Efficient algorithms with restricted workspace: shortest paths in grid graphs, using budgeted recursion$^\diamondsuit$

David Kirkpatrick

Department of Computer Science
University of British Columbia

PIMS Workshop on:
Algorithmic Theory of Networks
March, 27-29, 2015

$\diamondsuit$ based on joint work with *Tetsuo Asano*

EFFICIENT ALGORITHMS WITH RESTRICTED WORKSPACE: shortest paths in grid graphs, using budgeted recursion$^\diamondsuit$

David Kirkpatrick

Department of Computer Science
University of British Columbia

PIMS Workshop on:
Algorithmic Theory of Networks
March, 27-29, 2015

$\diamondsuit$ based on joint work with *Tetsuo Asano*

Outline

Introduction
- algorithms for shortest (min-weight) paths
- memory-constrained algorithms

Min-weight paths in grid graphs – Asano&Doerr(2011)
- overview of basic algorithm
- applying a good idea recursively

Min-weight paths in grid graphs – Refinements & Extensions
- a different recursive formulation
- budgeted recursion – exploiting a universal sequence
- combining the ideas

Beyond grid graphs...
- min-weight paths in implicit graphs
- min-weight paths in general planar graphs

Outline

Introduction

- algorithms for shortest (min-weight) paths
- memory-constrained algorithms

Min-weight paths in grid graphs – Asano&Doerr(2011)

- overview of basic algorithm
- applying a good idea recursively

Min-weight paths in grid graphs – Refinements & Extensions

- a different recursive formulation
- budgeted recursion – exploiting a universal sequence
- combining the ideas

Beyond grid graphs...

- min-weight paths in implicit graphs
- min-weight paths in general planar graphs

Single-source shortest (minimum weight) paths and space-bounded computation

The *topic* lies at the confluence of two fundamental streams in the modern theory of algorithms

- algorithms for minimum-weight paths in graphs
- determining the limits of space-bounded computation, including time-space tradeoffs

Our *model* assumes an input graph provided in read-only memory. Space measures the number of (bounded-capacity) reusable words of working memory. We will write $\tilde{O}(s(n))$ space to acknowledge the fact that words typically have capacity $\Theta(\lg n)$.

Single-source shortest (minimum weight) paths and space-bounded computation

The *topic* lies at the confluence of two fundamental streams in the modern theory of algorithms

- algorithms for minimum-weight paths in graphs
- determining the limits of space-bounded computation, including time-space tradeoffs

Our *model* assumes an input graph provided in read-only memory. Space measures the number of (bounded-capacity) reusable words of working memory. We will write $\tilde{O}(s(n))$ space to acknowledge the fact that words typically have capacity $\Theta(\lg n)$.

Single-source shortest (minimum weight) paths and space-bounded computation

The *topic* lies at the confluence of two fundamental streams in the modern theory of algorithms

- algorithms for minimum-weight paths in graphs
- determining the limits of space-bounded computation, including time-space tradeoffs

Our *model* assumes an input graph provided in read-only memory. Space measures the number of (bounded-capacity) reusable words of working memory. We will write $\tilde{O}(s(n))$ space to acknowledge the fact that words typically have capacity $\Theta(\lg n)$.

Single-source shortest (minimum weight) paths and space-bounded computation

The *topic* lies at the confluence of two fundamental streams in the modern theory of algorithms

- algorithms for minimum-weight paths in graphs
- determining the limits of space-bounded computation, including time-space tradeoffs

Our *model* assumes an input graph provided in read-only memory. Space measures the number of (bounded-capacity) reusable words of working memory. We will write $\tilde{O}(s(n))$ space to acknowledge the fact that words typically have capacity $\Theta(\lg n)$.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Algorithms for min-weight paths

Finding min-weight paths (in directed graphs with n vertices and m edges)

- general edge weights $O(nm)$ [Bellman-Ford 1950's]
- non-negative edge weights $O(m + n \lg n)$ [Dijkstra, with Fibonacci heaps 1959; 1984]
- *planar* graphs (with non-negative weights) $O(n)$ [Henzinger et al. 1997]
- small integer weights...

All of these are naturally implemented with $\Omega(n + m)$ workspace.

Memory-constrained algorithms

In addition to the obvious practical advantages of space-efficient algorithms for min-weight paths, the basic graph reachability problem is a core problem in computational complexity theory.

- it is a canonical complete problem for non-deterministic log-space
- the open question **L**=**NL**?, asks if it can be solved deterministically in log-space
- Savitch's algorithm (1970) solves the problem in $O(\lg n)^2)$ space, but requires $n^{\Theta(\lg n)}$ time
- *undirected* graph reachability has a $O(\lg n)$-space solution [Reingold 2008]

Memory-constrained algorithms

In addition to the obvious practical advantages of space-efficient algorithms for min-weight paths, the basic graph reachability problem is a core problem in computational complexity theory.

