بسم الله الرحمن الرحیم

سیستم‌های چندعاملی

درس ۲۵

حل مسئله‌ی توزیع‌شده

جستجوی چندعاملی

Distributed Problem-Solving: Multiagent Searching

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/mas

حل مسئله با جستجو

SOLVING PROBLEMS BY SEARCHING

راه‌حل بسیاری از مسائل در هوش مصنوعی به «**جستجو**» وابسته است.

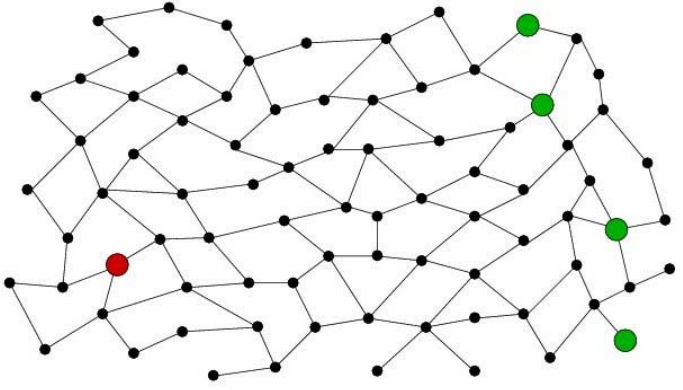
راه‌حل (نتیجه‌ی جستجو):
دنباله‌ای از کنش‌ها که اجرای آن عامل را از حالت آغازین به حالت هدف می‌رساند.

فضای حالت

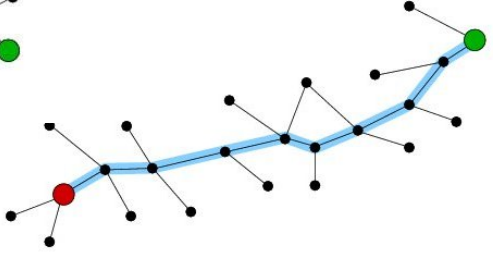
STATE SPACE

فضای حالت: مجموعه‌ی همه‌ی حالت‌های ممکن در مسئله و رابطه‌های آنها
در قالب یک گراف نمایش داده می‌شود.
مجموعه‌ی حالت‌های دسترس‌پذیر از حالت آغازین با حداقل یک دنباله از کنش‌ها

گراف فضای حالت:



راه حل:
مسیری از حالت آغازین به حالت نهایی



فرمول‌بندی مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله

مؤلفه‌های پنج‌گانه‌ی تعریف مسئله

حالت آغازین
Initial State
حالتی که عامل از آن شروع می‌کند.

کنش‌های ممکن
Available Actions
کنش‌های ممکن عامل در هر حالت s : $\text{ACTIONS}(s)$

مدل گذر
Transition Model
در هر حالت s ، هر کنش a چه می‌کند؟ : $\text{RESULT}(s,a)$

آزمون هدف
Goal Test
تعیین کننده‌ی اینکه حالت جاری هدف است یا خیر:
صریح (explicit): بیان مجموعه حالات هدف ضمنی (implicit): بیان ویژگی هدف

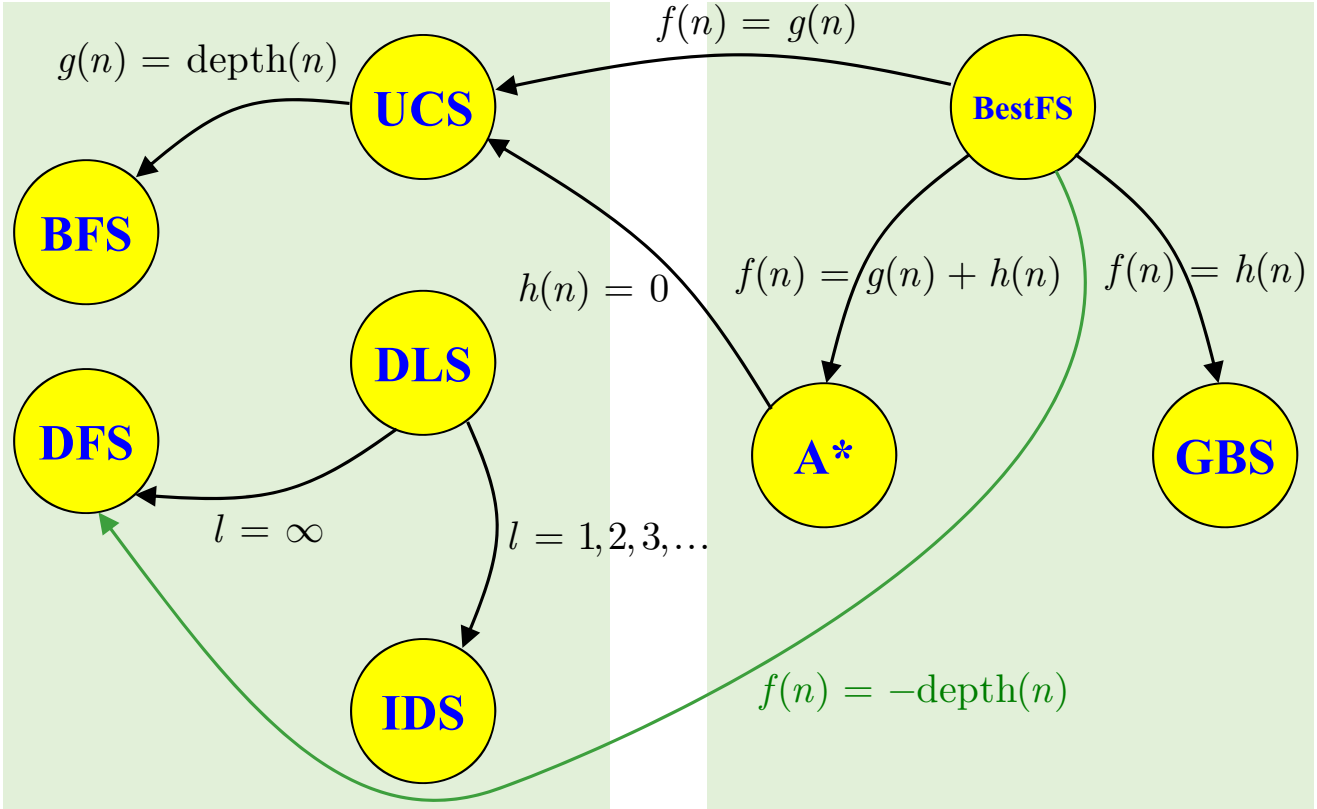
تابع هزینه‌ی مسیر
Path Cost Function
تابعی که به هر مسیر یک هزینه‌ی عددی نسبت می‌دهد.
هزینه‌ی گام (step cost): هزینه‌ی گذر از یک حالت به حالت دیگر با یک کنش $c(s,\ a,\ s')$

