UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

DEPARTAMENTO DE FÍSICA

VITOR APARECIDO TRALDI

ESTUDO DA DEFORMAÇÃO DE PEROVSKITAS DE HALETO COM ORGANOMETÁLICOS ATRAVÉS DO CÁLCULO DO NÚMERO DE COORDENAÇÃO DO SÍTIO B

São Carlos, SP
2024

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE FÍSICA

VITOR APARECIDO TRALDI

ESTUDO DA DEFORMAÇÃO DE PEROVSKITAS DE HALETO COM ORGANOMETÁLICOS ATRAVÉS DO CÁLCULO DO NÚMERO DE COORDENAÇÃO DO SÍTIO B

Trabalho de conclusão de curso submetido ao Curso de Engenharia Física da Universidade Federal de São Carlos como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Física.

Orientador: Prof. Dr. Matheus Paes Lima

São Carlos, SP
2024

Agradecimentos

Antes de mais nada gostaria de agradecer pela vida, por poder ter o privilégio de estar encerrando esse ciclo que muitos nem sequer puderam tentar ou tiveram que interromper.

Gostaria de agradecer imensamente aos meus pais que são a minha base, se estou aqui hoje é por causa deles, sempre fizeram de tudo para eu ter o que eles não tiveram, sempre estiveram ao meu lado me apoiando e sempre foram exemplo de pessoas. Pai, mãe nós conseguimos, Amo muito vocês.

Gostaria de agradecer a minha noiva que passou todo esse processo ao meu lado vibrando durante as conquistas e sendo meu porto seguro durante as dificuldades. Sem você eu não teria conseguido, te amo.

Gostaria de fazer um agradecimento especial ao professor Matheus por ter aceito ser meu orientador e me proporcionado todo suporte necessário para o desenvolvimento do trabalho. Além do Natan e Lucas que me ajudaram durante o desenvolvimento do projeto. Foi um prazer ter essa troca com vocês.

Gostaria também de agradecer meus amigos que partilharam os dias e as dificuldades comigo e que me ajudaram muito a viver tudo isso. O último agradecimento vai à minha avó que faleceu pouco antes de eu ingressar no curso e que cuidou de mim quando era pequeno. Se o mundo tivesse mais pessoas como ela tudo seria melhor. Prometi a ela que iria formar e hoje estou cumprindo isso. Um beijo e um abraço enorme para a senhora e para o vô sinto muita saudade de vocês.

Por fim gostaria de deixar uma frase dita por minha mãe que levo para a vida. Independente da situação em que estamos vivendo lembre-se a gente só não dá jeito para morte.

"A gente só não da jeito para a morte"
(Ana Celia)

Resumo

O crescimento da demanda energética aliado ao impacto ambiental causado pelas fontes fósseis desencadeou grande demanda e estudo de fontes alternativas limpas e renováveis, principalmente a solar. Neste contexto, o desenvolvimento de dispositivos de conversão de energia solar buscam novos materiais para compor as células fotovoltáicas, dentre os quais as perovskitas têm ganhado atenção devido a sua diversidade, baixo custo de produção, e propriedades optoeletrônicas superiores, possibilitando uma alta eficiência de conversão de energia. As propriedades optoeletrônicas das perovskitas estão intimamente relacionadas com distorções estruturais, e ser capaz de explorar essas distorções permite um entendimento mais profundo desses materiais. Devido à estrutura peculiar das perovskitas, que possuem fórmula química ABX_3. Um sistema cúbico ideal, sem distorções, apresenta coordenação 6 no sítio B, sendo este um valor superior de referência. Baseado nisso, este trabalho explora a coordenação no sítio B como forma de quantificar as deformações estruturais das perovskitas. Especificamente, foi implementado três métodos diferentes para o Número de Coordenação, a saber, CN, ECN e ECN_{av}, e comparado os resultados mutuamente para um conjunto de Perovskitas de Haleto com Organometálicos disponíveis em um banco de dados de domínio público. Os resultados mostraram que o CN é totalmente dependente e sensível ao fator de corte p, não sendo apropriado para comparar diferentes sistemas entre sí, uma vez que não há uma forma de definir este fator univocamente. Em contrapartida, o ECN e ECN_{av} possuem resultados próximos e compatíveis entre si. Entretanto, o método ECN_{av} possui uma leve tendência a gerar resultados mais próximos do valor ideal (6). Por outro lado, o ECN resulta em uma distribuição mais homogênea. Essas diferenças se justificam pela forma algébrica específica de cada implementação. De fato, o ECN_{av} mostrou-se um método superior por ser livre de parâmetros externos, ou seja, não limita-se ao número de vizinhos considerados, e define a distância média através de um cálculo autoconsistente, diferentemente do ECN que restringe-se aos seis primeiros vizinhos.

Palavras-chave: Demanda Energética. Fontes Alternativas. Perovskitas de Haleto com Organometálicos. Distorções Estruturais. Número de Coordenação.

Abstract

The growth in energy demand, coupled with the environmental impact caused by fossil fuel sources, has triggered a significant demand for and study of clean and renewable alternative sources, particularly solar energy. In this context, the development of solar energy conversion devices has been seeking new materials to compose photovoltaic cells, among which perovskites have gained attention due to their diversity, low production cost, and superior optoelectronic properties, enabling high energy conversion efficiency. The optoelectronic properties of perovskites are closely related to structural distortions, and being able to explore these distortions allows for a deeper understanding of these materials. Due to the peculiar structure of perovskites, which have a chemical formula of ABX_3, an ideal cubic system without distortions presents a coordination number of 6 at the B site, which serves as a higher reference value. Based on this, this work explores the coordination at the B site as a way to quantify the structural deformations of perovskites. Specifically, three different methods for the coordination number were implemented, namely CN, ECN, and ECN_{av}, and the results were compared among them for a set of Organometal Halide Perovskites available in a public domain database. The results showed that CN is entirely dependent on and sensitive to the cutoff factor p, making it unsuitable for comparing different systems, as there is no way to define this factor unambiguously. In contrast, ECN and ECN_{av} have similar and compatible results. However, the ECN_{av} method tends to generate results closer to the ideal value (6), while ECN results in a more homogeneous distribution. These differences are justified by the specific algebraic form of each implementation. In fact, ECN_{av} proved to be a superior method as it is free from external parameters, meaning it is not limited to the number of neighbors considered and defines the average distance through a self-consistent calculation, unlike ECN, which is restricted to the first six neighbors.

Keywords: Energetic Demand. Alternative Sources. Organometal Halide Perovskites. Structural Distortions. Coordination Number.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABNT	Associação Brasileira de Normas Técnicas
DF	Departamento de Física
UFSCAR	Universidade Federal de São Carlos
CN	Coordenation Number (Número de Coordenação)
ECN	Effective Coordenation Number (Número de Coordenação Efetivo)
ECN_{av}	Effective Coordenation Number Average (Número de Coordenação Efetivo Médio)
PVK	Perovskita
UFV	Usinas Fotovoltaicas
EPE	Empresa de Pesquisa Energética
MHPs	Metal Halide Perovskites (Perovskitas de haleto com Metal)
PSCs	Perovskite Sollar Cells (Células solares de perovskita)
CFC	Estrutura cúbica de face centrada
ASE	Atomic Simulation Environment (Ambiente de Simulação Atômica)
VESTA	Visualization for Eletronic and Structural Analysis (Visualização para Análise Eletrônica e Estrutural)

Sumário

1 Introdução

1.1 Motivação e Justificativa

O uso demasiado de fontes fósseis para suprimento da demanda energética ocorre desde o século XVIII com a revolução industrial. Todavia, estes recursos experimentam uma cadente diminuição em suas reservas além de outros fatores conflitantes como: a degradação do meio ambiente, riscos à saúde humana, debates políticos e questões econômicas. Devido a esses fatores e outros mais, nas últimas décadas o incremento de fontes de energia alternativas tem se tornado gradativamente necessário. Em face a esses pontos, a energia solar acaba ganhando forte destaque por: ser considerada uma fonte limpa, abundante e viabilizar a geração de energia em lugares remotos (LONGO; PAOLI, 2003).

Nos últimos anos, a matriz energética brasileira passou a incorporar mais intensamente fontes de energia renovável. Incentivos governamentais facilitaram a popularização de usinas fotovoltaicas (UFV), cogeração de energia por biomassa e geradores eólicos, impulsionados pela busca por fontes de energia limpa e pela redução de custos decorrentes da evolução tecnológica no setor. Apesar da alta disponibilidade de luz solar no Brasil e dos incentivos governamentais, a energia fotovoltaica ainda representa uma pequena parte da matriz energética nacional, com apenas 2,47% (6.703 MW) da capacidade instalada (Empresa de Pesquisa Energética (EPE), 2022). No entanto, o interesse por UFVs aumentou significativamente a partir de 2018, impulsionado pela redução dos custos e pela implementação de políticas de geração distribuída. Investimentos têm sido feitos tanto na construção de grandes UFVs quanto na expansão da microgeração fotovoltaica para residências, pequenas empresas e áreas rurais. Porém, o alto custo e a baixa flexibilidade das células fotovoltaicas de silício têm limitado sua popularização. Entre as tecnologias emergentes, as células de perovskita destacam-se como uma alternativa promissora. Elas podem ser fabricadas por métodos de impressão em larga escala, o que reduz custos e consumo de energia. Além disso, essas células podem ser construídas em substratos leves e flexíveis, adequados para várias aplicações, como coberturas, automóveis, janelas e dentro de construções, convertendo a luz das lâmpadas em energia elétrica (IoT). Além disso, a produção por sinterização química reduz significativamente os custos em comparação aos módulos de silício cristalino convencionais. Apesar de todos esses pontos, vale ressaltar que essas células apresentam fatores limitantes em seu uso, principalmente, relacionados à instabilidade e toxicidade (com o Pb por exemplo) (SILVA, 2012).

A utilização de perovskitas (PVK) em células fotovoltaicas destaca-se também devido a alta eficiência na conversão energética. Para se ter uma ideia, em junho de 2024 foi estabelecido um novo recorde de eficiência de 34,6% em testes controlados, enquanto que em 2022 esse valor era por volta de 25%. Já os painéis de silício altamente comercializados apresentam uma

eficiência que varia entre 16% e 20% (SILVA, 2012; LONGi, 2024; JONES, 2024). A primeira aplicação de PVK's em células solares ocorreu em 2009, com uma eficiência de conversão de 3,8% (RAPHAEL et al., 2018). Este rápido aumento motivou grande parte da comunidade científica para explorar diferentes possibilidades desta aplicação, incluindo ligas e compósitos de perovskita.

É importante ressaltar que as perovskitas, em geral, possuem a fórmula química ABX_3, onde A é um cátion monovalente, B é um cátion bivalente, e X é um haleto (Cl, Br ou I). Sua estrutura cristalina ideal é análoga à do $CaTiO_3$, com uma célula cúbica que possui gaiolas octaédricas BX_6 interconectadas (mais detalhes nas próximas seções). Entretanto, essa estrutura pode sofrer deformações, como rotações entre os octaedros e flutuações nos comprimentos das ligações químicas. Pesquisas recentes mostram que essas deformações estão intimamente relacionadas às propriedades optoeletrônicas das perovskitas. Este trabalho busca quantificar essas distorções através da coordenação do sítio B. Este estudo representa um passo relevante para o entendimento das perovskitas, uma vez que as quantificações das distorções apresentadas aqui podem ser úteis em estudos futuros que correlacionem essas distorções com outras propriedades.

1.2 Revisão da Literatura

1.2.1 Estrutura das Perovskitas

"Descobertas por Gustav Rose, em 1839, as perovskitas representam uma classe de materiais com características únicas que hoje estão revelando inúmeras e versáteis aplicações em uma ampla gama de dispositivos tecnológicos"(RAPHAEL et al., 2018, apud GIORGI; YAMASHITA, 2015). O termo perovskita é utilizado para se referir a um grande grupo de compostos que possuem uma estrutura cristalina semelhante à do mineral perovskita $CaTiO_3$. A fórmula geral das perovskitas é ABX_3, onde A e B representam cátions, e X é um ânion. Os cátions na posição A são geralmente maiores e mais eletropositivos em comparação com os cátions na posição B, enquanto o sítio X é comumente ocupado por íons óxido ($O_2{}^-$) ou íons haletos (Cl^-, Br^-, F^-) (PETROVIć; CHELLAPPAN; RAMAKRISHNA, 2015).

A estrutura cúbica é um cristal ideal de PVK com grupo espacial Pm-3m, onde o cátion B possui seis ânions X como átomos mais próximos, e o cátion A possui doze ânions X mais próximos (PEÑA; FIERRO, 2001).

As perovskitas são divididas em dois grupos: **Perovskitas organometálicas**: Neste grupo, o cátion na posição A é monovalente, como metilamônio [$CH_3NH_3{}^+$] ou formamidínio [$CH(NH_2)_2{}^+$]; o cátion na posição B é um metal de transição, como chumbo [$Pb_2{}^+$] ou estanho [$Sn_2{}^+$]; e o ânion na posição X é um haleto, geralmente iodo [I−], bromo [Br−] ou cloro [Cl−], ou uma mistura desses haletos. **Perovskitas inorgânicas**: Neste grupo, o sítio A é normalmente ocupado por um metal do grupo I ou II, o sítio B por um metal de transição, e o sítio X por um

ânion (LIMA, 2021).

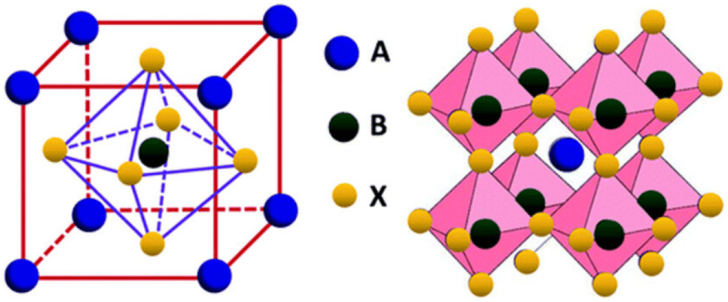


Figura 1.1 – Célula unitária e estrutura tridimensional da perovskita ideal. Fonte: (YI et al., 2019).

Algumas variações da estrutura ideal são conhecidas, representando as fases: ortorrômbica, romboédrica e tetragonal, como mostrado na figura 1.2. Essas estruturas distorcidas podem existir à temperatura ambiente, mas em altas temperaturas transformam-se em estruturas cúbicas, podendo passar por várias etapas intermediárias com fases distorcidas. Essas mudanças podem resultar de uma simples distorção da célula cúbica, de uma ampliação da mesma, ou de ambas (PEÑA; FIERRO, 2001).

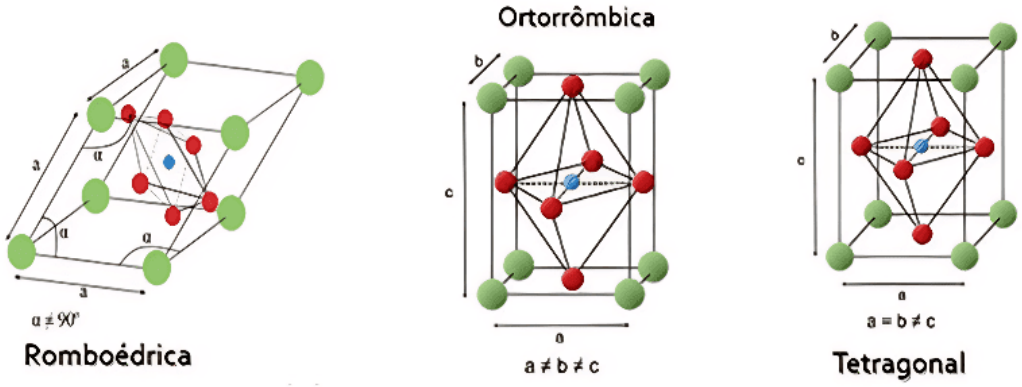


Figura 1.2 – Variações na estrutura da perovskita. Fonte: Adaptado de (MAYRINCK; FONSECA; SCHIAVON, 2020).

A formação da perovskita segue uma reação química que pode ser resumida como:

$$BX_2 + AX \longrightarrow ABX_3 \tag{1.1}$$

Onde A, B e X são íons que compõem as perovskitas híbridas. Um exemplo comum na literatura é a reação entre os percursores PbI_2 e CH_3NH_3I, que resulta em $CH_3NH_3PbI_3$ (conhecido como $MAPbI_3$). Com a facilidade de alterar os materiais das camadas dos dispositivos, a arquitetura e montagem das células solares foram modificadas, visando obter células mais eficientes e estáveis (MAYRINCK; FONSECA; SCHIAVON, 2020).

1.2.2 Perovskitas de Haleto com Metal

As Perovskitas de Haleto com Metal (MHPs) têm sido amplamente estudadas na última década devido ao seu grande potencial como semicondutores para dispositivos optoeletrônicos de alta performance, especialmente em células solares. As células solares de perovskita (PSCs) já alcançaram uma eficiência de conversão de energia superior a 25%. Este rápido avanço sugere uma revolução na tecnologia fotovoltaica de terceira geração. As MHPs possuem propriedades optoeletrônicas notáveis, como alto coeficiente de absorção, bandgap ajustável entre 400 e 800 nm, longo comprimento de difusão de portadores de carga (cerca de 1 μm) e alta condutividade de portadores de carga. O interesse por essas perovskitas também se deve à sua fabricação a partir de soluções com precursores baratos e abundantes. A versatilidade química das MHPs permite uma ampla gama de propriedades estruturais, ópticas e eletrônicas, resultantes de processos físico-químicos complexos durante a síntese. Além dos dispositivos fotovoltaicos, as propriedades ajustáveis das MHPs permitem aplicações em LEDs, fotodetectores e lasers (ZHAN et al., 2022).

As MHPs diferem de seus equivalentes convencionais de calcogeneto, que possuem grandes cátions bivalentes (por exemplo, Ca_2^+, Sr_2^+) no sítio A, pequenos cátions tetravalentes (por exemplo, Ti_4^+, Zr_4^+) no sítio B e ânions de oxigênio no sítio X. Em contraste, as estruturas das MHPs incluem cátions monovalentes no sítio A, cátions metálicos bivalentes no sítio B e ânions de halogênio no sítio X (ZHAN et al., 2022).

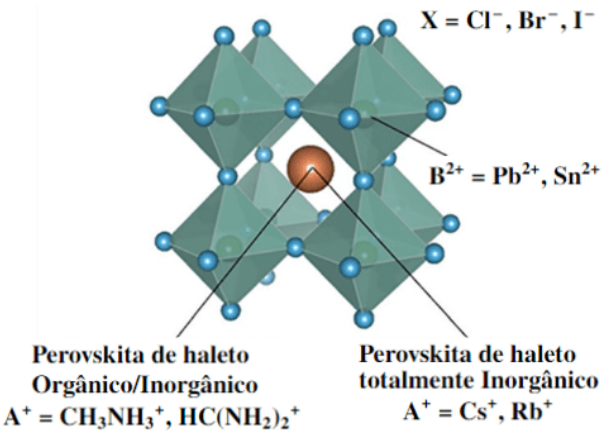


Figura 1.3 – Estruturas MHPs. Fonte: (ZHOU; ZHAO, 2019).

1.2.3 Distorções em Perovskitas

Embora as perovskitas ABX_3 tenham simetria cúbica em altas temperaturas, a maioria delas perde essa simetria em baixas temperaturas devido a deslocamentos atômicos. As distorções da estrutura ideal podem ocorrer de três maneiras: inclinação dos octaedros, distorções dos octaedros e deslocamentos dos cátions. A inclinação, que ocorre ao variar o raio iônico do sítio A, geralmente afeta mais os parâmetros da rede. Já as distorções dos octaedros são causadas pelo efeito Jahn-Teller (GOODENOUGH, 2004).

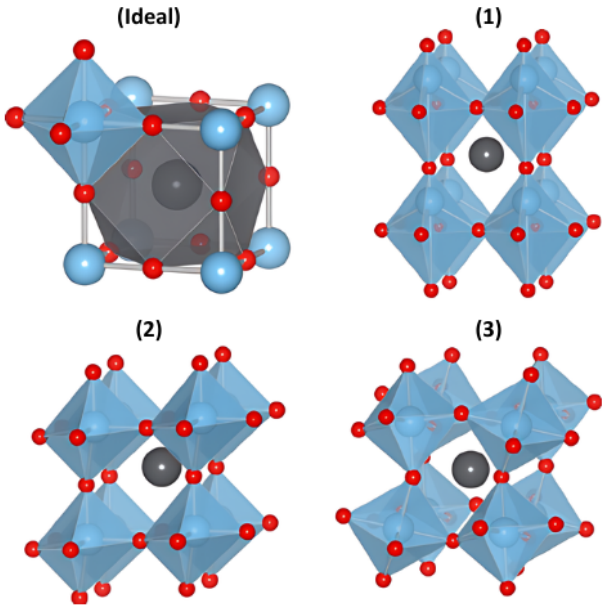


Figura 1.4 – Distorções da estrutura ideal. (Ideal) - Estrutura cúbica ideal, (1) - Deslocamento dos cátions, (2) - Distorção dos octaedros, (3) - Inclinação dos octaedros. Fonte: Adaptado de (ARMIENTO et al., 2014).

