AI Planning Project Report

Hosein Hassani 98301239

Mohammad Amin Banayeean Zade 98208835

۱) مقدمه

روش fast forward (FF) که پیش از این در درس با آن آشنا شدیم، به عنوان یکی از الگوریتم‌های موفق در زمینه برنامه‌ریزی شناخته می‌شود. این برنامه‌ریز توانست در مسابقات AIPS-2000 با اختلاف زیادی از سایر رقبای خود پیشی بگیرد و از این طریق به شهرت زیادی دست یافت[1]. این برنامه‌ریز مبتنی بر روش جستجوی فضای حالت و به صورت جلورو عمل می‌کند اما آن چیزی که این برنامه ریز را متفاوت ساخته است، استفاده از تابع ابتکاری مبتنی بر Graph Plan ریلکس شده است. این تابع ابتکاری به منظور هرس کردن و اولویت دهی به نودهای درخت جستجو استفاده می‌شود. در واقع این برنامه ریز با ارائه یک تابع ابتکاری جدید و همچنین هرس کردن درخت جستجو با کمک مفهوم helpful actions، توانسته بهبود بسیار زیادی ایجاد نماید.

ما در این پروژه قصد داریم تا این برنامه‌ریز را پیاده‌سازی نموده و ضمن تحلیل کارایی آن، سعی در بهبود این روش با ارئه‌های ایده‌های جدید خواهیم داشت. برای این منظور در بخش اول جزئیات ساختار پیاده‌سازی شده با کمک زبان برنامه نویسی پایتون توضیح داده می‌شود. سپس به توضیح برنامه‌ریزهای پیاده‌سازی شده و روش‌های جدید پیشنهادی می‌پردازیم. ما از دو شیوه نمایشی مناسب به منظور ارائه خروجی برنامه‌ریز بهره می‌بریم (نمایش متنی و تصویری). در قسمت دوم گزارش، ضمن معرفی دقیق‌تر این دو نحوه نمایش، نتایج اجرای ورژن‌های مختلف الگوریتم FF روی بنچمارک‌های طراحی شده را بیان می‌کنیم. همچنین به منظور انجام یک ارزیابی مناسب، دو برنامه‌ریز جستجوی جلورو و عقب رو را پیاده‌سازی کرده و با نتایج حاصل از FF مقایسه می‌کنیم. ما برای مقایسه میان برنامه‌ریزها، از دو معیار «طول برنامه به دست آمده» و «مدت زمان اجرای الگوریتم» استفاده می‌کنیم. همچنین توانایی الگوریتم در «یافتن پاسخ مناسب بدون رسیدن به محدودیت‌های اجرایی» از دیگر معیارهای ارائه شده در این بخش است. برای بررسی بیشتر عملکرد برنامه‌ریزها، علاوه بر بنچمارک‌هایی که در اختیار گذاشته شده‌اند، آزمایش‌های جدیدی طراحی شده و روش‌ها روی آن‌ها هم بررسی می‌شوند.

لازم به ذکر است که مشارکت اعضا در انجام این پروژه مطابق جدول زیر است:

وظیفه	مسئول
طراحی ساختار و پیاده‌سازی چارچوب‌ها	بنائیان زاده
دریافت اطلاعات مسئله از ورودی	حسنی
پیاده‌سازی تابع ابتکاری	بنائیان زاده
ورژن‌های مختلف برنامه ریز FF	حسنی
برنامه‌ریزهای جلورو و عقب رو	بنائیان‌زاده
تحلیل کارایی FF	حسنی
پیاده سازی ماژول گرافیکی	بنائیان زاده
تهیه‌ی دیاگرام‌ها و تصاویر	مشترک
تدوین و نگارش گزارش	مشترک

۲) جزئیات پیاده‌سازی و بررسی چارچوب کد

ما سعی کرده‌ایم تا در انجام این پروژه با کمک گرفتن از اصول برنامه‌نویسی شیء گرا، یک ساختار و چارچوب منظم به منظور حل مسئله برنامه‌ریزی ارائه کنیم. پایبندی به این مسئله به ما اجازه داد تا یک محیط کارا به منظور آزمودن الگوریتم‌های مختلف برنامه‌ریزی را توسعه دهیم. در بستر ارائه شده، عیب‌یابی به شدت ساده است و خوانایی کد بسیار بالا رفته است.

در این قسمت از گزارش، کلاس‌های پیاده سازی شده در کد را به صورت مفصل ذکر می‌کنیم. همچنین توابع پیاده سازی شده در هر بخش و انتظاراتی که از آن توابع می‌رود را ذیل یک جدول ارائه می‌دهیم:

۱- کلاس Proposition:

این کلاس پایه‌ای ترین مفهوم پیاده‌سازی شده در این پروژه می‌باشد. در واقع این کلاس حاوی جزئیات مورد نیاز برای مدل‌سازی یک گزاره در محیط است. از آنجایی که برنامه‌ریز FF یک برنامه‌ریز Ground می‌باشد، ما سعی کرده‌ایم تا بیشتر از فرم نمایش Set-Theoritic مسائل بهره ببریم. لذا همانطور که در قسمت‌های بعدی نیز توضیح می‌دهیم، برای توصیف یک State کافیست مجموعه گزاره‌های صحیح در آن را نگه‌داری نماییم.

در دو جدول زیر، متغیرهای به کار گرفته شده و توابع پیاده‌سازی شده ذیل این کلاس را معرفی می‌کنیم:

متغیرها		
نام متغیر	وظیفه	مثال
name	نشان دهنده نام گزاره	On
vars	نشان دهنده متغیرهای موجود در گزاره	[“a”,”b”]

توابع		
نام تابع	وظیفه	ورودی
substitute	جایگزین کردن متغیرهای گزاره داده شده با متغیرهای دلخواه	نگاشتی شامل نقشه تغییر متغیرهای فعلی به متغیرهای جدید

گفتنی است ما در این پروژه به کرات از ویژگی‌های برجسته زبان پایتون برای بهبود کیفیت استفاده کرده‌ایم. به عنوان مثال، یکی از موارد انجام شده در این پروژه، بازنویسی توابع داخلی __eq__، __repr__ و __hash__ برای کلاس‌های پیاده‌سازی شده می‌باشد. وظیفه این سه تابع به ترتیب عبارت است از: «ایجاد امکان مقایسه برای بررسی تساوی بین دو گزاره»، «ارائه یک شیوه نمایش یکتا برای هر گزاره» و «اعمال یک مبنای hash کردن مناسب به منظور وارد کردن گزاره‌ها به مجموعه».

۲- کلاس Action:

از این کلاس به منظور مدل‌سازی کنش‌های مسئله استفاده شده است. برای این منظور در هر کنش مجموعه پیش‌شرط‌های مثبت و منفی به همراه اثرات مثبت و منفی در نظر گرفته شده است. این مجموعه‌ها هر کدام از مجموعه گزاره‌هایی تشکیل شده‌اند که در بخش قبل توضیح داده شد.

