Formation of Nitrogenase NifDK Tetramers in the Mitochondria of *Saccharomyces cerevisiae*

**AUTHORS**
Stefan Burén†, Eric M. Young‡, Elizabeth A. Sweeny‡, Gema Lopez-Torrejón‡, Marcel Veldhuizen‡, Christopher A. Voigt*‡, Luis M. Rubio*†

†Centro de Biotecnología y Genómica de Plantas, Universidad Politécnica de Madrid (UPM) - Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA), Campus Montegancedo UPM, 28223-Pozuelo de Alarcón, Madrid, España

‡Synthetic Biology Center, Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

* Corresponding authors. E-mail: lm.rubio@upm.es; E-mail: cavoigt@gmail.com
# SUPPORTING INFORMATION

## SUPPLEMENTARY TABLES AND FIGURES

| Table/Figure S1          | Description                                           |
|-------------------------|-------------------------------------------------------|
| Supplementary Table S1  | Sequences                                             |
| Supplementary Table S2  | Part assignment                                       |
| Supplementary Table S3  | Assembly                                              |
| Supplementary Table S4  | Quality control and results                           |
| Supplementary Table S5  | Mitochondrial signal efficiency                       |
| Supplementary Table S6  | Promoter efficiency                                   |
| Supplementary Table S7  | Regression analysis                                   |
| Supplementary Figure S1 | Analysis of promoter efficiency in generating detectable NifEN protein |
| Supplementary Figure S2 | Migration of NifD and NifK polypeptides               |
| Supplementary Figure S3 | Mitochondria targeting of Nif proteins                |
| Supplementary Figure S4 | Expression of Nif proteins in DSN14                  |
| Supplementary Figure S5 | NifDK tetramer formation in DSN14                     |
| Supplementary Figure S6 | Detailed TypeII S hierarchy for cluster assembly     |
| Supplementary Figure S7 | Level 1 destination vector design                     |
| Supplementary Figure S8 | Spacer design                                         |
Supporting Figure S1. Analysis of promoter efficiency in generating detectable NifEN protein. Solid bars indicate % positive clones as detected by Western blotting using antibodies targeting NifE (a) and NifN (b) (left axis). Striped bars indicate observed fluorescence when GFP was expressed by corresponding transcription unit, with the geometric mean and standard deviation of biological replicates plotted in arbitrary units for each promoter-terminator combination (right axis, see also Figure 1). Data is ordered according to GFP fluorescence levels. Note that NifE and NifN could not be sufficiently separated by SDS-PAGE, and Western blotting result therefore reflects total NifEN protein. bdl, below detection limit.
Supporting Figure S2. Migration of NifD and NifK polypeptides. (a) 50 ng samples of purified *A. vinelandii* apo-NifDK protein was loaded on 7% SDS-PAGE gel. Following separation and transfer, membrane was cut in two and probed with NifD and NifK specific antibodies, and finally developed side by side. Dotted line indicate cut of the membrane. (b) Immunoblot of purified *A. vinelandii* NifDK protein, together with total protein extracts from selected yeast clones, using antibodies recognizing NifDK. NifD (red arrow) and NifK (green arrow) proteins are indicated. Yeast strains expressing NifD or NifDK (DSN14, DSN24 and DOE1), also show presence of faster migrating polypeptide (yellow arrows). (c) Immunoblot analysis of DSN20 developed with antibodies