- it is a canonical complete problem for non-deterministic log-space
- the open question **L**=**NL**?, asks if it can be solved deterministically in log-space
- Savitch's algorithm (1970) solves the problem in $O(\lg n)^2)$ space, but requires $n^{\Theta(\lg n)}$ time
- *undirected* graph reachability has a $O(\lg n)$-space solution [Reingold 2008]

Outline

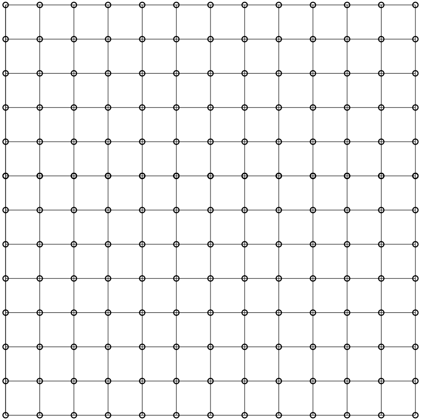
Introduction
algorithms for shortest (min-weight) paths
memory-constrained algorithms

Min-weight paths in grid graphs – Asano&Doerr(2011)
overview of basic algorithm
applying a good idea recursively

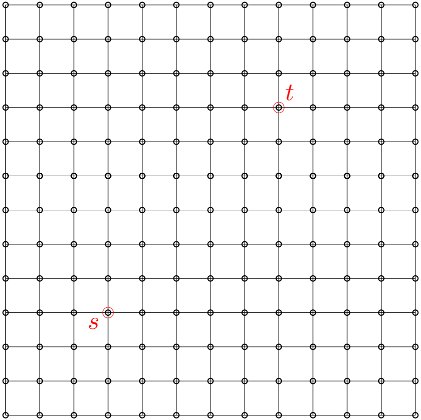
Min-weight paths in grid graphs – Refinements & Extensions
a different recursive formulation
budgeted recursion – exploiting a universal sequence
combining the ideas

Beyond grid graphs...
min-weight paths in implicit graphs
min-weight paths in general planar graphs

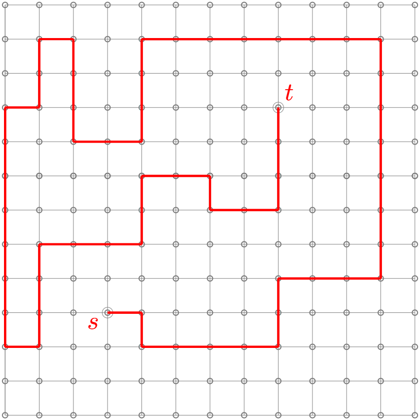
Suppose we are given an edge-weighted grid graph...



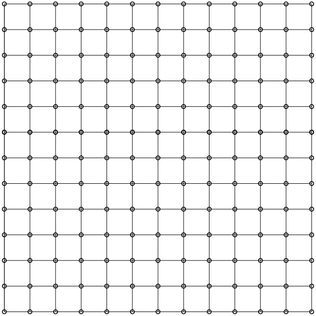
...with two distinguished vertices s and t



We want to find an s-t path of minimum weight

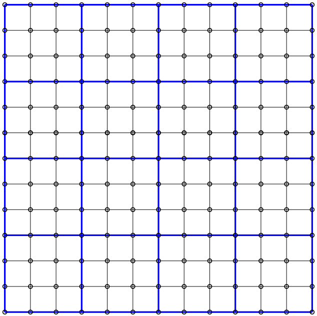


Asano-Doerr algorithm



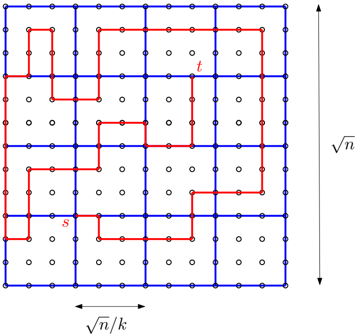
Start with a $\sqrt{n} \times \sqrt{n}$ grid...

Asano-Doerr algorithm (following Fredrickson '87)



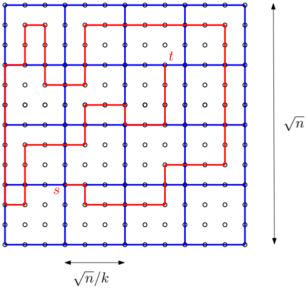
...and partition it into k^2 cells, each of size $\sqrt{n}/k \times \sqrt{n}/k$.

Asano-Doerr algorithm



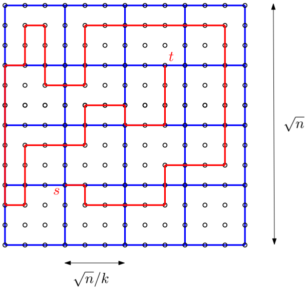
View an *s*-*t* path as a sequence of hops between (cell) boundaries

Asano-Doerr algorithm



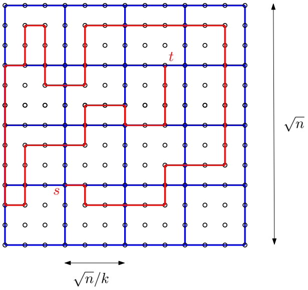
- cell interiors act as quasi-edges connecting boundary vertices
- solve a min-weight path problem on boundary vertices, each "step" of which involves a min-weight path problem (within a cell)

Asano-Doerr algorithm – general edge weights



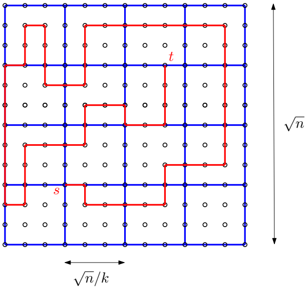
- $O(\sqrt{n}k)$ phases
- each phase involves a "relaxation" of all k^2 quasi-edges
- since each "relaxation" has cost $[(\sqrt{n}/k)^2]^2$, total cost is $O(n^{2.5}/k)$

Asano-Doerr algorithm – non-negative edge weights



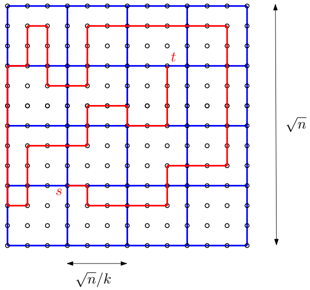
- $O(\sqrt{n}k)$ phases
- each phase involves a "relaxation" of $O(1)$ quasi-edges
- since each "relaxation" costs time $\tilde{O}((\sqrt{n}/k)^2)$, total time is reduced to $\tilde{O}(n^{1.5}/k)$