در جدول زیر جزئیات پیاده‌سازی دیده می‌شود:

متغیرها		
نام متغیر	وظیفه	مثال
Name	نشان دهنده نام کنش	Unstack
pre_pos	مجموعه شامل پیش‌شرط‌های مثبت یک کنش	On(a,b) Clear(a) Hand_empty
pre_neg	نشان دهنده پیش‌شرط‌های منفی یک کنش	-
eff_pos	نشان دهنده اثرات مثبت یک کنش	Clear (b)
eff_neg	نشان دهنده اثرات منفی یک کنش	On(a,b) Clear(a) hand-empty
variables	نام متغیرهای به کار رفته در کنش	“a” , “b”

توابع		
نام تابع	وظیفه	ورودی
substitute_and_copy	جایگزین کردن متغیرهای کنش داده شده با متغیرهای دلخواه	نگاشتی شامل نقشه تغییر متغیرهای فعلی به متغیرهای جدید
relax_action	ساده‌سازی اکشن با حذف پیش‌شرط‌ها و اثرات منفی	-
get_vars	استخراج نام متغیرهای به کار رفته در کنش	-
get_short_name	دریافت خلاصه‌ای از نام کنش	-

۳- کلاس State:

حالت‌های موجود در یک مسئله برنامه‌ریزی توسط این کلاس مدل می‌شوند. در واقع یک حالت عبارت است از مجموعه‌ای از گزاره‌ها که درباره محیط اطلاعاتی ارائه می‌کنند. توجه کنید که ما در پیاده‌سازی خود «فرض دنیای بسته» را رعایت کرده‌ایم. لذا گزاره‌هایی که در توصیف State نمی‌آیند، خود به خود نادرست خواهند بود.

جدول راهنمای موارد پیاده سازی شده در ادامه آمده است:

متغیرها		
نام متغیر	وظیفه	مثال
propositions	مجموعه‌ای از گزاره‌هایی که در این state برقرار هستند	On(a,b) Clear(a) Hand_empty On-table(b)

توابع		
نام تابع	وظیفه	ورودی
isGoal	با دریافت توصیف حالت هدف، بررسی می‌کند که state فعلی در تعریف هدف صدق می‌کند یا خیر	یک حالت هدف از جنس Goal
relax_action	ساده‌سازی اکشن با حذف پیش‌شرط‌ها و اثرات منفی	-
get_vars	استخراج نام متغیرهای به کار رفته در حالت	-
isAppliable	با دریافت یک کنش بررسی می‌کند که آیا آن کنش در حالت فعلی قابل اعمال است یا خیر	یک کنش
apply_unified_action	با دریافت یک کنش، آن را روی حالت فعلی اعمال می‌کند و حالت جدید را تولید می‌کند	یک کنش که قابل اعمال کردن در حالت فعلی باشد
get_all_unifications	nification های مختلف یک اکشن که در حالت فعلی قابل اعمال هستند را تولید می‌کند	یک کنش
get_all_possible_actions	با دریافت یک مجموعه lifted از کنش‌ها، تمامی اکشن‌های Groundای که قابل اعمال در حالت فعلی هستند را تولید می‌نماید	یک مجموعه از کنش‌ها

مجدداً متذکر می‌شویم که اکشن‌ها به صورت lifted نگه داری می‌شوند اما در حین ساخت plan، تمامی unificationهای قابل اعمال بر روی یک حالت خاص در همان لحظه استخراج می‌شود و تمامی نسخه‌های Ground کنش‌های قابل اعمال بازگردانی می‌شود.

۴- کلاس Goal:

این کلاس مدل کننده یک هدف می‌باشد. طبیعتاً در یک هدف، هم گزاره‌های مثبت و هم گزاره‌های منفی می‌توانند وجود داشته باشند. به منظور پیاده‌سازی الگوریتم جستجوی عقب رو، قسمت‌هایی به تعریف هدف اضافه شده است که در جداول زیر قابل مشاهده است:

متغیرها		
نام متغیر	وظیفه	مثال
propos_pos	این مجموعه دربرگیرنده گزاره‌هایی است که باید در حالت هدف برقرار باشند	On(a,b) Clear(a)
propos_neg	این مجموعه دربرگیرنده گزاره‌هایی است که نباید در حالت هدف برقرار باشند	On-table(b)

توابع		
نام تابع	وظیفه	ورودی
get_vars	استخراج نام متغیرهای به کار رفته در حالت	-
isBackwardAppliable	با دریافت یک کنش بررسی می‌کند که آیا عکس آن کنش بر هدف فعلی قابل اعمال است یا خیر	یک کنش
apply_inverse_unified_action	با دریافت یک کنش، عکس آن را روی هدف فعلی اعمال می‌کند و هدف جدید را تولید می‌کند	یک کنش که عکس آن قابل اعمال کردن در goal فعلی باشد
get_all_backward_unifications	unification های مختلف یک اکشن که در عکس آن‌ها در هدف فعلی قابل اعمال هست را تولید می‌کند	یک کنش
get_all_possible_backward_actions	با دریافت یک مجموعه lifted از کنش‌ها، تمامی اکشن‌های Groundای که عکس آن‌ها قابل اعمال در goal فعلی هستند را تولید می‌نماید	یک مجموعه از کنش‌ها

۵- کلاس Graphlayer:

این کلاس مدل کننده یک لایه گزاره‌ای در الگوریتم Graph plan می‌باشد. توجه کنید که تعریف این کلاس خیلی شبیه به پیاده‌سازی State می‌باشد لذا از آن کلاس ارث بری کرده است. تنها تفاوت جدی Graphlayer با State، وجود گزاره‌های منفی در توصیف آن است. از آن‌جایی که مفهوم مطرح شده در اکثر توابع مربوط به این کلاس با کلاس پدر خود مشترک می‌باشد، از ذکر مجدد آن‌ها خودداری می‌کنیم.

کلاس‌های معرفی شده در بالا، کلاس‌های پایه‌ای هستند که در قلب طراحی ما قرار دارد. به جز این کلاس‌ها، ماژول‌های زیر نیز پیاده‌سازی شده‌اند که آن‌ها را شرح می‌دهیم:

- ماژول GraphPlan : این ماژول حاوی کدهای پیاده‌سازی الگوریتم GraphPlan ریلکس شده می‌باشد. از این ماژول به منظور محاسبه تابع ابتکاری در برنامه‌ریز FF استفاده شده است.
- کلاس plan: این کلاس در واقع مجموعه‌ای از اکشن‌ها را در بر می‌گیرد که در کنار هم یک برنامه خطی را تشکیل می‌دهند. گفتنی است که الگوریتم سرچ جلورو اکشن‌ها را از ابتدا به سمت انتهای پلن تکمیل می‌کند در حالی که در عقب‌رو خلاف این مسئله اتفاق می‌افتد. برای هر دو این حالت‌ها توابع متناظری در این کلاس قرار داده شده است.
- ماژول Graphic: این ماژول یک ابزار نمایشی برای دامنه مکعب‌ها در اختیار ما قرار می‌دهد. توضیحات مفصل‌تر این ماژول و نمونه خروجی آن در بخش‌های بعدی گزارش خواهد آمد.
- ماژول Planner : تمام برنامه‌ریزهای پیاده سازی شده در این پروژه که در بخش بعد از آن‌ها صحبت می‌شود در این فایل قرار داده شده‌اند.
- ماژول Benchmark: این ماژول به منظور ارزیابی و مقایسه پلنرهای پیاده‌سازی شده معرفی می‌شود. از طریق این ماژول هم امکان مقایسه روی Benchmarkهای معروف فراهم شده است و هم از طریق تولید تعدادی مسئله رندم. در قسمت‌های بعدی نتایج این ماژول به صورت مفصل‌تر شرح داده می‌شود.

