ASYNCHRONOUS ALGORITHMS

for for exact dynamic programming this is a subject relating to the implementation of the various algorithms such as value duration in policy duration there's a lot of interest these days

对于精确动态规划来说，异步算法是一个很多算法都可以使用的技巧，比如这几天我们关心的值迭代与策略迭代

on a synchronous algorithms not only within the context of dynamic programming but in the context of of other optimization algorithms convex optimization

异步算法不仅可以使用在动态规划算法上，还可以用在其他算法上，比如凸优化

so on now what we mean by in a synchronous algorithm to give you an example in value iteration what we have is safe iterate function cost function and then we go at state one and calculate a new function then at state two and so on

我想要给你举一个异步值迭代算法的例子，从状态一，状态二，这么一直持续下去

and we do it simultaneously for all states so we produce a vector JK plus 1 by updating its components simultaneously given all the components of JK

根据现有的J\_k的值，我们对所有的状态同时改变状态价值，更新为J\_{k+1}

an asynchronous algorithm is one that picks some component let's say component 5 and that's a value iteration on that and then at another iteration picks component 10 at another iteration it picks component 3 7 and 15 and does a value iteration on that

值迭代的异步算法就是选择一个组件(借用线程的概念，叫线程比较好)，比如线程5进行一次值迭代，然后选择线程10、线程3，7，15进行值迭代

not only that the values that we use to make the value directions may not be up-to-date because they may be outdated because these iterations may be done at different processors and different computers and these computers may be having out-of-date information about the computations of other computers the computers may be out of sync

这些价值可能不是最新的，因为我们对这些价值进行值迭代计算的处理器和电脑是不同的，还在使用不是最新的信息进行迭代，即异步算法的不同线程是在独立地进行计算

also in the context of so we have distributed distributed computations setting involving many computers that exchange information but information is out of date and they not be may not be synchronized in their iterations

为了进行异步值迭代，我们已经部署了分布式计算设备，包括许多交换信息的计算机，但信息不是最新的，同时它们的迭代可能不同步

we also have a simulation context whereby the components on which we do value duration may depend on a simulation trajectory

我们还有一个模拟器，这些线程的值迭代就依赖于这些仿真路径

we let the system run we encounter state five we do a value direction on that then we move and encounter state seven we do a value direction on that so the the sequence of components that we update is determined randomly by a simulation trajectory

我们让这个系统继续运行，观察到状态5，然后进行值迭代，之后继续让系统运行，观察到状态7，再进行一次值迭代，就这样这次运行随机地在某个终止状态结束，并且得到一个仿真轨迹

and that from that context is very common in in approximate dynamic programming well we often use simulation very often the algorithms in approximate dynamic programming are are synchronous because okay we can't do them for all at the same time we have do some coop some states but not for others and that's the way the setting is

在近似动态规划中，这种做法很常见，我们通常通过仿真来进行异步近似动态规划，因为我们不能同时对所有状态进行值迭代，通过这种异步的方式我们只能对一部分状态进行更新

okay so the bottom line is that value direction and policy duration surprisingly work no matter how chaotically you implement them turns out that they are valid under the most difficult of circumstances

最重要的事情是无论执行这个算法时价值更新有多混乱，值迭代和策略迭代都是有效的，所以在很多情况下都可以使用它们

and this is due primarily to the contraction property that's involved in there and also more fundamentally the monotonicity property involved in dynamic programming now so that's the bottom line

原因是收缩性存在，更基本的收敛性也存在，这是一个很重要的内容

so let's go one step at a time here's a general framework we have a state space X and we partition it in subsets x 1 up to X M and we use a separate processor L to update the components that are in the L‘s set

有一个基本的框架，我们有一个状态空间X，把状态空间X划分为x1到x\_m这m个子集合，然后使用一个独立的处理器L更新状态子集合L中的状态

as a special case these sets in the partition may consist of a single state so you may have one processor for each different state unless partition functions of course J cost functions J conformally