Asano-Doerr algorithm



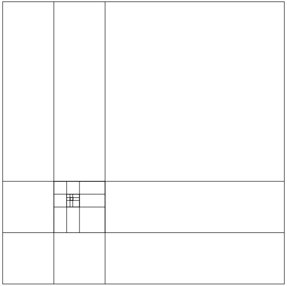
In both cases, space cost is $O(k\sqrt{n} + (\sqrt{n}/k)^2)$,
which is minimized at $O(n^{2/3})$, when $k = n^{1/6}$.

Asano-Doerr algorithm



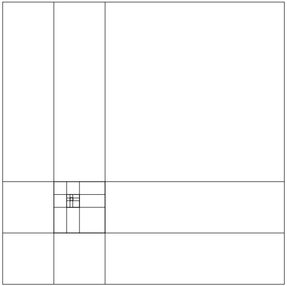
In both cases, space cost is $O(k\sqrt{n} + (\sqrt{n}/k)^2)$, which is minimized at $O(n^{2/3})$, when $k = n^{1/6}$.

Asano-Doerr algorithm – applied recursively



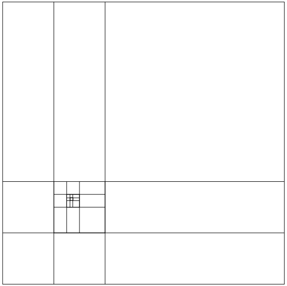
- If the same idea is applied recursively on the cells ($\sqrt{n}/k \times \sqrt{n}/k$ subgrids), with the same splitting factor at m levels of recursion, we get a total time cost of $O(\frac{(\sqrt{n})^m}{k^{m(m+1)/2}} n^2)$ (for general edge weights).

Asano-Doerr algorithm – applied recursively



- The space cost is $O(\sqrt{n}k + n/k^{2m})$, which is minimized when $k = n^{\frac{1}{2(2m+1)}}$, giving space $n^{1/2+\epsilon}$ and time $n^{O(1/\epsilon)}$, when $m = \Theta(1/\epsilon)$.

Asano-Doerr algorithm – optimized



- In fact, if the same idea is applied recursively on the cells with a differentiated splitting factor (chosen to balance the space cost) at each of the m levels of recursion, we get a space cost of $n^{1/2+\epsilon}$ and time $n^{O(\lg(1/\epsilon))}$, when $m = \Theta(\lg(1/\epsilon))$.

Outline

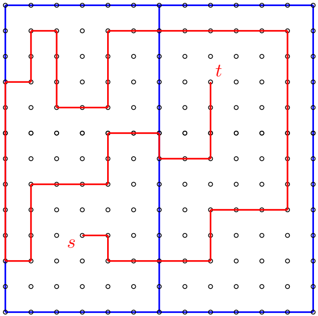
Introduction
- algorithms for shortest (min-weight) paths
- memory-constrained algorithms

Min-weight paths in grid graphs – Asano&Doerr(2011)
- overview of basic algorithm
- applying a good idea recursively

Min-weight paths in grid graphs – Refinements & Extensions
- a different recursive formulation
- budgeted recursion – exploiting a universal sequence
- combining the ideas

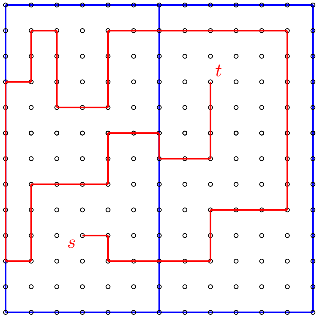
Beyond grid graphs...
- min-weight paths in implicit graphs
- min-weight paths in general planar graphs

Alternative recursive algorithm



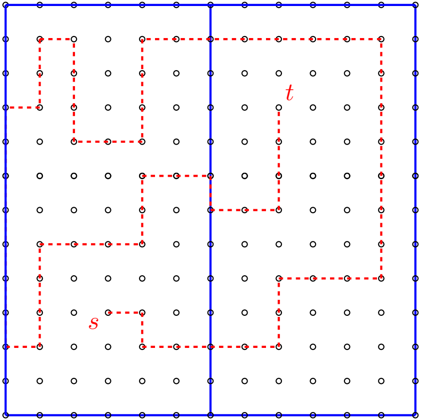
View the *fundamental problem* as one of updating path estimates on boundary vertices (using paths that lie strictly interior to cells).

Alternative recursive algorithm

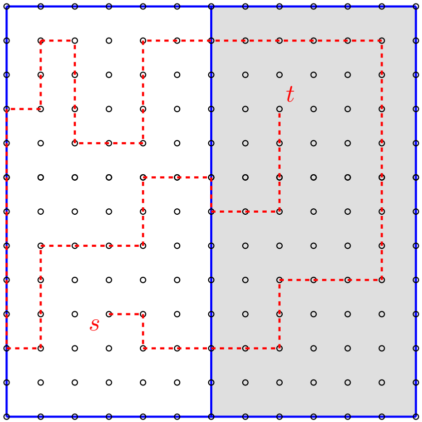


View the *fundamental problem* as one of updating path estimates on boundary vertices (using paths that lie strictly interior to cells).