فضای حالت مسئله *problem state-space*

مجموعه‌ی حالت‌های دسترس‌پذیر از حالت آغازین با حداقل یک دنباله از کنش‌ها

روش‌های جستجوی کلاسیک «تک‌عاملی»

روابط میان روش‌ها



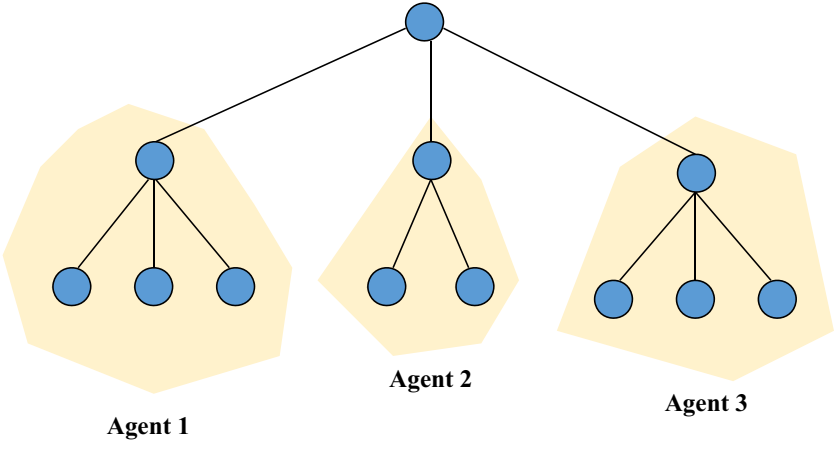
حل مسئله‌ی توزیع‌شده / جستجوی چندعاملی

DISTRIBUTED PROBLEM-SOLVING: MULTIAGENT SEARCHING

می‌توان فرآیند جستجو برای حل مسئله را توسط چند عامل انجام داد

هر عامل بخشی از فضای حالت را جستجو می‌کند؛
راه‌حل از طریق تجمیع محاسبات محلی حاصل می‌شود.

محاسبات محلی می‌توانند به‌صورت ناهمگام *(Asynchronous)* و همروند *(Concurrent)* انجام شوند.

