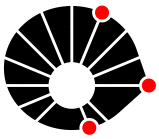

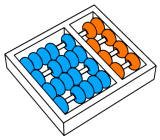

PLI aplicado ao jogo *Path of Exile*

L. F. André *F.K. Miyazawa*

Relatório Técnico - IC-PFG-22-26
Projeto Final de Graduação
2022 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

PLI aplicado ao jogo *Path of Exile*

Lucas Fernandes André* Flávio Keidi Miyazawa†

Resumo

A aplicação de técnicas de redução exata na resolução de jogos atuais é algo pouco explorado. Neste trabalho, aplica-se a abordagem de Programação Linear Inteira e suas derivadas análises para tratar de um problema muito comum no jogo *Path of Exile*. Avalia-se o desempenho de duas formulações, uma por fluxo e outra por cortes, baseados na resolução do Problema da Árvore de Steiner com Coleta de Prêmios. Para a representação da rede de conexão do jogo, bem como algoritmos para exploração, foi utilizada a biblioteca para grafos *LEMON* e para a resolução dos modelos de programação linear inteira, o solver *Gurobi*. Avaliou-se os resultados obtidos para diferentes instâncias do problema e foram sugeridos próximos passos para dar continuidade à esta análise.

1 Introdução

Path of Exile é um *Action Role Playing Game* (*ARPG*), criado pela empresa *Grinding Gear Games* e lançado no ano de 2013, em que o jogador controla um personagem, escolhido dentro de um conjunto de sete possibilidades denominadas "exilados". Dentro do jogo, o jogador é capaz de subir de nível ao derrotar monstros, cada um desses níveis recompensa o personagem com um ponto de habilidade (*skill point*). *Skill points* podem ser usados para adquirir atributos para seu personagem, sendo alocados na rede de habilidade do jogo (*skill network*). Esta *skill network* ganhou notoriedade dentre a comunidade por causa de seu tamanho e complexidade à primeira vista. Cada vértice possui um prêmio associado, que é uma moeda diferente dos pontos de habilidade. As regras para percorrê-la são apresentadas a seguir:

- O jogador deve escolher um "exilado"que possui um ponto inicial na *skill network* e começa a percorrer sua rede à partir deste ponto.
- Ao se derrotar monstros, são recompensados *skills points* que serão usados para percorrer a *skill network*.
- O jogador possui um número n de *skill points* para gastar.

*Graduando em Engenharia de Computação, Instituto de Computação, Universidade Estadual de Campinas.

†Professor do Instituto de Computação, Universidade Estadual de Campinas. Av. Albert Einstein, 1251. Cidade Universitária. Caixa Postal 6176, CEP 13083-852, Campinas, SP - Brasil.

