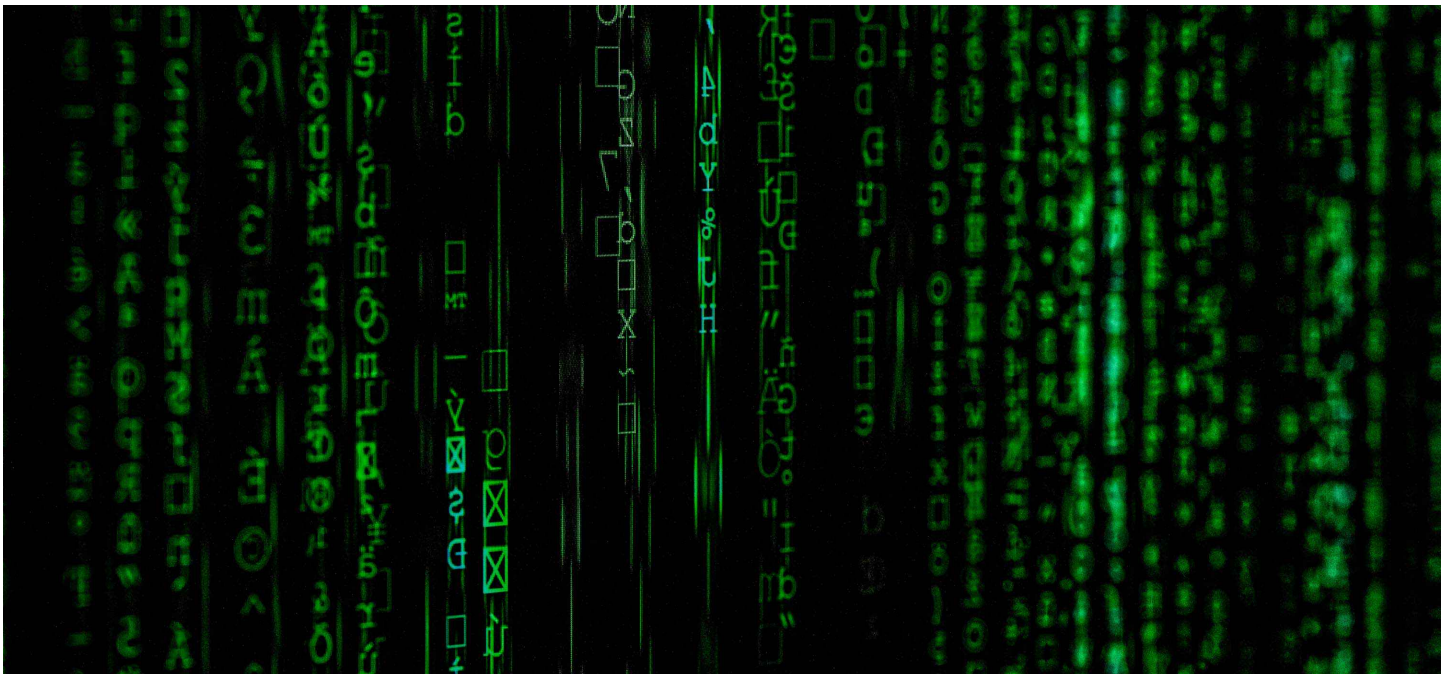


博弈决策算法综述

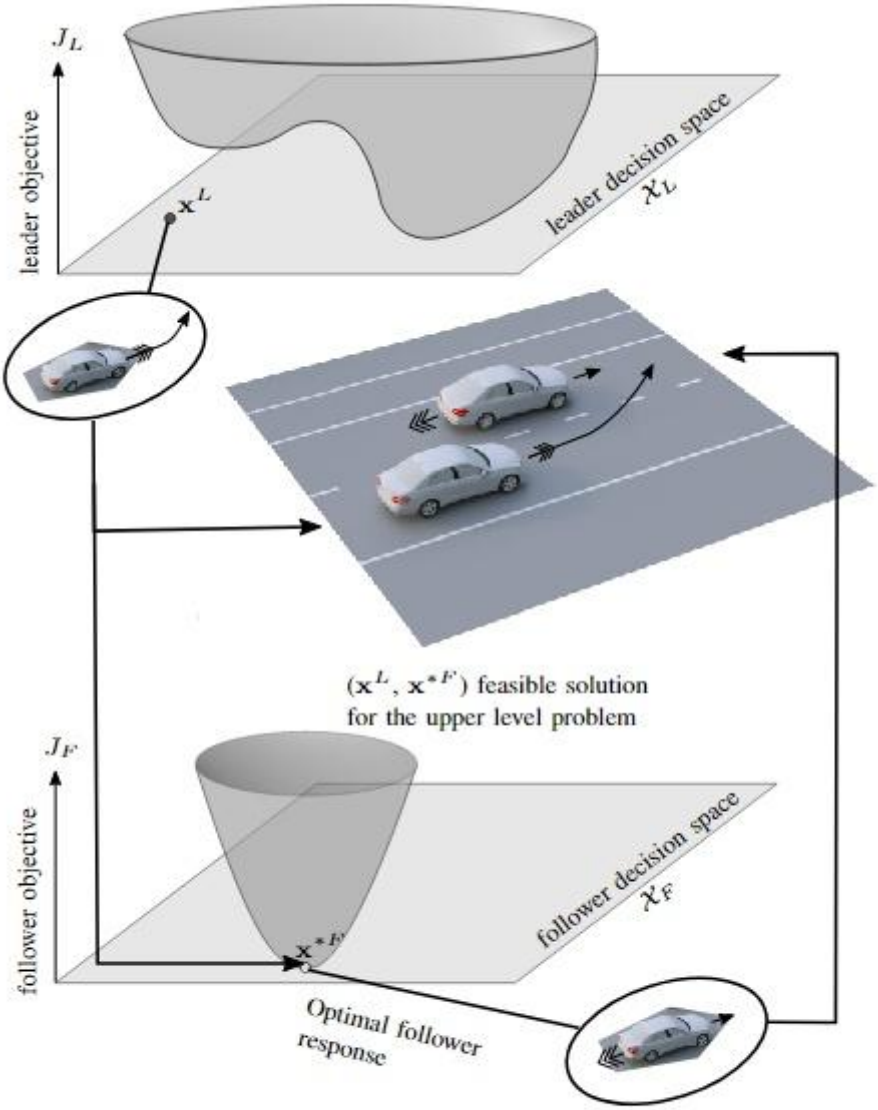
滕昱棠 2023-07-11

1. 宏观博弈和微观博弈

根据对博弈算法考虑的复杂度和颗粒度的区分，研究领域实际上可以区分为两种层面上的博弈算法考虑：

- 宏观博弈： 在较粗的时间和action空间上， 基于博弈理论寻找全局最优的action 决策序列，并根据这个决策序列确定语义决策（是否变道，是否yield）。宏观博弈通常是需要在非凸空间上寻找全局最优。
  - 受限于算力和Memory，通常决策空间的时间跨度都比较粗（2-10s 一步）
  - Action 空间都是高度离散化，剪枝后的 （比如加速度采样0,1,2m/ss 这样非常粗的颗粒度）
  - 需要考虑较多的博弈方(>2个，多的甚至同时考虑30个目标的博弈)
  - 通过采样搜索的方式来求解
- 微观博弈： 在较细的时间步上，基于博弈理论寻找局部的action 决策序列， 这个优化求解问题针对的是给定的凸空间下，针对博弈双方轨迹点的最优化规划，因此微观博弈实际上在做规划，而非决策了。因此其要解决的是轨迹点层面，基于博弈交互的最优。
  - 微观博弈一般都已经是在Motion Planning层面做优化求解了，因此每一步的时间跨度都很小（1s 粗轨迹规划，0.1s 细轨迹规划）

- 微观博弈必须在已经有前序决策给定凸空间的基础上来做优化
- Action 空间就是Motion Planning的控制量的连续空间，不做任何离散化
- 受限于算力和memory ,考虑的博弈方不会超过3个
- 直接使用ILQR，DDP 等优化算法来求解
- 具体的问题建模和求解方法参见 博弈决策算法综述



理想情况下，将整个决策规划分为三层：

第一层做宏观博弈决策，确定交互空间下的最优宏观语义决策

第二层为微观博弈，在给定凸空间上基于博弈均衡理论，计算交互方的粗轨迹

第三层为根据道路环境，舒适性，安全性，规划细致的轨迹

但是，实际上以目前的硬件算力，以及软件复杂度来说，同时考虑到这么细的程度不现实，而微观博弈真正对整个系统性能的提升非常有限，远不如宏观博弈选大方向来的重要。因此这里重点介绍宏观博弈。

2. 博弈模型

博弈模型一般分为合作博弈（cooperative game）和非合作博弈（non-cooperative game）。人类在道路环境中进行交互时，是以一个非合作博弈的方式，处理与周边环境的的交互关系。非合作博弈（Non-Cooperative Game）： 博弈双方只考虑最大化自身的收益，不考虑整体收益的最大化。典型的类型有：纳什均衡，斯塔伯格均衡，帕累托均衡。

非合作博弈根据参与者的行动顺序可以分为静态博弈和动态博弈。

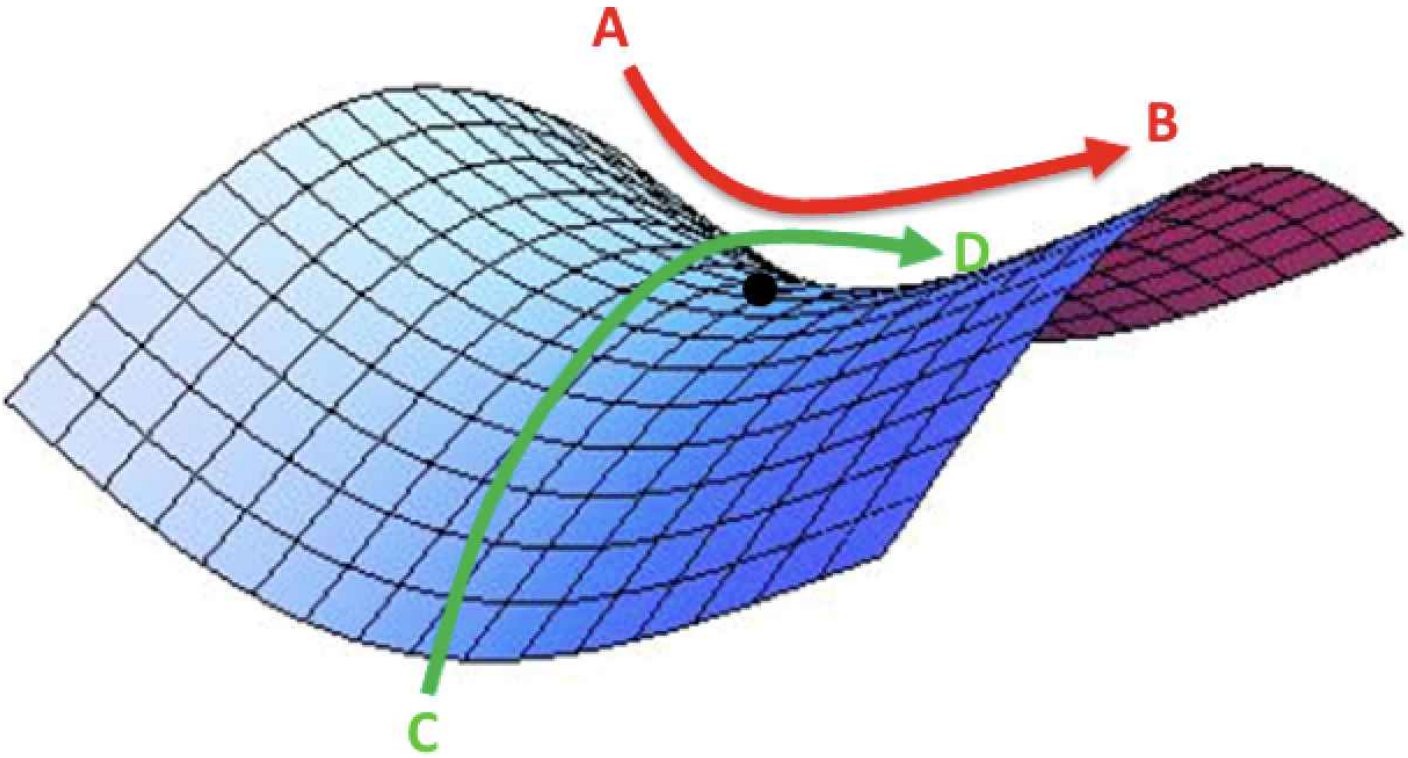
- 静态博弈是指参与者同时选择行动或虽非同时但后行动者并不知前行动者采取了何种行动；（比如手头剪刀布游戏）
- 动态博弈是指参与者的行动有先后顺序且后行动者能够观察到先行动者所选择的行动；（比如棋类游戏）

车辆间的交互建模，即可以使用不断的静态博弈模型建模（比如我们目前使用的加速度采样对的方式）； 也可以使用动态博弈模型，例如斯塔伯格博弈模型。

2.1 纳什均衡

博弈双方达成一种动态的平衡，即任何一方的任何策略变化，都会造成对方的策略响应改变，从而造成自身的收益下降。（典型的如，冷战中的核威慑平衡）。数学上，可以表述为：

$$
\begin{aligned}
\gamma_X^{t,*} &= \arg\min_{\gamma_X^t} \sum_{k=0}^{n-1} c_X(s^t, \gamma_X^{t+k}, \gamma_Y^{t,*}; \theta_X), \\
\gamma_Y^{t,*} &= \arg\min_{\gamma_Y^t} \sum_{k=0}^{n-1} c_Y(s^t, \gamma_X^{t,*}, \gamma_Y^{t+k}; \theta_Y),
\end{aligned}
$$



2.2 斯塔伯格均衡（Leader/Follower Game）

$$\gamma_l^*(\bar{s}_t) \in \underset{\gamma_{l,t}\in\Gamma_l(\bar{s}_t)}{\operatorname{argmax}} Q_l(\bar{s}_t, \gamma_{l,t}), \tag{8}$$

$$\gamma_f^*(\bar{s}_t) \in \underset{\gamma_{f,t}\in\Gamma_f(\bar{s}_t)}{\operatorname{argmax}} Q_f(\bar{s}_t, \gamma_{f,t}), \tag{9}$$

$$Q_l(\bar{s}_t, \gamma_{l,t}) = \min_{\gamma_{f,t}\in\Gamma_f^*(\bar{s}_t)} \mathbf{R}_l(\bar{s}_t, \gamma_{l,t}, \gamma_{f,t}), \tag{10}$$

$$Q_f(\bar{s}_t, \gamma_{f,t}) = \min_{\gamma_{l,t}\in\Gamma_l(\bar{s}_t)} \mathbf{R}_f(\bar{s}_t, \gamma_{l,t}, \gamma_{f,t}), \tag{11}$$

where $\Gamma_f^*(\bar{s}_t) = \{\gamma'_{f,t} \in \Gamma_f(\bar{s}_t) : Q_f(\bar{s}_t, \gamma'_{f,t}) \geq Q_f(\bar{s}_t, \gamma_{f,t}), \forall \gamma_{f,t} \in \Gamma_f(\bar{s}_t)\}$.