Estruturas do tipo perovskita são caracterizadas pela disposição BX_3 estável. Caso B apresente um raio (r_B) que seja menor que 0,51 Å, a separação B-X ideal não é atingida e consequentemente o arranjo se estabilizará em uma estrutura que apresente Número de Coordenação aniônica reduzido. Ao adicionar um cátion grande a este arranjo, uma distorção é gerada ao mesmo, levando à estrutura adotar outros grupos espaciais. Além disso, por haver um comprimento de ligação A-X ideal, uma estabilidade diferente é visualizada. Dessa forma, para avaliar os limites aceitáveis para o tamanho do cátion A, Goldschmidt formulou o fator de tolerância (PEREZ, 2000). O fator de tolerância é definido por:

$$t = \frac{(r_A + r_X)}{\sqrt{2}(r_B + r_X)}$$

onde, r_A, r_X e r_B são os raios teóricos dos íons.

Devido à sua geometria, a estrutura cúbica ideal tem um fator de tolerância t = 1. Assim, o fator de tolerância mede o quanto uma estrutura se desvia dessa configuração ideal. Estruturas do tipo perovskita geralmente ocorrem dentro do intervalo de 0,75 a 1,00 para t. No entanto, esse intervalo não é uma condição suficiente por si só, pois os cátions A e B também devem ser estáveis em suas respectivas coordenações octaédricas e dodecaédricas. Essa configuração reduz os enlaces para o raio iônico; em óxidos, esses enlaces correspondem a $r_A > 0,9$ Å e $r_B > 0,8$ Å. O valor de t a pressão e temperatura ambiente pode ser calculado a partir da soma dos raios iônicos, conforme tabelas empíricas. No entanto, o comprimento de ligação de equilíbrio A-X e B-X apresenta compressibilidade e expansão térmica diferentes. Por isso, o valor ideal de $t = 1$ só é alcançado em condições específicas de temperatura e pressão (PEREZ, 2000).

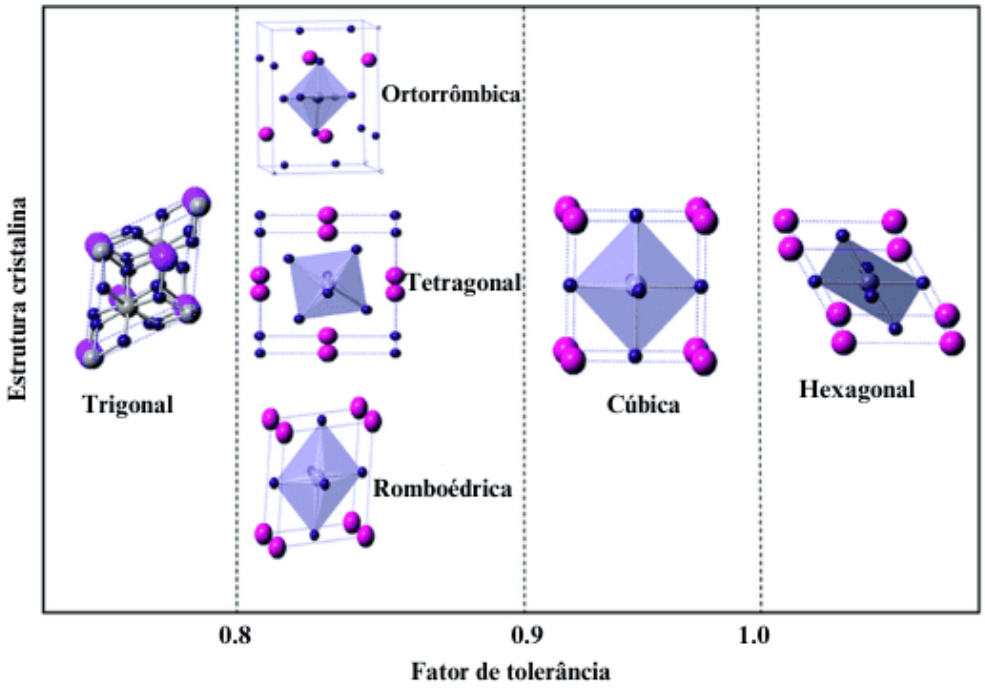


Figura 1.5 – Estrutura cristalina em função do Fator de tolerância. Fonte: Adaptado de (YI et al., 2019).

Outro aspecto importante para avaliar a estabilidade de uma perovskita é o fator octaédrico, que é calculado com base na relação entre os raios iônicos do cátion B e do ânion X. Para que o octaedro formado por esses íons seja estável, o valor do fator octaédrico deve estar entre 0,414 e 0,732 (QUEVEDO, 2022, apud LI; SOH; WU, 2004). O fator octaédrico é definido por:

$$\mu = \frac{r_B}{r_X}.$$

1.2.4 Número de Coordenação como Forma de Medir as Distorções em Perovskitas

As perovskitas de haleto apresentam uma estrutura flexível, devido ao íon haleto que forma pontes duplas (B-X-B). Isso permite diversas transições estruturais e distorções nos octaedros quando há aplicação de pressão ou mudanças de temperatura. Esses fatores causam inclinações e distorções nos octaedros com baixo custo energético, facilitando múltiplas transições de fase nas perovskitas ABX_3 (DIAS; LIMA; SILVA, 2021).

Dada a estrutura ideal da perovskita ABX_3. É possível observar que o octaedro é formado por um elemento central do sítio B envolto pelos elementos do sítio X. Se estes elementos envoltos apresentam a mesma distância de ligação com o átomo central temos um estrutura simétrica ou cúbica. Caso essas distâncias sejam divergentes ocorre o que conhecemos como distorção do octaedro. Dessa forma, podemos obter estruturas como a tetragonal, ortorrômbica e romboédrica conforme apresentado na Seção 1.2.1.

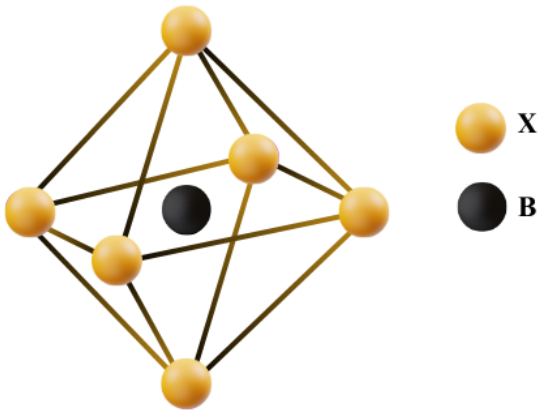


Figura 1.6 – Representação do octaedro.

Tendo em vista a relevância destas distâncias interatômicas na distorção dos octaedros, inicia-se a utilização do Número de Coordenação como forma de quantificar essas distorções visto que ele parte do seguinte princípio: dado um átomo central o Número de Coordenação representa o número de átomos vizinhos que o circundam dado um raio de corte que normalmente acaba sendo a distância interatômica entre o átomo central e o vizinho mais próximo. Dentre as estruturas já mencionadas no trabalho, a cúbica, ao qual é chamada de ideal, apresenta coordenação igual a 6. As demais estruturas apresentam coordenação diferente a 6 com amplo range de variação.

Todavia, como será visto nos próximos capítulos, o Número de Coordenação (CN) é limitado e como forma de o aperfeiçoar novos métodos mais requintados foram desenvolvidos para determinação, como o Número de Coordenação Efetivo (ECN) e o Número de Coordenação Efetivo Médio (ECN_{av}). A aplicação desses métodos juntamente ao estudo da distorção podem ser encontrados em: (DIAS; LIMA; SILVA, 2021) com o estudo do "Papel das fases estruturais e distorções de octaedros nas propriedades optoeletrônicas e excitônicas de perovskitas $CsGeX_3$ (X = Cl, Br, I)"e (DANELON et al., 2024) ao discutir o "Contraste da estabilidade, distorções octaédricas e propriedades optoeletrônicas de 3D $MABX_3$ e 2D $(BA)_2(MA)B_2X_7$ (B = Ge, Sn, Pb; X = Cl, Br, I) perovskitas".

1.3 Objetivo

Apesar de todos os estudos e trabalho já apresentados na literatura. Uma análise mais ampla (com maior número de materiais e outras metodologias para o cálculo da coordenação) poderá demonstrar o potencial emprego da coordenação na descrição das distorções de perovskitas, que pode ser útil em trabalhos futuros e melhor difundido no meio científico, principalmente, nos estudos teóricos de perovskita.

Dessa forma, busca-se analisar com este trabalho a aplicação das técnicas conhecidas/abordadas junto ao cálculo da coordenação (CN, ECN e ECN_{av}) para o maior número de perovskitas possíveis. Comparando seus resultados e discutindo suas eficiências, tendências e limitações.

2 Metodologia

2.1 Introdução ao Número de Coordenação

Embora o modelo atômico quântico ofereça uma compreensão mais rica das interações eletrônicas e da natureza dos átomos. Para a definição do conceito de número de coordenação utiliza-se o modelo de esferas rígidas por contribuir na visualização e análise das estruturas, especialmente em contextos onde a geometria e as interações espaciais são mais relevantes.

Segundo MULLER (1994), o Número de Coordenação de um átomo especificado em uma espécie química é o número de outros átomos ligados diretamente a esse átomo. Por exemplo, o Número de Coordenação do carbono no metano é quatro, e é cinco no metano protonado.

De acordo com HELMENSTINE (2019), o Número de Coordenação de um átomo em uma molécula é o número de átomos ligados a esse átomo. Na química e na cristalografia, o Número de Coordenação descreve o número de átomos vizinhos em relação a um átomo central.

Em física do estado sólido, o conceito de "Número de Coordenação"(CN) é empregado para indicar quantos átomos ou grupos estão ligados a um átomo. É essencial destacar que o Número de Coordenação é relevante em diversas estruturas químicas que envolvem ligações, abrangendo moléculas, complexos de íons metálicos e arranjos cristalinos de sólidos (CHEMISTRYTALK, s.d.).

Com ele é possível compreender o empacotamento e o arranjo estrutural dos átomos em um cristal. Por exemplo, em uma estrutura cúbica de face centrada (cfc), cada átomo está rodeado por 12 vizinhos mais próximos, resultando em um Número de Coordenação de 12. Esse número afeta diretamente propriedades como densidade, ponto de fusão e resistência mecânica do material.

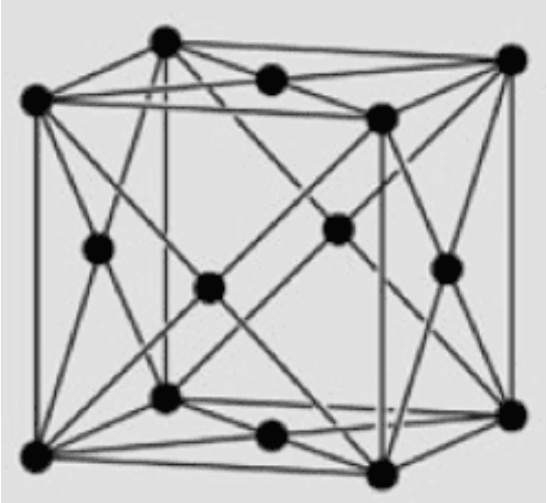


Figura 2.1 – Estrutura cúbica de face centrada. Fonte: (KITTEL, 2006).

Diversos fatores influenciam o Número de Coordenação, incluindo características dos íons metálicos e dos ligantes. O tamanho e a forma dos ligantes são determinantes, com ligantes maiores geralmente resultando em Números de Coordenação menores. O tamanho do íon metálico também é crucial, pois íons maiores podem acomodar mais ligantes, aumentando o número de coordenação. Além disso, a configuração eletrônica do íon metálico é significativa, já que certas configurações preferem Números de Coordenação específicos para estabilizar as energias eletrônicas. É importante considerar as interações intermoleculares, que no estado sólido ocorrem entre complexos vizinhos e em solução entre o complexo e as moléculas do solvente, resultando em diferentes Números de Coordenação e estruturas dependendo do estado (LANDSKRON, s.d.).

2.1.1 Número de Coordenação (CN)

Para determinar o Número de Coordenação dado um átomo (X) e suas espécies vizinhas (X'), inicialmente defini-se o raio de corte, fator para determinar quais átomos são vizinhos a um átomo, como:

$$R_{\text{cut}}(X, X') = D_{\text{min}}(X, X') \times (1 + p) \tag{2.1}$$

onde, $D_{\text{min}}(X, X')$ é a distância mínima entre o átomo da espécie (X) e os átomos vizinhos de espécie (X') e p, porcentagem manualmente definida, representa as possíveis flutuações nas distâncias interatômicas entre as espécies. Vale ressaltar que essas flutuações podem ocorrer devido à fatores como: vibrações térmicas, pressão externa, efeitos quânticos, desordem estrutural entre outros.

Definido o raio de corte, é possível obter o CN através da expressão:

$$CN_i(X, X') = \left(\sum_{j | D(X^i, X'^{(j)}) \leq R_{cut}(X, X')} 1 \right) \tag{2.2}$$

Ou seja, dado um átomo de índice i, seu Número de Coordenação será o somatório de todas as distâncias entre o par de espécie (XX') que forem menor e iguais ao raio de corte.

2.1.2 Número de Coordenação Efetivo (ECN)

O Número de Coordenação representa o total de átomos ligados a um átomo central em um poliedro de coordenação. Entretanto, em poliedros de coordenação que são relativamente distorcidos, descrever a coordenação do átomo central por meio de um único número pode ser um desafio (MOMMA; IZUMI, 2008).

Ainda conforme MOMMA; IZUMI (2008), diversas metodologias são sugeridas para determinar o Número de Coordenação Médio ou "Efetivo"(ECN). Essas metodologias, somam todos os átomos ao redor usando um esquema de ponderação. Nesse esquema, os átomos circundantes não são contados como inteiros, mas como frações, com valores variando entre 0 e

1. Assim, conforme aumenta-se a distância entre átomo central e circundante o número varia inversamente tendendo a zero. Dessa forma, o ECN é definido como apresentado a seguir.

Primeiramente, determina-se a distância média ponderada entre os pares de espécie, considerando os 6 primeiros vizinhos.

$$D_{\text{av}}(X, X') = \frac{\sum_{ij} D(X^i, X'^j) e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\min}(X,X')}\right)^6\right]}}{\sum_{ij} e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\min}(X,X')}\right)^6\right]}} \tag{2.3}$$

onde, $D_{\text{min}}(X, X')$ é a distância mínima entre o átomo central da espécie X e os 6 primeiros átomos vizinhos de espécie X' e $D(X, X'^j)$ é a distância entre o átomo central de índice i da espécie X e o átomo vizinho de índice j e espécie X'.

Após determinar a distância média ponderada, O ECN é obtido pela seguinte expressão:

$$\text{ECN}_i(X, X') = \sum_j e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\text{av}}(X,X')}\right)^6\right]} \tag{2.4}$$

2.1.3 Número de Coordenação Efetivo Médio (ECN_{av})

Segundo (SILVA, 2011), o conceito padrão de coordenação, atribui um peso ($\text{W}_{XX'}$ = 1,0) para todos os comprimentos de ligação entre um átomo central e os átomos circundantes ($\text{D}_{XX'}$) que estejam dentro de um determinado parâmetro de corte (D_{cut}). Desse modo, o Número de Coordenação é obtido ao contar os comprimentos de ligação que são menores que esse parâmetro (D_{cut}), resultando em valores inteiros. Este conceito é facilmente aplicável a estruturas simétricas, onde o parâmetro (D_{cut}) pode ser estabelecido com base nas distâncias dos vizinhos mais próximos (NNs). Em contraste, no conceito de coordenação efetiva, um peso diferente é calculado para cada comprimento de ligação ($\text{D}_{XX'}$) usando uma função de peso, ou seja, ($\text{W}_{XX'} \neq 1{,}0$) para todos os pares XX'.

Essa abordagem fundamenta-se na observação de que um átomo X específico forma ligações mais fortes com os átomos X' mais próximos. Assim, ajustes sutis nos ambientes de coordenação podem ser considerados. Todos os valores de $\text{W}_{XX'}$ são calculados em relação ao comprimento de ligação ponderado pelo átomo, D_{av}i, que deve ser determinado para cada átomo X. Como exemplo, se os comprimentos de ligação forem menores (ou maiores) que o comprimento de ligação ponderado local, D_{av}, contribuirão com valores de $\text{W}_{XX'}$ maiores (ou menores) que a unidade. Sendo assim, o Número de Coordenação Efetivo average (ECN_{av}) para um átomo específico X (ECN_i) será calculado pela soma de todos os pesos $\text{W}_{XX'}$, e, consequentemente, pode não resultar em um número inteiro. Tornando, dessa forma, aplicável a estruturas tanto simétricas quanto distorcidas (SILVA, 2011).

Para esta abordagem, será utilizado função exponencial de sexta potência buscando aferir o ECN_i para todos os átomos. Função essa, que tem seus primórdios na qúimica orgânica e foi rigorosamente avaliada para as redes de Bravais mais comuns no trabalho apresentado por SILVA (2011).

Como para o ECN, inicialmente determina-se a distância média. Porém diferentemente do método anterior este não limita-se apenas aos 6 primeiros vizinhos.

$$D_{\mathrm{av}}(X, X') = \frac{\sum_{ij} D(X^i, X'^j) e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\mathrm{av}}(X,X')}\right)^6\right]}}{\sum_{ij} e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\mathrm{av}}(X,X')}\right)^6\right]}} \tag{2.5}$$

Além disso, este cálculo é autoconsistente. Dessa forma, é atribuído $D_{\mathrm{av}}(X, X')$ igual a $D_{\mathrm{min}}(X, X')$ do lado direito da equação e obtido o valor correspondente de $D_{\mathrm{av}}(X, X')$. Obtido o primeiro valor de $D_{\mathrm{av}}(X, X')$ inicia-se a autoconsistência, até que a seguinte relação seja satisfeita:

$$|D_{\mathrm{av}}^{\mathrm{new}}(X, X') - D_{\mathrm{av}}^{\mathrm{old}}(X, X') < 0,00010| \tag{2.6}$$

Satisfeita a relação, O $\mathrm{ECN}_{\mathrm{av}}$ de uma dado átomo X, índice i e X' vizinhos, é obtido por:

$$\mathrm{ECN}_i(X, X') = \sum_j e^{\left[1-\left(\frac{D(X^i,X'^j)}{D_{\mathrm{av}}(X,X')}\right)^6\right]} \tag{2.7}$$

2.1.4 Banco de Dados de Perovskitas de Haleto com Organometálicos

Todas perovskitas analisadas durante o desenvolvimento deste trabalho advém de um banco de dados denominado Organometal Halide Perovskites.

Como descrito na página que disponibiliza os dados, para a criação do banco, foi realizado cálculos de estrutura eletrônica de 240 perovskitas compostas de Cs, CH_3NH_3 e $HC(NH_2)_2$ como cátion A, Sn e Pb como íon B e uma combinação de Cl, Br e I como ânions (CASTELLI et al., 2014). Vale ressaltar que através deste conjunto de dados é possível obter informações de bandgap indireto, bandgap direto, grupo espacial, simetria, fórmula, vetores de rede entre outras. Portanto, a técnica de explorar bancos de dados com algum fim específico pode ser empregada para outras finalidades.

2.1.5 Jupyter Notebook

Para criação do código desenvolvido neste trabalho foi utilizado o Jupyter. De acordo com DATABRICKS (2024), ele é "um aplicativo da web de código aberto usado principalmente por cientistas de dado para criar e compartilhar documentos contendo códigos ao vivo, fórmulas

e outros recursos multimídia". Além disso, sua utilização abrange várias atividades de análise de dados, como análise exploratória, limpeza e transformação de dados, visualização, modelagem estatística, aprendizado de máquina e deep learning.

Vale ressaltar que a ferramenta possibilita a criação de documentos dinâmicos que incluem não apenas códigos, mas também textos formatados, imagens, gráficos e equações matemáticas. Esses documentos podem ser compartilhados para permitir a colaboração de múltiplos usuários, seja por meio de repositórios no GitHub, Inc. ou pelo serviço online Jupyter Notebook Viewer (CARDOSO, 2023).