这些集合的划分有一种特殊情况，就是有些子集合只有一个状态，这时候你就可以使用一个处理器计算每一个不同的状态，除非成本函数J是 标黄

so J is the component J 1 is a component of J that corresponds to X 1 and so on

J1是J中与X1相关的一个函数

so if X 1 consists of states 1 2 and 3 J 1 is J's of 1 J of 2 and J's of 3 and so on and JL is the restriction of J on the set on the on the subset of the state space

如果状态子集合X1包括状态1，2和3，那么函数集合J1包括J1，J2和J3，JL是集合J中最后的一个

now the synchronous version of value iteration has this form now T here is time okay

and there's a reason why we use a different notation here

同步值迭代的形式是这样的，t是时间，它与异步算法符号不同这就是我们使用另一套符号来表示的原因

the L component at time T plus 1 is updated by using all the components at time L at time at time T

时间t+1的函数L更新要使用集合中所有的元素时间t的信息

and and and we apply and we take the component of T at attacks X belongs to X L

使用所有子集X\_L中的x对应的时间t的函数进行更新

so this is a parallel version of the value iteration algorithm except that all the components are updated simultaneously

这是一个值迭代算法的并行视角，所有函数都被同步更新

so it's a synchronous implementation and a synchronous algorithm updates only some of the components and also uses values of components that were way back so here in this a synchronous iteration I update the component l only for t only for times in a special subset for other for other components for components for if L does not belong to this T does not belong to the subsets I lived and changed

这是一个同步算法的流程，同步算法只更新一部分函数，还返回了这些函数更新后的值，所以这次同步迭代过程中，我只更新在这个时间应该更新的函数，如果这个函数L不应该在这个时间更新，就不更新，直接把当前函数赋值给下一个时间的函数

so this allows basically two so let's say component one for l equals one component one may be updated at time 1 5 15 and 500 okay and then 5000

也就是说，函数1只在时间1，5，15和500还有5000的时候被更新

whatever our l would consist of 1 5 15 1000 but can't remember the numbers ok you understand the meaning

也就是说我只考虑时间是1，5，15和1000(此处应为5000)，我忘记刚才都说了哪些数了，你懂就行

so this is the set of times where we have components updated and there are different times for every component moreover the values that are being you in this update they are not t but rather Assam tau tau being perhaps less than T

这是我更新函数的时间，每个函数都有不同的更新时间，而且更新函数时使用的其他函数的值不是时间t时的值，而是tau时的函数值，tau必须小于t

所以这是我们更新组件的时间，每个组件都有不同的时间，而且这个更新中的值不是茶，而是Assam tau tau可能小于T

the difference between current time and the time when this component was evaluated is this and can be viewed as a delay a communication delay between processors

是一个不同于当前时间的时间，这个时间是函数被评价的时间，这种方式被叫做延迟更新，延迟指的是两个处理器通信的延迟

so there's a huge difference between this all the all the processors have to wait for its other values to become globally known and then the components can be updated in lockstep with each other here everything goes chaotically whatever components i have from other processors i can use them to do my own update if i want okay if i don't feel like doing an update i don't have to do it

这是一个很大的不同点，所有的处理器必须等待其他处理器的函数值被所有处理器都知道，也就是他们需要充分通信，然后当前的函数才能被更新。而延迟更新会造成混乱，我从其他处理器获得了当前的函数值，我可以选择不更新，也可以选择更新，这样每一个处理器使用的值都不能保证是最新的，这也是异步算法和同步算法的最大不同点

ONE-STATE-AT-A-TIME ITERATIONS

okay now a special case of this an important special case is when you update one component at a time update want to state for state 5 now and then get the results known to other processors and update estate 7 then a state 1 and so on so only one state at a time rather than all states at once

