# Tumor Heterogeneity

Alex Munoz, Emily Simons, Margaux Hujoel

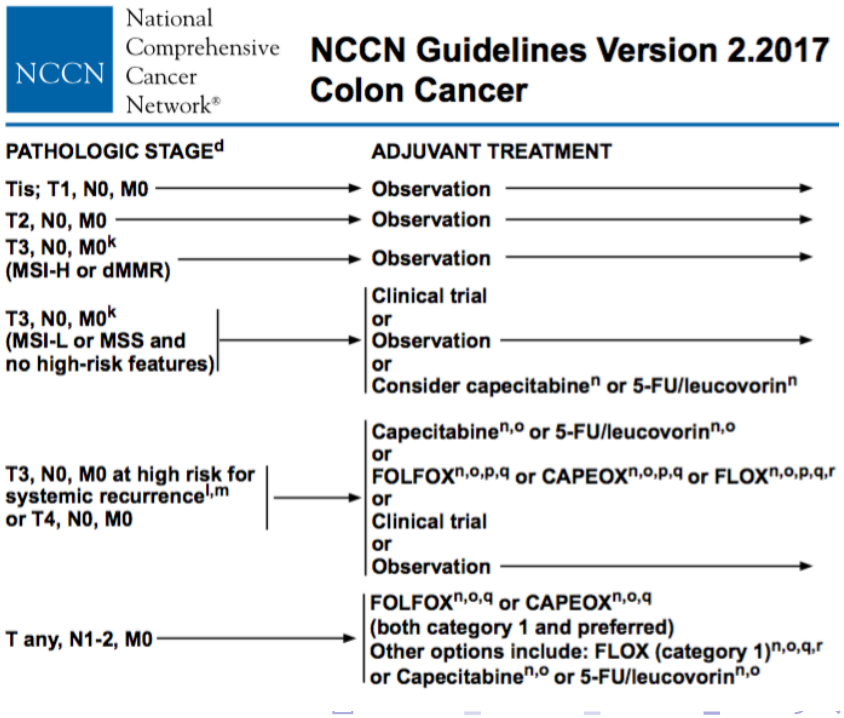
May 3, 2017

# Overview

# Clinical basis

- Treatment and surveillance hinges on pathologic stage + a basket of prognostic factors
- What is pathologic stage?
- Example: stages and management algorithm for colon cancer
  - Prognostic factors for colon cancer: e.g. perineural or lymphovascular invasion, CEA level, MSS, BRAF, VEGF, CNA, CpG Methylator Phenotype

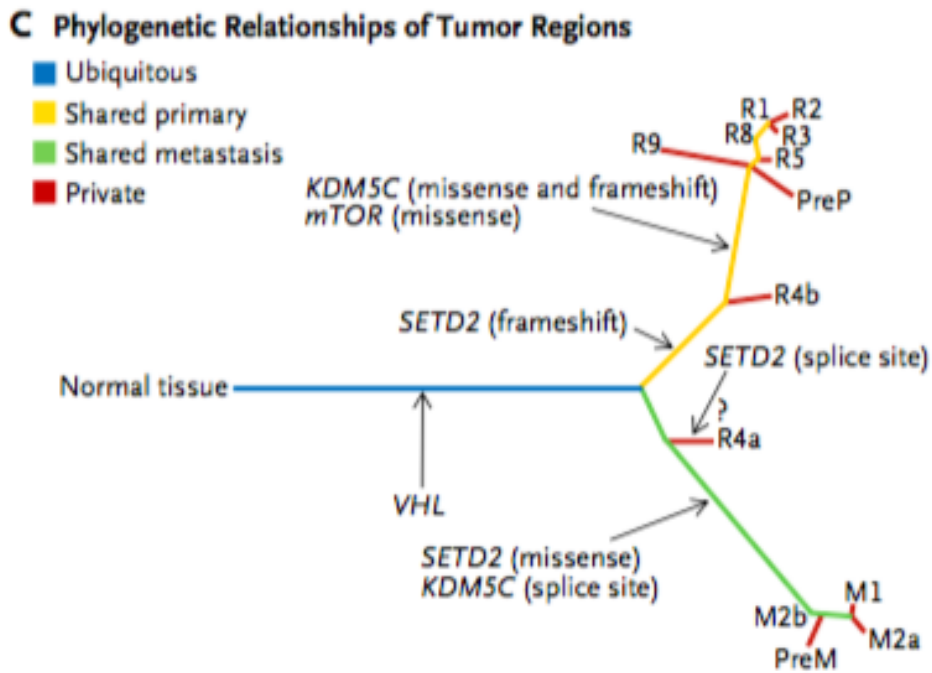
| Stage | TNM |
|---|---|
| I | T1 or T2, N0, M0 |
| IIA | T3, N0, M0 |
| IIB | T4a, N0, M0 |
| IIC | T4b, N0, M0 |
| IIIA | -T1 or T2, N1, M0<br>-T1, N2a, M0 |
| IIIB | -T3 or T4a, N1, M0<br>-T2 or T3, N2a, M0<br>-T1 or T2 N2b, M0 |
| IIIC | -T4a, N2a, M0<br>-T3 or T4a, N2b, M0<br>-T4b, N1 or N2, M0 |
| IVA | Any T, Any N, M1a |
| IVB | Any T, Any N, M1b |

