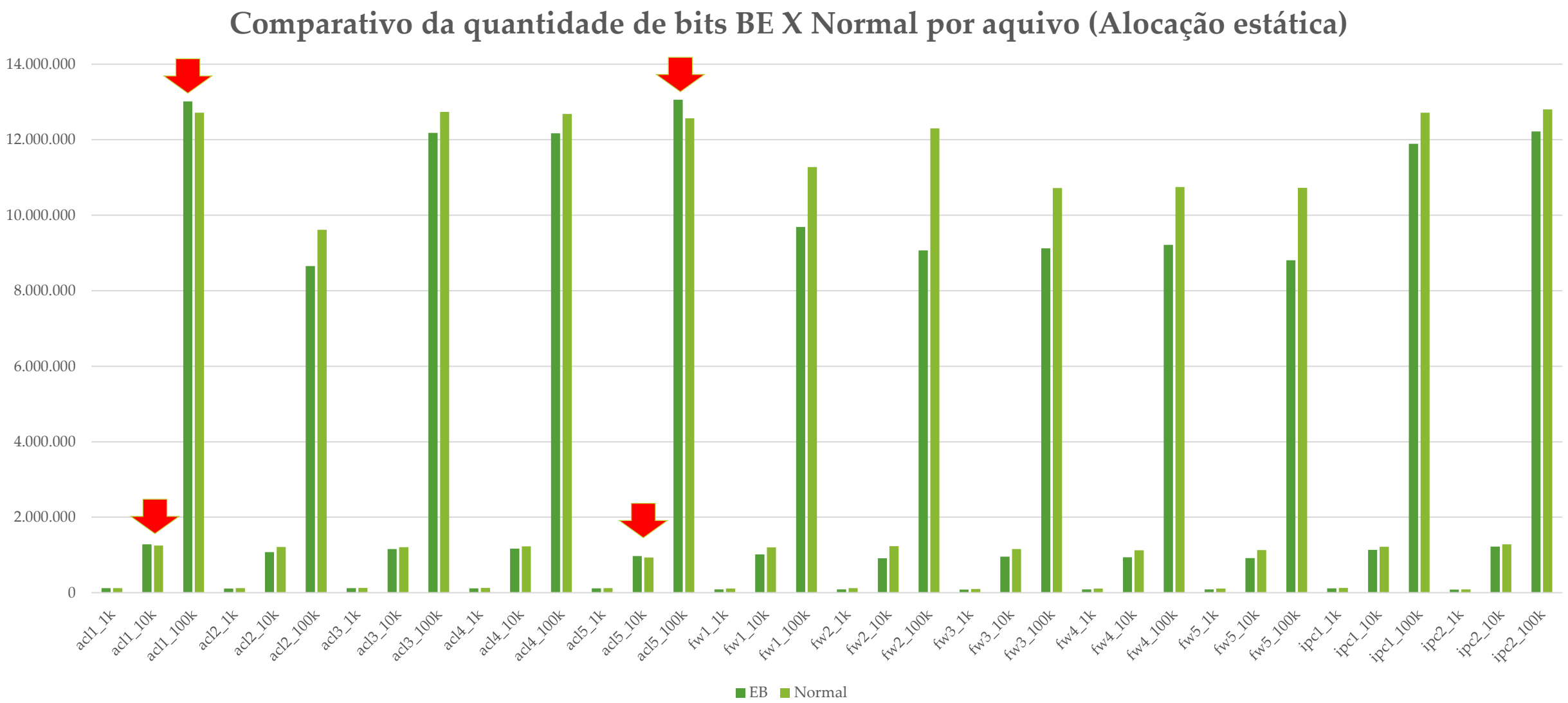


Média de bits economizado por endereço e arquivo

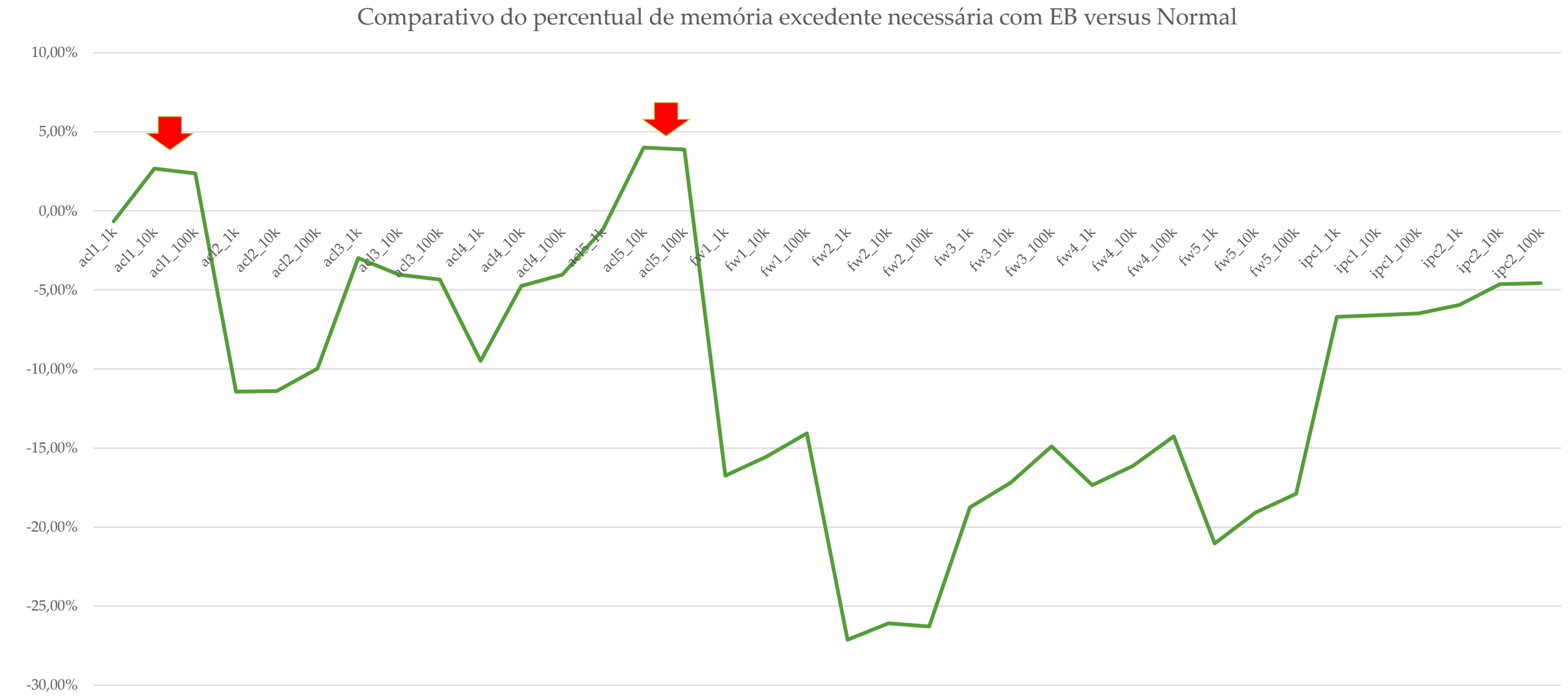
Percentual de melhora médio na