Vágásgenerálás az egészértékű programozásban gépi tanulással

Becsó Gergely

1. Bevezető

Az egészértékű programozási feladatok megoldása az élet sok területén rendkívül hasznos, de nem ismerünk rá polinomiális algoritmust. Érdekes és hasznos lenne valamilyen gyorsítást elérni a jelenlegi megoldóprogramokon. Ehhez a legjobb jelenleg ismert módszer a Branch&Cut, amelyben van 3 fajta döntési helyzet: 1) el kell dönteni, hogy a Branch fának mely ágát fejtjük tovább, 2) kell Branchelni, azaz elválasztani a feladatot két részfeladatra, itt a kérdés, hogy melyik változó mentén ágazunk el, 3) végül pedig a vágósíkok hozzáadásánál dönteni kell, hogy az adott részfeladatokhoz melyik lehetséges vágást adjuk hozzá.

Az első döntési helyzettel jelenleg nem foglalkozunk. A másodikkal az IPDL kutatócsoport behatóan foglalkozott Gasse és társai munkássága [Gas+19] nyomán. A harmadik döntési helyzettel foglalkozom én az önálló projektem keretében Yunhao Tang és társai [TAF19] nyomán.

2. Megvalósítás

A feladatot a megerősítéses tanulás sémája szerint felírjuk egy egyszemélyes játékként állapotok, akciók és egy jutalomfüggvény segítségével.

- Az állapotok az IP feladatok leírásai $Ax \leq b$, $\min(cx)$,
- az akciók a lehetséges Gomory vágások: $(-A_i^* + \lfloor A_i^* \rfloor)^T x \leq -b_i^* + \lfloor b_i^* \rfloor$ adott állapotban,
- a jutalomfüggvény pedig $r(t+1) = cx^*_{LP(t)} - cx^*_{LP(t+1)}$, ahol $x^*_{LP}(t)$ a t-edik LP feladat optimális megoldása.

Ezt a játékot fogja játszani és gyakorlással megtanulni egy ügynök, amely a valóságban egy neurális hálózat.

Fix méretű adaton dolgozó hálókkal próbálkozni életszerűtlen elképzelés (minden feladatméretre külön háló kellene és a feladatméretek 1×1-től 1000×1000-es nagyságrendig bármi előfordulhat). Tehát csak olyan architektúrák jöhetnek szóba, amelyek tudnak rugalmas méretű adaton is dolgozni. Ezért a hálózatot érdemes az LSTM és az Attention rétegekre alapozni.

2.1. Architektúra

Az attention jól boldogul a változó számú bemenet-kimenettel, ha az egyes bemenő vektorok azonos méretűek. Az azonban nem igaz, hogy minden IP feladatban adott konstans számú vál-

tozó szerepel, így egy LSTM hálózat segítségével elkódoljuk 64 dimenziós vektorokba az adott feltételeket, egy másik, azonos dimenziójú LSTM háló segítségével pedig elkódoljuk a lehetséges vágásokat szintén 64 dimenziós vektorokba. Az így kapott vektorokat átengedjük egy sűrű rétegen, ebből kapjuk a feltételek és a lehetséges vágások végső F_i, V_i beágyazásait.

Végül alkalmazunk egy attention jellegű réteget, ami egy P eloszlást eredményez a vágásokon.