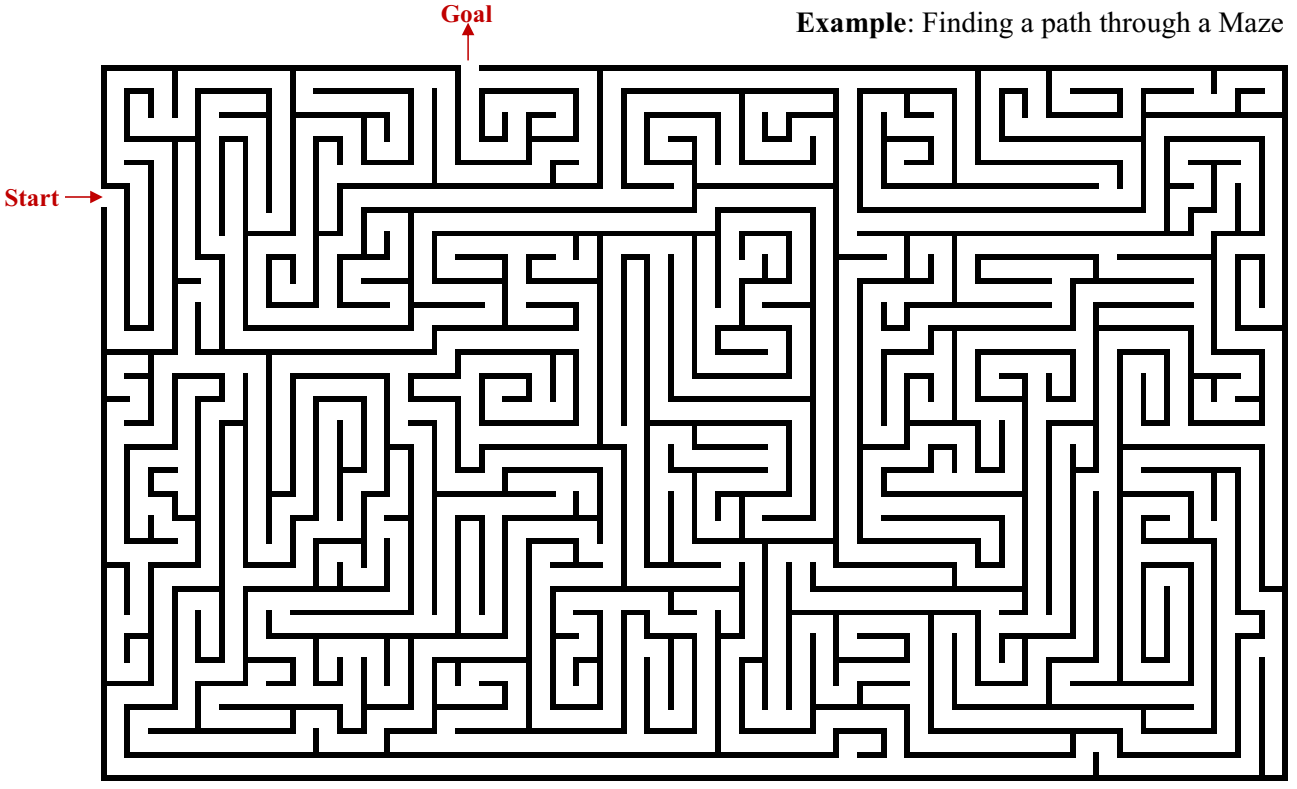


مسائل مسیریابی

مثال

PATH FINDING PROBLEMS

Example: Finding a path through a Maze

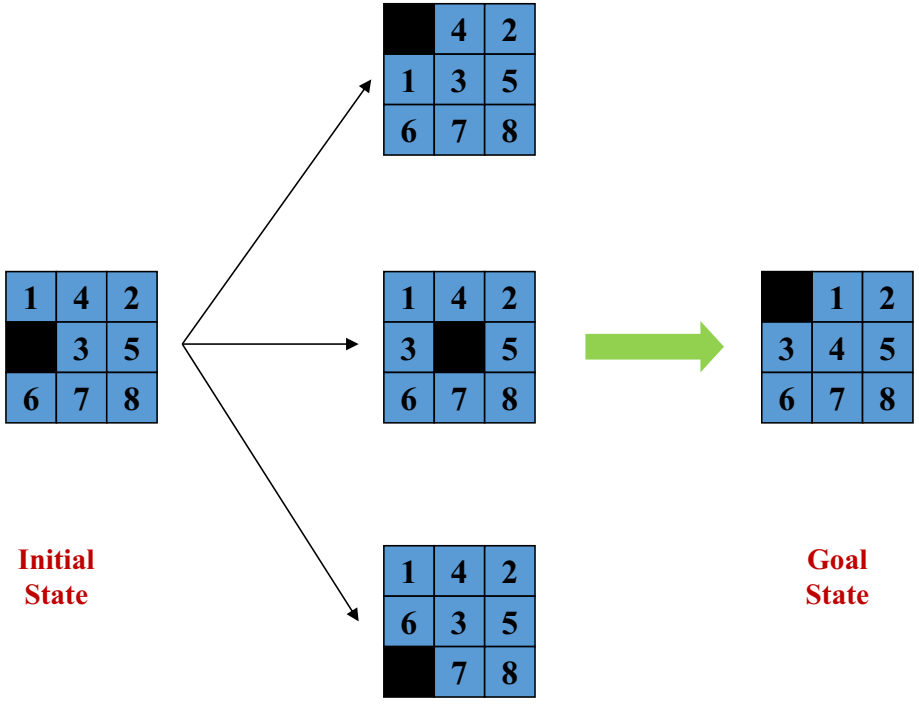


مسائل مسیریابی

مثال

PATH FINDING PROBLEMS

Example: Solving the 8-puzzle problem



مسئله‌ی مسیریابی

تعریف صوری

PATH FINDING PROBLEM: FORMAL DEFINITION

یک مسئله‌ی مسیریابی از مؤلفه‌های زیر تشکیل می‌شود:

$$(N, L, S, G, W)$$

N	**مجموعه‌ی گره‌ها** *Set of Nodes*	هر گره یک **حالت** را بازنمایی می‌کند.
L	**مجموعه‌ی پیوندهای جهت‌دار** *Set of Directed Links*	هر پیوند، یک کنش قابل انجام توسط عامل را نشان می‌دهد.
S	**حالت آغازین** *Initial State*	حالت شروع یکتا
G	**مجموعه‌ی حالت‌های هدف** *Set of Goal States*	چند حالت هدف
W	**مجموعه‌ی وزن‌ها** *Set of Weights*	به هر پیوند یک وزن منتسب می‌شود: هزینه‌ی کنش / فاصله

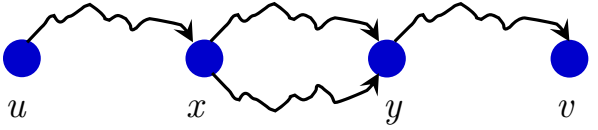
همسایه‌ی یک گره، گره‌ای است که پیوند جهت‌دار بین آنها وجود دارد.

اصل بهینگی

PRINCIPLE OF OPTIMALITY

اصل بهینگی
Principle of Optimality

یک مسیر بهینه است اگر و فقط اگر
هر قطعه‌ی آن مسیر بهینه باشد.



اگر این کوتاه‌ترین مسیر بین u و v باشد:

آن‌گاه این کوتاه‌ترین مسیر بین x و y است:

استفاده از برنامه‌ریزی پویا برای حل یک مسئله‌ی بهینه‌سازی فقط در صورت برقراری **اصل بهینگی** ممکن است.

الگوریتم‌های مسیریابی توزیع‌شده

برنامه‌ریزی پویای ناهمگام
Asynchronous Dynamic Programming

جستجوی A* بی‌درنگ یادگیرنده
Learning Real-Time A (LRTA*)*

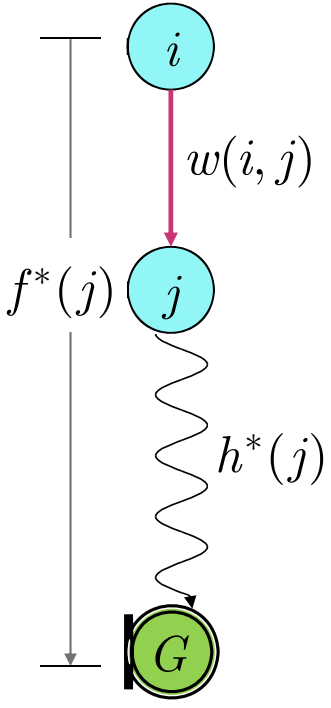
جستجوی A* بی‌درنگ
Real-Time A (RTA*)*

جستجوی مقصد متحرک
Moving Target Search (MTS)

جستجوی دوطرفه‌ی بی‌درنگ
Real-Time Bidirectional Search (RTBS)

برنامه‌ریزی پویای ناهمگام

ASYNCHRONOUS DYNAMIC PROGRAMMING



$h^*(i)$ = کوتاه‌ترین فاصله از گره‌ی i تا هدف

$w(i,j)$ = هزینه‌ی پیوند بین گره‌ی i و گره‌ی j

$f^*(j)$ = کوتاه‌ترین فاصله از گره‌ی i تا گره‌ی هدف از طریق گره‌ی همسایه j

$$f^*(j) = w(i,j) + h^*(j)$$

طبق **اصل بهینگی** داریم:

$$h^*(i) = \min_j f^*(j)$$

برنامه‌ریزی پویای ناهمگام،
h^* را با تکرار محاسبات محلی برای هر گره
محاسبه می‌کند

برنامه‌ریزی پویای ناهمگام

الگوریتم

ASYNCHRONOUS DYNAMIC PROGRAMMING

برای هر گره‌ی i یک عامل i فرآیند را اجرا می‌کند.

هر عامل مقدار $h(i)$ را ثبت می‌کند (تخمین مقدار $h^*(i)$: مقدار آغازین دلخواه مثلاً ∞ ، برای هدف : $h(G) = 0$)

هر عامل می‌تواند به مقدار h گره‌های همسایه‌اش مراجعه کند (از طریق حافظه‌ی اشتراکی یا گذر دادن پیام)

procedure ASYNCHDP (node i)
if i *is a goal node* **then**
  $h(i) \leftarrow 0$
else
  initialize $h(i)$ arbitrarily (e.g., to ∞ or 0)
repeat
  forall *neighbors* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $h(i) \leftarrow \min_j f(j)$

عامل i برای گره‌ی i :