utilização do espaço por tipo de arquivo



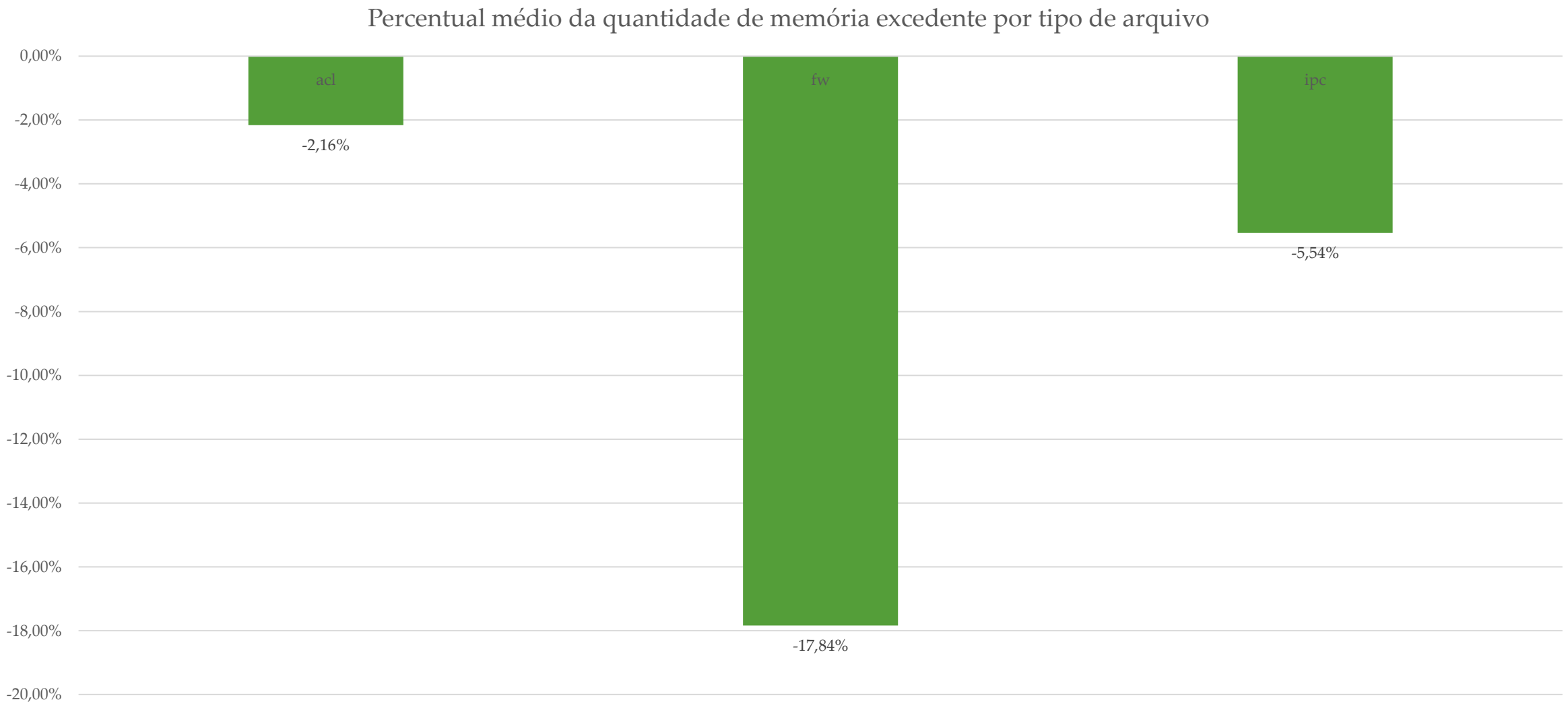
Comparativo da quantidade de bits BE X Normal por arquivo (Alocação estática)