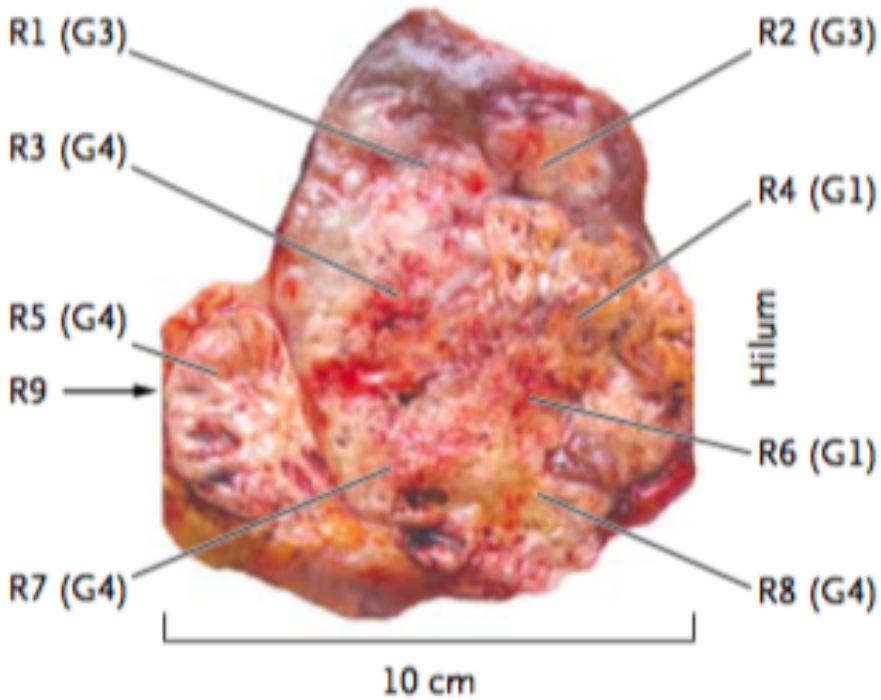


# Intratumor genetic heterogeneity as molecular anatomy

Metastatic renal cell carcinoma:

- Region of tumor biopsied– subclones with different driver mutations
  - False to assume a single sample is representative of tumor genetics
  - Grade (also prognostic) not reflective of genomics
- Metastasis can be from minor subclone- would want to characterize subclones to maximal resolution to predict resistance




Sources: Gerlinger et al. NEJM. 2012; Samuel et al. Clinical Chemistry. 2013

# Intratumor genetic heterogeneity as a prognostic factor

- Chronic lymphocytic leukemia:
  - Subclonal sSNVs increase with treatment and number of treatments
  - Subclonal evolution enriched by treatment associated with
    - emergence of driver mutations
    - decreased progression free survival

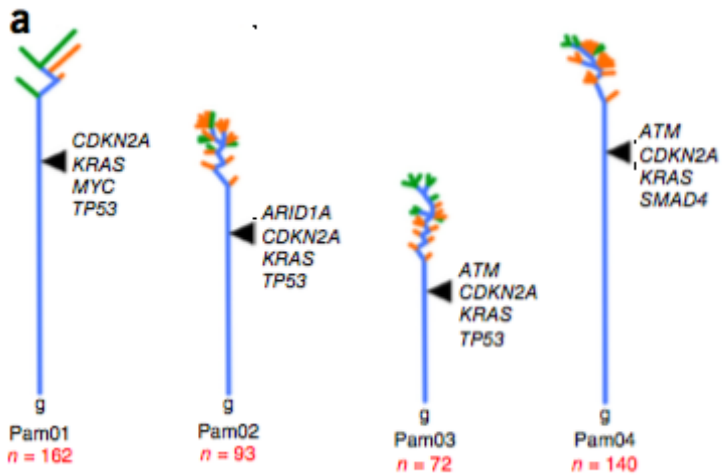
- Head and neck cancer:
  - MATH: width of the distribution of MAF across mutated loci within an individual tumor
  - Increase in MATH (intratumor heterogeneity) – shorter survival time
    (even when controlling for risk factors: age, HPV status, tumor grade, nodal involvement, TP53 mutation)