有一个比较重要的特殊情况，就是每次只更新一个函数，比如你现在在状态5并且直到其他函数的值，就只更新状态5，然后跳到状态7，就只更新状态7，然后是状态1，之类的，每次只更新一个状态的价值，而不是每次更新所有状态的价值

so we assume that there are n states a separate process for each state in no delays and we generate a sequence of states somehow perhaps by simulation however every state is supposed to be generated infinitely often and then a synchronous value iteration does an update only for one state the state that happens to be happens to be generated by this trajectory

假设有n个状态，一个独立的处理器处理每一个状态，他们的价值没有延迟，我们通过仿真生成一个状态序列，假设每一个状态都能够被生成无数次，同步值迭代每次只更新这条轨迹上被生成的当前状态

a special case where I go through all the states cyclically first update for 1 then 2 up to n and then back to 1 and so on this is a well-known method this is called the Gauss Seidel version of value iteration and it turns out it's faster than the ordinary value duration method okay

有一个特殊的情况就是周期性地访问所有状态，首先更新状态1，然后更新状态2，状态3，状态4直到状态n，然后再回头更新状态1，这是一种很出名的方法，叫做高斯-赛德尔迭代法这种值迭代方法被证明比原始的值迭代快得多

Gauss Seidel start with state 1 to a value duration use the results to do a value iteration for state 2 3 and so on that's different than doing a value iteration for all states simultaneously

高斯赛德尔迭代从状态1开始，使用其他状态的价值进行值迭代，然后使用状态1的结果对状态2进行值迭代，然后状态3，一直进行下去，这是一种与同步对所有状态进行值迭代的方法不同的方法

okay so this gives you an idea that perhaps a synchronous iteration not only works but also it may work faster and indeed that can be shown in some lab a convincing way and also in practice

这给你了一个想法，同步迭代不仅可以工作，还可以工作的比其他方法更快，这被一些实验室用可信的方法证明了，实践中也证明了它的快速性

okay now when you have an algorithm that's so wild like this

现在你有了一种非常疯狂的算法

that it worked why should it work

它能够工作，那么它为什么它工作呢

it turns out that it does work and there is a theorem that dates back now to I don't know maybe 30 years

大概在30年前就已经有证明，显示它能够工作了

which gives you a pretty good idea why what makes the such algorithms work

这个证明告诉你为什么同步算法能够让值迭代和策略迭代这样的算法工作

the bottom like as I said is that value iteration and also policy duration still work on implemented synchronous

就像我说的那样，值迭代和策略迭代一直能够在同步迭代时工作，

by the way what would be an a synchronous version of policy duration

in policy direction we don't have just updates of costbut we have updates of policies as well

顺便说一下什么是同步策略迭代就是策略迭代过程中不只更新策略的评价值，即成本，还更新策略

so example I processor one that controls state one updates policy okay just as policy then processor two updates updates at state two its cost then another processor updates either policy or cost or groups of processors updater policy circles

举个例子，处理器1控制状态1更新策略，只更新策略，然后处理器2更新状态2的成本

然后处理器更新策略或者成本，或者一组处理器循环更新策略

totally a synchronous updates of the policies the local policies and the local cost and again communication to other processors with delays

总之，异步策略更新更新局部策略和局部成本，然后把有延迟的成本和策略互相通信

ASYNCHRONOUS CONV. THEOREM I

now you need a little bit of a modification in the ordinary version of policy direction for this to work fully but basically it does work now what I'm going to do is give you a theorem that allows you to analyze that it nations that are a synchronous

现在你需要对普通版本的策略迭代做一点修改才能让异步算法完全发挥作用，但是它现在上基本可以工作，我想要做的是给你一个定理，它允许你分析异步迭代过程

first of you all you need a certain assumption you need an assumption that every processor updates infinitely often

首先你需要一个假设成立，这个假设就是每一个处理器都能够进行无限次数的更新

ok if a processor stops updating then you can ever hope for the algorithm to work