Comparativo do percentual de memória excedente necessária BE X Normal por arquivo (Alocação estática)



Percentual médio da quantidade de memória por tipo de arquivo



Comparativo da quantidade de bits BE X Normal por arquivo (Alocação estática)

Arquivo	Número Endereços	Endereço com 1 byte	Endereço com 2 byte	Endereço com 3 byte	Endereço com 4 byte	Endereço com 5 byte	Limitação da Alo-cação (bits)	Qtde Normal(bits)	Qtde Memória Excedente
acl1_1k	3.768	31	87	145	3.193	312	119.776	120.576	-0,66%
acl1_10k	39.096	198	960	3.317	24.604	10.017	1.284.560	1.251.072	2,68%
acl1_100k	397.276	1.786	8.149	21.570	285.005	80.766	13.013.152	12.712.832	2,36%
acl2_1k	3.844	310	380	852	1.518	784	108.944	123.008	-11,43%
acl2_10k	37.904	2.570	5.106	6.676	16.228	7.324	1.074.736	1.212.928	-11,39%
acl2_100k	300.400	13.723	38.881	55.742	137.263	54.791	8.653.744	9.612.800	-9,98%
acl3_1k	3.960	336	157	371	1.876	1.220	122.936	126.720	-2,99%
acl3_10k	37.676	2.867	1.628	4.251	18.915	10.015	1.156.888	1.205.632	-4,04%
acl3_100k	397.924	31.251	15.950	37.862	218.471	94.390	12.180.568	12.733.568	-4,34%
acl4_1k	3.960	308	244	312	2.875	221	114.696	126.720	-9,49%
acl4_10k	38.374	2.438	2.322	4.327	20.289	8.998	1.169.672	1.227.968	-4,75%
acl4_100k	396.208	25.367	21.115	37.051	221.226	91.449	12.167.192	12.678.656	-4,03%
acl5_1k	3.732	0	63	420	2.878	371	118.024	119.424	-1,17%
acl5_10k	29.192	0	636	2.238	18.133	8.185	971.544	934.144	4,00%
acl5_100k	392.720	9	7.111	21.520	267.369	96.711	13.054.576	12.567.040	3,88%
fw1_1k	3.428	899	212	104	1.284	929	91.328	109.696	-16,74%
fw1_10k	37.516	9.248	2.281	840	15.343	9.804	1.013.776	1.200.512	-15,55%
fw1_100k	352.324	81.238	21.080	7.680	147.180	95.146	9.687.104	11.274.368	-14,08%

Comparativo da quantidade de bits BE X Normal por arquivo (Alocação estática)