2.1.6 Implementação do Código

Inicialmente foi escolhido um banco de dados que dispusesse número considerável de perovskitas e propriedades físico-químicas para análise. Determinado o conjunto de dados, o primeiro método a ser implementado foi o CN que é o mais simples e de maior aplicação na literatura. Na sequência, foi implementado o método ECN e por fim o ECN_{av}.

Para manipulação e obtenção dos dados advindos do banco, foi utilizado linguagem python e algumas bibliotecas como: ASE (ase.db, ase.io, scipy.spatial, ase.geometry), Numpy, dentre outras.

De modo geral, os códigos leem as informações de cada perovskita presente no banco, extraem informações relevantes como: símbolos, átomos e vetores de rede; fazem a classificação do que é A, B e X; determinam os vizinhos e suas respectivas distâncias a partir da célula unitário; e calculam a coordenação, de acordo com cada abordagem, como apresentado nas seções 2.1.1, 2.1.2 e 2.1.3.

Os códigos completos, juntamente com os comentários relevantes para o entendimento, estão disponíveis no apêndice na seção B.

3 Resultados e Discussão

Neste capítulo serão apresentados, interpretados e analisados todos os resultados alcançados com a aplicação dos métodos para as perovskitas disponíveis no banco de dados utilizado.

Devido ao elevado número de perovskitas examinadas e à consequente dificuldade durante a plotagem e apresentação gráfica, foram selecionadas 10 perovskitas que fossem representativas do montante presente no conjunto de dados, para realização de algumas análises mais detalhadas. A seleção foi baseada na coordenação, de acordo com os valores encontrados nos três métodos, de forma a incluir representantes de todas as coordenações encontradas nos materiais em estudo.

As perovskitas selecionadas para algumas análises trazidas neste capítulo estão disponíveis na tabela a seguir. O acesso a todas disponíveis no banco pode ser encontrado no apêndice do trabalho na seção A.

Índice	**Perovskita**
1	$PbBrCl_2Cs$
2	$PbIBrClCs$
3	$SnCl_3CH_5N_2$
4	$Br_{12}Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
5	$Br_2Cl_4Br_2I_4Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
6	$I_8Br_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
7	$I_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
8	$I_4Cl_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
9	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_8Cl_4Pb_4$
10	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Br_4Cl_4Sn_4$

Tabela 3.1 – Lista de perovskitas representativas do conjunto. Fonte: (CASTELLI et al., 2014).

Na figura 3.1 é possível observar o valor da coordenação para cada PVK descrita acima considerando os três diferentes métodos, além do elemento que ocupa o sítio B do material e o valor da coordenação ideal (6) para perovskitas sem deformação estrutural. Vale ressaltar que o cálculo do CN foi feito levando em consideração 4 diferentes valores do parâmetro p, ou seja, 5%, 10%, 20% e 30%. A mesma visualização para todos os materiais do banco está disponível no apêndice do trabalho na seção C.

Analisando os resultados da imagem 3.1, nota-se grande variação no método CN para os diferentes valores de p. Conforme aumenta-se o parâmetro maior se torna o raio de corte, mais átomos são considerados como ligados ao átomo central e consequentemente maior será o Número de Coordenação. Esse comportamento pode ser visualizado, principalmente, nas estruturas 4, 5 e 9. Evidenciando-se o quão dependente e sensível o método se mostrou para com este fator, principalmente para PVK com deformação estrutural (coordenação diferente a 6). Vale ressaltar que este comportamento discutido estende-se para as demais estruturas não citadas.

Os métodos ECN e ECN_{av}, de maneira geral, apresentaram valores semelhantes, com uma leve tendência de coordenação maior para o ECN_{av}, especialmente em perovskitas com deformações, exceto para a PVK 4 com maior divergência. Sem depender de definições ou ajustes adicionais manuais. Essa maior coordenação é justificada por o ECN limitar-se aos primeiros vizinhos na determinação das distâncias fazendo com que sejam consideradas distâncias de ligação um pouco menores do que as obtidas pelo ECN_{av}.

Além disso, as perovskitas 1 e 2 mostraram coordenação próximas à 6 em todas as variações e metodologias empregadas. O que também foi possível constar nas demais PVK's não abordadas. Reforçando assim, que para estruturas simétricas os 3 métodos apresentam resultados válidos para aferição das distorções.

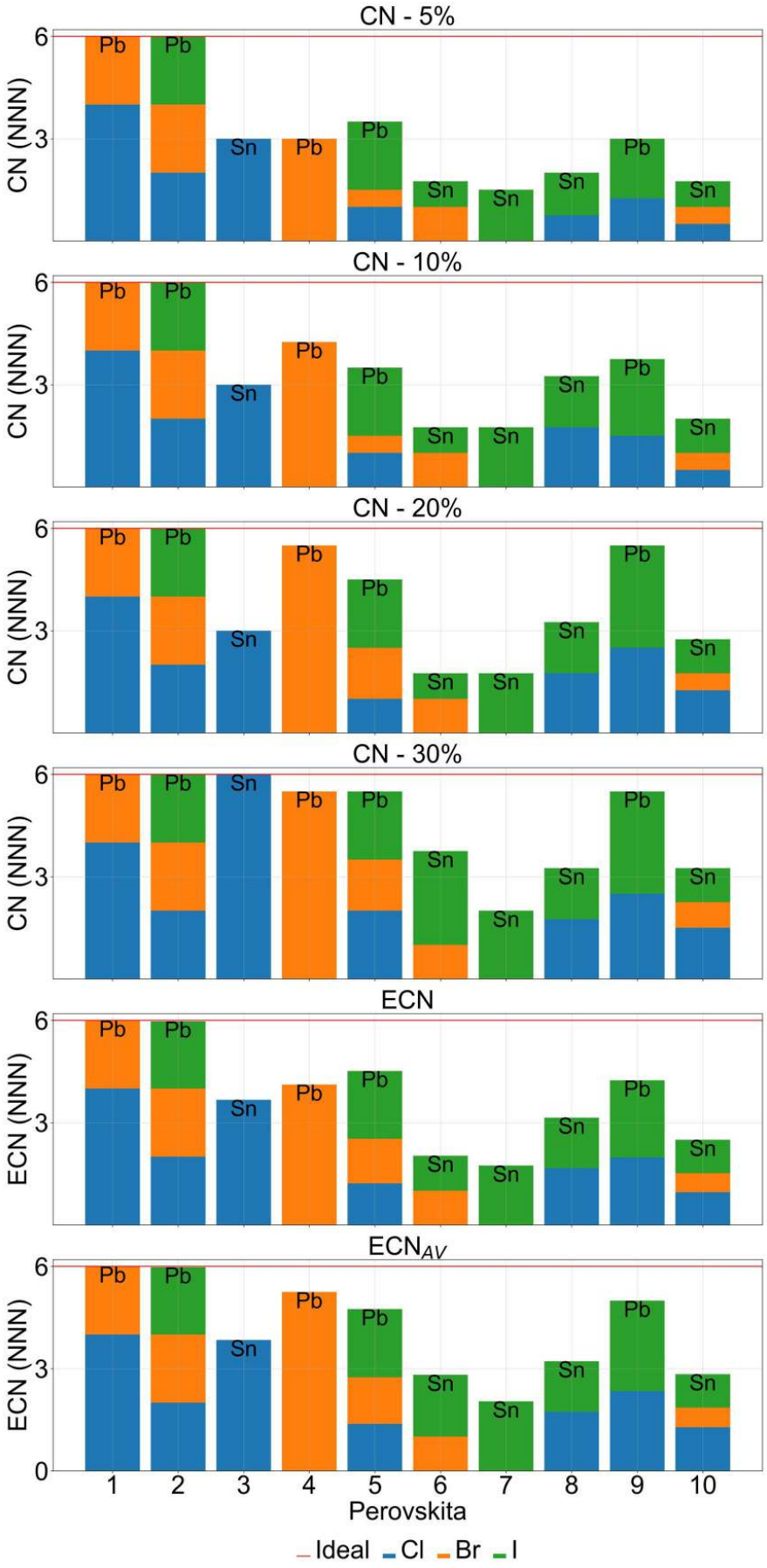


Figura 3.1 – Coordenação para os diferentes métodos, com segmentação por ânion X e indicação do íon B em função da perovskita. Para as 10 estruturas selecionadas.

Ainda com os 10 casos, através da figura 3.2 é possível verificar a deferença entre as distâncias calculadas em cada expressão. Os espaços vazios nos gráficos demonstram que a perovskita em questão não apresenta o elemento em sua composição. Além do mais, os gráficos presentes na imagem em razão dos elementos do sítio X (Cl, Br e I) podem ser justificados por cada elemento do sítio ter uma respectiva distância. Assim, por exemplo, a perovskita 1 tem em sua composição Cl e Br e consequentemente duas distâncias diferentes. Enquanto que a 4 apenas uma.

Os três métodos se mostraram consistentes e próximos na determinação das distâncias com uma pequena alteração para o ECN_{av} que pode ser justificada pela forma como são determinadas essas distâncias. Para o CN é utilizado a distância mínima entre o átomo central e os vizinhos, no ECN é feita uma média ponderada considerando os seis primeiros vizinhos e no ECN_{av} também é feita uma média ponderada porém não limitando-se aos primeiros vizinhos e realizando um cálculo autoconsistente até a relação 2.6 ser satisfeita.

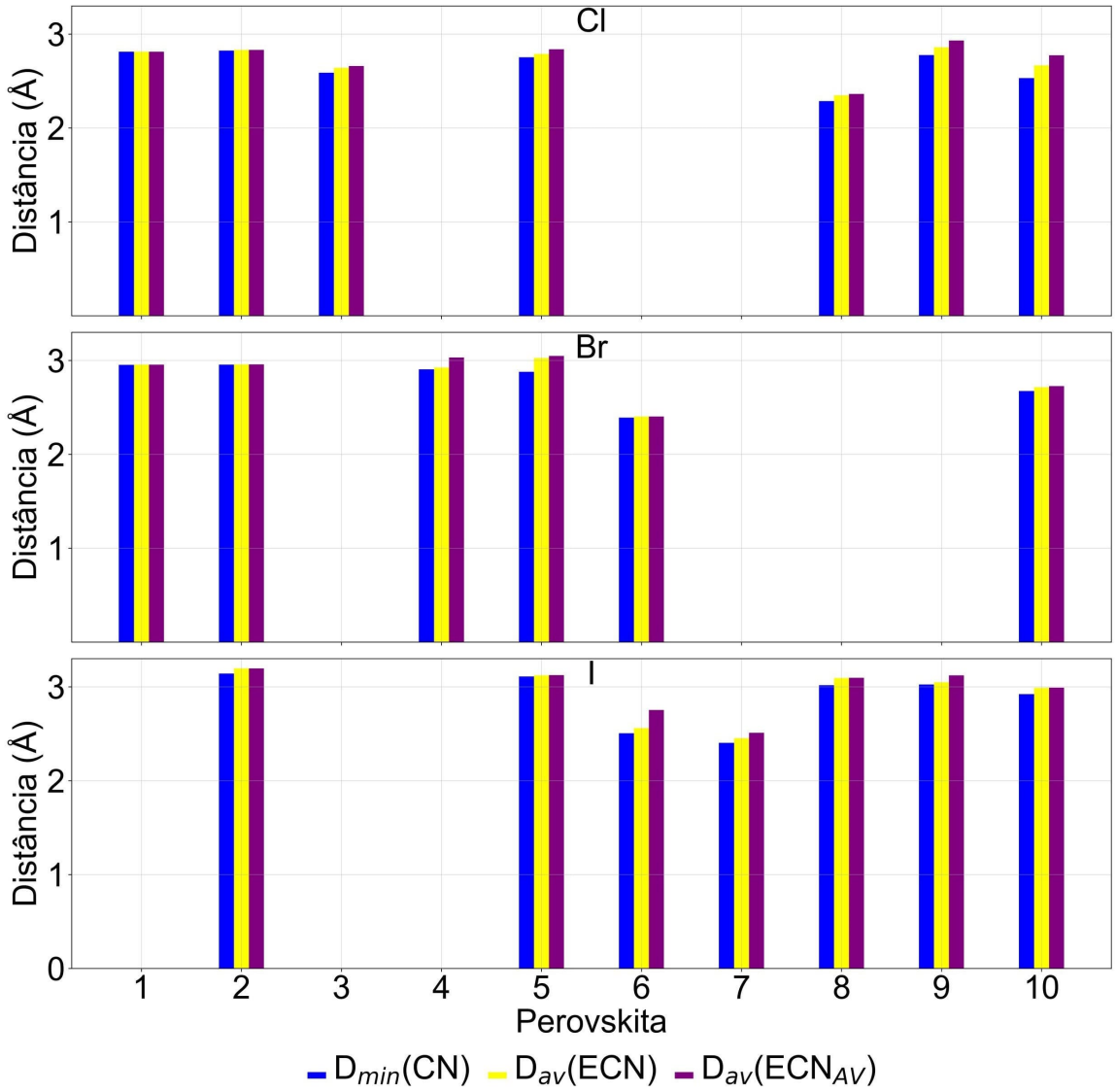


Figura 3.2 – Distância para cada ânion X, com segmentação por método em função da perovskita. Para as 10 estruturas selecionadas.

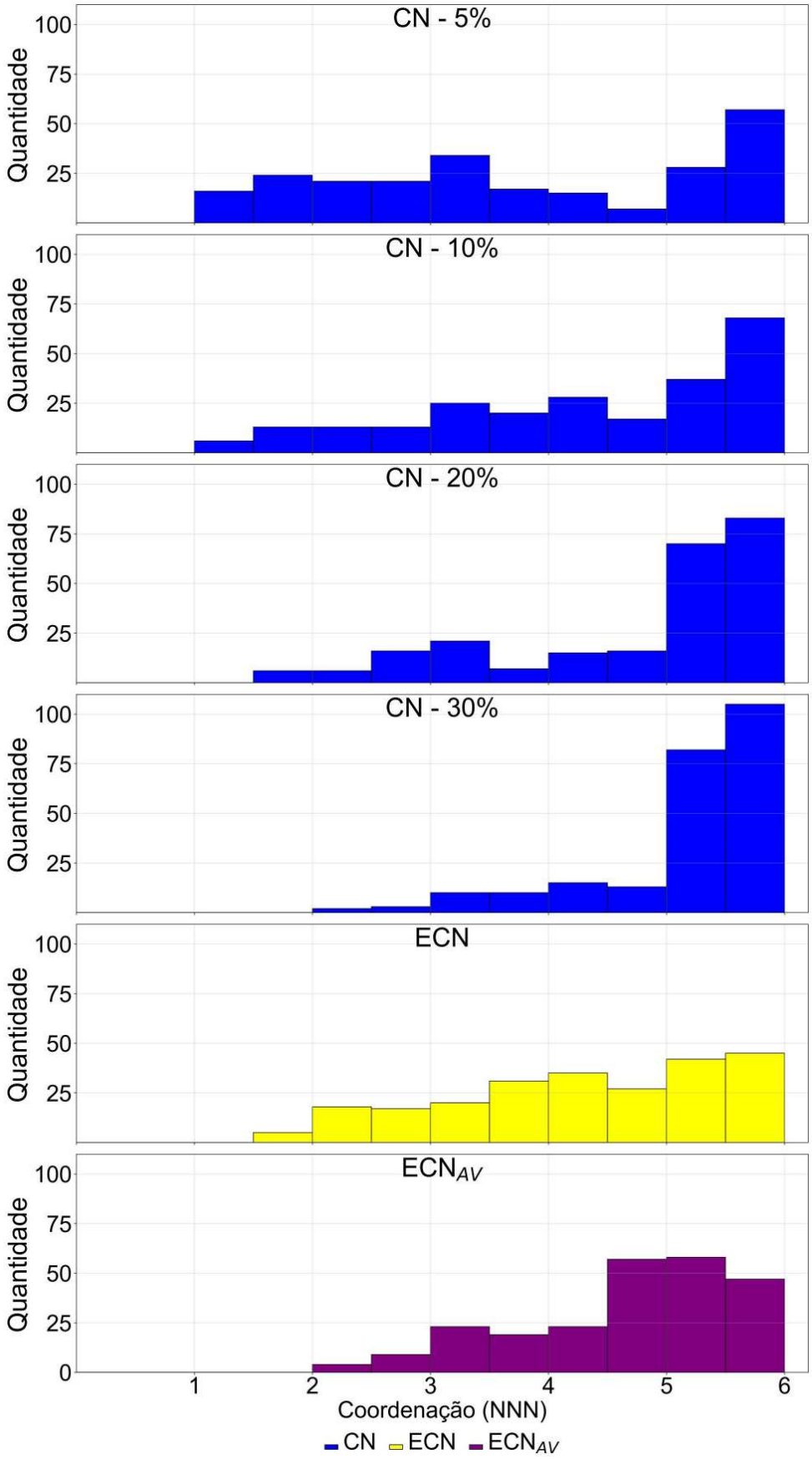


Figura 3.3 – Frequência de coordenação global para cada metodologia em estudo. Para todas as estruturas do banco de dados.

A imagem 3.3 ilustra a frequência de coordenação para cada método levando em consideração todas as 240 perovskitas presentes no banco de dados. Analisando os quatro primeiros gráficos que contemplam o CN é possível verificar o aumento da coordenação junto ao aumento da porcentagem. Comparando as porcentagens 5% e 30% respectivamente, nota-se que a primeira apresenta coordenação mínima igual a 1 enquanto que a segunda igual a 2. Ao ser analisado também a quantidade entre o intervalo 5-6, para ambas as porcentagens, observa-se quantidade próxima a 75 para a primeira e 180 para a segunda. Reforçando desse modo, a idéia levantada anteriormente da sensibilidade/dependência do método ao parâmetro p.

Olhando os outros dois métodos (ECN e ECN_{av}) constata-se que ambos apresentam uma certa similaridade entre as distribuições, no entanto o ECN_{av} tem maior concentração no intervalo de 4,5-6 e maior coordenação mínima começando em 2 enquanto que o ECN tem certa homogeneidade na distribuição das coordenações e coordenação mínima menor começando em 1,5. Quantificando desta forma a diferença presente nos métodos apresentada anteriormente.

As duas ilustrações que seguem (3.4 e 3.5) também demonstram a frequência de coordenação para cada método. Contudo, diferentemente da figura 3.3, agora a segmentação é feita por íon B (3.4) e ânion X (3.5).

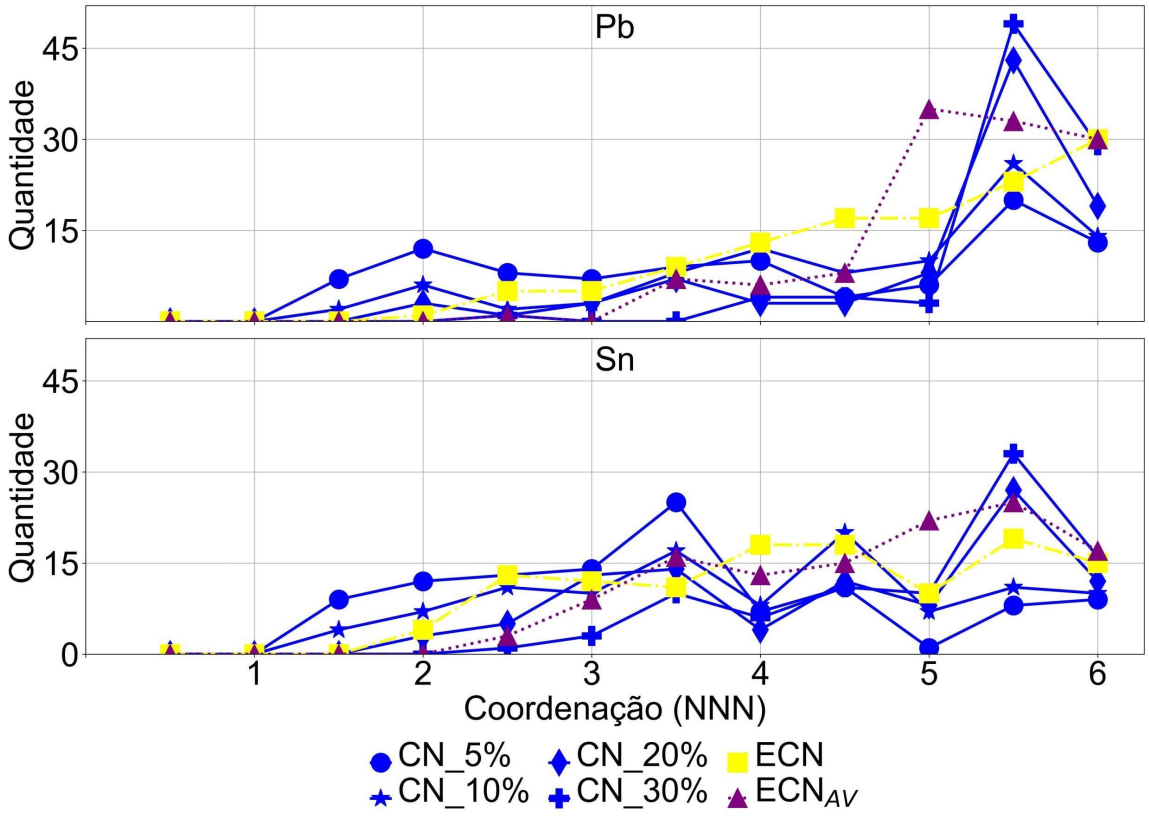


Figura 3.4 – Frequência de coordenação global para cada íon B, com segmentação por metodologia em estudo. Para todas as estruturas do banco de dados.

