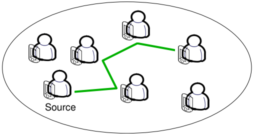
Searching a Needle in (Linear) Opportunistic Networks



Esa Hyytiä†, Suzan Bayhan‡, Jörg Ott†
and Jussi Kangasharju‡

Aalto University, Finland †
University of Helsinki, Finland ‡

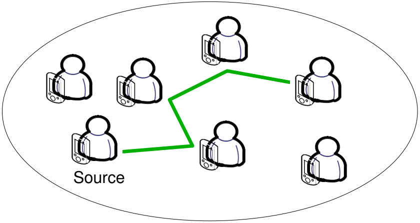
The 17th ACM International Conference on Modeling, Analysis
and Simulation of Wireless and Mobile Systems (MSWiM 2014)

Outline

1. **Motivation**: searching content in opportunistic networks
2. **Searching a specific content**
3. **General search with partial answers**
4. **Conclusions**

1. Motivation

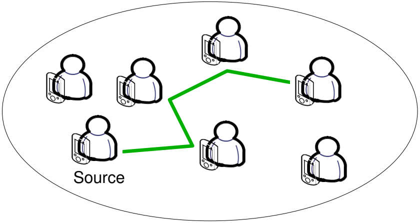
Searching content in opportunistic networks



Scenario

- Opportunistic networking with mobile nodes

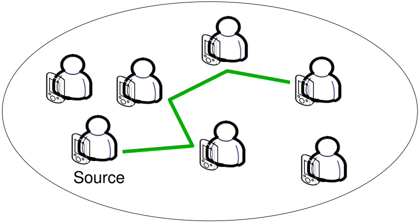
Searching content in opportunistic networks



Scenario

- Opportunistic networking with mobile nodes
- Nodes may carry information that is relevant also to other nodes

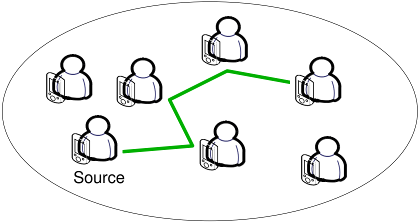
Searching content in opportunistic networks



Scenario

- Opportunistic networking with mobile nodes
- Nodes may carry information that is relevant also to other nodes
- Searching involves
  - **Query** that tries to locate the information

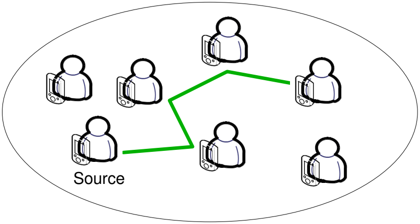
Searching content in opportunistic networks



Scenario

- Opportunistic networking with mobile nodes
- Nodes may carry information that is relevant also to other nodes
- Searching involves
  - **Query** that tries to locate the information
  - **Response** delivers the content back to the source

Searching content in opportunistic networks



Scenario

- Opportunistic networking with mobile nodes
- Nodes may carry information that is relevant also to other nodes
- Searching involves
  - **Query** that tries to locate the information
  - **Response** delivers the content back to the source

Dynamic distributed decision making problem with partial information!

Model: Linear Network

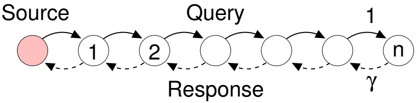


Figure 1: Query travels to right and a possible response to left.

Model: Linear Network

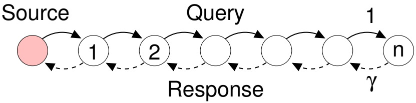


Figure 1: Query travels to right and a possible response to left.

Cost structure:

- Each transmission has a unit cost e

Model: Linear Network

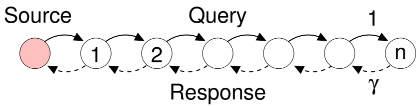


Figure 1: Query travels to right and a possible response to left.