targeting NifD or NifK specifically, or NifDK, as indicated to the right of each panel. NifD (red arrows) and NifK (green arrows) proteins are indicated, together with the faster migrating NifD polypeptide (yellow arrows). (d) Immunoblot analysis of protein extracts from yeast transformed with plasmid encoding mitochondria (SU9) targeted versions of His-tagged NifD and non-tagged NifK. Cells were grown in SD media supplemented with glucose (non-induced, Glc) or galactose (induced, Gal). Antibodies recognizing his-tag detect full-length NifD (red arrows), but not the faster migrating NifD polypeptide (yellow arrow).
Supporting Figure S3. Mitochondria targeting of Nif proteins. Immunoblot analysis of total extracts (TE) and mitochondria isolations (Mito) from three yeast strains with Nif proteins expressed and targeted using SU9, SOD2 or INDH mitochondria leader sequences. Antibodies recognizing cytoplasmic (tubulin) and mitochondria (HSP60) control proteins were used as controls. s.e. and l.e., short and long exposure.
**Supporting Figure S4.** Expression of Nif proteins in DSN14. Immunoblot analysis to compare migration of Nif proteins from *A. vinelandii* and yeast strain DSN14.
Supporting Figure S5. NifDK tetramer formation in DSN14. Co-purification of NifK with His-tagged NifD. NifK (green arrows) co-migrate with His-tagged NifD (yellow arrows), indicating complex formation of NifK and His-NifD polypeptides.
Supporting Figure S6. Detailed TypeIIS hierarchy for cluster assembly. (a) Collection of Level 0 transcriptional parts. Promoter and terminator parts in Level 0 part maintenance vectors. (b) Construct Level 0 nif genes. Assembly of nif cluster genes into Level 0 part maintenance vectors. Scars are standardized, except at unique tag-gene junctions to maintain amino acid sequence. (c) Construct Level 1 transcription units. Assembly of promoters, terminators, and genes into a positional Level 1 vector with spacers. See Supporting Figure S7 for more details on the positional Level 1 destination vectors. See Supporting Figure S8 for more details on the spacer design. (d) Construct Level 2 subclusters. Assembly of transcription units into subclusters. (e) Cluster integration. Subclusters were constructed simultaneously with transformation. Subclusters were linearized with BsaI and mixed with homology fragments to target the subclusters to the genome. Every assembly step is tabulated in Supporting Table S3.
Supporting Figure S7. Level 1 destination vector design. Each Level 1 vector contains the A scar (GTGC) and the D scar (CCTC) for receiving transcription units upon BsaI digestion. Successful ligation of the transcription unit parts into the vector will eliminate the constitutive bacterial RFP expression cassette, causing loss of a red colony color. On either side of the A and D scars, there is a half spacer. See Spacer Design for more details. Outside of the spacers, positional scars are located. These are designated E, F, G, H, I, J, K, L, M, N, and O. When assembling subclusters, these positional scars will dictate the order in which the transcription units assemble. This step will also match the spacers as intended (S1 with S2, S3 with S4, etc.).
Supporting Figure S8. Spacer design. (a) Minimal terminator elements used to design spacers. One efficiency element, four different positioning elements, and two polyadenylation elements used. (b) Spacer with forward and reverse terminator elements. These elements were randomly combined, with random DNA in between the elements, in the orientation shown here.
**SUPPLEMENTARY TEXT**