如果算法停止更新，你希望这时算法生效了

the second is that the communication delays do not stay finite so as time progresses to infinity then the tau times also go to infinity even though there may be a gap forever that we be the processors keep using up more up more and more up-to-date components

第二个假设时延迟沟通信息不会在有限步内停止，即他会一直互相传递(延迟的)信息，迭代随着时间进行并趋于无穷，tau同样趋于无穷即使tau与时间t永远会有gap，但我们还是希望处理器能够使用比较新的信息来更新策略与成本

so that's the basic assumption

这就是这两个基本的假设

and here's the theorem

下面是我要告诉你的定理

suppose that you have a map in T think of this as the value as the map in T of dynamic program it's actually more general than that

假设你有一个动态规划的映射，这是一个很常见的映射

supposedly has a unique fixed point J star

假设这个映射存在一个唯一的不动点

and suppose that you can find a sequence of sets of functions s of 0 s of 1 s 2 and so on

假设你能找到一个序列的函数集合s0，s1，s2等等

that are nested they are shrinking and they have two properties

他们是嵌套的并且在不断缩小

the first property is that if you have a sequence of functions with JK being the corresponding set SK

第一个性质时如果你有一个与s\_k相关的序列函数J\_k

then then then such a sequence converges point-wise to j star

然后这个序列逐点收敛到J\*

that basically says that you have a sequence of sets that serve like so luckily up no function and they're shrinking in at the intersection is this J star

这基本上说你有一系列的服务，就像幸运地没有任何功能，他们在十字路口萎缩是这个J星

and you have the following property that if you start with in the set SK applying the T mapping to all components simultaneously takes you within the smaller set

然后你就有了这两个性质，如果你从集合s\_k开始，使用映射T对所有元素同步进行映射，你可以获得一个更小的集合

so this is a condition that asserts that the the the value iteration Albert the the fixed point iteration using T converges you start out from the big set in one iteration you get one set inside and so on all the way to the intersection

所以这是一个断言条件，在值迭代中不断使用映射T，你可以从一个比较大的集合随着迭代逐渐获得这个大的集合里面的小集合，直到收敛到一个最里面的集合

now the key additional assumption that you need which which guarantees are synchronous convergence is that presets are not just nested but they are boxes they're partition brought boxes of component sets and they are and they have and in that's that's the the the key thing

有一个额外的假设可以保证同步算法收敛，这个假设就是集合不仅是嵌套的，而且是分区的

if you have these two assumptions then the conclusion is that for every J function every starting function that's within s 0 the sequence generated by the a synchronous algorithm converges to J star

如果你有这两个假设，结论就是你从任意一个s0的成本函数J出发，这个序列一定会再同步算法的作用下收敛到J\*

ASYNCHRONOUS CONV. THEOREM II

okay perhaps this a little bit too long but let me illustrate by means of a figure there are two assumptions here okay

这两个假设可能有点长，我要用图来解释一下这两个假设

and the second assumption the box condition says that these sets are half box structure like this

第二个假设-盒子条件说这些集合都是这种矩形结构的

so f0 is this outside box s1 is another box within it s2 is another box within that and these boxes are nested within each other and have as intersection J star

s0是最外面的这个盒子，s1是s0里面的一个盒子，s2是s1里面的一个盒子，

这些盒子是逐层嵌套的并且有一个共同的点J\*

the box s0 this assumes just two processors

假设盒子s0有两个处理器

the box s0 constants or Cartesian product of s1 which is for the first component J1 and and s2 which is for the second component J2

盒子s0有两个元素s\_1和s\_2，通过笛卡儿积得到s(k)，s\_1是第一个元素成本函数是J\_1，s\_2是第一个元素成本函数是J\_2

J vectors consists of two components here because this two-dimensional picture

向量J包括两个元素，因为图片有两个维度

and what we what our assumptions say is that if you have a J that's within Sk this box here then T sub J brings you within SK plus 1 this is the synchronous convergence condition and the box condition is of course that the sets SK have this structure

