www.Prozhe.com

بسم الله الرحمن الرحیم

www.Prozhe.com

www.Prozhe.com

دانشگاه پیام نور مرکز سنندج

پایان‌نامه کارشناسی

رشته کامپیوتر گرایش نرم‌افزار

عنـوان پـــایان‌نامه:

# الگوریتم مسیریابی زنبور عسل BeeHive

استاد راهنما:

عبدالله ناصری

دانشجو:

محمد عرفان درویشی

تاریخ ارائه: شهریور ۱۳۹۰

www.Prozhe.com

www.Prozhe.com


## چکیده:

سیستم‌های طبیعی مختلف به ما یاد می‌دهند که ارگانیسم خارجی بسیار ساده توان تولید سیستم‌هایی با قابلیت انجام کارهایی بسیار پیچیده را دارند. حشرات اجتماعی (زنبور عسل، زنبور معمولی، مورچه‌ها و موریانه‌ها) برای میلیون‌ها سال بر روی کره زمین زندگی کرده‌اند، آشیانه‌های مختلف ساخته‌اند و آذوقه خود را سازمان دهی کرده‌اند. پویاگرایی جمعیت حشارت نتیجه‌ای از عملکردها و تعاملات بین حشرات با یکدیگر و با محیط اطراف است. این تعامل بر اساس یکسری عوامل فیزیکی و شیمیایی امکان پذیر است. مثالی برای چنین رفتارهایی، حرکت خاص مورچه‌ها در هنگام جمع آوری محصول است. مثال دیگر ترشح هورمون فنومون در مورچه‌ها که موجب راه گذاری برای سایرین می‌شود. این سیستم‌های ارتباطی بین حشرات موجب به وجود آمدن مقوله ای به نام "هوش اشتراکی" شده‌است.

زنبورها فعالیت‌های خوراک‌جویی‌شان را بصورت اجتماعی سازمان‌دهی می‌کنند، زنبورهای خوراک‌جو فاصله و کیفیت منابع غذایی را با یک نوع رقص به سایر زنبوران اطلاع می‌دهند. در این پایان‌نامه ما یک الگوریتم مسیریابی نو را معرفی می‌کنیم، *BeeHive* الهام گرفته از روش‌ها و رویه‌های زنبورهای عسل می‌باشد. در این الگوریتم، عامل زنبور از میان یک منطقه بسیار وسیع و بی‌انتها حرکت می‌کند، که ناحیه کاوش(*foraging zones*) نامیده می‌شود. اطلاعات زنبورها در مورد وضعیت شبکه برای به هنگام‌سازی جداول مسیریابی تحویل داده می‌شود. کندو اطلاعات محلی یا ناحیه‌ای را به ترتیب حساب می‌کند. از میان شبیه سازی‌های انجام شده نشان می‌دهیم که *BeeHive* یک الگوریتم پیشرفته را انجام می‌دهد.


www.Prozhe.com

www.Prozhe.com

# فهرست مطالب


۱- بخش اول ۸

۱-۱ مقدمه ۹

۱-۲ الگوریتم *AntNet* ۱۰

۱-۳ مراحل مختلف اجرای الگوریتم *AntNet* ۱۱

۱-۴ توصیف یک مثال ۱۳

۲- بخش دوم ۱۷

۲-۱ پروتکل *OSPF* ۱۸

۲-۱-۱ روش سیل آسا ۱۸

۲-۱-۲ شبکه *AS* ۱۸

۲-۱-۳ ناحیه یا *Area* ۱۸

۲-۱-۴ ستون فقرات *OSPF* ۱۹

۲-۱-۵ مسیریاب *ABR* ۱۹

۲-۲ الگوریتم *LS* برای پیدا کردن بهترین مسیر در شبکه ۱۹

۲-۳ کوتاهترین مسیر ۱۹

۲-۳-۱ تجزیه و تحلیل الگوریتم *Shortest Path* ۲۲

۲-۴ انواع بسته‌های *OSPF* ۲۳

۳- بخش سوم ۲۶

۳-۱ کلونی زنبور در طبیعت ۲۷

۳-۲ مدل عامل زنبور عسل ۲۹

۳-۲-۱ بسته‌بندها ۲۹

۳-۲-۲ شناسایی کنندگان ۲۹

۳-۲-۳ خوراک جویان ۳۰

۳-۲-۴ حرکت دسته‌جمعی ۳۰

۳-۳ معماری *BeeHive* ۳۰

۳-۳-۱ تالار بسته‌بندی ۳۱

۳-۳-۲ ورودی ۳۲

۳-۳-۳ سالن رقص ۳۳


www.Prozhe.com

www.Prozhe.com


۴-۳ الگوریتم *BeeHive* ۳۵

۵-۳ جداول مسیریابی در الگوریتم *BeeHive* ۳۷

۴- بخش چهارم ۴۱

۱-۴ محیط شبیه سازی برای *BeeHive* ۴۲

۲-۴ نتایج آزمایش ۴۴

۱-۲-۴ بارهای اشباع کننده (*Saturating Loads*) ۴۴

۲-۲-۴ اندازه بخش‌های کاوش ۴۵

۳-۲-۴ اندازه جدول مسیریابی ۴۵

۴-۲-۴ نقاط خطرناک (*Hot Spot*) ۴۶

۵-۴-۴ از کارافتادن مسیریاب (*Router Crash*) ۴۷

۶-۴-۴ هزینه سربار *BeeHive* ۴۷


www.Prozhe.com

www.Prozhe.com

# فهرست اشکال


شکل ۱-۱ نمونه‌ای از رفتار عامل مورچه ..... ۱۴
شکل ۲-۱ عوامل تصمیم مورچه پیشرو در تقاطع ..... ۱۵
شکل۳-۱ مورچه پیشرو مورچه پسرو را در مقصد بوجود می‌آورد ..... ۱۵
شکل۴-۱ جزئیات رفتار مورچه پیشرو ..... ۱۵
شکل۵-۱ جزئیات رفتار مورچه پسرو ..... ۱۶
شکل۶-۱ احتمالات به‌روزرسانی توسط مورچه پسرو ..... ۱۶
شکل ۱-۲ شبکه *OSPF* ..... ۱۹
شکل۲-۲ نمایش گراف ..... ۲۱
شکل ۳-۲ درخت پوشای کوتاهترین مسیر به مبدا رأس $a$ برای گراف شکل ۲-۲ ..... ۲۲
شکل ۱-۳ رفتار زنبور عسل برای خوراک‌جویی ..... ۳۰
شکل ۲-۳ دید کلی از معماری *BeeHive* ..... ۳۲
شکل ۳-۳ تالار بسته‌بندی ..... ۳۳
شکل ۴-۳ ورودی ..... ۳۴
شکل ۵-۳ سالن رقص ..... ۳۵
شکل ۶-۳ الگوریتم سیل‌آسا زنبور عسل ..... ۳۷
شکل ۴-۱ اینترنت ستون فقرات *NTTNet* ژاپن ..... ۴۳
شکل ۲-۴ رفتار الگوریتم‌ها در مقابل بارهای اشباع کننده ..... ۴۶
شکل ۳-۴ اندازه بخش‌های کاوش ..... ۴۶
شکل ۴-۴ اندازه جداول مسیریابی ..... ۴۷
شکل ۵-۴ نقاط خطرناک الگوریتم *AntNet* ، *BeeHive* ..... ۴۷


www.Prozhe.com

www.Prozhe.com

## فهرست جداول

جدول۱-۲ مراحل مختلف اجرای الگوریتم بر روی شکل ۲-۲ ..... ۲۲
جدول ۱-۳ جدول مسیریابی داخل منطقه خوراک‌جویی (*IFZ*) ..... ۳۸

## فهرست الگوریتم‌ها

الگوریتم۱-۱ *AntNet* ..... ۱۳
الگوریتم ۱-۲ برنامه تعیین کوتاهترین مسیر ..... ۲۴
الگوریتم ۱-۳ *BeeHive* ..... ۴۰
الگوریتم ۱-۴ اینترنت ستون فقرات ژاپن، شبیه سازی شده در *OMNeT++* ..... ۴۵

www.Prozhe.com

www.Prozhe.com

# بخش اول:

# الگوریتم AntNet

www.Prozhe.com

www.Prozhe.com

# بخش دوم:

# الگوریتم OSPF

www.Prozhe.com

www.Prozhe.com

# بخش سوم:

# الگوریتم BeeHive

www.Prozhe.com

www.Prozhe.com

# بخش چهارم:

# شبیه‌سازی BeeHive

www.Prozhe.com

www.Prozhe.com



## ۱-۱ مقدمه

مورچه ها در طبیعت این توانایی را دارند که به کمک مواد شیمیایی که از خود در محیط به جا می‌گذارند کوتاهترین مسیر بین منبع غذایی و لانه خود را علامت گذاری کنند. یافتن کوتاهترین مسیر تنها با استفاده از اطلاعات محلی برای شبکه های کامپیوتری بسیار مفید است. *شاندرورد* الگوریتم مسیریابی جدیدی بر پایه این روش در سال ۱۹۹۷ م. ارائه کرد. او در روش خود از عامل‌هایی شبیه مورچه استفاده کرد. در این الگوریتم هر گره در بازه‌های زمانی مشخص، یک مورچه به سمت مقصد تصادفی $d$ می‌فرستد. هرگاه مورچه به گره $i$ برسد، گره بعدی خود ($j$) را به کمک جدول مسیریابی گره $i$، به دست آورده و پس از تغییر جدول مسیریابی گره $i$، به $j$ خواهد رفت. برای تغییر جدول مسیریابی، کافیست احتمال انتخاب $j$ به عنوان گره بعدی برای مقصد $d$ افزایش یابد و احتمال انتخاب سایر همسایه‌های $i$ کاهش یابد. *Caro* و *Dorigo* نیز در سال ۱۹۹۷ م. الگوریتم *AntNet* را بر اساس روش مسیریابی مورچه ها پیشنهاد کردند. در این الگوریتم دو نوع عامل به نام مورچه پیشرو و مورچه پسرو وجود دارد. مشابه الگوریتم *شاندرورد* در بازه‌های زمانی مشخص، هر گره مورچه پیشرویی به سمت مقصدی مشخص روانه می‌کند. ولی بر خلاف الگوریتم قبلی این مورچه فقط وظیفه دارد به سمت مقصد مورد نظر پیش رفته و اطلاعاتی در مورد مسیر بدست آورد. مورچه پیشرو به مقصد که می‌رسد مورچه دیگری به نام مورچه پسرو ایجاد می‌کند و پس از انتقال اطلاعات خود به او از بین می‌رود. مورچه پسرو از همان مسیری که مورچه پیشرو طی کرده‌است به عقب بر می‌گردد و به هر گره که می‌رسد جدول مسیریابی آن را تغییر می‌دهد. مورچه پسرو با اطاعات در مورد کل مسیر، جدول گره‌ها را به روز رسانی می‌کند، بنابراین تغییرات مناسبی روی جدول‌ها اعمال می‌کند. از بین الگوریتم‌های فوق، تحقیقات متعدّی روی *AntNet* انجام شده‌است به عنوان مثال *Baran* و *Sosa* الگوریتم *AntNet* را توسعه دادند. آنها مفاهیم متعدّی به *AntNet* افزودند از جمله مقداردهی اولیه و هوشمندانه کردن جدول‌های مسیریابی، به روز رسانی هوشمندانه جدول‌های مسیریابی پس از خطادار شدن منابع شبکه، افزودن عامل نویز و کنترل تعداد مورچه‌ها در شبکه از آن جمله‌اند. *Marwaha* نیز با ارائه الگوریتمی که ترکیبی از *AODV* و *AntNet* می‌باشد، نحوه استفاده از *AntNet* را در شبکه‌های متحرک نشان داده‌است. *Subbing* و *Zemin* مفهوم تضمین کیفیت سرویس را نیز به *AntNet* افزودند.

در این الگوریتم‌های *AntNet* برای همگرایی نیاز به اطلاعات کلی شبکه می‌باشد که از طریق همسایه‌ها به مسیریاب می‌روند. جابجایی این اطلاعات سربار زیادی به شبکه وارد می‌کند. از طرفی مسیریاب‌ها کاملا وابسته به همسایه‌هایشان هستند که باعث می‌شود مقاومت کمتری نسبت به خطاها و تغییرات شبکه داشته باشند. از طرفی در مدل *AntNet* هر گره کاملا محلی عمل می‌کند، بنابراین گره‌ها برای یافتن مسیرهای بهینه باید مورچه‌های زیادی را به شبکه بفرستند تا به همگرایی برسند.

www.Prozhe.com

www.Prozhe.com



دو مسیریاب که همسایه هم هستند کاملا مستقل از هم کار می‌کنند. با وجود اینکه این استقلال به علت جلوگیری از انتشار مسیرهای نادرست یک مزیت *AntNet* تلقی می‌گردد، اما باعث می‌شود مسیریاب‌ها نتوانند از تجربیات همسایه‌های خود استفاده کنند در نتیجه رسیدن به همگرایی به تاخیر می‌افتد.

## ۲-۱ الگوریتم *AntNet*

این الگوریتم الهام گرفته از زندگی مورچه‌ها در طبیعت می‌باشد. همان طور که می‌دانیم مورچه‌ها این توانایی را دارند که به کمک مواد شیمیایی که در محیط از خود به جا می‌گذارند با یکدیگر ارتباط برقرار کرده و کارهای مشکلی را با همکاری هم انجام دهند. (مانند یافتن کوتاه‌ترین مسیر بین منبع غذایی تا لانه.)

*AntNet* با استفاده از عامل‌های متحرک، به گونه‌ای کار مورچه‌ها را شبیه‌سازی می‌کند. ماده شیمیایی فرومون [۳] در اینجا همان احتمالات موجود در جدول‌های مسیریابی است. هرگاه مورچه‌ای از مسیری عبور کند به کمک فرمول‌های مشخص، احتمال رفتن از آن مسیر افزایش خواهد یافت. در این مدل دو نوع عامل وجود دارد:

**مورچه پیشرو:** این نوع عامل وظیفه دارد که در شبکه جستجو کرده و اطلاعات لازم (شناسه گره‌های موجود در مسیرهای طی شده و زمان رسیدن به آنها) را جمع آوری کند. وقتی به مقصد خود می‌رسد عامل نوع دوم را ایجاد و خودش از بین می‌رود.

**مورچه پسرو:** این مورچه‌ها از همان مسیری که مورچه‌های پیشرو طی کرده‌اند به عقب برمی‌گردند. آنها در هنگام بازگشت، به هر گره که می‌رسند جدول مسیریابی شان را اصلاح می‌کنند.