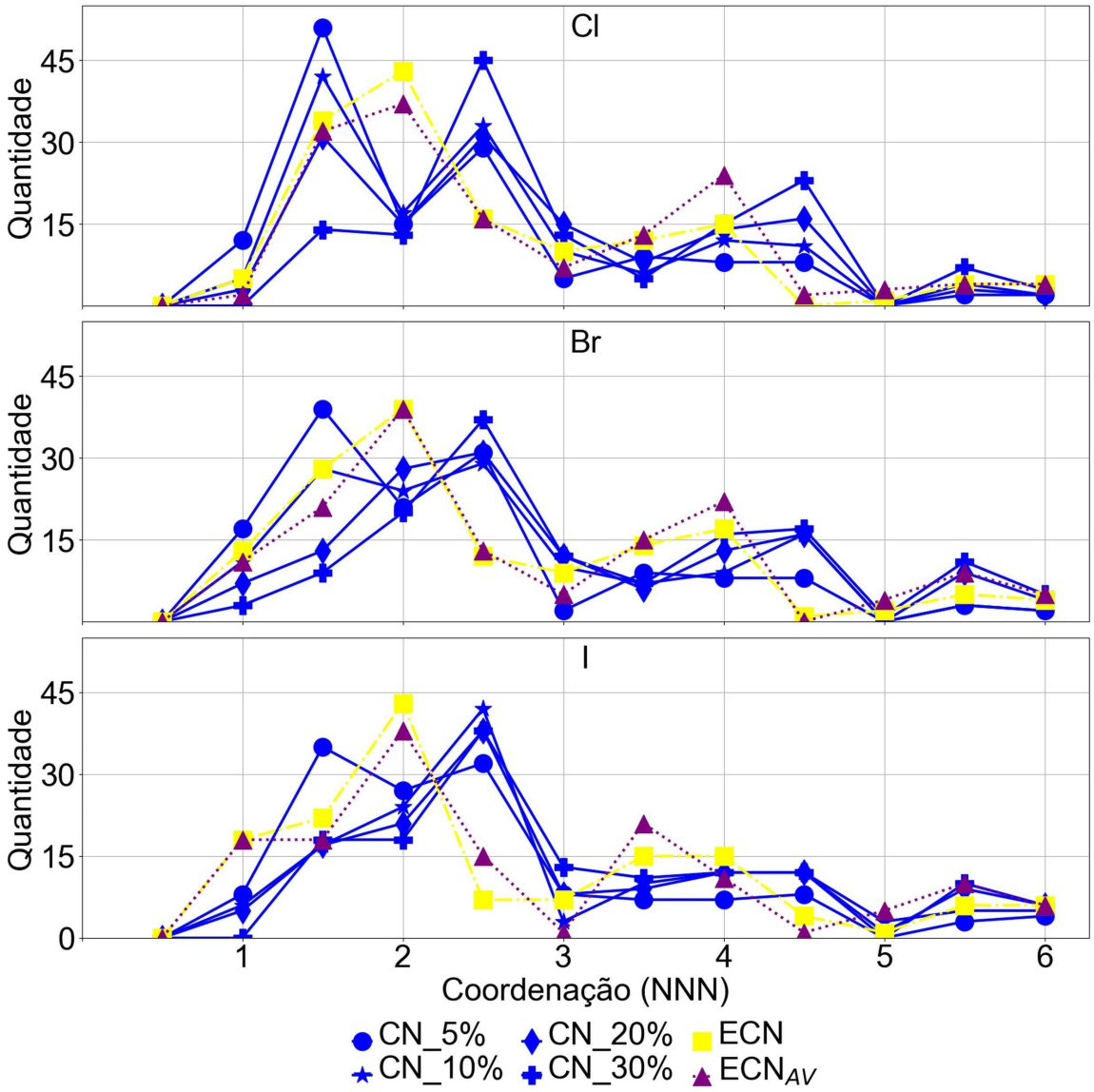


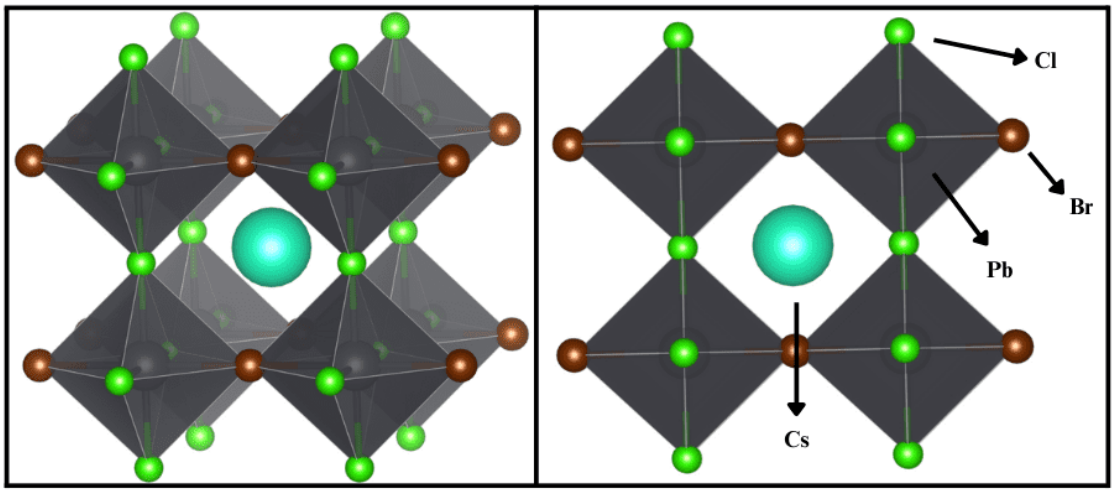
Figura 3.5 – Frequência de coordenação global para cada ânion X, com segmentação por metodologia em estudo. Para todas as estruturas do banco de dados.

Averiguando-as, foi possível constatar que as perovskitas com íon B = Pb apresentaram maior concentração de coordenação entre 5-6 enquanto que as com íon B = Sn demonstram valores de coordenação mais homogeneamente distribuídos. Com isso, percebe-se que o chumbo tende a gerar octaedros mais estáveis que o estanho.

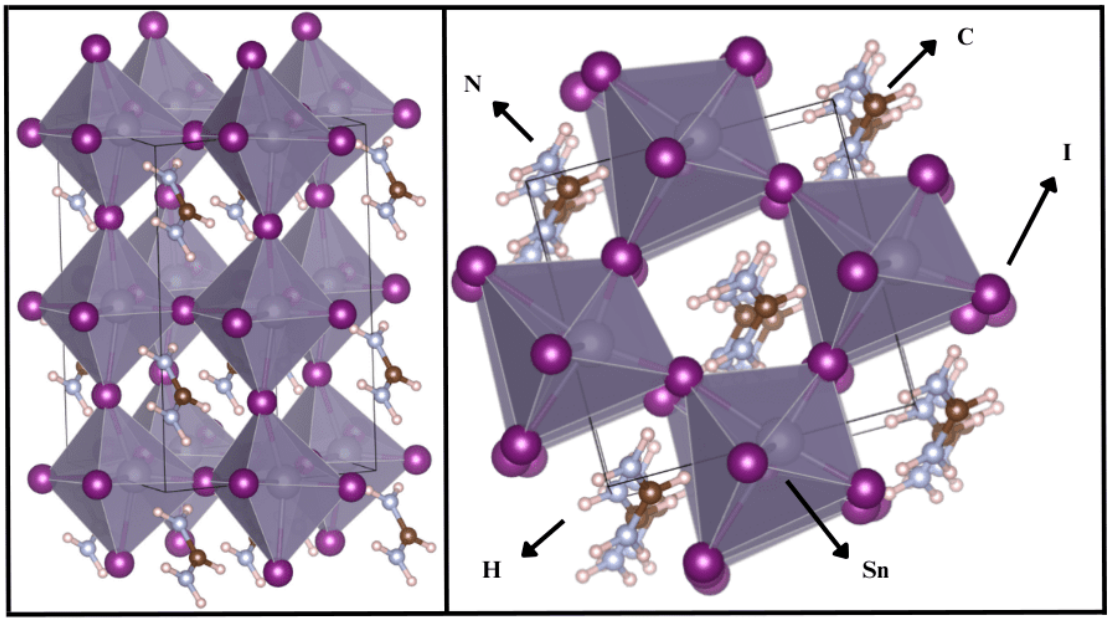
Além disso, os resultados obtidos quando feita a quebra por ânion X se demonstraram uniformes para os três elementos (Cl, Br, I). Atestando desse jeito, que o fator central para determinação das deformações não é o sítio X mas sim o sítio B.

Por fim, temos nas imagens 3.6a e 3.6b a visualização estrutural obtida através do software VESTA (MOMMA; IZUMI, 2011) para uma perovskita com coordenação próxima a 6 e outra próxima a 2 respectivamente. Ambas as perovskitas fazem parte das 10 selecionadas para as visões iniciais, logo é possível visualizar o comportamento das distâncias além da coordenação. Além disso, é possível pontuar que a perovskita com coordenação 2 apresenta considerável

deformação em sua estrutura enquanto que a com 6 grande simetria. Esses resultados ressaltam o emprego da coordenação do sítio B como forma de demonstrar deformações na estrutura.



(a) Perovskita $PbBrCl_2Cs$ com coordenação próxima a 6 e respectiva vista superior.



(b) Perovskita $I_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$ com coordenação próxima a 2 e respectiva vista superior.

Figura 3.6 – Visualização da deformação estrutural em perovskitas com coordenações distantes. Fonte: (MOMMA; IZUMI, 2011).

4 Conclusão

Neste trabalho, foi empregado três metodologias com formulações matemáticas distintas no estudo das distorções em perovskitas de haleto com organometálicos presentes em um banco de dados. São elas; CN, ECN e ECN_{av}. A primeira, e mais encontrada na literatura, determina a coordenação como sendo o número de átomos próximos a um átomo central dentro de uma distância limite (raio de corte) que é determinada manualmente e depende de um parâmetro (p). Não preocupando-se assim com as diferentes distâncias de ligação e seus impactos.

As demais metodologias, ECN e ECN_{av} respectivamente, surgem na tentativa de melhor quantificar as distorções encontradas nos materiais considerando os diferentes comprimentos de ligação encontrados entre os pares de espécie através da adição de uma função peso a eles. No entanto, a primeira limita-se os cálculos aos 6 primeiros vizinhos enquanto que a segunda não limita-se e apresenta um processo autoconsistente em sua aferição.

Com isso, foi possível demonstar que: (i) De maneira geral, as três metodologias permitem a detecção de distorções em perovskitas, contudo o valor da coordenação em cada abordagem difere consideravelmente entre elas. (ii) O CN é a metodologia mais limitada visto que possui um parâmetro em porcentagem p que afeta significativamente o valor final para uma mesma estrutura. Ao final de sua análise notou-se $p = 20\%$ como um bom parâmetro, em relação aos valores encontrados quando comparado aos demais métodos, para estudo de perovskitas. (iii) Os valores das distâncias médias B-X são quase equivalentes nas metodologias ECN e ECN_{av} respectivamente. Com leve maior valor para a segunda. Entretanto, resultam em coordenações distintas como resultado de diferentes formulações matemáticas. (iv) As abordagens, ECN e ECN_{av}, demonstraram atender ao objetivo central de detectar distorções. Apesar desta última formulação ser mais sofisticada, e não limitar o número de vizinhos considerados na determinação da coordenação de cada átomo.

Por fim, vale ressaltar, a técnica utilizada neste trabalho, nomeadamente, explorar bancos de dados disponíveis com uma finalidade específica, pode ser expandida para outros estudos. Por exemplo, explorar correlações entre diferentes propriedades (estruturais, óticas, eletrônicas, etc.) e até mesmo filtrar materiais com propriedades específicas. Logo, o código aqui desenvolvido pode servir como ponto de partida para estudos futuros. Além disso, as técnicas de aprendizado de máquina e análise de grandes volumes de dados (big data) apresentam expressiva tendência de crescimento nos próximos anos, o que demonstra o grande potencial desta abordagem empregada.

Referências

ARMIENTO, R.; KOZINSKY, B.; HAUTIER, G.; FORNARI, M.; CEDER, G. High-throughput screening of perovskite alloys for piezoelectric performance and thermodynamic stability. *Physical Review B*, v. 89, 03 2014.

ASE, E. *Ambiente de Simulação Atômica (ASE)*. 2024. Acesso em: 26 de agosto de 2024. Disponível em: <https://wiki.fysik.dtu.dk/ase/>.

CARDOSO, R. *Jupyter Notebook: saiba o que é e como utilizar a ferramenta*. 2023. Https://www.locaweb.com.br/blog/. Disponível em: <https://www.locaweb.com.br/blog/temas/codigo-aberto/jupyter-notebook-o-que-e/>. Acesso em: 23 de abril de 2024.

CASTELLI, I. E.; GARCíA-LASTRA, J. M.; THYGESEN, K. S.; JACOBSEN, K. W. Bandgap calculations and trends of organometal halide perovskites. *APL Materials*, v. 2, n. 8, p. 081514, 08 2014. ISSN 2166-532X. Disponível em: <https://doi.org/10.1063/1.4893495>.

CHEMISTRYTALK. *Coordination Number in Chemistry*. s.d. Https://chemistrytalk.org/. Disponível em: <https://chemistrytalk.org/coordination-number-in-chemistry/>. Acesso em: 23 de abril de 2024.

DANELON, J. G.; SANTOS, R. M.; DIAS, A. C.; SILVA, J. L. D.; LIMA, M. P. Contrasting the stability, octahedral distortions, and optoelectronic properties of 3d mabx 3 and 2d (ba) 2 (ma) b 2 x 7 (b= ge, sn, pb; x= cl, br, i) perovskites. *Physical Chemistry Chemical Physics*, Royal Society of Chemistry, v. 26, n. 10, p. 8469–8487, 2024.

DATABRICKS. *Jupyter Notebook*. 2024. Acesso em: 03 de agosto de 2024. Disponível em: <https://www.databricks.com/br/glossary/jupyter-notebook>.

DIAS, A. C.; LIMA, M. P.; SILVA, J. L. D. Role of structural phases and octahedra distortions in the optoelectronic and excitonic properties of csgex3 (x= cl, br, i) perovskites. *The Journal of Physical Chemistry C*, ACS Publications, v. 125, n. 35, p. 19142–19155, 2021.

Empresa de Pesquisa Energética (EPE). *Balanço Energético Nacional 2022: ano base 2021*. Rio de Janeiro, 2022. Acesso em: 20 jul. 2024. Disponível em: <https://www.epe.gov.br/pt/publicacoes-dados-abertos/publicacoes/balanco-energetico-nacional-2022>.

GIORGI, G.; YAMASHITA, K. Organic–inorganic halide perovskites: an ambipolar class of materials with enhanced photovoltaic performances. *Journal of Materials Chemistry A*, Royal Society of Chemistry, v. 3, n. 17, p. 8981–8991, 2015.

GitHub, Inc. *GitHub*. 2024. Acesso em: 15 agosto de 2024. Disponível em: <https://github.com/>.

GOODENOUGH, J. B. Electronic and ionic transport properties and other physical aspects of perovskites. *Reports on Progress in Physics*, IOP Publishing, v. 67, n. 11, p. 1915, 2004.

HELMENSTINE, A. M. *Coordination Number Definition in Chemistry*. 2019. Https://www.thoughtco.com/. Disponível em: <https://www.thoughtco.com/definition-of-coordination-number-604956>. Acesso em: 23 de abril de 2024.

JONES, F. *A corrida pelas células solares de perovskita*. 2024. Https://revistapesquisa.fapesp.br/. Disponível em: <https://revistapesquisa.fapesp.br/a-corrida-pelas-celulas-solares-de-perovskita/#:~:text=Uma%20tecnologia%20mais%20recente%2C%20que,Kaust)%2C%20na%20Ar%C3%A1bia%20Saudita.>. Acesso em: 23 de abril de 2024.

Jupyter. *Project Jupyter*. 2024. Acesso em: 18 agosto de 2024. Disponível em: <https://jupyter.org/>.

KITTEL, C. Física do estado sólido, 8. *Ed. Rio de Janeiro*, 2006.

LANDSKRON, K. *Coordination Numbers and Structures*. s.d. Https://chem.libretexts.org/. Disponível em: <https://chem.libretexts.org/Bookshelves/Inorganic_Chemistry/Inorganic_Coordination_Chemistry_(Landskron)/>. Acesso em: 23 de abril de 2024.

LI, C.; SOH, K. C. K.; WU, P. Formability of abo3 perovskites. *Journal of alloys and compounds*, Elsevier, v. 372, n. 1-2, p. 40–48, 2004.

LIMA, P. P. d. Revisão sistemática de perovskitas inorgânicas aplicadas em fotocatalisadores. 2021.

LONGi. *LONGi estabelece novo recorde mundial de eficiência de célula solar tándem de silício-perovskita na SNEC 2024*. 2024. Acesso em: 22 ago. 2024. Disponível em: <https://www.longi.com/br/news/2024-snec-silicon-perovskite-tandem-solar-cells-novo-recorde/>.

LONGO, C.; PAOLI, M.-A. D. Dye-sensitized solar cells: a successful combination of materials. *Journal of the Brazilian Chemical Society*, SciELO Brasil, v. 14, p. 898–901, 2003.

MAYRINCK, C. d.; FONSECA, A. F. V. d.; SCHIAVON, M. A. Nanocristais de perovskitas coloidais: Histórico, propriedades e aplicações. *Química Nova*, SciELO Brasil, v. 43, p. 1264–1276, 2020.

MOMMA, K.; IZUMI, F. Vesta: a three-dimensional visualization system for electronic and structural analysis. *Journal of Applied crystallography*, International Union of Crystallography, v. 41, n. 3, p. 653–658, 2008.

MOMMA, K.; IZUMI, F. VESTA3 for three-dimensional visualization of crystal, volumetric and morphology data. *Journal of Applied Crystallography*, v. 44, n. 6, p. 1272–1276, Dec 2011. Disponível em: <https://doi.org/10.1107/S0021889811038970>.

MULLER, P. Glossary of terms used in physical organic chemistry (iupac recommendations 1994). *Pure and Applied Chemistry*, v. 66, n. 5, p. 1077–1184, 1994. Disponível em: <https://doi.org/10.1351/pac199466051077>.

PEÑA, M. A.; FIERRO, J. L. Chemical structures and performance of perovskite oxides. *Chemical reviews*, ACS Publications, v. 101, n. 7, p. 1981–2018, 2001.

PEREZ, O. A. G. *Estudos estruturais a baixas temperaturas em compostos com estrutura perovskita*. Tese (Doutorado) — [sn], 2000.

PETROVIć, M.; CHELLAPPAN, V.; RAMAKRISHNA, S. Perovskites: Solar cells & engineering applications – materials and device developments. *Solar Energy*, v. 122, p. 678–699, 2015. ISSN 0038-092X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0038092X15005277>.

QUEVEDO, Í. M. d. Síntese e caracterização da perovskita srtio3. 2022.

RAPHAEL, E.; SILVA, M. N.; SZOSTAK, R.; SCHIAVON, M. A.; NOGUEIRA, A. F. Celulas solares de perovskitas: Uma nova tecnologia emergente. *Química Nova*, Sociedade Brasileira de Química, v. 41, n. 1, p. 61–74, Jan 2018. ISSN 0100-4042. Disponível em: <https://doi.org/10.21577/0100-4042.20170127>.

SILVA, I. B. da. *Módulos Fotovoltaicos em Perovskita: Tecnologia Emergente P&D GT 660 – Interconexão de Minimódulos de Perovskita*. 2012. Https://inova.cemig.com.br/. Disponível em: <https://inova.cemig.com.br/publicacoes/modulos-fotovoltaicos-em-perovskita-tecnologia-emergente-pd-gt-660-interconexao-de-minimodulos-de-perovskita/>. Acesso em: 23 de abril de 2024.