با توضیحات فوق ساختار کدهای پیاده‌سازی شده تا حد زیادی روشن شد. در قسمت بعد، انواع برنامه‌ریزهای پیاده سازی شده را به صورت مفصل شرح می‌دهیم.

۳) روش‌های پیاده‌سازی شده

۱) برنامه‌ریز جلو رو

این برنامه‌ریز ساده‌ترین الگوریتم جستجوی رو به جلو است که در فصل ۴ کتاب معرفی شده است. ما در این الگوریتم مکانیزم جلوگیری از حلقه را وارد کرده‌ایم تا از بروز حلقه‌های بی‌نهایت جلوگیری کنیم. توجه کنید که در صورت نامحدود بودن زمان و حافظه این الگوریتم کامل است اما از آن‌جایی که چنین فرضی در عمل شدنی نیست، ما یک کران بالا برای عمق برای این الگوریتم در نظر گرفته‌ایم. به عبارت دیگر طول برنامه‌های مجاز در این برنامه‌ریز به این عدد محدود خواهد شد.

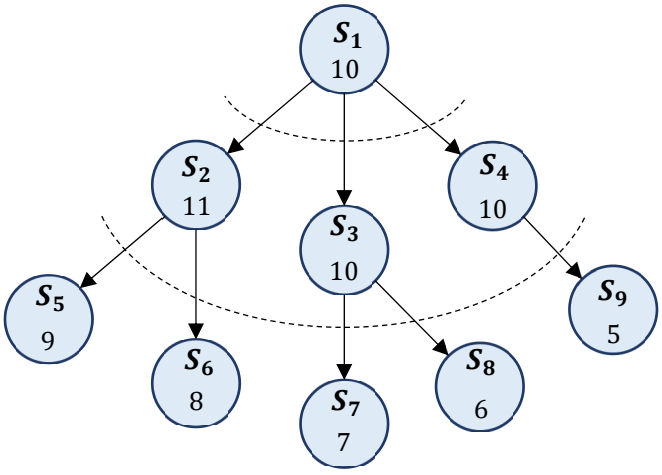
۲) برنامه‌ریز عقب رو

این الگوریتم دقیقاً همان الگوریتم رو به عقبی است که در فصل ۴ کتاب آمده است. مجدداً برای این برنامه‌ریز هم یک کران بالا برای محدود کردن طول plan در نظر گرفته شده است.

۳) برنامه‌ریز FF

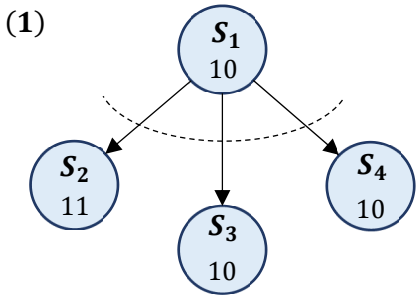
این برنامه‌ریز هدف اصلی این پروژه بوده است و ما به طور ویژه روی پیاده سازی چند نسخه ابتکاری از آن تمرکز کرده‌ایم. در هسته اصلی این برنامه‌ریز یک تابع ابتکاری مبتنی بر گراف‌پلن وجود دارد که آن را بسیار سریع می‌کند. گراف‌پلن ذکر شده از روی اکشن‌های ریلکس شده در مرتبه خطی ساخته می‌شود. سپس در گام بعد با یک بار برگشت از لایه آخر به لایه اول یک برنامه موازی ساخته می‌شود (که البته الزاماً بهینه نیست). از تعداد اکشن‌های موجود در این برنامه به عنوان تخمین هزینه (هیورستیک) و از کنش‌های موجود در سطح اول این برنامه به عنوان مرجع هرس کردن درخت جستجو (یا همان helpful actions) استفاده می‌شود.

تمرکز اصلی ما در این الگوریتم امتحان کردن ورژن‌های مختلف تپه‌نوردی بوده است. لذا برای این برنامه‌ریز چندین ورژن مختلف پیاده سازی شد که در آن سه ورژن naïve_bestchild، modified_enforced، probabilistic_modified_enforced ابتکاری از این گروه بوده است و دو ورژن دیگر از روی مقاله اصلی پیاده‌سازی شده است. برای آن که تفاوت این الگوریتم‌ها به صورت دقیق‌تر مشخص شود، ما از درخت جستجوی نمونه زیر برای نمایش بهتر استفاده می‌کنیم که در هر راس شماره حالت و مقدار هیورستیک آن نوشته شده است. لذا فرض کنید به دنبال جستجو در چنین درختی هستیم، در قسمت پیش‌رو، تک تک الگوریتم‌ها را معرفی کرده و نشان می‌کنیم که هر یک روی این درخت چه رفتاری دارند:



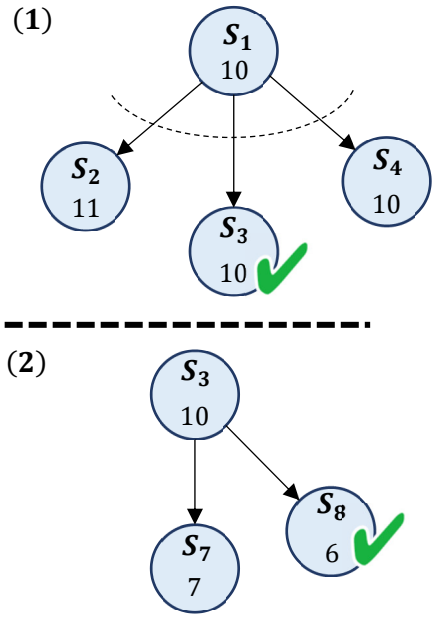
۱) نسخه‌ی naïve_greedy:

در این نسخه طبق خواسته‌ی صورت پروژه از ورژن ساده شده‌ی تپه‌نوردی استفاده شده است. به این صورت که برای جست‌وجو، مقدار هیورستیک صرفا تا یک سطح بعد محاسبه می‌شود و بین کنش‌های helpful اولین کنشی که منجر به رسیدن به مقداری هیورستیک



stuck in local minima!

(A) ***naive greedy***



(B) *naive best child*

کمتر از گره ریشه بشود، به عنوان حالت بعدی در نظر گرفته می‌شود. اگر تمام فرزندان دارای هیورستیک بزرگتر از ریشه باشند روش در مینیمم موضوعی گیر کرده و به جواب نمی‌رسد (مانند شکل زیر).