(محاسبه‌ی محلی) به‌ازای هر همسایه‌ی j :

$$f(j) \leftarrow w(i,j) + h(j)$$

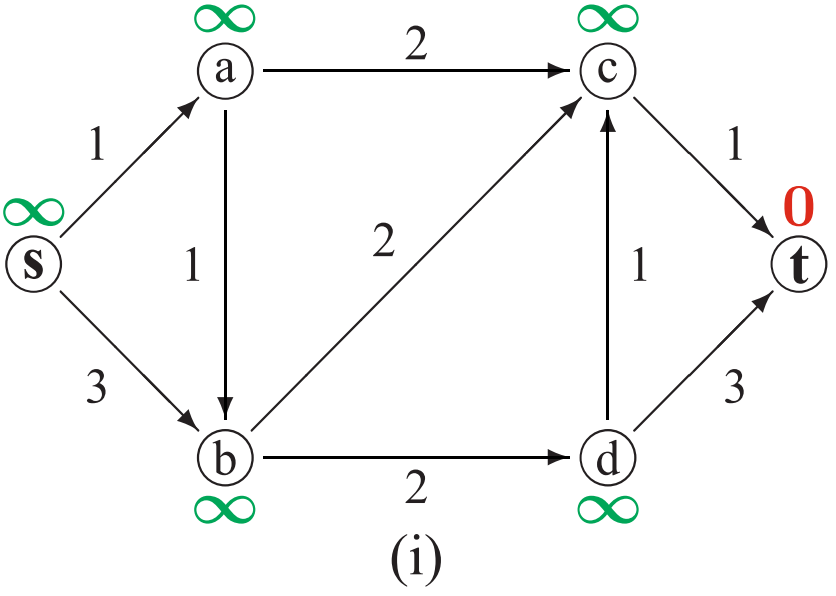
(به‌هنگام‌سازی تخمین) :

$$h(i) \leftarrow \min_j f(j)$$

برنامه‌ریزی پویای ناهمگام

الگوریتم: مثال (۱ از ۳)

ASYNCHRONOUS DYNAMIC PROGRAMMING

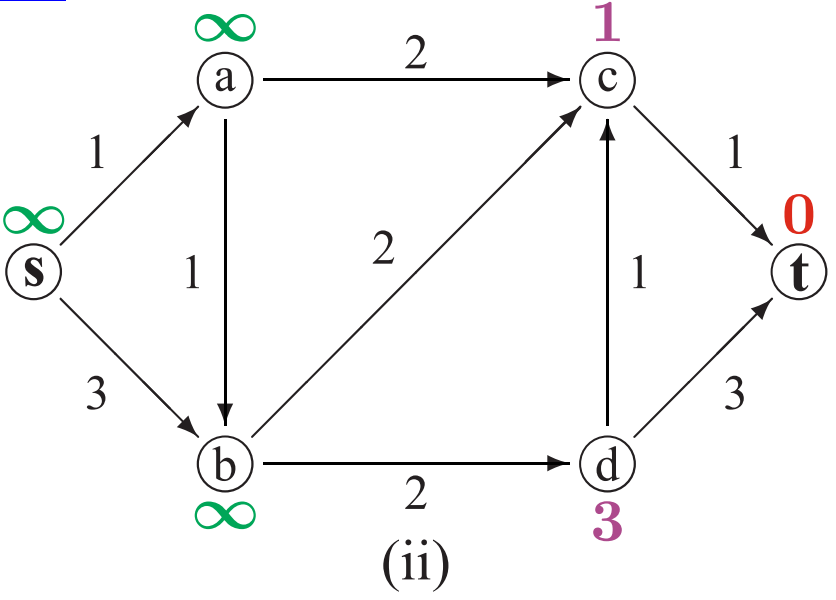


procedure ASYNCHDP (node i)
if i *is a goal node* **then**
  $h(i) \leftarrow 0$
else
  initialize $h(i)$ arbitrarily (e.g., to ∞ or 0)
repeat
  forall *neighbors* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $h(i) \leftarrow \min_j f(j)$

برنامه‌ریزی پویای ناهمگام

الگوریتم: مثال (۲ از ۳)

ASYNCHRONOUS DYNAMIC PROGRAMMING

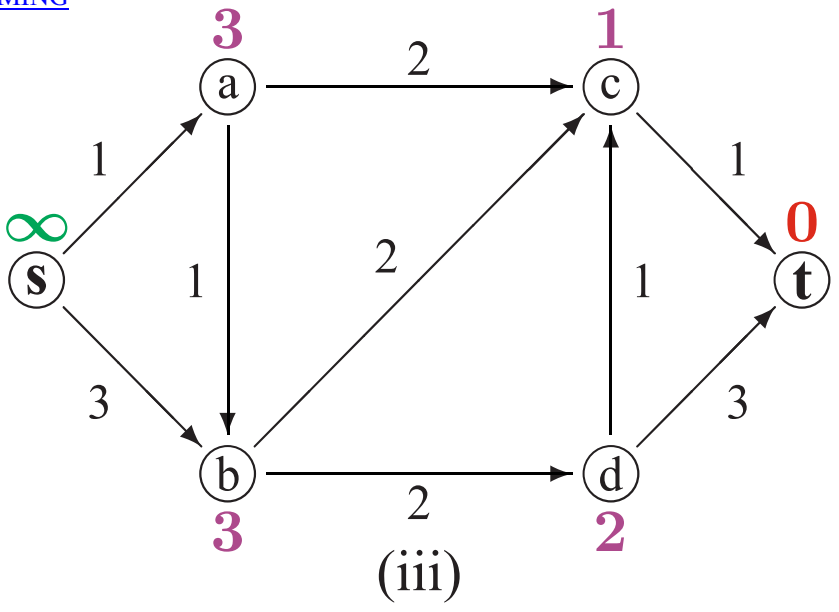


procedure ASYNCHDP (node i)
if i *is a goal node* **then**
  $h(i) \leftarrow 0$
else
  initialize $h(i)$ arbitrarily (e.g., to ∞ or 0)
repeat
  forall *neighbors* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $h(i) \leftarrow \min_j f(j)$

برنامه‌ریزی پویای ناهمگام

الگوریتم: مثال (۳ از ۳)

ASYNCHRONOUS DYNAMIC PROGRAMMING



procedure ASYNCHDP (node i)
if i *is a goal node* **then**
  $h(i) \leftarrow 0$
else
  initialize $h(i)$ arbitrarily (e.g., to ∞ or 0)
repeat
  forall *neighbors* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $h(i) \leftarrow \min_j f(j)$

برنامه‌ریزی پویای ناهمگام

ویژگی‌های الگوریتم

ASYNCHRONOUS DYNAMIC PROGRAMMING

بهینگی *Optimality*	تمامیت *Completeness*
آیا این استراتژی لزوماً راه‌حل بهینه را پیدا می‌کند؟	آیا تضمینی وجود دارد این الگوریتم راه‌حلی را در صورت وجود بیابد؟
بله	**بله**

مشکل: به تعداد گره‌ها، عامل نیاز داریم!

A* بی‌درنگ یادگیرنده

LEARNING REAL-TIME A* (LRTA*)

در اصل یک الگوریتم تک‌عاملی (متمرکز) است.