SILVA, J. L. D. Effective coordination concept applied for phase change (gete) m (sb2te3) n compounds. *Journal of Applied Physics*, AIP Publishing, v. 109, n. 2, 2011.

YI, Z.; LADI, N. H.; SHAI, X.; LI, H.; SHEN, Y.; WANG, M. Will organic–inorganic hybrid halide lead perovskites be eliminated from optoelectronic applications? *Nanoscale Advances*, Royal Society of Chemistry, v. 1, n. 4, p. 1276–1289, 2019.

ZHAN, Y.; KHALID, M.; VIVO, P.; ARSHID, N. *Low-Dimensional Halide Perovskites: Structure, Synthesis, and Applications*. [S.l.]: Elsevier, 2022.

ZHOU, Y.; ZHAO, Y. Chemical stability and instability of inorganic halide perovskites. *Energy Environ. Sci.*, The Royal Society of Chemistry, v. 12, p. 1495–1511, 2019. Disponível em: <http://dx.doi.org/10.1039/C8EE03559H>.

Apêndices

APÊNDICE A – Perovskitas

Índice	Perovskita	Índice	Perovskita
1	$PbBr_2ClCs$	41	$PbBr_2ClCH_5N_2$
2	$PbBr_3Cs$	42	$PbBr_3CH_5N_2$
3	$PbBrCl_2Cs$	43	$PbBrCl_2CH_5N_2$
4	$PbCl_3Cs$	44	$PbCl_3CH_5N_2$
5	PbI_2BrCs	45	$PbI_2BrCH_5N_2$
6	PbI_2ClCs	46	$PbI_2ClCH_5N_2$
7	PbI_3Cs	47	$PbI_3CH_5N_2$
8	$PbIBr_2Cs$	48	$PbIBr_2CH_5N_2$
9	$PbIBrClCs$	49	$PbIBrClCH_5N_2$
10	$PbICl_2Cs$	50	$PbICl_2CH_5N_2$
11	$SnBr_2ClCs$	51	$SnBr_2ClCH_5N_2$
12	$SnBr_3Cs$	52	$SnBr_3CH_5N_2$
13	$SnBrCl_2Cs$	53	$SnBrCl_2CH_5N_2$
14	$SnCl_3Cs$	54	$SnCl_3CH_5N_2$
15	SnI_2BrCs	55	$SnI_2BrCH_5N_2$
16	SnI_2ClCs	56	$SnI_2ClCH_5N_2$
17	SnI_3Cs	57	$SnI_3CH_5N_2$
18	$SnIBr_2Cs$	58	$SnIBr_2CH_5N_2$
19	$SnIBrClCs$	59	$SnIBrClCH_5N_2$
20	$SnICl_2Cs$	60	$SnICl_2CH_5N_2$
21	$PbBr_2ClNH_3CH_3$	61	$Br_2Cl_4Br_6Pb_4Cs_4$
22	$PbBr_3NH_3CH_3$	62	$Br_{12}Pb_4Cs_4$
23	$PbBrCl_2NH_3CH_3$	63	$Cl_8Br_4Pb_4Cs_4$
24	$PbCl_3NH_3CH_3$	64	$Cl_{12}Pb_4Cs_4$
25	$PbI_2BrNH_3CH_3$	65	$I_2Br_4I_6Pb_4Cs_4$
26	$PbI_2ClNH_3CH_3$	66	$I_2Cl_4I_6Pb_4Cs_4$
27	$PbI_3NH_3CH_3$	67	$I_{12}Pb_4Cs_4$
28	$PbIBr_2NH_3CH_3$	68	$Br_8I_4Pb_4Cs_4$
29	$PbIBrClNH_3CH_3$	69	$Br_2Cl_4Br_2I_4Pb_4Cs_4$
30	$PbICl_2NH_3CH_3$	70	$Cl_8I_4Pb_4Cs_4$
31	$SnBr_2ClNH_3CH_3$	71	$Br_2Cl_4Br_6Sn_4Cs_4$
32	$SnBr_3NH_3CH_3$	72	$Br_{12}Sn_4Cs_4$
33	$SnBrCl_2NH_3CH_3$	73	$Cl_8Br_4Sn_4Cs_4$
34	$SnCl_3NH_3CH_3$	74	$Cl_{12}Sn_4Cs_4$
35	$SnI_2BrNH_3CH_3$	75	$I_2Br_4I_6Sn_4Cs_4$
36	$SnI_2ClNH_3CH_3$	76	$I_2Cl_4I_6Sn_4Cs_4$
37	$SnI_3NH_3CH_3$	77	$I_{12}Sn_4Cs_4$
38	$SnIBr_2NH_3CH_3$	78	$Br_8I_4Sn_4Cs_4$
39	$SnIBrClNH_3CH_3$	79	$Br_2Cl_4Br_2I_4Sn_4Cs_4$
40	$SnICl_2NH_3CH_3$	80	$Cl_8I_4Sn_4Cs_4$

Tabela A.1 – Perovskitas presentes no banco de dados. Parte 1.

Índice	**Perovskita**
81	$Br_2Cl_4Br_6Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
82	$Br_{12}Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
83	$Cl_8Br_4Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
84	$Cl_{12}Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
85	$I_2Br_4I_6Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
86	$I_2Cl_4I_6Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
87	$I_{12}Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
88	$Br_8I_4Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
89	$Br_2Cl_4Br_2I_4Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
90	$Cl_8I_4Pb_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
91	$Br_2Cl_4Br_6Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
92	$Br_{12}Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
93	$Cl_8Br_4Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
94	$Cl_{12}Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
95	$I_2Br_4I_6Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
96	$I_2Cl_4I_6Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
97	$I_{12}Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
98	$Br_8I_4Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
99	$Br_2Cl_4Br_2I_4Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
100	$Cl_8I_4Sn_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3$
101	$Br_2Cl_4Br_6Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
102	$Br_{12}Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
103	$Cl_8Br_4Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
104	$Cl_{12}Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
105	$I_2Br_4I_6Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
106	$I_2Cl_4I_6Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
107	$I_{12}Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
108	$Br_8I_4Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
109	$Br_2Cl_4Br_2I_4Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
110	$Cl_8I_4Pb_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
111	$Br_2Cl_4Br_6Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
112	$Br_{12}Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
113	$Cl_8Br_4Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
114	$Cl_{12}Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
115	$I_2Br_4I_6Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
116	$I_2Cl_4I_6Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
117	$I_{12}Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
118	$Br_8I_4Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
119	$Br_2Cl_4Br_2I_4Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
120	$Cl_8I_4Sn_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2$
121	$Br_8Cl_4Cs_4Pb_4$
122	$Br_{12}Cs_4Pb_4$
123	$Br_4Cl_8Cs_4Pb_4$
124	$Cl_{12}Cs_4Pb_4$
125	$I_8Br_4Cs_4Pb_4$

Tabela A.2 – Perovskitas presentes no banco de dados. Parte 2.

Índice	**Perovskita**
126	$I_8Cl_4Cs_4Pb_4$
127	$I_{12}Cs_4Pb_4$
128	$I_4Br_8Cs_4Pb_4$
129	$I_4Br_4Cl_4Cs_4Pb_4$
130	$I_4Cl_8Cs_4Pb_4$
131	$Br_8Cl_4Cs_4Sn_4$
132	$Br_{12}Cs_4Sn_4$
133	$Br_4Cl_8Cs_4Sn_4$
134	$Cl_{12}Cs_4Sn_4$
135	$I_8Br_4Cs_4Sn_4$
136	$I_8Cl_4Cs_4Sn_4$
137	$I_{12}Cs_4Sn_4$
138	$I_4Br_8Cs_4Sn_4$
139	$I_4Br_4Cl_4Cs_4Sn_4$
140	$I_4Cl_8Cs_4Sn_4$
141	$Br_8Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
142	$Br_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
143	$Br_4Cl_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
144	$Cl_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
145	$I_8Br_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
146	$I_8Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
147	$I_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
148	$I_4Br_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
149	$I_4Br_4Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
150	$I_4Cl_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Pb_4$
151	$Br_8Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
152	$Br_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
153	$Br_4Cl_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
154	$Cl_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
155	$I_8Br_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
156	$I_8Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
157	$I_{12}NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
158	$I_4Br_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
159	$I_4Br_4Cl_4NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
160	$I_4Cl_8NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Sn_4$
161	$Br_8Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
162	$Br_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
163	$Br_4Cl_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
164	$Cl_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
165	$I_8Br_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
166	$I_8Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
167	$I_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
168	$I_4Br_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
169	$I_4Br_4Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$
170	$I_4Cl_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Pb_4$

Tabela A.3 – Perovskitas presentes no banco de dados. Parte 3.

Índice	**Perovskita**
171	$Br_8Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
172	$Br_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
173	$Br_4Cl_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
174	$Cl_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
175	$I_8Br_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
176	$I_8Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
177	$I_{12}CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
178	$I_4Br_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
179	$I_4Br_4Cl_4CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
180	$I_4Cl_8CH_5N_2CH_5N_2CH_5N_2CH_5N_2Sn_4$
181	$Cs_4Br_8Cl_4Pb_4$
182	$Cs_4Br_{12}Pb_4$
183	$Cs_4Br_4Cl_8Pb_4$
184	$Cs_4Cl_{12}Pb_4$
185	$Cs_4I_8Br_4Pb_4$
186	$Cs_4I_8Cl_4Pb_4$
187	$Cs_4I_{12}Pb_4$
188	$Cs_4I_4Br_8Pb_4$
189	$Cs_4I_4Br_4Cl_4Pb_4$
190	$Cs_4I_4Cl_8Pb_4$
191	$Cs_4Br_8Cl_4Sn_4$
192	$Cs_4Br_{12}Sn_4$
193	$Cs_4Br_4Cl_8Sn_4$
194	$Cs_4Cl_{12}Sn_4$
195	$Cs_4I_8Br_4Sn_4$
196	$Cs_4I_8Cl_4Sn_4$
197	$Cs_4I_{12}Sn_4$
198	$Cs_4I_4Br_8Sn_4$
199	$Cs_4I_4Br_4Cl_4Sn_4$
200	$Cs_4I_4Cl_8Sn_4$
201	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_8Cl_4Pb_4$
202	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_{12}Pb_4$
203	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_4Cl_8Pb_4$
204	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Cl_{12}Pb_4$
205	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_8Br_4Pb_4$
206	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_8Cl_4Pb_4$
207	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_{12}Pb_4$
208	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Br_8Pb_4$
209	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Br_4Cl_4Pb_4$
210	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Cl_8Pb_4$
211	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_8Cl_4Sn_4$
212	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_{12}Sn_4$
213	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Br_4Cl_8Sn_4$
214	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3Cl_{12}Sn_4$

Tabela A.4 – Perovskitas presentes no banco de dados. Parte 4.

Índice	**Perovskita**
215	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_8Br_4Sn_4$
216	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_8Cl_4Sn_4$
217	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_{12}Sn_4$
218	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Br_8Sn_4$
219	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Br_4Cl_4Sn_4$
220	$NH_3CH_3NH_3CH_3NH_3CH_3NH_3CH_3I_4Cl_8Sn_4$
221	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_8Cl_4Pb_4$
222	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_{12}Pb_4$
223	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_4Cl_8Pb_4$
224	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Cl_{12}Pb_4$
225	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_8Br_4Pb_4$
226	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_8Cl_4Pb_4$
227	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_{12}Pb_4$
228	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Br_8Pb_4$
229	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Br_4Cl_4Pb_4$
230	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Cl_8Pb_4$
231	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_8Cl_4Sn_4$
232	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_{12}Sn_4$
233	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Br_4Cl_8Sn_4$
234	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2Cl_{12}Sn_4$
235	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_8Br_4Sn_4$
236	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_8Cl_4Sn_4$
237	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_{12}Sn_4$
238	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Br_8Sn_4$
239	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Br_4Cl_4Sn_4$
240	$CH_5N_2CH_5N_2CH_5N_2CH_5N_2I_4Cl_8Sn_4$

Tabela A.5 – Perovskitas presentes no banco de dados. Parte 5. Fonte: (CASTELLI et al., 2014).

APÊNDICE B – Códigos

B.0.1 Bibliotecas

```
from ase.db import connect
import ase.io
from scipy.spatial import distance_matrix
import numpy as np
import ase.geometry
```

B.0.2 Função Distância Mínima

```
def minima_distancia(symbols, aux_b, atom):
    # Definição de listas:
    aux_xCl = [] # Índices do elemento Cl presente na PVK
    aux_xBr = [] # Índices do elemento Br presente na PVK
    aux_xI = [] # Índices do elemento I presentes na PVK
    aux_x = [] # Junção das listas anteriores com os índices
    distancias_Cl = [] # Distâncias calculadas entre (X,Cl)
    distancias_Br = [] # Distâncias calculadas entre (X,Br)
    distancias_I = [] # Distâncias calculadas entre (X,I)
    dist = [] # Mínima distância entre (X,X') com X' em Cl,Br,I
    posi_b = [] # Posições do elemento do sítio B
    posi_x = [] # Posições do elemento do sítio X

    # For para quebrar em listas os índices do Cl,Br,I
    for indice, elemento in enumerate(symbols):
        if elemento == 'Cl':
            aux_xCl.append(indice)
        if elemento == 'Br':
            aux_xBr.append(indice)
        if elemento == 'I':
            aux_xI.append(indice)

    # Junção das listas em uma só
    aux_x.append(aux_xCl)
    aux_x.append(aux_xBr)
    aux_x.append(aux_xI)

    # For para percorrer as 3 listas com as distâncias para cada
   elemento do sítio X
    for x in range(0, 3):
        aux_distancias = []

```

```
32          # If para filtrar quais elementos a PVK em análise tem no sítio
    X
33          if not aux_x[x]:
34              continue
35          else:
36              # For para pegar as posições separadamente dos átomos Cl,Br,
    I
37              for p in aux_x[x]:
38                  positions_x = atom.positions[p]
39                  posi_x.append(positions_x)
40
41                  # For para pegar posição de cada átomo do elemento do sí
    tio B
42                  for i in aux_b:
43                      positions_b = atom.positions[i]
44                      posi_b.append(positions_b)
45
46                      # For para percorrer os inteiros utilizados na
    determinação das células vizinhas e das respectivas distâncias
47                      for k in valor:
48                          # If == 0 colocado pois quando o valor do
    inteiro é 0 obtemos 3 matrizes iguais, dessa forma temos apenas 1
49                          if k == 0:
50                              vizinho = positions_x
51                              distancia = ase.geometry.get_distances(
    positions_b, vizinho)
52                              aux_distancias.append(distancia[1])
53                          else:
54                              # Aqui é pego os vizinhos e as respectivas
    distâncias considerando os inteiros 1 e -1
55                              for n in rede:
56                                  aux = k * cell[n, :]
57                                  vizinho = positions_x + aux
58                                  distancia = ase.geometry.get_distances(
    positions_b, vizinho)
59                                  aux_distancias.append(distancia[1])
60
61              # Armazenando as distâncias para cada elemento em listas
    separadas
62              if x == 0:
63                  distancias_Cl.append(np.array(aux_distancias))
64              elif x == 1:
65                  distancias_Br.append(np.array(aux_distancias))
66              else:
67                  distancias_I.append(np.array(aux_distancias))
68
69      # Determinação da mínima distância de cada elemento
```

```
    if distancias_I:
        I = min(distancias_I[0])
        dist.append(I)
    if distancias_Br:
        Br = min(distancias_Br[0])
        dist.append(Br)
    if distancias_Cl:
        Cl = min(distancias_Cl[0])
        dist.append(Cl)

    # Retorna lista com as distâncias mínimas na sequência I,Br,Cl
    return dist
```

B.0.3 CN

```
# Conexão com o banco
db = connect("organometal.db")

# Definição de listas
atoms = []# PVK's do banco
perovskita = []# Índice das PVK's
valor = [0,-1,1]# Inteiros para determinação dos vizinhos
rede = [0,1,2]# Vetores de rede
elex = []# Armazena qual o elemento do sítio X

# for para extrair as PVK's do banco
for row in db.select():
    atoms.append(row.toatoms())

# Seleção de PVK's do banco
indices = [2,8,53,81,108,174,176,179,225,238]
atoms = [atoms[i] for i in indices]

# for para percorrer as PVK's selecionadas
for i, atom in enumerate(atoms):

    # Fórmula da PVK
    formula = atom.symbols
    # Quebra da fórmula em átomos
    symbols = atom.get_chemical_symbols()
    perovskita.append(i+1)
    # Traz os vetores de rede
    cell = atom.get_cell().todict()['array']

    # Definição de listas para armazenar os índices e elementos de cada
    átomo dos sítios
    aux_b = []# Índice sítio B
```

```
    aux_bele = []# Elemento sítio B
    aux_xs = []# Índice sítio X
    aux_xele = []# Elemento sítio X
    aux_a = []# Índice sítio A
    aux_aele = []# Elemento sítio A

    # Classificação dos átomos por sítio, salvando seus índices em
listas
    for indice, elemento in enumerate(symbols):
        if elemento in ['Au', 'Pb', 'Sn']:
            aux_b.append(indice)
            aux_bele.append(elemento)
        elif elemento in ['Cl', 'Br', 'I']:
            aux_xs.append(indice)
            aux_xele.append(elemento)
        else:
            aux_a.append(indice)
            aux_aele.append(elemento)

    # Definição de listas auxiliares para a obtenção das posições
    posi_b = []# Sítio B
    posi_x = []# Sítio X

    # Chamando a função para pegar a mínima distância
    dist = minima_distancia(symbols, aux_b, atom)

    # Definição de listas auxiliares para confecção de gráficos
    auxCN_I = []
    auxCN_Br = []
    auxCN_Cl = []

    # if para determinar qual o elemento do sítio X
    if 'Pb' in aux_bele:
        elex.append('Pb')
    else:
        elex.append('Sn')

    # for para pegar o positions(vetor posição) do átomo do sítio B
    for b in aux_b:
        # Definição de listas auxiliares para obtenção dos vizinhos e
das distâncias com o sítio B
        aux_vizinhos_I = []
        aux_distancias_I = []
        aux_vizinhos_Br = []
        aux_distancias_Br = []
        aux_vizinhos_Cl = []
        aux_distancias_Cl = []
```

```
77          distancias = []
78          positions_b = atom.positions[b]
79          posi_b.append(positions_b)
80
81          # for para obter o índice do átomo do sítio x e sua respectiva
    posição
82          for i, elemento in enumerate(aux_xele):
83              if elemento == 'I':
84                  ind_atomo = aux_xs[i]
85                  positions_x = atom.positions[ind_atomo]
86
87                  # for para percorrer os inteiros utilizados na determina
    ção das células vizinhas
88                  for k in valor:
89                      # if == 0 colocado pois quando o valor do inteiro é
    0 obtemos 3 matrizes iguais, dessa forma temos apenas 1
90                      # Determinação dos vizinhos e das respectivas distâ
    ncias em relação ao sítio B
91                      if k == 0:
92                          vizinho = positions_x
93                          distancia = ase.geometry.get_distances(
    positions_b, vizinho)
94                          aux_vizinhos_I.append(vizinho)
95                          aux_distancias_I.append(distancia[1])
96                          pass
97                      else:
98                          # Determinação dos vizinhos e das respectivas
    distâncias em relação ao sítio B considerando os inteiros 1 e -1
99                          for n in rede:
100                             aux = k*cell[n,:]
101                             vizinho = positions_x + aux
102                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
103                             aux_vizinhos_I.append(vizinho)
104                             aux_distancias_I.append(distancia[1])
105
106             elif elemento == 'Br':
107                 ind_atomo = aux_xs[i]
108                 positions_x = atom.positions[ind_atomo]
109
110                 for k in valor:
111                     if k == 0:
112                         vizinho = positions_x
113                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
114                         aux_vizinhos_Br.append(vizinho)
115                         aux_distancias_Br.append(distancia[1])
```

```
116                         pass
117                     else:
118                         for n in rede:
119                             aux = k*cell[n,:]
120                             vizinho = positions_x + aux
121                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
122                             aux_vizinhos_Br.append(vizinho)
123                             aux_distancias_Br.append(distancia[1])
124 
125             else:
126                 ind_atomo = aux_xs[i]
127                 positions_x = atom.positions[ind_atomo]
128 
129                 for k in valor:
130                     if k == 0:
131                         vizinho = positions_x
132                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
133                         aux_vizinhos_Cl.append(vizinho)
134                         aux_distancias_Cl.append(distancia[1])
135                         pass
136                     else:
137                         for n in rede:
138                             aux = k*cell[n,:]
139                             vizinho = positions_x + aux
140                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
141                             aux_vizinhos_Cl.append(vizinho)
142                             aux_distancias_Cl.append(distancia[1])
143 
144         # Armazenando as distâncias em lista ordenando por elemento do s
    ítio X
145         if aux_distancias_I:
146             distancias.append(np.array(aux_distancias_I))
147         if aux_distancias_Br:
148             distancias.append(np.array(aux_distancias_Br))
149         if aux_distancias_Cl:
150             distancias.append(np.array(aux_distancias_Cl))
151 
152 
153 
154         # Determinação do comprimento da lista
155         compri_distancias = len(distancias)
156 
157         # Realizando o rateio do Cn por sítio X
158         CN_I = []# Valor Cn (X,I)
```

```
159         CN_Br = []# Valor Cn (X,Br)
160         CN_Cl = []# Valor Cn (X,Cl)
161 
162         # for para percorrer todas as distâncias de acordo com o
    comprimento obtido anteriormente
163         for e in range(0, compri_distancias):
164 
165             # Distância mínima para cada elemento do sítio X
166             dist_min = dist[e]
167 
168             # Cálculo o raio de corte
169             rcut = dist_min*1.05
170 
171             # Cálculo do Cn, onde é soma o número de distâncias onde a
    distância é <= ao rcut
172             cn = sum(i <= rcut for i in distancias[e])
173 
174             if len(aux_b) == 1:
175                 if compri_distancias == 1:
176                     if 'I' in aux_xele:
177                         CN_I.append(cn[0][0])
178                         CN_Br.append(None)
179                         CN_Cl.append(None)
180                     elif 'Br' in aux_xele:
181                         CN_I.append(None)
182                         CN_Br.append(cn[0][0])
183                         CN_Cl.append(None)
184                     else:
185                         CN_I.append(None)
186                         CN_Br.append(None)
187                         CN_Cl.append(cn[0][0])
188                 if compri_distancias == 2:
189                     if 'I' in aux_xele and 'Br' in aux_xele:
190                         if e == 0:
191                             CN_I.append(cn[0][0])
192                         if e == 1:#(else)
193                             CN_Br.append(cn[0][0])
194                             CN_Cl.append(None)
195                     elif 'I' in aux_xele and 'Cl' in aux_xele:
196                         if e == 0:
197                             CN_I.append(cn[0][0])
198                             CN_Br.append(None)
199                         if e == 1:
200                             CN_Cl.append(cn[0][0])
201                     else:#(Br,Cl)
202                         if e == 0:
203                             CN_I.append(None)
```

```
                            CN_Br.append(cn[0][0])
                        if e == 1:
                            CN_Cl.append(cn[0][0])
                if compri_distancias == 3:
                    if e == 0:
                        CN_I.append(cn[0][0])
                    elif e == 1:
                        CN_Br.append(cn[0][0])
                    else:
                        CN_Cl.append(cn[0][0])
            else:
            #else len(aux_b) > 1:
                if compri_distancias == 1:
                    if 'I' in aux_xele:
                        auxCN_I.append(cn[0][0])
                    elif 'Br' in aux_xele:
                        auxCN_Br.append(cn[0][0])
                    else:
                        auxCN_Cl.append(cn[0][0])
                if compri_distancias == 2:
                    if 'I' in aux_xele and 'Br' in aux_xele:
                        if e == 0:
                            auxCN_I.append(cn[0][0])
                        if e == 1:#(else)
                            auxCN_Br.append(cn[0][0])
                    elif 'I' in aux_xele and 'Cl' in aux_xele:
                        if e == 0:
                            auxCN_I.append(cn[0][0])
                        if e == 1:
                            auxCN_Cl.append(cn[0][0])
                    else:#(Br,Cl)
                        if e == 0:
                            auxCN_Br.append(cn[0][0])
                        if e == 1:
                            auxCN_Cl.append(cn[0][0])
                if compri_distancias == 3:
                    if e == 0:
                        auxCN_I.append(cn[0][0])
                    elif e == 1:
                        auxCN_Br.append(cn[0][0])
                    else:
                        auxCN_Cl.append(cn[0][0])

                if b == aux_b[-1] and e == compri_distancias - 1:
                    denominador = len(aux_b)
                    if auxCN_I:
                        ni = sum(auxCN_I)/denominador
```

```
                        CN_I.append(ni)
                    else:
                        CN_I.append(None)

                    if auxCN_Br:
                        nbr = sum(auxCN_Br)/denominador
                        CN_Br.append(nbr)
                    else:
                        CN_Br.append(None)

                    if auxCN_Cl:
                        ncl = sum(auxCN_Cl)/denominador
                        CN_Cl.append(ncl)
                    else:
                        CN_Cl.append(None)




    print("Fórmula:", formula)
    print("Symbols:", symbols)
    print("Índice_b:", aux_b)
    print("Índice_x:", aux_xs)
    print("Índice_a:", aux_a)
    print("Elementos_x:", aux_xele)
    print("CN_I:", CN_I)
    print("CN_Br:", CN_Br)
    print("CN_Cl:", CN_Cl)
    print("Índice_perovskita", perovskita)
    print("-------------------------------------")
```