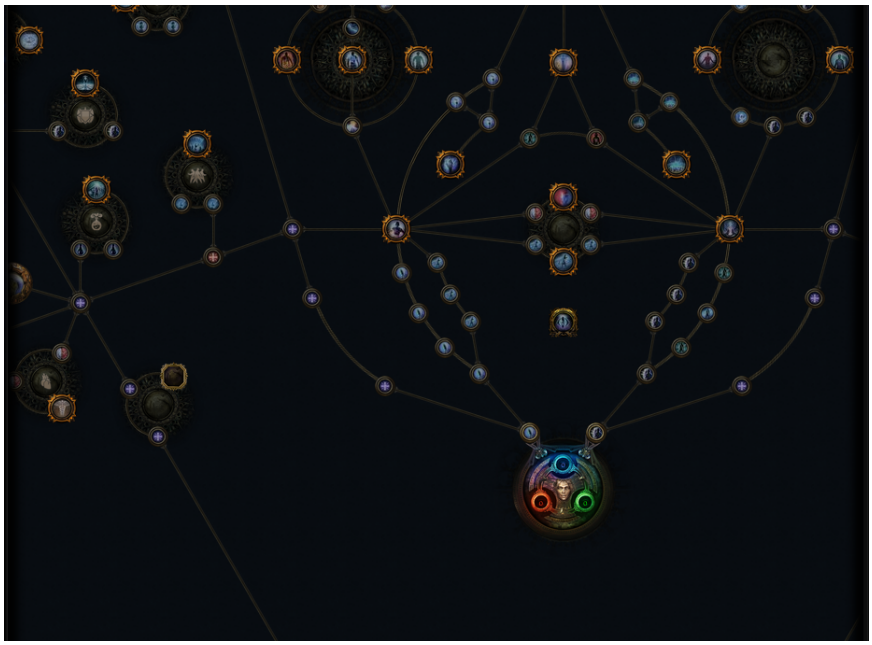

Figura 1: Trecho da rede de habilidades do jogo

- Ao se chegar à um vértice qualquer, ganha-se seu prêmio, que são bônus que deixam o personagem mais forte.
- Todas as arestas do grafo possuem pesos iguais (se gasta um ponto de habilidade para obter cada aresta).
- O jogador só pode obter o bônus/prêmio de um vértice uma vez.
- Um ponto de habilidade só pode ser adquirido se houver uma aresta conectando-o à um vértice já obtido.

Existem diversas maneiras de percorrer esta rede com os mais diversos objetivos. Neste trabalho, será abordada uma maneira específica. Construir uma árvore que obtém a maior quantidade de prêmios possíveis com a restrição de ter um certo conjunto de pontos que necessitam ser alcançados e um número máximo n de arestas a serem pegas. A resolução deste problema pode auxiliar jogadores a otimizarem suas redes de habilidade que é a principal motivação deste trabalho.

Colocando em termos mais próximos de grafos, tem-se um grafo inicial, conexo e não dirigido. Tem-se então um conjunto de pontos que precisam ser alcançados e além disso, um número máximo de arcos que podem ser pegos, anteriormente denominado n. Como cada vértice tem uma premiação associada, esta enunciação se faz parecer muito com uma variação do problema da árvore de Steiner com coleta de prêmios, com a diferença que os pesos de suas arestas são sempre iguais[6].

1.1 Redução

O problema K-MST[1] é um problema muito famoso de otimização em que o objetivo é encontrar uma árvore de peso mínimo que possuí um número k de vértices e que mínimize o custo total de suas arestas. Esse problema é NP-difícil, o que significa que não se conhece algoritmo que possa encontrar a solução ótima para todos os casos em tempo computacional polinomial e é improvável que tais algoritmos existam. No entanto existem algoritmos heurísticos que podem encontrar soluções de valores próximos da ótima, em tempo polinomial.

Uma definição formal do problema k-MST seria: Dado grafo $G = (V, E)$, custos nas arestas $c : E \to R^+$, e inteiro positivo k. O objetivo é encontrar uma árvore $T = (V_T, E_T)$ em G, tal que V_T tem exatamente k vértices e $\sum_{e \in E_T} c(e)$ é mínimo.

E uma definição formal do problema n-Path of Exile (n-PE): Dado um grafo $G = (V, E)$, conjunto de exilados $X \subset V$, prêmios nos vértices V dados por função $p : V \to R^+$, e inteiro positivo n. O objetivo é encontrar uma árvore $H = V_H, E_H)$ de G, contendo exatamente n vértices, tal que $\sum_{v \in V_H} p(v)$ é máximo.

Nota-se que o k-MST e o n-PE diferem no sentido da função de otimização. O primeiro é de minimização e o segundo de maximização. Mas a função de custo c e a função de prêmio são não negativas.

Assim, primeiramente, podemos mostrar a equivalência entre o problema k-MST para um problema de maximização que chamaremos de Max-k-MST que consiste exatamente no problema k-MST, porém com uma árvore com k-vértices em que o custo das arestas é máximo. Para isso, a entrada de k-MST será parecida com a entrada para Max-k-MST, e a única diferença será a função de custo nas arestas, que para este último problema, usará uma função w. Será utilizado um valor bem grande, denotado por $M = (1 + |V| * max_{e \in E} c(e))$. Isto é, M é maior que a soma de todas os pesos de todas as arestas do grafo. Agora, define-se uma função de peso nas arestas $w : E \to R^+$, definindo w da seguinte forma: $w(e) = M - c(e)$. Com isso, encontrar uma árvore $T = (V_T, E_T)$ em G tal que V_T tem k arestas e $\sum_{e \in E_T} c(e)$ é mínimo, é o mesmo que encontrar uma árvore $H = (V_H, E_H)$ em G tal que V_H tem k vértices e $\sum_{e \in E_T} w(e)$ é máximo. Pois,

$$\begin{aligned} \sum_{e \in E_T} w(e) &= \sum_{e \in E_T} (M - c(e)) \\ &= (\sum_{e \in E_T} M) - (\sum_{e \in E_T} c(e)) \\ &= (k-1) * M - (\sum_{e \in E_T} c(e)) \end{aligned} \tag{1}$$

e como $(k-1) * M$ é um valor fixo, portanto buscar uma árvore de k vértices que minimiza a soma dos custos das arestas pela função c, é o mesmo que buscar uma árvore de k vértices que maximiza a soma dos custos nas arestas pela função w.