Cost structure:

- Each transmission has a unit cost e
- Transmissions to right succeed with probability of 1
  - “You always find **some node** to transmit to”

Model: Linear Network

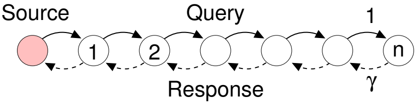


Figure 1: Query travels to right and a possible response to left.

Cost structure:

- Each transmission has a unit cost e
- Transmissions to right succeed with probability of 1
  - “You always find **some node** to transmit to”
- Return path is unreliable (cf. mobility)
  - Each link remains “available” at fixed probability of γ

Model: Linear Network

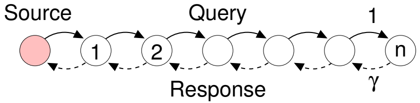


Figure 1: Query travels to right and a possible response to left.

Cost structure:

- Each transmission has a unit cost e
- Transmissions to right succeed with probability of 1
  - “You always find **some node** to transmit to”
- Return path is unreliable (cf. mobility)
  - Each link remains “available” at fixed probability of γ
- If the content is recovered, it has some value v

Model: Linear Network

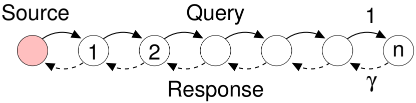


Figure 1: Query travels to right and a possible response to left.

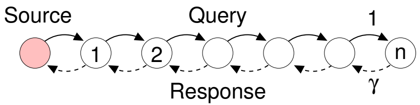
Cost structure:

- Each transmission has a unit cost *e*
- Transmissions to right succeed with probability of 1
  - "You always find **some node** to transmit to"
- Return path is unreliable (cf. mobility)
  - Each link remains "available" at fixed probability of γ
- If the content is recovered, it has some value *v*

The net gain is $\boxed{\text{“value of content” - “transmission costs”}}$

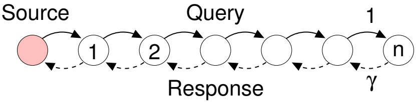
2. Searching Specific Content

Searching specific content



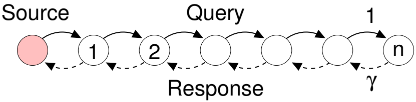
- Bernoulli case, $V_i = 0$ or $V_i = 1$

Searching specific content



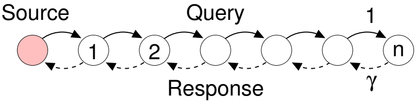
- Bernoulli case, $V_i = 0$ or $V_i = 1$
  - A node either has the content, or not

Searching specific content



- Bernoulli case, $V_i = 0$ or $V_i = 1$
  - A node either has the content, or not
  - Single availability parameter $p = \mathrm{P}\{V_i = 1\}$

Searching specific content



- Bernoulli case, $V_i = 0$ or $V_i = 1$
  - A node either has the content, or not
  - Single availability parameter $p = \mathrm{P}\{V_i = 1\}$

What is the optimal search strategy?

Static strategy

Static Search Strategy

- *Search n nodes, and return the best answer*

Static strategy

Static Search Strategy

- *Search n nodes, and return the best answer*

- Not optimal! Irrational to continue if content already found

Static strategy

Static Search Strategy

- *Search n nodes, and return the best answer*

- Not optimal! Irrational to continue if content already found
- Nonetheless, the optimal search depth can be derived

Static strategy

Static Search Strategy

- *Search n nodes, and return the best answer*

- Not optimal! Irrational to continue if content already found
- Nonetheless, the optimal search depth can be derived

$$\boxed{n^* = \left\lceil \frac{\log(2e/p)}{\log(1-p)} \right\rceil} \qquad (p > 2e) \qquad (1)$$

- p = availability of the content (value normalized to 1)
- e = unit transmission cost

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen
- Assuming that p is known *a priori*,

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen
- Assuming that p is known *a priori*,

$$\boxed{n^* = \left\lceil \frac{1}{e} - \frac{1}{p} \right\rceil - 1} \qquad (2)$$

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen
- Assuming that p is known *a priori*,

$$\boxed{n^* = \left\lceil \frac{1}{e} - \frac{1}{p} \right\rceil - 1} \tag{2}$$

- In practice, we do not know p *a priori*

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen
- Assuming that p is known *a priori*,

$$\boxed{n^* = \left\lceil \frac{1}{e} - \frac{1}{p} \right\rceil - 1} \tag{2}$$

- In practice, we do not know p *a priori*
  - It depends on the content searched!