۲) نسخه‌ی **naïve_bestchild**:

مشکل روش قبل در این است که بعد از گیر کردن در مینیمم موضعی دیگر به جستوجو ادامه نمی‌دهد. برای رفع این مشکل از چندین ایده‌ی ساده استفاده شده است که در عمل کارایی خوبی دارد.

در این روش هم مانند روش قبل جست و جو فقط معطوف به فرزندان گره ریشه بوده و به سطوح پایین‌تر ادامه پیدا نمی‌کند. اما بر خلاف روش قبل، در هر مرحله تمام فرزندان بررسی می‌شوند و بین آن‌ها فرزندی که مقدار هیورستیک آن کمینه است انتخاب می‌شود حتی اگر مقدار هیورستیک آن بیشتر یا مساوی گره ریشه باشد.

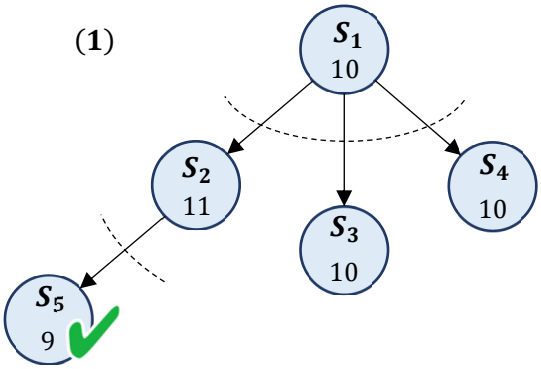
از آن‌جا که ممکن است مقدار هیورستیک حالت بعدی بیشر از حالت قبل باشد لازم است مکانیزم جلوگیری از حلقه اتخاذ شود. برای جلوگیری از حلقه از یک حافظه برای تاریخچه‌ی حالت‌های دیده شده استفاده شد تا از گذر از حالت‌های قبلا دیده شده اجتناب شود.

با در نظر گرفتن این قابلیت بعضا ممکن است حالتی پیش بیاید که تمام کنش‌های helpful منتج به حالت‌های تکراری شوند، در این حالت انتخاب کنش را بین تمام کنش‌های ممکن قرار دادیم تا جستوجو متوقف نشود.

در گراف ذکر شده، عملیات جستوجو در دو مرحله و به صورت زیر خواهد بود:

۳) نسخه‌ی **enforced**:

این ورژن منطبق با نسخه‌ی اصلی الگوریتم است، یعنی جستوجو در یک فلات تا چندین سطح ممکن است ادامه پیدا کند تا به یک حالت دارای هیورستیک **کمتر** نسبت به حالت ریشه برسیم. یک مرحله از عملیات جستوجو در گراف ذکر شده به این صورت است که با جستوجو در سطح اول هیچ کاندیدایی برای جایگزینی یافت نمی‌شود ولی در سطح دوم به حالت‌های دارای هیورستیک کمتر دسترسی پیدا می‌کند:



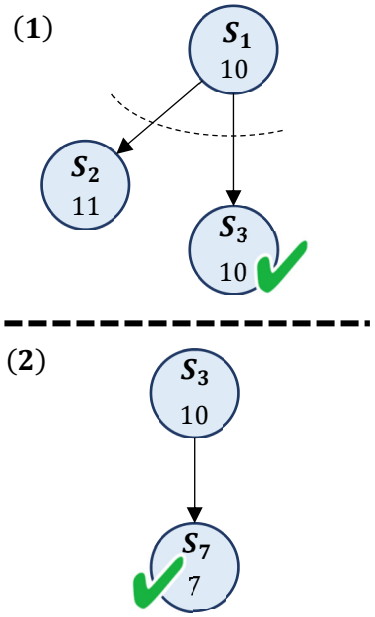
(C) enforced

۴) نسخه‌ی **modified_enforced**:

نسخه‌ی اصلی الگوریتم در بعضی حالت‌ها (مانند محیط large-a) به فلات بسیار گسترده برمی‌خورد که تا مدت‌ها در آن جست‌وجوی BFS را ادامه می‌دهد. در نسخه‌ی modified_enforced تغییراتی داده شده که ممکن است در بعضی محیط‌ها نسبت به نسخه‌ی اصلی مقاله بسیار سریع‌تر جست‌وجو را به اتمام برساند.

ایده‌ی اصلی به این صورت است که در جست‌وجوی BFS اگر یک حالت با هیورستیک برابر با حالت ریشه هم مشاهده شد، در صورت تکراری نبودن انتخاب می‌شود. این ویژگی ممکن است باعث رفتن به حالتی شود که فلات کوچک‌تری دارد و همان‌طور که در عمل نشان داده می‌شود باعث سریع‌تر شدن جست‌وجو در برخی محیط ها (مانند large-a) می‌شود. از طرف دیگر ممکن است تعداد اکشن‌های پلن نهایی کمی طولانی‌تر بشود.

در گراف ذکر شده مشاهده می‌کنیم این نسخه در همان سطح اول هم به S_3 که دارای هیورستیک یکسان با ریشه است بر می‌خورد و دیگر نیازی نیست جست‌وجو را ادامه داده و یا به سطح بعد برود. در گراف ذکر شده دو مرحله از جست‌وجو به صورت زیر است:



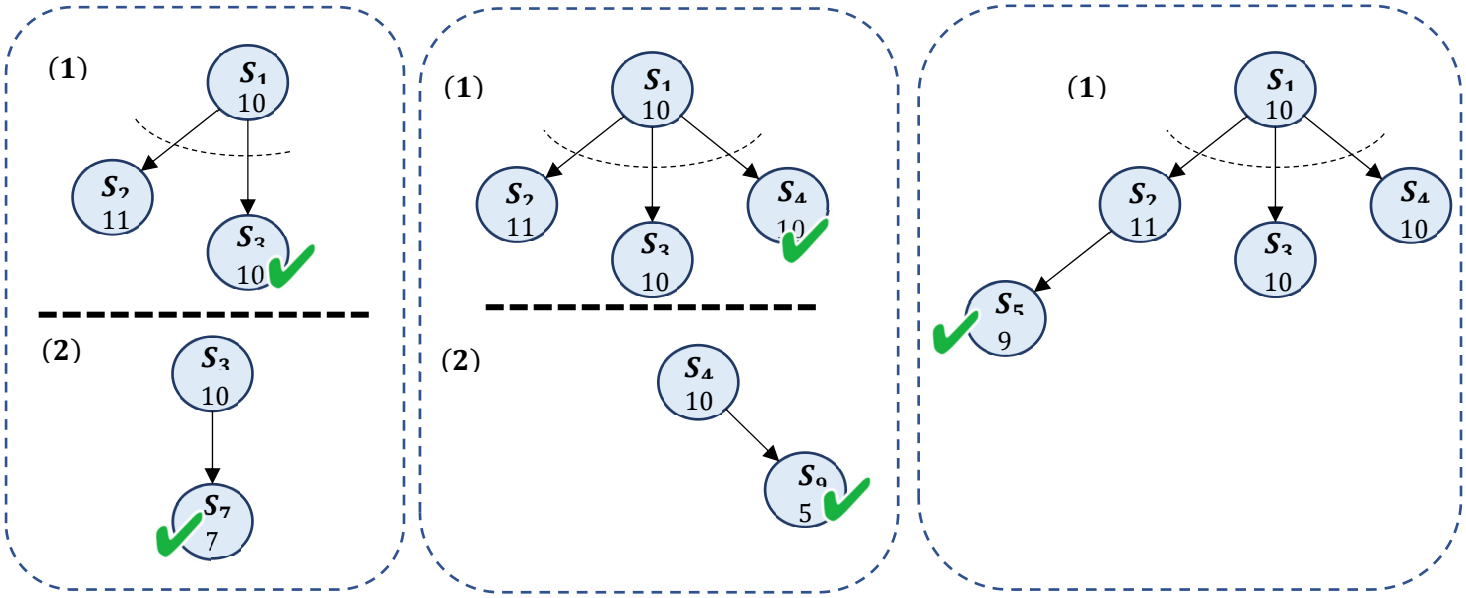
(D) modified enforced

۵) نسخه‌ی **probabilistic_modified_enforced**:

دو نسخه‌ی قبلی هر کدام مزایا و معایبی نسبت به هم دارند. نسخه‌ی modified_enforced ممکن است باعث سریع شدن جست‌وجوی نسخه enforced در برخی مسائل شود. ولی در برخی مسائل مانند (twelve-step) ممکن است به یک حالتی (دارای هیورستیک ۶) برسد و تعداد بسیار زیادی حالت غیرتکراری با هیورستیک یکسان با حالت ریشه را انتخاب کند.

برای برطرف کردن مشکل بالا، می‌توان انتخاب حالت‌های دارای هیورستیک یکسان با ریشه را به صورت احتمالاتی ($\epsilon - greedy$) انجام داد. در این روش سعی شده است مزایای دو روش قبل ترکیب شود. نتایج آزمایش‌ها نشان‌دهنده‌ی موفق بودن این نسخه در عمل است.

در شکل زیر نحوه‌ی عملکرد این روش روی گراف ذکر شده نشان داده شده است. اگر احتمال انتخاب حالت دارای هیورستیک یکسان با ریشه p در نظر گرفته باشد، در جست‌وجو هر یک از سناریوهای زیر با احتمال نوشته شده ممکن است پیش بیاید:



$prob = p$ $prob = p(1-p)$ $prob = (1-p)(1-p)$

(E)random enforced

حال که برنامه‌ریزهای مختلف را معرفی کردیم، در قسمت بعد نحوه اجرای برنامه‌ریز روی یک ورودی دلخواه را بررسی می‌کنیم.

۴) اجرای برنامه‌ریزی برای یک مسئله دلخواه

برای آن که بتوان روی یک مسئله دلخواه برنامه‌ریزی را انجام داد، کافیست مراحل زیر طی شوند:

- هر دو فایل دامنه و توصیف مسئله را در پوشه‌ای در مجاورت مجموعه کدها قرار دهید. (با پسوند .txt)
- وارد کد main.py شوید و آدرس‌های نوشته شده در ابتدای فایل را طوری تغییر دهید که با آدرس مد نظر یکسان شود.

```
parent_path = 'blocks-world/'
domain_file_name = 'domain.txt'
problem_file_name = 'reversal4.txt'
```

- حال کافیست به سادگی فایل پایتون main.py را اجرا کنید:

```
C:\Users\Amin\Documents\AIPlanning>python main.py
```

برنامه‌ریز پیشفرض روش enforced است.

در صورتی که قدم‌های فوق به درستی برداشته شوند، سه نوع خروجی به دست می‌آیند:

۱) خروجی متنی:

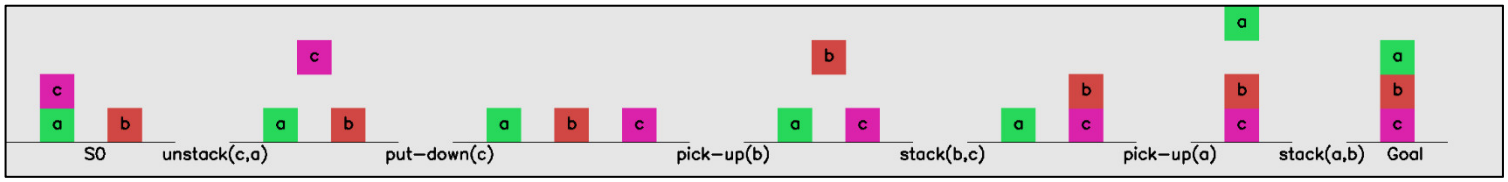
در صورتی متغیر standard_print با False مقداردهی شود و اجرای برنامه‌ریز با موفقیت همراه باشد، یک خروجی متنی در کنسول چاپ می‌شود که نشان دهنده کنش‌هایی است که ما را از حالت شروع به حالت پایان می‌رساند. نمونه‌ای از این خروجی را برای مسئله Sussman در شکل زیر مشاهده می‌کنید:

```
Initial State: on(c,a),clear(c),clear(b),on-table(b),on-table(a),arm-empty()
GOAL: on(a,b),on(b,c),~


elapsed time (s): 0.1669 , length of plan: 6
 ====================  final plan with FF search: ============================
0: unstack(c,a)
1: put-down(c)
2: pick-up(b)
3: stack(b,c)
4: pick-up(a)
5: stack(a,b)
 ============================================================================
```

۲) خروجی گرافیکی:

ما به منظور افزایش خوانایی و ساده‌سازی دیباگ کردن پروژه، یک ابزار گرافیکی قدرت توسعه دادیم. این ابزار گرافیکی مختص مسئله بلاک‌ها طراحی شده است و با دریافت حالت شروع و پلن استخراج شده یک نمایش گرافیکی از وضعیت‌های میانی تولید می‌کند. یک نمونه از موارد تولید شده با این روش را در شکل زیر مشاهده می‌کنید. (برای مشاهده موارد بیشتر با وضوح بالاتر می‌توانید به پوشه Result مراجعه کنید)



۳) خروجی فایل با فرمت استاندارد : در صورتی که standard_print با True مقداردهی شود خروجی به صورت ساده و منطبق با خواسته‌های صورت پروژه چاپ می‌شود.

```
0: (unstack a c)
1: (put-down c)
2: (pick-up b)
3: (stack b c)
4: (pick-up a)
5: (stack b a)
```