- Similar findings of heterogeneity in esophageal, glioblastoma, breast cancer

Sources: Landau et al. Cell. 2013; Mroz et al. PLOS Medicine. 2015

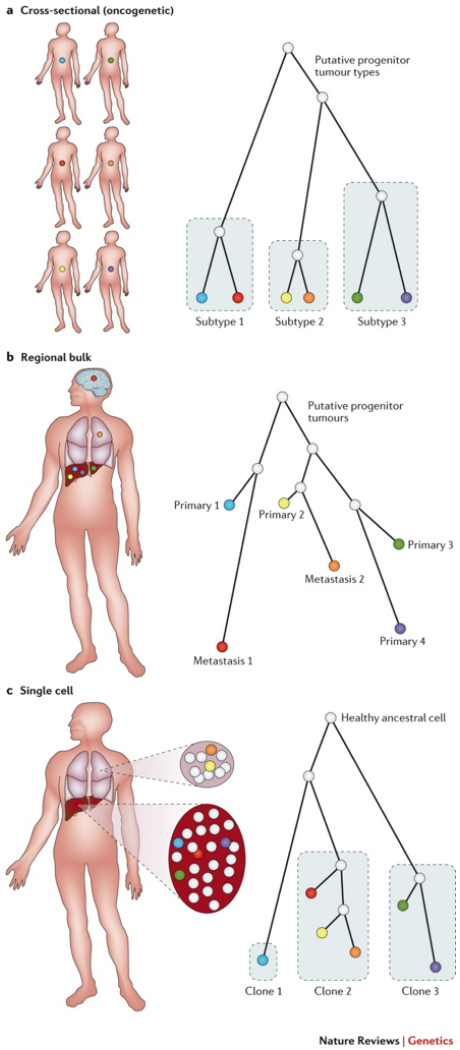
# Clinical basis caveats

- Not a new finding: FISH demonstrated intratumor structural genetic diversity in 1990s in many cancer types
  - What is new: proof that drivers can be present in low allelic fraction via minor subclones
  - may not initially confer a selective advantage
  - but may become drivers when selective pressure is altered by chemotherapy (or microenvironment in case of metastasis)
- Not a universal finding: only trunkal driver mutations were found in 4 patients with 26 metastatic pancreatic cancer lesions




Sources: Moore et al. Nature Genetics. 2017; Samuel et al. Clinical Chemistry. 2013

# 3 Main Phylogeny Problems




- Cross-sectional tumor phylogenetics
- Regional bulk tumor phylogenetics
- Single-cell tumor phylogenetics

Schwartz, Russell and Alejandro A. Schaffer. "The evolution of tumour phylogenetics: principles and practice" Nature Review Genetics, 2017: (18) 213-229.

# Setting

- Input: $C$, mutational CCF.
- Output: $Z$, $P$: the tree and clonal CCF
- Simplifications: ignoring CNVs and only looking at SNVs
- Big picture: $C = ZP$, given number of clones $c$ we wish to find clusters (columns of $Z$) and optimize such that $P$ is valid (all columns in $P$ must sum to 1).

# Setting: the Matrices

$$s = \#samples; \qquad n = \#loci;$$

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1s} \\ c_{21} & c_{22} & \cdots & c_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{ns} \end{pmatrix}_{loci \times samples}$$

$$Z = \begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1c} \\ z_{21} & z_{22} & \cdots & z_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nc} \end{pmatrix}_{loci \times clones} \qquad P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ p_{c1} & p_{c2} & \cdots & p_{cs} \end{pmatrix}_{clones \times samples}$$

# Algorithm

Fix the number of clones at $c$.

$$C_i = \begin{pmatrix} c_{1i} \\ c_{2i} \\ \vdots \\ c_{ni} \end{pmatrix} = p_{1,i} \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + p_{2,i} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \cdots + p_{2^n,i} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Optimize the above for $p_{j,i}$ with the constraint that only $c$ $p_{j,i}$ values can be non-zero (only have $c$ clones) across the $C_i$, $i = 1, \ldots, s$, where the non-zero $p_{j,i}$ can be (and will be) different across the $C_i$, but for the $2^n - c$ $p_{j,i} = 0$, these must be the same $j$ across the $i$. Also $\sum_j p_{j,i} = 1 \forall i$.

Let $s = \#samples = 2, n = \#loci = 3$. Then we have:

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix}_{loci*samples}.$$

Then we find we are optimizing the below:

$$\begin{pmatrix} c_{11} \\ c_{21} \\ c_{31} \end{pmatrix} = p_1 \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + p_2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + p_3 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + p_4 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + p_5 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + p_6 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + p_7 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + p_8 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix};$$

$$\begin{pmatrix} c_{12} \\ c_{22} \\ c_{32} \end{pmatrix} = p_1^* \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + p_2^* \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + p_3^* \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + p_4^* \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + p_5^* \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + p_6^* \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + p_7^* \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + p_8^* \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

under the constraint that $\sum p_i^* = \sum p_i = 1$ and, given $c$ clones, only $c$ $p_i, p_i^*$ values are allowed to be non-zero.