**Methods**

*Ordinary Least Squares Regression Script*

```
Title: OLS Polynomial Regression in Python with Output to Excel
Created on Wed Mar 18 16:10:53 2015

@author: Eric M. Young
@email: ericyoung7@gmail.com

Version: Python 3.5.2
```

```python
import pandas as pd
import statsmodels.stats.api as sms
from statsmodels.formula.api import ols
from openpyxl import load_workbook
from openpyxl import Workbook

regression_name = 'dsn'
regression_model = 'da'

filename = '%(r)s_%(o)s.xlsx' % {'r':regression_name, 'o': regression_model}

def create_workbook():
    wb = Workbook()
    # this is needed to rename the default first sheet
    ws = wb.active
    ws.title = 'equation'
    wb.save(filename)

def load_df():
    worksheet = regression_name
    df = pd.read_excel('nif_regression.xlsx', worksheet)
    return df

def ols_regression(df):
    # Specify C for categorical variables (i.e. T/F) in the formula
modelsD = {'fnifs': '''nifS_WB ~ nifS_GFP''',
    'fnife': '''nifE_WB ~ nifE_GFP''',
    'fnifn': '''nifN_WB ~ nifN_GFP''',
    'fnifd': '''nifD_WB ~ nifD_GFP''',
    'fnifk': '''nifK_WB ~ nifK_GFP''',
    'fnifh': '''nifH_WB ~ nifH_GFP''',
    'fnifm': '''nifM_WB ~ nifM_GFP''',
    'fnifb': '''nifB_WB ~ nifB_GFP''',
    'fnifu': '''nifU_WB ~ nifU_GFP''',
    'fa': '''SUM_WB ~ nifS_GFP + nifE_GFP + nifN_GFP + nifD_GFP + nifK_GFP + nifH_GFP + nifM_GFP + nifB_GFP
```
+ nifU_GFP''',
'dnifs' : '''nifs_WB ~ Strength''',
'dnife' : '''nifE_WB ~ Strength*(C(nifENB_indh) + C(nifENB_su9))'''
'dnifn' : '''nifN_WB ~ Strength*(C(nifENB_indh) + C(nifENB_su9))''',
'dnifd' : '''nifD_WB ~ Strength''',
'dnifik' : '''nifK WB ~ Strength''',
'dnifh' : '''nifH WB ~ Strength*(C(nifHM_indh) + C(nifHM_su9) +
C(nifHM_sod2))'''
'dnifm' : '''nifM_WB ~ Strength*(C(nifHM_indh) + C(nifHM_su9) +
C(nifHM_sod2))'''
'dnifb' : '''nifB WB ~ Strength*(C(nifENB_indh) + C(nifENB_su9))'''
'dnifu' : '''nifU_WB ~ Strength''',
'da' : '''SUM WB ~ Strength*(C(nifHM_indh) + C(nifHM_su9) +
C(nifENB_indh) + C(nifENB_su9))'''
regression=ols(modelsD[regression_model],data=df).fit()
return regression

def equation(res):
    ef = pd.DataFrame(res.params, columns=['coef'])
    ef['t'] = res.tvalues
    ef['p'] = res.pvalues
    ef['se'] = res.bse
    output_dataframe(ef, 'equation')

def fit_stats(res):
    ff = pd.DataFrame([1], columns=['R Squared'])
    ff['R Squared'] = res.rsquared
    ff['Adj R Squared'] = res.rsquared_adj
    ff['# Observations'] = res.nobs
    ff['F Value'] = res.fvalue
    ff['Prob(F Value)'] = res.f_pvalue
    ff['Log-Likelihood'] = res.llf
    ff['Jarque-Bera'] = sms.jarque_bera(res.resid)[0]
    ff['Prob(JB)'] = sms.jarque_bera(res.resid)[1]
    ff['Skewness'] = sms.jarque_bera(res.resid)[2]
    ff['Kurtosis'] = sms.jarque_bera(res.resid)[3]
    ff['Omnibus'] = sms.omni_normtest(res.resid)[0]
    ff['Prob(Omnibus)'] = sms.omni_normtest(res.resid)[1]
    ff['AIC'] = res.aic
    ff['BIC'] = res.bic
    output_dataframe(ff, 'fit')
    return [res.nobs, res.rsquared, res.fvalue, res.f_pvalue, res.llf, sk, ku,
    omni, pomni]

def output_to_regression_history(statsL):
    #Assemble data to be written to file
    new_row = [regression_name, regression_model]
for n, x in enumerate(statsL):
    new_row.append(statsL[n])

# Open history workbook
his_book = load_workbook('nif_regression_history.xlsx')
ws = his_book.get_sheet_by_name('reg_history')

ws.append(new_row)

his_book.save('nif_regression_history.xlsx')

def output_dataframe(df, sheetname):
    # Implement openpyxl workaround to prevent overwriting the whole file
    book = load_workbook(filename)
    writer = pd.ExcelWriter(filename, engine='openpyxl')
    writer.book = book
    writer.sheets = dict((ws.title, ws) for ws in book.worksheets)

    # Write new dataframe to Excel
    df.to_excel(writer, sheet_name=sheetname)
    writer.save()

    # Load data to be fitted from Excel
    df = load_df()

    results = ols_regression(df)

    # All of this is to output the regression to Excel in various forms
    create_workbook()
    fsL = fit_stats(results)
equation(results)
output_to_regression_history(fsL)