$s_t = (s_0^t, s_1^t, s_2^t, ..., s_n^t)$ 为当前t时刻交通情况，即当前涉及博弈的所有目标的位姿状态组合；

$\gamma_l^*(S_t), \gamma_f^*(S_t)$ 分别为leader/follower 在当前时刻的状态下的最优博弈轨迹

$\Gamma_l(S_t), \Gamma_f(S_t)$ 分别为leader/follower 在当前时刻的状态下的可选轨迹空间

$R_l(S_t, \gamma_{l,t}, \gamma_{f,t})$ 为对应$leader$、$follower$ 轨迹对下$leader$ 方的$reward$；

$Q_f(S_t, \gamma_{f,t})$ 为给定一条$follower$ 轨迹，其$worst\ case$ 下的$reward$

$Q_l(S_t, \gamma l, t)$ 为给定一条$leader$ 轨迹，其与预测的$follower$ 可能采用采用轨迹的$worst\ case$ 下的$reward$

如上的公式可以这样理解：

- Follower 角色的一方，在已经确定放弃leader角色下，最大化其在最差情况下的收益（Max-Min策略）；
- Leader角色的一方，预设follower 的一方会避让，因此leader的一方，将先站在follower 一方的角色下，为其预测其将规划的轨迹，并在此基础上，去规划自己的轨迹，去最大化自己的收益；
- 这个假设的前提是，双方已经为***各自的角色达成一致***，且***双方对各自的动作空间都有明确的认识***。follower 一方因为不确定leader一方会采取什么样的动作，而不得已采取更为保守的策略。
- 公式11，可理解为，给定follower的一条轨迹，然后遍历leader的所有可能轨迹，并找到leader轨迹空间中对follower威胁最大（reward最小）的一条轨迹，并以此轨迹对的reward 作为该follower轨迹的worst case reward
- 公式9，可理解为，follower 遍历自己的所有轨迹空间，为自己的每一条轨迹通过公式11，找到对应的worst case reward; 并查找worst case reward 最大的一条轨迹作为自己的follower角色下的最优轨迹；
- 公式10，可以理解为， leader 作为先手方，知道follower 将会采取的策略，因此leader会用follower同样的策略，先预测下follower将会采取的轨迹（即如下公式的意义）

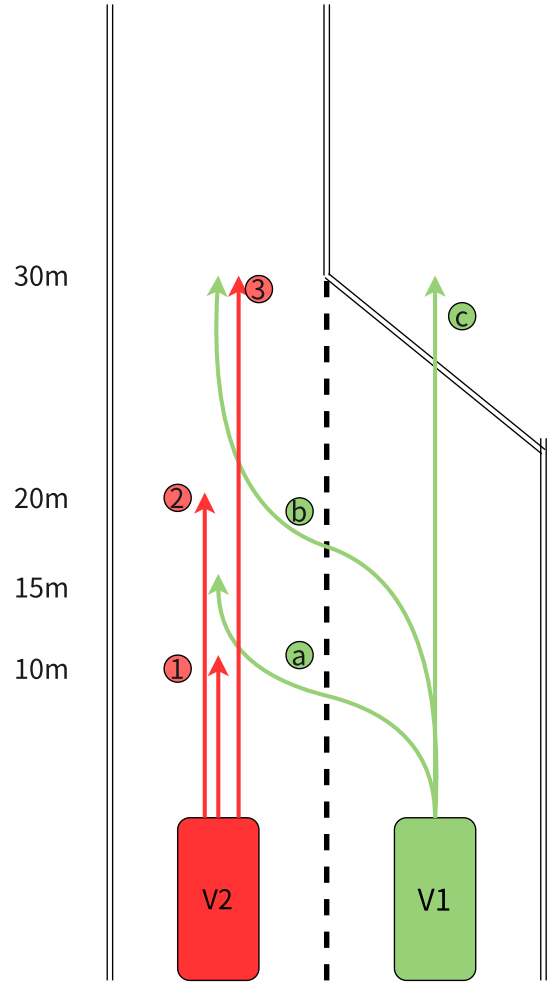
$$\text{where } \Gamma_f^*(\bar{s}_t) = \{\gamma'_{f,t} \in \Gamma_f(\bar{s}_t) : Q_f(\bar{s}_t, \gamma'_{f,t}) \geq Q_f(\bar{s}_t, \gamma_{f,t}), \forall \gamma_{f,t} \in \Gamma_f(\bar{s}_t)\}.$$

假如follower一方可能有多条轨迹均有最大的worst case reward， 则使用公式10，先遍历follower方的这多条轨迹与给定leader 轨迹的 reward，找出造成leader方reward最小的一条follower 轨迹，并以其reward作为给定leader 轨迹的worst case reward.

- 公式11，可以理解为，leader 遍历自己的轨迹空间，先为自己的每一条轨迹计算，在follower 方可能采取的轨迹中，reward最小的情况下的情况（worst case reward）； 通过比较自己每条轨迹的worst case reward，选择拥有最大的worst case reward的轨迹作为leader 的最优轨迹；

2.2.1 Leader follower game 示例

为了帮助大家理解博弈算法，我们构建了一个虚拟的车道汇流的场景，并简化了博弈双方的轨迹空间。并以此作为案例介绍博弈算法如何进行轨迹预测。



	a	b	c
1	-3 / -3.5	2 / 0	-4 / 0
2	-3 / -2.5	1 / 0	-4.5 / 0.5
3	0 / 1.5	-4 / -4	-5 / 1

r1: 产生碰撞为-1*碰撞距离*0.4， 没有碰撞0
r2: 偏离可行驶范围：-4， 没有则为0
r3: 纵向行驶距离*0.1-2m
r4: 离目标车道的距离*-0.5
r5：辆车间距离*0.1-1m

2.2.1.1 case1：如果V1 是leader， V2是follower

我们先预测follower V2会采取的轨迹：

	A	B	C
1	follower选择的轨迹	leader最有威胁轨迹	Q_f
2	1	a	-3.5
3	2	a	-2.5
4	3	b	-4

因此，我们可以预测，该种状态下V2会选择轨迹2作为自己的最优轨迹

其次，我们再在此基础上，估计V1会选择的最优轨迹：

	A	B	C
1	follower选择的轨迹	对应leader可选择的最优轨迹	Q_l
2	2	a	-3
3		b	1
4		c	-4.5

因此，我们可以预测，该种状态下V1会选择轨迹b作为自己的最优轨迹

2.2.1.2 case2: V2 为leader ,V1为follower

我们先预测follower V1会采取的轨迹

	A	B	C
1	follower选择的轨迹	leader最有威胁轨迹	Q_f
2	a	1,2	-3
3	b	3	-4
4	c	3	-5

因此，我们可以预测，该种状态下V1会选择轨迹a作为自己的最优轨迹。

其次，我们再在此基础上，估计V2会选择的最优轨迹：

	A	B	C
1	follower选择的轨迹	对应leader可选择的最优轨迹	Q_l
2	a	1	-3.5
3		2	-2.5
4		3	1.5

因此，我们可以预测，该种状态下V2会选择轨迹3作为自己的最优轨迹.

2.3 帕累托均衡

相比于纳什均衡和斯塔伯格均衡，帕累托均衡更为理想主义，其对优化做了一个约束，一方的收益的改善，不以降低另一方的收益下降为前提。这样相当于对优化做了个约束，即博弈一方在最大化自己收益时，需要确保对方的收益不会下降（或高于一个阈值）。（这个符合习主席的中国特色发展观:)）.

但是由于帕累托均衡实际上成立带约束的优化问题，非常不利于数学上求解。在实际处理时，我们将其简化为一种合作博弈，即将博弈双方的cost 加权后作为总的cost.

$$(\gamma_X^{t,*},\gamma_Y^{t,*}) = \arg\min_{\gamma_X^t,\gamma_Y^t}[\sum_{k=0}^{n-1} c_X(\gamma_X^{t+k},\gamma_Y^{t+k},\theta_X) + \sum_{k=0}^{n-1} c_Y(\gamma_X^{t+k},\gamma_Y^{t+k},\theta_Y)]$$

2.4 Level-K Game Theory

之前的博弈模型，可以看到其决策空间的维度是随着博弈方的数量增加成指数级增加的，所以大部分直接使用博弈模型求解的方法在处理多目标博弈时均只能简化处理，把多目标博弈处理成自车与多个

目标 一对多星型交互博弈方式。这样就牺牲了整个决策空间的闭环性，假定其他多个冲突目标间是不会产生交互的。

2.4.1 level-k的原理

最近几年出现了一种新的博弈建模方式Level-K reasoning 可以较好的解决多目标博弈多对多交互的问题。

- Level-K 的核心处理方式很简单，就是预判对方的预判
- 模型把人的思维归纳成多个思维层次L0,L1,L2...
  - Level 0: 即司机在做决策时不考虑周边agent 的交互，我该怎么开怎么开，默认周边agent会主动来适应我（Reckless driver）
  - Level 1: 司机会假定交互的agent都是Level 0 的司机，而采取对应的最优action 来应对（Coutesy Driver）
  - Level 2: 司机预设周边的agent 都是Level 1的司机，而采取对应的action 来最大化自己的收益（Confidence Driver）
- 通过层叠递推的方式，就将多目标间相互耦合的优化问题，简化为逐层优化的问题，问题复杂度大幅下降

	Level0 reasoning	Level1 Reasoning			Level2 Reasoning			Final decision
V1（ego）	V1-L0	V1-L1	V1-L0	V1-L0	V1-L2	V1-L1	V1-L1	Best decision(
V2	V2-L0	V2-L0	V2-L1	V2-L0	V2-L1	V2-L2	V1-L1	V2-L0
V3	V3-L0	V3-L0	V3-L0	V3-L1	V3-L1	V3-L1	V3-L2	V3-L2)

以上面这个表为例，有三个目标相互之间进行博弈：

1. 在第0层推理时，先对三个目标进行开环推演（仅考虑静态环境，不考虑周边动态障碍物），计算出每个目标Level 0 时的轨迹
2. 在第1层推理时，每个目标都以假定其他agent 使用Level 0 轨迹的前提下，优化求解出自身最佳的Level1 轨迹
3. 在做第2层推理时，每个目标都以假定其他agent 使用Level 1 轨迹的前提下，优化求解出自身最佳的Level2 轨迹
4. 对于N个目标间的博弈，我们仅需要进行N*（N-1）+1次轨迹寻优

由于在每一层寻优自身的最优决策时，假定的是对方是已知的上一层的轨迹，因此本层的优化求解过程就是一个常规的确定性环境下，对推理方轨迹的一次优化求解，因此问题可以大幅简化，通常博弈空间考虑的颗粒度可以考虑的更细。同时，计算量也仅随着博弈方数量的增加仅成线性增加。

可以看到，Level-K reasoning 的推理，类似在多变量优化问题中，先固定其它变量，优化一个变量的处理方式。理论上，如果我们不断一层层往下迭代，而所有的agent 都具有无限层推理的能力的话，最终整个系统将会稳定在一个均衡点，可以证明这是一个纳什均衡点。

但是，大量的统计实验证明，人类的推理能力是有限的，99% 以上的人处于L0，L1，L2 这三层。因此实际上，使用Level-K模型比Nash 均衡更接近实际场景中的博弈。

2.4.1 Ego Level 的选择

那么ego 的Level是越高越好吗?

显然不是，自车应该选择博弈方的Level+1的Level，这才是最佳的策略。

https://www.bilibili.com/video/BV1Eo4y137uk/?buvid=XYD01E1DD72714A980739B22CC0165D8D91C2&is_story_h5=false&mid=%2Fcmbvnci1wBGLRPcRMHSFw%3D%3D&p=1&plat_id=114&share_from=ugc&share_medium=android&share_plat=android&share_session_id=6e01f6a6-1d73-4d81-af19-0a5c80028dc0&share_source=COPY&share_tag=s_i&timestamp=1689090738&unique_k=K0kds5g&up_id=93354997

上面这个场景就是一个典型的Level-K 冲突场景：

- 首次冲撞时，在转角因为双方都不知道对方的存在，因此都按自己既定的路线行进， 因此第一次相撞时，为Level-0 Vs Level-0
- 第二次相撞时，双方都以为对方会继续选之前路线，因此都选择避开对方之前路线的轨迹，然后再次相撞，Level-1 Vs Level-1
- 之后双方都不断在前一次的路线基础上，继续修正估计，结果就出现不断的Level 冲突，Level-2 Vs Level-2 , Level-3 Vs Level-3...

由此可见Level高并不一定能解决冲突，与对方的Level 错开一层才是解决冲突的方式。

多篇论文中均有对应的仿真和测试分析，当自车选择的level 和对方实际的Level 一样或者是对方Level+2 时，双方冲突的概率是较高的。最理想情况就是只比对方高一层。

- 所以Level0 vs Level 0 ,双方都是莽汉，肯定要打架；
- Level0 Vs Level 2 , 说明Level2 的一方想多了，一样会打架。

TABLE III
STATISTICS OF COLLISION AVOIDANCE WITH RESPECT TO LEVELS

Scenario	Collision avoidance stats.
level-0 & level-0	52%
level-0 & level-1	86%
level-0 & level-2	57%
level-1 & level-1	80%
level-1 & level-2	82%
level-2 & level-2	63%

因此Level-K 理论在实际决策中怎么使用呢？ 使用Level-K 算法推算出每个Level 下目标的期望状态，然后根据目标的实际状态去估计目标实际所处的Level，最后ego 的策略在对目标估计的基础上进行最终的优化求解：

- 如果是单目标博弈，或多目标且所处Level相同，直接使用目标Level+1的最佳策略
- 如果是多目标博弈，且目标间Level不同，则根据确定的目标Level， 最后再做一次自车策略最优搜索Final decision(v2_L0,V3_L2)

2.4.2 level-k的使用方法

在具体使用中，我们根据对agent 的理解方式不同，有两种对应处理方式：

- 认为人类司机在同一时间只会处于一个Level 层次上，然后根据对目标实际位置观测，选出一个最符合的Level 作为当前司机所处的Level，并以其对应的Level +1的轨迹作为决策的输出，为后续Planning提供凸空间和粗轨迹；
- 认为人类司机可能处于不同Level 下的混合状态，只是概率分布存在变化，这样就需要根据对目标位置的连续观测，对博弈方所处的L0~L2 的概率分布进行belief update。 最终输出的是多模态的轨迹预测和概率分布，Level 0,Level1, Level 2 的对应轨迹以及对应的概率，后续的规划可以直接使用这些多模态信息进行Contingency Planning。

事实上，这也是交互算法下通用的两种方式，不仅用在Level-K 模式下，也可以用在斯塔伯格均衡leader/follower 角色确定下。具体在第五章详细介绍。

不管是哪种处理方式，我们都需要通过一个观测模型，根据对目标实际状态的连续观测，来估计目标实际所处的Level 或对应处于对应Level 的概率分： 即对level-k进行belief update。一般来说，需要使用到贝叶斯推断，以及对应基础上上的多模态混合估计算法。

📌 Li, Nan, Ilya Kolmanovsky, Anouck Girard, and Yildiray Yildiz. 2018. “Game Theoretic Modeling of Vehicle Interactions at Unsignalized Intersections and Application to Autonomous Vehicle Control.” In *2018 Annual American Control Conference (ACC)*. doi:10.23919/acc.2018.8430842.

Tian, Ran, Sisi Li, Nan Li, Ilya Kolmanovsky, Anouck Girard, and Yildiray Yildiz. 2018. “Adaptive Game-Theoretic Decision Making for Autonomous Vehicle Control at Roundabouts.” *Cornell University - arXiv*, October.

Karimi, Shahab, and Ardalan Vahidi. 2020. “Receding Horizon Motion Planning for Automated Lane Change and Merge Using Monte Carlo Tree Search and Level-K Game Theory.” In *2020 American Control Conference (ACC)*. doi:10.23919/acc45564.2020.9147369.