Portanto, o problema Max-k-MST é um problema NP-difícil (ou NP-completo na versão de decisão).

Agora, resta reduzir o problema Max-k-MST para o problema n-PE.

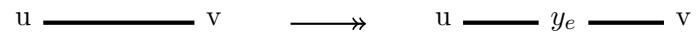

Figura 2: Transformação das arestas do grafo

Dado uma entrada $(G = (V, E), k, w)$ para o problema Max-k-MST, irá se construir uma entrada $(G' = (V', E'), Exilados, n, p)$ para o problema n-PE, de maneira que a solução em um problema leva a solução no outro problema, diferindo basicamente de um valor fixo.

Nesta redução, constrói-se um grafo G' a partir de G da seguinte forma: Para cada vértice v de G, é acrescentado um novo vértice x_v e ligue-se o vértice v ao vértice x_v. Assim, o grafo inicial que tinha começado apenas com o grafo G, agora vai conter vários vértices (um vértice novo para cada vértice de G) que será uma 'folha' / vértice pendurado de grau 1). Defina o conjunto de Exilados como o conjunto dos vértices 'folhas' x_v, para todo v em V.

Agora, para cada aresta $e = \{u, v\}$, divida ela colocando um vértice novo bem no meio da aresta e. Vamos chamar este vértice novo, de y_e. Dada a aresta $e = \{u, v\}$ em G, vamos denotar as arestas $\{u, y_e\}$ e $\{y_e, v\}$ como e' e e'', respectivamente, e chamá-las de 'arestas irmãs'. Essa transformação, para uma aresta, está representada na figura 2.

Como o problema n-PE tenta maximizar o peso nos vértices pegos na árvore escolhida (e não o peso das arestas), coloca-se o peso da aresta $e = \{u, v\}$, dado por $w(e)$, como o prêmio do vértice y_e. Isto é, define-se o prêmio $p(y_e)$ com o valor $c(e)$.

Define-se também o valor do prêmio dos vértices originais como sendo $N = M * M$. Nota-se que o valor N é muito maior que o valor de M.

Agora, considere a definição do prêmio dos vértices 'folhas' que são os vértices exilados o jogador deve escolher um destes vértices para ser a raiz da árvore). Para um vértice exilado x_v, é definido seu prêmio como 0 (zero). Assim, no problema de maximização, só será vantajoso pegar/escolher apenas um vértice exilado. Coloca-se cada vértice original, como os vértices u e v na figura 2 com o valor N.

O objetivo é forçar que o algoritmo sempre prefira pegar o trecho alterado inteiro mostrado na figura 2 pois os pesos de (u) e de (v) são muito maiores que o peso de (y_e). Já que u e v tem pesos N e y_e tem peso $w(e) = M - c(e)$.

Agora, basta definir o valor de n como sendo $n = 2 * (k - 1) + 2$. Com isso, a árvore do problema n-PE vai terá n vértices e consequentemente, $n - 1$ arestas. Isto é, deverá ter $2 * (k - 1) + 1$ arestas. Como a solução vai pegar apenas um vértice exilado, diga-se x_v (ligado a seu correspondente vértice v), a árvore terá essa aresta (x_v, v) e as outras arestas restantes dessa árvore vão ser selecionadas em pares, como na figura 2, pegando as duas arestas correspondentes. Assim, se no grafo G' existir uma aresta $e = \{u, v\}$ e em G' existe o trecho representado pela transformação da figura 2, o trecho inteiro será adquirido.

Agora, supondo que a solução ótima de G' adquire apenas uma das arestas 'irmãs' $e' = u, y_e$ ou $e'' = \{y_e, v\}$, então deve haver alguma outra aresta f' que também foi pega apenas f' ou f''. Se isso acontecer, descarta-se as duas arestas pegas em e', e'', f', f'' (o restante da árvore continua acíclico e conexo) e pega-se um par completo de arestas consecutivas g' e g'' que liga a um vértice (i) da árvore atual com um vértice novo (j) que não pertence a árvore atual (nota-se que é possível adquirir estas duas novas arestas, pois removeu-se da

árvore outras duas arestas anteriormente.