假设你现在处于S(k)，有成本函数J，这样使用映射T，则TJ可以得到S(k+1)这就是同步收敛条件，同样box条件要求集合S(k)具有这个结构

so these are the assumptions you have a sequence of sets of this type that's all that that you need

现在有另一个假设需要有一个集合的序列，

and the convergence mechanism is that you can do iterations on any one component but you get independent improvement of other components

现在你可以在每次迭代的时候一个元素一个元素地改进他们，而且你可以发现你改善某元素的时候不会影响到其他元素，现在你需要直到这个方法的收敛性

so let me illustrate this by starting at this vector J

我要从这个向量J开始解释

suppose that processor one does an iteration

假设处理器1进行了一次迭代

then the second component is not going to change just the first component is going to change and it's going to get within this range okay because that's the synchronous convergence condition

第二个元素没有改变，因为收敛条件，只有第一个元素在这个范围内变化

now processor one may do many iterations but no matter how many it does it will still stay within here

现在一号处理器执行了很多次迭代，但是无论迭代多少次，他的值都在这附近

at the first time that processor do will do an iteration it will bring you into the into into within this interval here it will get into the smaller set as k plus 1

处理器第一次进行一次迭代的时候，就可以让解进入里面的集合，也就是内层更小的集合k+1

so you have this iterations going on but the first time that the first time that you have any direction from all processors you end up in the better set and then in the better set and so on

所以第一次所有处理器都完成过一次迭代的时候，你就可以获得一个更好的集合，下一次都完成一次迭代的时候，继续获得更好的集合，就这么一直持续下去

I'm not sure I'm explaining this very well I'm doing this in a carry in perhaps you don't have much background

我不确定有没有把这个内容解释清楚，因为这需要一些背景知识

but it's not a difficult proof to understand if you understand these figures here

但是如果你能理解这个图，那么他们的证明就不会很难理解

now this is a general convergence theorem for a synchronous iterations and it applies dynamic programming because the boxes that you have in dynamic programming are spheres associated with the maximum norm

这是一个一般性的同步迭代收敛定理，它可以应用在动态规划中因为动态规划中的box是与最大化范数相关的球体

the maximum norm within for which T is a contraction has these spheres as as the spheres are pub are nested and each one of these spheres is scaled by the discount factor alpha relative to the proceedings fears

T是压缩映射，最大化norm的这些球体是逐层嵌套的，这些球体被折扣因子alpha缩放

so by taking these boxes to be sup-norm contract since fears the theorem applies and it's actually a very simple argument to see that

the argument for a synchronous convergence of policy direction is a little bit more complicated but still based on similar ideas