2.4.5 level-k 中每一层的求解方法

在每一层每一个目标对应的level 状态下的最优action 序列的求解问题就是在一个确定环境下，求解目标的最优序列搜索。

2.4.1.1 使用优化规划算法

如果我们在系统建模时，每个目标的决策，观测，执行模型都是确定性的。 则周边其他车辆的action是可以直接转化为的Level K-1的轨迹的，因此直接且简单的方式就是一个轨迹规划问题（只不过一般是粗轨迹，且目标函数不需要很复杂）。因此直接使用动态规划，或者ILQR等常规的轨迹规划算法就可以直接实现了。

2.4.1.2 使用MCTS 方法

另一种方式则是使用MCTS来对寻优目标的action 序列进行最优搜索。由于Level-K 已经将多目标优化问题降维为单目标优化问题，因此搜索的空间就被压缩为自车的多步action序列了。

2.4.1.3 使用模仿学习模型

虽然，Level-K理论对交互决策的问题规模进行了降维，但是其依然面临了较大的计算压力，特别是当多目标博弈和需要较多step的推演时。

以n 个目标博弈为例，每个目标的Level-0 通过驾驶员模型推演得到，Level-1， Level-2 通过优化搜索得到，则需要进行3*n+1 次的单层优化搜索。而如果每个目标的action 空间为14 个，推演三个step，则每一层的搜素空间为2744. 因此，在大部分域控平台上，Level-K依然面临算力问题。

同时，还有一个最主要的问题是，我们使用驾驶员模型来推演Level-0 轨迹，而Level-1， Level-2 轨迹都是在Level-0 轨迹上搜索出来的，所以一旦驾驶员模型与实际情况差异较大，则很可能将整个决策结果偏离实际最优结果。

为此，一个显而易见的方法是：

- 首选使用空旷低密度交通下的数据，使用模仿学习来训练Level-0 的驾驶员模型；
- 而通过强化学习模型给定周边障碍物来训练Level-1 Level-2 的驾驶员模型，给定周边动态障碍物轨迹下，driver 的最优决策序列结果；

通过这两个模型，在每一层目标寻优时，即可以直接使用两个模型直接获得每个目标的最优的action序列。同时由于模仿学习和强化学习学习得到的策略，就是每个action的概率分布，因此后续我们也可以使用这个分布来估计目标实际所处的Level 概率分布。

3. 博弈决策空间

宏观博弈空间

在具体的对博弈问题建模的方式上，我们有多种方式。由于需要平衡算力和模型复杂度，一般决策阶段都只需要非常粗颗粒度的action 离散采样：

- 在纵向上，通常对速度、加速度在当前车辆实际值上下范围内进行采样，且可以进行非均匀采样
- 在横向上，根据采样的时间范围，进行相应yawrate采样，比如按8s 一个点采样，就按照Pi/2/8, Pi/4/8，0 采样，如果按4s 一个点采样，即按照pi/4/4, pi/8/4,0 进行采样
- 采样时间上，根据算力和博弈方的数量，通常可以按8s 一个点，4s 一个点，2s 一个点，1s 一个点的方式进行采样

有些特殊场景下的特殊问题，甚至可以进一步降低维度和采样时间，比如自车变道时，仅考虑侧后方车辆纵向的加速度，而默认对方横向会一直保持在车辆中心，这样就把横向action空间取消了。

微观博弈的空间

除了宏观的博弈，其实还有一种微观博弈，即在轨迹规划阶段，也直接考虑博弈。使用博弈算法来精细规划，博弈双方的Motion Planning的轨迹，是的双方在0.5s~1s 级别的粗轨迹上实现对应均衡。这种方式下，实际上去博弈空间就是Motion Planning 中的连续空间了，不需要做action 离散化了。

3.1 轨迹空间

> 📌 Interaction-Aware Trajectory Prediction and Planning for Autonomous Vehicles in Forced Merge Scenarios

采用轨迹空间的方式，即通过事先对自车和博弈方的轨迹进行采样和剪枝，事先准备好轨迹级的策略空间。

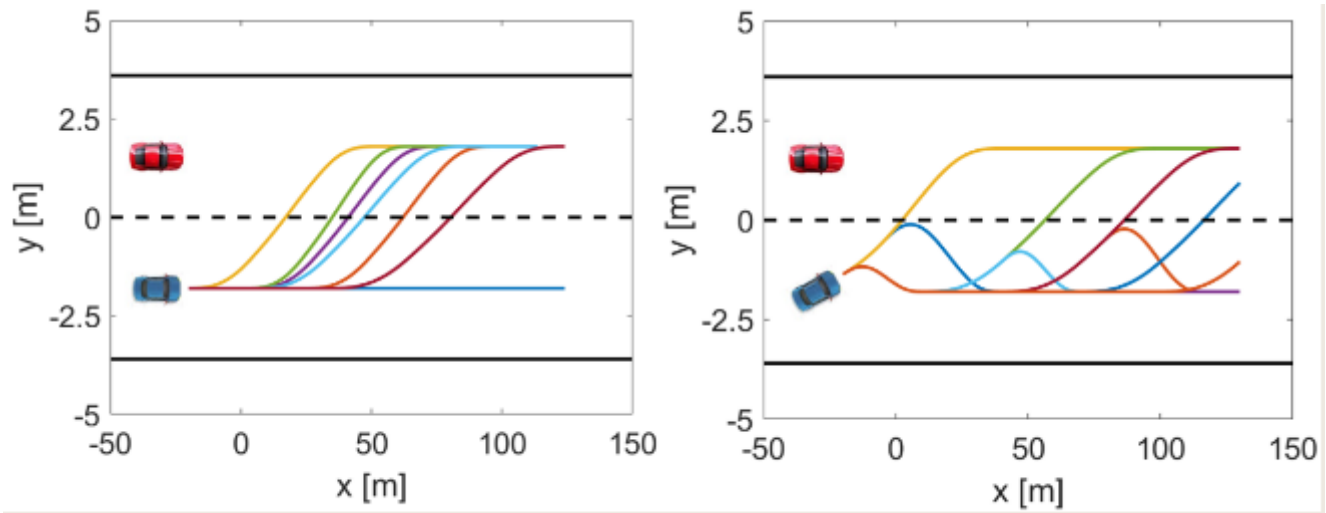


Fig. 3: Sample trajectories for the merging vehicle.

这种处理方式有很多优势：

- 在事先构建决策空间的时候就已经考虑的道路约束，这样就把很多明显的不符合道路约束的轨迹消除了，而不是留到后面寻优阶段再去检查（相对于action空间的方式），大幅降低了复杂度

- 在构建决策空间的时候就已经可以考虑一些rule based 的剪枝，一样可以降低寻优的复杂度。（比如，在整个决策时间范围内，只允许有一次变道，或者变道取消的可能，对于连续两次变道的轨迹可以事先减掉）；（再比如，可以直接对某些超过横纵向jerk 的轨迹进行剪枝）
- 同样轨迹采样还可以使用一些驾驶员模型来帮助剪枝轨迹空间，比如只采样终点状态，使用驾驶员模型推演出轨迹，这样一条轨迹就只是一个节点。

但是其也会有缺点：

构建的轨迹空间是建立在场景和道路空间上，通常只适合标准的结构化道路，这就代表了，轨迹空间的构建需要区分场景，泛化性差。在具体实施时，变道场景，十字路口场景，换道场景，多合一道路，开放空间等场景下，轨迹空间的构建就需要使用不同的方式来构建，以保证采样的轨迹能够覆盖可行的所有空间。

3.2 状态采样空间

不同于直接采样action，部分博弈算法直接对系统状态进行采样。在选定state后，反算状态之间的边作为action。这种情况下，state 是选定的，action则不是固定的，根据两个state之间的差去反算。这种方式的好处是，如果对应的reward或cost只涉及state的话，则就省去了从action 到state 的状态推算过程。而且reward 之和state相关，整个系统具备了马尔科夫性。

3.3 Action 采样空间

相比于轨迹空间，另一种方式则是直接对每一个时间点上，各个agent 可以采取的action 进行离散化采样定义。

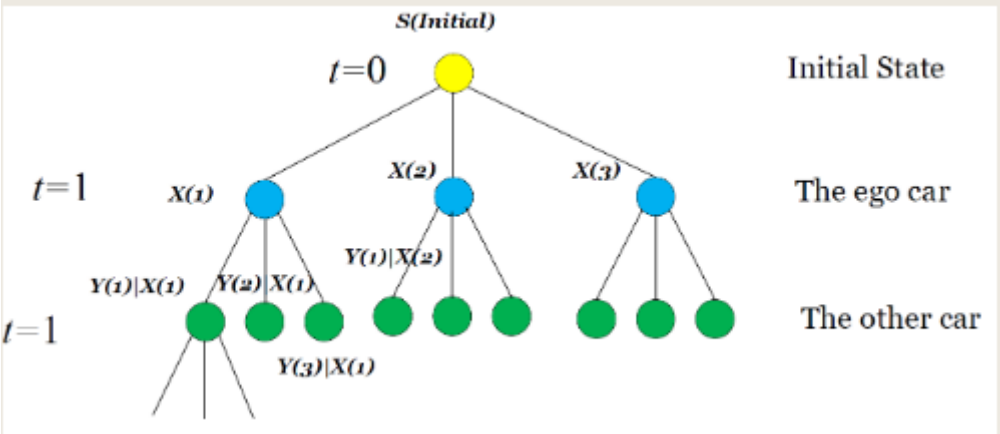
- 如果只考虑纵向博弈，则可以考虑对加速度进行采样（-3，-2，-1.5，-1, -0.5，0, 0.5 ,1, 1.5, 2 ）
- 如果需要同时考虑横纵向博弈，则可以考虑合并考虑横纵向action, 将整个action空间 简化为一个横纵向组合的候选空间。例如：

> 📌 Receding Horizon Motion Planning for Automated Lane Change and Merge Using Monte Carlo Tree Search and Level-K Game Theory

ACTION SPACE, DETAILS AND DESCRIPTION OF ACTIONS

#	Description/ combination	Acceleration (m/s^2)	Yaw rate (rad/s)
1	maintain	0.00	0.00
2	low brake	-1.50	0.00
3	low accelerate	1.50	0.00
4	mid brake	-3.50	0.00
5	high accelerate	2.50	0.00
6	high brake	-5.00	0.00
7	low left steer	0.00	π / 4
8	low right steer	0.00	-π / 4
9	high left steer	0.00	π / 2
10	high right steer	0.00	-π / 2
11	acc. + left str.	1.50	π / 4
12	acc. + rightstr.	1.50	- π / 4
13	brk. + left str.	-1.50	π / 4
14	brk. + right str.	-1.50	-π / 4

3.3.1 纳什均衡下的action决策空间

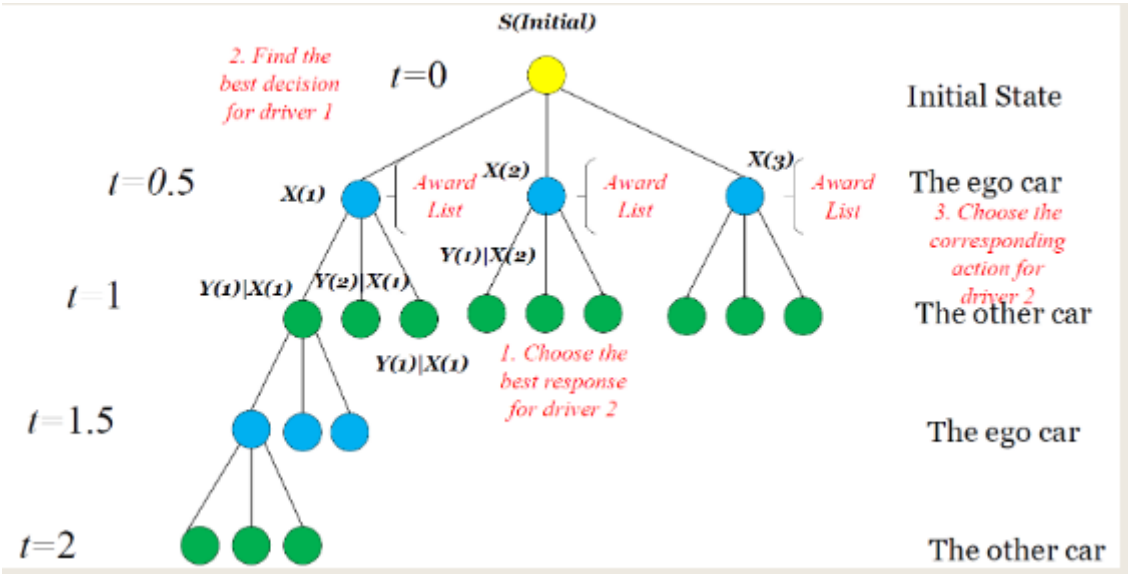


针对自车的每一种action ，遍历博弈方的各种action 空间，这样一层层扩展博弈树。

- 按照博弈双方都有14 个候选action, 博弈推演3s, 每1s 做一层博弈，则决策空间共有(14*14)^3= 752万个组合
- 按照博弈双方都有14 个候选action, 博弈推演4s, 每2s 做一层博弈，则决策空间共有(14*14)^2= 3.8 万个组合

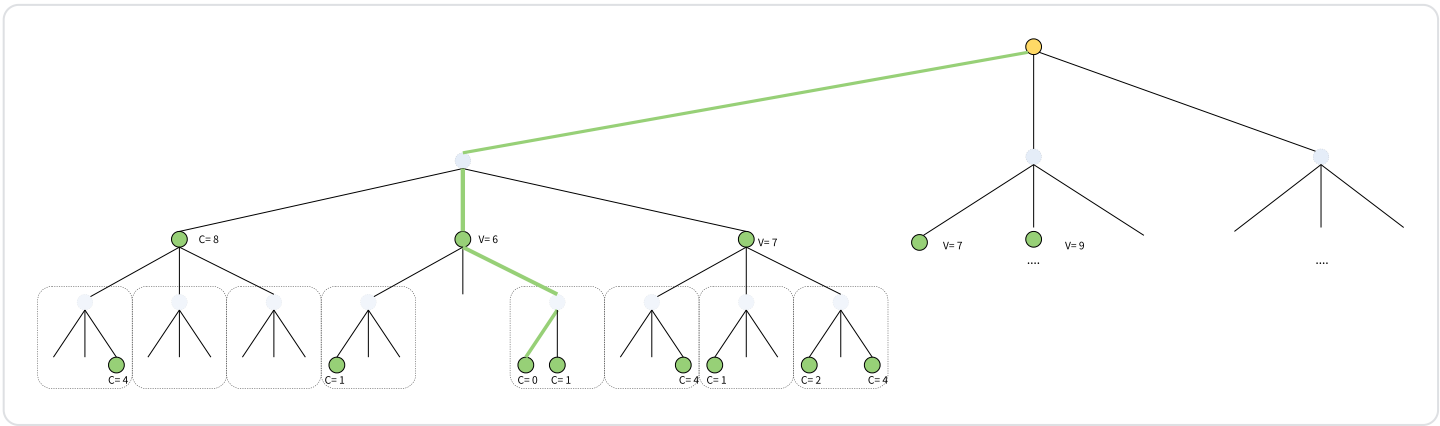
因此可见，这种方式如果使用暴力遍历寻优已经是不可能了。

3.3.2 斯塔伯格均衡下的action决策空间

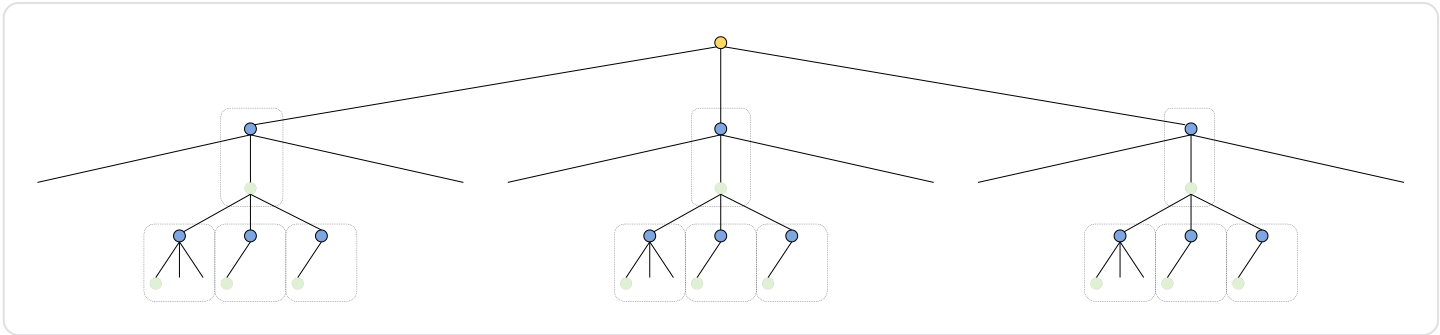


按照斯塔伯格博弈的定义，其寻优的顺序和纳什均衡的方式不同，假定自车为leader， 博弈方为follower.

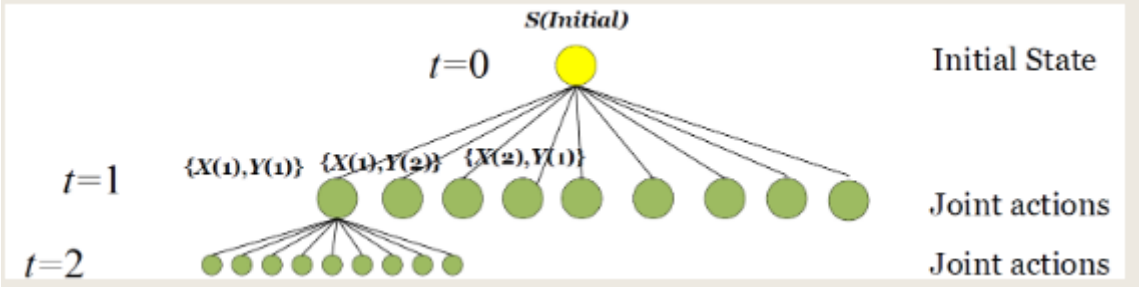
1. 先针对每一个leader的action 节点，计算follower各种action下的，follower的cost. 并将cost list暂存到其对应的自车节点上。
2. 接下来分别对follower节点和leader 节点做两次MCTS。 先使用MCTS对follower的节点（只看follower节点的cost）进行最优搜索（固定次数），得到follower 的最优轨迹（每一层follower的action）



3. 然后针对每一层确定的follower action, 再使用MCTS对leader 节点再做一次搜索，得到leader的最优action list.