هر گره $k$، دارای دو ساختار داده می‌باشد که توسط عامل‌ها قابل به روز رسانی هستند. ($N$ تعداد کل گره‌های شبکه است و $N_k$ تعداد همسایه‌های گره می‌باشد.) جدول مسیریابی $P_k$ دارای $N$ سطر و $N_k$ ستون می‌باشد. در این جدول برای هر زوج گره $(d,n)$، احتمال $P_{dn}$ نگهداری می‌شود که احتمال خوب بودن انتخاب گره $n$ به عنوان گره بعدی برای مقصد $d$ می‌باشد. مجموع احتمالات هر سطر یک می‌باشد.

$$\sum_{n \in NK} P_{dn} = 1, d \in [1...N], n \in [1...N_k]$$

ماتریس $j=1...N$، $M_k = [(\mu_j, \sigma_j^2, T_{best_j})]$ ، گره $k$ به ازای هر $j$ $(j=1...N)$ اطلاعات آماری فوق را نگهداری می‌کند. $\mu_j$ میانگین و $\sigma_j^2$ واریانس و $T_{best_j}$ کوتاهترین زمانی است که تاکنون

www.Prozhe.com

www.Prozhe.com



عامل‌ها برای رسیدن از $k$ به $j$ صرف کرده‌اند. این داده‌ها حافظه‌ای است از آنچه این مسیریاب تا کنون از کل شبکه دیده‌است.

## ۳-۱ مراحل مختلف اجرای الگوریتم *AntNet*

۱. در دوره‌های زمانی منظم یک مورچه پیشرو از هر گره شبکه به سمت مقصدی مشخص به حرکت در می‌آید تا یک مسیر کم هزینه به آن مقصد را کشف کند و حالت ترافیکی شبکه را نیز بررسی کند. مقصدها بنابر بار ترافیکی محلی با احتمالی که بر اساس رابطه زیر بدست می‌آید، مشخص می‌شوند.

$$probability_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}$$

$f_{sd}$ حجم داده‌ایست (تعداد بیت یا بسته) که از $s$ به $d$ رفته‌است و $probability_d$ احتمال فرستادن مورچه پیشرو از گره $s$ به مقصد $d$ می‌باشد.

۲. مورچه پیشرو در هنگام حرکت شناسه گره‌های ملاقات شده و زمان صرف شده از ابتدای حرکت تا ملاقات این گره‌ها را در یک پشته ذخیره می‌کند.

۳. هر عامل برای یافتن گره بعدی از اطلاعاتی که در جدول مسیریابی گره جاری ذخیره شده‌است و همچنین طول صف در پیوندهای مختلف استفاده می‌کند. در هر گره $k$، مورچه پیشرو گره بعدی را از بین همسایه‌هایی که تا به‌حال ملاقات نشده‌اند و در صورتی که تمام همسایه‌ها ملاقات شده باشند از بین تمام $N_k$ همسایه‌های گره $k$ ام انتخاب می‌کند. هر همسایه با احتمال $P'_{nd}$ به عنوان گره بعدی انتخاب می‌گردد. $P'_{nd}$ بر اساس فرمول زیر محاسبه می‌گردد.

$$P_{nd} = \frac{P'_{nd} + \alpha l_n}{1 + \alpha(N_k - 1)}$$

$l_n$ (پارامتر آمادگی همسایه‌ها) مقدار نرمالیزه‌ای بین $[0,1]$ می‌باشد که به $q_n$، طول صف بسته‌هایی است که از گره $k$ به همسایه‌اش $n$ فرستاده شده‌است، بستگی دارد. این ارتباط در زیر نمایش شده‌است. $\alpha$ نیز عددی است ($۰/۲ < \alpha < ۰/۵$) برای وزن دادن به $l_n$ در محاسبه $P'_{nd}$ که به صورت تجربی به‌دست آمده‌است.

$$l_n = 1 - \frac{q_n}{\sum_{n=1}^{N_k} q'_n}$$

۴. اگر یک حلقه تشخیص داده شد، مورچه گره‌های موجود در حلقه را از پشته خارج کرده و گره دیگری را برای حرکت انتخاب می‌کند.

www.Prozhe.com

www.Prozhe.com



۵. مورچه پیشرو وقتی به گره مقصد رسید، عامل دیگری که مورچه پسرو نامیده می‌شود تولید کرده و تمام حافظه و داده‌هایش را به او منتقل می‌کند و خودش از بین می‌رود. (اصلاح جدول‌های مسیریابی به عهده مورچه‌های پسرو می‌باشد.)

۶. مورچه پسرو مسیری مشابه عامل مورچه پیشرو طی می‌کند (در جهت عکس).

۷. مورچه پسرو هنگام ورود به هر گره، همچنان که در ذیل توضیح داده شده‌است ساختار داده آن گره را به روز می‌رساند.

۸. پس از رسیدن به مبدا مورچه پسرو هم از بین می‌رود. [۴]

```
t       := Current time;
t_end := Time length of the simulation;
Δt      := Time interval between ants generation;
foreach (Node)        / * Concurrent activity over the network * /
  M = Local traffic model;
  T = Node routing table;
  while ( t ≤ t_end)
    in_parallel      / * Concurrent activity on each node * /
      if ( t mod Δt = 0)
        destination_node := SelectDestinationNode (data_traffic_distribution);
        LaunchForwardAnt (destination_node, source_node);
      end if
      foreach (ActiveForwardAnt[source_node, current_node, destination_node])
        while (current_node ≠ destination_node)
          next_hop_node := SelectLink (current_node, destination_node, T, link_queues);
          PutAntOnLinkQueue (current_node, next_hop_node);
          WaitOnDataLinkQueue (current_node, next_hop_node);
          CrossTheLink (current_node, next_hop_node);
          PushOnTheStack (next_hop_node, elapsed_time);
          Current_node := next_hop_node;
        end while
        LaunchBackwardAnt (destination_node, source_node, stack_data);
        Die();
      end foreach
      foreach (ActiveBackwardAnt [source_node, current_node, destination_node])
        while (current_node ≠ destination_node)
          next_hop_node := PopTheStack();
          WaitOnHighPriorityLinkQueue (current_node, next_hop_node);
          CrossTheLink (current_node, next_hop_node);
          UpdateLocalTrafficModel (M, current_node, source_node, stack_data);
          Reinforcement:=GetReinforcement (current_node,source_node,stack_data,M);
          UpdateLocalRoutingTable (T, current_node, source_node, reinforcement);
        end while
      end foreach
    end in_parallel
  end while
end foreach
```

الگوریتم ۱-۱. *AntNet*

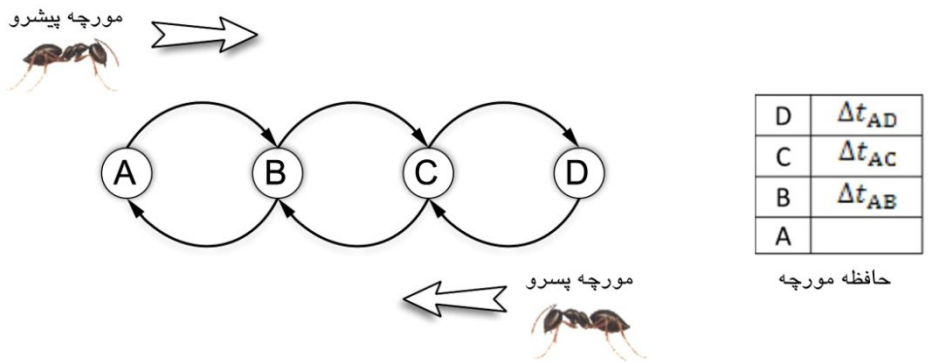
www.Prozhe.com

www.Prozhe.com



## ۱-۴ توصیف یک مثال

شبکه معین شده در شکل ۱-۱ یک مثال ساده از نحوه کار عامل مورچه را نشان می‌دهد، که هسته الگوریتم *AntNet* را تشکیل می‌دهد. مورچه پیشرو $F_{AD}$ مبدا $A$ را برای کشف یک مسیر موجه به طرف مقصد $D$ ترک می‌کند، از یک تقاطع تا تقاطع بعدی را بر مبنی یک قاعده تصمیم احتمالی حرکت خواهدکرد. رویه تصمیم (شکل ۲-۱) اساساً بر روی جدول احتمالات مواجه شدن به هر گره، حافظه هر مورچه و نحوه اتصال هر صف گره وابسته‌است و تضمین نمی‌کند که مورچه پیشرو $F_{AD}$ موقع رسیدن به مقصد در یک حلقه قرار می‌گیرد، اما مورچه می‌تواند از حافظه‌ای که در دسترس دارد، برای اجتناب از برخورد با حلقه استفاده کند. اجازه دهید فرض کنیم مورچه پیشرو برای رسیدن به مقصد $D$ مسیر $A \to B \to C \to D$ را ادامه می‌دهد. (شکل ۴-۱) وقتی که وارد مقصد $D$ می‌شود، مورچه پیشرو یک مورچه متناظر پسرو بوجود خواهد آورد، (شکل ۵-۱) مسیر $A \to B \to C \to D$ را معکوس می‌پیماید و از بین خواهد رفت. مورچه پیشرو در طول سفرش، نام تقاطع‌های جاری، زمانی که مبدا را ترک کرد، را در حافظه پشته قرار می‌دهد. محاسبه زمان سفر کردن، $t_{id}$ بوسیله مورچه پسرو بکار می‌رود، میانگین زمان سفر $\mu_d$ ، برای مقصد جاری می‌باشد. میانگین مدت مسافرت معلوم می‌کند، یک مسیر چقدر مطلوب است. وقتی وارد یک تقاطع مقصد می‌شود، یک مسیر مطلوب بین مبدا و مقصد وجود خواهدداشت. نه‌تنها مسیر بین مبدا و مقصد بهینه است بلکه بین هر تقاطع میانی و تقاطع مقصد بهینه است، برای آنکه مورچه پیشرو حافظه‌اش را به طور کامل در پشته قرار می‌دهد $S_{AD}(k)$ ، مورچه پسرو قادر است این احتمالات را به روز رسانی نماید.



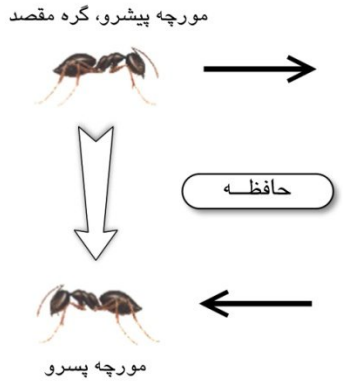

شکل ۱-۱. نمونه‌ای از رفتار عامل مورچه

مورچه پیشرو در طول مسیر $A \to B \to C \to D$ حرکت می‌کند، وقتی مورچه پیشرو به گره مقصد می‌رسید، به مورچه پسرو تبدیل می‌شود. در شکل حافظه مورچه پسرو و پیشرو نمایش داده شده‌است.

www.Prozhe.com

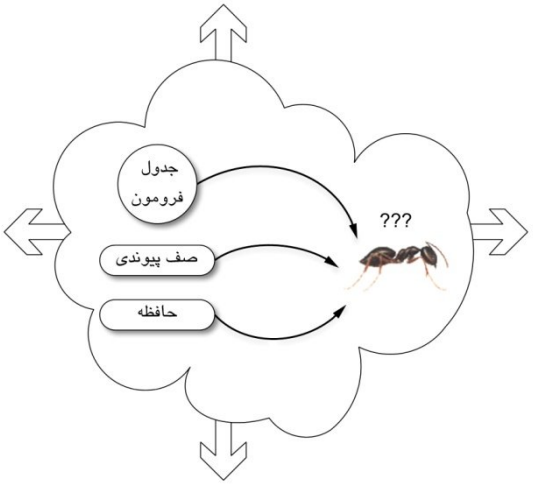
www.Prozhe.com





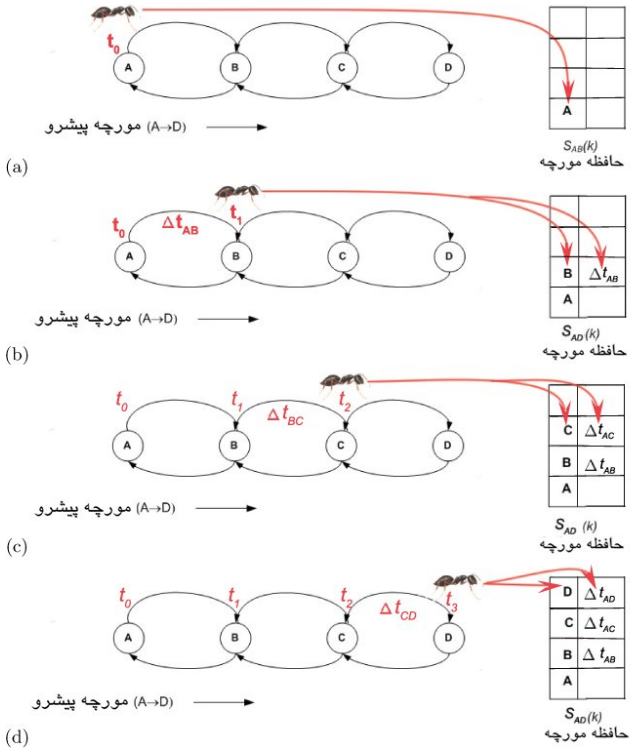

شکل۳-۱ . مورچه پیشرو مورچه پسرو را در مقصد بوجود می‌آورد.

مورچه پسرو تمام حافظه مورچه پیشرو را به ارث می‌برد.