Assume we have 4 clones. Then $2^n - c = 2^3 - 4 = 4$ of the $p$s will be 0. Assume our optimization found
$p_1 = p_1^* = p_2 = p_2^* = p_3 = p_3^* = p_6 = p_6^* = 0$. Then we find:

$$\begin{pmatrix} c_{11} \\ c_{21} \\ c_{31} \end{pmatrix} = p_4 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + p_5 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + p_7 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + p_8 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} p_5 + p_8 \\ p_5 + p_7 + p_8 \\ p_4 + p_7 + p_8 \end{pmatrix};$$

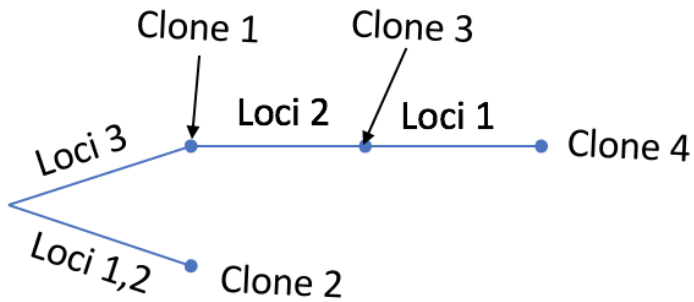
$$\begin{pmatrix} c_{12} \\ c_{22} \\ c_{32} \end{pmatrix} = p_4^* \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + p_5^* \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + p_7^* \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + p_8^* \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} p_5^* + p_8^* \\ p_5^* + p_7^* + p_8^* \\ p_4^* + p_7^* + p_8^* \end{pmatrix}.$$

Or, succinctly,

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_4 & p_4^* \\ p_5 & p_5^* \\ p_7 & p_7^* \\ p_8 & p_8^* \end{pmatrix} = ZP.$$

Zare et al. noted that if we let clone number exceed number of samples it is not guaranteed that the conditions for perfect phylogeny are met. We let clone number (3) exceed sample number (2) in this simple case.

| | Clone 1 | Clone 2 | Clone 3 | Clone 4 |
|---|---|---|---|---|
| Loci 1 | 0 | 1 | 0 | 1 |
| Loci 2 | 0 | 1 | 1 | 1 |
| Loci 3 | 1 | 0 | 1 | 1 |



- We enumerate all possible $Z$, then determine the best $P$ for this given $Z$ (linearly transform the $P$ so it adheres to the condition that sum of every column is 1) and see which $Z, P$ pair results in the "closest" approximation of $C$
- The computationally intensive step is to enumerate all the possible $Z$

# Perfect Phylogeny

- We wish to have constraints on the enumeration of $Z$ such that we only test $Z$ which correspond to trees possible under the perfect phylogeny assumption
- We developed 2 methods - one of which is computationally more efficient (relies on the assumption of perfect phylogeny).

# Method 1

- generate all possible combinations of columns of zeros and ones where no columns are repeated
- doesn't incorporate the assumption of perfect phylogeny
- $O(\binom{2^n}{c})$ where n is the number of loci and c is the number of clones.

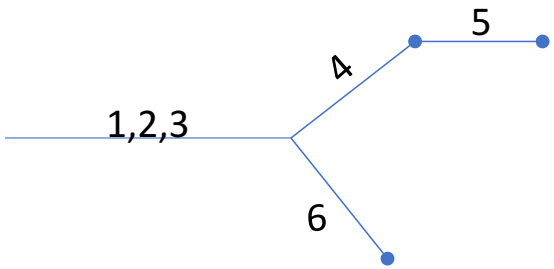
# Method 2

- More computationally tractable if we take advantage of the perfect phylogeny assumption
- how to generate only $Z$ from which the perfect phylogeny assumption is held
- Consider a new matrix $M$: all of the same number must be in the same column; if there are two numbers that cannot be in the same column, $Z$ doesn't fit the perfect phylogeny assumption

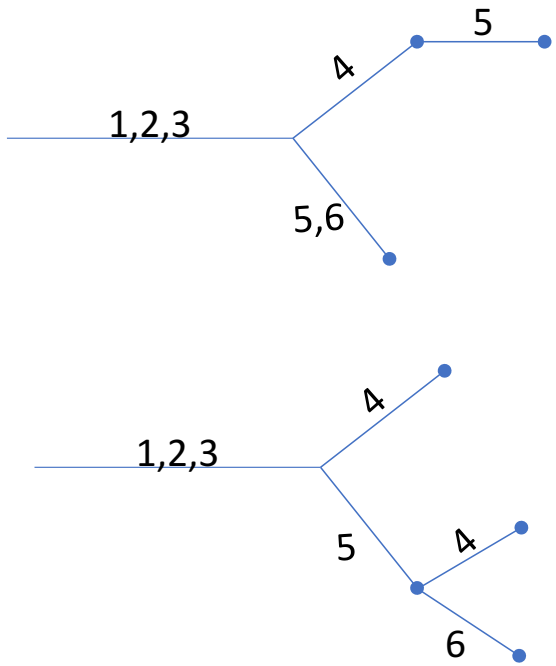
$$Z = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 \\ 1 & 2 & 3 & 6 & 0 & 0 \end{pmatrix} = M.$$

# Method 2

$$Z = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 \\ 1 & 2 & 3 & 0 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 \\ 1 & 2 & 3 & 5 & 6 & 0 \end{pmatrix} = M.$$




A.