جستجوی برخط (On-line):
عامل باید جستجو و کنش را یک‌درمیان انجام دهد.

procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i, j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

هر عامل: تکرار سه مرحله‌ی زیر:

(نگاه به جلو) به‌ازای هر همسایه‌ی j:
$f(j) \leftarrow w(i, j) + h(j)$

(به‌هنگام‌سازی تخمین):
$h(i) \leftarrow \min_j f(j)$

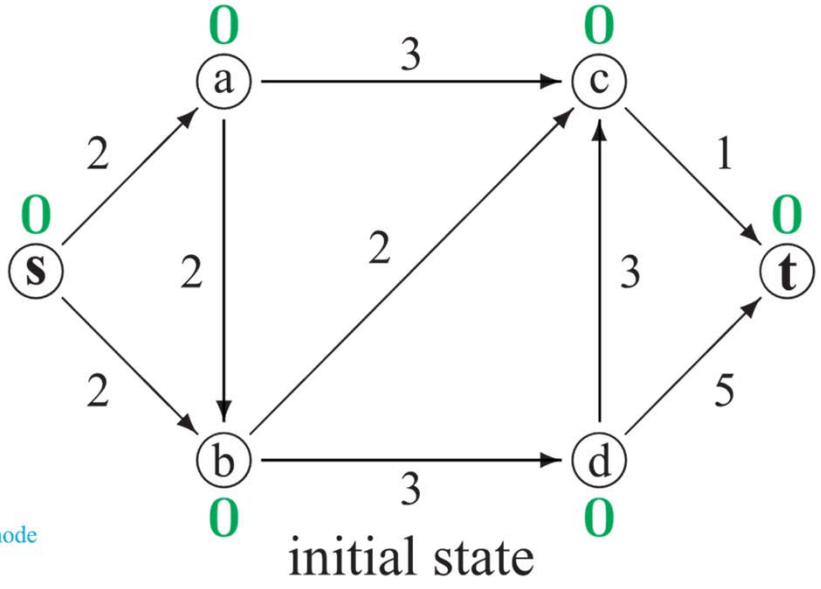
(انتخاب کنش):
رفتن به همسایه‌ی j با می‌نیمم $f(j)$

h never overestimates the distance to the goal, that is, $h(i) \leq h^*(i)$

A* بی‌درنگ یادگیرنده

مثال (۱ از ۵)

LEARNING REAL-TIME A* (LRTA*)

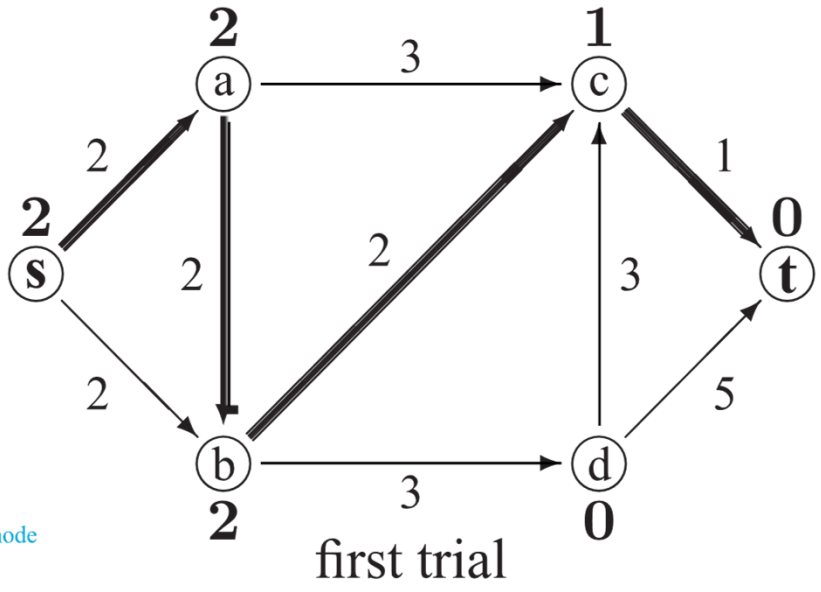


procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

A* بی‌درنگ یادگیرنده

مثال (۲ از ۵)

LEARNING REAL-TIME A* (LRTA*)

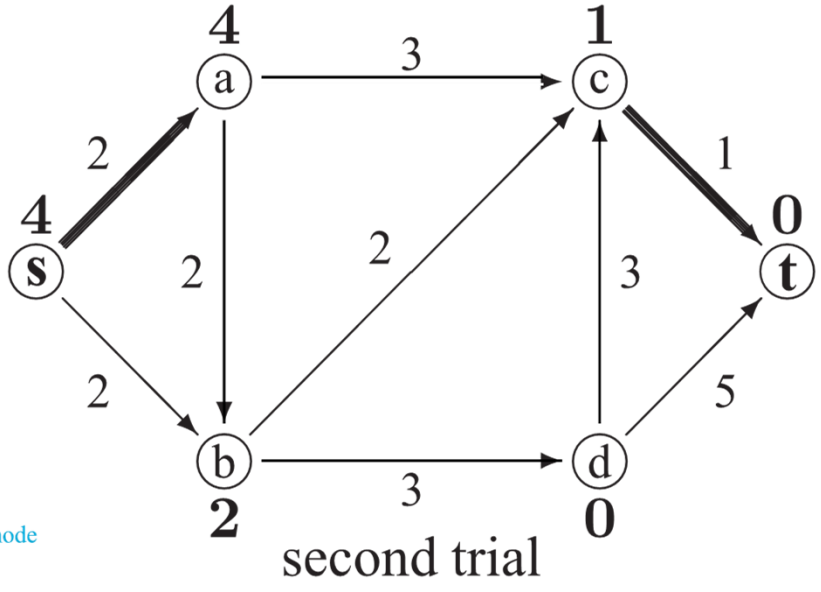


procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

A* بی‌درنگ یادگیرنده

مثال (۳ از ۵)

LEARNING REAL-TIME A* (LRTA*)

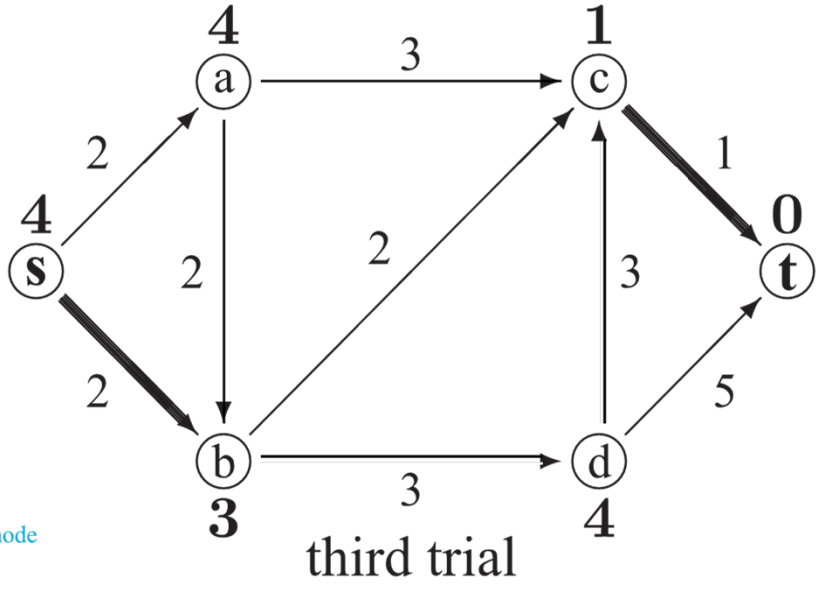


procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

A* بی‌درنگ یادگیرنده

مثال (۴ از ۵)