3.3.3 帕累托均衡下的action 决策空间



在帕累托均衡下，我们使用联合优化的处理方式，因此决策树的深度被减半，而博弈方两车的action的一种组合成为一个joint action 节点，因此树的宽度成为action 数量的平方。

以14 个candidate action 为例， 帕累托决策树每一层就有14*14个节点。

3.4 驾驶员决策模型

人类司机的决策和行为天然就具有异构，时变，随机的特性，即便同样的初始状态下，同一个司机每次做出的结果可能总会略有差异，更何况我们在决策时，对全世界所有司机的统计性估计。因此，取决于我们需要考虑的系统不确定性的方面，可以将司机的决策，司机的观察，司机的执行（比如司机决策给定 2s 内j加速度10m/s, 实际每个司机的执行结果也有差异，2s 后的实际位置也会不一样）都可以认为是一个概率分布。当然取决于我们决策算法中，需要考虑哪些不确定性，我们不一定需要都考虑。（比如EPSILON中，就只考虑的司机的决策（而且是语义决策），而忽略了司机的观测，执行不确定性）

通常Jaakkola RL 中，会同时考虑这三种不确定性，把人类驾驶员的决策和控制行为考虑成一个POMDP 过程，则如下的三个概率模型其实就对应了强化学习中的贝尔曼公式中的三个概率分布。

$$v_\pi(s) = \sum_a \pi(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s') \right]$$

3.4.1 驾驶员概率决策模型（Boltzmann Rational Policy）

在给定候选action 空间下，我们对人类驾驶员进行建模时，可以采用两种模型：

- 贪心模型： 假定人类驾驶员将唯一选择reward （Q值）最高的那一个action。
- Boltzmann 分布模型：人类将根据每一个action获得的收益（Q值），按照Boltzmann 分布归一化以后计算得到对每一个action 采用的概率，这样驾驶员在给定Ego的action下，做出的反应不是确定的，而是按照给定收益按概率分布。

$$P(a_i) = \frac{\exp(Q_{a_i})}{\sum_{a \in A} \exp(Q_a)}$$

3.4.2 驾驶员观测模型（一般在强化学习中考虑）

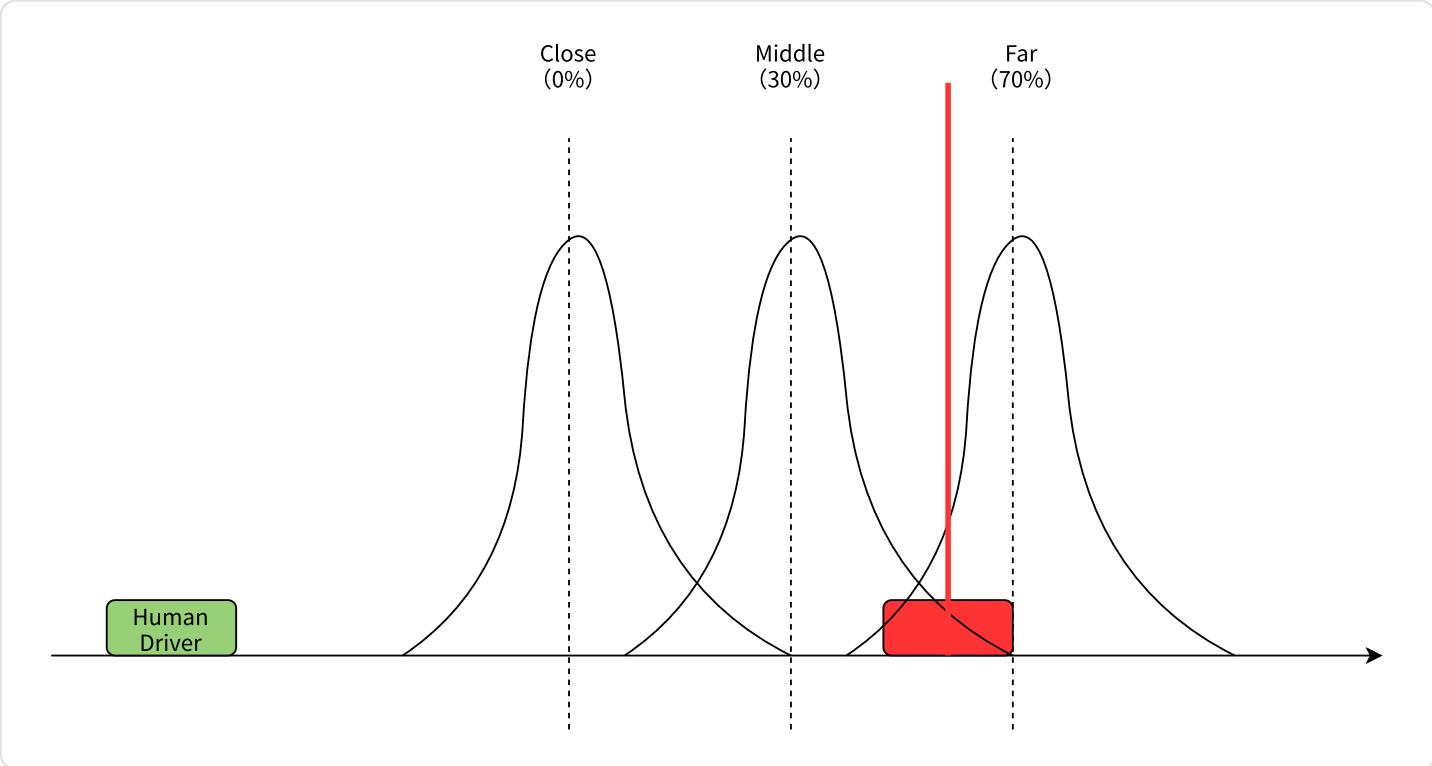
除了人类驾驶员决策的不确定性外，部分博弈算法还考虑的驾驶员观测的不确定性。 例如由于驾驶员受到遮挡，或者精力不集中等各种原因，不管是人类驾驶员，还是自动驾驶车辆对当前系统状态（相对距离）的观测总是有一定误差的。

同时，人类驾驶员其实并不能对周边障碍物的状态进行一个精准的识别，而往往只有一个宏观的，语义级别的状态判断，在进行决策时，往往也是基于这些宏观的状态判断来进行决策。

例如，司机对前方的车辆的状态判断，往往是语义级的：

位置状态： close, middle, far

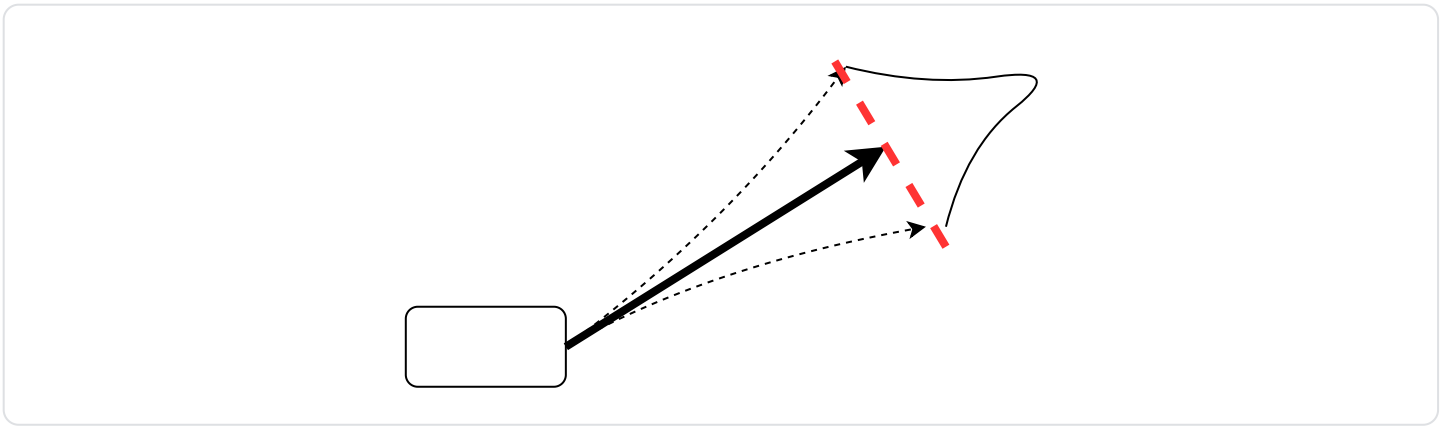
速度状态：approaching, keep , leave away



因此，在进行推演时，根据推演出的车辆位置，我们通过观测不确定模型，估计出司机对相应位置的语义分类的分布概率。

3.4.3 驾驶员状态转移模型（一般在强化学习中考虑）

由于我们在做驾驶员action 采样时，受限于算力，我们做了非常粗的离散化（action的值和决策时间step），因此如果我们直接用采样的action做目标推演时，最终司机真正控制出来的结果，其实也是一个在以采样action推算出来的状态周围的概率分布。



因此，对于决策模型来说，整个决策过程是有三个概率更新层串起来的，实际算法中，可以根据需要考虑的不确定性，将对应的概率模型加入到决策算法中，不过一般来说，只会在强化学习的方法中考虑所有的概率模型。在rule based 的算法中，状态转移模型和观测模型都简化为确定性的模型。

4. 目标函数的设计

📌 Receding Horizon Motion Planning for Automated Lane Change and Merge Using Monte Carlo Tree Search and Level-K Game Theory

4.1 自车收益reward

针对每一个节点上状态计算reward, 通常来讲，为了简化计算，都是用indicator 的方式计算reward. 在每个节点上条件满足得1分，不满足的0分。最后通过权重设计，加权得到每个节点上的总reward. 实际使用中可以从中选择需要的项。

$$
\begin{aligned}
r(a_k, s_k) = \\
&\pi_1.\Gamma_{\text{collision}} + \pi_2.\Gamma_{\text{safe dist.}} + \pi_3.\Gamma_{\text{off-road}} \\
&+\pi_4.\Gamma_{\text{bet. lines}} + \pi_5.\Gamma_{\text{speed}} + \pi_6.\Gamma_{\text{yaw}} + \pi_7.\Gamma_{\text{decel.}}
\end{aligned}
$$

4.1.1 Collision

指定节点上，博弈双方没有发生碰撞得1分，否则为0分。

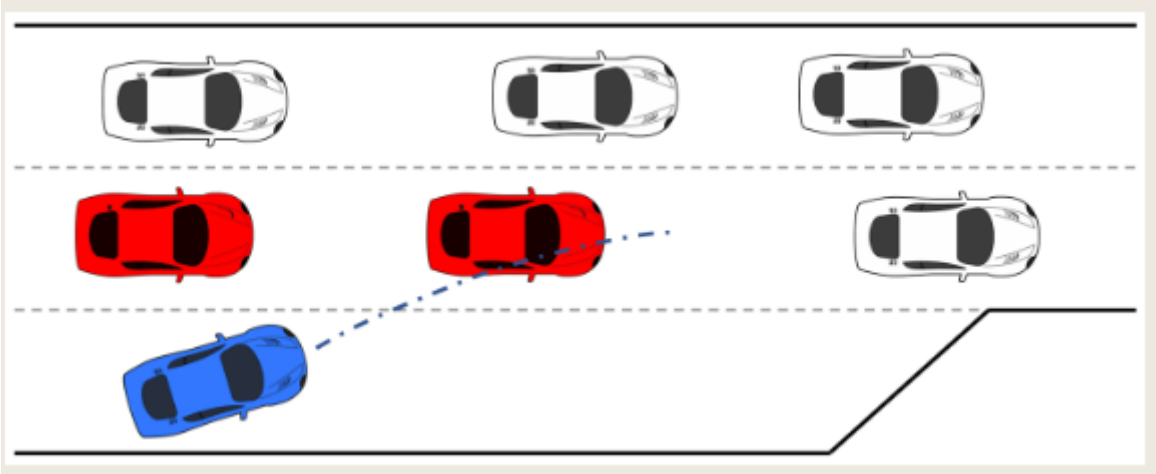
4.1.2 Safety distance

在指定节点上，博弈双方的bounding box 没有重叠得1分，否则为0分

4.1.3 Off-Road

博弈任何一方如果行驶出道路边界，或者跨过了实线，则得0分，否则得1分。这里就可以考虑对应场景产生的非对称博弈效果，例如对于匝道汇入场景，对于接近匝道结束段的目标，我们可以预判到目

标会面临无路可走的情况，因此其一定会产生更为激烈的cut-in 动机。（如下图蓝车）。 这些每辆车自身面临的静态环境差异对每个目标的决策影响，可以通过这一项考虑进来。



4.1.4 Between line

惩罚车辆骑线行驶的状态（压虚线），间接奖励了换道时间越短越好。（如果是压实线，则是off-road 项）

在指定节点上，agent 的bounding box 和车道的虚线有overlap ,则得0分，否则得1分。

4.1.5 Desired speed

奖励车辆越接近设定速度的状态（司机设定续航速度和限速的较小值）

$$\Gamma_{\text{speed}}(s_i) = \begin{cases} 1 & \text{if} \quad |v - v_{\text{dsrd.}}| \leq 1 \\ 0 & \text{if} \quad v_{\text{dsrd.}} < |v - v_{\text{dsrd.}}| \\ 1 - \frac{|v - v_{\text{dsrd.}}|}{v_{\text{dsrd.}}} & \text{otherwise} \end{cases}$$

4.1.6 Yaw reward

奖励车辆横向保持稳定的状态

$$\Gamma_{\text{yaw}}(s_i) = \begin{cases} 1 & \text{if} \quad |\psi| \leq 0.01 \\ 1 - \frac{4}{\pi} & \text{if} \quad 0.01 < |\psi| \leq \frac{\pi}{4} \\ 0 & \text{if} \quad \frac{\pi}{4} < |\psi| \end{cases}$$

4.1.7 Unnecessary Decelation

如果车辆前方给定时距内没有目标时，agent 出现制动， 得0分，否则得1分。

4.2 Coutesy reward

除了以上常规的cost 计算项，实际测试中，很多研究发现仅仅依靠以上的只针对优化自身的reward，还不能很好的描述人类司机的表现。 人类司机在交互过程中，绝大部分司机并不是一个完全自私的极大化自身收益的状态。因此，很多博弈论文中还引入了礼让收益项。

📌
- Socially-Compatible Behavior Design of Autonomous Vehicles with Verification on Real Human Data
- Social behavior for autonomous vehicles
- On Social Interactions of Merging Behaviors at Highway On-Ramps in Congested Traffic

不同论文中，对礼让收益的建模方式也不同，这里我们介绍Socially-Compatible Behavior Design of Autonomous Vehicles with Verification on Real Human Data 这篇论文里的方法。

Countesy Cost 被计算成为自车的action 对博弈方原有Planning的影响程度，或者直观的理解，自己行为对别人的打搅程度。

- 使用noisy rational Boltzmann policy 根据博弈目标在Ego 给定action下，其不同action 下收益计算其采取该action的概率，并重新归一化后得到，得到博弈车在ego 对应action 下的其action的概率分布

$$P(\mathbf{u}_O|\mathbf{x}^0,\mathbf{u}_E)=\frac{e^{R_O(\mathbf{u}_O|\mathbf{x}^0,\mathbf{u}_E)}}{\sum_{\mathbf{u}_O\in\mathbf{U}_O}e^{R_O(\mathbf{u}_O|\mathbf{x}^0,\mathbf{u}_E)}},$$