شکل ۲-۱. عوامل تصمیم مورچه پیشرو د هر تقاطع

به طور کلی، خطی مشی تصمیمات مورچه پیشرو بوسیله جدول فرومون، وضعیت صف پیوندی (اطلاعات ذهنی)، حافظه گره های قبلا دیده شده. (برای اجتناب از حلقه در کاوش)



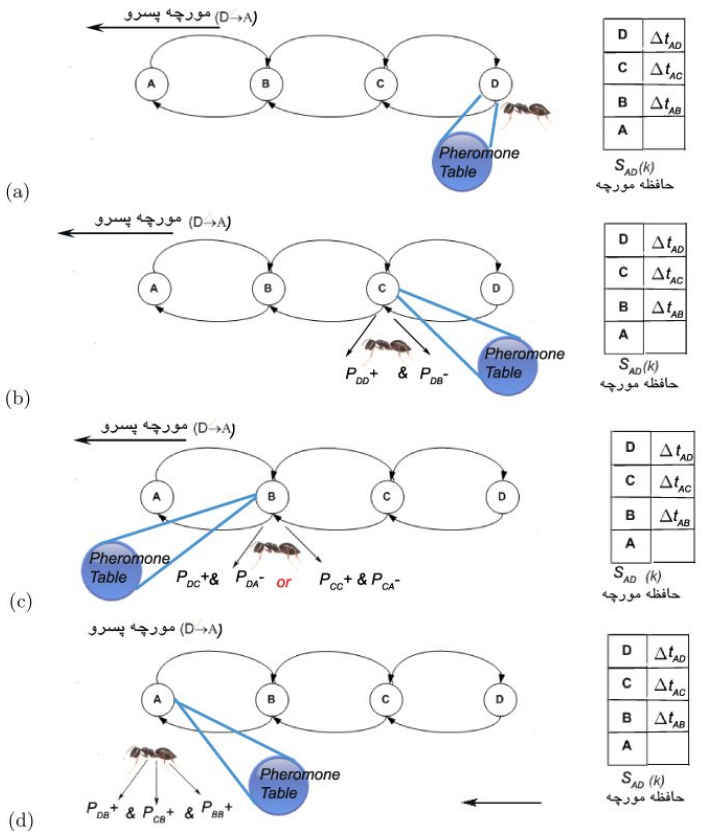

شکل۴-۱. جزئیات رفتار مورچه پیشرو

مورچه پیشرو $F_{AD}$ از مبدا $s=A$ شروع به کار می‌کند و از یک تقاطع به تقاطع دیگر حرکت می‌کند، گره $i$ ، $i \in \{A,B,C,D\}$ را در پشته $S_{AD}(k)$ به همراه زمان رسیدن $t_{Ai}$ ذخیره می‌کند.

www.Prozhe.com

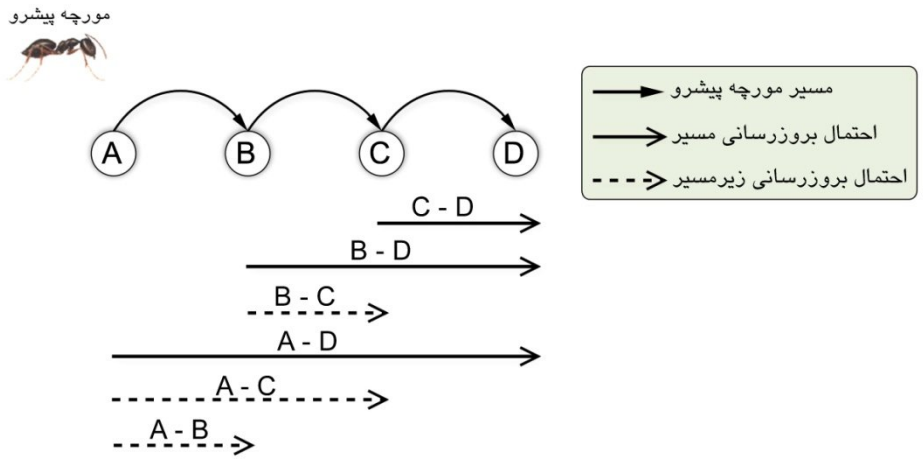
www.Prozhe.com






شکل۵-۱ . جزئیات رفتار مورچه پسرو

مورچه پسرو $B_{AD}$ از $d=D$ بطرف $s=A$ با ردگیری مسیر مورچه پیشرو $F_{AD}$ می‌گذرد. گره‌های پشته $S_{AD}(k)$ احتمالا موفق به‌روزرسانی می‌شوند. زمان سفر از حافظه‌ای که برای بروز رسانی میانگین سفر هر گره $i$ در مسیر انتخاب شده حذف می‌شود.




شکل۶-۱ . احتمالات به‌روزرسانی توسط مورچه پسرو

www.Prozhe.com

www.Prozhe.com



[۱] *آزاده سلطانی، محمدرضا اکبرزاده توتونچی ومحمود نقیبزاده* "معرفی مورچه‌های به‌روز رسان جدولهای مسیریابی گره‌های مجاوردرالگوریتم مسیریابی AntNet" *نشریه مهندسی برق ومهندسی کامپیوترایران سال۵ شماره۱ بهار ۱۳۸۶*

[۲] *سعید صفاری امان، محمود نقیب زاده، محمد رضا اکبرزاده توتونچی*"مسیریابی توزیع شده با استفاده از کلونی مورچه‌ها در شبکه‌ای رایانه‌ای" *سیزدهمین کنفرانس ملی انجمن کامپیوتر ایران، اسفند ۱۳۸۶*

[3] *Ando,Y.,O.Masutani, H.Sasaki, H.Iwasaki, Y.Fukazawa,andS.Honiden. 2005.* Pheromon model: Application to traffic congestion prediction. *In Engineering Self Organising Systems, 182-196.*

[4] *Di Caro, G., and M. Dorigo. 1998.* Antnet: distributed stigmergetic control for communica- tion networks, *Journal of Articial Intelligence Research (JAI R),9:317-365.*

[5] *G. Di Caro and M. Dorigo.* AntNet: A mobile agents approach to adaptive routing. *Technical Report 97-12, IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, 1997.*

[6] *G. Di Caro.* Ant Colony Optimization and its Application to adaptive Routing in Telecommunication Networks.*PhD thesis, Universite Libre de Bruxelles, Faculte de Sciences Appliquees, Brussels, Belgium, September 2004.*

[7] *Adriana Camelia Suson,* Dynamic Routingusing Ant-Based Control .*Supervisor: Prof. Dr. Drs. L. J. M. Rothkrantz, Delft, August2010*

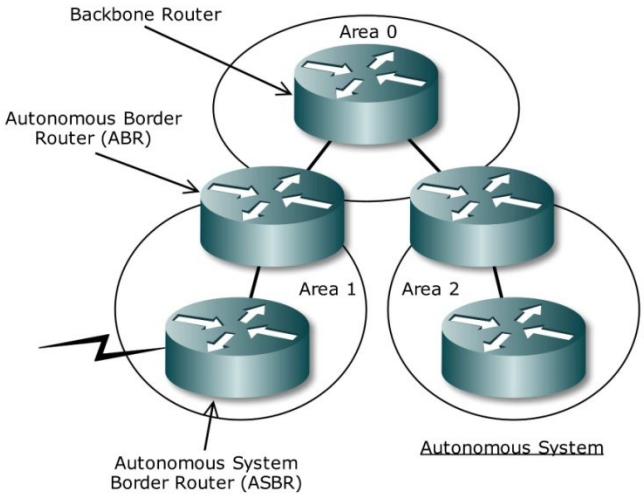
www.Prozhe.com

www.Prozhe.com



## ۲-۱ پروتکل *OSPF* (*Open Shortest Path First*)

یک پروتکل مسیریابی درونی است یعنی درون یک شبکه خودمختار و مستقل (شبکه *AS*) بکار گرفته می‌شود، از این لحاظ همانند سایر پروتکل‌های مسیر یابی عمل می‌کند. ولی برخلاف این پروتکل‌ها هسته آن مبتنی بر الگوریتم *LS* است. و بدین دلیل همگرایی جداول مسیریابی در آن بسیار سریع و تضمینی است.




شکل ۱-۲. شبکه *OSPF*

### ۲-۱-۱ روش سیل‌آسا (*Flooding*)

از لحاظ بهنگام سازی هر مسیریاب اطلاعات مربوط به مسیریاب‌های مجاور و هزینه هر لینک را گردآوری کرده و به روش سیل‌آسا برای سایر مسیریاب‌های هم‌ناحیه می‌نماید. تمام مسیریاب‌ها درون ناحیه این اطلاعات را دریافت کرده و بر اساس آنها جداول مسیریابی توپولوژی کل ناحیه را به درستی مشخص می‌کند و موضوع عدم همگرایی جدول مسیریابی و عدم تشابه این جداول در مسیریاب‌های مختلف منتفی است.

### ۲-۱-۲ شبکه *AS*

یک شبکه بزرگ و مستقل که دارای هویت یکتاست و تحت مدیریت واحد سازماندهی و پیکربندی می‌شود. یک شبکه *AS* ممکن است دارای صدها شبکه کوچک و بزرگ درونی باشد.

### ۲-۱-۳ ناحیه یا *Area*

درون هر شبکه *AS* تعدادی ناحیه تعریف می‌شود. هر ناحیه مجموعه ای از شبکه‌های متصل بهم

www.Prozhe.com

www.Prozhe.com



است که توسط مسیریاب‌های درونی بهم متصل می‌شوند. ویژگی نواحی آنست که مسیرهای پیچیده و متعدد با دیگر نواحی ندارند و لازم نیست نواحی دیگر را بدانند.

### ۲-۱-۴ ستون فقرات *OSPF* (*OSPF Backbone*)

مجموعه‌ای از مسیریاب‌ها که بجای اتصال شبکه‌ها به یکدیگر نواحی را به هم متصل می‌کنند. مجموعه این مسیریاب‌ها به همراه لینک‌های ما بین آنها ستون فقرات *OSPF* را تشکیل می‌دهند.

### ۲-۱-۵ مسیریاب *ABR*

هر مسیریاب که ارتباط بین نواحی و ستون فقرات را برقرار می‌کند.

## ۲-۲ الگوریتم *LS* برای پیدا کردن بهترین مسیر در شبکه

الگوریتم مسیریابی *LS* از لحاظ تئوری بسیار ساده است. پس از آنکه بسته‌های *LS* بروش سیل‌آسا ارسال شدند و هر مسیریاب با دریافت نسخه‌ای از آنها پایگاه اطلاعات توپولوژی شبکه را تشکیل می‌دهد، در حقیقت مشخصات یک گراف را در اختیار دارد که گراف همان مسیریاب‌ها، خطوط گراف، لینک‌های فیزیکی بین مسیریاب‌ها هستند. هزینه هر لینک نیز مشخص است. برای پیدا کردن بهترین مسیر بین دو نقطه کافی است که الگوریتم کوتاهترین مسیر *دایجکسترا* استفاده شود. هر چه تعداد مسیریاب‌های شبکه افزایش یابد، پیچیدگی زمانی این الگوریتم نیز افزایش خواهد یافت و زمان پیدا شدن مسیرهای بهینه از حد متعادل خارج خواهدشد.

## ۲-۳ کوتاهترین مسیر

از گراف می‌توان برای نمایش مسیرهای یک شبکه استفاده کرد، که در آن رأس‌های گراف نماینده گره‌های شبکه و یال‌ها نمایانگر مسیرهای ارتباطی هستند. بدین ترتیب می‌توان با یال‌ها وزن داد، این وزن می‌تواند فاصله بین دو مقصد و یا میانگین زمان لازم برای طی این فاصله باشد؛ یک بسته می-خواهد از مقصد $A$ به $B$ برود و مایل است پاسخ سوالات زیر را بداند:

۱. آیا مسیری از $A$ به $B$ وجود دارد؟
۲. اگر بیش از یک مسیر از $A$ به $B$ وجو داشته باشد کوتاه‌ترین مسیر کدام است؟

واژه‌های وزن، هزینه و طول معادلند و به‌جای یکدیگر استفاده می‌کنیم. اکنون طول (هزینه، وزن) یک مسیر را به جای تعداد یال‌ها، مجموع طول (هزینه، وزن) یال‌های روی این مسیر تعریف می‌کنیم. رأس شروع مسیر را مبدا و رأس پایانی را مقصد می‌نامند.

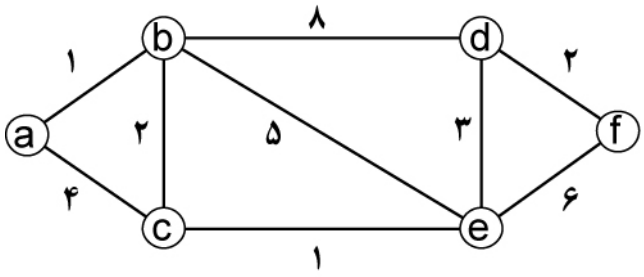
www.Prozhe.com

www.Prozhe.com



## مسئله: یک مبدا چند مقصد، یال‌هایی با هزینه غیر منفی

در این مسئله گراف جهت دار *G=(V,E)*، تابع طول *lenght(i,j)*، $lenght(i,j) \geq 0$ برای یال‌های $G$ و رأس مبدا $V$ به بقیه رأس‌ها در $G$ است. به عنوان مثال، گراف جهت‌دار شکل۲-۲ را در نظر بگیرید.




شکل۲-۲. نمایش گراف

اعداد روی یال‌ها نشان دهنده طول‌ها هستند. اگر $a$ رأس مبدا گراف باشد اعداد روی لبه‌ها نشان دهنده وزن آنها هستند. اگر $a$ رأس مبدا باشد آنگاه کوتاه‌ترین مسیر از $a$ به $b$ ، $e$ ، $f$ هیچ مسیری وجود ندارد. مراحل مختلف اجرای الگوریتم در جدول ۱-۲ نشان داده شده است.

| رأس | S | dist | مسیر |
|---|---|---|---|
| a | ۱ | - | - |
| b | ۰ | ۱ | a,b |
| c | ۰ | ۴ | a,c |
| d | ۰ | ∞ | - |
| e | ۰ | ∞ | - |
| f | ۰ | ∞ | - |

مرحله ۱) $x=b$

| رأس | S | dist | مسیر |
|---|---|---|---|
| a | ۱ | - | - |
| b | ۱ | ۱ | a,b |
| c | ۰ | ۳ | a,b,c |
| d | ۰ | ۹ | a,b,d |
| e | ۰ | ۶ | a,b,e |
| f | ۰ | ∞ | - |

مرحله ۲) $x=c$

| رأس | S | dist | مسیر |
|---|---|---|---|
| a | ۱ | - | - |
| b | ۱ | ۱ | a,b |
| c | ۱ | ۳ | a,b,c |
| d | ۰ | ۹ | a,b,d |
| e | ۰ | ۴ | a,b,c,e |
| f | ۰ | ∞ | - |

مرحله۳) $x=e$

| رأس | S | dist | مسیر |
|---|---|---|---|
| a | ۱ | - | - |
| b | ۱ | ۱ | a,b |
| c | ۱ | ۳ | a,b,c |
| d | ۱ | ۸ | a,b,c,e,d |
| e | ۱ | ۴ | a,b,c,e |
| f | ۰ | ۱۰ | a,b,c,e,f |

مرحله ۴) $x=d$

www.Prozhe.com

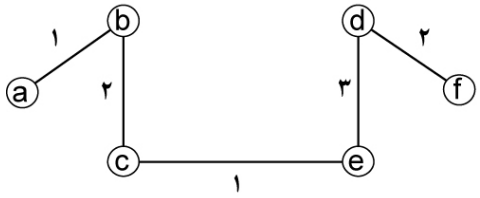
www.Prozhe.com



| رأس | $S$ | $dist$ | مسیر |
|---|---|---|---|
| $a$ | ۱ | - | - |
| $b$ | ۱ | ۱ | $a,b$ |
| $c$ | ۱ | ۳ | $a,c$ |
| $d$ | ۱ | ۸ | $a,b,c,e,d$ |
| $e$ | ۱ | ۴ | $a,b,c,e$ |
| $f$ | ۱ | ۹ | $a,b,c,e,d,f$ |

جدول۱-۲ مراحل مختلف اجرای الگوریتم بر روی شکل ۲-۲

در مثال بالا اگر کوتاهترین مسیرها از راس $a$ را به بقیه رئوس رسم کنیم درخت پوشای شکل۳-۲ حاصل می‌شود.




