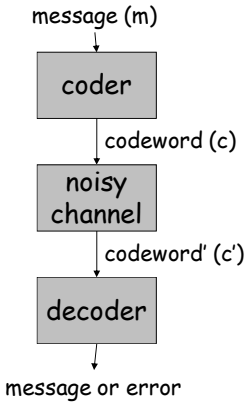
15-853:Algorithms in the Real World

Error Correcting Codes II
- -Reed-Solomon Codes
- -Concatenated Codes
- -Overview of some topics in coding
  - -Low Density Parity Check Codes (aka Expander Codes)
  - -Network Coding
  - -Compressive Sensing
  - -List Decoding

Block Codes



Each message and codeword is of fixed size

Σ = codeword alphabet

k = |m| **n** = |c| **q** = $|\Sigma|$

$C \subseteq \Sigma^n$ (codewords)

$\Delta(x,y)$ = number of positions s.t. $x_i \neq y_i$

d = $\min\{\Delta(x,y) : x,y \in C, x \neq y\}$

s = $\max\{\Delta(c,c')\}$ that the code can correct

Code described as: $(n,k,d)_q$

Linear Codes

If Σ is a field, then Σ^n is a vector space

Definition: *C is a linear code if it is a linear subspace of Σ^n of dimension k.*

This means that there is a set of k independent vectors $v_i \in \Sigma^n$ $(1 \leq i \leq k)$ that span the subspace.

i.e. every codeword can be written as:

$c = a_1 v_1 + a_2 v_2 + \ldots + a_k v_k$ where $a_i \in \Sigma$

The sum of two codewords is a codeword.

Minimum distance = weight of least-weight codeword

Generator and Parity Check Matrices

Generator Matrix:

A k x n matrix **G** such that: $C = \{ xG \mid x \in \Sigma^k \}$

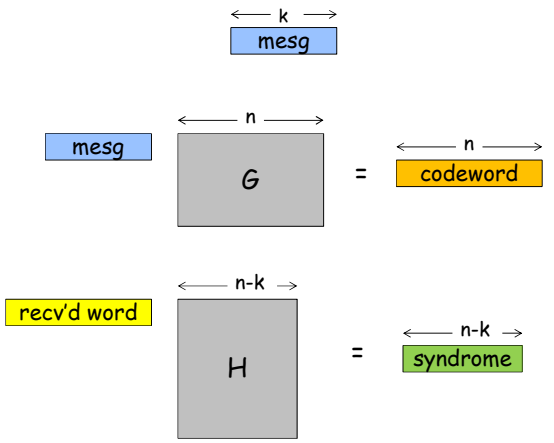
Made from stacking the spanning vectors

Parity Check Matrix:

An (n – k) x n matrix **H** such that: $C = \{y \in \Sigma^n \mid Hy^T = 0\}$

(Codewords are the null space of H.)

These **always exist for linear codes**



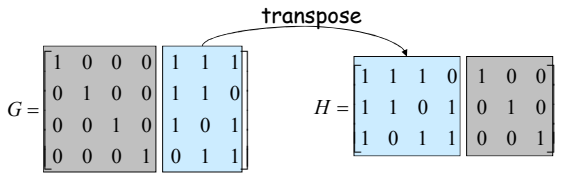
if syndrome = 0, received word = codeword

else have to use syndrome to get back codeword

Relationship of G and H

For linear codes, if G is in standard form $[I_k \; A]$
then $H = [-A^T \; I_{n-k}]$

Example of (7,4,3) Hamming code:



Two Codes

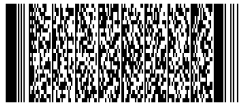
Hamming codes are binary $(2^r-1-1, 2^r-1-r, 3)$ codes.
Basically $(n, n - \log n, 3)$

Hadamard codes are binary $(2^r-1, r, 2^{r-1})$.
Basically $(n, \log n, n/2)$

The first has great rate, small distance.
The second has poor rate, great distance.
Can we get $\Omega(n)$ rate, $\Omega(n)$ distance?
Yes.
One way is to use a random linear code.
Let's see some direct, intuitive ways.