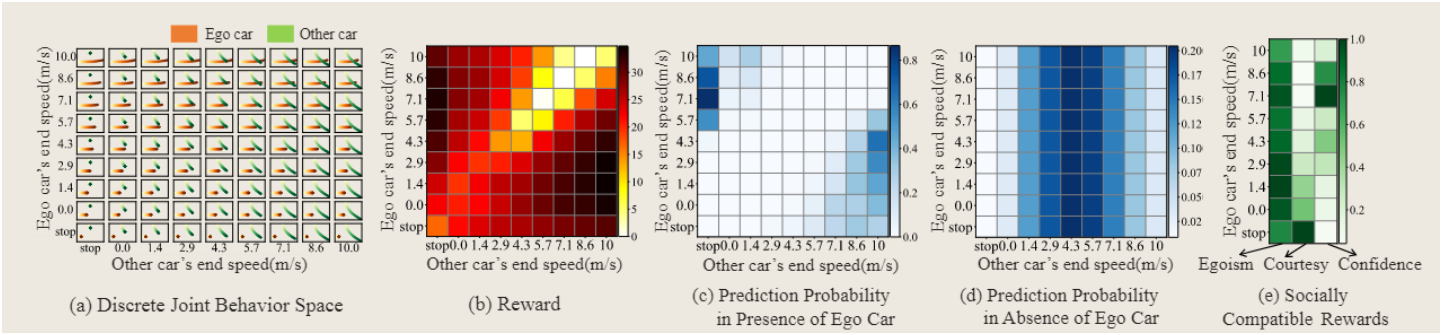
- 然后假定博弈车不受自车影响的情况下，在给定时间点下，其收益值，同样按照noisy rational Boltzmann policy 计算其各个action 的概率分布

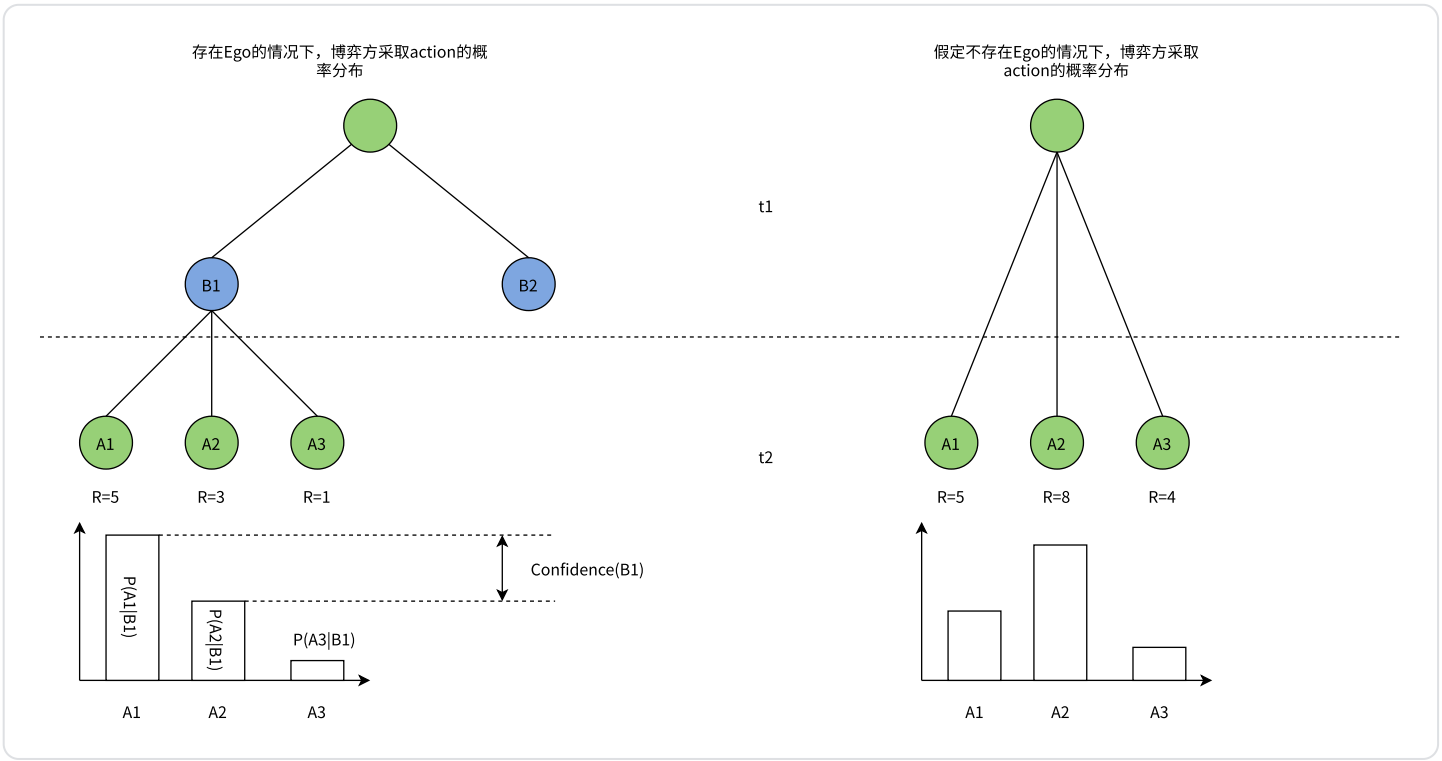
$$P(\mathbf{u}_O|x_O^0)=\frac{e^{R_O(x_O^0,\mathbf{u}_O)}}{\sum_{\mathbf{u}_O\in\mathbf{U}_O}e^{R_O(x_O^0,\mathbf{u}_O)}}$$

- 使用Kullback-Leibler 散度公式量化计算两种概率分布间的差异，以此量化评估自车从头到当前action 对目标的影响程度

（Kullback–Leibler divergence，缩写 KLD）是一种统计学度量，表示的是一个概率分布相对于另一个概率分布的差异程度，在信息论中又称为**相对熵**（Relative entropy）

$$R_{E_{courtesy}}(\mathbf{x}^0, \mathbf{u}_E) = e^{-D_{KL}\left[P(\mathbf{u}_O|x_O^0)||P(\mathbf{u}_O|\mathbf{x}^0,\mathbf{u}_E)\right]} \quad (10)$$
$$= e^{-\sum_{\mathbf{u}_O \in \mathbf{U}_O} P(\mathbf{u}_O|x_O^0)\log\frac{P(\mathbf{u}_O|x_O^0)}{P(\mathbf{u}_O|\mathbf{x}^0,\mathbf{u}_E)}}.$$





4.3 Confidence reward

除了礼让的因素，研究中还发现，人类驾驶员还存在对不确定性的厌恶。人类驾驶员会本能的去选择未来更为确定的决策。因此，很多文献中都引入了confidence 的惩罚项。该文献中，使用了第一，第二最大概率action的概率差来描述，自车选择对应action下，博弈方选择的应对action的不确定性。

$$Conf(\mathbf{x}^0, \mathbf{u}_E) = P(\mathbf{u}_O^1|\mathbf{x}^0, \mathbf{u}_E) - P(\mathbf{u}_O^2|\mathbf{x}^0, \mathbf{u}_E)$$

5. 博弈优化求解方法

5.1 暴力遍历求解

如果决策规划空间足够小，就可以使用暴力遍历求解的方式寻找最优的决策方法。比如我们目前使用的gaming 算法，两车博弈，仅考虑纵向加速度采样，按照0-10s 的平均加速度采样，每个agent最多15个点，因此最多只需要计算求解225个加速度pair, 通过规则的剪枝还可以进一步降低到100个pair左右。因此完全可以遍历求解。

5.2 动态规划求解

> 📌 Hierarchical Game-Theoretic Planning for Autonomous Vehicles

该论文将博弈算法为两层： Strategic planner （宏观博弈规划）+ Tactical planner （微观博弈规划）

在Strategic planner 一层其使用了动态规划来求解博弈问题。动态规划可以在全局空间上实现寻优，从而解决了非凸空间优化的问题。但是其依然需要遍历所有决策空间，因此如果决策规划空间非常大，则传统的动态规划方法无法实现实时的求解,。论文中：

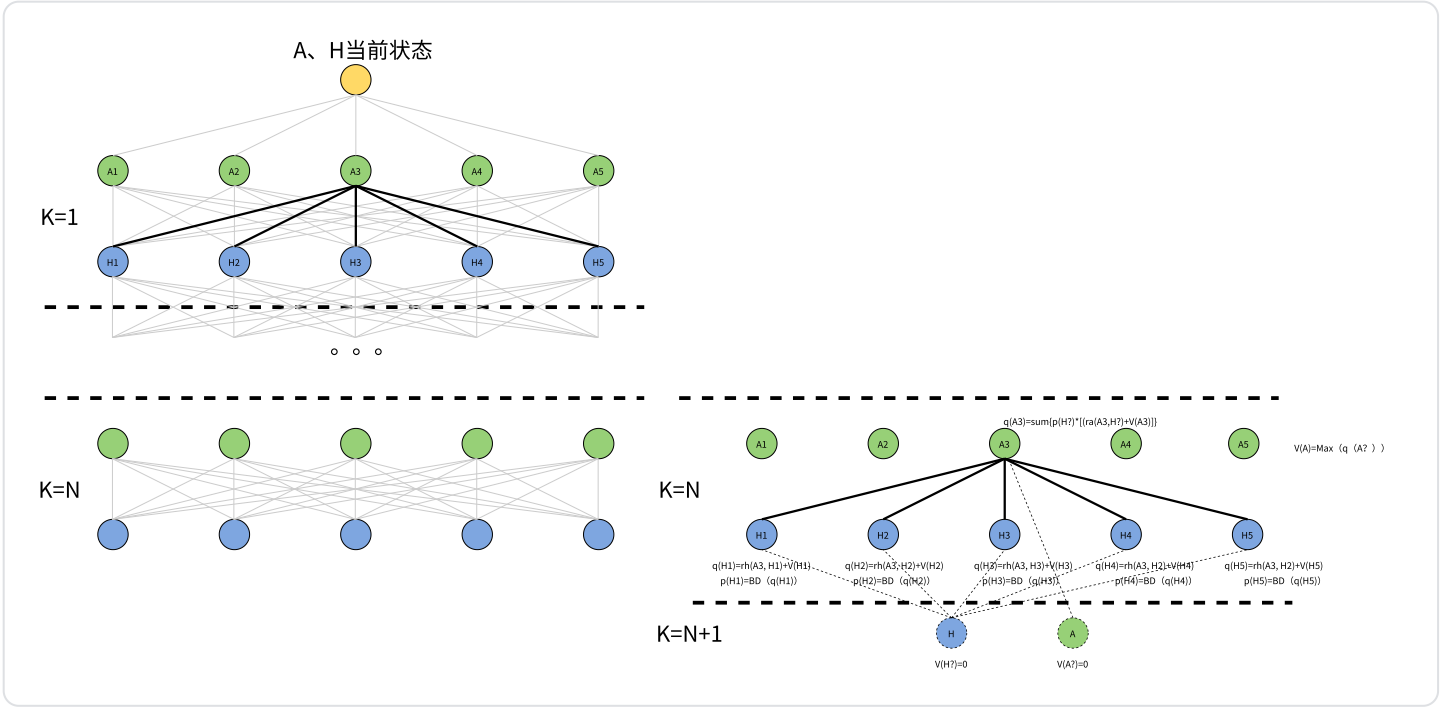
- 仅对1对1 的博弈进行了建模
- 仅考虑标准变道场景，不考虑道路环境限制（比如，存在断头车道等非对称道路环境）
- 对自车、人类动作进行采样，并假定车辆状态和action pair 之间存在直接的对应关系 $S(k+1)=\phi(S(k),a_A,a_H)$
- 博弈模型为自车固定为Leader，人类司机为Follower的斯塔伯格博弈
- Ego action 选择贪心策略（选择最大收益action），而Human action 通过Boltzmann 分布模型，通过价值来估计每个action 采用的概率

Algorithm 1: Feedback Stackelberg Dynamic Program

Data: $\hat{r}_A(\hat{s}, \hat{a}_A, \hat{a}_H)$, $\hat{r}_H(\hat{s}, \hat{a}_A, \hat{a}_H)$
Result: $\hat{V}_A(\hat{s}, k)$, $\hat{V}_H(\hat{s}, k)$, $\hat{a}^*_A(\hat{s}, k)$, $\hat{a}^*_H(\hat{s}, k)$

Initialization
for $\hat{s} \in \hat{\mathcal{S}}$ **do**
A0 $\quad \hat{V}_A(\hat{s}, K+1) \leftarrow 0;$
H0 $\quad \hat{V}_H(\hat{s}, K+1) \leftarrow 0;$

Backward recursion
for $k \leftarrow K$ **to** 0 **do**
  for $\hat{s} \in \hat{\mathcal{S}}$ **do**
    for $\hat{a}_A \in \hat{\mathcal{A}}_A$ **do**
      for $\hat{a}_H \in \hat{\mathcal{A}}_H$ **do**
H1 $\quad\quad q_H(\hat{a}_H) \leftarrow \hat{r}_H(\hat{s}, \hat{a}_A, \hat{a}_H) + \hat{V}_H(\phi(\hat{s}, \hat{a}_A, \hat{a}_H), k+1);$
H2 $\quad P(\hat{a}_H \mid \hat{a}_A) \leftarrow \pi_H[q_H](\hat{a}_H);$
H3 $\quad q^*_H(\hat{a}_A) \leftarrow \sum_{\hat{a}_H} P(\hat{a}_H \mid \hat{a}_A) \times q_H(\hat{a}_H);$
A1 $\quad q_A(\hat{a}_A) \leftarrow \sum_{\hat{a}_H} P(\hat{a}_H \mid \hat{a}_A) \times \Big(\hat{r}_A(\hat{s}, \hat{a}_A, \hat{a}^*_H(\hat{a}_A)) + \hat{V}_A(\phi(\hat{s}, \hat{a}_A, a^*_H(\hat{a}_A)), k+1)\Big);$
A2 $\quad \hat{a}^*_A(\hat{s}, k) \leftarrow \arg\max_{\hat{a}_A} q_A(\hat{a}_A);$
A3 $\quad \hat{V}_A(\hat{s}, k) \leftarrow q_A(\hat{a}^*_A(\hat{s}, k));$
H4 $\quad \hat{a}^*_H(\hat{s}, k) \leftarrow a^*_H(\hat{a}^*_A(\hat{s}, k));$
H5 $\quad \hat{V}_H(\hat{s}, k) \leftarrow q^*_H(\hat{a}^*_A(\hat{s}, k));$



Value function 计算：

论文中，另一个值得学习的点是，其在计算自车（AD）的动作收益时，使用了Human的概率模型(具体介绍，参见第3.3)来计算加权的收益（这里已经非常接近于强化学习的建模方式了），该方法在2018年之后的相关博弈论文中被大量使用，成为一种常用的计算动作收益的方法。

- 即在评估自车的某一个action的价值时，先遍历在自车action下，博弈方选择各种action, 相对于人类会获得的价值qH（A? ）；

$$q_H(\hat{a}_H)$$

- 根据估计得到的Human采用对应action获得的价值，使用Boltzmann Rational Policy 模型估计人类驾驶员采用该action的归一化概率；

$$P(\hat{a}_H \mid \hat{a}_A)$$

- 根据自车采用action和Human采用的action对，评估ego 在对应状态下的价值q(a_A,a_H)（当前action pair 获得的收益+完成状态时的价值）
- 获得Human采取每个action的概率后，对Ego在对应human action 下的价值进行加权平均，作为ego 在该action下的价值q(a_A)

$$q_A(\hat{a}_A) \leftarrow \sum_{\hat{a}_H} P(\hat{a}_H \mid \hat{a}_A) \times \left(\hat{r}_A(\hat{s}, \hat{a}_A, \hat{a}_H^*(\hat{a}_A)) + \hat{V}_A(\phi(\hat{s}, \hat{a}_A, a_H^*(\hat{a}_A)), k+1) \right)$$

5.3 ILQR/DDP迭代优化求解Nash均衡问题

主要用在微观博弈中，直接在规划优化求解的过程中，考虑轨迹的优化。自然而然的规划中的ILQR和DDP算法又可以用到这里来了。

这些基于优化的方法，相比于基于采样搜索的方法：

缺点：只能在粗解（各目标初始轨迹）所在的凸空间上进行局部的优化求解，而不能保证能在全局空间上得到最优解（这是其最大的缺陷），因此使用优化方法进行博弈寻优，最后还是要搭配采样搜索的方法，获得比较好的粗解后才能保证能够或则全局最优结果

优点：类似于ILQR轨迹规划，通过对目标的状态转移方程和cost 方程的线性化，提高了优化求解的效率。 相比于采样搜索，可以考虑更为细致的道路环境，甚至可以直接和后续的ILQR轨迹规划共用部分约束条件，实现决策，规划一体化。

📌
- Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games
- Differential Dynamic Programming for Nonlinear Dynamic Games
- ALGAMES: A Fast Solver for Constrained Dynamic Games

我们这里以Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games论文中的方法为例，介绍ilqr的思路如何应用在迭代求解纳什均衡中。