شکل ۳-۲. درخت پوشای کوتاهترین مسیر به مبدا رأس $a$ برای گراف شکل ۲-۲

یک الگوریتم حریصانه کوتاه‌ترین مسیرها را تولید می‌کند؛ فرض کنید که $s$ مجموعه راس‌ها (از جمله مبدا $V$ ) باشد که دارای کوتاه‌ترین مسیری است که تاکنون به‌دست آمده‌است. برای $W$ که در $S$ نمی‌باشد، $dist[w]$ طول کوتاه‌ترین مسیر از مبدا $V$ خواهد بود. به طوری که از راس‌هایی می‌گذرد که در $S$ هستند و به $W$ ختم می‌شود. ملاحظه می‌کنیم وقتی مسیرها به ترتیب غیر نزولی طول ایجاد می‌شوند:

۱. اگر کوتاه‌ترین مسیر بعدی به راس $u$ باشد، آن‌گاه مسیری از $V$ آغاز شده و به $u$ ختم می‌شود، فقط از راس‌هایی می‌گذرد که در $S$ وجود دارند. برای اثبات این مطلب باید نشان دهیم که تمام راس‌های میانی روی کوتاه‌ترین مسیر به $U$ بایستی در $S$ باشند. فرض کنید راسی مانند $W$ روی این مسیر هست که در $S$ وجود ندارد. آن‌گاه مسیر $V$ به $U$ شامل مسیری از $V$ به $W$ است که طول آن کمتر از مسیر $V$ به $U$ می‌باشد. بنا به فرض، کوتاه‌ترین مسیرها به ترتیب غیرنزولی طول‌هایش تولید مس‌شوند و لذا، مسیر کوتاه‌تر از $V$ به $W$ قبلا تولید شده‌است. بنابراین، هیچ راس میانی نمی‌تواند وجود داشته باشد که در $S$ موجود نباشد.
۲. بین همه راس‌هایی که در $S$ نیستند، مقصد مسیر تولید شده بعدی باید راس $U$ باشد زیرا این راس دارای کمترین فاصل ($dist[w]$) است. این مطلب از تعریف $dist$ و نکته اولی که ذکر شد، نتیجه می‌شود در حالتی که چندین راس با $dist$ مساوی وجود داشته باشد که در $S$ نباشد، آن-گاه ممکن است هر کدام از آنها انتخاب شوند.

www.Prozhe.com

www.Prozhe.com



۳. با انتخاب یک راس $U$ در مرحله دوم و تولید کوتاه‌ترین مسیر از $V$ به $U$، $U$ در $S$ ذخیره و یا در واقع به عنوان عضوی از $S$ در نظر گرفته می‌شود. اضافه کردن $U$ به $S$ می‌تواند طول کوتاهترین مسیرهایی را کاهش دهد که از $V$ آغاز شده و فقط از راس‌های موجود در $S$ می‌گذرند و به راسی مانند $W$ که در $S$ ختم می‌شوند (یعنی مقدار *dist[w]* ممکن است تغییر کند.) اگر این طول تغییر کند آنگاه بایستی کوتاه‌ترین مسیری باشد که از $V$ آغاز شده و با گذشتن از $U$ به $W$ می‌رسد. تمام راس‌های میانی روی مسیر $V$ به $U$ و مسیر $U$ به $W$ باید در $S$ وجود داشته باشند. به‌علاوه مسیر $V$ تا $U$ باید کوتاه‌ترین مسیر باشد، در غیر این صورت *dist[w]* درست تعریف نشده است. هم‌چنین، مسیر $U$ به $W$ می‌تواند طوری انتخاب شود که شامل هیچ راس میانی نباشد. بنابراین می‌توانیم نتیجه بگیریم که اگر *dist[w]* تغییر کند (یعنی کاهش یابد)، آن‌گاه مسیر از $V$ به $U$ و سپس $W$ تغییر می‌کند، که در آن مسیر $V$ به $U$ کوتاه‌ترین مسیر است و مسیر از $U$ به $W$ یال $<U,W>$ است. مسلما طول این مسیر برابر *dist[U]+length(<V,W>)* است.

الگوریتم *Shortest Path* [۵] که اولین بار توسط دیجکسترا (*Edsger Dijkstra*) [۴] ارائه شده با بهره‌گیری از این مشاهدات توانست طول کوتاهترین مسیرها از $V$ به دیگر راس‌های $G$ را محاسبه کند تولید مسیرها تعمیم کوچک‌تری از این الگوریتم است. برای ایجاد و پیاده سازی الگوریتم دیجکسترا، فرض می‌کنیم $n$ راس یک گراف از *0* تا *n-1* شماره‌گذاری شده‌اند. مجموعه $S$ به صورت یک آرایه بولین (*Boolean*) نگهداری می‌شود. اگر راس $i$ در $S$ نباشد *S[i]=FALSE* و در غیر این صورت *S[i]=TRUE* است.

برنامه زیر این الگوریتم را شرح می‌دهد. کلاس *Graph* به صورت زیر نشان داده می‌شود:

```
class Graph
{
private:
  int length[nmax][nmax];
  int dist[nmax];
  boolean s[nmax];
public:
  void ShortestPath(const int,const int);
  int choose(const int);
};
```

## ۱-۳-۲ تجزیه و تحلیل الگوریتم *shortest path*

زمان اجرایی این الگوریتم روی گراف با $n$ راس $O(n^2)$ است. با استفاده از هرم‌های فیبوناچی و لیست‌های مجاورتی، الگوریتم حریصانه برای مساله کوتاهترین مسیر را می‌توان با پیچیدگی $O(n\log n+e)$ پیاده سازی کرد. برای گراف های پراکنده پیاده‌سازی برنامه ذکر شده بهتر است.

www.Prozhe.com

www.Prozhe.com



```
void Graph::ShortestPath (const int n,const int v)
//dist[j],0<=j<n, is set to the lenght of the path from vertex v to vertex j
//in a digraph G with n vertices and edge lenghts given by lenght [i][j]
{
 for(i=0;i<n;i++)
 {
  s[i]=FALSE;
  dist[i]=lenght[v][i]; //initialize
 }
 s[v]=TRUE;
 dist[v]=0;
 for(i=0;i<n-2;i++)
 {   //determine n-1 path from vertex v
     //choce returns a value u such that:
  int u=choose(n);
     //dist[u]=minimum dist[w],where s[w]=FALSE
  s[u]=TRUE;
  for (int w=0;w<n;w++)
   if (!s[w])
    if (dist[u]+length[u][w]<dist[w])
     dist[w]=dist[u]+length[u][w];
 }    //end of for(i=0;...)
}
```

الگوریتم ۱-۲. برنامه تعیین کوتاهترین مسیر

## ۴-۲ انواع بسته‌های *OSPF*

در *OSPF* پنج نوع بسته مختلف برای به‌هنگام نگاه داشتن مسیریاب‌های درون یک ناحیه تعریف شده است:

۱. پیغام سلام (*Hello Message*)
۲. تقاضای مشخصات و حالت لینک (*Link State Reguest*)
۳. ارسال مقادیر به هنگام سازی حالت لینک (*Link State Update*)
۴. تصدیق و تائید رسیدن مقادیر حالت لینک (*Link State Acknowledgment*)
۵. *Database Description*

**پیغام سلام** (*Hello Message*). هر مسیریاب به محض ورود به شبکه یعنی به محض روشن و بوت شدن موظف است یک بسته سلام بفرستد و ضمن ارائه مشخصات خود هزینه لینکش را به مسیرهاب‌های مجاور اعلام نماید. ارسال بسته سلام فقط در بدو ورود مسیریاب به شبکه نیست، بلکه در فواصل زمانی منظم این کار انجام می‌شود.

www.Prozhe.com

www.Prozhe.com



**تقاضای مشخصات و حالت لینک** (*Link State Reguest*). نوع دوم بسته‌های *OSPF* بسته تقاضای مشخصات حالت لینک است. هرگاه این بسته روی شبکه ظاهر شود همه مسیریاب‌ها بروش سیل‌آسا اطاعات خود از مسیریاب‌های مجاور و هزینه لینک‌ها را در کل شبکه اعلام می‌کنند. بط.ر معمول وقتی یک مسیریاب روشن شد و پس از ارسال بسته سلام خود را معرفی کرده این تقاضا را منتشر می‌کند؛ تا کل شبکه با ارسال بسته‌های *LS* پایگاه اطلاعات توپولوژی (جدول مسیریابی) خود را به‌هنگام می کنند و مسیرها در شبکه اعمال شود. فراموش نکنید که عمل ارسال بسته های *LS* و به‌هنگام سازی، بصورت متناوب و منظم انجام می‌شود و ارسال این نوع بسته‌ها تقاضای به هنگام‌سازی قبل از موعد محسوب می‌شود.

**ارسال مقادیر به هنگام سازی حالت لینک** (*Link State Update*). نوع سوم بسته‌های *LS* است، که بصورت سیل‌آسا برروی شبکه منتشر می‌شود.این بسته‌ها که توسط یکایک مسیریاب‌ها تولید و ارسال می‌شوند، حاوی آخرین اطلاعات در مورد حالت لینک‌های هر مسیریاب هستند. از مجموع این بسته‌ها برای تشکیل پایگاه اطلاعات توپولوژی و سپس انتخاب بهترین مسیر استفاده می‌شود.

**تصدیق و تائید رسیدن مقادیر حالت لینک** (*Link State Acknowledgment*). نوع چهارم بسته‌های *LS* باید پس از دریافت آن در پاسخ با ارسال بسته *Link State Acknowledgment* به مبدا بسته، رسیدن آنرا تصدیق و تایید کنند. ارسال بسته تصدیق (*ACK*) بدین منظور است که هر مسیریاب با مقایسه مجموعه پیام‌های تصدیق و تایید با پایگاه‌اطلاعات توپولوژیکی، فعال و عملیاتی بودن تمام مسیریاب‌هایی را که مشخصات آن‌ها در این پایگاه درج شده است را بررسی و تایید نمایند.

www.Prozhe.com

www.Prozhe.com



[۱]. *جی‌اف دیمارزیو* "آموزش مسیریاب‌های سیسکو در ۲۱ روز" *مترجم: دکتر احسان ملکیان، انتشارات نص، بهار ۱۳۹۰*

[۲]. *وندل اودام* "مرجع کامل CCNP ROUTE" *مترجم: دکتر حسین محسن‌زاده، انتشارات فراهوش، ۱۳۸۹*

[۳]. *الیس هوریتز، سارتج ساهنی، دنیس مهتا* "ساختمان داده‌ها به زبان c++" *ترجمه: امیر علیخانزاده، چاپ دهم، بهار ۱۳۸۷، انتشارات باغانی*

[4]. *Dijkstra, E. W. (1959).* A note on two problems in connection with graphs. *Numer. Math., 1, 269-271.*

[5]. *Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1994).* Shortest paths algorithms: Theory and experimental evaluation. *In Sleator, D. D. (Ed.), Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 94), pp. 516-525 Arlington, VA. ACM Press.*

www.Prozhe.com

www.Prozhe.com



## ۱-۳ کلونی زنبور در طبیعت

کلونی زنبورها در طبیعت شامل دو مؤلفه منابع غذایی و زنبورهای خوراک‌جو می‌باشد. زنبور خوراک‌جو به دو دسته زنبور تقسیم می‌شود که عبارتند از : زنبور بیکار و زنبور خوراک‌جوی مشغول. زنبور بیکار به دو نوع زنبور شناسایی‌گر و زنبور تازه نفس طبقه‌بندی می‌شود. زنبور شناسایی‌گر عمل جستجو را بدون هیچ دانشی و به صورت تصادفی شروع می‌کند و زنبور تازه نفس به سالن رقص رفته و جستجو خود را بوسیله استفاده از دانش رقص دم (*waggle dance*) زنبورهای موجود در سالن شروع می‌کند. گروه دوم زنبور خوراک‌جوی مشغول می‌باشد، هرگاه زنبور تازه نفس منبعی غذایی را پیدا کند یک درجه ارتقا پیدا کرده و به زنبور خوراک‌جوی مشغول تبدیل شده و جایگاه منبع غذایی را حفظ می‌کند و به سمت منبع غذایی پرواز کرده و یک بخش از شهد را برداشته و به کندو بر می‌گردد. در این صورت:

- اگر مقدار شهد به یک سطح کم کاهش پیدا کند زنبور خوراک‌جوی مشغول منبع غذایی را ترک کرده و به یک زنبور بیکار تبدیل می‌شود.
- اگر به مقدار کافی شهد در منبع غذایی وجود داشته باشد زنبور خوراک‌جوی مشغول می‌تواند بدون تبادل اطلاعات به خوراک‌جویی خود ادامه دهد و به منبع غذایی برگردد.
- اگر به مقدار کافی شهد در منبع غذایی وجود داشته باشد زنبور خوراک‌جوی مشغول می‌تواند در سالن رقص به تبادل اطلاعات بپردازد.

مهمترین بخش یک کندو محل تبادل اطلاعات است که تالار رقص نام دارد، ارتباطات در بین زنبورها در رابطه با کیفیت منابع غذایی با رقص دم انجام می‌گیرد. انتخاب منبع توسط زنبورهای تازه نفس به طور کامل مشخص نیست اما فرض بر این است که این انتخاب به صورت تابعی از کیفیت منبع غذایی با توجه به رقص دم می‌باشد. زنبورهای شناسایی‌گر نیز با احتمالی متناسب با کیفیت منبع غذایی اطلاعات خود را به اشتراک می‌گذارند.

ترشکو [۱،۲] مدلی از رفتار خوراک جویی کلونی زنبور عسل معرفی نموده است. این مدل بر مبنای هوش اشتراکی گردآوری شده است، کلونی زنبور تشکیل شده از سه قسمت اصلی است: منابع غذایی، خوراک‌جویان مشغول، خوراک‌جویان بی‌کار.

۱. **منابع غذای:** برای انتخاب یک منبع غذایی، هر زنبور خوراک‌جو مواردی همچون نزدیکی به کندو، غنای انرژی شهد، مزه شهد و سهولت یا سختی استخراج شهد برای ارزیابی منابع غذایی در نظر می‌گیرد. برای سادگی، کیفیت منابع غذایی تنها یک مقدار می‌تواند باشد، اگرچه به پارامترهای مختلف ذکر شده در بالا نیز بستگی دارد.

www.Prozhe.com

www.Prozhe.com



۲. **خوراک‌جوی مشغول:** هر خوراک‌جوی مشغول، در یک منبع غذایی درحال استخراج شهد می‌باشد. خوراک‌جو اطلاعات در مورد منبعش را به کندو منتقل با سایر زنبورهای کندو به اشتراک می‌گذارد. اطلاعاتی از جمله: فاصله، مسیر و سودمندی منابع غذایی را در برمی‌گیرد.
۳. **خوراک‌جوی بیکار:** هر زنبور خوراک‌جوی که بدنبال منابع غذایی برای استخراج شهد می‌گردد، بیکار خوانده می‌شود. میتواند یک شناسایی‌گر باشد که محیط را به طور اتفاقی برای پیدا کردن منابع غذایی جستجو می‌کند، تا در اختیار زنبورهای خوراک‌جو قرار دهد. میانگین تعداد زنبورهای شناسایی‌گر ۵ تا ۱۰ درصد می‌باشد.