Reed-Solomon Codes

Irving S. Reed and Gustave Solomon



PDF-417

QR code

Aztec code

DataMatrix code

All 2-dimensional Reed-Solomon bar codes

images: wikipedia

Reed-Solomon Codes in the Real World

$(204,188,17)_{256}$: ITU J.83(A)[2]
$(128,122,7)_{256}$: ITU J.83(B)
$(255,223,33)_{256}$: Common in Practice
- Note that they are all byte based (i.e., symbols are from $GF(2^8)$).

Decoding rate on 1.8GHz Pentium 4:
- (255,251) = 89Mbps
- (255,223) = 18Mbps

Dozens of companies sell hardware cores that operate 10x faster (or more)
- (204,188) = 320Mbps (Altera decoder)

Applications of Reed-Solomon Codes

- **Storage**: CDs, DVDs, "hard drives",
- **Wireless**: Cell phones, wireless links
- **Sateline and Space**: TV, Mars rover, Voyager,
- **Digital Television**: DVD, MPEG2 layover
- **High Speed Modems**: ADSL, DSL, ..

Good at handling burst errors.
Other codes are better for random errors.
- e.g., Gallager codes, Turbo codes

Viewing Messages as Polynomials

A (n, k, n-k+1) code:
Consider the polynomial of degree k-1

$$p(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

Message: $(a_{k-1}, \ldots, a_1, a_0)$
Codeword: $(p(1), p(2), \ldots, p(n))$

To keep the p(i) fixed size, we use $a_i \in GF(q^r)$
To make the i distinct, $n \le q^r$

For simplicity, imagine that $n = q^r$. So we have a $(n, k, n-k+1)_{\log n}$ code.
Alphebet size increasing with the codeword length.
A little awkward. (But can be fixed.)

Viewing Messages as Polynomials

A (n, k, n-k+1) code:

Consider the polynomial of degree k-1

$p(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$

Message: $(a_{k-1}, \ldots, a_1, a_0)$

Codeword: $(p(1), p(2), \ldots, p(n))$

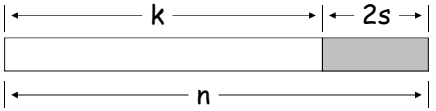
To keep the p(i) fixed size, we use $a_i \in GF(q^r)$

To make the i distinct, $n \le q^r$

Unisolvence Theorem: Any subset of size k of $(p(1), p(2), \ldots, p(n))$ is enough to (uniquely) reconstruct p(x) using polynomial interpolation, e.g., Lagrange interpolation formula.

Polynomial-Based Code

A (n, k, 2s +1) code:



Can **detect** 2s errors

Can **correct** s errors

Generally can correct α erasures and β errors if $\alpha + 2\beta \le 2s$

Correcting Errors

Correcting s errors:

1. Find k + s symbols that agree on a polynomial p(x). These must exist since originally k + 2s symbols agreed and only s are in error
2. There are no k + s symbols that agree on the wrong polynomial p'(x)
   - Any subset of k symbols will define p'(x)
   - Since at most s out of the k+s symbols are in error, p'(x) = p(x)

A brute-force approach.

Better algos exist (maybe next lecture).

RS and "burst" errors

Let's compare to Hamming Codes (which are "optimal").

	code bits	check bits
RS $(255, 253, 3)_{256}$	2040	16
Hamming $(2^{11}-1, 2^{11}-11-1, 3)_2$	2047	11

They can both correct 1 error, but not 2 random errors.

- The Hamming code does this with fewer check bits

However, RS can fix 8 contiguous bit errors in one byte

- Much better than lower bound for 8 arbitrary errors

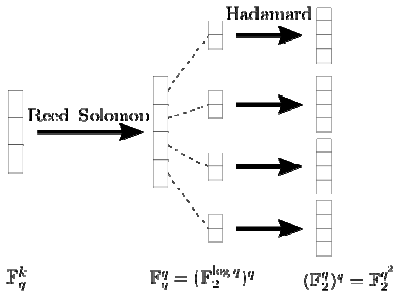
$$\log\left(1 + \binom{n}{1} + \cdots + \binom{n}{8}\right) > 8\log(n-7) \approx 88 \text{ check bits}$$

Concatenated Codes


David Forney

Take a RS code $(n,k,n-k+1)_{q=n}$ code.