B.0.4 ECN

```
def media_distancia(symbols, aux_b, minima, atom):
    # Definição de listas
    aux_xCl = []# Índices do elemento Cl presentes na PVK
    aux_xBr = []# Índices do elemento Br presentes na PVK
    aux_xI = []# Índices do elemento I presentes na PVK
    aux_x = []# Junção das listas anteriores com os índices
    distancias_Cl = []# Distâncias calculadas entre (X,Cl)
    distancias_Br = []# Distâncias calculadas entre (X,Br)
    distancias_I = []# Distâncias calculadas entre (X,I)
    dist = []# Mínima distância entre (X,X') com X' em Cl,Br,I
    posi_b = []# Posições do elemento do sítio B
    posi_x = []# Posições do elemento do sítio X
    dist_med = []# Distância média
```

```
    numerador_I = []# Numerador auxiliar para cálculo da distância para sítio I
    denominador_I = []# Denominador auxiliar para cálculo da distância para sítio I
    numerador_Br = []# Numerador auxiliar para cálculo da distância para sítio Br
    denominador_Br = []# Denominador auxiliar para cálculo da distância para sítio Br
    numerador_Cl = []# Numerador auxiliar para cálculo da distância para sítio Cl
    denominador_Cl = []# Denominador auxiliar para cálculo da distância para sítio Cl

    # for para quebrar em listas os índices do Cl,Br,I
    for indice, elemento in enumerate(symbols):
        if elemento == 'Cl':
            aux_xCl.append(indice)
        if elemento == 'Br':
            aux_xBr.append(indice)
        if elemento == 'I':
            aux_xI.append(indice)

    # Junção das listas em uma só
    aux_x.append(aux_xI)
    aux_x.append(aux_xBr)
    aux_x.append(aux_xCl)

    # for para percorrer as 3 listas com as distâncias, numerador e denominador para cada elemento do sítio X
    indc = -1
    for x in range(0, 3):
        aux_distancias = []
        aux_numerador = []
        aux_denominador = []

        # if para filtrar quais elementos a PVK em análise tem no sítio X e a respectiva distância mínima
        if not aux_x[x]:
            continue
        else:
            indc = indc + 1
            dist_min = minima[indc][0][0]

            # for para pegar as posições separadamente dos átomos Cl,Br,I
            for p in aux_x[x]:
                positions_x = atom.positions[p]
```

```
52                posi_x.append(positions_x)
53
54                # for para pegar posição de cada átomo do elemento do sí
    tio B
55                for i in aux_b:
56                    positions_b = atom.positions[i]
57                    posi_b.append(positions_b)
58
59                    # for para percorrer os inteiros utilizados na
    determinação das células vizinhas, das respectivas distâncias e o
    numerador e denominador para cálculo da distância média
60                    for k in valor:
61                        # if == 0 colocado pois quando o valor do
    inteiro é 0 obtemos 3 matrizes iguais, dessa forma temos apenas 1
62                        if k == 0:
63                            vizinho = positions_x
64                            distancia = ase.geometry.get_distances(
    positions_b, vizinho)
65                            aux_distancias.append(distancia[1][0][0])
66                            numerador = distancia[1][0][0] * np.exp(1 -
    (distancia[1][0][0] / dist_min) ** 6)
67                            denominador = np.exp(1 - (distancia[1][0][0]
     / dist_min) ** 6)
68                            aux_numerador.append(numerador)
69                            aux_denominador.append(denominador)
70                        else:
71                            # Aqui é pego os vizinhos, as respectivas
    distâncias e o numerador e denominador para cálculo da distância mé
    dia, considerando os inteiros 1 e -1
72                            for n in rede:
73                                aux = k * cell[n, :]
74                                vizinho = positions_x + aux
75                                distancia = ase.geometry.get_distances(
    positions_b, vizinho)
76                                aux_distancias.append(distancia
    [1][0][0])
77                                numerador = distancia[1][0][0] * np.exp
    (1 - (distancia[1][0][0] / dist_min) ** 6)
78                                denominador = np.exp(1 - (distancia
    [1][0][0] / dist_min) ** 6)
79                                aux_numerador.append(numerador)
80                                aux_denominador.append(denominador)
81
82            # Determinação dos 6 primeiros vizinhos para cada elemento
    do sítio X
83            if x == 0:
84                seis_menores_I = sorted(enumerate(aux_distancias), key=
```

```
    lambda x: x[1])[:6]
85                  for posicao, numero in seis_menores_I:
86                      distancias_I.append(np.array(aux_distancias[posicao
    ]))
87                      numerador_I.append(np.array(aux_numerador[posicao]))
88                      denominador_I.append(np.array(aux_denominador[
    posicao]))
89              elif x == 1:
90                  seis_menores_Br = sorted(enumerate(aux_distancias), key=
    lambda x: x[1])[:6]
91                  for posicao, numero in seis_menores_Br:
92                      distancias_Br.append(np.array(aux_distancias[posicao
    ]))
93                      numerador_Br.append(np.array(aux_numerador[posicao])
    )
94                      denominador_Br.append(np.array(aux_denominador[
    posicao]))
95              else:
96                  seis_menores_Cl = sorted(enumerate(aux_distancias), key=
    lambda x: x[1])[:6]
97                  for posicao, numero in seis_menores_Cl:
98                      distancias_Cl.append(np.array(aux_distancias[posicao
    ]))
99                      numerador_Cl.append(np.array(aux_numerador[posicao])
    )
100                     denominador_Cl.append(np.array(aux_denominador[
    posicao]))
101
102     # Determinação das distâncias médias
103     if distancias_I:
104         num = np.sum(numerador_I)
105         den = np.sum(denominador_I)
106         d_med = num / den
107         dist_med.append(d_med)
108     if not distancias_I:
109         dist_med.append(None)
110     if distancias_Br:
111         num = np.sum(numerador_Br)
112         den = np.sum(denominador_Br)
113         d_med = num / den
114         dist_med.append(d_med)
115     if not distancias_Br:
116         dist_med.append(None)
117     if distancias_Cl:
118         num = np.sum(numerador_Cl)
119         den = np.sum(denominador_Cl)
120         d_med = num / den
```

```
            dist_med.append(d_med)
        if not distancias_Cl:
            dist_med.append(None)

        # Retorna lista com as distâncias médias na sequência I,Br,Cl
        return dist_med
```

```
# Conexão com o banco
db = connect("organometal.db")

# Definição de listas
atoms = []# PVK's do banco
perovskita = []# Índice das PVK's
valor = [0,-1,1]# Inteiros para determinação dos vizinhos
rede = [0,1,2]# Vetores de rede
elex = []# Armazena qual o elemento do sítio X

# for para extrair as PVK's do banco
for row in db.select():
    atoms.append(row.toatoms())

# Seleção de PVK's do banco
indices = [2,8,53,81,108,174,176,179,225,238]
atoms = [atoms[i] for i in indices]

# for para percorrer as PVK's selecionadas
for i, atom in enumerate(atoms):

    # Fórmula da PVK
    formula = atom.symbols
    # Quebra da fórmula em átomos
    symbols = atom.get_chemical_symbols()
    perovskita.append(i+1)
    # Traz os vetores de rede
    cell = atom.get_cell().todict()['array']

    # Definição de listas para armazenar os índices e elementos de cada
    átomo dos sítios
    aux_b = []# Índice sítio B
    aux_bele = []# Elemento sítio B
    aux_xs = []# Índice sítio X
    aux_xele = []# Elemento sítio X
    aux_a = []# Índice sítio A
    aux_aele = []# Elemento sítio A

    # Classificação dos átomos por sítio, salvando seus índices em
    listas
    for indice, elemento in enumerate(symbols):
```

```
40          if elemento in ['Au', 'Pb', 'Sn']:
41              aux_b.append(indice)
42              aux_bele.append(elemento)
43          elif elemento in ['Cl', 'Br', 'I']:
44              aux_xs.append(indice)
45              aux_xele.append(elemento)
46          else:
47              aux_a.append(indice)
48              aux_aele.append(elemento)
49
50      # Definição de listas auxiliares para a obtenção das posições
51      posi_b = []# Sítio B
52      posi_x = []# Sítio X
53
54      # Chamando a função para pegar a mínima distância
55      minima = minima_distancia(symbols, aux_b, atom)
56
57      # Chamando a função para pegar a distância média
58      distmed = media_distancia(symbols, aux_b, minima, atom)
59
60      # Definição de listas auxiliares para confecção de gráficos
61      auxECN_I = []
62      auxECN_Br = []
63      auxECN_Cl = []
64
65      # if para determinar qual o elemento do sítio X
66      if 'Pb' in aux_bele:
67          elex.append('Pb')
68      else:
69          elex.append('Sn')
70
71      # for para pegar o positions(vetor posição) do átomo do sítio B
72      for b in aux_b:
73          # Definição de listas auxiliares para obtenção dos vizinhos e
    das distâncias com o sítio B e cálculo do ECN
74          aux_vizinhos_I = []
75          aux_distancias_I = []
76          aux_vizinhos_Br = []
77          aux_distancias_Br = []
78          aux_vizinhos_Cl = []
79          aux_distancias_Cl = []
80          distancias = []
81          aux_vesta_I = []
82          aux_vesta_Br = []
83          aux_vesta_Cl = []
84          ecn_vest = []
85          positions_b = atom.positions[b]
```

```
86          posi_b.append(positions_b)
87
88          # for para obter o índice do átomo do sítio x e sua respectiva
    posição
89          for i, elemento in enumerate(aux_xele):
90              if elemento == 'I':
91                  ind_atomo = aux_xs[i]
92                  positions_x = atom.positions[ind_atomo]
93
94                  # for para percorrer os inteiros utilizados na determina
    ção das células vizinhas
95                  for k in valor:
96                      # if == 0 colocado pois quando o valor do inteiro é
    0 obtemos 3 matrizes iguais, dessa forma temos apenas 1
97                      # Determinação dos vizinhos, das respectivas distâ
    ncias e do ECN parcial por sítio X em relação ao sítio B
98                      if k == 0:
99                          vizinho = positions_x
100                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
101                         aux_dist_med = distmed[0]
102                         vesta = np.exp(1-(distancia[1]/aux_dist_med)**6)
103                         aux_vizinhos_I.append(vizinho)
104                         aux_distancias_I.append(distancia[1])
105                         aux_vesta_I.append(vesta)
106                         pass
107                     else:
108                     # Determinação dos vizinhos, das respectivas distâ
    ncias e do ECN parcial por sítio X em relação ao sítio B considerando
     os inteiros 1 e -1
109                         for n in rede:
110                             aux = k*cell[n,:]
111                             vizinho = positions_x + aux
112                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
113                             aux_dist_med = distmed[0]
114                             vesta = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
115                             aux_vizinhos_I.append(vizinho)
116                             aux_distancias_I.append(distancia[1])
117                             aux_vesta_I.append(vesta)
118
119             elif elemento == 'Br':
120                 ind_atomo = aux_xs[i]
121                 positions_x = atom.positions[ind_atomo]
122
123                 for k in valor:
```

```
124                     if k == 0:
125                         vizinho = positions_x
126                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
127                         aux_dist_med = distmed[1]
128                         vesta = np.exp(1-(distancia[1]/aux_dist_med)**6)
129                         aux_vizinhos_Br.append(vizinho)
130                         aux_distancias_Br.append(distancia[1])
131                         aux_vesta_Br.append(vesta)
132                         pass
133                     else:
134                         for n in rede:
135                             aux = k*cell[n,:]
136                             vizinho = positions_x + aux
137                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
138                             aux_dist_med = distmed[1]
139                             vesta = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
140                             aux_vizinhos_Br.append(vizinho)
141                             aux_distancias_Br.append(distancia[1])
142                             aux_vesta_Br.append(vesta)
143             else:
144                 ind_atomo = aux_xs[i]
145                 positions_x = atom.positions[ind_atomo]
146 
147                 for k in valor:
148                     if k == 0:
149                         vizinho = positions_x
150                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
151                         aux_dist_med = distmed[2]
152                         vesta = np.exp(1-(distancia[1]/aux_dist_med)**6)
153                         aux_vizinhos_Cl.append(vizinho)
154                         aux_distancias_Cl.append(distancia[1])
155                         aux_vesta_Cl.append(vesta)
156                         pass
157                     else:
158                         for n in rede:
159                             aux = k*cell[n,:]
160                             vizinho = positions_x + aux
161                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
162                             aux_dist_med = distmed[2]
163                             vesta = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
164                             aux_vizinhos_Cl.append(vizinho)
```

```
165                             aux_distancias_Cl.append(distancia[1])
166                             aux_vesta_Cl.append(vesta)
167
168         # Determinação do ECN quebrando por elemento do sítio X
169         if aux_vesta_I:
170             ecn_I = np.sum(aux_vesta_I)
171             ecn_vest.append(ecn_I)
172         if aux_vesta_Br:
173             ecn_Br = np.sum(aux_vesta_Br)
174             ecn_vest.append(ecn_Br)
175         if aux_vesta_Cl:
176             ecn_Cl = np.sum(aux_vesta_Cl)
177             ecn_vest.append(ecn_Cl)
178
179         # Determinando o número de elementos distintos do sítio X
    presentes na PVK para auxílio no rateio do ECN por elementos do sítio
     X
180         numero_de_none = distmed.count(None)
181         compri_distancias = len(distmed) - numero_de_none
182
183         # Realizando o rateio do ECN por sítio X
184         ECN_I = []# Valor ECN (X,I)
185         ECN_Br = []# Valor ECN (X,Br)
186         ECN_Cl = []# Valor ECN (X,Cl)
187
188         if len(aux_b) == 1:
189             if compri_distancias == 1:
190                 if 'I' in aux_xele:
191                     ECN_I.append(ecn_vest[0])
192                     ECN_Br.append(None)
193                     ECN_Cl.append(None)
194                 elif 'Br' in aux_xele:
195                     ECN_I.append(None)
196                     ECN_Br.append(ecn_vest[0])
197                     ECN_Cl.append(None)
198                 else:
199                     ECN_I.append(None)
200                     ECN_Br.append(None)
201                     ECN_Cl.append(ecn_vest[0])
202             if compri_distancias == 2:
203                 if 'I' in aux_xele and 'Br' in aux_xele:
204                     ECN_I.append(ecn_vest[0])
205                     ECN_Br.append(ecn_vest[1])
206                     ECN_Cl.append(None)
207                 elif 'I' in aux_xele and 'Cl' in aux_xele:
208                     ECN_I.append(ecn_vest[0])
209                     ECN_Br.append(None)
```

```
                    ECN_Cl.append(ecn_vest[1])
                else:#(Br,Cl)
                    ECN_I.append(None)
                    ECN_Br.append(ecn_vest[0])
                    ECN_Cl.append(ecn_vest[1])
            if compri_distancias == 3:
                    ECN_I.append(ecn_vest[0])
                    ECN_Br.append(ecn_vest[1])
                    ECN_Cl.append(ecn_vest[2])
        else:
            #else len(aux_b) > 1:
            if compri_distancias == 1:
                if 'I' in aux_xele:
                    auxECN_I.append(ecn_vest[0])
                elif 'Br' in aux_xele:
                    auxECN_Br.append(ecn_vest[0])
                else:
                    auxECN_Cl.append(ecn_vest[0])
            if compri_distancias == 2:
                if 'I' in aux_xele and 'Br' in aux_xele:
                    auxECN_I.append(ecn_vest[0])
                    auxECN_Br.append(ecn_vest[1])
                elif 'I' in aux_xele and 'Cl' in aux_xele:
                    auxECN_I.append(ecn_vest[0])
                    auxECN_Cl.append(ecn_vest[1])
                else:#(Br,Cl)
                    auxECN_Br.append(ecn_vest[0])
                    auxECN_Cl.append(ecn_vest[1])
            if compri_distancias == 3:
                    auxECN_I.append(ecn_vest[0])
                    auxECN_Br.append(ecn_vest[1])
                    auxECN_Cl.append(ecn_vest[2])

            if b == aux_b[-1]:
                denominador = len(aux_b)
                if auxECN_I:
                    ni = sum(auxECN_I)/denominador
                    ECN_I.append(ni)
                else:
                    ECN_I.append(None)

                if auxECN_Br:
                    nbr = sum(auxECN_Br)/denominador
                    ECN_Br.append(nbr)
                else:
                    ECN_Br.append(None)