برای بهتر فهمیدن مشخصات اساسی رفتار خوراک‌جویان اجازه بدهید مثالی انجام بدهیم، (شکل ۱-۳) فرض کنید دو منبع غذایی $A$ و $B$ کشف شده‌اند، در همان ابتدا عامل بالقوه خوراک‌جوی یعنی خوراک‌جوی بیکار شروع خواهدکرد، زنبور خوراک‌جو هیچ شناختی درباره منابع غذایی دور لانه نخواهد داشت. دو انتخاب برای چنین زنبوری وجود دارد:

۱. می‌تواند یک شناسایی‌گر باشد، اطراف کندو را جستجو کند. ($S$ در شکل ۱-۳)
۲. به عنوان یک عضو جدید، بعد از تماشا رقص دم سایر زنبورها شروع به جستجو کند. ($R$ در شکل ۱-۳ )

پس از یافتن منبع غذا، زنبور توانایی خود را به کار می‌ببرد و موقعیت منبع را به حافظه می‌سپارد و استخراج آنرا را فورا شروع می‌کند. از این رو زنبور یک خوراک‌جوی مشغول خواهد شد. زنبور خوراک‌جو شهد را از مبدا بارمی‌کند و به کندو برمی‌گرداند. پس از تبدیل شهد به ذخیره غذایی، زنبور انتخاب های زیر را دارد:

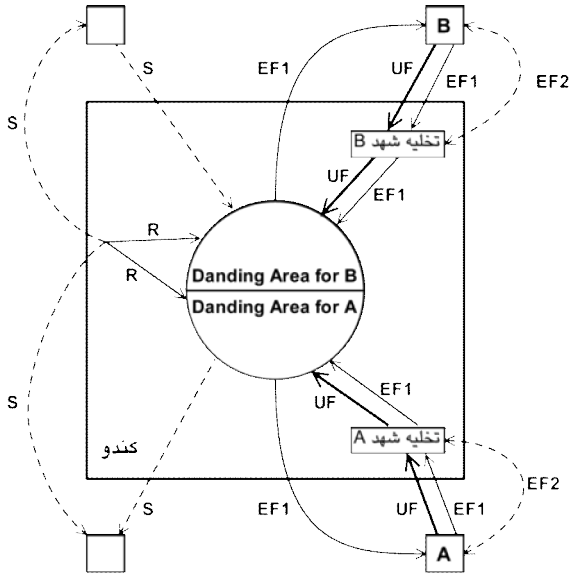
*UF (uncommitted follower)* : بعد از خاتمه منبع غذایی زنبور به یک خوراک‌جوی بدون تعهد تبدیل می‌شود.
*EF1* : ممکن است در کندو رقص انجام دهد و سایر زنبور ها را به همان منبع هدایت کند.
*EF2* : ممکن است بدون جذب سایر زنبورها به برداشت از منبع غذایی ادامه دهد.

نکته مهم آن است، تمام زنبورها خوراک‌جویی را با هم انجام ندهند. آزمایشات آنرا تایید کرده است، زنبوران جدید با یک سرعت متناسب با اختلاف بین تعداد زنبوران و تعداد زنبوران در حال خوراک‌جویی شروع به فعالیت کنند. [۳]

www.Prozhe.com

www.Prozhe.com






شکل ۱-۳. رفتار زنبور عسل برای خوراک‌جویی

## ۲-۳ مدل عامل زنبور عسل

مدل عامل زنبور ما از رفتار خوراک جویی کلونی زنبور عسل الهام گرفته شده است. مدل عامل زنبور عسل از چهار عامل تشکیل شده است: بسته بندها (*packers*)، شناسایی‌گرها (*scouts*)، خوراک‌جویان (*foragers*) و حرکت دسته‌جمعی (*swarms*). در ادامه مطالب ما از واژه شناسایی‌کننده به جای عامل شناسایی‌کننده، خوراک‌جو، بجای عامل خوراک‌جو وغیره استفاده می‌کنیم.

### ۱-۲-۳ بسته بندها

بسته بندها کار انبار کردن غذای زنبور را تقلید می‌کنند. آنها همیشه در داخل گره اقامت دارند، بسته داده‌ها را از لایه انتقال دریافت و در حافظه قرار می‌دهند. کار اصلی آنها پیدا کردن، یک خوراک‌جو برای بسته‌های داده‌شان است، زمانی که به خوراک‌جو تحویل دهند، یکباره از بین می‌روند.

### ۲-۲-۳ شناسایی کنندگان

شناسایی‌کنندگان راه‌های جدید را از گره شروع به گره مقصدشان کشف می‌کنند. هر شناسایی کننده با استفاده از اصل منتشر شدن به تمام گره‌ها و با تنظیم کردن زمان برای زندگی (*TTL*) انتقال داده می‌شود، که زمانی که هر شناسایی کننده می تواند دوباره انتشار پیدا کند را نظارت می‌کند. هر شناسایی کننده یک شماره شناسایی یکتا با یک کلید بر مبنای *id* و گره مبنایش تشکیل می‌دهد. وقتی هر شناسایی‌گر به مقصد می‌رسد، پسروی به عقب روی همان مسیر برای رسیدن به مقصد

www.Prozhe.com

www.Prozhe.com



شروع می‌شود، گره مقصد همه شناسایی کننده‌ها رسیده را برای اطمینان از کشف مسیرهای چندگانه بر می‌گرداند. وقتی یک شناسایی کننده به گره مبداش برمی‌گردد، خوراک‌جویان را برای استفاده از مسیرش بوسیله رقص مجازی تشویق می‌کند. (بوسیله زنبوران شناسایی‌کننده در طبیعت انجام می‌شود.) هر رقص شناسایی‌کننده از سایر رقص شناسایی‌کنندگان جداست. (برای تشویق زنبوران خوراک‌جو در طبیعت).

## ۳-۲-۳ خوراک جویان

خوراک جویان کارگران اصلی در الگوریتم *BeeHive* هستند. آنها بسته‌های داده را از بسته بندها (*Packer*) دریافت و سپس به مقصدشان انتقال می‌دهند، هر خوراک‌جو یک نوع مخصوص دارد: تاخیری یا مدت عمر. خوراک‌جویان تاخیری اطلاعات تاخیری را از شبکه جمع می‌کنند. اول سعی می‌کنند مسیری را که کمترین تاخیر را دارند انتخاب کنند، در حالی که در مرحله بعد سعی می‌شود زمان زندگی شبکه را افزایش دهند.

هر خوراک‌جو مسیر کامل به شکل توالی از گره‌ها که منجر به مقصد می‌شوند از شناسایی کننده‌ها یا سایر خوراک جویان دریافت می‌کند. هر خوراک‌جو از حالت نظیر به نظیر (*point to point*) پخش تا مقصد پیروی می‌کند و اطلاعاتی در مورد حالت شبکه و نوع وابستگی‌شان جمع آوری می‌کند. هربار که هر خوراک جو به مقصدش می‌رسد آنوقت آنجا باقی می‌ماند تا ترافیک شبکه گره مقصد به گره مبدا منتقل شود. همچنین خوراک‌جویان هنگام رسیدن به منبع‌شان، از رقص مجازی مانند شناسایی کننده‌ها انجام می‌دهند.

## ۳-۲-۴ حرکت دسته جمعی

پروتکل انتقال غیرقابل اعتماد مانند *UDP* ، هیچ تصدیق واضحی برای بسته‌های داده مورد قبول نمی‌فرستد. چنین پروتکلی مسیر بازگشت برای خوراک جویان منتظر نمی‌تواند فراهم کند، بنابراین برخی، هرگز به مبدا برنمی‌گردند. در نتیجه، امکان دارد گره منبا خوراک جویانش را تمام کند و نتواند به ارتباط ادامه دهد. ما این مسئله را به کمک حرکت دسته جمعی حل نموده‌ایم. وقتیکه تفاوت بین خوراک جویان وارد شده از گره $i$ و خوراک جویان خارج شده از همان گره $i$ و دسترسی به حد نصاب در گره $j$ باعث می‌شود گره $j$ شروع به حرکت دسته جمعی به سوی گره $i$ کنند. وقتی یک حرکت دسته جمعی به گره $i$ می‌رسد، بنابراین خوراک‌جویان از بارمفید بیرون کشیده می‌شوند.

## ۳-۳ معماری *BeeHive*

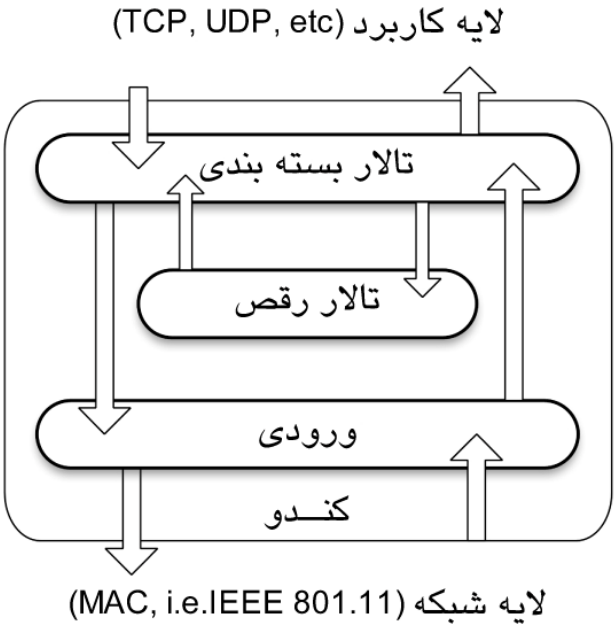
هر کندو از سه بخش تشکیل می‌شود: تالار بسته‌بندی (*packing floor*)، ورودی (*entrance*) و تالار رقص (*dance floor*). ساختار کندو در شکل ۲-۳ نشان داده شده‌است. ورودی یک رابط کنترل

www.Prozhe.com

www.Prozhe.com



دسترسی به رسانه (*MAC*) است. تالار بسته‌بندی واسط لایه انتقال است. همه بسته‌ها در کندو از ورودی وارد می‌شوند. سالن رقص شامل خوراک‌جویان است (اطلاعات مسیریابی) برای مسیریابی بسته‌های داده در گره‌های آغازین.




شکل ۲-۳. دید کلی از معماری *BeeHive*

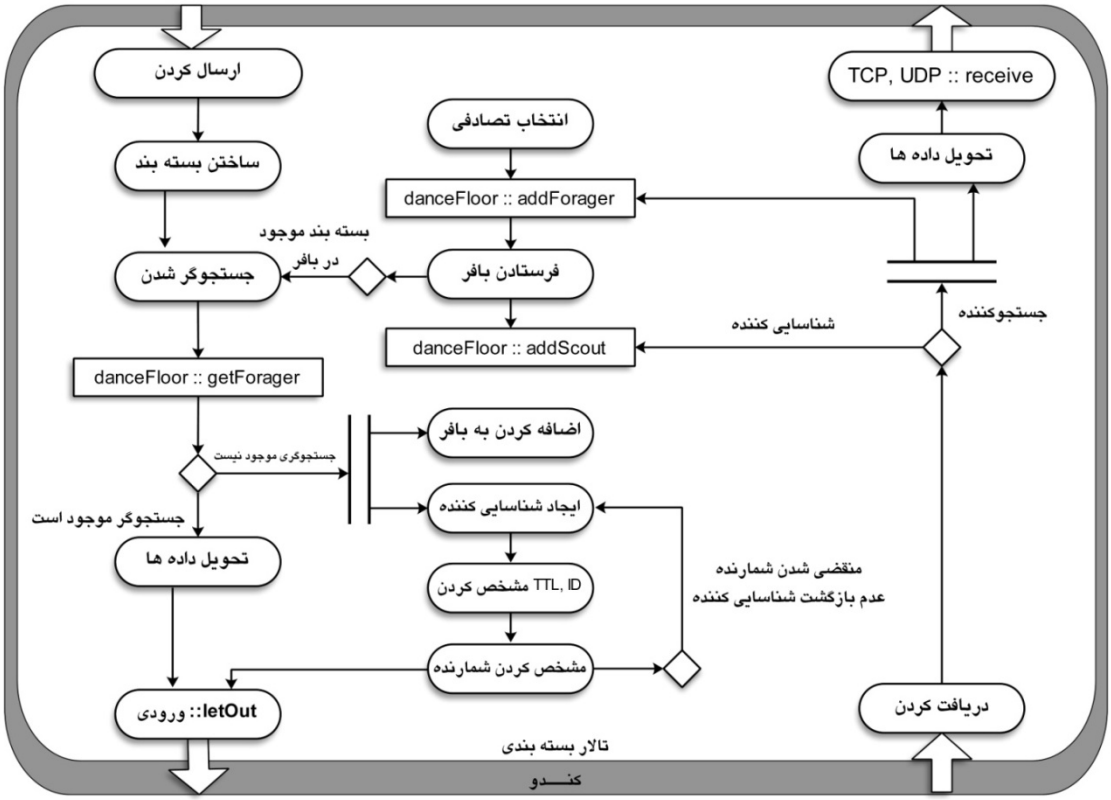
## ۱-۳-۳ تالار بسته بندی

تالار بسته بندی یک واسط به لایه بالاتر انتقال است. (مانند *TCP, UDP*) وقتی یک بسته داده از لایه انتقال می‌رسد، بسته‌بند بوجود می‌آید و در تالار بسته‌بندی بسته داده‌ها را ذخیره می‌کند. سپس بسته‌بند سعی می‌کند خوراک‌جوی مناسب برای بسته‌های داده از تالار رقص پیدا کند. اگر بسته‌بند خوراک‌جوی مناسب را پیدا کند، بسته داده‌ها را تحویل می‌دهد و از بین می‌رود. وگرنه تا یک زمانی منتظر می‌ماند، اگر خوراک‌جویی تا آن زمان نرسید، آنوقت یک شناسایی‌گر فعالیت را ادامه می‌دهد و مسئول کشف راه‌های جدید به مقصد است. شکل ۳-۳ مجموعه اعمال در یک تالار رقص را شرح می‌دهد.