Arquivo	Número Endereços	Endereço com 1 byte	Endereço com 2 byte	Endereço com 3 byte	Endereço com 4 byte	Endereço com 5 byte	Limitação da Alo-cação (bits)	Qtde Normal(bits)	Qtde Memória Excedente
fw2_1k	3.884	1.271	289	449	1.249	626	90.576	124.288	-27,12%
fw2_10k	38.620	12.302	2.697	4.325	12.980	6.316	913.368	1.235.840	-26,09%
fw2_100k	384.312	123.818	25.943	44.022	127.342	63.187	9.064.584	12.297.984	-26,29%
fw3_1k	3.196	972	205	52	987	980	83.088	102.272	-18,76%
fw3_10k	36.148	9.363	2.480	962	14.193	9.150	957.848	1.156.736	-17,19%
fw3_100k	335.008	82.568	20.242	6.701	130.226	95.271	9.123.312	10.720.256	-14,90%
fw4_1k	3.388	783	201	475	1.054	875	89.608	108.416	-17,35%
fw4_10k	35.100	7.535	1.908	4.056	13.777	7.824	941.976	1.123.200	-16,13%
fw4_100k	335.868	64.367	15.859	41.107	140.239	74.296	9.214.736	10.747.776	-14,26%
fw5_1k	3.456	1.059	255	82	1.200	860	87.320	110.592	-21,04%
fw5_10k	35.332	9.848	2.641	878	13.230	8.735	914.872	1.130.624	-19,08%
fw5_100k	335.188	89.515	23.597	8.575	129.036	84.465	8.807.224	10.726.016	-17,89%
ipc1_1k	3.896	222	220	836	1.720	898	116.320	124.672	-6,70%
ipc1_10k	38.072	2.439	2.150	6.614	18.693	8.176	1.137.864	1.218.304	-6,60%
ipc1_100k	397.300	24.686	20.235	70.782	199.317	82.280	11.889.360	12.713.600	-6,48%
ipc2_1k	2.784	400	80	75	1.457	772	83.784	89.088	-5,95%
ipc2_10k	40.000	6.292	1.141	719	17.393	14.455	1.220.624	1.280.000	-4,64%
ipc2_100k	400.000	64.309	9.976	5.730	174.466	145.519	12.215.280	12.800.000	-4,57%
Total	4.910.854	674.328	256.481	401.718	2.402.209	1.176.118	143.054.960	157.147.328	-8,97%

Avaliação do BE (NeuroCutsBE) quando comparado ao NeuroCuts

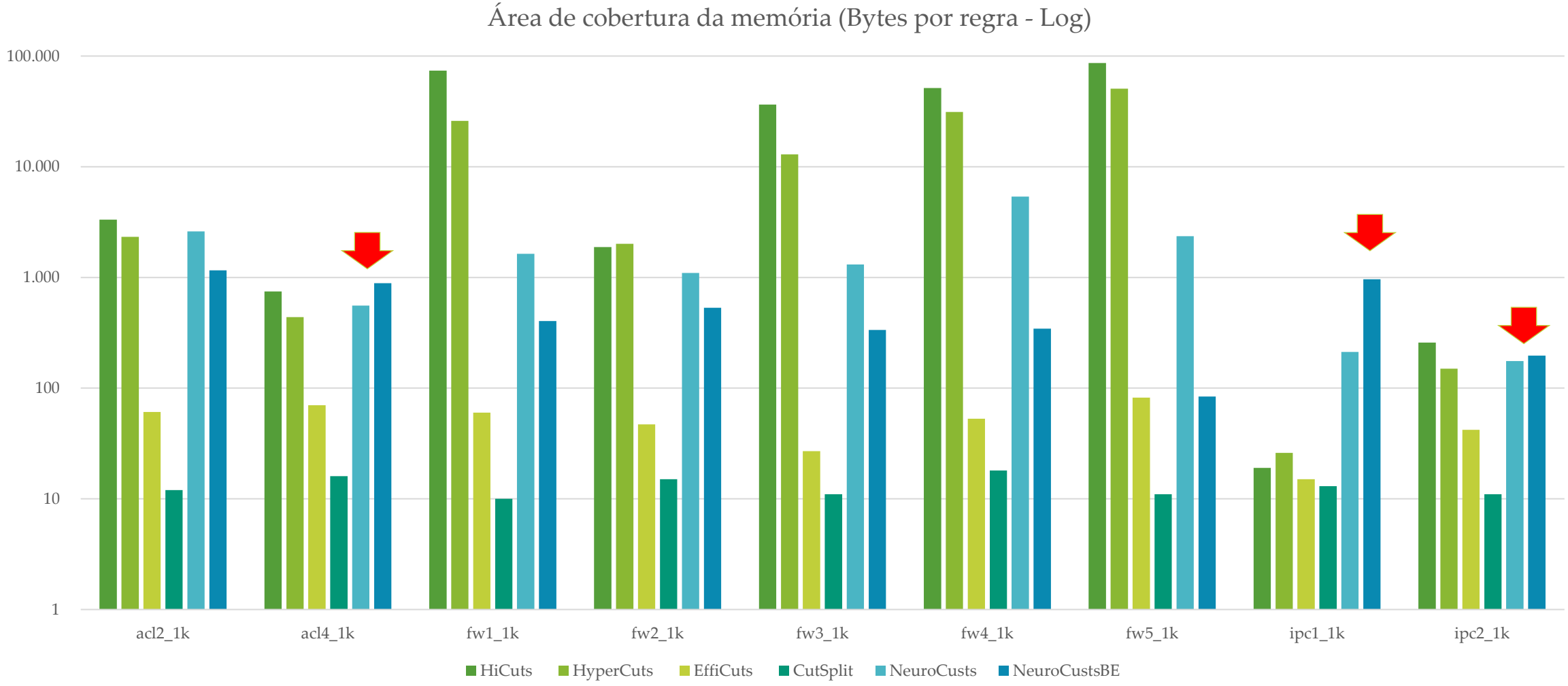
- Aplicamos os algoritmos:
  - HiCuts
  - HyperCuts
  - EffiCuts
  - CutSplit
  - NeuroCuts
  - NeuroCutsBE
- Utilizamos como parâmetros:
  - –partition-mode : None
  - –depth-weight: 0 e 1 (uma rodada para cada)