B.

# Method 2

```
def __possible_M_cols_helper(self, clone_num):
    ''' Generate possible M matrix columns '''
    if clone_num == 0:
        return np.array([])
    start = np.array([1 for i in range(clone_num)])
    possibles = [start]
    for i in range(clone_num-1):
        new_possibles = []
        for p in possibles:
            curr_adder = np.array([0 for j in range(i+1)] + \
                                  [1 for j in range(clone_num-(i+1))])
            new_possibles.append(p + curr_adder)
        possibles += new_possibles
    possibles += [np.append(j,0) for j in self.__possible_M_cols_helper(clone_num-1)]
    return possibles
```

# Method 2

```
def __stochastic_Ms_helper(self, empty_clones, prev_max):
    ''' select a stochastic M '''
    if prev_max == self.N or empty_clones == self.C_num:
        return np.zeros((self.C_num, 1))
    curr_poss_cols = self.__possible_M_cols(self.C_num-empty_clones, prev_max)
    looping = True
    while looping:
        random_col_truncated = curr_poss_cols[np.random.choice(len(curr_poss_cols))]
        random_col = np.concatenate((random_col_truncated, np.zeros(empty_clones)))
        if max(random_col) <= self.N:
            looping = False
    zero_inds = np.where(random_col==0)[0]
    if len(zero_inds > 0):
        new_empties = self.C_num - min(zero_inds)
    else:
        new_empties = 0
    return np.column_stack((random_col, \
                            self.__stochastic_Ms_helper(empty_clones=new_empties, prev_max=max(random_col))))
```

# Method 2

```
def __validate_M(self, M_org):
    ''' Reorder an M for common mutations on left, then check for perfect phylogency (mutations arise once)'''
    M_new = M_org
    curr_swap_col = 0
    for row_count in range(M_org.shape[0]): #first swap columns for most common mutations on left
        for col_num in range(M_org.shape[1]):
            curr_col = M_new[:,col_num]
            if row_count == 0 and len(set(curr_col)) == 1 and curr_col[0] != 0:
                M_new[:,[curr_swap_col, col_num]] = M_new[:,[col_num, curr_swap_col]]
                curr_swap_col += 1
            elif row_count > 0 and set(curr_col[-1*row_count:]) == {0} \
                               and len(set(curr_col[:-1*row_count])) == 1 and curr_col[0] != 0:
                M_new[:,[curr_swap_col, col_num]] = M_new[:,[col_num, curr_swap_col]]
                curr_swap_col += 1
    curr_swap = 1 #order feature enumeration by mutation count
    for search_req in np.arange(2,M_org.shape[0]+1,1)[::-1]:
        for col_num in range(M_org.shape[1]):
            for p in Counter([a for a in M_new[:,col_num] if a!=0]).most_common():
                if p[0] in np.arange(1,curr_swap,1):
                    pass
                elif p[1] == search_req and p[0] == curr_swap:
                    curr_swap += 1
                elif p[1] == search_req and p[0] != curr_swap:
                    for col_num_adj in range(M_org.shape[1]):
                        olds = np.where(M_new[:,col_num_adj]==curr_swap)[0]
                        if olds.size != 0:
                            M_new[:,col_num_adj] = [p[0] if a==curr_swap else a for a in M_new[:,col_num_adj]]
                            break
                    M_new[:,col_num] = [curr_swap if a==p[0] else a for a in M_new[:,col_num]]
                    curr_swap += 1
    for row_num in range(M_org.shape[0]): #then sort by feature number & shift zeros to the right of a given row
        curr_row = M_new[row_num, :]
        new_row = np.concatenate((np.sort(curr_row[curr_row!=0]), curr_row[curr_row==0]))
        M_new[row_num, :] = new_row
    possible_vals = set(np.arange(np.amax(M_tst_new))+1) #now validate each number appears in only one column
    for col_num in range(M_org.shape[1]):
        curr_col = set(M_new[:,col_num])-{0}
        if curr_col.issubset(possible_vals):
            possible_vals = possible_vals - curr_col
        else:
            return (False, M_new)
    return (True, M_new)
```