5.3.1 Local Nash Equilibrium for Dynamic Game

不同于通过采样决策空间后进行树搜索的路线，通过优化来求解Nash 均衡点的方法，实际还是将整个博弈交互环境建模成一个整体的动态环境。

对于有n 个目标的交互博弈环境，我们将每个目标的控制量作为系统的输入，而将每个目标的状态一起作为整个系统的状态。

$$
\begin{aligned}
x_{k+1} &= f_k(x_k, u_{:,k}) \\
x_0 &\ is\ fixed.
\end{aligned}
$$

$u_{:,k}$ 为所有目标在时间点k时的控制量

X_k 为所有目标在时间点k 的状态（位置，速度，航向角）

f_k 为各个目标状态转移矩阵的组合（注意这里每个目标的状态转移函数都是非线性函数）

而对于优化问题的目标函数，不同于轨迹规划的优化问题，Nash 均衡的优化目标为，每个目标都想最小化他自己的cost function（也是非线性函数）

$$
J_n(u) = \sum_{k=0}^{T} c_{n,k}(x_k, u_{:,k})
$$

这里需要特别注意，对于博弈问题，每个目标的cost不仅取决于自己的状态（舒适性、效率，静态环境约束），同时还受到其他目标状态的影响（避免碰撞）。而其他目标的状态受到其控制量决定，所以最终来说，每个目标的cost是所有博弈方控制量的函数。

博弈论证明，在给定初始状态附近一定有个最优均衡点，满足：

$$\begin{aligned} J_i^* &\triangleq J_i(\gamma_1^*; \ldots; \gamma_{i-1}^*; \gamma_i^*; \gamma_{i+1}^*; \ldots \gamma_N^*) \\ &\leq J_i(\gamma_1^*; \ldots; \gamma_{i-1}^*; \gamma_i; \gamma_{i+1}^*; \ldots \gamma_N^*), \end{aligned}$$

即，在均衡点上，任何一方单方面的想单方面的改变其自己的轨迹，都会造成自身cost 的增加。

为了求解这样一个优化问题，这篇文章使用了类似ILQR轨迹优化的方法，将优化问题处理成一个迭代求解LQR 的问题。这里的做法与ilqr类似，ilqr通过反向迭代，正向迭代的方式完成一次lqr更新，而本文则是采用类似思路，也进行反向迭代，正向迭代完成一次lqgame的计算，伪代码如下：

Algorithm 1: Iterative LQ Games

Input: initial state $x(0)$, control strategies $\{\gamma_i^0\}_{i\in[N]}$, time horizon T, running costs $\{g_i\}_{i\in[N]}$

Output: converged control strategies $\{\gamma_i^*\}_{i\in[N]}$

for *iteration* $k = 1, 2, \ldots$ **do**
- $\xi^k \equiv \{\hat{x}(t), \hat{u}_{1:N}(t)\}_{t\in[0,t]} \leftarrow$ getTrajectory$\left(x(0), \{\gamma_i^{k-1}\}\right)$;
- $\{A(t), B_i(t)\} \leftarrow$ linearizeDynamics$\left(\xi^k\right)$;
- $\{l_i(t), Q_i(t), r_{ij}(t), R_{ij}(t)\} \leftarrow$ quadraticizeCost$\left(\xi^k\right)$;
- $\{\tilde{\gamma}_i^k\} \leftarrow$ solveLQGame$\left(\{A(t), B_i(t), l_i(t), Q_i(t), r_{ij}(t), R_{ij}(t)\}\right)$;
- $\{\gamma_i^k\} \leftarrow$ stepToward$\left(\{\gamma_i^{k-1}, \tilde{\gamma}_i^k\}\right)$;
- **if** *converged* **then**
  - **return** $\{\gamma_i^k\}$

与之前介绍的ILQR的轨迹规划方法类似，通过对局部LQR 问题的反复迭代求解，直至每个目标的最优轨迹收敛。在每一轮迭代中的步骤如下：

1. 先根据上一轮迭代优化计算得到的最优控制量（先根据优化得到的最优增益，根据上一轮的状态量，计算得到这一轮的最优控制量），通过状态转移方程计算得到每个目标的轨迹
2. 在新计算得到的每个目标新状态周围，对状态转移方程进行线性化，得到所有目标在新轨迹点上的A，B矩阵

3. 在新计算得到的每个目标新状态周围，对每个目标的cost进行泰勒展开，得到二次型delta cost 函数
4. 针对每一个目标的delta cost函数优化就是一个 LQR 形式的game优化问题，因此就可以通过黎卡提方程来求解，每个目标的控制优化量序列
5. 根选定的line search 因子，往优化的控制方向迭代一定比例，得到新的最优控制序列

5.3.2 LQ game 问题定义和求解

按照之前在上一轮轨迹点附近进行线性化以后得结果，优化问题成为了一个 LQ Game问题。 LQ Game就是LQR 优化问题扩展到多目标空间上的耦合优化问题。

即整个系统（包括了所有博弈目标）的系统状态方程为：

$$x_{t+1} = A_t x_t + \sum_{j=1}^{N} B_t^j u_t^j \, .$$

其中每个目标的目标是分别最小化其cost 函数

$$J^i = \frac{1}{2} \sum_{t=1}^{T} \left[\left(x_t^T Q_t^i + 2 q_t^{iT} \right) x_t + \sum_{j=1}^{N} \left(u_t^{jT} R_t^{ij} + 2 r_t^{ijT} \right) u_t^j \right]$$

每个目标的cost 函数不仅与当时其它车辆的状态相关，也和当时其它目标的控制量相关，因此不同于标准的LQR 控制或者轨迹规划，解LQ Game 需要使用coupled optimization问题。

求解LQ Game有Open-Loop（使用庞特里亚金最小值原理） 和feedback （使用HJB 动态规划原理）两种方式，这里我们介绍feedback 的方式。

省去完整的推导过程（需要的话可以到github 上看对应项目的文档），我们可以得到如下求解LQ game 的过程：

每个博弈目标都根据当前整个系统状态来计算控制量：

$$u_t^{i*} = -P_t^i x_t - \alpha_t^i$$

而每个目标在给定时间点的状态，是系统状态的二次型：

$$V_t^i(x_t) = \frac{1}{2} \left(x_t^T Z_t^i + 2 \zeta_t^{iT} \right) x_t + n_t^i$$

而解LQ Game 问题就是要迭代求出 P_t^i , α_t^i , Z_t^i , ζ_t^i , n_t^i (n_t^i 是个常量项，可以不算)。 注意这些量是agent dependent 的，有几个博弈方就有几个对应矩阵. i 对应了agent 的编号。

对于任意一个agent, 其 $\begin{matrix} Z_{N+1}^i = 0 \\ \zeta_{N+1}^i = 0 \\ n_t^i = 0 \end{matrix}$, 我们就从这个终端状态进行backward 推导。

1. 从所有目标上一步的 Z_{t+1}^i ，通过如下公式计算这一步的 P_t^i ，（注意这里针对每一个目标都有如下对应的方程，因此其实我们是一个i 个变量的方程组）

$$(R_t^{ii} + B_t^{iT} Z_{t+1}^i B_t^i) P_t^i + B_t^{iT} Z_{t+1}^i \sum_{j \neq i} B_t^j P_t^j = B_t^{iT} Z_{t+1}^i A_t$$

2. 与上一步类似，通过所有目标的上一步的 ζ_{t+1}^i ， 如果如下公式计算这一步的 α_t^i （类似的，也是一个线性方程组，一次将所有目标的 α_t^i 求解出来）

$$(R_t^{ii} + B_t^{iT} Z_{t+1}^i B_t^i) \alpha_t^i + B_t^{iT} Z_{t+1}^i \sum_{j \neq i} B_t^j \alpha_t^j = B_t^{iT} \zeta_{t+1}^i + r_t^{ii}$$

3. 通过第1步计算得到的 P_t^i ， 通过如下公式计算得到这一步，所有目标的 Z_t^i

$$Z_t^i = Q_t^i + \sum_{j=1}^{N} P_t^{jT} R_t^{ij} P_t^j + F_t^T Z_{t+1}^i F_t$$

其中：

$$F_t = A_t - \sum_{j=1}^{N} B_t^j P_t^j$$

4. 通过第2步计算得到的 α_t^i ，通过如下公式计算得到这一步所有目标的 ζ_t^i

$$\zeta_t^i = q_t^i + \sum_{j=1}^{N} \left(P_t^{jT} R_t^{ij} \alpha_t^j - P_t^{jT} r_t^{ij} \right) + F_t^T \left(\zeta_{t+1}^i + Z_{t+1}^i \beta_t \right)$$

其中：

$$\beta_t = - \sum_{j=1}^{N} B_t^j \alpha_t^j$$

通过以上1-4步反复的迭代，就可以计算得到每个目标在每个时间步上的最优控制gain P_t^i 和 α_t^i 从而当前状态 X（0） Forward 推导出最优控制量和最优状态。

5.4 蒙特卡洛树搜索MCTS

📌
- A Survey of Monte Carlo Tree Search Methods
- Receding Horizon Motion Planning for Automated Lane Change and Merge Using Monte Carlo Tree Search and Level-K Game Theory

一般来说对宏观决策，我们都需要在非常大的决策空间内进行最优搜索，常规的优化搜索算法已经不可能满足实时性的要求了。例如，在十字路口，有三个博弈对象，每个博弈对象在任一时间点有15个横纵向耦合的action可供选择，我们离散化决策horizon为每2s 做一次选择，做三步推演6s。这样一个决策空间有多大呢?

Action space = 15^3^3= 380亿种组合, 在这么大一个空间内，进行寻优，常规方法已经完全不可能了。因此，一种可行的方法是使用Level-K 理论+MCTS + 启发剪枝算法 来共同降低博弈寻优的复杂度。

- 首先Level-K 博弈理论将多目标之间的耦合优化问题，解耦成多层的寻优问题。每个目标在本层寻优最优action序列时，其他目标的action 是确定的。这样在每一层上，对该目标的最优轨迹寻优维度为15^3=3375. 按照Level-K理论，三个目标之间，我们需要寻优7次 （Level-0 轨迹是直接推出来的，不需要寻优搜索），这样整个优化空间缩减为3375*7=23625 ，搜素空间极大的缩减了。
- 其次，即便是3375个搜索节点，依然过高，我们依然需要采用MCTS来提高搜索效率。例如，这3375的空间上，我们依然不可能遍历所有可能组合，例如，每个目标每层搜索，只能做200次搜索，如果保证我们能够在有限的搜索空间上，最大概率的选择出相对较优的策略序列来？ 这里我们就需要用到MCTS。

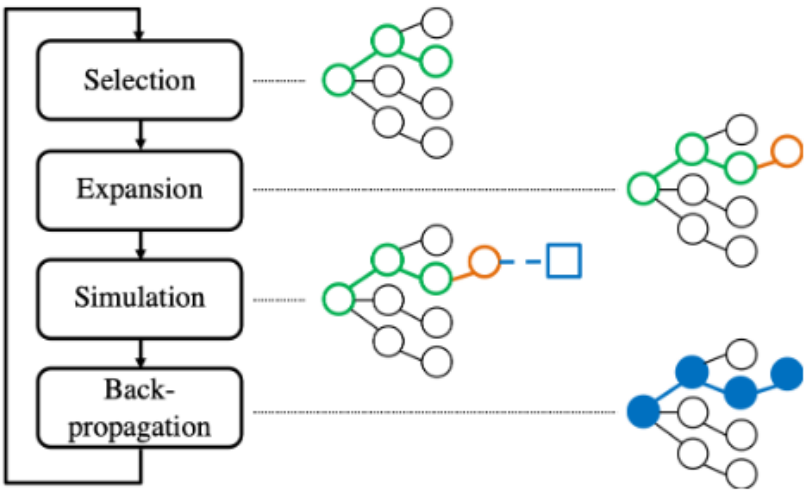
5.4.1 MCTS 基本原理

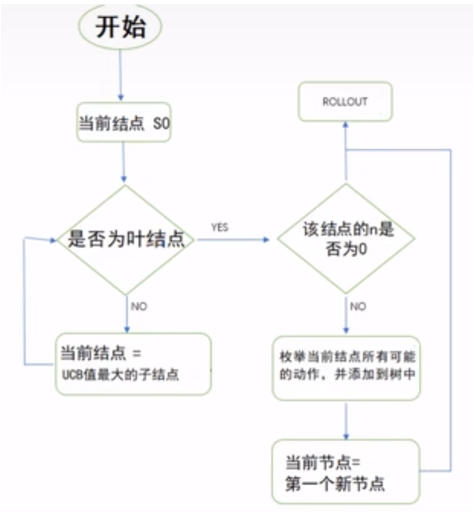
MCTS 设计出来的目的就是用来解决在巨大决策空间中，进行最优搜索的问题。既然决策空间极为巨大，我们不可能去完全遍历，我们所有的资源（计算时间和memory）只能让我们在很小的一部分空间内完成搜索，那么问题就成为了，如何在给定的资源下，尽量提高搜索效率，避开无效空间，尽量逼近全局最优解。

- MCTS的基本思想是在搜索的过程中，对树状空间进行实时的评估，优先去搜索有较大可能是最优空间的分支。是一种广度优先和深度优先的理想平衡方法。并且MCTS并不要求，必须完成对整个空间的搜索，而是以对最优性做出妥协，来降低计算时间。核心概念，有多少钱，办多少事，提高花钱的决策效率，有限的计算资源下，选出较优的结果。
- MCTS不需要实现构建完整的决策空间，而是一遍搜索，一遍扩展树空间，从而不仅能够提高搜索效率，也能降低对内存的占用。
- 每一次MCTS由4个步骤组成：

- 选择：从顶层往下搜索UCB函数值最高的节点，直至叶节点
- 扩展：当搜索到叶节点时，检查该叶节点是否被访问过，如果有被访问过，则枚举该叶节点下所有可能的动作，添加到博弈树中，并从中随机选择一个动作节点，作为新的叶节点，然后进行随机采样推演；如果是尚未访问过的叶节点，则直接进行随机采样推演；
- 推演：从叶节点状态，随机采样n 步action（这也为什么叫蒙特卡洛搜索的来源），直至终点步（比如我们确定推演n步，那从就是叶节点随机采样action，直至第n步），计算从根据节点到最终节点的完整采样序列的cost 并作为叶节点的cost;

  其中从根节点到叶节点，按照当前步的最优搜索路径确定，从叶节点到终点节点按照随机采样确定。对拼出的这条轨迹，使用车辆状态方程评估其轨迹的reward.
- 回溯：从叶节点往上回溯到根节点，更新一路上每个节点的访问次数和UCB 函数值





5.4.1.1 UCB（Upper Confidence Bound）函数值

用来体现该节点是否值得去探索的启发值，其平衡对已有的信息的利用（Exploitation）和对未知空间的探索（Exploration）

$$UCT = \frac{r_a}{n_a} + c.\sqrt{\frac{\log N}{n_a}}$$

r_a 为对应节点的下游节点累积$value$
n_a 为对应节点被访问过的次数
c：为$explorer$ 系数
N：为树的总搜索次数

通过这个函数来作为启发值，评估该节点是否值得搜索，其由两部分组成：

$$\frac{r_a}{n_a}$$

代表根据之前的搜索历史评估的平均价值，代表了对之前搜索结果的总结

$$\cdot\sqrt{\frac{\log N}{n_a}}$$

则代表了对该节点未知程度的评估

而参数C则代表了在Exploitation 和Exploration 之间的平衡程度，其值越大，代表越倾向探索未知的节点。

5.4.1.2 UCT-MCTS

伪代码：

算法根据是对已有的节点进行选择的策略和对尚未添加到树中节点的选择策略，分为了两部分：

- Tree Policy：对应选择和扩展步
- Default Policy： 对应推演和回溯步

后续针对两个策略的改进上，就形成了MCTS的各种变形。

Algorithm 2: The UCT algorithm

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))
function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v
function EXPAND(v)

choose $a \in$ untried actions from $A(s(v))$
add a new child v' to v
with $s(v') = f(s(v), a)$
and $a(v') = a$
return v'

function BESTCHILD(v, c)