Dynamic strategy

Dynamic Search Strategy

- *Search at most n nodes*
- *When content is found*
  - *Terminate the query*
  - *Return the content*

- Rational policy when n is properly chosen
- Assuming that p is known *a priori*,

$$\boxed{n^* = \left\lceil \frac{1}{e} - \frac{1}{p} \right\rceil - 1} \tag{2}$$

- In practice, we do not know p *a priori*
  - It depends on the content searched!
- Idea: *learn p* as the search progresses!

Learning strategy

Learning Search Strategy

- *At every step i, update our estimate on $\hat{p}_i$*
- *If $\hat{p}_i$ is "too low", terminate the search*
- *Otherwise, when content is found*
  - *Terminate the query*
  - *Return the content*

Learning strategy

Learning Search Strategy

- *At every step i, update our estimate on $\hat{p}_i$*
- *If $\hat{p}_i$ is "too low", terminate the search*
- *Otherwise, when content is found*
  - *Terminate the query*
  - *Return the content*

- After some mathematical manipulations . . .

Learning strategy

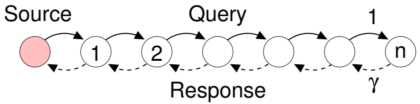
Learning Search Strategy

- *At every step i, update our estimate on $\hat{p}_i$*
- *If $\hat{p}_i$ is "too low", terminate the search*
- *Otherwise, when content is found*
  - *Terminate the query*
  - *Return the content*

- After some mathematical manipulations ...

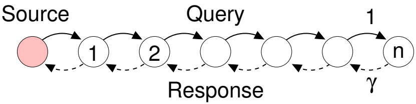
$$\boxed{n^* = \left\lceil \frac{1}{2e} \right\rceil - 2} \qquad (e < 0.25) \tag{3}$$

Search strategies with a general γ



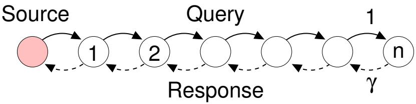
- Previous results assume "no mobility", $\gamma = 1$
  - The return path always exists and is reliable

Search strategies with a general γ



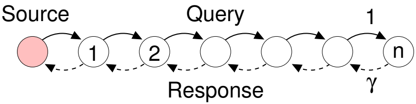
- Previous results assume "no mobility", $\gamma = 1$
  - The return path always exists and is reliable
- In general case, $0 < \gamma \leq 1$

Search strategies with a general γ



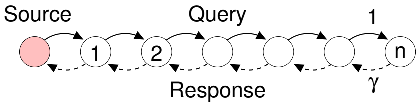
- Previous results assume "no mobility", $\gamma = 1$
  - The return path always exists and is reliable
- In general case, $0 < \gamma \leq 1$
- All three strategies can be analyzed

Search strategies with a general γ



- Previous results assume “no mobility”, $\gamma = 1$
  - The return path always exists and is reliable
- In general case, $0 < \gamma \leq 1$
- All three strategies can be analyzed
  - Optimal (max.) search depth can be determined

Search strategies with a general γ



- Previous results assume “no mobility”, $\gamma = 1$
  - The return path always exists and is reliable
- In general case, $0 < \gamma \leq 1$
- All three strategies can be analyzed
  - Optimal (max.) search depth can be determined
  - See details in the paper

Example #1

- Searching specific content:
  1. Value $v = 1$ if retrieved
  2. Otherwise $v = 0$

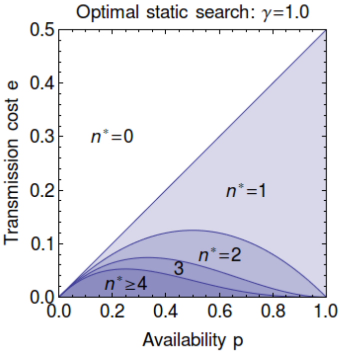
Example #1

- Searching specific content:
  1. Value $v = 1$ if retrieved
  2. Otherwise $v = 0$
- Two scenarios:
  1. "no mobility", $\gamma = 1$
  2. "low mobility", $\gamma = 0.7$