# Method 2

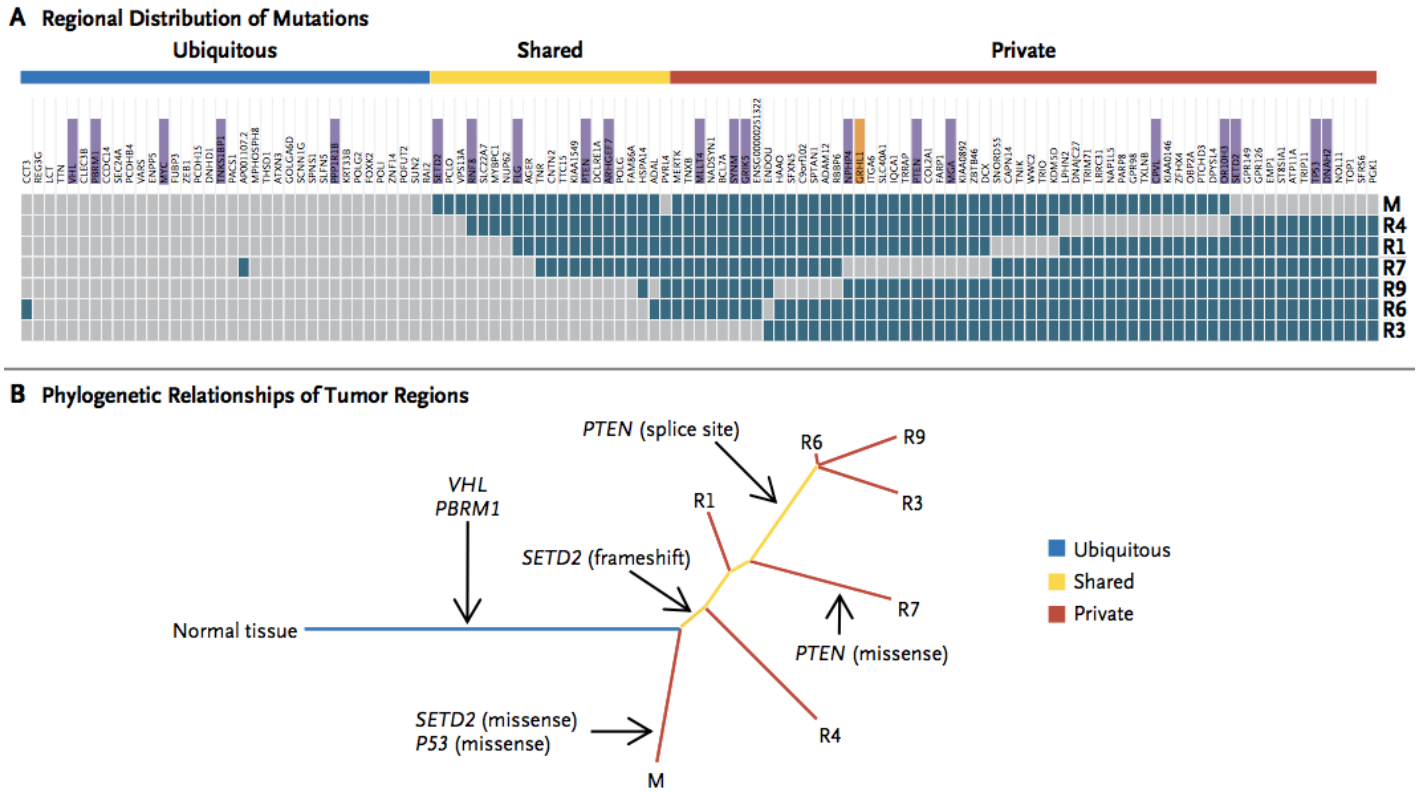
- We generate $M$ that satisfy the condition that all the same numbers have to be in the same column
- We then can from these $M$ determine all the $Z$ that satisfy perfect phylogeny
- $O(2^{n*c})$ where $n$ is the number of loci and $c$ is the number of clones

# Method Comparison

- When $c << n$ (or the number of clones is much less than the number of mutated loci, which is typically the case) method 2 vastly outperforms method 1.
- Complete enumeration is computationally intensive so we generate a random collection of $M$ matrices (10,000) - convert to $Z$ - see which $Z, p$ pair is the "best" out of these 10,000 - repeat 10 times and see if the $Z$ converges (i.e. whether for each random collection we get the same "best" $Z$).