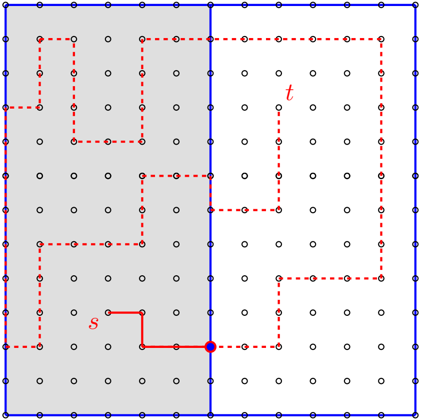
Alternative recursive algorithm



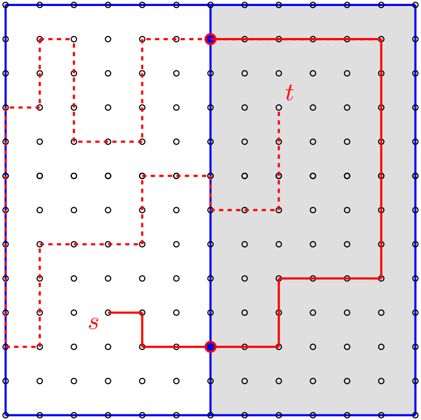
Alternative recursive algorithm



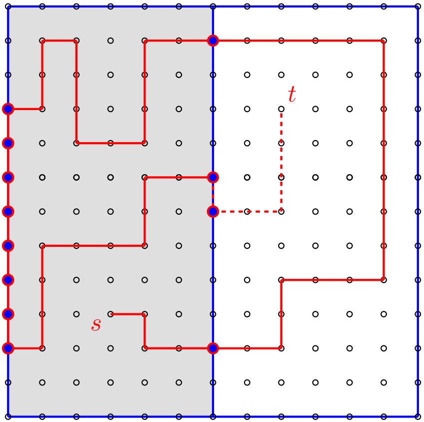
Alternative recursive algorithm



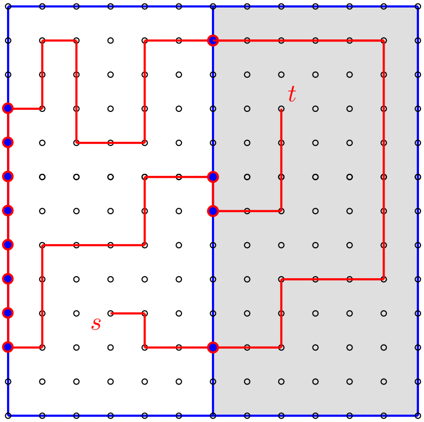
Alternative recursive algorithm



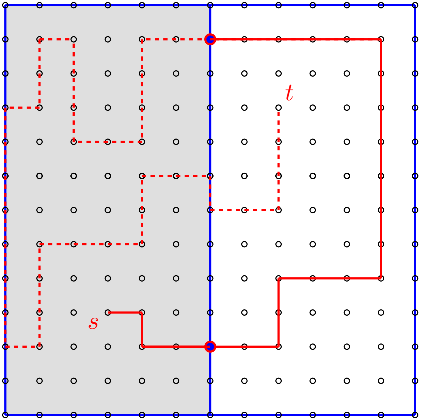
Alternative recursive algorithm



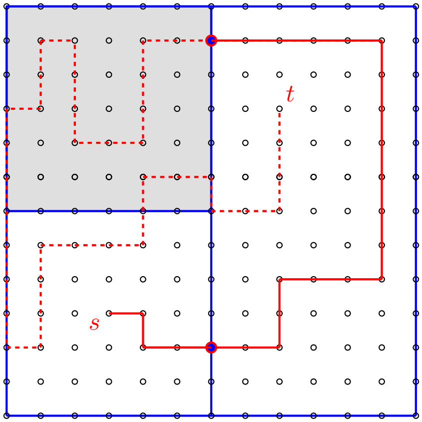
Alternative recursive algorithm



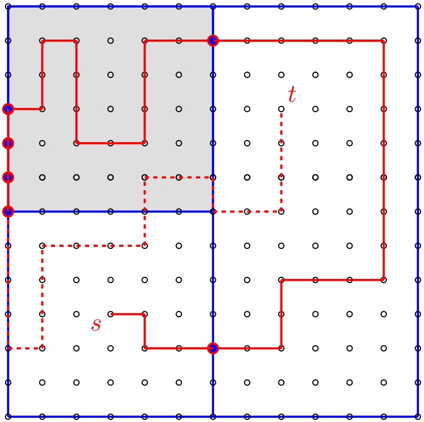
Alternative recursive algorithm



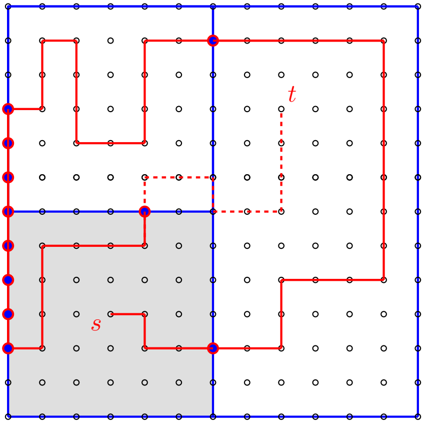
Alternative recursive algorithm



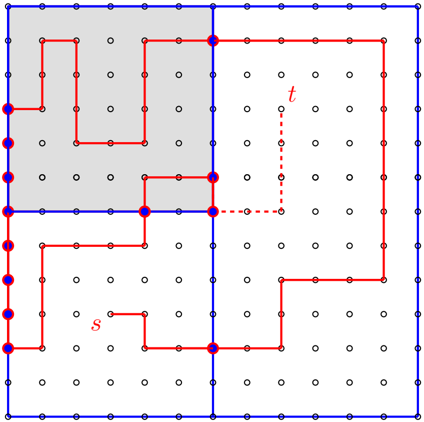
Alternative recursive algorithm



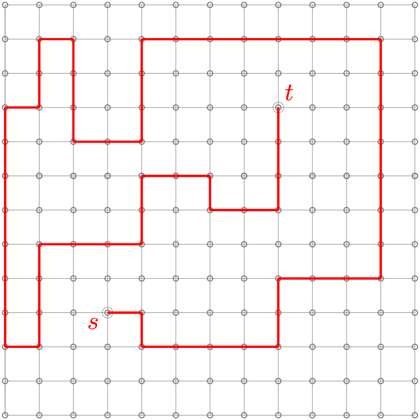
Alternative recursive algorithm



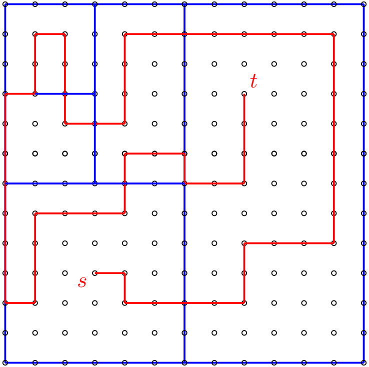
Alternative recursive algorithm



What is the cost of this approach?

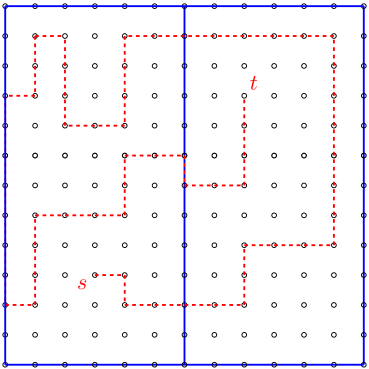


The good news...



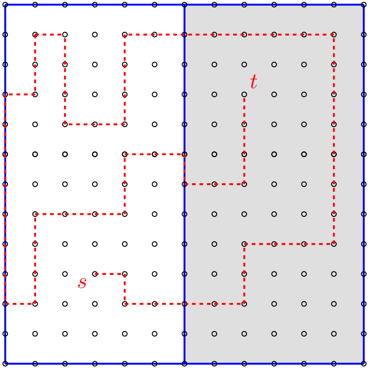