LEARNING REAL-TIME A* (LRTA*)

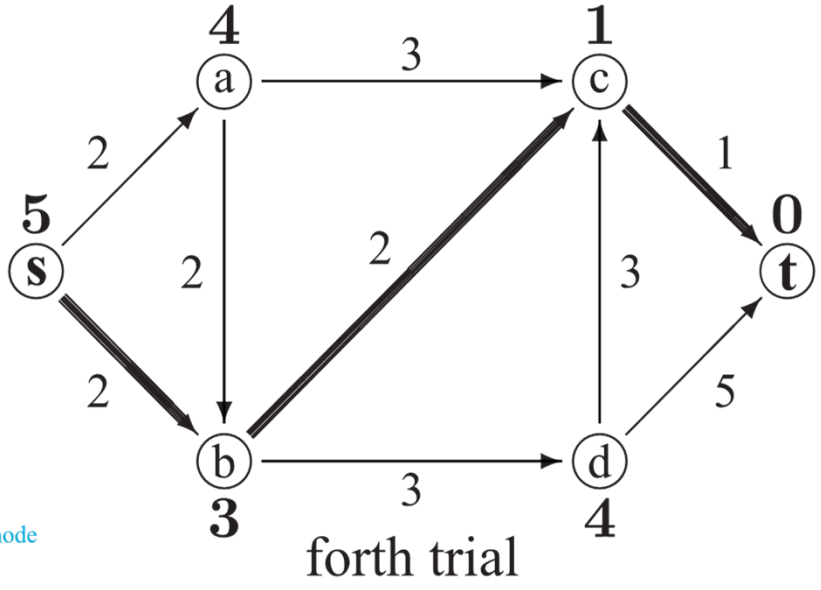


procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

A* بی‌درنگ یادگیرنده

مثال (۵ از ۵)

LEARNING REAL-TIME A* (LRTA*)



procedure LRTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i,j) + h(j)$
  $i' \leftarrow \arg\min_j f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

A* بی‌درنگ یادگیرنده

خصوصیات

LEARNING REAL-TIME A* (LRTA*)

- The h-values never decrease, and remain admissible.
- LRTA* terminates; the complete execution from the start node to termination at the goal node is called a *trial*.
- If LRTA* is repeated while maintaining the h-values from one trial to the next, it eventually discovers the shortest path from the start to a goal node.
- If LRTA* find the same path on two sequential trials, this is the shortest path. (However, this path may also be found in one or more previous trials before it is found twice in a row. Do you see why?)

A* بی‌درنگ یادگیرنده

ویژگی‌های الگوریتم

LEARNING REAL-TIME A* (LRTA*)

بهینگی *Optimality*	تمامیت *Completeness*
آیا این استراتژی لزوماً راه‌حل بهینه را پیدا می‌کند؟	آیا تضمینی وجود دارد این الگوریتم راه‌حلی را در صورت وجود بیابد؟
بله	**بله**
نیازمند تکرار برای رسیدن به بهینگی: اگر تخمین‌های آغازین قابل قبول باشند، مقادیر یادگرفته شده به مقادیر واقعی مسیر بهینه همگرا می‌شود.	به این شرط که ۱) تعداد گره‌ی دارای وزن مثبت، متناهی باشد ۲) از هر گره تا هدف مسیری موجود باشد. ۳) تخمین‌های آغازین نامنفی باشد.

A* بی‌درنگ یادگیرنده‌ی چندعاملی

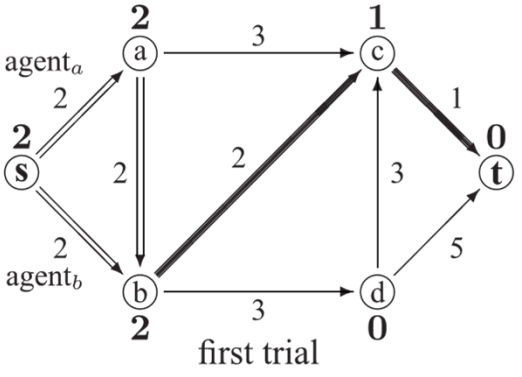
MULTIAGENT LEARNING REAL-TIME A* (LRTA*(n))

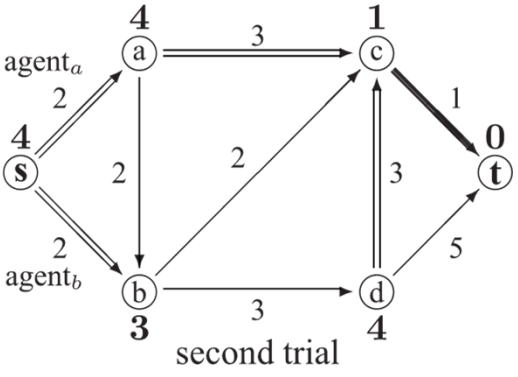
LRTA* is a centralized procedure. However, we note that rather than have a single agent execute this procedure, one can have multiple agents execute it. The properties of the algorithm (call it LRTA*(n), with n agents) are not altered, but the convergence to the shortest path can be sped up dramatically. First, if the agents each break ties differently, some will reach the goal much faster than others. Furthermore, if they all have access to a shared h-value table, the learning of one agent can teach the others. Specifically, after every round and for every i, $h(i) = \max_j h_j(i)$, where $h_j(i)$ is agent j's updated value for $h(i)$. Figure 2.5 shows an execution of LRTA*(2)—that is, LRTA* with two agents—starting from the same initial state as in Figure 2.4. (The hollow arrows show paths traversed by a single agent, while the dark arrows show paths traversed by both agents.)

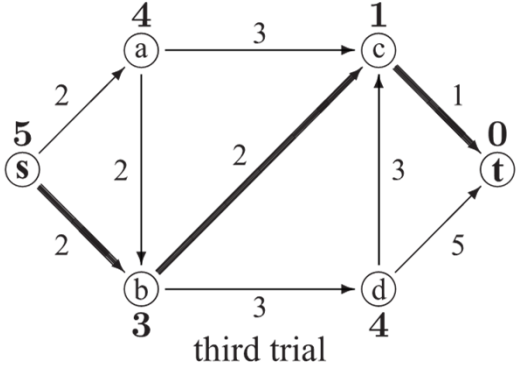
A* بی‌درنگ یادگیرنده‌ی چندعاملی

مثال

MULTIAGENT LEARNING REAL-TIME A* (LRTA*(*n*))







A* بی‌درنگ

REAL-TIME A* (RTA*)

دقیقاً مشابه LRTA* اما به‌جای کوچک‌ترین مقدار هزینه‌ی همسایه، دومین کوچک‌ترین مقدار در نظر گرفته می‌شود.