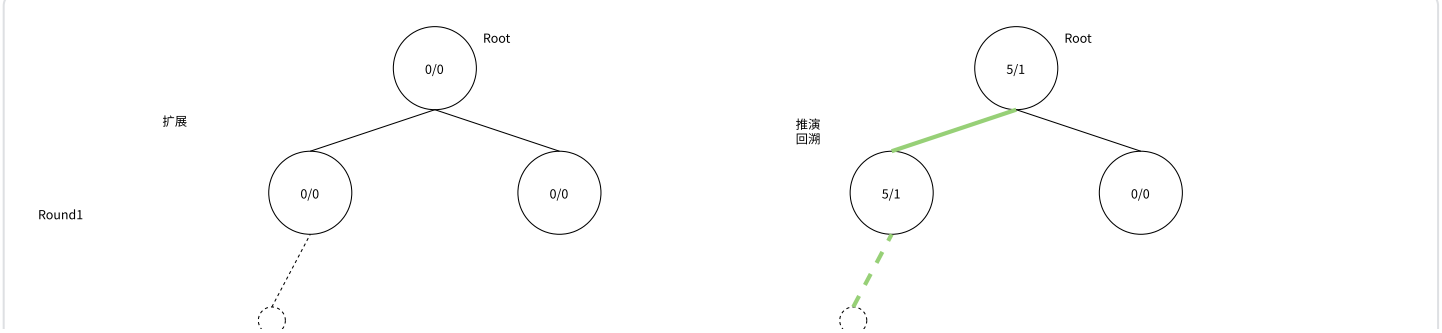
return $\arg\max_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

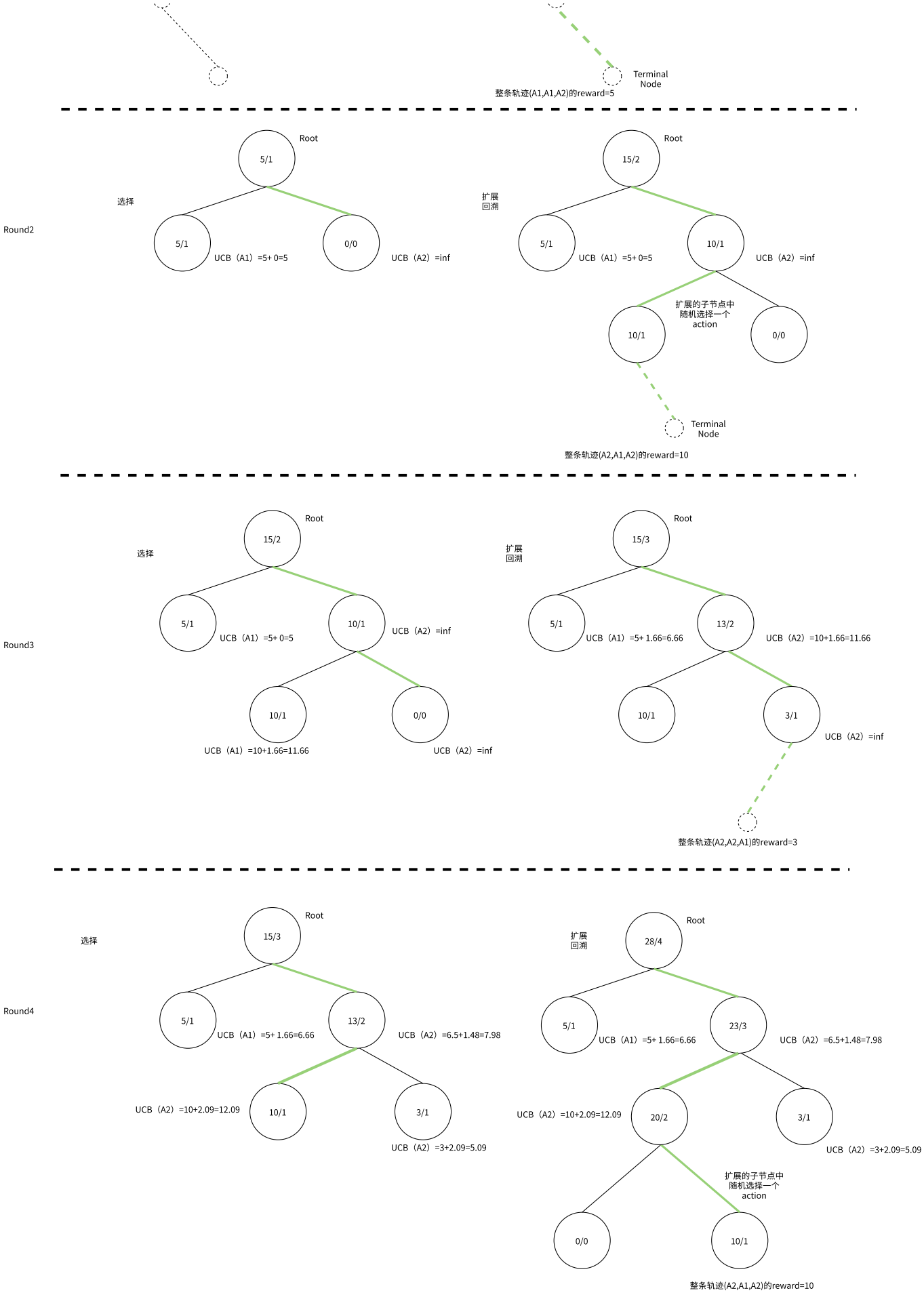
function DEFAULTPOLICY(s)
while s is non-terminal **do**
choose $a \in A(s)$ uniformly at random
$s \leftarrow f(s, a)$
return reward for state s

function BACKUP(v, Δ)
while v is not null **do**
$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta(v, p)$
$v \leftarrow$ parent of v

$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta$
$\Delta \leftarrow -\Delta$
$v \leftarrow$ parent of v

示例：





5.4.2 MCTS的启发加速

MCTS虽然已经在搜索过程中利用到了中间value 值来指导搜索方向，但是对于大空间下的决策，依然不够高效，无法满足实时性（100-300ms) , 因此依然需要其他的方法来辅助MCTS 提高搜索效率。

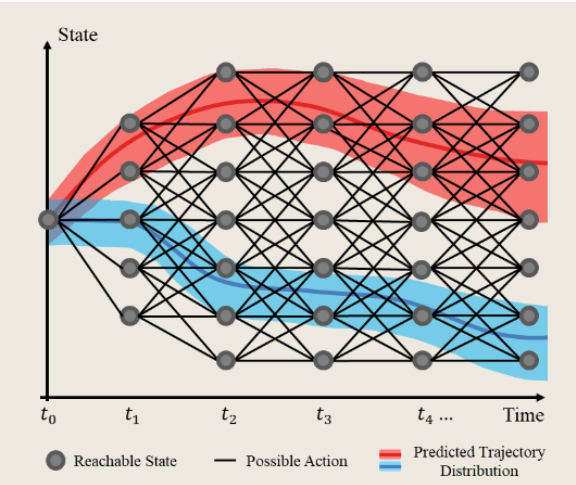
5.4.2.1 利用预测轨迹启发MCTS

> 📌 Efficient Game-Theoretic Planning with Prediction Heuristic for Socially-Compliant Autonomous Driving

一种可行且自然而然的方式就是使用已有的预测轨迹和安全约束来启发和指导树搜索。general的蒙特卡洛树搜索，完全不考虑已有的信息，因此存在大量的无效搜索，特别是真正有价值的分支（符合驾驶习惯，符合运动约束，符合交通规则）是非常稀疏的，因此进一步充分利用这些信息是非常必要的。

而恰好预测就为我们提供了这些信息，而博弈过程中，自车的action 对opponent 的影响，可以看做是对原始预测模块的扰动。我们优先在原始预测的轨迹周围进行搜索，可以大幅提高搜索效率。

因此，在general MCTS算法的基础上，可以进行如下改动来提高效率：



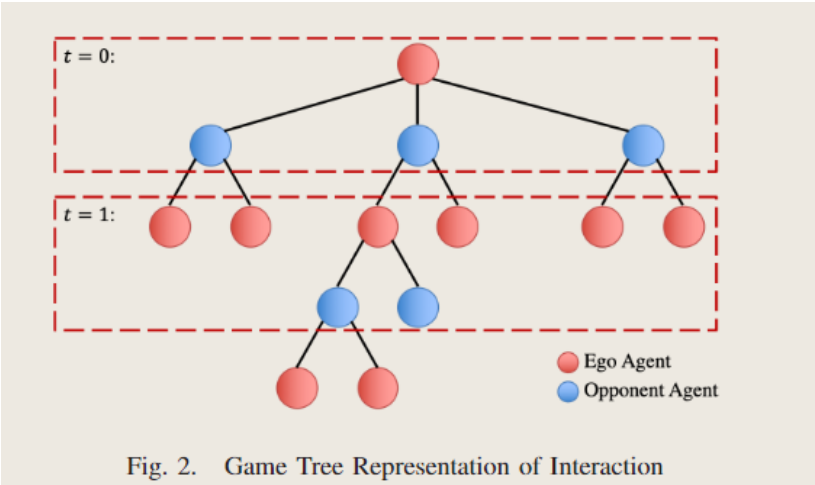


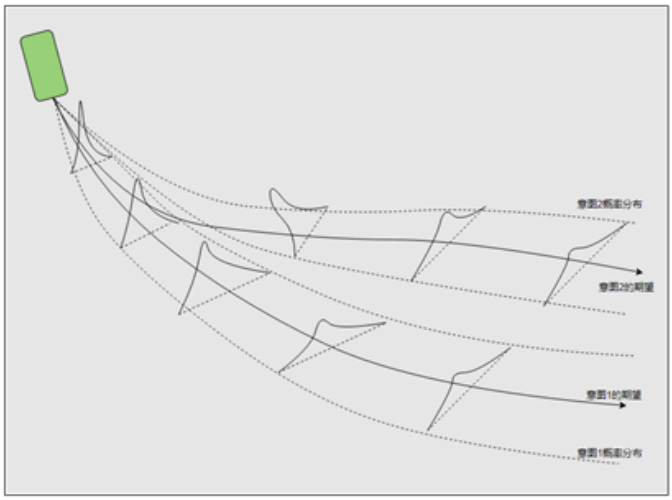
Fig. 2. Game Tree Representation of Interaction

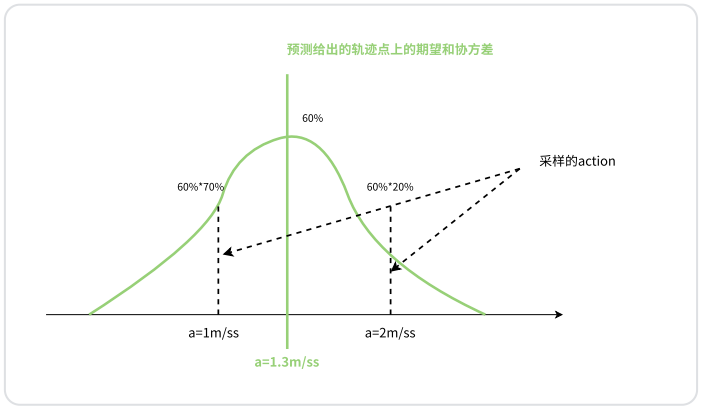
- 将opponent目标原始的预测轨迹根据其轨迹点上的协方差矩阵，来计算原始轨迹点周边action 的confidence.

$$G_t^i = \{\mathbf{x}_t^O | (\mathbf{x}_t^O - \hat{\mathbf{y}}_t^i)^T \Sigma_t^{i^{-1}} (\mathbf{x}_t^O - \hat{\mathbf{y}}_t^i) \leq \rho\}$$

具体方法是通过轨迹点上的协方差根据马氏距离计算预测轨迹action其周边action 的confidence. 注意这里可以考虑预测给出的多模态轨迹的，根据轨迹概率和轨迹点上的协方差矩阵，可以计算得到预测轨迹点周边action的分布。

同时对应节点上的概率也会被标记上其对应那一条预测轨迹，这样action在roll-out 的时候将优先选择与父节点同属一条预测轨迹的action 节点。





- 对MCTS 树搜索做了一定改进，其伪代码：

其中，第1行为根据如上的方法，根据预测轨迹对相应的opponent action 计算confidence.

Algorithm 1 Prediction Heuristic Search Architecture

Input: Current state $\mathbf{x}_0$, K predicted trajectories $\hat{\mathbf{Y}}_N^i = \{\hat{\mathbf{y}}_1^i, ..., \hat{\mathbf{y}}_N^i\}, i = 1, ..., K$.
Output: Root node n_0^E of the built tree

Initialize $G = \{G_1^1, ...G_N^K\}$ from $\hat{\mathbf{Y}}_N^i$, for $i = 1, ..., K$ and $t = 1, ..., N$.
Initialize the root node $n_0^E = (\mathbf{x}_0, \emptyset, \emptyset)$.
Initialize iteration counter $Iter = 0$.
while $Iter <$ Maximum Iterations **do**
    $n_l \leftarrow \text{SELECTION}(n_0^E)$
    if n_l is non-terminal **then**
        $n_N, U_N^E, U_N^O \leftarrow \text{Roll-out}(n_l)$
    else
        $n_N, U_N^E, U_N^O \leftarrow n_l, U_N^E(n_l), U_N^O(n_l)$
    end if
    if $I_{unsafe}(n_N)$ **then**
        $q^E, q^O \leftarrow 0, 0$
    else
        $q^E, q^O \leftarrow R^E(\mathbf{x}_0, U_N^E, U_N^O), R^O(\mathbf{x}_0, U_N^E, U_N^O)$
    end if
    while n_l is not NULL **do**
        $C(n_l) \leftarrow C(n_l) + 1$
        $w_{\text{conf}} \leftarrow 1$
        if n_l is a ego node **then**
            $w_{\text{conf}} \leftarrow w_{\text{conf}}(n_l, G)$
        end if
        $q_s^E, q_s^O \leftarrow w_{\text{conf}} \times q^E, w_{\text{conf}} \times q^O$
        $Q^E(n_l), Q_s^E(n_l) \leftarrow Q^E(n_l) + q^E, Q_s^E(n_l) + q_s^E$
        $Q^O(n_l), Q_s^O(n_l) \leftarrow Q^O(n_l) + q^O, Q_s^O(n_l) + q_s^O$
        $n_l \leftarrow$ parent of n_l
    end while
end while
return n_0^E

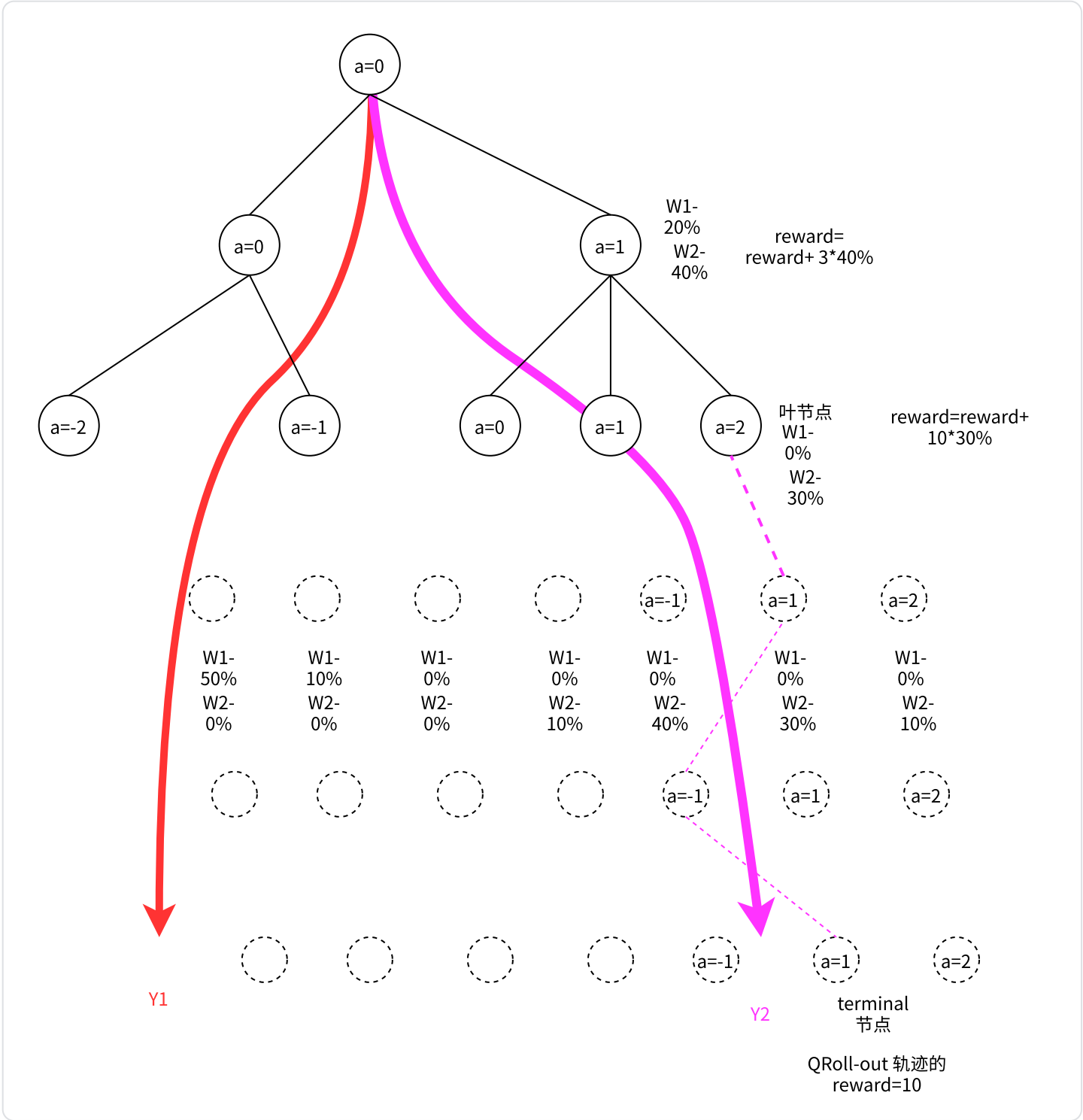
其中的Selection 函数为：

Algorithm 2 Function for Algorithm 1

```
function SELECTION(n_t)
    while n_t is non-terminal do
        if n_t is not fully expanded then
            n_l ← expand n_t with an untried action.
            while I_unsafe(n_l) do
                n_l ← expand n_t with an untried action.
            end while
            return n_l
        else
            n_l ← solving Eq. 5 with Q_s^E when n_t is ego node; Q_s^O when n_t is opponent node.
        end if
    end while
end function
```