# Results

An example found in literature: Gerlinger et al. produced a phylogenetic tree given mutations found in different tumors within a patient




6 clinically relevant mutations: 3 distinct primary samples and 1 distinct metastatic sample

# Implementation

```
class Z_P_Optimizer(object):
    '''
    Generates a C from standard uniform, then determines corresponding Z and P
    - N is number of loci,
    - M is number of samples,
    - C_num is number of clustered clones
    '''

    def __init__(self, loci_num=5, samples_num=5, clones_num=3):
        self.N = loci_num
        self.M = samples_num
        self.C_num = clones_num
```

# Implementation

```
def __generate_Zs_Ps_error(self, Zs_count=1000, C_in=None):
    ''' Generate corresponding P matrices from Z and C
        Returns Zs, valid Ps, and sum of residuals (squared Euclidean 2-norm) '''
    if C_in is None:
        C_matrix = self.__draw_C_matrix()
    else:
        C_matrix = C_in
    possible_Zs = self.__stochastic_Zs(Zs_count=Zs_count)
    possible_Ps = []
    errors = []
    for possible_Z in possible_Zs:
        curr_P_solved = np.linalg.lstsq(possible_Z, C_matrix)
        curr_P_scaled = self.__scale_P(curr_P_solved[0])
        curr_resids = (C_matrix - np.dot(possible_Z, curr_P_scaled))**2
        curr_error = np.sum(np.sum(curr_resids))
        possible_Ps.append(curr_P_scaled)
        errors.append(curr_error)
    return (possible_Zs, possible_Ps, errors)
```

# Implementation

In [303]:
```
Z = np.array([[1,0,0],[0,1,1],[0,1,0],[0,0,1]])
opt = Z_P_Optimizer(loci_num=4, samples_num=3, clones_num=3)
P = opt.draw_P_matrix()
C = np.dot(Z, P)
print(Z)
```

```
[[1 0 0]
 [0 1 1]
 [0 1 0]
 [0 0 1]]
```

In [304]:
```
opt_z, opt_p, opt_err = opt.optimal_Z_P(C_in=C, Zs_count=5000)
opt_z
```

Out[304]:
```
array([[ 1.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.],
       [ 0.,  1.,  0.]])
```

# Implementation

In [448]:
```
Gerlinger_data = pd.read_csv('Data/Gerlinger.csv')
loci = [0,1,2,4,6]
Gerlinger_Z = Gerlinger_data.ix[loci,[3,6,7,9]]
opt = Z_P_Optimizer(loci_num=len(loci), samples_num=3, clones_num=4)
P = opt.draw_P_matrix()
C = np.dot(Gerlinger_Z, P)
print('Loci genotyped:', list(Gerlinger_data.ix[loci,1]))
Gerlinger_Z
```

```
Loci genotyped: ['PBRM1', 'VHL', 'PTENsplice', 'PTENmis', 'TP53']
```

Out[448]:

| | R3 | R7 | R1 | M |
|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 1 |
| **1** | 1 | 1 | 1 | 1 |
| **2** | 1 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 |
| **6** | 0 | 0 | 0 | 1 |

In [449]:
```
opt_z, opt_p, opt_err = opt.optimal_Z_P(C_in=C, Zs_count=5000)
opt_z
```

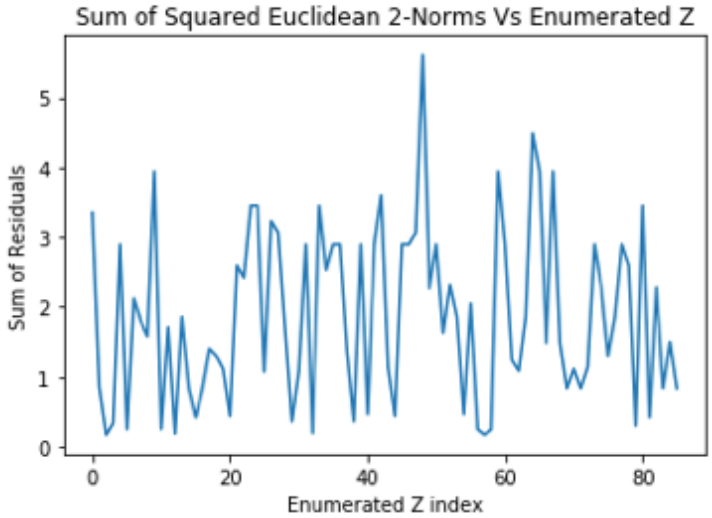
Out[449]:
```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.]])
```