Com isso, tem-se uma nova árvore de mesmo tamanho, sem dois vértides do tipo y_e ou y_f, e inseriu-se dois vértices, um deles sendo o vértice j que tem um peso bem grande (definido anteriormente com valor de prêmio igual a $N = M * M$). Então, o valor total desta nova árvore melhorou, mas isso nos leva a uma contradição, pois começa-se com uma solução que era ótima em G. Portanto, em uma solução ótima em G', se uma aresta irmã está na solução ótima, a aresta 'irmã' dela também está nessa solução ótima.

Com isso, é obtido um mapeamento direto do prêmio do vértice que liga as arestas 'irmãs' com o custo da correspondente aresta em G, já que prêmio do vértice que liga as arestas irmãs é o mesmo que o custo da correspondente aresta. Isto é, se o ótimo na solução do problema Max-k-MST tem uma aresta e, então a correspondente solução em G' tem as arestas irmãs e' e e'', e com isso terá pego o vértice entre essas duas que tem prêmio w_e. Assim, essa solução ótima terá no total $2 * (k - 1)$ vértices, além de um vértice exilado. Estas $2 * (k - 1)$ arestas foram pegas em pares, considerando as arestas irmãs e que levam a uma árvore em G contendo uma árvore de $(k - 1)$ arestas em G, e portanto uma árvore com k vértices em G. O valor do prêmio total na solução ótima de G' (do problema n-PE) com o valor ótimo da solução de G (do problema Max-k-MST) diferem do peso dos vértices originais em G', que nos dá $k * N$ (que é $k * M * M$) e que é um valor fixo). Assim, tirando o valor $k * N$ no valor da solução do grafo G' do problema n-PE, temos um valor/mapeamento para uma solução Max-k-MST no grafo G. Assim, ambos os problemas são equivalentes, onde a solução em um problema leva a uma solução do outro problema, diferindo apenas de um valor fixo.

2 Metodologia

Abaixo serão apresentadas todas as etapas teóricas e práticas deste trabalho, como foram obtidas, suas explicações e dificuldades encontradas.

2.1 Preparação

Como foi dito anteriormente, a instância a ser explorada tem origem em um jogo, para tanto então, é necessário exportar os dados desta rede de habilidades. A partir do *github* da empresa *Grinding Gear Games*, é possível obter um arquivo *JSON* com os conteúdos desta rede. A partir deste arquivo, foram realizadas explorações no dicionário e criada uma tabela à partir do pacote *python pandas* contendo todos os vértices, seus nomes, suas ligações e posição no plano xy. Após, foi feita uma limpeza de alguns vértices que não fariam parte desta análise pois não apresentam um bônus atribuído à eles.

Após a realização inicial desta fase de limpeza, se construiu um grafo não-direcionado para representar a rede de habilidades do jogo. Este grafo foi armazenado em um arquivo contendo a quantidade de nós, a quantidade de arestas, o nome de cada um dos nós, suas posições, um atributo binário que indica se um nó é terminal ou não (valores iguais a 1 indicam que é terminal) e o prêmio associado ao vértice. Em seguida são apresentados as arestas do grafo, com os vértices das extremidades da aresta e seu peso, que neste caso é igual a 1.

2.2 Implementação

Apesar da proposta deste trabalho ser auxiliar jogadores de Path of Exile, os formatos, linguagens e dados trabalhados são de alta especifidade técnica e não são facilmente palatáveis para a comunidade do jogo. Isto se deve ao fato de não existirem as ferramentas para este tipo de análise dentro da comunidade. Espera-se que com as elucidações e trabalhos realizados, seja de maior facilidade a implementação e melhoramento das ferramentas para que possam ser postas a uso pela comunidade.

Dito isto, utilizou-se de abordagens de PLI, reconhecidas por estarem entre os principais métodos de resolução de problemas NP-difíceis. Neste caso, utilizou-se o programa de resolução *Gurobi* 9.5.1[3] para resolver o PLI, utilizando-se uma licença acadêmica. Utilizou-se também a biblioteca *LEMON* 1.3.1[4] que providencia rotinas para representação e manipulação de grafos, tornando a confecção do código mais robusta.

O código foi implementado com base em rotinas já desenvolvidas pelo supervisor[2].