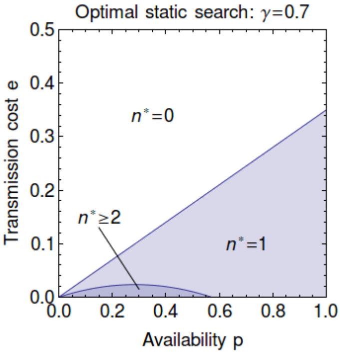
Example #1

- Searching specific content:
  1. Value $v = 1$ if retrieved
  2. Otherwise $v = 0$
- Two scenarios:
  1. "no mobility", $\gamma = 1$
  2. "low mobility", $\gamma = 0.7$
- Unit transmission cost e is a free parameter

Example #1: Static strategy

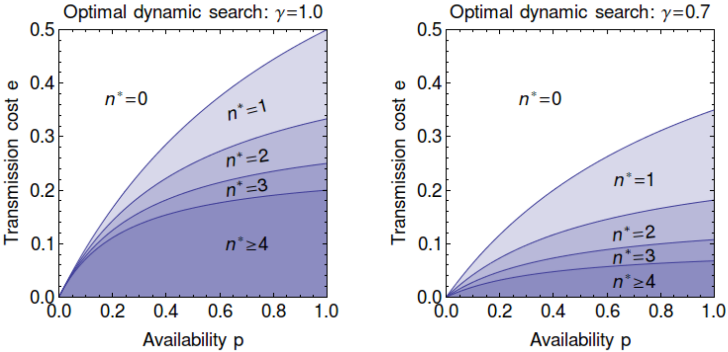


(a) No mobility ($\gamma = 1$)



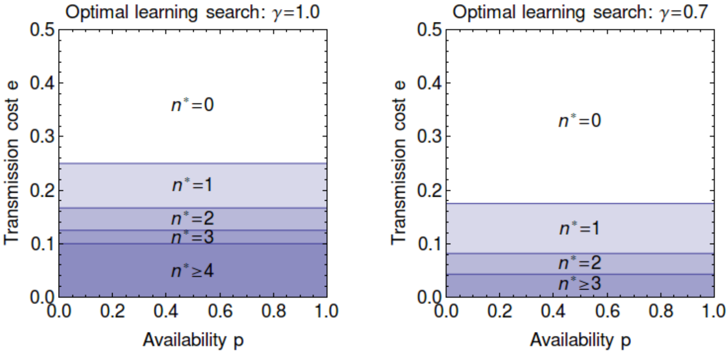
(b) Low mobility ($\gamma = 0.7$)

Example #1: Dynamic strategy



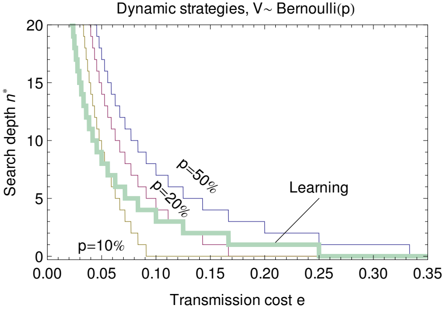
(a) No mobility ($\gamma = 1$) (b) Low mobility ($\gamma = 0.7$)

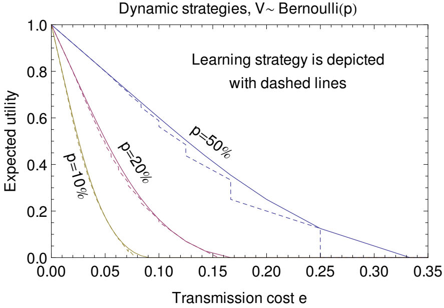
Example #1: Learning strategy



(a) No mobility ($\gamma = 1$) (b) Low mobility ($\gamma = 0.7$)