جستجوی برخط (On-line):
عامل باید جستجو و کنش را یک‌درمیان انجام دهد.

هر عامل: تکرار سه مرحله‌ی زیر:

(نگاه به جلو) به‌ازای هر همسایه‌ی j:
$f(j) \leftarrow w(i, j) + h(j)$

(به‌هنگام‌سازی تخمین):
$h(i) \leftarrow \text{second-min}_j\, f(j)$

(انتخاب کنش):
رفتن به همسایه‌ی j با می‌نیمم $f(j)$

procedure RTA*
$i \leftarrow s$ // the start node
while i *is not a goal node* **do**
  foreach *neighbor* j **do**
    $f(j) \leftarrow w(i, j) + h(j)$
  $i' \leftarrow \arg \text{second-min}_j\, f(j)$ // breaking ties at random
  $h(i) \leftarrow \max(h(i), f(i'))$
  $i \leftarrow i'$

h never overestimates the distance to the goal, that is, $h(i) \leq h^*(i)$

RTA* نسبت به LRTA* کارآمدی بهتری دارد، زیرا میزان بیشتری اکتشاف باید انجام بدهد.

A* بی‌درنگ

ویژگی‌های الگوریتم

REAL-TIME A* (RTA*)

بهینگی *Optimality*	تمامیت *Completeness*
آیا این استراتژی لزوماً راه‌حل بهینه را پیدا می‌کند؟	آیا تضمینی وجود دارد این الگوریتم راه‌حلی را در صورت وجود بیابد؟
بله	**بله**
نیازمند تکرار برای رسیدن به بهینگی: اگر تخمین‌های آغازین قابل قبول باشند، مقادیر یادگرفته شده به مقادیر واقعی مسیر بهینه همگرا می‌شود.	به این شرط که ۱) تعداد گره‌ی دارای وزن مثبت، متناهی باشد ۲) از هر گره تا هدف مسیری موجود باشد. ۳) تخمین‌های آغازین نامنفی باشد.

جستجوی مقصد متحرک

MOVING TARGET SEARCH (MTS)

حالت هدف می‌تواند در طول جستجو تغییر پیدا کند.

مثال:
یک ربات متحرک می‌خواهد به یک ربات متحرک دیگر برسد

ربات مقصد می‌تواند:
- به‌طور **همکارانه** سعی کند به ربات حل‌کننده‌ی مسئله برسد.
- به‌طور فعال از ربات حل‌کننده‌ی مسئله **دوری** کند.
- **مستقل** از ربات حل‌کننده‌ی مسئله حرکت کند.

برای تضمین موفقیت:
عامل حل‌کننده‌ی مسئله باید بتواند سریع‌تر از مقصد حرکت کند.

جستجوی مقصد متحرک

MOVING TARGET SEARCH (MTS)

این الگوریتم تعمیم *LRTA است.

الگوریتم جستجوی مقصد متحرک

- یک هیوریستیک **واحد** را برای فاصله تا هدف مقصد نگهداری نمی‌کند.
- در عوض، سعی می‌کند اطلاعات هیوریستیک را برای **هر** محل بالقوه‌ی مقصد به‌دست آورد.
  - پس MTS یک ماتریس از مقادیر هیوریستیک را نگهداری می‌کند؛
    که بازنمایی‌کننده‌ی تابع $h(x,y)$ برای همه‌ی جفت حالت‌های x و y است.
  - این ماتریس در هر حرکت عامل حل‌کننده‌ی مسئله و عامل مقصد به‌هنگام می‌شود.

جستجوی مقصد متحرک

MOVING TARGET SEARCH (MTS)

x_i = موقعیت فعلی عامل حل‌کننده‌ی مسئله
x_j = موقعیت همسایه‌ی عامل حل‌کننده‌ی مسئله
y_i = موقعیت فعلی عامل مقصد
y_j = موقعیت همسایه‌ی عامل مقصد

فرض می‌کنیم که
همه‌ی یال‌های گراف،
هزینه‌ی یکسان دارند.

When Problem-Solver Moves:

for each neighbor x_j calculate $h(x_j, y_i)$

update $h(x_i, y_i) \leftarrow \max(h(x_i, y_i), \min_{x_j'}\{h(x_j', y_i) + 1\})$

move to $x_j = \arg\min_{x_j} h(x_j, y_i)$

$x_i \leftarrow x_j$

When Target Moves:

for new position (neighbor) y_j calculate $h(x_i, y_j)$

update $h(x_i, y_i) \leftarrow \max(h(x_i, y_i), \{h(x_i, y_j) - 1\})$

move to y_j

$y_i \leftarrow y_j$ (new goal for problem-solver)

جستجوی مقصد متحرک

ویژگی‌های الگوریتم

MOVING TARGET SEARCH (MTS)

بهینگی *Optimality*	تمامیت *Completeness*
آیا این استراتژی لزوماً راه‌حل بهینه را پیدا می‌کند؟	آیا تضمینی وجود دارد این الگوریتم راه‌حلی را در صورت وجود بیابد؟
خیر	**بله**
روش حریصانه	اگر (۱) مقصد از بعضی از حرکت‌هایش صرف نظر کند، (۲) تابع هیسوریستیک قابل قبول باشد و (۳) مسیری موجود باشد.

جستجوی دوطرفه‌ی بی‌درنگ

Real-Time Bidirectional Search (RTBS)

دو عامل حل مسئله به‌صورت فیزیکی از حالت‌های آغازین و هدف شروع می‌کنند و به سمت یکدیگر حرکت می‌کنند.

جستجوی برخط (On-line):
عامل‌ها باید جستجو و کنش را یکدرمیان انجام دهد.

گام‌های زیر به صورت تکراری اجرا می‌شوند
تا دو عامل حل‌کننده‌ی مسئله در فضای مسئله یکدیگر را ملاقات کنند:

استراتژی کنترل Control Strategy	حرکت پیش‌رو Forward Move	حرکت پس‌رو Backward Move
انتخاب حرکت پیش‌رو یا پس‌رو	حل‌کننده‌ی مسئله از مرحله‌ی آغازین شروع می‌کند و به‌سمت حل‌کننده‌ی مقابل حرکت می‌کند.	حل‌کننده‌ی مسئله از مرحله‌ی هدف شروع می‌کند و به‌سمت حل‌کننده‌ی مقابل حرکت می‌کند.