2.3 Formulações

Os problemas da Árvore de Steiner com coleta de prêmios podem apresentar tanto peso nas arestas, quanto prêmios nos vértices para minimizar a função objetivo. Como este problema possui algumas diferenças, serão propostas duas formulações adaptadas, uma sendo por Cortes Direcionados e outra por fluxo dos vértices. Para especificar o problema de interesse, coloca-se o peso das arestas com valor uniforme igual a 1, fazendo com que não haja diferença entre coletar duas arestas diferentes que levam ao mesmo vértice e também que quer-se obter a maior premiação possível dos vértices atingidos.

O problema foi formulado através de um grafo direcionado $G = (V, A)$, em que suas arestas tem peso igual e cada vértice $i \in V$ tem uma premiação associada p_i, um vértice raiz r (um ponto artificial que se liga à cada um dos pontos iniciais dos "exilados") e com um conjunto $T \subset V$ de vértices terminais que precisam ser alcançados, quer-se então um subgrafo que alcance todos os terminais $t \in T$ e que maximize a premiação coletada, tendo em conta que o número de arestas adquiridas seja igual a um número dado, chamado de n.

2.3.1 Formulação por fluxo

Tomando em conta as informações apresentadas anteriormente, é necessário definir mais algumas notações. Inicialmente, haverá um vértice r que gera uma quantidade de fluxo[5] igual a k, que é o número de vértices do problema. Cada vértice pego "consome"uma unidade de fluxo. O fluxo é transportado por arcos a serem escolhidos para a solução. Além disso, também são criadas arestas, denominadas vermelhas, que saem de r e vão para todos os vértices não terminais. Se faz isso para que o somatório de fluxo seja igual a k(que é o número de vértices do problema) e embora sejam colocados todos os vértices na solução, aqueles conectados por arestas vermelhas não farão parte da solução do jogo. Cria-se então um conjunto de arcos chamado VR, para que o prêmios dos vértices em que tais arcos entram seja desconsiderado, seu peso é definido como 0.Usa-se a notação $\delta^-(v)$ que representam o conjunto de arcos que entram em v e $\delta^+(v)$ representa o conjunto de arcos que saem de v. Tem-se então a seguinte função objetivo, dado que tem-se que pegar

no máximo n vértices, seguida de suas restrições:

$$max \sum_{ij \in A} p_j \cdot x_{ij} \cdot w_{ij} \tag{2}$$

$$\sum_{e \in A} x_e = n, \tag{3}$$

$$\sum_{e \in \delta^+(r)} f_e = k, \tag{4}$$

$$\sum_{e \in \delta^+(v)} f_e = \sum_{e \in \delta^-(v)} f_e - 1, \quad \text{para todo } v \in V \setminus r, \tag{5}$$

$$f_e \leq k \cdot x_e, \tag{6}$$

$$x_{ij} \leq 1 - x_{ri}, \quad \text{para todo } ij \in \delta^+(i) \text{ e } x_{ri} \in VR, \tag{7}$$

$$\sum_{e \in \delta^-(v)} x_e = 1, \quad \text{para todo } v \in V \setminus r, \tag{8}$$

$$\sum_{e \in \delta^+(r)} x_e = 1, \quad \text{para todo } e \in A, \tag{9}$$

$$x_e \in \{0,1\} \text{ para todo } e, f_e \geq 0 \text{ para todo } e, p_i > 0 \text{ para todo } i \tag{10}$$

$$w_e = \begin{cases} 1, \text{ if } e \in A \\ 0, \text{ if } e \in VR \end{cases} \tag{11}$$

A restrição (3) limita que o número de arcos pegos seja igual à n. A restrição (4) define que o fluxo produzido em r seja igual ao número de vértices (k). A restrição (5) faz com que o fluxo total que entra em um vértice em um vértice que não é a raiz seja um a mais que o fluxo que saí do mesmo vértice. Isto permite representar o ocnsumo de uma unidade de fluxo por vértice que não é a raiz. A restrição (6) define que o fluxo de uma aresta que não faz parte da solução seja 0. A restrição (7) define que se o arco vermelho que entra no vértice faz parte da solução, então não se pode sair nenhuma aresta deste vértice. A restrição (8) define que cada vértice só pode ser pego uma vez. A restrição (9) define que só pode sair uma aresta não vermelha da raiz (que é a escolha do "exilado"). A Restrição (10) define que as variáveis de arcos são binárias, que não existe fluxo negativo e que todos os prêmios são positivos. Por fim, a restrição (11) define que o peso dos arcos vermelhos é 0, pois não se consome um *skill point* para obter arcos vermelhos.