```

```
                if auxECN_Cl:
                    ncl = sum(auxECN_Cl)/denominador
                    ECN_Cl.append(ncl)
                else:
                    ECN_Cl.append(None)

    print("Fórmula:", formula)
    print("Symbols:", symbols)
    print("Índice_b:", aux_b)
    print("Índice_x:", aux_xs)
    print("Índice_a:", aux_a)
    print("elementos_x:", aux_xele)
    print("ECN_I:", ECN_I)
    print("ECN_Br:", ECN_Br)
    print("ECN_Cl:", ECN_Cl)
    print("Índice_perovskita", perovskita)
    print("-----------------------------------")
```

B.0.5 ECN_{av}

```
def media_distancia(symbols, aux_b, minima, atom):
    # Definição de listas
    aux_xCl = []  # Índices do elemento Cl presentes na PVK
    aux_xBr = []  # Índices do elemento Br presentes na PVK
    aux_xI = []   # Índices do elemento I presentes na PVK
    aux_x = []    # Junção das listas anteriores com os índices
    distancias_Cl = []  # Distâncias calculadas entre (X,Cl)
    distancias_Br = []  # Distâncias calculadas entre (X,Br)
    distancias_I = []   # Distâncias calculadas entre (X,I)
    dist = []           # Mínima distância entre (X,X') com X' em Cl,Br,
   I
    posi_b = []         # Posições do elemento do sítio B
    posi_x = []         # Posições do elemento do sítio X
    dist_med = []       # Distância média
    numerador_I = []    # Numerador auxiliar para cálculo da distância
   para sítio I
    denominador_I = []  # Denominador auxiliar para cálculo da distância
    para sítio I
    numerador_Br = []   # Numerador auxiliar para cálculo da distância
   para sítio Br
    denominador_Br = [] # Denominador auxiliar para cálculo da distância
    para sítio Br
    numerador_Cl = []   # Numerador auxiliar para cálculo da distância
   para sítio Cl
    denominador_Cl = [] # Denominador auxiliar para cálculo da distância
    para sítio Cl

```

```
21      # For para quebrar em listas os índices do Cl, Br, I
22      for indice, elemento in enumerate(symbols):
23          if elemento == 'Cl':
24              aux_xCl.append(indice)
25          if elemento == 'Br':
26              aux_xBr.append(indice)
27          if elemento == 'I':
28              aux_xI.append(indice)
29
30      # Junção das listas em uma só
31      aux_x.append(aux_xI)
32      aux_x.append(aux_xBr)
33      aux_x.append(aux_xCl)
34
35      # For para percorrer as 3 listas com as distâncias, numerador e
    denominador para cada elemento do sítio X
36      indc = -1
37      for x in range(0, 3):
38          aux_distancias = []
39          aux_numerador = []
40          aux_denominador = []
41
42          # If para filtrar quais elementos a PVK em análise tem no sítio
    X e a respectiva distância mínima
43          if not aux_x[x]:
44              continue
45          else:
46              indc += 1
47              dist_min = minima[indc]
48
49              # For para pegar as posições separadamente dos átomos Cl, Br
    , I
50              for p in aux_x[x]:
51                  positions_x = atom.positions[p]
52                  posi_x.append(positions_x)
53
54                  # For para pegar posição de cada átomo do elemento do sí
    tio B
55                  for i in aux_b:
56                      positions_b = atom.positions[i]
57                      posi_b.append(positions_b)
58
59                      # For para percorrer os inteiros utilizados na
    determinação das células vizinhas, das respectivas distâncias e o
    numerador e denominador para cálculo da distância média
60                      for k in valor:
61                          # If == 0 colocado pois quando o valor do
```

```
    inteiro é 0 obtemos 3 matrizes iguais, dessa forma temos apenas 1
62                      if k == 0:
63                          vizinho = positions_x
64                          distancia = ase.geometry.get_distances(
    positions_b, vizinho)
65                          aux_distancias.append(distancia[1][0][0])
66                          numerador = distancia[1][0][0] * np.exp(1 -
    (distancia[1][0][0] / dist_min) ** 6)
67                          denominador = np.exp(1 - (distancia[1][0][0]
     / dist_min) ** 6)
68                          aux_numerador.append(numerador)
69                          aux_denominador.append(denominador)
70                      else:
71                          # Aqui é pego os vizinhos, as respectivas
    distâncias e o numerador e denominador para cálculo da distância mé
    dia, considerando os inteiros 1 e -1
72                          for n in rede:
73                              aux = k * cell[n, :]
74                              vizinho = positions_x + aux
75                              distancia = ase.geometry.get_distances(
    positions_b, vizinho)
76                              aux_distancias.append(distancia
    [1][0][0])
77                              numerador = distancia[1][0][0] * np.exp
    (1 - (distancia[1][0][0] / dist_min) ** 6)
78                              denominador = np.exp(1 - (distancia
    [1][0][0] / dist_min) ** 6)
79                              aux_numerador.append(numerador)
80                              aux_denominador.append(denominador)
81
82          # Determinação das distâncias e do numerador/denominador por
     elemento do sítio X para cálculo da distância média
83          if x == 0:
84              distancias_I.append(np.array(aux_distancias))
85              numerador_I.append(np.array(aux_numerador))
86              denominador_I.append(np.array(aux_denominador))
87          elif x == 1:
88              distancias_Br.append(np.array(aux_distancias))
89              numerador_Br.append(np.array(aux_numerador))
90              denominador_Br.append(np.array(aux_denominador))
91          else:
92              distancias_Cl.append(np.array(aux_distancias))
93              numerador_Cl.append(np.array(aux_numerador))
94              denominador_Cl.append(np.array(aux_denominador))
95
96     # Determinação das distâncias médias
97     if distancias_I:
```

```
98          num = np.sum(numerador_I)
99          den = np.sum(denominador_I)
100         d_med = num / den
101         dist_med.append(d_med)
102     if distancias_Br:
103         num = np.sum(numerador_Br)
104         den = np.sum(denominador_Br)
105         d_med = num / den
106         dist_med.append(d_med)
107     if distancias_Cl:
108         num = np.sum(numerador_Cl)
109         den = np.sum(denominador_Cl)
110         d_med = num / den
111         dist_med.append(d_med)
112
113     # Retorna lista com as distâncias médias na sequência I,Br,Cl
114     return dist_med
```

```
1  # Conexão com o banco
2  db = connect("organometal.db")
3
4  # Definição de listas
5  atoms = []# PVK's do banco
6  perovskita = []# Índice das PVK's
7  valor = [0,-1,1]# Inteiros para determinação dos vizinhos
8  rede = [0,1,2]# Vetores de rede
9  elex = []# Armazena qual o elemento do sítio X
10
11 # for para extrair as PVK's do banco
12 for row in db.select():
13     atoms.append(row.toatoms())
14
15 # Seleção de PVK's do banco
16 indices = [2,8,53,81,108,174,176,179,225,238]
17 atoms = [atoms[i] for i in indices]
18
19 # for para percorrer as PVK's selecionadas
20 for i, atom in enumerate(atoms):
21
22     # Fórmula da PVK
23     formula = atom.symbols
24     # Quebra da fórmula em átomos
25     symbols = atom.get_chemical_symbols()
26     perovskita.append(i+1)
27     # Traz os vetores de rede
28     cell = atom.get_cell().todict()['array']
29
30     # Definição de listas para armazenar os índices e elementos de cada
```

```
    átomo dos sítios
31   aux_b = []# Índice sítio B
32   aux_bele = []# Elemento sítio B
33   aux_xs = []# Índice sítio X
34   aux_xele = []# Elemento sítio X
35   aux_a = []# Índice sítio A
36   aux_aele = []# Elemento sítio A
37
38   # Classificação dos átomos por sítio, salvando seus índices em
    listas
39   for indice, elemento in enumerate(symbols):
40       if elemento in ['Au', 'Pb', 'Sn']:
41           aux_b.append(indice)
42           aux_bele.append(elemento)
43       elif elemento in ['Cl', 'Br', 'I']:
44           aux_xs.append(indice)
45           aux_xele.append(elemento)
46       else:
47           aux_a.append(indice)
48           aux_aele.append(elemento)
49
50   # Definição de listas auxiliares para a obtenção das posições
51   posi_b = []# Sítio B
52   posi_x = []# Sítio X
53
54   # Chamando a função para pegar a mínima distância
55   minima = minima_distancia(symbols, aux_b, atom)
56   minima = [arr[0][0] for arr in minima]
57
58   # Listas para auxiliar no cálculo da distância média autoconsistente
59   aux_distmedBr = []
60   aux_distmedI = []
61   aux_distmedCl = []
62
63   # While para calcular a distância média autoconsistente
64   result = 'false'
65   while (result == 'false'):
66       distmed0 = media_distancia(symbols, aux_b, minima, atom)
67
68       distmed1 = media_distancia(symbols, aux_b, distmed0, atom)
69
70       comp = len(distmed1)
71
72       if comp == 1:
73           if 'I' in aux_xele:
74               if distmed1[0] - distmed0[0] < 0.00010:
75                   result = 'true'
```

```
                    distmed = [distmed1[0], None, None]
                else:
                    result = 'false'
                    minima = distmed1
            elif 'Br' in aux_xele:
                if distmed1[0] - distmed0[0] < 0.00010:
                    result = 'true'
                    distmed = [None, distmed1[0], None]
                else:
                    result = 'false'
                    minima = distmed1
            else:
                if distmed1[0] - distmed0[0] < 0.00010:
                    result = 'true'
                    distmed = [None, None, distmed1[0]]
                else:
                    result = 'false'
                    minima = distmed1
        elif comp == 2:
            if 'I' in aux_xele and 'Br' in aux_xele:
                if distmed1[0] - distmed0[0] < 0.00010:
                    resultI = 'true'
                    aux_distmedI.append(distmed1[0])
                else:
                    resultI = 'false'
                    minima = distmed1

                if distmed1[1] - distmed0[1] < 0.00010:
                    resultBr = 'true'
                    aux_distmedBr.append(distmed1[1])
                else:
                    resultBr = 'false'
                    minima = distmed1

                if resultI == 'true' and resultBr == 'true':
                    result = 'true'
                    distmed = [aux_distmedI[0], aux_distmedBr[0], None]
                else:
                    result = 'false'
            elif 'I' in aux_xele and 'Cl' in aux_xele:
                if distmed1[0] - distmed0[0] < 0.00010:
                    resultI = 'true'
                    aux_distmedI.append(distmed1[0])
                else:
                    resultI = 'false'
                    minima = distmed1

```

```
123                 if distmed1[1] - distmed0[1] < 0.00010:
124                     resultCl = 'true'
125                     aux_distmedCl.append(distmed1[1])
126                 else:
127                     resultCl = 'false'
128                     minima = distmed1
129 
130                 if resultI == 'true' and resultCl == 'true':
131                     result = 'true'
132                     distmed = [aux_distmedI[0], None, aux_distmedCl[0]]
133                 else:
134                     result = 'false'
135             else:#(Br,Cl)
136                 if distmed1[0] - distmed0[0] < 0.00010:
137                     resultBr = 'true'
138                     aux_distmedBr.append(distmed1[0])
139                 else:
140                     resultBr = 'false'
141                     minima = distmed1
142 
143                 if distmed1[1] - distmed0[1] < 0.00010:
144                     resultCl = 'true'
145                     aux_distmedCl.append(distmed1[1])
146                 else:
147                     resultCl = 'false'
148                     minima = distmed1
149 
150                 if resultBr == 'true' and resultCl == 'true':
151                     result = 'true'
152                     distmed = [None, aux_distmedBr[0], aux_distmedCl[0]]
153                 else:
154                     result = 'false'
155         else: #compri_distancias == 3:
156             if distmed1[0] - distmed0[0] < 0.00010:
157                 resultI = 'true'
158                 aux_distmedI.append(distmed1[0])
159             else:
160                 resultI = 'false'
161                 minima = distmed1
162 
163             if distmed1[1] - distmed0[1] < 0.00010:
164                 resultBr = 'true'
165                 aux_distmedBr.append(distmed1[1])
166             else:
167                 resultBr = 'false'
168                 minima = distmed1
169 
```

```
170
171             if distmed1[2] - distmed0[2] < 0.00010:
172                 resultCl = 'true'
173                 aux_distmedCl.append(distmed1[2])
174             else:
175                 resultCl = 'false'
176                 minima = distmed1
177
178             if resultI == 'true' and resultBr == 'true' and resultCl ==
    'true':
179                 result = 'true'
180                 distmed = [aux_distmedI[0], aux_distmedBr[0],
    aux_distmedCl[0]]
181             else:
182                 result = 'false'
183
184     # Definição de listas auxiliares para confecção de gráficos
185     auxECN_I = []
186     auxECN_Br = []
187     auxECN_Cl = []
188
189     # if para determinar qual o elemento do sítio X
190     if 'Pb' in aux_bele:
191         elex.append('Pb')
192     else:
193         elex.append('Sn')
194
195     # for para pegar o positions(vetor posição) do átomo do sítio B
196     for b in aux_b:
197         # Definição de listas auxiliares para obtenção dos vizinhos e
    das distâncias com o sítio B e cálculo do ECN
198         aux_vizinhos_I = []
199         aux_distancias_I = []
200         aux_vizinhos_Br = []
201         aux_distancias_Br = []
202         aux_vizinhos_Cl = []
203         aux_distancias_Cl = []
204         distancias = []
205         aux_av_I = []
206         aux_av_Br = []
207         aux_av_Cl = []
208         ecn_av = []
209         positions_b = atom.positions[b]
210         posi_b.append(positions_b)
211
212         # for para obter o índice do átomo do sítio x e sua respectiva
    posição
```

```
213         for i, elemento in enumerate(aux_xele):
214             if elemento == 'I':
215                 ind_atomo = aux_xs[i]
216                 positions_x = atom.positions[ind_atomo]
217 
218                 # for para percorrer os inteiros utilizados na determina
    ção das células vizinhas
219                 for k in valor:
220                 # if == 0 colocado pois quando o valor do inteiro é 0
    obtemos 3 matrizes iguais, dessa forma temos apenas 1
221                 # Determinação dos vizinhos, das respectivas distâncias
    e do ECN parcial por sítio X em relação ao sítio B
222                     if k == 0:
223                         vizinho = positions_x
224                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
225                         aux_dist_med = distmed[0]
226                         cord = np.exp(1-(distancia[1]/aux_dist_med)**6)
227                         aux_vizinhos_I.append(vizinho)
228                         aux_distancias_I.append(distancia[1])
229                         aux_av_I.append(cord)
230                         pass
231                     else:
232                     # Determinação dos vizinhos, das respectivas distâ
    ncias e do ECN parcial por sítio X em relação ao sítio B considerando
     os inteiros 1 e -1
233                         for n in rede:
234                             aux = k*cell[n,:]
235                             vizinho = positions_x + aux
236                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
237                             aux_dist_med = distmed[0]
238                             cord = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
239                             aux_vizinhos_I.append(vizinho)
240                             aux_distancias_I.append(distancia[1])
241                             aux_av_I.append(cord)
242 
243             elif elemento == 'Br':
244                 ind_atomo = aux_xs[i]
245                 positions_x = atom.positions[ind_atomo]
246 
247                 for k in valor:
248                     if k == 0:
249                         vizinho = positions_x
250                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
```

```
251                         aux_dist_med = distmed[1]
252                         cord = np.exp(1-(distancia[1]/aux_dist_med)**6)
253                         aux_vizinhos_Br.append(vizinho)
254                         aux_distancias_Br.append(distancia[1])
255                         aux_av_Br.append(cord)
256                         pass
257                     else:
258                         for n in rede:
259                             aux = k*cell[n,:]
260                             vizinho = positions_x + aux
261                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
262                             aux_dist_med = distmed[1]
263                             cord = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
264                             aux_vizinhos_Br.append(vizinho)
265                             aux_distancias_Br.append(distancia[1])
266                             aux_av_Br.append(cord)
267
268             else:
269                 ind_atomo = aux_xs[i]
270                 positions_x = atom.positions[ind_atomo]
271
272                 for k in valor:
273                     if k == 0:
274                         vizinho = positions_x
275                         distancia = ase.geometry.get_distances(
    positions_b, vizinho)
276                         aux_dist_med = distmed[2]
277                         cord = np.exp(1-(distancia[1]/aux_dist_med)**6)
278                         aux_vizinhos_Cl.append(vizinho)
279                         aux_distancias_Cl.append(distancia[1])
280                         aux_av_Cl.append(cord)
281                         pass
282                     else:
283                         for n in rede:
284                             aux = k*cell[n,:]
285                             vizinho = positions_x + aux
286                             distancia = ase.geometry.get_distances(
    positions_b, vizinho)
287                             aux_dist_med = distmed[2]
288                             cord = np.exp(1-(distancia[1]/aux_dist_med)
    **6)
289                             aux_vizinhos_Cl.append(vizinho)
290                             aux_distancias_Cl.append(distancia[1])
291                             aux_av_Cl.append(cord)
292
```

```
293         # Determinação do ECN quebrando por elemento do sítio X
294         if aux_av_I:
295             ecn_I = np.sum(aux_av_I)
296             ecn_av.append(ecn_I)
297         if aux_av_Br:
298             ecn_Br = np.sum(aux_av_Br)
299             ecn_av.append(ecn_Br)
300         if aux_av_Cl:
301             ecn_Cl = np.sum(aux_av_Cl)
302             ecn_av.append(ecn_Cl)
303 
304         # Determinando o número de elementos distintos do sítio X
    presentes na PVK para auxílio no rateio do ECN por elementos do sítio
     X
305         numero_de_none = distmed.count(None)
306         compri_distancias = len(distmed) - numero_de_none
307 
308         # Realizando o rateio do ECN por sítio X
309         ECN_I = []# Valor ECN (X,I)
310         ECN_Br = []# Valor ECN (X,Br)
311         ECN_Cl = []# Valor ECN (X,Cl)
312 
313         if len(aux_b) == 1:
314             if compri_distancias == 1:
315                 if 'I' in aux_xele:
316                     ECN_I.append(ecn_av[0])
317                     ECN_Br.append(None)
318                     ECN_Cl.append(None)
319                 elif 'Br' in aux_xele:
320                     ECN_I.append(None)
321                     ECN_Br.append(ecn_av[0])
322                     ECN_Cl.append(None)
323                 else:
324                     ECN_I.append(None)
325                     ECN_Br.append(None)
326                     ECN_Cl.append(ecn_av[0])
327             if compri_distancias == 2:
328                 if 'I' in aux_xele and 'Br' in aux_xele:
329                     ECN_I.append(ecn_av[0])
330                     ECN_Br.append(ecn_av[1])
331                     ECN_Cl.append(None)
332                 elif 'I' in aux_xele and 'Cl' in aux_xele:
333                     ECN_I.append(ecn_av[0])
334                     ECN_Br.append(None)
335                     ECN_Cl.append(ecn_av[1])
336                 else:#(Br,Cl)
337                     ECN_I.append(None)
```

```
338                     ECN_Br.append(ecn_av[0])
339                     ECN_Cl.append(ecn_av[1])
340             if compri_distancias == 3:
341                     ECN_I.append(ecn_av[0])
342                     ECN_Br.append(ecn_av[1])
343                     ECN_Cl.append(ecn_av[2])
344         else:
345             #else len(aux_b) > 1:
346             if compri_distancias == 1:
347                 if 'I' in aux_xele:
348                     auxECN_I.append(ecn_av[0])
349                 elif 'Br' in aux_xele:
350                     auxECN_Br.append(ecn_av[0])
351                 else:
352                     auxECN_Cl.append(ecn_av[0])
353             if compri_distancias == 2:
354                 if 'I' in aux_xele and 'Br' in aux_xele:
355                     auxECN_I.append(ecn_av[0])
356                     auxECN_Br.append(ecn_av[1])
357                 elif 'I' in aux_xele and 'Cl' in aux_xele:
358                     auxECN_I.append(ecn_av[0])
359                     auxECN_Cl.append(ecn_av[1])
360                 else:#(Br,Cl)
361                     auxECN_Br.append(ecn_av[0])
362                     auxECN_Cl.append(ecn_av[1])
363             if compri_distancias == 3:
364                     auxECN_I.append(ecn_av[0])
365                     auxECN_Br.append(ecn_av[1])
366                     auxECN_Cl.append(ecn_av[2])
367 
368             if b == aux_b[-1]:
369                 denominador = len(aux_b)
370                 if auxECN_I:
371                     ni = sum(auxECN_I)/denominador
372                     ECN_I.append(ni)
373                 else:
374                     ECN_I.append(None)
375 
376                 if auxECN_Br:
377                     nbr = sum(auxECN_Br)/denominador
378                     cnbr = nbr
379                     ECN_Br.append(cnbr)
380                 else:
381                     ECN_Br.append(None)
382 
383                 if auxECN_Cl:
384                     ncl = sum(auxECN_Cl)/denominador
```

```
385                     ECN_Cl.append(ncl)
386                 else:
387                     ECN_Cl.append(None)
388
389
390
391
392     print("Fórmula:", formula)
393     print("Symbols:", symbols)
394     print("Índice_b:", aux_b)
395     print("Índice_x:", aux_xs)
396     print("Índice_a:", aux_a)
397     print("Elementos_x:", aux_xele)
398     print("ECNav_I:", ECN_I)
399     print("ECNav_Br:", ECN_Br)
400     print("ECNav_Cl:", ECN_Cl)
401     print("Índice_perovskita", perovskita)
402     print("-------------------------------------")
```

APÊNDICE C – Gráficos

C.0.1 Barras Empilhadas

Devido ao número de PVK's e consequente dificuldade em exposição, foi feita a quebra em intervalos de 60 materiais.