جستجوی دوطرفه‌ی بی‌درنگ

انواع

Real-Time Bidirectional Search (RTBS)

دو دسته الگوریتم RTBS

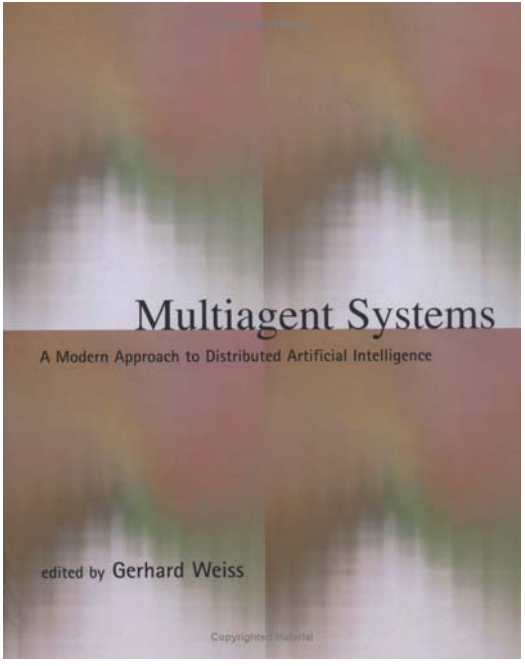
جستجوی دوطرفه‌ی بی‌درنگ
Real-Time Bidirectional Search (RTBS)

متمرکزشده Centralized RTBS	**جداشده** Decoupled RTBS
○ بهترین کنش از میان همه‌ی حرکت‌های ممکن دو عامل حل‌کننده‌ی مسئله انتخاب می‌شود.	○ دو عامل حل‌کننده‌ی مسئله تصمیم‌های خود را به‌طور مستقل اخذ می‌کنند.
○ استراتژی کنترل انتخاب می‌کند که کدام‌یک از دو حل‌کننده‌ی مسئله باید اجرا کند (بر اساس بهترین کنش مشخص‌شده)	○ استراتژی کنترل به طور متناوب حل‌کننده‌های پیش‌رو و پس‌رو را اجرا می‌کند.
○ دو الگوریتم RTBS متمرکز می‌تواند بر اساس LRTA* یا RTA* پیاده‌سازی شود.	○ برای پیاده‌سازی می‌تواند از جستجوی مقصد متحرک MTS استفاده کرد.

حل مسئله‌ی توزیع‌شده
جستجوی چندعاملی

منابع

منبع اصلی



Gerhard Weiss (ed.),
Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,
MIT Press, 1999.
Chapter 4

4 Search Algorithms for Agents

Makoto Yokoo and Toru Ishida

4.1 Introduction

In this chapter, we introduce several search algorithms that are useful for problem solving by multiple agents. Search is an umbrella term for various problem solving techniques in AI. In search problems, the sequence of actions required for solving a problem cannot be known *a priori* but must be determined by a trial-and-error exploration of alternatives. Since virtually all AI problems require some sort of search, search has a long and distinguished history in AI.

The problems that have been addressed by search algorithms can be divided into three classes: path-finding problems, constraint satisfaction problems, and two-player games.

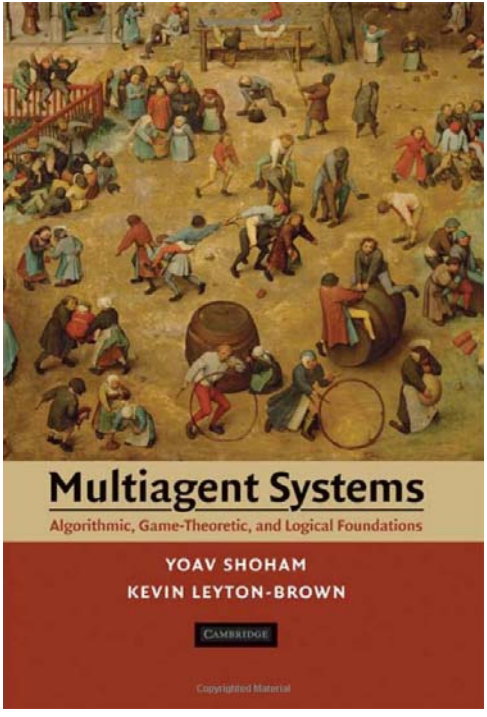
A typical example of the first class, i.e., path-finding problems, is a puzzle called the *n-puzzle*. Figure 4.1 shows the 8-puzzle, which consists of eight numbered tiles arranged on a 3×3 board (in a generalized case, there are $n = k^2 - 1$ tiles on a $k \times k$ board). The allowed moves are to slide any tile that is horizontally or vertically adjacent to the empty square into the position of the empty square. The objective is to transform the given initial configuration to the goal configuration by making allowed moves. Such a problem is called a path-finding problem, since the objective is to find a path (a sequence of moves) from the initial configuration to the goal configuration.

A constraint satisfaction problem (CSP) involves finding a goal configuration rather than finding a path to the goal configuration. A typical example of a CSP is a puzzle called 8-queens. The objective is to place eight queens on a chess board (8×8 squares) so that these queens will not threaten each other. This problem is called a constraint satisfaction problem since the objective is to find a configuration that satisfies the given conditions (constraints).

Another important class of search problems is two-player games, such as chess. Since two-player games deal with situations in which two *competitive* agents exist, it is obvious that these studies have a very close relation with DAI/multiagent systems where agents are competitive.

On the other hand, most algorithms for the other two classes (constraint satisfaction and path-finding) were originally developed for single-agent problem solving.

منبع کمکی



Yoav Shoham and Kevin Leyton-brown,
Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations,
Cambridge University Press, 2009.
Chapter 2

2 *Distributed Optimization*

In the previous chapter we looked at distributed ways of meeting global constraints. Here we up the ante; we ask how agents can, in a distributed fashion, optimize a global objective function. Specifically, we consider four families of techniques and associated sample problems. They are, in order:

- Distributed dynamic programming (as applied to path-planning problems).
- Distributed solutions to Markov Decision Problems (MDPs).
- Optimization algorithms with an economic flavor (as applied to matching and scheduling problems).
- Coordination via social laws and conventions, and the example of traffic rules.

2.1 Distributed dynamic programming for path planning

Like graph coloring, path planning constitutes another common abstract problem-solving framework. A path-planning problem consists of a weighted directed graph with a set of n nodes N, directed links L, a weight function $w : L \mapsto \mathbb{R}^+$, and two nodes $s, t \in N$. The goal is to find a directed path from s to t having minimal total weight. More generally, we consider a set of goal nodes $T \subset N$, and are interested in the shortest path from s to any of the goal nodes $t \in T$.

This abstract framework applies in many domains. Certainly it applies when there is some concrete network at hand (e.g., a transportation or telecommunication network). But it also applies in more roundabout ways. For example, in a planning problem the nodes can be states of the world, the arcs actions available to the agent, and the weights the cost (or, alternatively, time) of each action.

2.1.1 Asynchronous dynamic programming

Path planning is a well-studied problem in computer science and operations research. We are interested in distributed solutions, in which each node performs a local computation, with access only to the state of its neighbors. Underlying our