2.3.2 Formulação por cortes

A segunda formulação proposta é uma formulação baseada em cortes[5]. Um corte no grafo é caracterizado por um conjunto de vértices $S \subset V$ e tomando-se então seu complemento $\overline{S} = V \setminus S$, existem então dois cortes direcionados: $\delta^+(S) = \{(i,j)|i \in S, j \in \overline{S}\}$ e $\delta^-(S) = \{(i,j)|i \in \overline{S}, j \in S\}$. Também se define $x(A') = \sum_{e \in A'} x_e$ para qualquer subconjunto de arcos $A' \subset A$, além de usar a mesma notação para arestas individuais, tem-se então as restrições e função objetivo desta formulação abaixo.

$$max \sum_{ij \in A} p_j \cdot x_{ij} \tag{12}$$

$$\sum_{e \in A} x_e = n, \tag{13}$$

$$x(\delta^-(S)) \geq \sum_{e \in \delta^-(j)} x_e, \quad \text{para todo } j \in S, r \notin S, \text{ para todo } S \subset V, \tag{14}$$

$$\sum_{e \in \delta^-(v)} x_e \leq 1, \quad \text{para todo } v \in V \setminus T, \tag{15}$$

$$\sum_{e \in \delta^-(t)} x_e = 1, \quad \text{para todo } t \in T, \tag{16}$$

$$\sum_{e \in \delta^+(r)} x_e = 1, \tag{17}$$

A restrição (13) limita que o número de arcos pegos seja igual à n. A restrição (14) determina que caso um vértice seja pego, é necessário que haja um arco que ligue o corte à raiz. A restrição (15) determina que todos os vértices terminais só podem ser pegos até uma vez. A restrição (16) determina que todos os terminais precisam ser pegos uma vez. A restrição (17) define que só pode sair uma aresta não vermelha da raiz (que é a escolha do "exilado")

3 Resultados

Todos os experimentos foram realizados utilizando o processador Intel® Core™ i7-10750H CPU @ 2.60GHz x 6, 32GB de RAM DDR4 @ 2666 MHz, *Gurobi* 9.5.1 (licença acadêmica), C++ 11.2.0 e Ubuntu 22.04 64-bit com ambiente gráfico GNOME 41.7.

Foram executadas as duas formulações propostas no ambiente descrito acima, com dados reais da Rede de Habilidades. São realizados os pré-processamentos, responsáveis por transformar a entrada que é um grafo não-direcionado em um grafo direcionado e no caso da formulação por fluxo, adicionar os arcos "vermelhos"à partir da raiz. Esta instância apresenta 1722 vértices e 2015 arcos, dentre estes, optou-se por apenas 19 pontos serem caracterizados como terminais afim de dar uma maior margem de otimização de premiação coletada ao programa, visto que quantos mais vértices fossem caracterizados como obrigatórios, menor seria o espaço de se aumentar a premiação. A premiação de cada um dos

pontos foi definida como um inteiro aleatório no intervalo (0,100]. Tal decisão foi tomada pois existem dezenas de bônus diferentes que podem ser adquiridos no jogo e cada um deles tem um valor diferente com base no objetivo final do personagem de cada jogador, portanto a parte mais dificil de implementar este trabalho no jogo talvez seja definir uma premiação que normalize todos os vértices.

Utilizou-se o limitante de 120 arcos ao máximo tanto para a formulação por fluxo quanto pela formulação por cortes, escolheu-se este número pois é um valor próximo do máximo que pode ser obtido para um personagem e portanto aproxima ainda mais o algoritmo da realidade de se trabalhar na rede de habilidades. A formulação por fluxo foi executada durante 80 minutos para se obter o resultado máximo de premiação 7422.

Abaixo, apresenta-se este resultado mostrando a árvore com premiação associada gerada(Figura 2). A formulação por cortes não foi capaz de gerar resultados dado os mesmos parâmetros iniciais, foi determinado um limite de 12h máximas para o programa chegar a uma solução e ele não foi capaz de obter nenhuma solução possível neste tempo.

4 Análise

Nesta seção, os algoritmos serão testados para diferentes configurações do grafo original, com diferentes números de terminais e terminais diferentes. Estas instâncias foram obtidas embaralhando os valores que indicam se um vértice é terminal ou não. Foram feitas diversas instâncias diferentes, com 1,10,15,20 e 30 terminais além da original que continha 20. Para todas estas instâncias, utilizou-se o número de 120 arcos a serem pegos.