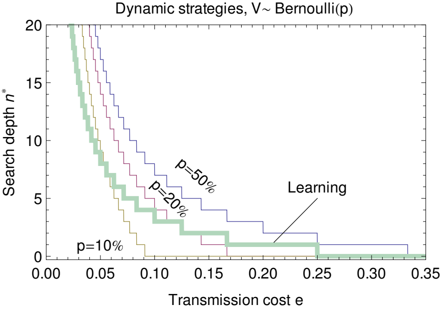
Example #1: Performance

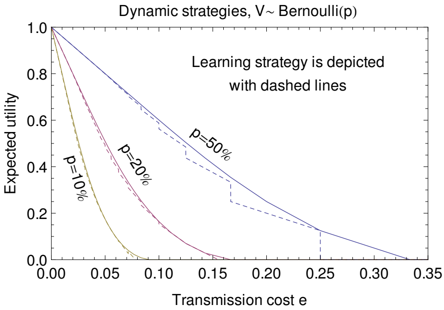




- Learning strategy operates “between” the dynamic policies

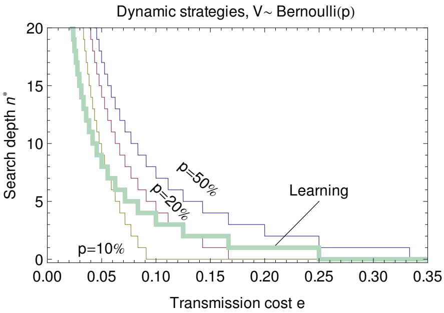
Example #1: Performance

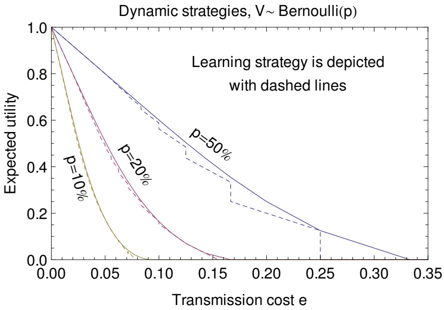




- Learning strategy operates “between” the dynamic policies
- Performance of the learning strategy close to the optimal!

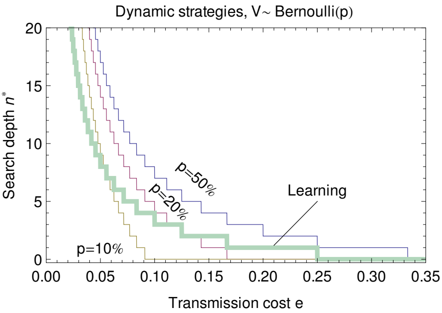
Example #1: Performance

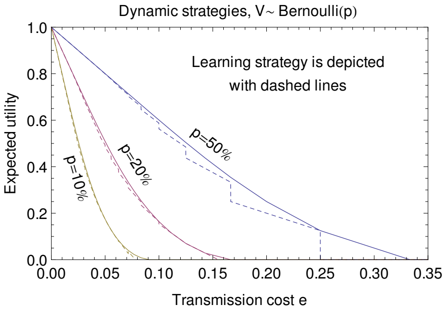




- Learning strategy operates “between” the dynamic policies
- Performance of the learning strategy close to the optimal!
  - Largest differences when p is high;

Example #1: Performance





- Learning strategy operates "between" the dynamic policies
- Performance of the learning strategy close to the optimal!
  - Largest differences when *p* is high;
  - Short searches and no time to learn

3. General Search with Partial Answers

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)
- Let $z = (m, n, d, b)$ denote **the state** of the search
  - m is the number of transmissions so far
  - n is the distance to the source (in hops)
  - d is the highest valued response already sent
  - b is the highest valued response available now,

$$b = \max\{V_1, \ldots, V_n\}$$

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)
- Let $z = (m, n, d, b)$ denote **the state** of the search
  - m is the number of transmissions so far
  - n is the distance to the source (in hops)
  - d is the highest valued response already sent
  - b is the highest valued response available now,

$$b = \max\{V_1, \ldots, V_n\}$$

- Search strategy defines for each state:

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)
- Let $z = (m, n, d, b)$ denote **the state** of the search
  - m is the number of transmissions so far
  - n is the distance to the source (in hops)
  - d is the highest valued response already sent
  - b is the highest valued response available now,

$$b = \max\{V_1, \ldots, V_n\}$$

- Search strategy defines for each state:
  1. Whether to send a response with value b

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)
- Let $z = (m, n, d, b)$ denote **the state** of the search
  - m is the number of transmissions so far
  - n is the distance to the source (in hops)
  - d is the highest valued response already sent
  - b is the highest valued response available now,

$$b = \max\{V_1, \ldots, V_n\}$$

- Search strategy defines for each state:
  1. Whether to send a response with value b
  2. Whether to continue further to node $n + 1$

Search in general case

- Each node i has an answer with value $V_i \sim V$ (i.i.d.)
- Let $z = (m, n, d, b)$ denote **the state** of the search
  - m is the number of transmissions so far
  - n is the distance to the source (in hops)
  - d is the highest valued response already sent
  - b is the highest valued response available now,

$$b = \max\{V_1, \ldots, V_n\}$$

- Search strategy defines for each state:
  1. Whether to send a response with value b
  2. Whether to continue further to node $n + 1$

What is the optimal search strategy?

Dynamic programming

Dynamic programming: write the expected final utility for each action, and choose the best among them

Dynamic programming

Dynamic programming: write the expected final utility for each action, and choose the best among them

$$w(m, n, d, b) = \max\{a_1, a_2, a_3, a_4\}$$

where a_j denotes the (expected) final utility with action j

1) stop the search
2) stop the search and send a response back
3) continue the search further to depth $n + 1$
4) continue the search, but also send a response back

Dynamic programming

Dynamic programming: write the expected final utility for each action, and choose the best among them

$$w(m, n, d, b) = \max\{a_1, a_2, a_3, a_4\}$$

where a_j denotes the (expected) final utility with action j

1) stop the search
2) stop the search and send a response back
3) continue the search further to depth $n + 1$
4) continue the search, but also send a response back

Formally,

$$\begin{cases} a_1 = d - m \cdot e \\ a_2 = b - (m + n) \cdot e \\ a_3 = \mathrm{E}[w(m + 1, n + 1, d, \max\{b, V_{n+1}\})] \\ a_4 = \mathrm{E}[w(m + n + 1, n + 1, b, \max\{b, V_{n+1}\})] \end{cases}$$

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

- Applying blindly leads to infinite recursion
  - Actions a_3 and a_4 defined in terms of $w(\cdot, n+1, \cdot, \cdot)$

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

- Applying blindly leads to infinite recursion
  - Actions a_3 and a_4 defined in terms of $w(\cdot, n+1, \cdot, \cdot)$
- However,
  - Actions a_2 and a_4 make no sense when $d = b$

    (no better response than already delivered is available)

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

- Applying blindly leads to infinite recursion
  - Actions a_3 and a_4 defined in terms of $w(\cdot, n+1, \cdot, \cdot)$
- However,
  - Actions a_2 and a_4 make no sense when $d = b$
    (no better response than already delivered is available)
  - Actions a_3 and a_4 can be excluded when n, m are large
    (even the best response would yield negative net gain)

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

- Applying blindly leads to infinite recursion
  - Actions a_3 and a_4 defined in terms of $w(\cdot, n+1, \cdot, \cdot)$
- However,
  - Actions a_2 and a_4 make no sense when $d = b$
    (no better response than already delivered is available)
  - Actions a_3 and a_4 can be excluded when n, m are large
    (even the best response would yield negative net gain)
- Recursion is finite . . .

Dynamic programming (cont.)

The optimal action in state z is given by

$$\arg\max_i \{a_1, a_2, a_3, a_4\}$$

- Applying blindly leads to infinite recursion
  - Actions a_3 and a_4 defined in terms of $w(\cdot, n+1, \cdot, \cdot)$
- However,
  - Actions a_2 and a_4 make no sense when $d = b$
    (no better response than already delivered is available)
  - Actions a_3 and a_4 can be excluded when n, m are large
    (even the best response would yield negative net gain)
- Recursion is finite ...