Since we maintain path weights at boundary vertices along one separating line at each level of recursion, the space cost is $O(\sqrt{n})$.

The bad news...



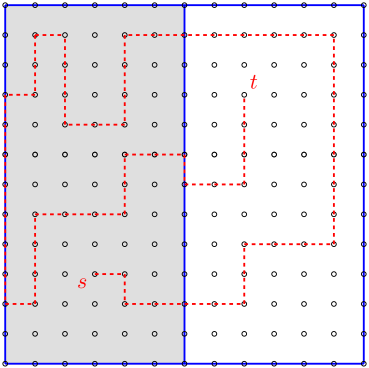
We need to make many (expensive) recursive calls at each level.

The bad news...



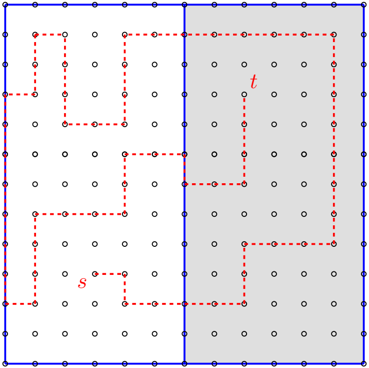
We need to make many (expensive) recursive calls at each level.

The bad news...



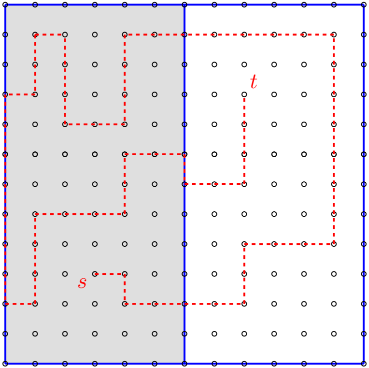
We need to make many (expensive) recursive calls at each level.

The bad news...



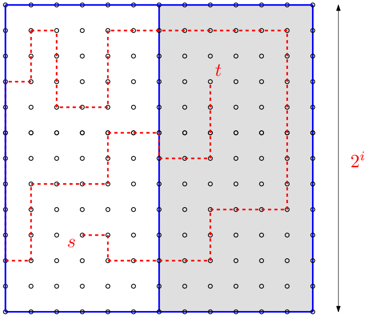
We need to make many (expensive) recursive calls at each level.

The bad news...



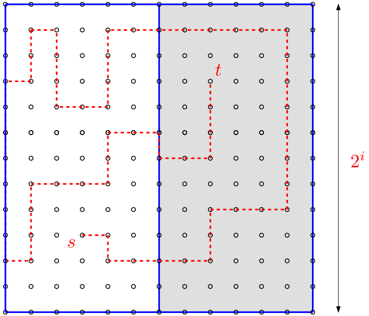
We need to make many (expensive) recursive calls at each level.

The bad news...



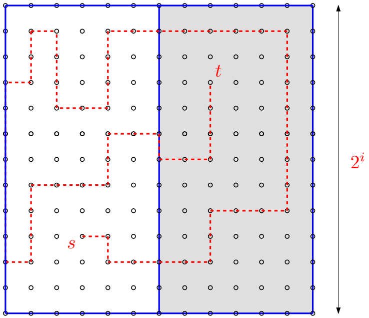
In a $2^i \times 2^i$ grid, a simple path could cross the separating line up to 2^i times. Hence, we need to make $O(2^i)$ recursive calls to subproblems at the next level.