Tem-se a hipótese que o algoritmo será mais rápido com um número baixo de terminais e com um número grande pois este problema é um híbrido, um deles é encontrar o menor caminho enquanto o outro é maximizar a coleta dos prêmios. Quanto menor o número de terminais, mais o problema se assemelha à um algoritmo que apenas se preocupa com coletar a maior quantidade de premiação associada. Enquanto isto, quanto maior o número de terminais, o espaço para otimizar a coleta de prêmios se torna menor, visto que este grafo não possui um número grande de arcos, dado seu número de vértices.

Tal afirmação se torna evidente quando se observa que o digrafo possui em média 1,7 arcos para cada vértice, então usualmente não existem tantos caminhos a serem percorridos dado um vértice. A seguir é apresentado uma tabela que compara os tempos e o resultado da premiação obtida. O nome das instâncias indica o número de terminais (Ex: t1 é a instância com um terminal apenas).

Ao se analisar a tabela 1, pode-se notar que o comportamento previsto anteriormente é evidenciado, em que as instâncias com um pequeno número de terminais obtiveram somas de premiações maiores em um tempo baixo enquanto as instâncias intermediárias demoraram muito devido ao ser necessário um equilibrio entre a maximização de premiação e a busca dos nós terminais. Inicialmente, o planejado seria realizar com instâncias com um número de terminais maiores, porém, após 30 terminais, mesmo tentando dezenas de vezes, não foi possível obter um grafo com nós terminais em que se fosse possível alcançá-los com 120 arestas a serem pegas, isto se dá pois o grafo possui poucos arcos em relação aos seus nós,