(Details in the paper)

Example #2

- Let $V_i \sim \mathrm{U}(0,3)$ (values $0, 1, \ldots, 3$)

Example #2

- Let $V_i \sim \mathrm{U}(0, 3)$ (values $0, 1, \ldots, 3$)
- Dynamic programming approach gives the optimal policy

Example #2

- Let $V_i \sim \mathrm{U}(0,3)$ (values $0, 1, \ldots, 3$)
- Dynamic programming approach gives the optimal policy
- Two reference policies

Example #2

- Let $V_i \sim \mathrm{U}(0, 3)$ (values $0, 1, \ldots, 3$)
- Dynamic programming approach gives the optimal policy
- Two reference policies

Static

- *Search n nodes, and return the best answer.*

Example #2

- Let $V_i \sim \mathrm{U}(0,3)$ (values $0, 1, \ldots, 3$)
- Dynamic programming approach gives the optimal policy
- Two reference policies

Static

- *Search n nodes, and return the best answer.*

Dynamic-0

- *Search at most n nodes*
- *Stop immediately if the perfect answer is found*
- *Return the best answer, unless it has no value ($v = 0$)*

Example: $V_i \sim \mathrm{U}(0, 3)$

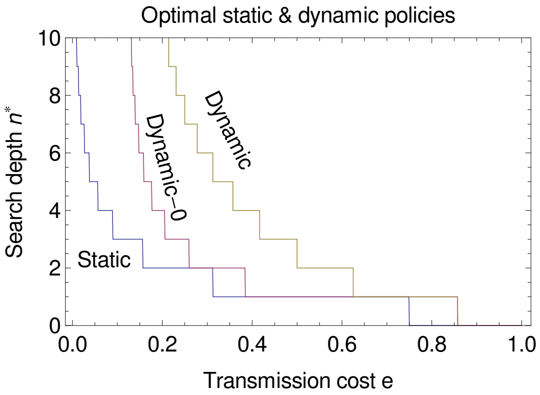


Figure 2: Optimal (max.) search depth with different strategies.

Example: $V_i \sim \mathrm{U}(0,3)$

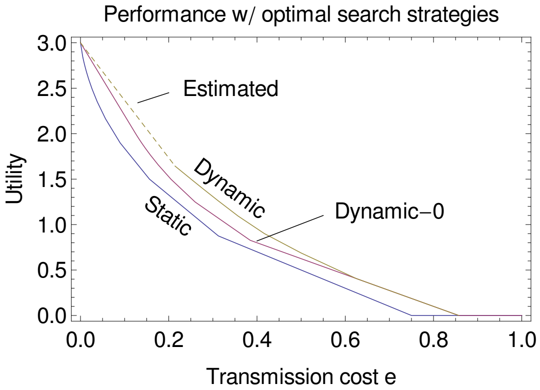


Figure 3: Performance (mean utility) with different strategies.

4. Conclusions

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!

- Consider a single query (no replications)
  - Reduces to a linear network

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!
- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the “optimal stopping” problem(s)

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!
- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the "optimal stopping" problem(s)
  - How far to search?

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!

- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the “optimal stopping” problem(s)
  - How far to search?
  - When to send responses back

Conclusions

- Searching content in opportunistic networks:

Overwhelmingly complicated task!

- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the “optimal stopping” problem(s)
  - How far to search?
  - When to send responses back
- Exact search depths when searching a specific content

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!
- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the “optimal stopping” problem(s)
  - How far to search?
  - When to send responses back
- Exact search depths when searching a specific content
- General case with dynamic programming

Conclusions

- Searching content in opportunistic networks:

  Overwhelmingly complicated task!
- Consider a single query (no replications)
  - Reduces to a linear network
- Formulate the “optimal stopping” problem(s)
  - How far to search?
  - When to send responses back
- Exact search depths when searching a specific content
- General case with dynamic programming
- Future work: concurrent searches, dynamic topology, . . .

Thanks!

http://www.netlab.tkk.fi/tutkimus/pdp/