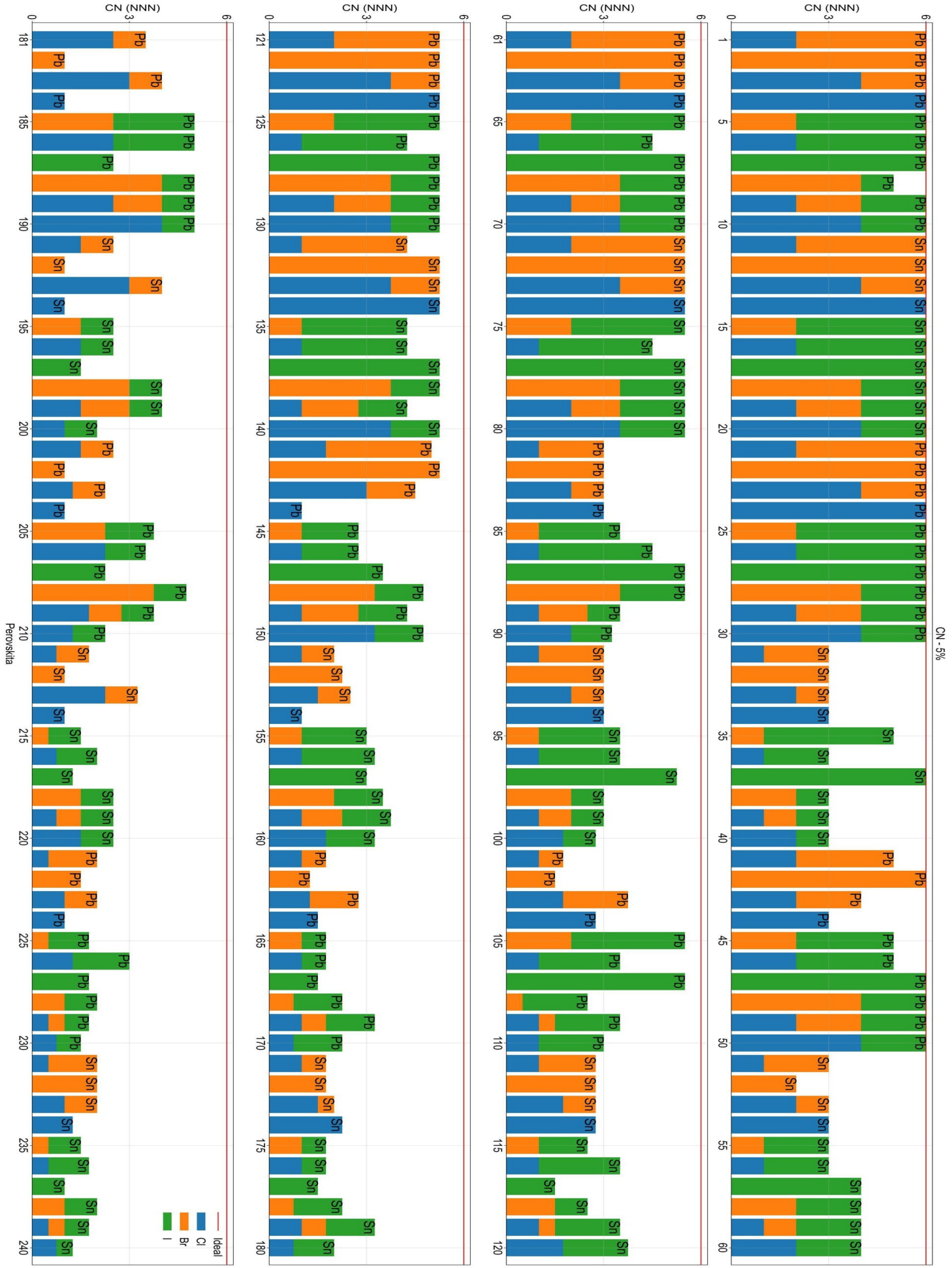


Figura C.1 – CN - 5%.

Figura C.2 – CN - 10%.

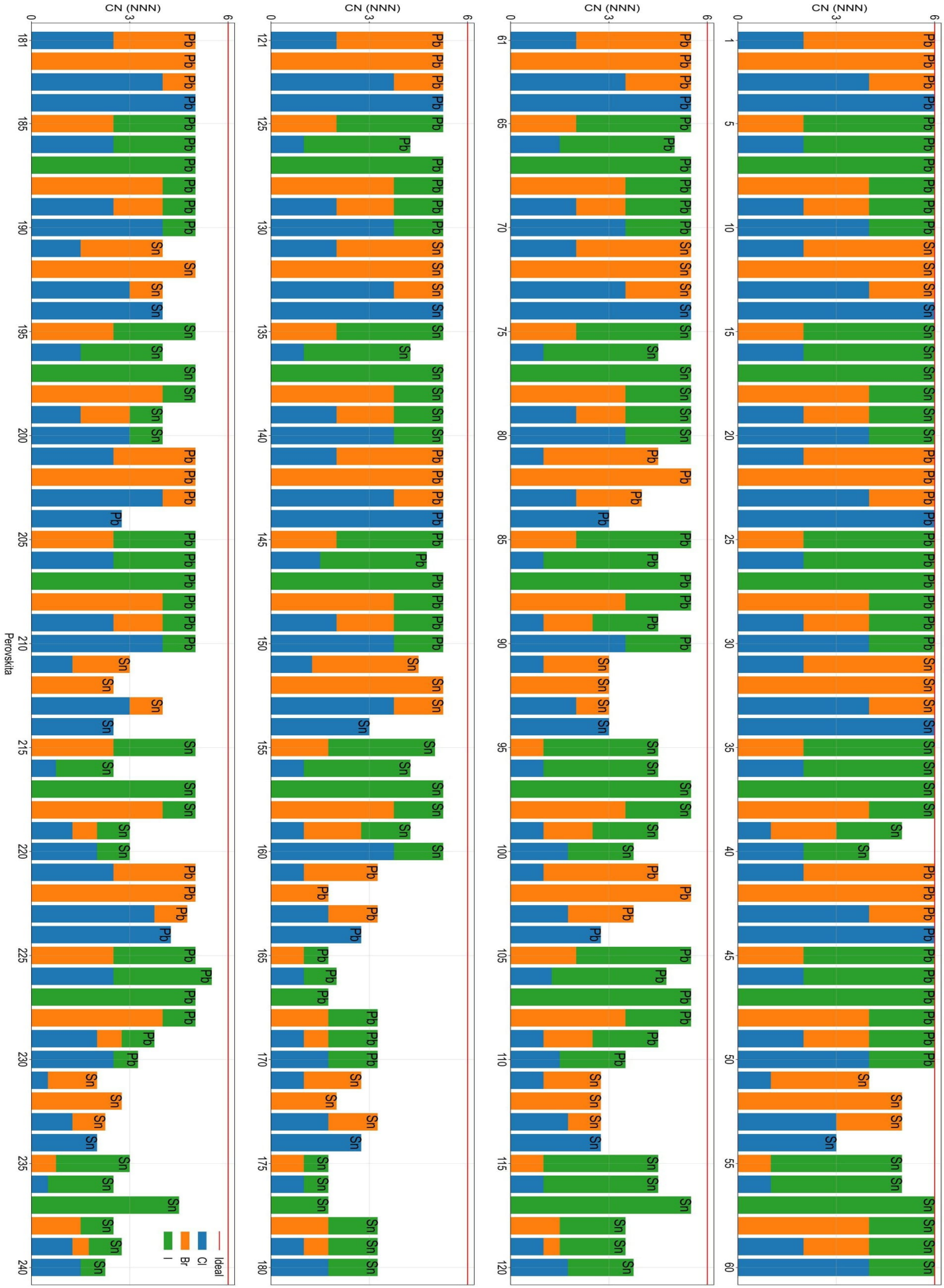


Figura C.3 – CN - 20%.

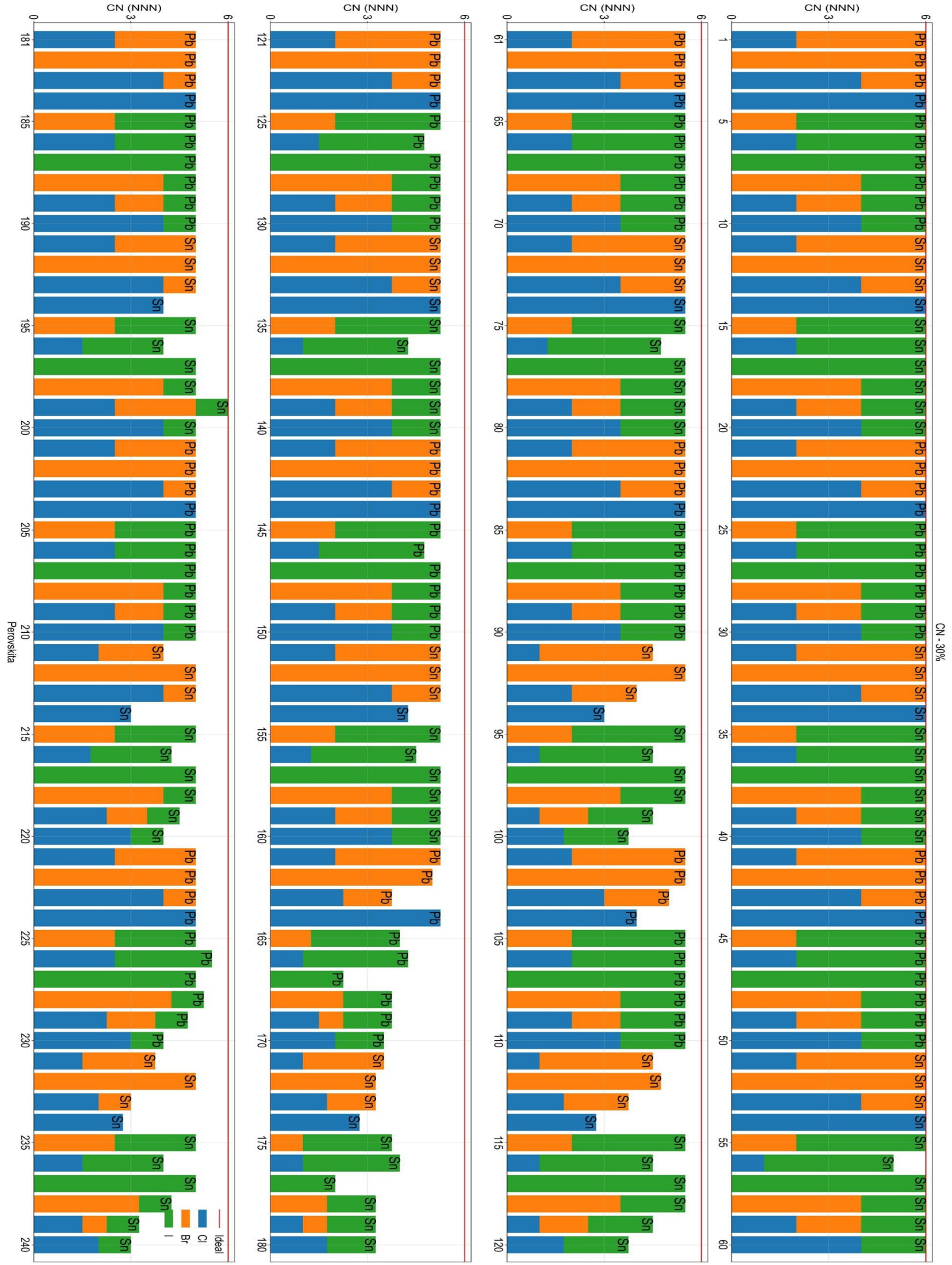


Figura C.4 – CN - 30%.

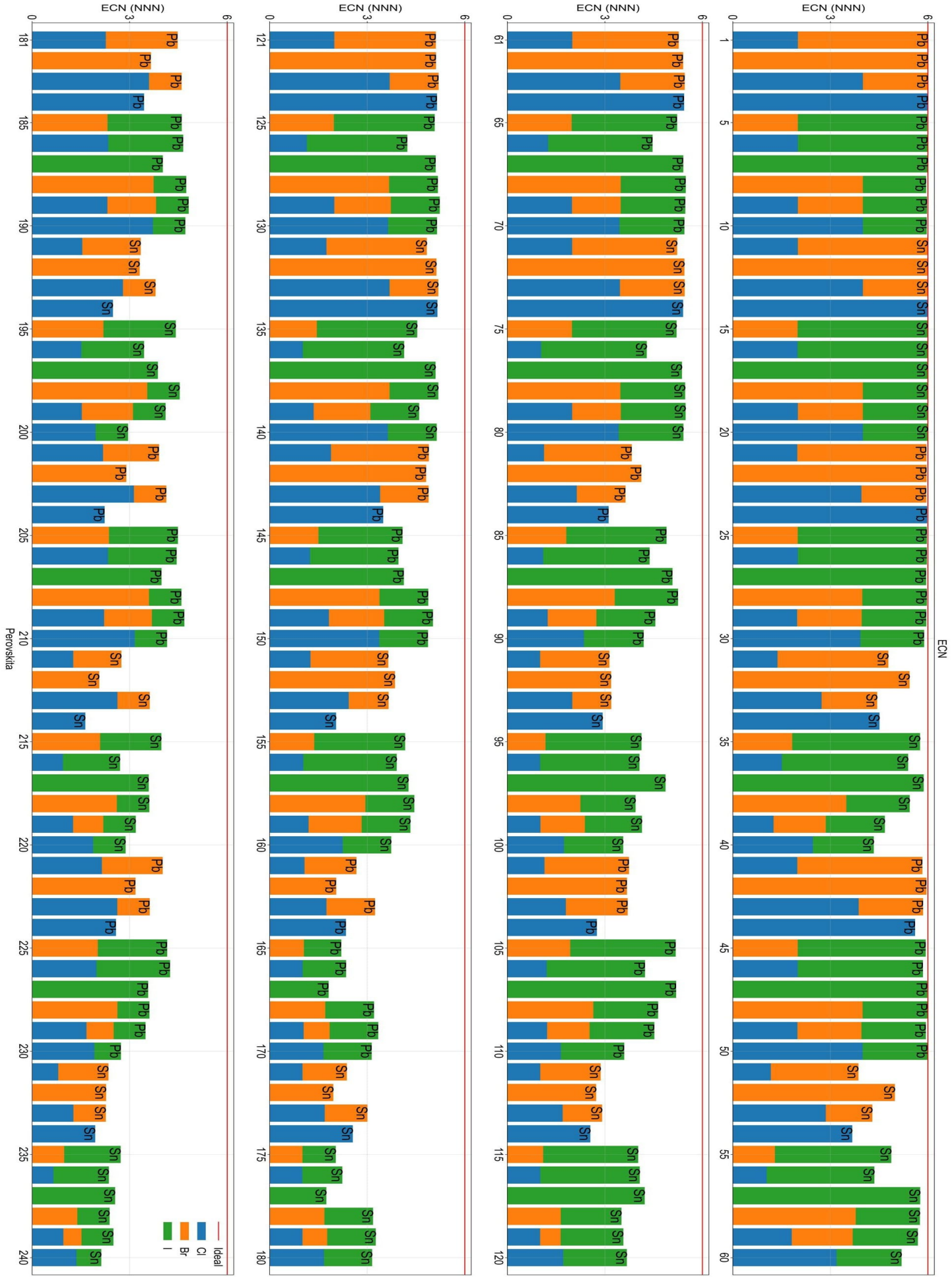


Figura C.5 – ECN.

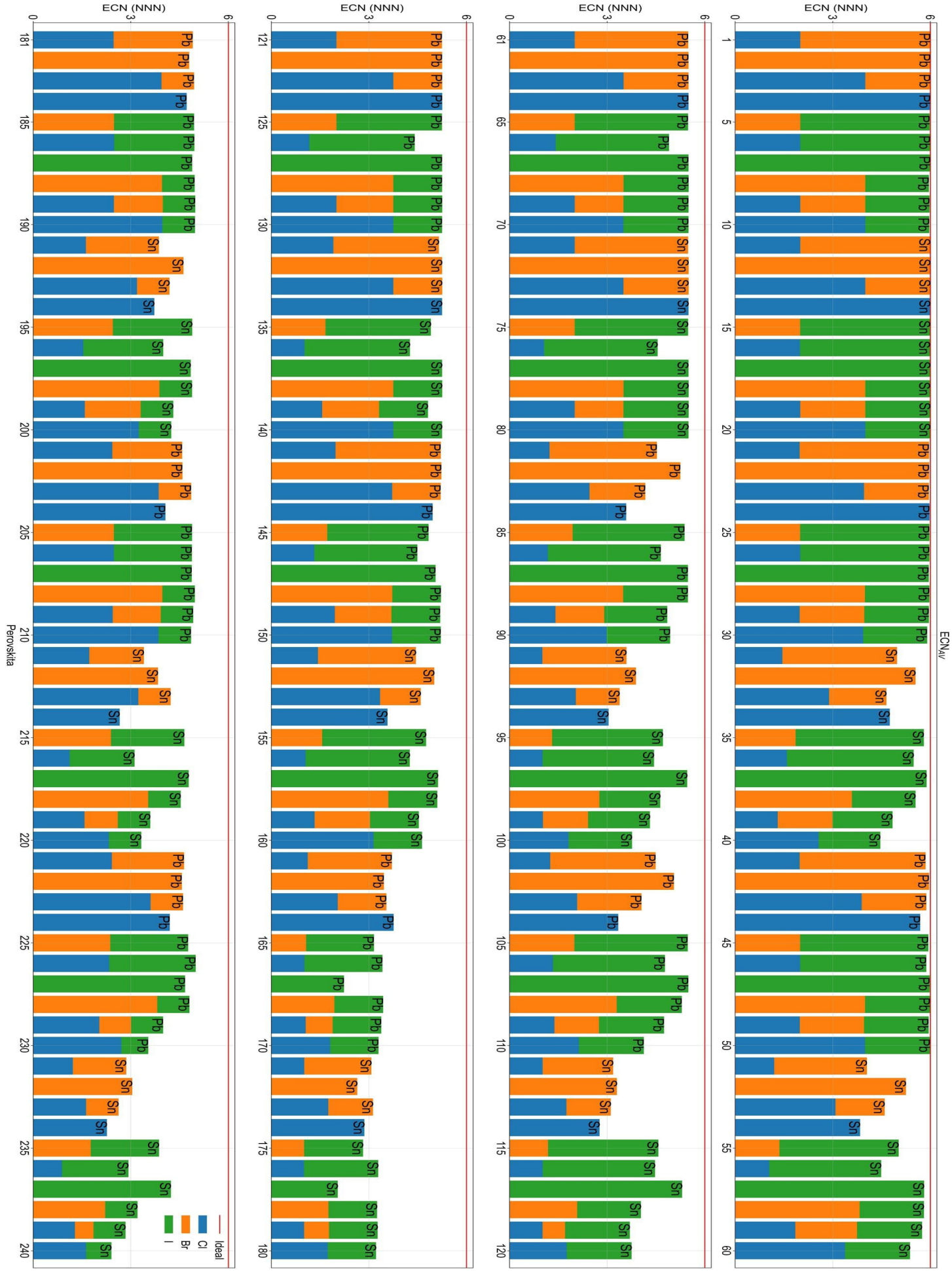


Figura C.6 – ECN_{av}.