The bad news...



Thus $\mathrm{Cost}(i)$, the cost of finding a min-weight path in a $2^i \times 2^i$ grid, satisfies $\mathrm{Cost}(i) \le 2^i \mathrm{Cost}(i-1)$,
which means $\mathrm{Cost}((\lg n)/2) = n^{O(\lg n)}$.

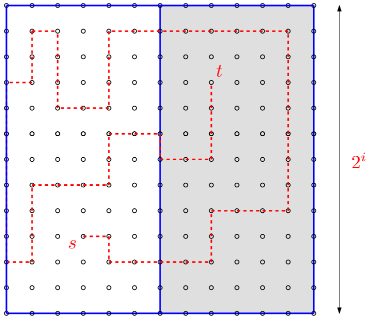
If we were *lucky*...



...we could *guess* the amount of time we should devote to individual recursive calls, so that we do work on a subproblem just when it will pay off...

...but we still would not be able to *certify* the solution

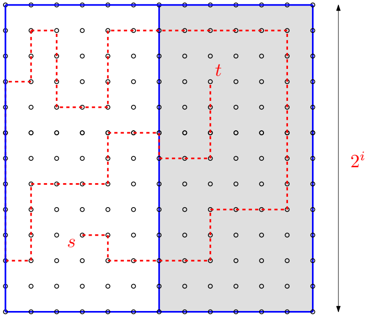
If we were *lucky*...



...we could *guess* the amount of time we should devote to individual recursive calls, so that we do work on a subproblem just when it will pay off...

...but we still would not be able to *certify* the solution

Since we can't count on being lucky...



...instead, we should construct an resource allocation scheme (*budgeted recursion*) that will be sure to subsume all possible optimal budget allocations.

Universal budget sequences...

A sequence of budgets (think bounds on the exploration length of paths) for successive subproblems at the same level of recursion is *universal* if it contains as a subsequence a sequence of budgets that is guaranteed to uncover the minimum-cost path.

- Clearly the sequence $2^{2^i}, 2^{2^i}, \ldots, 2^{2^i}$ of length 2^i is universal.
- However, we can do better...
  Consider instead the sequence σ_{2^i} defined inductively by

$$\sigma_s = \begin{cases} \langle 1 \rangle & \text{if } s = 0, \text{ and} \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & \text{otherwise,} \end{cases}$$

  (where $\diamond$ signifies concatenation of sequences) .

Universal budget sequences...

A sequence of budgets (think bounds on the exploration length of paths) for successive subproblems at the same level of recursion is *universal* if it contains as a subsequence a sequence of budgets that is guaranteed to uncover the minimum-cost path.

- Clearly the sequence $2^{2^i}, 2^{2^i}, \ldots, 2^{2^i}$ of length 2^i is universal.
- However, we can do better...
  Consider instead the sequence σ_{2^i} defined inductively by

$$\sigma_s = \begin{cases} \langle 1 \rangle & \text{if } s = 0, \text{ and} \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & \text{otherwise,} \end{cases}$$

(where $\diamond$ signifies concatenation of sequences) .

Universal budget sequences...

A sequence of budgets (think bounds on the exploration length of paths) for successive subproblems at the same level of recursion is *universal* if it contains as a subsequence a sequence of budgets that is guaranteed to uncover the minimum-cost path.

- Clearly the sequence $2^{2i}, 2^{2i}, \ldots, 2^{2i}$ of length 2^i is universal.
- However, we can do better...

  Consider instead the sequence σ_{2i} defined inductively by

$$\sigma_s = \begin{cases} \langle 1 \rangle & \text{if } s = 0 \text{, and} \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & \text{otherwise,} \end{cases}$$

  (where $\diamond$ signifies concatenation of sequences) .

Universal budget sequences...

A sequence of budgets (think bounds on the exploration length of paths) for successive subproblems at the same level of recursion is *universal* if it contains as a subsequence a sequence of budgets that is guaranteed to uncover the minimum-cost path.

- Clearly the sequence $2^{2^i}, 2^{2^i}, \ldots, 2^{2^i}$ of length 2^i is universal.
- However, we can do better...
  Consider instead the sequence σ_{2^i} defined inductively by

$$\sigma_s = \begin{cases} \langle 1 \rangle & \text{if } s = 0 \text{, and} \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & \text{otherwise,} \end{cases}$$

(where $\diamond$ signifies concatenation of sequences) .

Properties of this *ruler* sequence

- the sequence σ_s is computable in $O(2^s)$-time and $\tilde{O}(1)$-space;
- the sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i, for all $i \in [s]$, and nothing else;
- (universality) for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\sum_{i \in [x]} d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ of σ_s such that $d_j \leq c_{i_j}$ holds for all $j \in [x]$

Properties of this *ruler* sequence

- the sequence σ_s is computable in $O(2^s)$-time and $\tilde{O}(1)$-space;
- the sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i, for all $i \in [s]$, and nothing else;
- (universality) for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\sum_{i\in[x]} d_i \le 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ of σ_s such that $d_j \le c_{i_j}$ holds for all $j \in [x]$

Properties of this *ruler* sequence

- the sequence σ_s is computable in $O(2^s)$-time and $\tilde{O}(1)$-space;
- the sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i, for all $i \in [s]$, and nothing else;
- (universality) for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\sum_{i \in [x]} d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ of σ_s such that $d_j \leq c_{i_j}$ holds for all $j \in [x]$