Can encode each alphabet symbol using another code.



Wikipedia

Concatenated Codes

Take a RS code $(n,k,n-k+1)_{q=n}$ code.

Can encode each alphabet symbol of $k' = \log q = \log n$ bits using another code.

E.g., use $((k' + \log k'), k', 3)_2$-Hamming code. Now we can correct one error per alphabet symbol with little rate loss. (Good for sparse periodic errors.)

Or $(2^{k'}, k', 2^{k'-1})_2$ Hadamard code. (Say $k = n/2$.) Then get $(n^2, (n/2) \log n, n^2/4)_2$ code.
Much better than plain Hadamard code in rate, distance worse only by factor of 2.

Concatenated Codes

Take a RS code $(n,k,n-k+1)_{q=n}$ code.

Can encode each alphabet symbol of $k' = \log q = \log n$ bits using another code.

Or, since k' is $O(\log n)$, could choose a code that requires exhaustive search but is good.

Random linear codes give $((1+f(\delta))k', k', \delta k'))_2$ codes.

Composing with RS (with $k = n/2$), we get
$((1+f(\delta))n \log n, (n/2) \log n, \delta(n/2)\log n)_2$

Gives constant rate and constant distance! And poly-time encoding and decoding.

Error Correcting Codes Outline

Introduction
Linear codes
Reed Solomon Codes
Expander Based Codes
- Expander Graphs
- Low Density Parity Check (LDPC) codes
- Tornado Codes

Why Expander Based Codes?

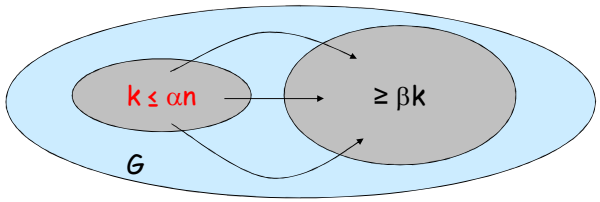
These are linear codes like RS & random linear codes

RS/random linear codes give good rates but are slow:

Code	Encoding	Decoding
Random Linear	$O(n^2)$	$O(n^3)$
RS	$O(n \log n)$	$O(n^2)$
LDPC	$O(n^2)$ or better	$O(n)$
Tornado	$O(n \log 1/\varepsilon)$	$O(n \log 1/\varepsilon)$

Assuming an $(n, (1-p)n, (1-\varepsilon)pn+1)_2$ tornado code

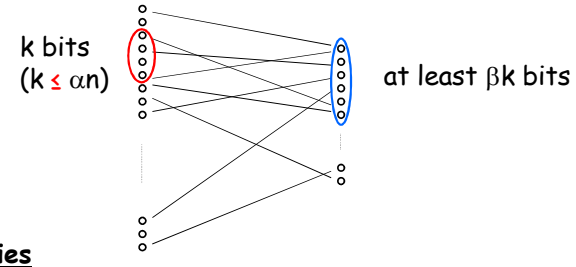
(α, β) Expander Graphs (non-bipartite)



Properties

- **Expansion:** every small subset ($k \leq \alpha n$) has many ($\geq \beta k$) neighbors
- **Low degree** – not technically part of the definition, but typically assumed

(α, β) Expander Graphs (bipartite)



Properties

- **Expansion:** every small subset ($k \leq \alpha n$) on left has many ($\geq \beta k$) neighbors on right
- **Low degree** – not technically part of the definition, but typically assumed

Expander Graphs

Useful properties:

- Every set of vertices has many neighbors
- Every balanced cut has many edges crossing it
- A random walk will quickly converge to the stationary distribution (rapid mixing)
- Expansion is related to the eigenvalues of the adjacency matrix

Expander Graphs: Applications

Pseudo-randomness: implement randomized algorithms with few random bits

Cryptography: strong one-way functions from weak ones.

Hashing: efficient n-wise independent hash functions

Random walks: quickly spreading probability as you walk through a graph

Error Correcting Codes: several constructions

Communication networks: fault tolerance, gossip-based protocols, peer-to-peer networks

d-regular graphs

An undirected graph is **d-regular** if every vertex has d neighbors.

A bipartite graph is **d-regular** if every vertex on the left has d neighbors on the right.