Avaliação do BE (NeuroCutsBE) quando comparado ao NeuroCuts

- Não utilizamos as demais variações possíveis ao parâmetro –partition-mode (simple, efficuts, cutsplit), pois como já dito anteriormente, nosso tempo e poder computacional foi limitado.

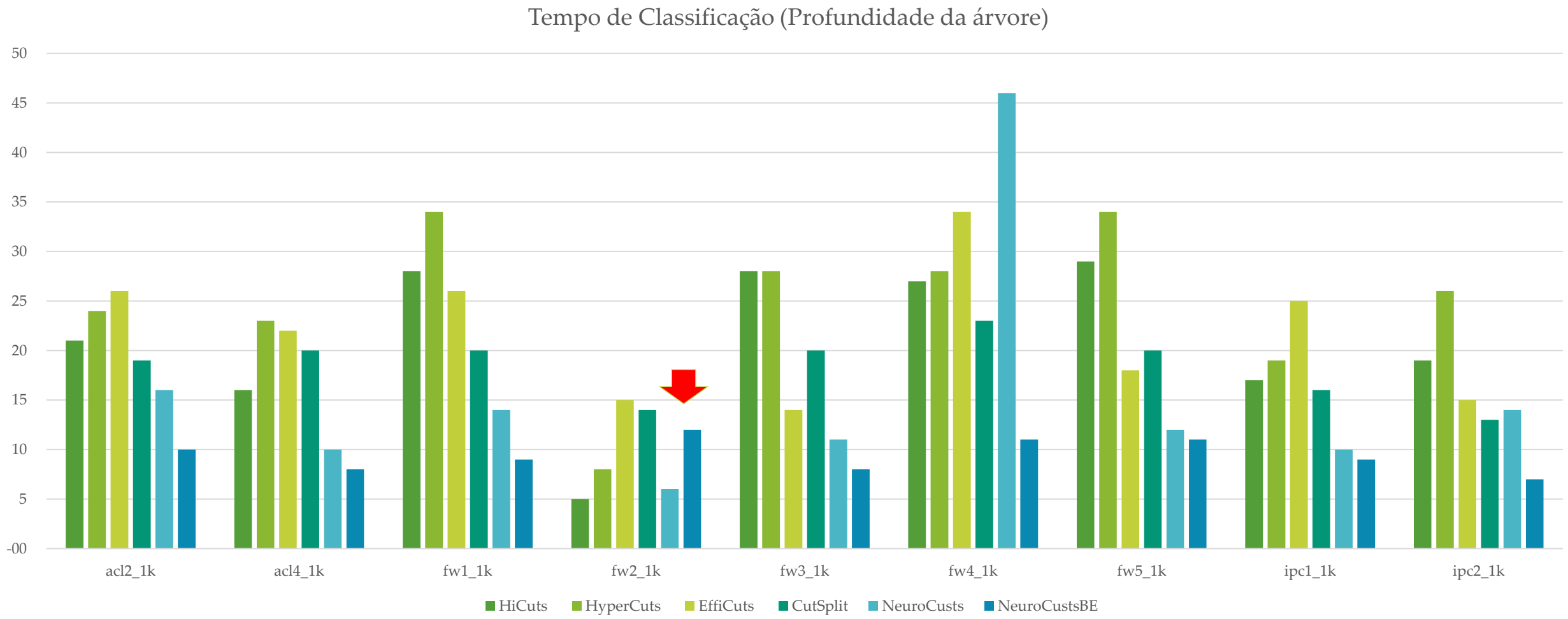
- Métricas avaliadas:
  - Quantidade de bytes por regra .
  - Profundidade da arvore.