www.Prozhe.com

www.Prozhe.com






شکل ۳-۳ تالار بسته‌بندی

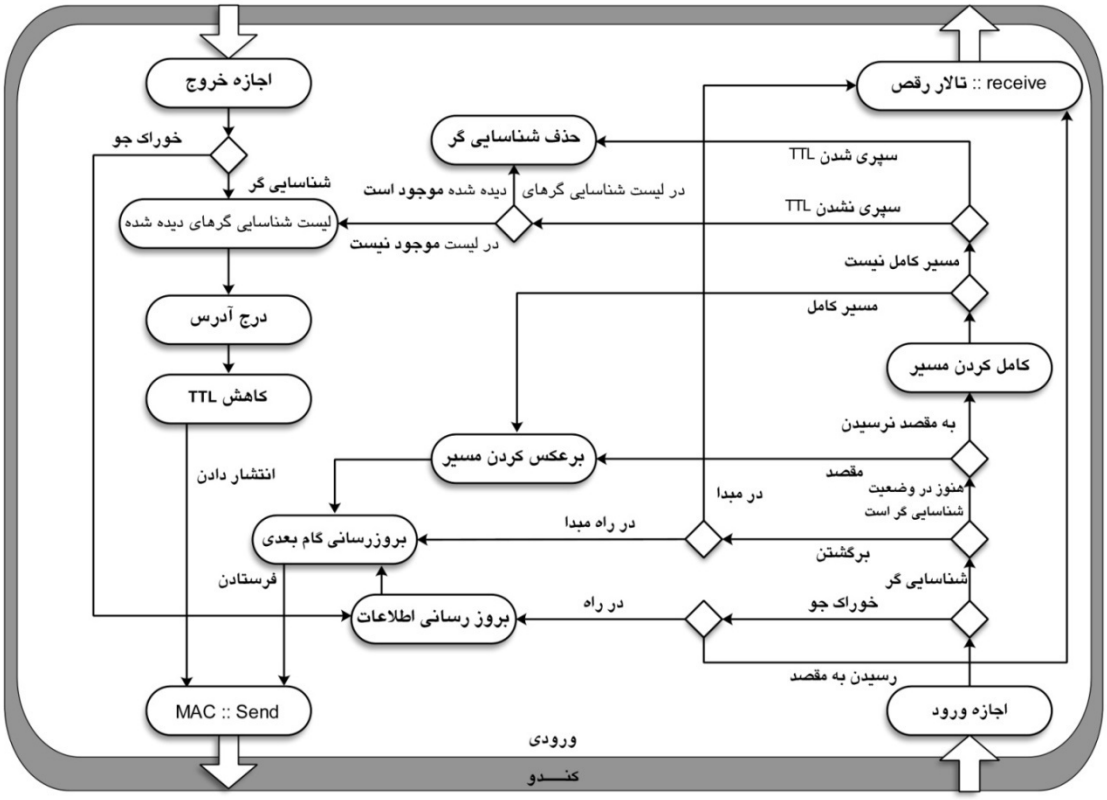
## ۳-۳-۲ ورودی

اعمالی که در ورودی انجام می‌گیرد در تصویر ۳-۴ نشان داده‌شده‌است. ورودی یک واسط به سطح زیرین لایه کنترل دسترسی به رسانه ($MAC$) است. ورودی تمام بسته‌های رسیده/خارج شده را اداره می‌کند. اگر یک شناسایی کننده در ورودی دریافت شود زمان زندگیش ($TTL$) منقضی نشده‌است. اطلاعات درمورد $ID$ شناسایی کننده گره منبع در جدول ذخیره می‌شود. اگر یک شناسایی‌گری که قبلا مورد تایید قرار گرفته وارد کندو شود تکرارهای بعدی از بین می‌روند. اگر گره جار مقصد یک خوراک‌جو باشد آنگاه به سمت تالار بسته بندی فرستاده می‌شود در غیر این صورت به واسط کنترل دسترسی به رسانه جهت اتصال به گام بعدی فرستاده می‌شود.

www.Prozhe.com

www.Prozhe.com






شکل ۴-۳. ورودی

## ۳-۳-۳ سالن رقص

سالن رقص قلب کندو است، برای اینکه در آن تصمیمات مهم مسیریابی صورت می‌گیرد. وقتی که یک زنبور خوراک‌جو از سفرش بر می‌گردد، نیروهای جدید خوراک‌جو را بوسیله رقص از کیفیت مسیر پیموده با اطلاع می‌کند. بهرحال، معیار سنجش برای هر منطقه کاوش متفاوت است. چنانچه قبلا ذکر کردیم، عمر منطقه کاوش بر اساس کیفیت ظرفیت مسیر و میانگین انباره باقیمانده گره‌ها در مسیر ارزیابی می‌شود. طول عمر یک منطقه کاوش ممکن است اجازه یابد چندین مرتبه شبیه سازی شود، بر این اساس دو سناریو وجود دارد: ۱- گره‌ها در مسیر گنجایش ظرفیت انباره کافی دارند. (مسیر خوب) ۲- اگر تعداد زیادی بسته منتظر باشند برای آنکه مسیر با گنجایش انباره تعیین شود. در مورد دوم معقول است که بسته ها با مسیر خوب بفرستند.

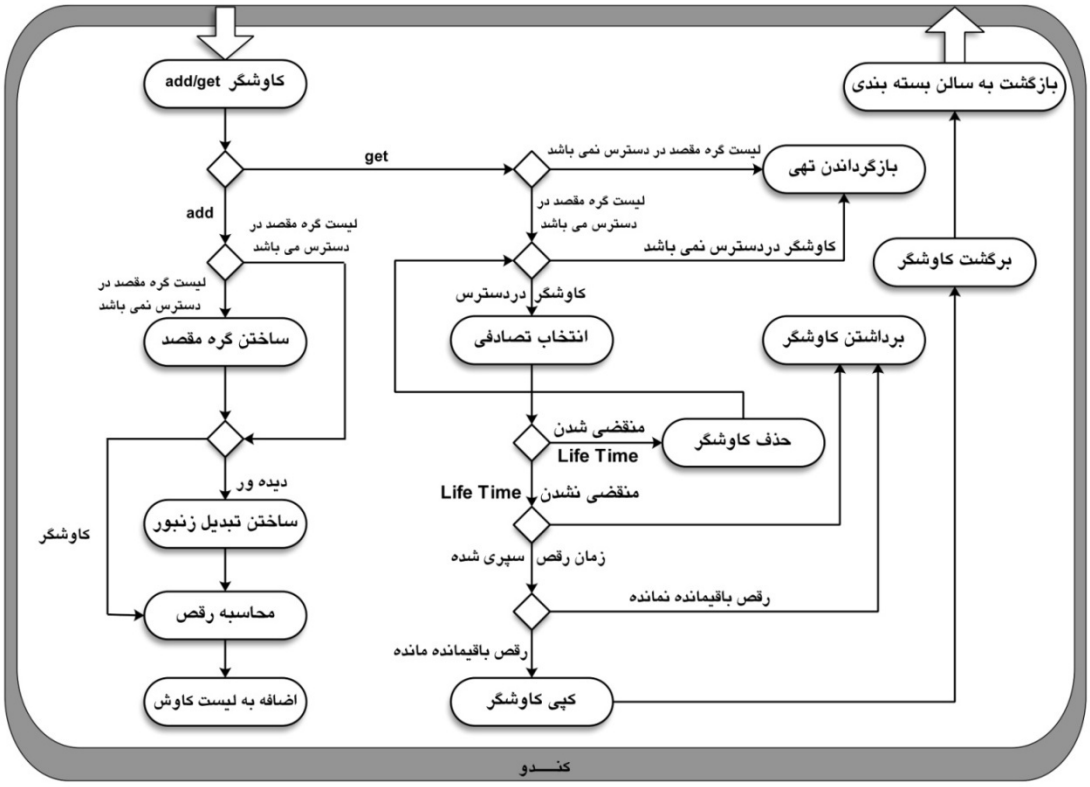
به عبارتی اگر هیچ کدام از بسته‌ها در یک منطقه کاوش برای یک مسیر خوب منتظر نباشد، سایر همکاران کار حمل بسته داده‌ها را انجام می‌دهند. این مفهوم که مستقیما رفتار زنبوران خوراک‌جو در طبیعت الهام گرفته شده‌است و در تنظیم شماره منطقه کاوش برای هر مسیر کمک می‌کند. سالن رقص همچنین در تطبیق کاوش و در پاسخ به درخواست بسته می‌فرستند. کاوش که طول عمرش در توابع تطبیقی تصور نشده‌است، اگر چندین منطقه کاوش موجود باشند آنوقت یک منطقه به صورت

www.Prozhe.com

www.Prozhe.com



احتمالی بین آنها انتخاب می‌شود. در انتخاب بهتر بسته‌ها بر روی مسیرهای چند گانه کمک می‌کند. از تراکم بالای انباره‌های گره‌های اتصال جلوگیری می‌کند. اگر شماره رقص صفر باشد، آنوقت بسته کاوش اصلی که فرستاده شده است،از ورودی سالن رقص حذف می‌شود. بر اساس قوانینی که ذکر شد مناطق کاوش جدیدتر آخرین مسیرها را نمایش می‌دهد که به احتمال بیشتر در آینده باقی می‌مانند. اگر یک خوراک‌جوی قدیمی‌تر کندو را ترک کند آنوقت کندو مسیر به آن مقصد ندارد. ما معتقدیم اگر مسیر به مقصد موجود باشد، بزودی یک خوراک‌جو به طرف کندو بازگشت می‌کند، اگر خوراک‌جو در مدت زمان معین نیاید احتمالا گره مسیر به گره مقصد را از دست خواهدداد. این مکانیزم را می‌توان در ارسال بسته‌های کنترلی سلام و نیز پیام‌های خطای مسیریابی بکار برد. [۴]




شکل ۵-۳. سالن رقص

www.Prozhe.com

www.Prozhe.com



## ۴-۳ الگوریتم *BeeHive*

در این بخش، اساس سازمانی رفتار زنبور را شرح می‌دهیم، که برای ایجاد یک مدل عامل تغییر شکل یافته، الهام بخش خواهد بود. زنبورهای عسل کیفیت هر سایت غذایی کشف شده را ارزیابی می‌کنند و *رقص دم* (*waggle dance*) را برای آن در تالار رقص انجام می‌دهند، اگر از کیفیت بالایش مطمئن باشند. بنابراین هر سایت کشف شده‌ای دریافت نمی‌شود. در نتیجه، پایگاه‌های گل با کیفیت به‌طور گسترده‌ای بهره‌برداری می‌شوند. از این رو تالار رقص را با جدول مسیریابی خلاصه می‌کنیم. جایی که عامل زنبور، از یک منبع شروع به فعالیت می‌نمایند اما به همسایه‌های متفاوت در یک گره معین می‌رسند و اطلاعات مسیریابی مدل حالت شبکه را در این گره معاوضه نمایند. اکثریت خوراک‌جویان منابع غذایی را در نزدیکی کندویشان بهره‌برداری می‌کنند، در حالیکه اقلیت منابع غذایی را در فاصله دوری از کندویشان می‌بینند. این مشاهده را به محلی که دو نوع عامل دارد تبدیل کردیم. عامل زنبور مسافت کوتاه (*short distance bee agents*) و عامل زنبور مسافت طولانی ( *long distance bee agents*). عامل زنبور مسافت کوتاه اطلاعات مسیریابی در همسایگی را (تا یک تعدادی از گام‌ها) از گره مبدا جمع آوری و منتشر می کند، درحالیکه عامل زنبور مسافت طولانی اطلاعات مسیریابی همه گره‌های شبکه را جمع‌آوری و منتشر می‌کند. به طور غیر رسمی، الگوریتم *BeeHive* و مشخصات اصلیش بصورت زیر خلاصه می‌شوند:

۱. شبکه از بخش‌های ثابت که مناطق خوراک‌جویی (*foraging regions*) نامیده می‌شوند، متشکل شده‌است. نتایج بخش‌ها از خصوصیات توپولوژی شبکه ناشی می‌شود. هر منطقه خوراک‌جویی یک گره نماینده دارد. در حال حاضر پایین‌ترین آدرس IP در مناطق خوراک‌جویی برای گره‌های نماینده انتخاب می‌شود. اگر این گره از بین برود گره با آدرس IP بالاتر این کار را انجام می‌دهد.
۲. هر گره از یک گره مخصوص منطقه خوراک‌جویی تشکیل شده، شامل تمام گره‌هایی است که زنبور مسافت کوتاه می‌توانند به آن برسند.
۳. هر گره غیر نماینده متناوباً یک عامل زنبور عسل مسافت کوتاه به سایت‌های همسایه می‌فرستد.
۴. موقعیکه یک مدل از عامل زنبور وارد یک سایت می‌شود، اطلاعات مسیریابی آنجا را به روش سیل‌آسا (*flooded*) به روز رسانی می‌کند. بهر حال، همسایه‌ها را به جایی که از آنجا آمده نخواهد فرستاد. این فرایند ادامه می‌یابد تا طول عمر عامل منقضی شود، یا مدلی از این عامل زنبور قبلا در سایت دریافت شده باشد، مدل جدید آنجا ازبین خواهد رفت.

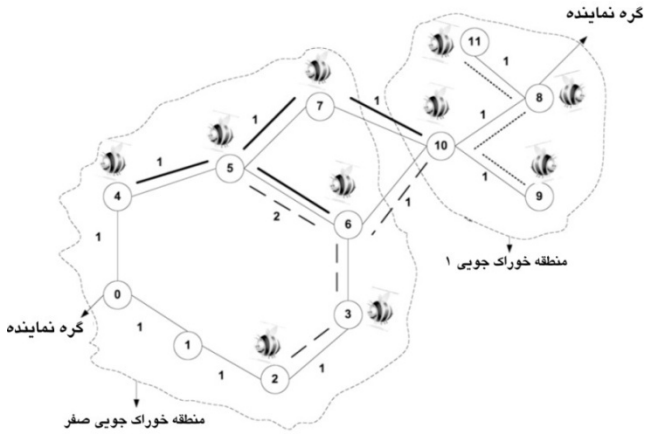
www.Prozhe.com

www.Prozhe.com



۵. گره‌های نماینده تنها زنبورهای مسافت طولانی را به کار می‌گیرند که از همسایه‌ها (همان طور که در مرحله ۴ ذکر شد) می‌رسند. بهرحال طول عمرشان (شماره گام) بوسیله حد راه دور محدود می شود.
۶. هر عامل هنگام مسافرت اطلاعات مسیر را وقتی آنجا را ترک می‌کند، جمع‌آوری و حمل می‌کند. برآورد زمان سفر، رسیدن از گره مبدا به گره جاری است. عامل زنبور از صف اولویت برای انتشار سریع اطلاعات مسیریابی استفاده می‌کنند.
۷. بدین‌گونه هر گره از اطلاعات مسیریابی جاری برای رسیدن گره‌ها به مناطق خوراک‌جویی نگه‌داری می‌کند، برای رسیدن گره نماینده مناطق خوراک‌جویی اعلام آمادگی می‌کند. در این مکانیزم هر گره برای مسیر بسته‌های داده فعال می‌گردد، (آنهایی که مقصدشان دورتر از منطقه خوراک جویی گره معین است) بطرف مسیر گره نماینده منطقه خوراک‌جویی که شامل گره مقصد است.
۸. گام بعدی برای یک بسته داده بر اساس احتمال معیار کیفیت همسایه انتخاب می‌شود، در نتیجه، همه بسته ها بهترین مسیر را ادامه نمی‌دهند. این در به حداکثر رساندن کارآیی سیستم کمک خواهد کرد، که یک بسته داده از بهترین مسیر ممکن پیروی نکند، این مفهوم مستقیما از رفتار زنبور اقتباس شده‌است: "هر زنبور می‌تواند سودمندی کولونیش به حداکثربرساند، اگر از روش‌های کنترلی تالار رقص برای مشخص نمودن مطلوبترین غذا خود داری کند." [۵] (در مقایسه با *OSPF* که همیشه گام بعدی را کوتاهترین مسیر انتخاب میکند.)