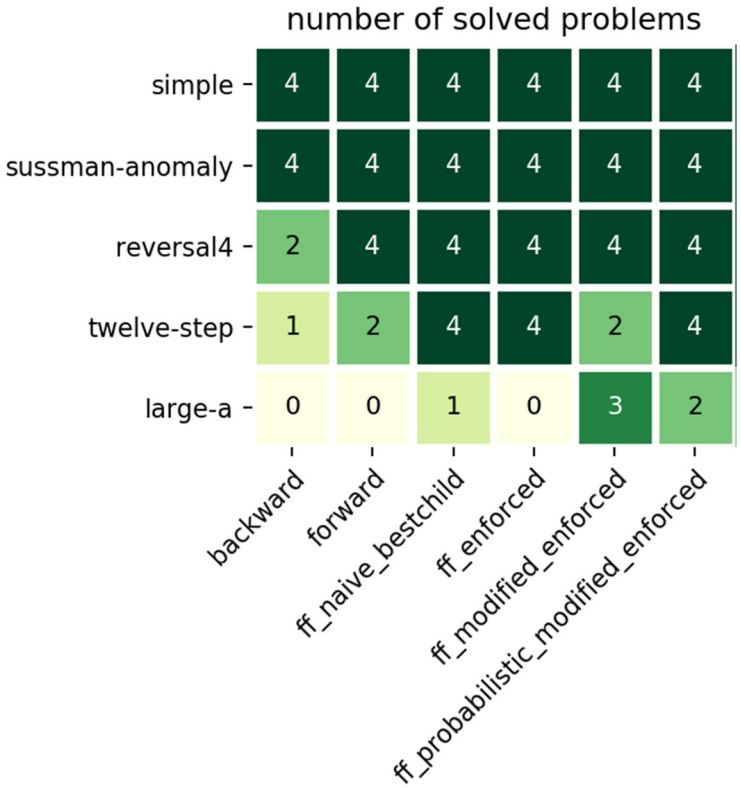
۵) ارزیابی کارایی برنامه‌ریزهای پیاده‌سازی شده

در این بخش به ارزیابی برنامه‌ریزها و مقایسه آن‌ها خواهیم پرداخت. برای انجام ارزیابی، ما از دو مجموعه مسائل کمک می‌گیریم: اول مجموعه مسائل استانداردی که از طرف تیم تدریس به همراه پروژه داده شده است و دوم مجموعه مسائلی که ما به صورت رندم تولید می‌کنیم:

۱) مقایسه برنامه‌ریزها روی مسائل استاندارد:

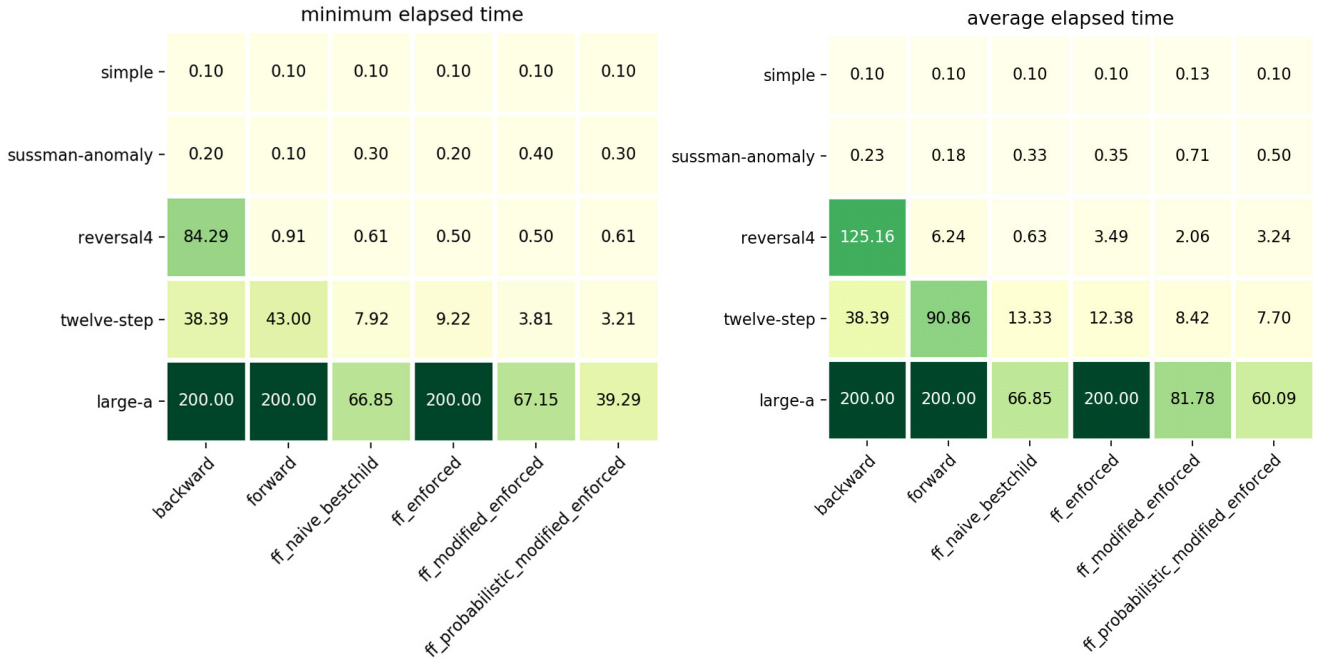
در فایل‌های تحویل شده از طرف تیم تدریس، توصیف ۵ مسئله استاندارد از دامنه مکعب‌ها قرار دارند. ما روی این ۵ مسئله، برنامه‌ریزهای خود را آزمایش می‌کنیم و در نمودارهای زیر مقایسه می‌کنیم. برای آزمایش هر برنامه‌ریز روی هر مسئله حداکثر زمان ۲۰۰ ثانیه برای حل مسئله در نظر گرفته شده است. همچنین به دلیل تصادفی بودن روش‌ها هر آزمایش هم ۴ بار تکرار استفاده شده است تا به طور مناسب‌تری بتوان آن‌ها را قضاوت کرد. توجه کنید که در آزمایش‌های فوق، یک محدودیت روی عمق درخت جست و جو و یک محدودیت زمانی روی مدت زمان اجرای الگوریتم در نظر گرفته شده است. بنابراین منطقی به نظر می‌رسد که برخی از الگوریتم‌ها با این محدودیت‌ها نتوانند به جواب برسند.

در شکل زیر تعداد دفعاتی که برنامه‌ریزها توانسته‌اند مسئله را در زمان ۲۰۰ ثانیه حل کنند آورده شده است (برنامه‌ریز naïve_greedy فقط موفق به حل مسئله‌ی simple بوده و در بقیه مسائل در کمینه‌ی موضعی گیر می‌کرد. لذا از آوردن آن در نتایج اجتناب شده است).