We consider only d-regular constructions.

Expander Graphs: Constructions

Important parameters:size (n), degree (d), expansion (β)

Randomized constructions

- A random d-regular graph is an expander with a high probability
- Construct by choosing d random perfect matchings
- Time consuming and cannot be stored compactly

Explicit constructions

- Cayley graphs, Ramanujan graphs etc
- Typical technique – start with a small expander, apply operations to increase its size

Expander Graphs: Constructions

Theorem: for every constant $0 < c < 1$, can construct bipartite graphs with

n nodes on left,
cn on right,
d-regular,

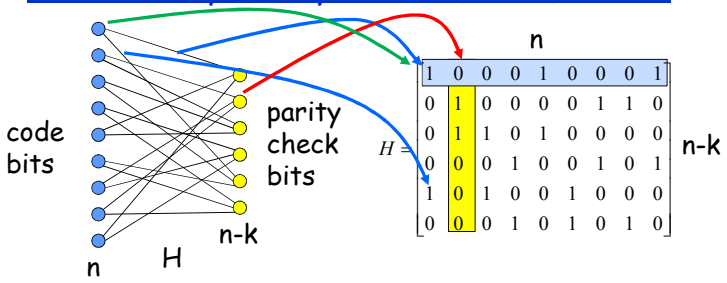
that are $(\alpha, 3d/4)$ expanders, for constants α and d that are functions of c alone.

"Any set containing at most alpha fraction of the left has (3d/4) times as many neighbors on the right"

Error Correcting Codes Outline

Introduction
Linear codes
Read Solomon Codes
Expander Based Codes
- Expander Graphs
- Low Density Parity Check (LDPC) codes
- Tornado Codes

Low Density Parity Check (LDPC) Codes



Each row is a vertex on the right and each column is a vertex on the left.
A codeword on the left is valid if each right "parity check" vertex has parity 0.
The graph has O(n) edges (**low density**)

Applications in the real world

10Gbase-T (IEEE 802.3an, 2006)
- Standard for 10 Gbits/sec over copper wire

WiMax (IEEE 802.16e, 2006)
- Standard for medium-distance wireless. Approx 10Mbits/sec over 10 Kilometers.

NASA
- Proposed for all their space data systems

History

Invented by Gallager in 1963 (his PhD thesis)

Generalized by Tanner in 1981 (instead of using parity and binary codes, use other codes for "check" nodes).

Mostly forgotten by community at large until the mid 90s when revisted by Spielman, MacKay and others.

Distance of LDPC codes

Consider a d-regular LPDC with (α,3d/4) expansion.

Theorem: Distance of code is greater than αn.

Proof. (by contradiction)

Linear code; distance= min weight of non-0 codeword.

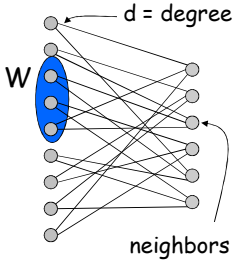
Assume a codeword with weight w ≤ αn.

Let W be the set of 1 bits in codeword

It has >3dw/4 neighbors on the right

Average # of 1s per such neighbor is < 4/3.

So at least one neighbor sees a single 1-bit. Parity check would fail!



Correcting Errors in LPDC codes

We say a vertex is *unsatisfied* if parity ≠ 0

Algorithm:

While there are unsatisfied check bits

1. Find a bit on the left for which more than d/2 neighbors are unsatisfied
2. Flip that bit

Converges since every step reduces unsatisfied nodes by at least 1.

Runs in linear time.

Why must there be a node with more than d/2 unsatisfied neighbors?

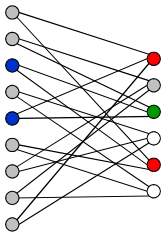
Coverges to closest codeword

Theorem: Assume (α,3d/4) expansion. If # of error bits is less than αn/4 then simple decoding algorithm converges to closest codeword.

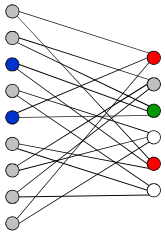
Proof: let:

- u_i = # of unsatisfied check bits on step i
- r_i = # corrupt code bits on step i
- s_i = # satisfied check bits with corrupt neighbors on step i

We know that u_i decrements on each step, but what about r_i?