تصویر شماره ۶-۳ نمونه‌ای از الگوریتم سیل‌آسا (*flooding algorithm*) را نشان می‌دهد. عامل زنبـــور مسافت کوتاه تا ۳ گام در این مثال می‌پیمایند. شماره‌های روی مسیرها هزینه‌هایشان را نشان می‌دهند. الگوریتم سیل‌آسا یک الگوریتم متغیر جستجوی عرضی ( *breadth first search algorithm*) است. گره‌های ۲،۳،۴،۵،۶،۷،۸،۹،۱۱ منطقه خوراک‌جویی برای گره ۱۰ تشکیل می‌دهند.




شکل ۶-۳. الگوریتم سیل‌آسا زنبور عسل

www.Prozhe.com

www.Prozhe.com



زمان پیمایش $t_{is}$ زمانی که یک بسته طول می‌کشد، تا از گره مبادا $s$ به گره جاری $i$ برسد. (با نادیده گرفتن زمان تاخیر پردازش پروتکل در گره‌های $i$ و $s$)

$$t_{is} \approx \frac{ql_{in}}{b_{in}} + tx_{in} + pd_{in} + t_{ns}$$

$ql_{in}$ اندازه صف (برحسب بیت) در گره $i$، برای همسایه $n$ . $b_{in}$ پهنای باند، اتصال بین گره $i$ و همسایه $n$ . $tx_{in}$ تاخیر مخابره (*transmission delay*)، $pd_{in}$ تاخیر انتشار (*propagation delay*)، اتصال بین گره $i$ و همسایه $n$. $t_{ns}$ زمان پیمایش از گره $n$ به گره $s$ . پهنای باند و تاخیر انتشار تمام اتصال‌های یک گره با فرستادن بسته سلام (*hello packets*) تخمین زده می‌شوند.

## ۳-۵ جدوال مسیریابی در الگوریتم *BeeHive*

در الگوریتم *BeeHive* هر گره $i$ سه نوع جدول مسیریابی نگه‌داری می‌کند: جدول مسیریابی داخل منطقه خواراک‌جویی (*Intra Foraging Zone-IFZ*)، جدول مسیریابی بین منطقه خوراک‌جویی (*Inter Foraging Region-IFR*)، جدول مسیریابی منطقه خوراک جویی اعضا ( *Foraging Region Membership-FRM*). اندازه جدول مسیریابی داخل منطقه خواراک‌جویی(*IFZ*) $R_i$ برابر با $|D(i)| \times |N(i)|$ است. $D(i)$ مجموعه مقصدها در منطقه خوراک‌جویی است و $N(i)$ مجموعه همسایه‌های گره $i$ است. هر ورودی $P_{jk}$ جفت تاخیر صف‌بندی و تاخیر انتشار $(q_{jk}, p_{jk})$ است، تجربه رسیدن بسته به مقصد $k$ از طریق همسایه $j$ است. جدول ۱ نمایشی از یک مثال $R_i$ است. در جدول مسیریابی بین منطقه خوراک جویی (*IFR*) مقادیر تاخیر صف‌بندی، تاخیر انتشار در رسیدن گره نمایندگی منطقه خوراک‌جویی توسط همسایه‌های یک گره ذخیره می‌شود. ساختار جدول مسیریابی بین منطقه خوراک جویی (*IFR*) همانند جدول ۱ است، جایی که بوسیله جفت (نماینده، منطقه) جایگزین می‌شود. جدول مسیریابی منطقه خوراک جویی اعضا (*FRM*) نقشه مقصدهای شناخته‌شده یک منطقه خوراک‌جویی را تامین می‌کند. در این روش ما احتیاج به ذخیره‌سازی را حذف کردیم. در مقایسه با الگوریتم *AntNet* که ورودی‌های در جدول مسیریابی ذخیره می‌شوند و مقدار قابل توجهی از حافظه را اشغال می‌نمایند.

| $R_i$ | $D_1(i)$ | $D_2(i)$ | ... | $D_d(i)$ |
|---|---|---|---|---|
| $N_1(i)$ | $(q_{11}, p_{11})$ | $(q_{12}, p_{12})$ | ... | $(q_{1d}, p_{1d})$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $N_n(i)$ | $(q_{n1}, p_{n1})$ | $(q_{n2}, p_{n2})$ | ... | $(q_{nd}, p_{nd})$ |

www.Prozhe.com

www.Prozhe.com



جدول ۱-۳ جدول مسیریابی داخل منطقه خوراک‌جویی (*IFZ*)

```
t:= current time, t_end := time to end simulation
// Short Limit:= 7, Long Limit:= 40, Bee Generation Interval:= 1
// i=current node, d=destination node, s=source node
// n=successor node of i, p=predecessor node of i
// z=Representative node of the foraging region containing s
// w=Representative node of the foraging region containing d
// q is queuing delay estimate of a bee agent from node p to s
// p is propagation delay estimate of a bee agent from node p to s
Δt:= Bee Generation Interval, Δh:= hello packet generation interval
b_ip :=estimated link band width to neighbor p
p_ip :=estimated propagation delay to neighbor p
h_i := hop limit for bees of i, l_ip :=size normal queue i to p(bits)
foreach Node // concurrent activity over the network
  while (t ≤ t_end)
    if (t mod Δt =0)
      if(i is representative node of the foraging region)
        set h_i := Long Limit, b_i is long distance bee agent
      else
        set h_i := Short Limit, b_i is short distance bee agent
      endif
      launch a bee b_i to all neighbors of i
    endif
    foreach bee b_s received at i from p
      if(b_s was launched by i or its hop limit reached)
        kill bee b_s
      elseif (b_s is inside foraging zone of node s)
        q:= q + l_ip/b_ip and p:= p + p_ip
        update IFZ routing table entries q_ps = q and p_ps = p
        update q (q := Σ_{k∈N(i)} (q_ks × g_ks)) and p (p := Σ_{k∈N(i)} (p_ks × g_ks))
      else
        q:= q + l_ip/b_ip and p:= p + p_ip
        update IFR routing table entries q_pz = q and p_pz = p
        update q (q := Σ_{k∈N(i)} (q_kz × g_kz)) and p (p := Σ_{k∈N(i)} (p_kz × g_kz))
      endif
      if (b_s already visited node i)
        kill bee b_s
      else
        use priority queues to forward b_s to all neighbors of i except p
      endif
```

www.Prozhe.com

www.Prozhe.com



```
    endfor
    foreach data packet d_sd  received at i from p
      if (node d is within foraging zone of node i)
        consult IFZ routing table of node i to find delays to node d
        calculate goodness of all neighbors for reaching d using equation 2
      else
        consult FRM routing table of node i to find node w
        consult IFR routing table of node i to find delays to node w
        calculate goodness of all neighbors for reaching w using equation 2
      endif
      probabilistically select a neighbor n (n != p) as per goodness
      enqueue data packet dsd in normal queue for neighbor n
    endfor
    if (t mod Δh =0)
      send a hello packet to all neighbors
        if (time out before a response from neighbor) (4th time)
          neighbor is down
          update the routing table and launch bees to inform other nodes
        endif
      endif
    endwhile
  endfor
```

الگوریتم ۱-۳. *BeeHive*

www.Prozhe.com

www.Prozhe.com



[1]. *V. Tereshko,* Reaction–diffusion model of a honeybee colony's foraging behaviour, *in: M. Schoenauer (Ed.), Parallel Problem Solving from Nature VI, Lecture Notes in Computer Science, vol. 1917, Springer–Verlag, Berlin, 2000, pp. 807–816.*

[2]. *V. Tereshko, T. Lee,* How information mapping patterns determine foraging behaviour of a honeybee colony, *Open Systems and Information Dynamics 9 (2002) 181–193.*

[3]. *Dervis Karaboga , Bahriye Akay* , A comparative study of Artificial Bee Colony algorithm, *Erciyes University, The Department of Computer Engineering, Melikgazi, 38039 Kayseri, Turkey, Applied Mathematics and Computation 214 (2009) 108–132*

[4]. *Horst. F. Wedde, Muddassar Farooq, Thorsten Pannenbaecker, Bjoern Vogel, Christian Mueller, Johannes Meth and Rene Jeruschkat, BeeAdHoc:* An Energy Efficient Routing Algorithm for Mobile Ad Hoc Networks Inspired by Bee Behavior, *University of Dortmund 44221, Germany, GECCO'05, June 25–29, 2005, Washington, DC, USA.*

[5]. *T. Seeley. The* Wisdom of the Hive. *Harvard University Press, London, 1995*

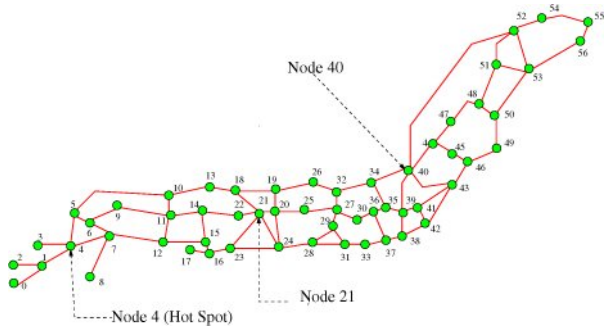
www.Prozhe.com

www.Prozhe.com



## ۴-۱ محیط شبیه سازی برای *BeeHive*

برای ارزیابی الگوریتم *BeeHive* در مقایسه با *AntNet* ، *OSPF* ، تمامی آنها را در شبیه ساز *OMNeT++* پیاده سازی کرده‌ایم. برای *OSPF* یک مسیریابی وضعیت لینک ایستا اجرا نمودیم که الگوریتم دایکسترا را پیاده سازی می‌کند، گام بعدی کوتاهترین مسیر از یک منبع به مقصد را انتخاب می‌کند. برای *AntNet* همان پارامترهایی را که توسط مولفانشان گزارش شده‌است استفاده کردیم. نمونه شبکه‌ای که ما از آن استفاده کرده‌ایم اینترنت ستون فقرات (*Backbone*) ژاپن است، (*NTTNET*) شکل ۴-۱ را ببینید. شامل ۵۷ گره و ۱۶۲ شبکه لینک‌های دوسویه است. پهنای باند ارتباط *mbit/sec* ۶ است و تاخیر انتشار از ۲ تا ۵ میلی ثانیه است. ترافیک شبکه در ضوابط جلسه-های کاری از تفاوت ما بین دو گره تعریف شده‌است. مقدار هر جلسه‌کاری توسط اندازه جلسه کاری مشخص شده‌است، (*mbits* ۲) زمان ورود ما بین دو جلسه کاری، منبع، مقصد و دو بسته از زمان ورود در طی یک جلسه کاری محاسبه می‌شود. اندازه بسته داده‌ها ۵۱۲ بایت، اندازه عامل زنبور عسل ۴۸ بایت است. اندازه صف در تمام آزمایش ۵۰۰*Kbytes* است. برای تغییر الگوهای دینامیکی ترافیک داده‌ها ما دو حالت تعریف کرده‌ایم: یکنواخت *Uniform*، دارای وزن *Weighted*. هر وضعیت ۱۰ ثانیه ادامه می‌یابد، سپس یک انتقال حالت به وضعیت دیگر رخ می‌دهد. در وضعیت *Uniform* (*U*) یک مقصد از یک توزیع یکنواخت انتخاب شده‌است، تا موقعی که در وضعیت *Weighted* (*W*) یک مقصد انتخاب گردد، در وضعیت *Uniform* بر روی مقصدهای دیگر قرار داده می‌شود. بکاربردن این شیوه در یک محیط تجربی ارزیابی بهتری را نسبت به *AntNet* فراهم می‌کند. برنامه شبیه‌ساز بکار گرفته شده در الگوریتم ۴-۱ نمایش داده شده است:




شکل ۴-۱. اینترنت ستون فقرات *NTTNet* ژاپن

```
package networks;
import node.Node;
import ned.DatarateChannel;
network Net60
{
   types:
      channel C extends DatarateChannel
      {
```

www.Prozhe.com

www.Prozhe.com



```
        parameters:
            delay = 2ms;
            datarate = 6 Mbps;
    }
  submodules:
    rte[57]: Node {
        parameters:
            address = index;
    }
  connections allowunconnected:
    rte[0].port++ <--> C <--> rte[1].port++;
    rte[1].port++ <--> C <--> rte[2].port++;
    rte[1].port++ <--> C <--> rte[4].port++;
    rte[3].port++ <--> C <--> rte[4].port++;
    rte[4].port++ <--> C <--> rte[5].port++;
    rte[4].port++ <--> C <--> rte[7].port++;
    rte[5].port++ <--> C <--> rte[6].port++;
    rte[5].port++ <--> C <--> rte[10].port++;
    rte[6].port++ <--> C <--> rte[7].port++;
    rte[6].port++ <--> C <--> rte[9].port++;
    rte[7].port++ <--> C <--> rte[8].port++;
    rte[7].port++ <--> C <--> rte[12].port++;
    rte[9].port++ <--> C <--> rte[11].port++;
    rte[10].port++ <--> C <--> rte[11].port++;
    rte[10].port++ <--> C <--> rte[13].port++;
    rte[11].port++ <--> C <--> rte[12].port++;
    rte[11].port++ <--> C <--> rte[14].port++;
    rte[12].port++ <--> C <--> rte[15].port++;
    rte[13].port++ <--> C <--> rte[18].port++;
    rte[14].port++ <--> C <--> rte[15].port++;
    rte[14].port++ <--> C <--> rte[22].port++;
    rte[15].port++ <--> C <--> rte[16].port++;
    rte[16].port++ <--> C <--> rte[17].port++;
    rte[16].port++ <--> C <--> rte[23].port++;
    rte[18].port++ <--> C <--> rte[19].port++;
    rte[18].port++ <--> C <--> rte[21].port++;
    rte[19].port++ <--> C <--> rte[20].port++;
    rte[19].port++ <--> C <--> rte[26].port++;
    rte[20].port++ <--> C <--> rte[21].port++;
    rte[20].port++ <--> C <--> rte[24].port++;
    rte[20].port++ <--> C <--> rte[25].port++;
    rte[21].port++ <--> C <--> rte[22].port++;
    rte[21].port++ <--> C <--> rte[23].port++;
    rte[21].port++ <--> C <--> rte[24].port++;
    rte[23].port++ <--> C <--> rte[24].port++;
    rte[24].port++ <--> C <--> rte[28].port++;
    rte[25].port++ <--> C <--> rte[27].port++;
    rte[26].port++ <--> C <--> rte[32].port++;
    rte[27].port++ <--> C <--> rte[29].port++;
    rte[27].port++ <--> C <--> rte[30].port++;
    rte[27].port++ <--> C <--> rte[32].port++;
    rte[28].port++ <--> C <--> rte[29].port++;
    rte[28].port++ <--> C <--> rte[31].port++;
    rte[29].port++ <--> C <--> rte[31].port++;
    rte[30].port++ <--> C <--> rte[36].port++;
    rte[31].port++ <--> C <--> rte[33].port++;
    rte[32].port++ <--> C <--> rte[34].port++;
    rte[33].port++ <--> C <--> rte[37].port++;
```