同步策略迭代的收缩性证明有一点复杂，但是思路还是比较简单

Q&A

okay I'm going to stop here and I can take your questions this is sort of takes us to the other edges of the theory and and we're not going to revisit it but I want you to be aware of the fact that when we talk about our synchronous algorithms there's something like this going on that makes things work questions yes the first condition the assumption that when you start with a J within one set then T so J will bring you to the next set that guarantees convergence in a synchronous fashion so if you were to apply the directional component simultaneously then from one box you'd get to the second box the third box and so on in this yeah the boxes converge to ones to one point that's the assumption in this in this theorem however there is a more general version of this theorem where the box is converged to a set of solutions okay that's right okay yeah your question has to do with whether what happens if you have not just a single fixed point but you have multiple fixed points then you have a whole set of solutions and the assumption would be then that the intersection of all these boxes is the set of solutions but there is no guarantee that there will be convergence to a single fixed point it maybe that you may be wandering around the periphery of the set of solutions and converge to the set rather than converge to a single point yeah that's true any other questions yes if we want to so approximate suggest that we have to construct okay okay the question is what happens if you have approximations and instead of considering exact value duration consider approximate value duration or even worse randomize kind of algorithm involving samples in which case the T mapping contains uncertainty contains stochastic stochastic element okay the best view of convergence has to be modified in that case the classical proof of V of convergence of Q learning I'm sorry the classical view of convergence of Q learning involves an argument like that but more sophisticated to take into account the presence of uncertainty so to give you an idea the assumption the synchronous convergence assumption that from J tfj brings you to the next set has to be relaxed and and with something like that eventually with sufficient number of samples you're going to get into a smaller set however it may be that just a single iteration will not bring you to the set but with many iterations over time you will get to the set the box ok so is the Box random no the box is not random but the algorithm involves random elements so it may be wandering around but you can prove in the case of Q learning that with probability it will get into the next box and then with probability we'll get to the second box and so on so there is a more complex version of this argument that involves an algorithm so cap stick iterative algorithm that's what I'm saying the boxes will go stay the same they're going to be soup norm spheres around the optimal solution ok this is actually the paper of the proofs convergence of Q learnings and actually very very sophisticated and it's hard for me to explain the values no one says but at it's at the heart of the proof it's this kind of picture that that is that underlies that yes in a bit so we're here now whether the optimal valerie's it's a single value [Music] okay yeah that's a good question how do we know that the intersection of this box is a single point the set of solutions a single point well in some cases we know that it's a single point for example if the mapping T is a contraction mapping the soup normal interaction mean has a unique fixed point in this balls converge to the unique fixed point if the mapping T is just monotone it may happen that you have multiple fixed points and then you need to be to be more careful it's also possible to have okay in the case of monotone mappings you may be able to you don't have a contraction but you may be able to show by other means [Music] the number of the areas I think your question has to do with the co-operative question of whether the solution set is connected or not in dynamic programming the set of solutions of balance equation is either a unique fixed point or else it has some nice structure in other words it is the set of it can be characterized in alpena as the set of all functions that lie above a certain floor function something like this you don't really seldom do you have where it is i don't know of cases where you have isolated solutions at in in space and i think that i think however this is true for exact dynamic programming for approximate dynamic programming you can have very wild set of attraction points of different algorithms you may have things like local minima that are very complicated in which the structure of which are not and is not understood that present convergence to the assumption I think this RL the second one when vehicles to infinity and this cow also goes to infinity that means every state or every region has to be to be updated meaning not terminating the finite number of iteration so I think intentionally it's very difficult to because some state may be trapped I mean the value of may be trapped in some subset of the state meaning if I have a long evaluation in a finite number of iterations and then okay I think I understand your question let me rephrase your question what you're saying is that is that suppose that you use simulation to pick the states at which you update the algorithm how do we know that we're going to visit all states if we can't do if we're not sure that we can visit all states then this algorithm will not work you have to be able to guarantee that all states over an infinite amount of time will be visited infinitely often this is this is frequently an assumption in various approximate dynamic programming algorithms so how do you enforce that okay I'm not sure that there's a foolproof recipe warrant yes okay so what is the practical significance of all this the practical significance is that just about everything in dynamic programming it revolves around a value duration policy duration even though my viewpoint here is a little abstract in mathematical that doesn't mean that the algorithms do not apply to practical problems there's a vast array of problems for which these algorithms are used and in many of these problems you have you want to use parallel computation or you the assimilation driven and I mentioned to learning earlier which is definitely a practical algorithm so so that's the reason we want we want to look at practical algorithms that have very wide applicability but we want to do them from a theoretical point of view in the most economical way at least that's my style I'd like to do them in a unified way so I don't have to prove many theorems about many different problems I'd like to prove just a few theorems that cover many problems ok but some legitimate question you know why do we have to go to this kind of abstraction or what's the meaning of all this do I have to go through all this this seems very difficult very legitimate question to ask any other questions okay so next time in the next lecture we are going to to go into approximations and the next lecture is going to be an orientation lecture it's an overview of the field of approximations and then in the second week next week we're going to focus on specific aspects of the field of approximate dynamic programming algorithms and we're going to go in more depth in those at that time okay [Applause]