相比于genenral 的MCTS主要有如下改动：

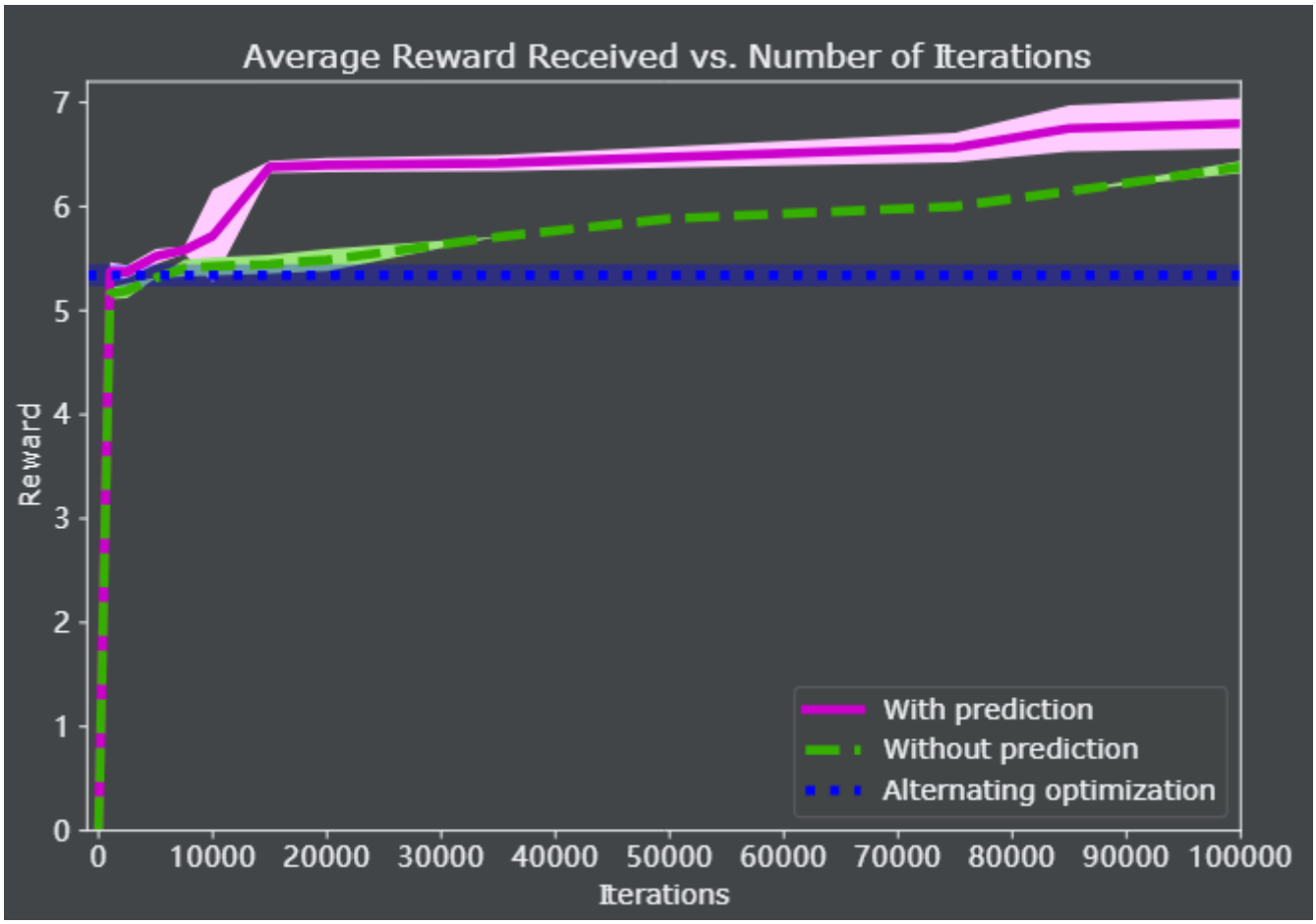
1. 每个节点的value 值除了默认的value, 对于自车节点还增加了一个加权value(Q_s)，而进行节点selection 的时候使用的是加权value来选择子节点
2. 在新节点 扩展的时候， 就对新采样的节点进行安全检查（有没有博弈方碰撞，有没有行驶出可行驶区域），如果是不安全的节点，则一票否决，重新选新的节点。（这一点是天然要遵循，不同于传统游戏中使用MCTS，自动驾驶安全第一，轨迹中任何一点不安全，整个轨迹就不可用）
3. 而在simulation 的时候（随机采样的时候），如果roll out 出的轨迹点不安全，则simulation 出的轨迹reward 被置为0，这样就会开始roll-out 的叶节点产生一定的惩罚，但是又不会完全屏蔽该叶节点下其他action的采样。
4. 在simulation的时候，不是随机采样action 节点，而是按照父节点所属轨迹的confidence 概率，以更高概率采样action 对应的action 节点； 同时roll-out 采样的子节点，与父节点的action需要满足一定的舒适性限制，比如子节点的加速度和父节点的加速度差应小于一定阈值



5. 在回溯阶段，根据当前节点对应confidence值对下游获得的reward 进行加权计算后再往上传递加权value (Q_s)

5.4.2.1.1 启发加速效果：

- 由于充分利用了预测轨迹的信息，相当于搜索会尽量在预测给定控制的周边进行搜索，实际上大幅减少了无效搜索次数
- 但是同时也有缺点，会比较依赖预测轨迹的质量，如果预测的轨迹出现较大的误差，很可能会将搜索结果限制在预测所在的凸空间内。本质上还是在Exploration 和 Exploitation之间做平衡。



5.4.3 使用强化学习进行MCTS搜索guide

另一种典型且主流的方式，则是使用强化学习模型进行对MCTS的搜索进行guide （e.g. AlphaGo 路线）。这种路线即避免了强化学习的长尾效应，保证了决策的可解释性和稳定性，又提高了MCTS的效率。

具体内容待补充...

6. 博弈决策结果输出方式

6.1 确定最优模态输出

如前述，如果认为博弈方在一个时间点上，只可能处于一个模态下（对于斯塔伯格博弈，要么是leader ,要么是follower），（对于Level-K reasoning, 则一个agent 要么是Level0，要么是Level1， 要么是Level2）。 决策算法需要根据agent 的实际状态，估计一个最优状态。

这种方式，通常相对保守，为了安全考虑。通常在对目标的模态Belief 更新到某一个阈值之前，自车先采样较为保守的策略，或者先假定对方是leader, 或者Level 0 的状态。直至目标模态的Belief update 收敛到足够高的概率。

6.2 多模态输出

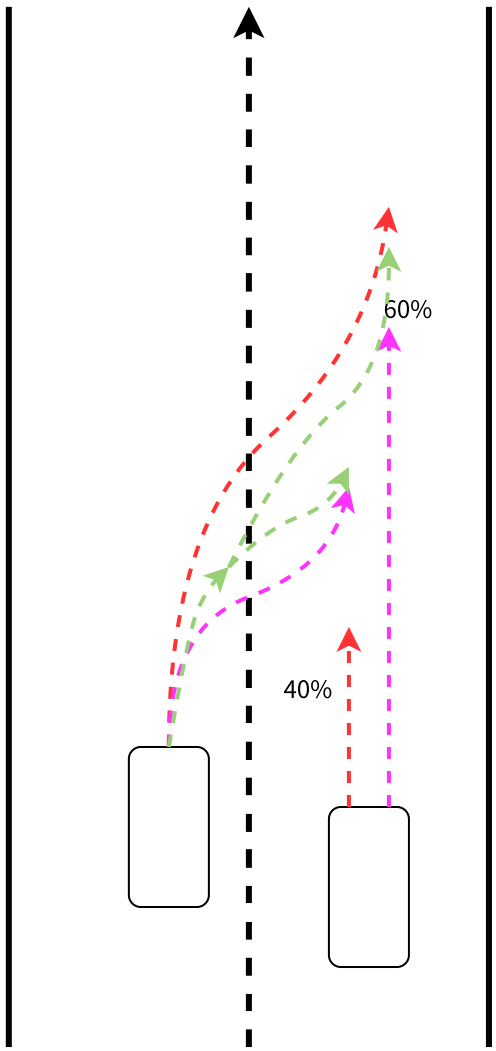
- A Game-Theoretic Strategy-Aware Interaction Algorithm with Validation on Real Traffic Data

不同于确定目标固定在某一个模态，多模态输出的认为目标可能同时处在两种不同的模态，只是概率分布不同。这个模态可以是斯塔伯格的leader /Follower Role， 也可以是Level-L中的Level，甚至是目标使用的不同的博弈模型。

多模态输出模型下，通过博弈模型实际上完成了对目标交互预测，并且结合后续的Contingency planning 实现更为灵活的决策规划。这种框架下，根据目标表现出来的模态概率，后续的Planning也可以根据概率来比例控制轨迹。这种方式就不再是确定的黑与白决策规划，而是一种模糊系统。

举例，以一个自车变道cut in 为例：

根据博弈方表现出来的概率，在保证足够的安全裕度的前提下（即便对目标的Belief 估计出错，自车能有足够的空间进行紧急规避），进行试探和施压。



7. 基于混合估计的博弈模态Belief update

~~2.4.3 level-k混合模型~~

~~在Game Theoretic Modeling of Vehicle Interactions at Unsignalized Intersections and Application to Autonomous Vehicle Control中，作者提供了一个混合模型的建模方法，将目标建模为两个level-k的平均组合：~~

$$\mathcal{R}_D(\gamma) = \frac{1}{2}\mathcal{R}(\gamma|\widehat{\gamma}_0^{\text{other}}) + \frac{1}{2}\mathcal{R}(\gamma|\widehat{\gamma}_1^{\text{other}}),$$
$$\widehat{\gamma}_D^{\text{ego}} \in \arg\max_{\gamma_i \in \Gamma} \mathcal{R}_D(\gamma),$$

~~并且在实验中发现，这样的组合建模方式并不会影响level-k推演的稳定性。因此，在进行belief update的过程中，如果仅仅需要单模态轨迹，则可以采用这种组合建模的方式更精准地刻画目标的level-k状态。~~

8. Meg下一代决策算法的初步构想

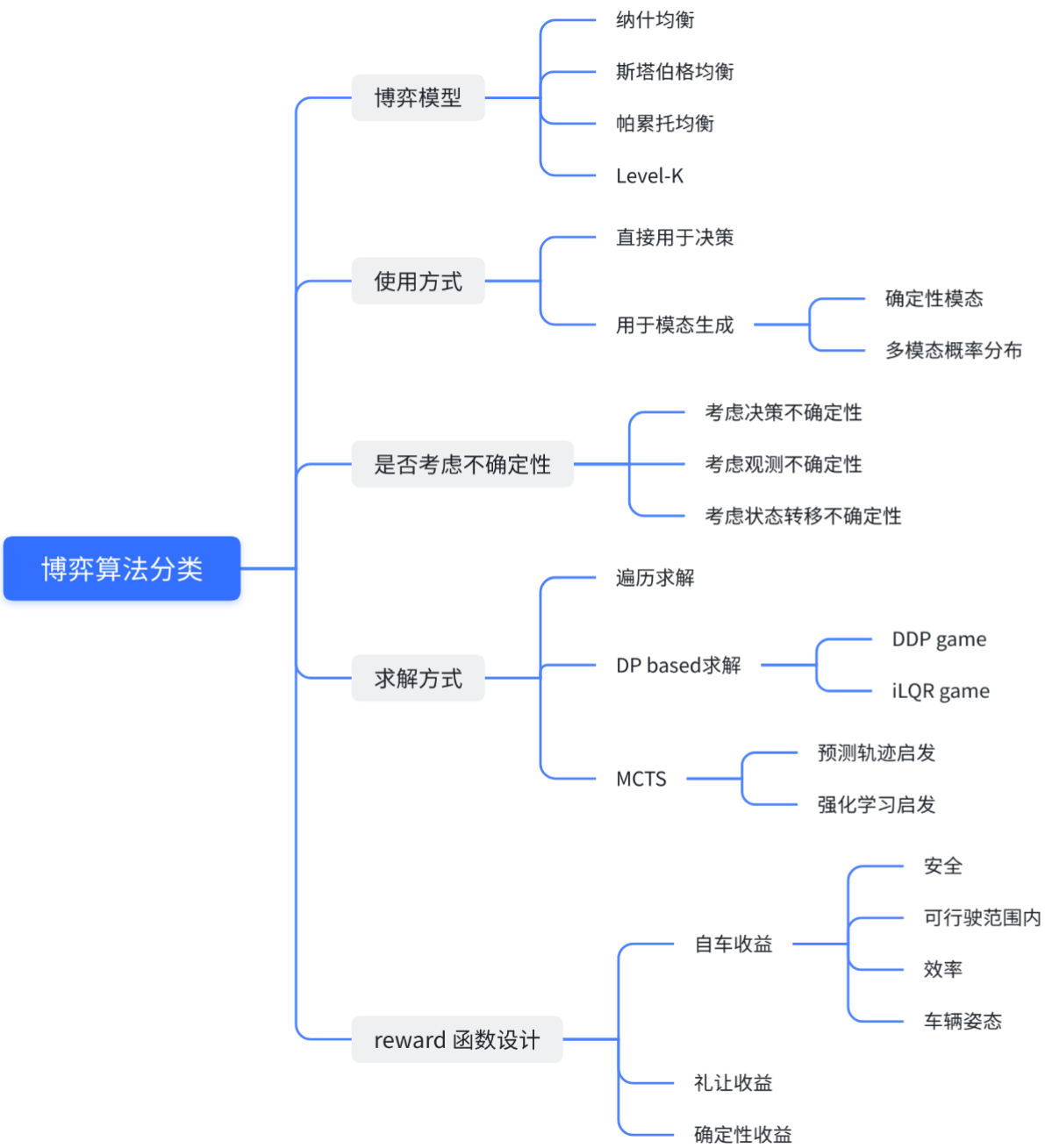
8.1 小算力平台（1个 A55核）

8.2 中算力平台 （2-3个 A55核）

8.3 大算力平台（1-2个 A78E核）

---

9. Appendix:


博弈算法分类
博弈模型
纳什均衡
斯塔伯格均衡
帕累托均衡
Level-K
使用方式
直接用于决策
用于模态生成
确定性模态
多模态概率分布
是否考虑不确定性
考虑决策不确定性
考虑观测不确定性
考虑状态转移不确定性
求解方式
遍历求解
DP based求解
DDP game
iLQR game
MCTS
预测轨迹启发
强化学习启发
reward 函数设计
自车收益
安全
可行驶范围内
效率
车辆姿态
礼让收益
确定性收益