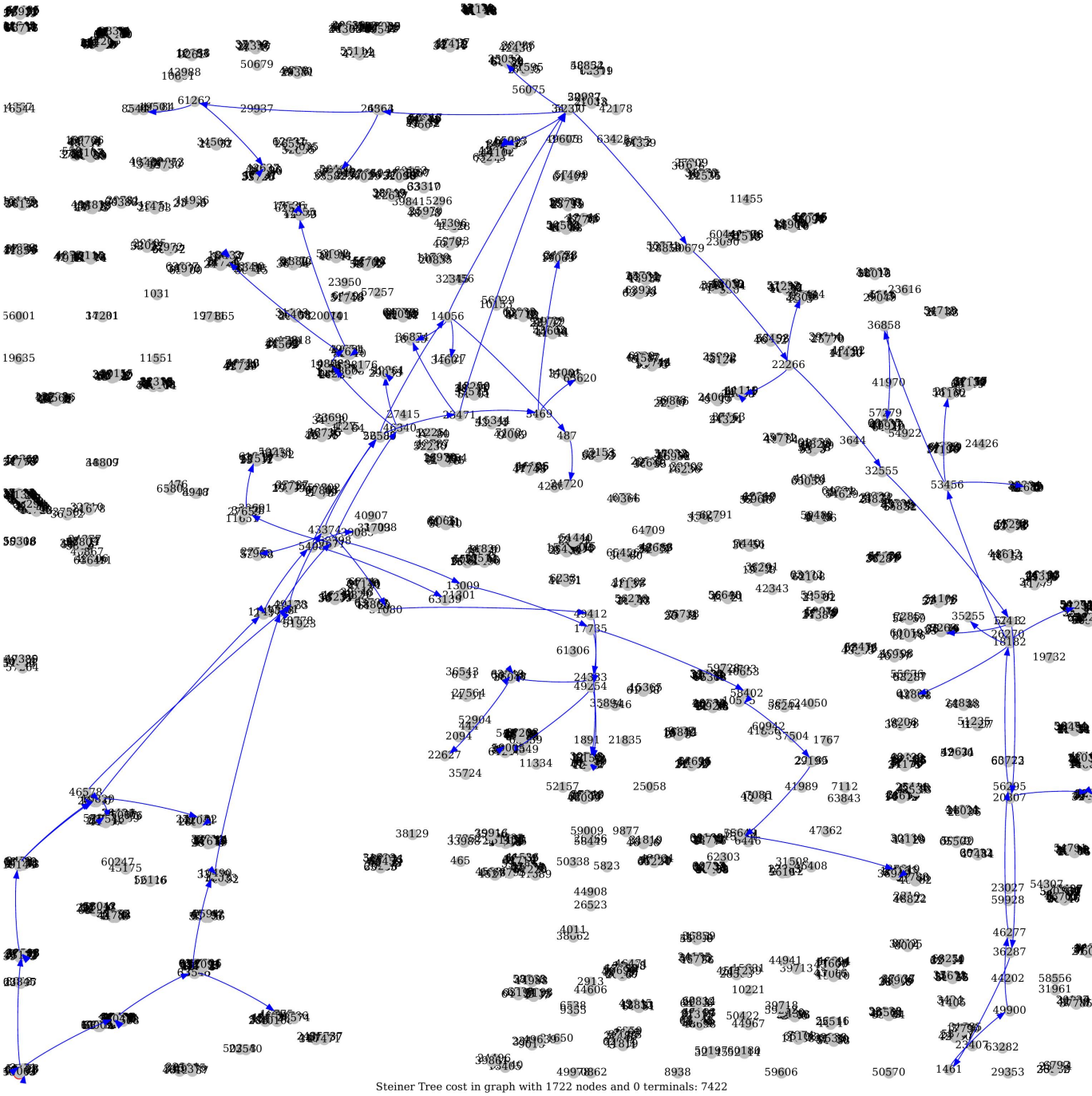

Figura 3: Solução por fluxo sem arestas vermelhas à mostra

Instância	Tempo de execução(s)	Soma de premiação
t1	355	9245
t10	325	8777
t15	4161	7713
t20	4800	7420
t30	1299	6793

Tabela 1: Tabela que compara o número de terminais com o tempo de execução e a premiação obtida

o que faz com que seja mais dificil de se obter um grafo possível com grande número de terminais, visto que estes podem facilmente estarem isolados em locais distintos com poucos caminho possíveis a partir da raiz, sendo estes possíveis provavelmente muito longos.

5 Conclusão

Inicialmente, foram propostas duas formulações deste problema porém, a formulação por cortes que funcionava em instâncias pequenas sem maiores problemas, quando posta sob as instâncias do jogo, não foi capaz de obter uma solução, mesmo ao se esperar um dia inteiro com o programa rodando. Suspeita-se que possam ser dois motivos: O primeiro é que a formulação não foi implementada da maneira correta no programa e o último é que o método por cortes não é a melhor maneira de resolver este problema, visto que é necessário a realização de diversos cortes mínimos para ter certeza que o subgrafo resultante seja conexo e sua grande quantidade de *lazy constraints*, o programa obtêm um peso gigantesco.

Fora isto, é possível evidenciar como este trabalho abre caminho para a comunidade do jogo *Path of Exile* otimizar suas criações de árvores de habilidades, expandindo e usufruindo das informações contidas neste trabalho.

5.1 Trabalhos futuros

Nota-se três principais caminhos a serem seguidos à partir deste ponto, o primeiro é validar/corrigir a implementação da formulação por cortes ou provar sua ineficácia neste problema, se ela se mostrar factível, então o próximo passo é realizar uma comparação entre os dois métodos e analisar qual se mostra melhor, dado cada uma de suas limitações. O outro ponto saí um pouco de teoria de otimização de grafos e resolução de problema NP-difícil que seria encontrar uma maneira padronizar os diferentes bônus de cada nó da árvore em uma numeração matemática, tal padrão mudaria de acordo com o objetivo de cada jogador para com seu personagem, porém isto faria com que este trabalho fosse facilmente aplicado dentro do jogo da maneira como ele está.

Por fim, a última maneira de prosseguir com este trabalho seria estudar melhor a formulação por fluxo e tentar implementar uma maneira que não limitasse o algoritmo a pegar todos os terminais mas que desse um incentivo maior para os terminais serem pegos, como por exemplo uma premiação de grandeza maior do que a dos outros vértices. Tal mudança

possibilitaria o estudo do comportamento do algoritmo na presença de um número maior de terminais.

Referências

[1] Sanjeev Arora and George Karakostas. A $2 + \epsilon$ approximation algorithm for the k-mst problem. *Mathematical Programming*, 107:491–504, 7 2006.

[2] F. K. Miyazawa. Rotinas para resolução de problemas de otimização.

[3] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[4] LEMON - Library for Efficient Modeling and Optimization in Networks. Egerváry Research Group on Combinatorial Optimization (EGRES), 2014.

[5] Ivana Ljubić. Solving steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77:177–204, 3 2021.

[6] Ivana Ljubić, René Weiskircher, Ulrich Pferschy, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical Programming*, 105:427–449, 2 2006.