Proof continued:



- u_i = unsatisfied
- r_i = corrupt
- s_i = satisfied with corrupt neighbors

$u_i + s_i > \frac{3}{4} dr_i$ (by expansion)

$2s_i + u_i \le dr_i$ (by counting edges)

$\frac{1}{2} dr_i \le u_i$ (by substitution)

$u_i < u_0$ (steps decrease u) $u_0 \le dr_0$ (by counting edges)

Therefore: $r_i < 2r_0$ i.e. number of corrupt bits cannot more than double

If we start with at most αn/4 corrupt bits we will never get αn/2 corrupt bits --- but the distance is αn

More on decoding LDPC

Simple algorithm is only guaranteed to fix half as many errors as could be fixed but in practice can do better.

Fixing (d-1)/2 errors is NP hard

Soft "decoding" as originally specified by Gallager is based on belief propagation---determine probability of each code bit being 1 and 0 and propagate probs. back and forth to check bits.

Encoding LPDC

Encoding can be done by generating G from H and using matrix multiply.

What is the problem with this?

Various more efficient methods have been studied

Error Correcting Codes Outline

Introduction

Linear codes

Read Solomon Codes

Expander Based Codes

- Expander Graphs
- Low Density Parity Check (LDPC) codes
- Tornado Codes

The loss model

Random Erasure Model:

- Each bit is lost independently with some probability μ
- We know the positions of the lost bits

For a **rate** of (1-p) can correct $(1-\varepsilon)p$ fraction of the errors.

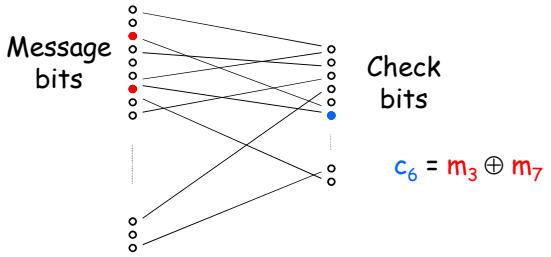
Seems to imply a

$(n, (1-p)n, (1-\varepsilon)pn+1)_2$

code, but not quite because of random errors assumption.

We will assume p = .5.

Error Correction can be done with some more effort

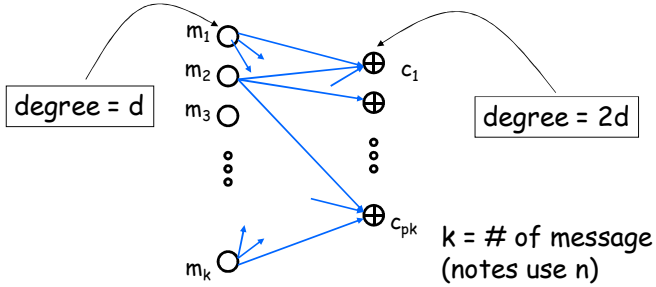


Similar to LDPC codes but check bits are not required to equal zero (i.e the graph does not represent H).

Tornado codes

Will use d-regular bipartite graphs with n nodes on the left and pn on the right (notes assume p = .5)

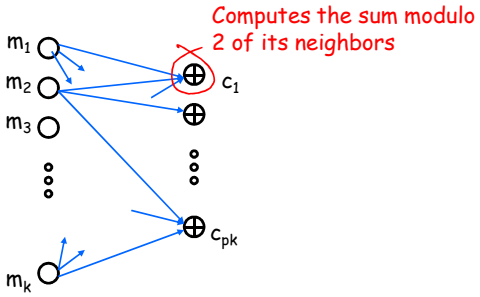
Will need $\beta > d/2$ expansion.



k = # of message bits (notes use n)

Tornado codes: Encoding

Why is it linear time?

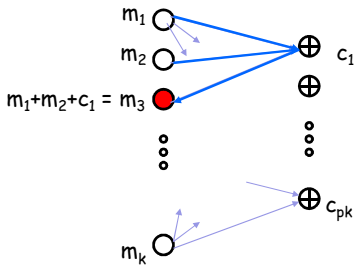


Tornado codes: Decoding

Assume that all the check bits are intact

Find a check bit such that only one of its neighbors is erased (an *unshared neighbor*)

Fix the erased code, and repeat.