Properties of this *ruler* sequence

- the sequence σ_s is computable in $O(2^s)$-time and $\tilde{O}(1)$-space;
- the sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i, for all $i \in [s]$, and nothing else;
- (universality) for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\sum_{i \in [x]} d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ of σ_s such that $d_j \leq c_{i_j}$ holds for all $j \in [x]$

Proof of universality

(By induction on s)
Suppose that $\sum_{i\in[x]} d_i \le 2^s$. Choose the smallest m such that $\sum_{i\in[m]} d_i > \frac{1}{2}\sum_{i\in[x]} d_i$. Then,
(i) by induction, both $\langle d_1, \ldots, d_{m-1}\rangle$ and $\langle d_{m+1}, \ldots, d_x\rangle$ are dominated by subsequences of σ_{s-1}, and
(ii) $d_m \le 2^s$.
Hence $\langle d_1, \ldots, d_x\rangle$ is dominated by $\sigma_s = \sigma_{s-1} \diamond \langle 2^s\rangle \diamond \sigma_{s-1}$.

Using budgeted recursion, guided by this universal; sequence...

Theorem

For any instance of the min-weight path problem on an $2^h \times 2^h$ grid the procedure determines the min-weight path in $O(2^{9h})$ time and $\tilde{O}(2^h)$ space.

Using budgeted recursion, guided by this universal; sequence...

Theorem

For any instance of the min-weight path problem on an $2^h \times 2^h$ grid the procedure determines the min-weight path in $O(2^{9h})$ time and $\tilde{O}(2^h)$ space.

Proof sketch...

- correctness follows directly from universality property
- space complexity is clear
- The cost at the m-th level of recursion, with budget 2^s, $\mathrm{Cost}(m, 2^s)$, satisfies $\mathrm{Cost}(m, 2^s) \leq c \cdot 2^h T(2h - m, s)$, where

$$T(r, s) \; = \; \begin{cases} 2^s & \text{if } r = 0, \\ 1 + 2\sum_{0 \leq j \leq s} 2^j T(r-1, s-j) & \text{if } r > 0. \end{cases}$$

Proof sketch...

- correctness follows directly from universality property
- space complexity is clear
- The cost at the m-th level of recursion, with budget 2^s, $\mathrm{Cost}(m, 2^s)$, satisfies $\mathrm{Cost}(m, 2^s) \leq c \cdot 2^h T(2h - m, s)$, where

$$T(r,s) \;=\; \begin{cases} 2^s & \text{if } r = 0, \\ 1 + 2\sum_{0 \leq j \leq s} 2^j T(r-1, s-j) & \text{if } r > 0. \end{cases}$$

Proof sketch...

- correctness follows directly from universality property
- space complexity is clear
- The cost at the m-th level of recursion, with budget 2^s, $\mathrm{Cost}(m, 2^s)$, satisfies $\mathrm{Cost}(m, 2^s) \leq c \cdot 2^h T(2h - m, s)$, where

$$T(r, s) \;=\; \begin{cases} 2^s & \text{if } r = 0, \\ 1 + 2\sum_{0 \leq j \leq s} 2^j T(r-1, s-j) & \text{if } r > 0. \end{cases}$$

Proof sketch...

- correctness follows directly from universality property
- space complexity is clear
- The cost at the m-th level of recursion, with budget 2^s, $\mathrm{Cost}(m, 2^s)$, satisfies $\mathrm{Cost}(m, 2^s) \leq c \cdot 2^h T(2h - m, s)$, where

$$T(r,s) \;=\; \begin{cases} 2^s & \text{if } r = 0, \\ 1 + 2\sum_{0 \leq j \leq s} 2^j T(r-1, s-j) & \text{if } r > 0. \end{cases}$$

Proof sketch...

It is straightforward to confirm that

$$T(r,s) \;=\; \begin{cases} 2^{r+1}-1 & \text{if } r>0 \text{ and } s=0,\\ 2T(r,s-1)+2T(r-1,s)-1 & \text{if } r>0 \text{ and } s>0.\end{cases}$$

Thus, $T(r,s) \le 2^{r+s+1}\binom{r+s}{s}$.
It follows that $\mathrm{Cost}(m,2^s) \le c\cdot 2^h 2^{2h-m+s+1}\binom{2h-m+s}{s}$.
In particular, $\mathrm{Cost}(0,2^{2h})$, the cost of our procedure is $O(2^{5h}\binom{4h}{2h})$ or $O(2^{9h})$.

Proof sketch...

It is straightforward to confirm that

$$T(r,s) \;=\; \begin{cases} 2^{r+1}-1 & \text{if } r>0 \text{ and } s=0, \\ 2T(r,s-1)+2T(r-1,s)-1 & \text{if } r>0 \text{ and } s>0. \end{cases}$$

Thus, $T(r,s) \leq 2^{r+s+1}\binom{r+s}{s}$.
It follows that $\mathrm{Cost}(m,2^s) \leq c \cdot 2^h 2^{2h-m+s+1}\binom{2h-m+s}{s}$.
In particular, $\mathrm{Cost}(0,2^{2h})$, the cost of our procedure is $O(2^{5h}\binom{4h}{2h})$ or $O(2^{9h})$.

Combining the two approaches...

- Of course, it makes sense to stop the recursion when the subproblem size falls below $\sqrt{n}$.
- In fact, it pays to stop even earlier and switch to the Asano-Doerr method.
- The optimal switch point depends on the desired time-space tradeoff.

Combining the two approaches...