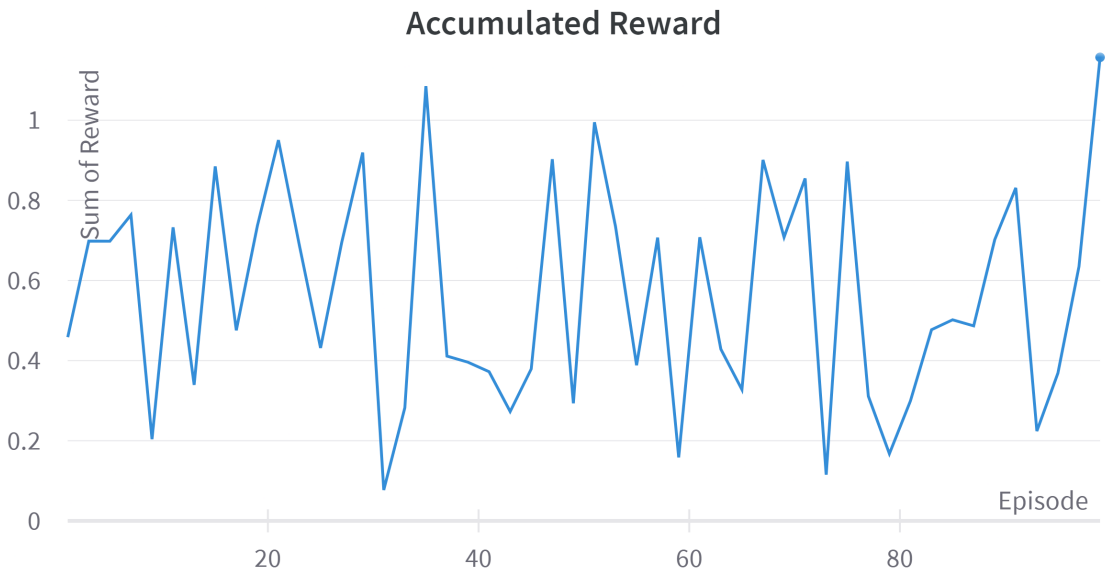
$$P = \text{soft}\max(\sum_{sorok} V \cdot F^T)$$

A tanulás folyamán úgy generáljuk a trajektóriákat, hogy bizonyos eséllyel felfedezünk (*explore*), ekkor P szerint vételezünk a lépések közül, bizonyos eséllyel pedig S mint pontszám által a legígéretesebb lépést tesszük meg (ezt hívjuk *exploit*-nak). Inferencia folyamán mindig *exploit*-álunk.

2.2. Mérések

Az elvégzett mérések azt mutatják, hogy a jelenlegi adathalmazon egy-egy trajektória első vágásai adnak érdemi javítást a célfüggvényen, majd 10^{-13} nagyságrendű javítások következnek.

100 trajektória összegyűjtött javításait alább láthatjuk. A vízszintes tengelyen vannak felsorolva a különböző trajektóriák, a függőleges tengely azt mutatja, hogy az adott feladatnál mennyit sikerült csökkenteni az egészértékűségi résen, azaz mennyi jutalmat gyűjtött a policy.



3. A félév eredményei

Ebben a félévben megértettem a két hivatkozott cikket, megértettem egy a cikkhez tatozó implementációt (GitHub repo). Ezt kis részben átalakítottam, hogy lehessen méréseket folytatni GPU szervereken is, majd reprodukáltam néhány mérésüket.

A cél, hogy egy jó Branch heurisztikával keverve egy hatékony Branch&Cut algoritmust hozzunk létre. Ehhez reményt keltő az a megfigyelés, hogy az algoritmus eleinte gyorsan csökkenti az egészértékűségi rést, majd egy elnyúló farokrész következik, ahol sok lépésen át csak epszilonnyi javítások vannak.

A területen rengeteg további kutatnivaló akad, hiszen a háló-architektúrával érdemes lehet kísérletezni, a Branch algoritmussal való ötvözés érdekes feladatnak ígérkezik és a különböző RL sémák között is rengeteg a felfedeznivaló.

Hivatkozások

[Gas+19] Maxime Gasse és tsai. *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*. 2019. DOI: 10.48550/ARXIV.1906.01629. URL: https://arxiv.org/abs/1906.01629.

[TAF19] Yunhao Tang, Shipra Agrawal és Yuri Faenza. *Reinforcement Learning for Integer Programming: Learning to Cut*. 2019. DOI: 10.48550/ARXIV.1906.04859. URL: https://arxiv.org/abs/1906.04859.