NeuroCuts versus NeuroCutsBE



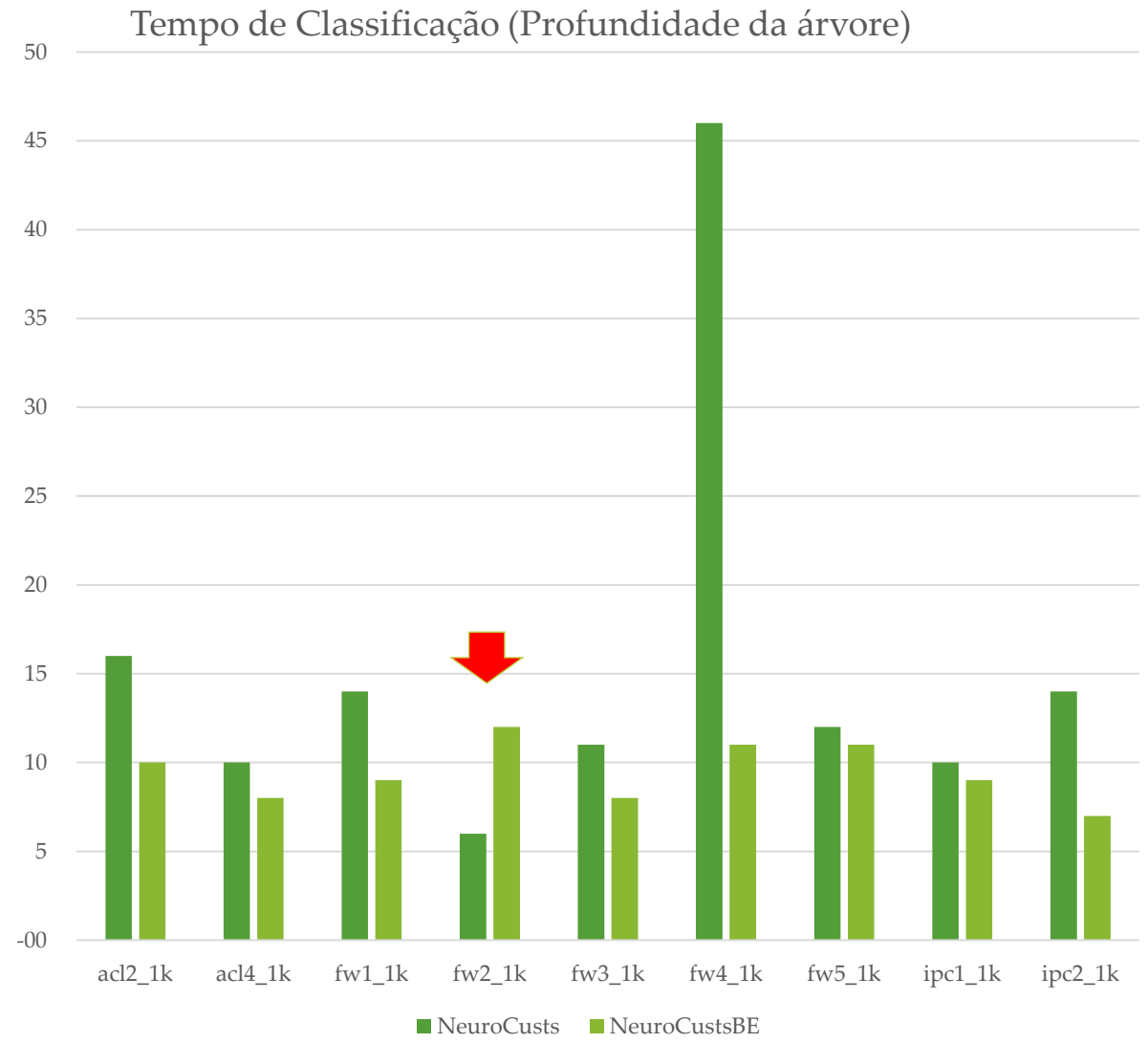
Podemos observar no gráfico que nos ClassBench (acl4_1k, ipc1_1k e ipc2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o resultado do NeuroCutsBE foi superior ao NeuroCuts original.