- Of course, it makes sense to stop the recursion when the subproblem size falls below $\sqrt{n}$.
- In fact, it pays to stop even earlier and switch to the Asano-Doerr method.
- The optimal switch point depends on the desired time-space tradeoff.

Combining the two approaches...

- Of course, it makes sense to stop the recursion when the subproblem size falls below $\sqrt{n}$.
- In fact, it pays to stop even earlier and switch to the Asano-Doerr method.
- The optimal switch point depends on the desired time-space tradeoff.

Combining the two approaches...

- Of course, it makes sense to stop the recursion when the subproblem size falls below $\sqrt{n}$.
- In fact, it pays to stop even earlier and switch to the Asano-Doerr method.
- The optimal switch point depends on the desired time-space tradeoff.

What about *reporting* the minimum-weight path?...

- Straightforward to maintain predecessor pointer for the target vertex t, and repeat (at a multiplicative cost proportional to the optimal path *length*);
- Alternatively, we can maintain minimum path values from s and to t at *all* vertices of the top level separator. Then solve a sequence of lower-level subproblems recursively. The (time) cost is dominated by the top-level problem.
  Recall the same idea (due to D. Hershberg) was used in the edit-distance problem...

What about *reporting* the minimum-weight path?...

- Straightforward to maintain predecessor pointer for the target vertex $\mathtt{t}$, and repeat (at a multiplicative cost proportional to the optimal path *length*);
- Alternatively, we can maintain minimum path values from $\mathtt{s}$ and to $\mathtt{t}$ at *all* vertices of the top level separator. Then solve a sequence of lower-level subproblems recursively. The (time) cost is dominated by the top-level problem.
  Recall the same idea (due to D. Hershberg) was used in the edit-distance problem...

What about *reporting* the minimum-weight path?...

- Straightforward to maintain predecessor pointer for the target vertex t, and repeat (at a multiplicative cost proportional to the optimal path *length*);
- Alternatively, we can maintain minimum path values from s and to t at *all* vertices of the top level separator. Then solve a sequence of lower-level subproblems recursively. The (time) cost is dominated by the top-level problem.

  Recall the same idea (due to D. Hershberg) was used in the edit-distance problem...

What about *reporting* the minimum-weight path?...

- Straightforward to maintain predecessor pointer for the target vertex `t`, and repeat (at a multiplicative cost proportional to the optimal path *length*);
- Alternatively, we can maintain minimum path values from `s` and to `t` at *all* vertices of the top level separator. Then solve a sequence of lower-level subproblems recursively. The (time) cost is dominated by the top-level problem.
  Recall the same idea (due to D. Hershberg) was used in the edit-distance problem...

Outline

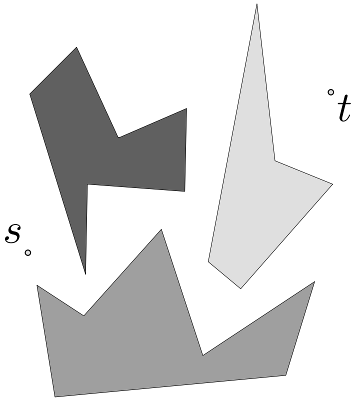
Introduction
- algorithms for shortest (min-weight) paths
- memory-constrained algorithms

Min-weight paths in grid graphs – Asano&Doerr(2011)
- overview of basic algorithm
- applying a good idea recursively

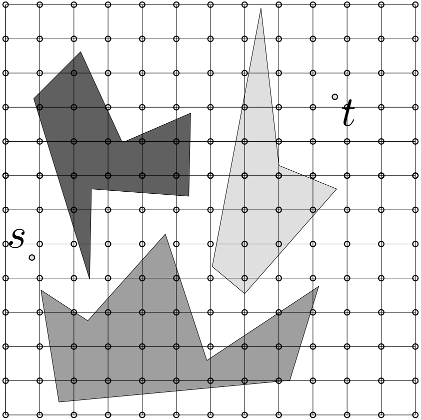
Min-weight paths in grid graphs – Refinements & Extensions
- a different recursive formulation
- budgeted recursion – exploiting a universal sequence
- combining the ideas

Beyond grid graphs...
- min-weight paths in implicit graphs
- min-weight paths in general planar graphs

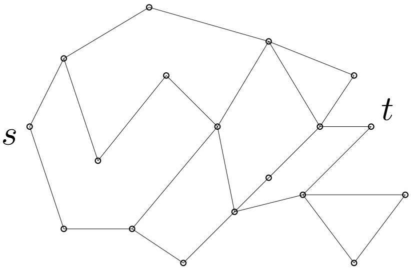
An arrangement of weighted regions with source and target...



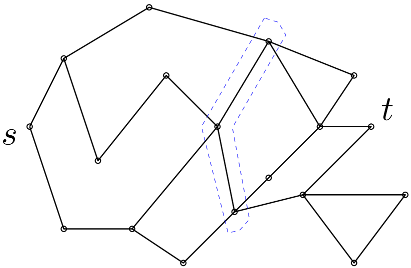
...and an overlaid grid



A planar graph



A planar graph...with a small separator



In joint work with Asano, Nakagawa and Wanatabe [MFCS 2014], this work is extended to arbitrary planar directed graphs.

Basic ideas for planar graphs...

- Use a space-efficient algorithm for constructing separators [Imai et al.]
- Maintain separators explicitly and (separated) components *implicitly* (using a representative point.
- Reconstruct triangulated components on-demand, using Reingold's log-space undirected reachability algorithm

That's it.....

And they all lived happily ever after.....

THE END

And they all lived happily ever after.....

THE END