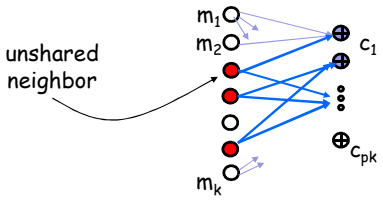
Tornado codes: Decoding

Need to ensure that we can always find such a check bit

"Unshared neighbors" property

Consider the set of corrupted message bit and their neighbors. Suppose this set is small.

=> at least one message bit has an unshared neighbor.



Tornado codes: Decoding

Can we always find unshared neighbors?

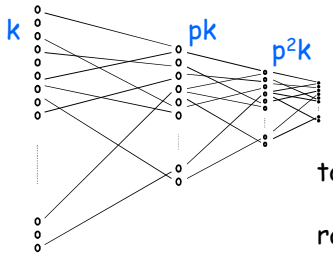
Expander graphs give us this property if $\beta > d/2$ (similar argument to one above)

Also, [Luby et al] show that if we construct the graph from a specific kind of degree sequence, then we can always find unshared neighbors.

What if check bits are lost?

Cascading

- Use another bipartite graph to construct another level of check bits for the check bits
- Final level is encoded using RS or some other code



$p^lk \leq \sqrt{n}$

total bits $n \leq k(1 + p + p^2 + \ldots)$
$= k/(1-p)$
rate = $k/n = (1-p)$

Cascading

Encoding time

- for the first k stages : $|E| = d \times |V| = O(k)$
- for the last stage: $\sqrt{k} \times \sqrt{k} = O(k)$

Decoding time

- start from the last stage and move left
- again proportional to $|E|$
- also proportional to d, which must be at least $1/\varepsilon$ to make the decoding work

Can fix $kp(1-\varepsilon)$ random erasures

Some extra slides

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Squaring

- G^2 contains edge (u,w) if G contains edges (u,v) and (v,w) for some node v
- $A' = A^2 - 1/d\ I$
- $\lambda' = \lambda^2 - 1/d$
- $d' \leq d^2 - d$

Size	≡
Degree	↑
Expansion	↑

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Tensor Product (Kronecker product)

- $G = A \times B$ nodes are (a,b) $\forall\ a \in A$ and $b \in B$
- edge between (a,b) and (a',b') if A contains (a,a') and B contains (b,b')
- $n' = n_1 n_2$
- $\lambda' = \max(\lambda_1, \lambda_2)$
- $d' = d_1 d_2$

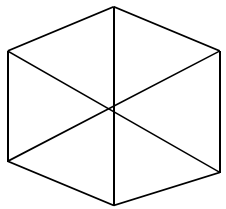
Size	↑
Degree	↑
Expansion	↓

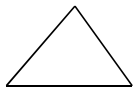
Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Zig-Zag product

- "Multiply" a big graph with a small graph





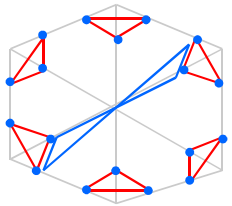
$n_2 = d_1$
$d_2 = \sqrt{d_1}$

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Zig-Zag product

- "Multiply" a big graph with a small graph



Size	↑
Degree	↓
Expansion	↓ (slightly)

Combination: square and zig-zag

For a graph with size n, degree d, and eigenvalue λ, define $G = (n, d, \lambda)$. We would like to increase n while holding d and λ the same.

Squaring and zig-zag have the following effects:

$(n, d, \lambda)^2 = (n, d^2, \lambda^2)$ $\equiv \uparrow\uparrow$

(n_1, d_1, λ_1) zz $(d_1, d_2, \lambda_2) = (n_1 d_1, d_2^2, \lambda_1 + \lambda_2 + \lambda_2^2)$ $\uparrow\downarrow\downarrow$

Now given a graph $H = (d^4, d, 1/5)$ and $G_1 = (d^4, d^2, 2/5)$

- $G_i = G_{i-1}^2$ zz H (square, zig-zag)

Giving: $G_i = (n_i, d^2, 2/5)$ where $n_i = d^{4i}$ (as desired)