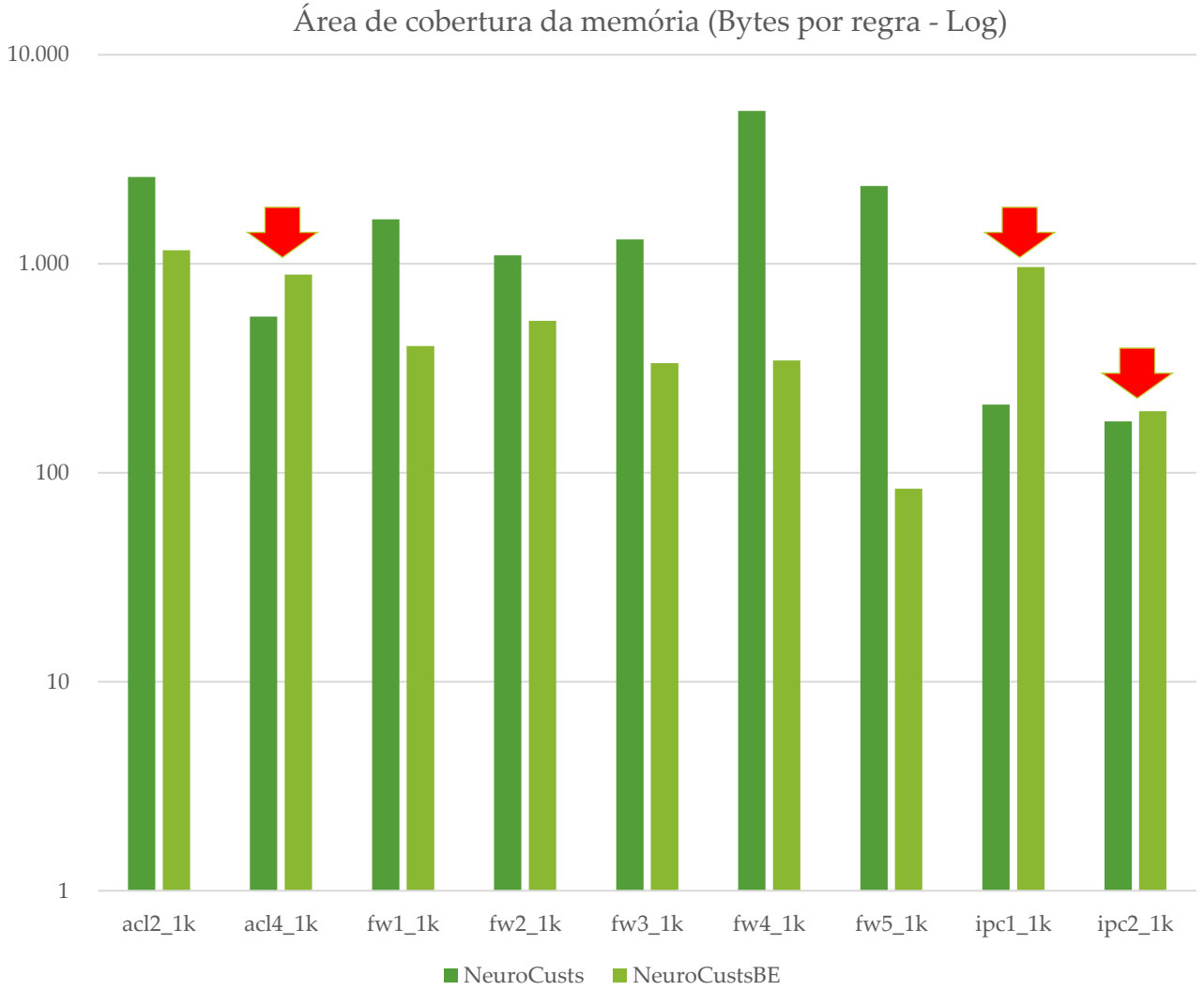
NeuroCuts versus NeuroCutsBE



Podemos observar que somente no ClassBench (fw2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o NeuroCutsBE produziu árvores que os demais algoritmos comparados. Esse resultado reforça que a utilização de BE faz com que o algoritmo produza árvores menores e utilize menos bytes por regra.

NeuroCuts versus NeuroCutsBE





Resultados:

- Apesar de adicionarmos 1 bit aos endereços de todos os arquivos analisados, o BE precisou de menos bits para representar o mesmo conjunto de endereços na maioria dos arquivos;
- O algoritmo BE precisa de menos memória para alocação dinâmica dos endereços IPv4, obtendo o valor médio de 12% quando comparado a valor original, o que representa o valor médio de 3 bits a menos por endereço.
- Em alguns casos, chegou a reduzir em 38% (fw2_1k) o espaço necessário para representar os endereços.

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Resultados:

- O algoritmo BE precisa de menos memória para alocação estática dos endereços IPv4, obtendo o valor médio de 8% quando comparado a valor original, mesmo que no pior caso precise de 5 bytes para representar o endereço.
- Observamos que o ganho muda de acordo com os tipos de Class-Bench[11] analisados (acl, fw, ipc). Sendo que as regras do tipo "fw", são as que o algoritmo BE, obtém o melhor ganho, seja com alocação dinâmica ou estática. Ou seja, a melhora depende da entrada, varia de acordo com os endereços e politicas.