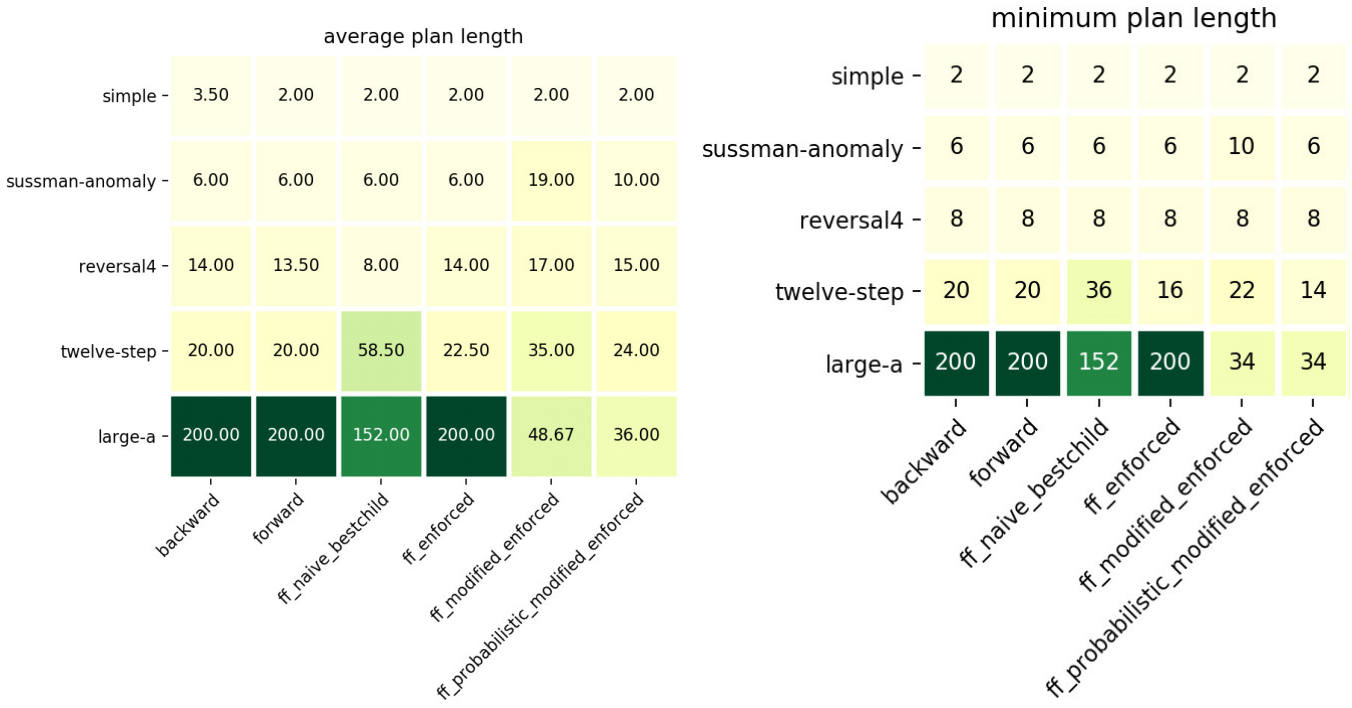
همان‌طور که مشاهده می‌شود نسخه‌های مختلف برنامه‌ریزهای FF در مسائل پیچیده‌تر بر روش‌های forward و backward برتری نسبی دارند. همچنین در بین نسخه‌های مختلف FF مشاهده می‌کنیم که نسخه اصلی مقاله موفق به حل هیچ مسئله‌ای در زمان ۲۰۰ ثانیه نشده است ولی حتی روش naïve_bestchild که جست‌وجوی آن تا یک سطح است توانسته است یک مسئله large-a را در زمان مناسب حل کند. همچنین روش modified_enforced نسبت به روش اصلی در حل مسئله‌ی large-a موفق‌تر است. در نهایت به نظر می‌رسد روش probabilistic_modified_enforced در مجموع بهتر از سایر روش‌هاست.

در شکل زیر مقدار کمینه و متوسط زمان مصرف شده در ۴ تکرار هر آزمایش نشان داده شده است (مقدار ۲۰۰ نشان دهنده‌ی عدم موفقیت حل مسئله در ۲۰۰ ثانیه و در ۴ تکرار مختلف است.)



همان‌طور که مشاهده می‌شود با توجه به هزینه‌بر بودن نحوه‌ی جست‌وجوی نسخه‌ی اصلی FF (enforced) در فلات‌های بزرگ، نسخه‌های ابتکاری ارائه شده توسط این گروه در مسائل سخت‌تر بسیار سریع‌تر هستند.

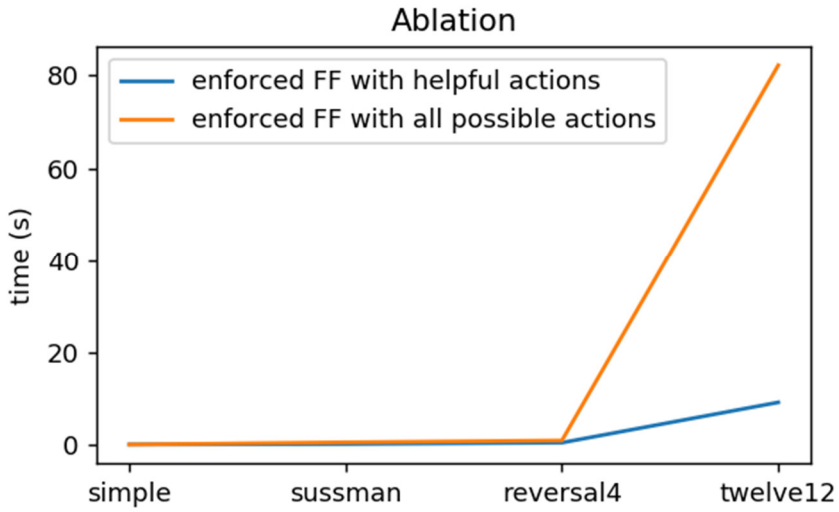
مقدار کمینه و متوسط برنامه‌ی نهایی به دست آمده در ۴ تکرار هر آزمایش نشان داده شده است (مقدار ۲۰۰ نشان دهنده‌ی عدم موفقیت حل مسئله در ۲۰۰ ثانیه و در ۴ تکرار مختلف است.)



همان‌طور که مشاهده می‌شود، هر چند نسخه‌های ابتکاری ارائه شده نسبت به روش اصلی fast-forward سریع‌تر هستند، به دلیل relaxed تر بودن جست‌وجو معمولا طول برنامه‌ی نهایی بهینه نیست و کنش‌های اضافی دارد. در بین این نسخه‌ها هم naïve_bestchild در دو مسئله‌ی دشوار برنامه‌ی طولانی تر به دست آورده است.

در مجموع با در نظر گرفتن مدت زمان جست‌وجو و طول برنامه‌ی نهایی و همچنین تعداد موفقیت‌ها، به نظر می‌رسد که نسخه‌ی probabilistic_modified_enforced در این مسائل موفق‌ترین روش است.

در نسخه‌های جدید ارائه شده از روش FF بیشتر تغییرات رو نحوه انتخاب حالت‌ها بر مبنای هیورستیک بوده که مشاهده کردیم می‌توان به روش‌های بهتری از الگوریتم اصلی FF دست یافت. برای بررسی بیشتر مناسب بودن ایده‌های ارائه شده در FF، با تغییر الگوریتم به این صورت که به جای کنش‌های مناسب (helpful) بین تمام کنش‌های ممکن انتخاب کند. همانطور که مشاهده می‌شود زمان جست‌وجو در مسائل دشوار بسیار زیاد می‌شود. حال آنکه گام برنامه‌های نهایی تقریبا یکسان است.