# Implementation

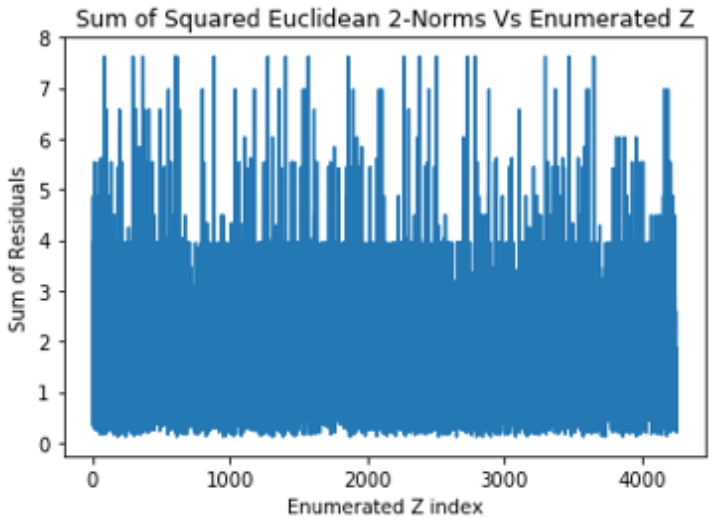
```
def plot_errors(self, C_in=None, Zs_count=1000):
    ''' Plot error graph (x-axis is enumerated Z, y-axis is sum of residuals) '''
    Zs, Ps, errs = self.__generate_Zs_Ps_error(C_in=C_in, Zs_count=Zs_count)
    plt.figure()
    plt.plot(errs)
    plt.title('Sum of Squared Euclidean 2-Norms Vs Enumerated Z')
    plt.xlabel('Enumerated Z index')
    plt.ylabel('Sum of Residuals')
    plt.show()
```

# Implementation

```
In [74]: opt.plot_errors(C_in=C, Zs_count=100)
```




```
In [75]: opt.plot_errors(C_in=C, Zs_count=5000)
```



# Implementation

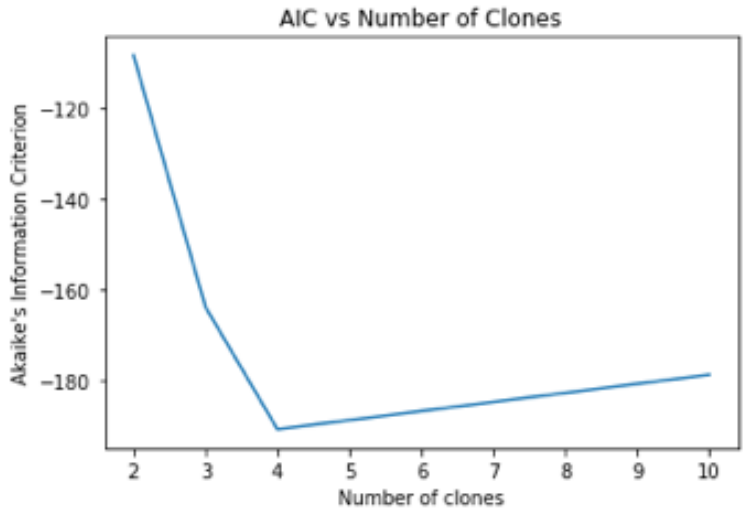
In [155]:
```
errors = []
for i in range(10000):
    z_opt, p_opt, err_opt = opt.optimal_Z_P(Zs_count=1)
    if len(errors) > 0:
        errors.append(min([err_opt, errors[-1]]))
    else:
        errors.append(err_opt)
plt.figure()
plt.plot(errors)
plt.xlabel('Number of Z matrices generated')
plt.ylabel('Minimum Euclidean 2-Norm Error')
plt.title('Error vs Number of Zs')
plt.show()
```

# Implementation

In [320]:
```
errors = []
clone_range = np.arange(2,11,1)
for clone_count in clone_range:
    opt = Z_P_Optimizer(loci_num=5, samples_num=3, clones_num=clone_count)
    z_opt, p_opt, err_opt = opt.optimal_Z_P(C_in=C, Zs_count=5000)
    errors.append(err_opt)
    print(clone_count, end=", ")
print("done.")
```

```
2, 3, 4, 5, 6, 7, 8, 9, 10, done.
```

In [322]:
```
plt.figure()
aic = 100*np.log(np.array(errors))+2*clone_range
plt.plot(clone_range, aic)
plt.ylabel('Akaike\'s Information Criterion')
plt.xlabel('Number of clones')
plt.title('AIC vs Number of Clones')
plt.show()
```



# Implementation

In [462]:
```
return_dict = opt.optimal_Z_P(C_in=C, Zs_count=5000, return_dict=True)
set_Zs, set_idx = np.unique([np.array_str(a) for a in return_dict['Zs']], return_index=True)
top_errs = np.argsort(np.array(return_dict['errors'])[set_idx])[:5]
for Z_curr in set_Zs[top_errs]:
    print(Z_curr)
    print('\n')
```

```
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 1.  0.  0.  0.]]


[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]]


[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 0.  1.  1.  0.]
 [ 0.  0.  0.  1.]
 [ 1.  0.  0.  0.]]


[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]


[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 0.  0.  1.  1.]
 [ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]]
```

# Discussion

To optimize tumor treatment:

- target clonally dominant truncal somatic events OR
- adopt multiple targeted therapies in a combinatorial approach

Methods like ours based on multiple samples would require further refinement

To optimize research beyond limits of computational resolution for multiple sample methods:

- increase library complexity and read depth
- single-cell approaches

# Future Directions

- Further improving upon the efficiency of the optimization

Thank you!