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Resultados:

- Nossa abordagem de BE pode ser aplicada a outras heurísticase algoritmos que precisam trabalhar com o conjunto de 32 bitsdos endereços e máscaras de rede do protocolo IPv4.

- Podemos observar que nos ClassBench (acl4_1k, ipc1_1k e ipc2_1k)o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o resultado do NeuroCutsBE foi superior ao NeuroCuts original.

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Resultados:

- Quando avaliamos a profundidade das árvores geradas, podemos observar que somente no ClassBench (fw2_1k) o NeuroCuts obteve resultado melhor que NeuroCutsBE, nos demais o NeuroCutsBE produziu árvores menores que o NeuroCuts original e os demais algoritmos comparados. Esse resultado reforça que a utilização de BE faz com que o algoritmo produza árvores menores e utilize menos bytes por regra.

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Resultados:

- Os arquivos que contem regras de firewall foram o que obtivemos os melhores resultados médios de aproximadamente 18%.

- Com as reformulações inseridas no NeuroCuts, tentamos propor uma solução mais eficaz para o problema de CP. Nossa ideia é fornecer melhorias significativas no consumo de memória em comparação com os algoritmos citados e com a versão original do NeuroCuts.

UFMS
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

Resultados:

- Se partimos do pressuposto que agora necessitamos de menos bits para representar o endereço IPv4 original, podemos construir árvores binárias que precisarão percorrer menos nós para chegara o nó de busca.

Trabalhos Futuros:

- Nossa abordagem de BE pode ser aplicada em algoritmos de construção de árvores binárias de endereços IPv4, para chegar mais rapidamente os nó de destino que representa o endereço, quando comparado aos 32 bits dos endereços de IP original. Deixamos este exemplo para que os pesquisadores validem a eficácia de BE em árvores binárias para endereços IP.
- Testar o algoritmo NeuroCuts com BE em conjunto de dados maiores para validar se efetivamente ele apresenta um resultado ainda melhor nesses conjunto de dados.

Trabalhos Futuros:

- Testar e comparar os algoritmos NeuroCuts e NeuroCutsBE com os parâmetros: simple, efficuts e cutsplit, para adicionarmos outras ações de partição de regra, além da ação de corte implementada do NeuroCuts.

Referências:

- [1]Christopher Amato and Guy Shani. 2010. High-Level Reinforcement Learning inStrategy Games. InProceedings of the 9th International Conference on AutonomousAgents and Multiagent Systems: Volume 1 - Volume 1 (AAMAS '10). Internatio-nal Foundation for Autonomous Agents and Multiagent Systems, Richland, SC,75–82.
- [2]Pankaj Gupta and Nick McKeown. 1999. Packet classification using hierarchicalintelligent cuttings. InHot Interconnects VII, Vol. 40.
- [3]Hasibul Jamil and Ning Weng. 2020. Multibit Tries Packet Classification withDeep Reinforcement Learning. In2020 IEEE 21st International Conference on HighPerformance Switching and Routing (HPSR). IEEE, 1–6.
- [4]Wenjun Li, Xianfeng Li, Hui Li, and Gaogang Xie. 2018. Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification. InIEEEINFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2645–2653.
- [5]Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification.InProceedings of the ACM Special Interest Group on Data Communication. 256–269.
- [6]Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis An-tonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deepreinforcement learning.arXiv preprint arXiv:1312.5602(2013).
- [7]Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2020. A ComputationalApproach to Packet Classification.arXiv preprint arXiv:2002.07584(2020).
- [8]David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, MatthewLai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel,et al.2018. A general reinforcement learning algorithm that masters chess, shogi,and Go through self-play.Science362, 6419 (2018), 1140–1144.
- [9]Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. 2003. Packetclassification using multidimensional cutting. InProceedings of the 2003 con-ference on Applications, technologies, architectures, and protocols for computercommunications. 213–224.
- [10]Ion Stoica. 2019. Solving System Problems with Machine Learning.Studies inInformatics and Control28, 2 (2019), 119–132.
- [11]David E Taylor and Jonathan S Turner. 2007. Classbench: A packet classificationbenchmark.IEEE/ACM transactions on networking15, 3 (2007), 499–511.
- [12]Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2010. EffiCuts:optimizing packet classification for memory and throughput.ACM SIGCOMMComputer Communication Review40, 4 (2010), 207–218.

UFMS
UNIVERSIDADE FEDERAL
DE MATO GROSSO DO SUL

Dúvidas?

Obrigado!!!!!!