www.Prozhe.com

www.Prozhe.com



```
    rte[34].port++ <--> C <--> rte[35].port++;
    rte[34].port++ <--> C <--> rte[40].port++;
    rte[35].port++ <--> C <--> rte[36].port++;
    rte[35].port++ <--> C <--> rte[39].port++;
    rte[36].port++ <--> C <--> rte[37].port++;
    rte[37].port++ <--> C <--> rte[38].port++;
    rte[38].port++ <--> C <--> rte[39].port++;
    rte[38].port++ <--> C <--> rte[42].port++;
    rte[39].port++ <--> C <--> rte[40].port++;
    rte[39].port++ <--> C <--> rte[41].port++;
    rte[40].port++ <--> C <--> rte[43].port++;
    rte[40].port++ <--> C <--> rte[44].port++;
    rte[40].port++ <--> C <--> rte[52].port++;
    rte[41].port++ <--> C <--> rte[42].port++;
    rte[41].port++ <--> C <--> rte[43].port++;
    rte[42].port++ <--> C <--> rte[43].port++;
    rte[43].port++ <--> C <--> rte[46].port++;
    rte[44].port++ <--> C <--> rte[45].port++;
    rte[44].port++ <--> C <--> rte[47].port++;
    rte[45].port++ <--> C <--> rte[46].port++;
    rte[46].port++ <--> C <--> rte[49].port++;
    rte[47].port++ <--> C <--> rte[48].port++;
    rte[48].port++ <--> C <--> rte[50].port++;
    rte[48].port++ <--> C <--> rte[51].port++;
    rte[49].port++ <--> C <--> rte[50].port++;
    rte[50].port++ <--> C <--> rte[53].port++;
    rte[51].port++ <--> C <--> rte[52].port++;
    rte[51].port++ <--> C <--> rte[53].port++;
    rte[52].port++ <--> C <--> rte[53].port++;
    rte[52].port++ <--> C <--> rte[54].port++;
    rte[53].port++ <--> C <--> rte[56].port++;
    rte[54].port++ <--> C <--> rte[55].port++;
    rte[55].port++ <--> C <--> rte[56].port++;
}
```

الگوریتم ۱-۴. اینترنت ستون فقرات ژاپن، شبیه سازی شده در *OMNeT++*

## ۲-۴ نتایج آزمایش

اکنون نتایج شبیه‌سازی را گزارش خواهیم‌داد. *MSIA* (*mean of session inter-arrival times*) ، *MPIA* (*mean of packet inetr-arrival times*) از توزیع‌هایی به صورت نمایی منفی با میانیگن به ترتیب *MSIA*، *MPIA* برداشته شده‌اند. نتایج همه گزارشات میانگین مقدارهای بدست آمده از ده اجرای مستقل است. گزارش بسته‌های تحویل داده شده (در اوج بار) برابر در صد بسته‌های تحویل شده که واقعاً تحویل داده شده‌اند است.

### ۱-۲-۴ بارهای اشباع کننده (*Saturating Loads*)

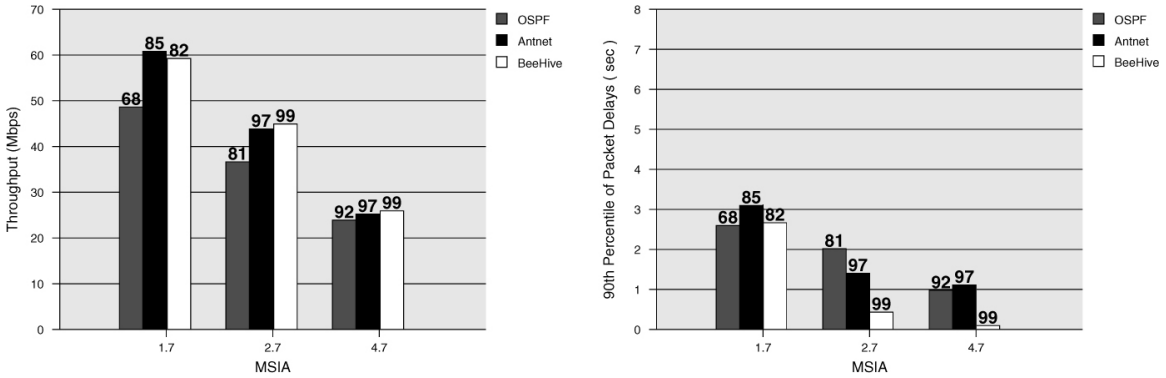
هدف از انجام آزمایش این بود که، رفتار الگوریتم‌ها را بوسیله افزایش بار ترافیک،با کاهش *MSIA* از ۴.۷ به ۱.۷ ثانیه مطالعه کنیم. در طی این آزمایش *MPIA* ۰.۰۰۵ ثانیه است. شکل ۲-۴ نمایش

www.Prozhe.com

www.Prozhe.com



توان عملیاتی میانگین و ۹۰ درصدی از توزیع تاخیر بسته‌ای است. آنچه از شکل ۲-۴ آشکار است، *BeeHive* همان شماره بسته داده ها *AntNet* را دریافت نموده‌است، اما با تاخیر بسته کمتر تحویل داده‌است. *OSFP* ناتوان است که از عهده یک بارکاری اشباع شده برآیند.

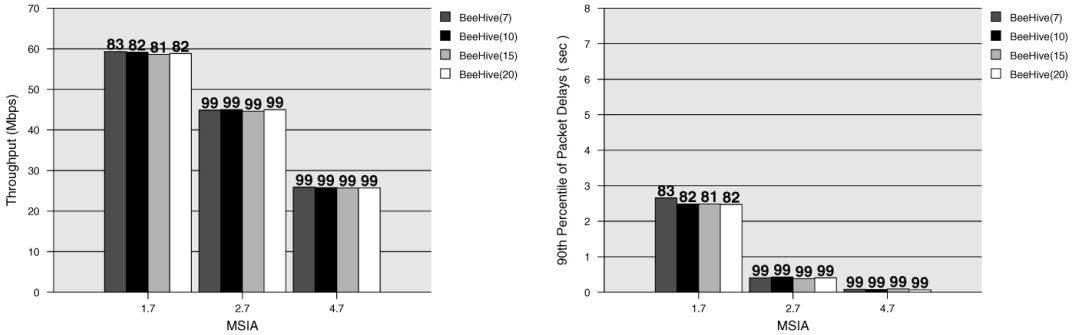



(a) Average Throughput (b) 90th percentile of delays distribution

شکل ۲-۴. رفتار الگوریتم‌ها در مقابل بارهای اشباع کننده

## ۲-۲-۴ اندازه بخش‌های کاوش

بعد از آنکه اندازه هر منطقه کاوش را تجزیه کردیم، عامل زنبور عسل جدول مسیریابی را به روز درمی‌آورد. نتایج برای اندازه‌های ۷، ۱۰، ۱۵ و ۲۰ گام در شکل ۳-۴ گزارش شده‌است. شکل نشان می‌دهد، افزایش اندازه منطقه کاوش بیشتر از ۱۰ کارایی معنی‌داری را بدست نمی‌آورد. در این نمایش توان *BeeHive* تنها در اندازه ۷ گام بهینه می‌باشد.




(a) Average Throughput (b) 90th percentile of delays distribution

شکل ۳-۴. اندازه بخش‌های کاوش

## ۳-۲-۴ اندازه جدول مسیریابی

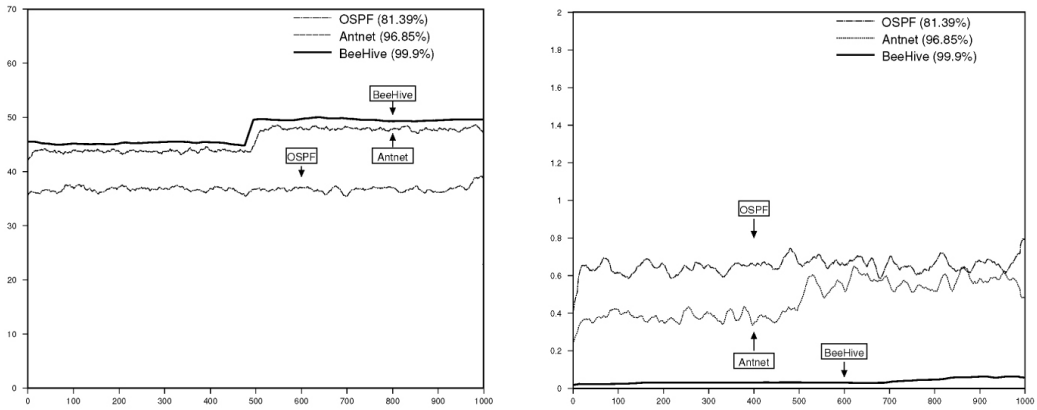
انگیزه اصلی در ایجاد مفهوم منطقه کاوش تنها این نبود، که نیاز برای اطلاع رسانی سراسری را حذف کند، بلکه نیاز به حافظه ذخیره شده یک جدول مسیریابی را بکاهد. *BeeHive* در مناطق خوراک‌جویی با ۸۸، ۹۴ و ۱۰۴ ورودی به طور میانگین در جدول مسیریابی به اندازه ۷، ۱۰ و ۲۰ گام

www.Prozhe.com

www.Prozhe.com



نیاز دارد. هم اکنون *OSFP* به ۵۷ ورودی نیاز دارد، در حالی که *AntNet* به ۱۶۲ ورودی به طور میانگین نیاز دارد. به این دلیل *BeeHive* از لحاظ کارایی همانند *AntNet* و از لحاظ اندازه جدول مسیریابی همانند *OSPF* است.

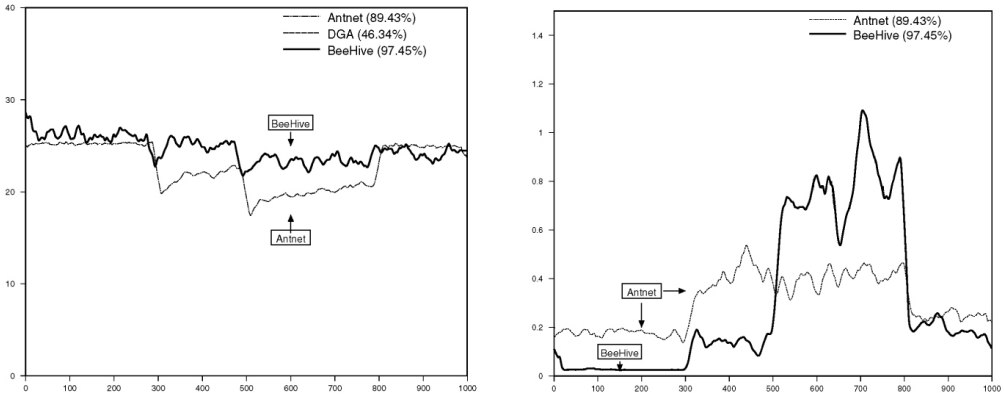



(a) Average Throughput (b) Average Packet Delay

شکل ۴-۴. اندازه جداول مسیریابی

## ۴-۲-۴ نقاط خطرناک (*Hot Spot*)

منظور از انجام این آزمایش آن است که اثر بارهای اضافی ناپایدار در یک شبکه مطالعه شود. ما ۴ نقطه را به عنوان نقاط خطرناک انتخاب کرده‌ایم. (شکل ۱-۴) نقاط خطرناک فعالیتشان از ۵۰۰ ثانیه تا ۱۰۰۰ ثانیه است و تمام گره‌ها داده‌هایشان را به گره ۴ با *MPIA=0.05s* می‌فرستند. این بار اضافی روی یک بار کاری طبیعی با *MSIA=2.7s* و *MPIA=0.03s* اعمال می‌شود. شکل ۵-۴ نشان می‌دهد، که هر دوی *BeeHive* و *AntNet* از عهده بار اضافی برآیند هر چند میانگین تاخیر بسته‌ای برای *BeeHive* کمتر از ۱۰۰ میلی ثانیه است، درحالیکه برای *AntNet* ۵۰۰ میلی ثانیه است.




(a) Average Throughput (b) Average Packet Delay

شکل ۵-۴. نقاط خطرناک الگوریتم *AntNet* ، *BeeHive*

www.Prozhe.com

www.Prozhe.com



## ۴-۴-۵ از کار افتادن مسیریاب (*Router Crash*)

در این آزمایش رفتار تحمل خرابی الگوریتم *BeeHive* بررسی می‌شود، بنابراین *MSIA=4.7s* و *MPIA=0.005* قرار دادیم تا اطمینان حاصل شود بخاطر تراکم هیچ بسته‌ای از بین نمی‌رود. یک سناریو که در آن مسیریاب ۲۱ در ثانیه ۳۰۰ و مسیریاب ۴۰ در ثانیه ۵۰۰ از کار افتاند را شبیه سازی، سپس هر دو طی ۸۰۰ ثانیه تعمیر شدند. شکل ۶-۴ نتیجه را نشان می‌دهد. الگوریتم *BeeHive* قادر است ۹۷٪ بسته ها را تحویل دهد؛ درمقایسه با ۸۹٪ *AntNet* . همانطور که مشاهده می‌کنید از ثانیه ۳۰۰ تا ثانیه ۵۰۰ (مسیریاب ۲۱ ازکار افتاده‌است)، *BeeHive* توان عملیاتی بالاتری دارد و تاخیر بسته کمتر اما وقتیکه مسیریاب ۴۰ از کارافتاد تاخیر بسته ای به ۵۰ افزایش می‌یابد، بارکاری بالاتر در مسیریاب ۴۳ است. آنچه از شکل ۱-۴ آشکار است تنها مسیر شبکه مسیریاب ۴۳ از طریق مسیر مسیریاب ۴۰ از کارافتاده است. ازآنجا که *BeeHive* قادر است بسته‌های بیشتری تحویل دهد طول صف بسته‌ها در مسیریاب ۴۳ افزایش یافته‌است و نسبت به *AntNet* با تاخیر بیشتری بسته‌ها را هدایت کند. همانطور که مشاهده می‌کنید مسیریاب ۲۱ بحرانی (*Critical*) است، اما در صورت مختل شدنش مسیرهای چند گانه از طریق توپولوژی ۱۵، ۱۸، ۱۹، ۲۰، ۲۴ برقرار است.

## ۴-۴-۶ هزینه سربار *BeeHive*

هزینه سربار نسبت ما بین پهنای باند اشغال شده توسط بسته‌های مسیریاب به پهنای باند در دسترس شبکه تعریف می‌شود. هزینه سربار *BeeHive* ۰.۰۰۶۳۳ است در مقایسه با ۰.۰۰۲۸۵ *AntNet*. *BeeHive* یک بیت سرباربالاتری نسبت به *AntNet* دارد، اما این سربار بالاتر برای توپولوژی معین باقی می‌میاند.

www.Prozhe.com