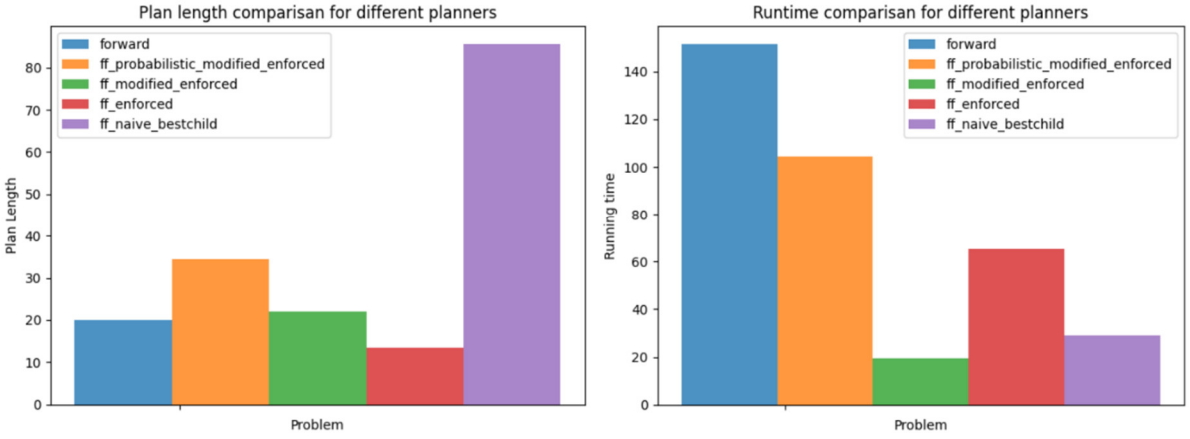


۲) مقایسه برنامه‌ریزها روی مسائل تصادفی:

برای ایجاد تنوع بیشتر در آزمایش‌ها، ما سعی کردیم دسته‌ای از مسائل تصادفی را در این بخش ایجاد کنیم و برنامه‌ریزهای خود را روی آن‌ها ارزیابی نماییم.

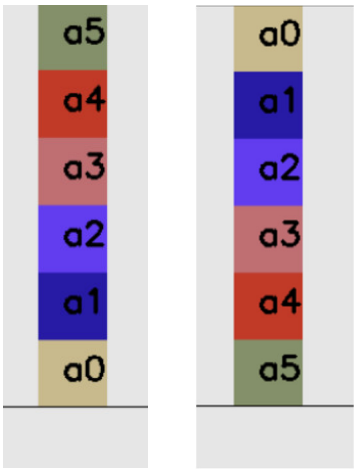
برای تولید مسئله رندم، از یک حالت آغازین از قبل داده شده شروع می‌کنیم و به صورت رندم کنش‌هایی را روی آن اعمال می‌کنیم. سپس بخشی از گزاره‌های به وجود آمده در حالت پایانی را به عنوان هدف انتخاب می‌کنیم و از این طریق یک مسئله برنامه‌ریزی تصادفی تشکیل می‌دهیم. ممکن است دشواری این مسائل به اندازه مسائل واقعی نباشد و یا معیار خیلی مناسبی برای مقایسه‌ی روش‌ها نباشد ولی در نبود بنچمارک‌های استاندارد می‌توان از این روش برای ارزیابی اولیه روش‌ها استفاده نمود.

نتایج این آزمایش به صورت زیر است:

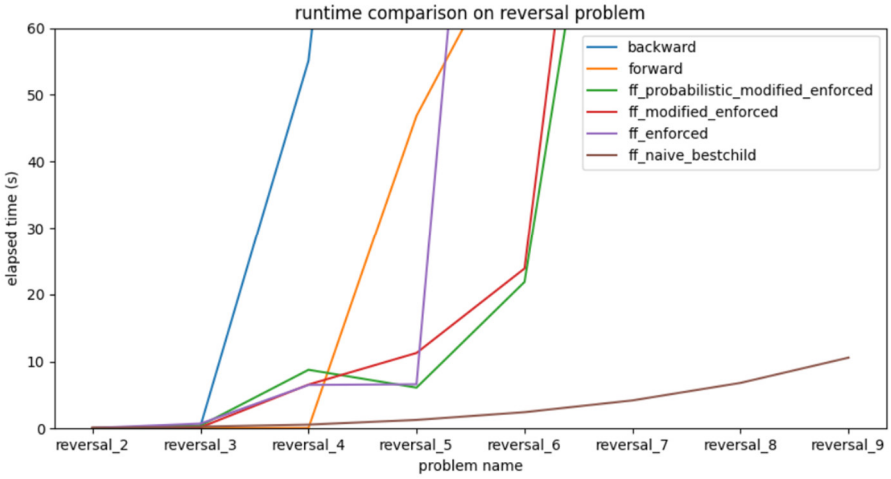


۳) آزمایش روی مسئله reversal n

به منظور انجام یک مقایسه دیگر، ما از مسئله مشهور reversal_n استفاده کردیم. در چنین مسئله‌ای هدف آن است که یک پشته از مکعب‌های داده شده در حالت شروع را کاملاً معکوس کنیم. به عنوان مثال در شکل زیر می‌توانید حالت شروع (سمت چپ) و حالت پایان (سمت راست) یک مسئله reversal_6 را مشاهده کنید:



نمودار مقایسه زمان اجرای الگوریتم‌های مختلف روی مسائل از جنس reversal به صورت زیر است:



چنانچه مشاهده می‌شود در این آزمایش هم روش‌های پیشنهادی عملکرد بهتری از روش اصلی FF دارند. البته در این مسئله‌ی خاص روش forward هم عملکردی قابل مقایسه با روش FF دارد. همچنین نکته قابل توجه دیگر عملکرد بسیار مناسب روش naïve_bestchild نسبت به سایر نسخه‌هاست. احتمالا دلیل آن این است که در این برنامه‌ریز در هر مرحله صرفا یک سطح بررسی می‌شود اما بر خلاف سایر روش‌ها تمام فرزندان ریشه بررسی شده و بهترین آن‌ها انتخاب می‌شود که گویا در این مسئله‌ی خاص این ویژگی موثر واقع شده است.

۶) جمع بندی

در این پروژه به پیاده‌سازی و بررسی برنامه‌ریزهای مبتنی بر FF پرداختیم و عملکرد آن‌ها روی بنچمارک‌های مختلف را با یکدیگر مقایسه کردیم. در پیاده‌سازی کدها از اصول برنامه‌نویسی شی‌گرا کمک گرفته شد تا این پروژه یک بستر مناسب برای پیاده‌سازی انواع و اقسام برنامه‌ریزها و شبیه‌سازی مختلف باشد. پیاده‌سازی‌ها محدود به نسخه‌ی ساده شده (بدون enforced) الگوریتم FF نبود. بلکه الگوریتم اصلی پیاده شد و علاوه بر آن سه روش جدید هم پیشنهاد شد که در بنچمارک‌ها و آزمایشات این پروژه، همگی از برتری نسبی (از نظر سرعت) نسبت به الگوریتم اصلی FF برخوردار بودند.