Navigating the Deployment and Downlink Tradespace for Earth Imaging Constellations

Sreeja Nag[1,2], Steven P. Hughes[1] and Jacqueline J. Le Moigne[1]

[1]NASA Goddard Space Flight Center

[2]Bay Area Environmental Research Institute

Tradespace Design Context

Distributed Spacecraft Missions

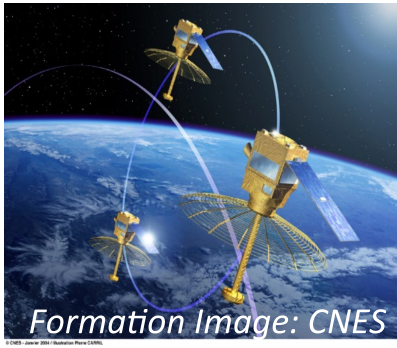
Performance: Improve sampling in spatial (synthetic apertures), temporal (constellations), spectral (fractionated S/C), angular (formations) dimensions

Cost: Need more inter-operability planning, autonomy, scheduling commands + data, ground station networks

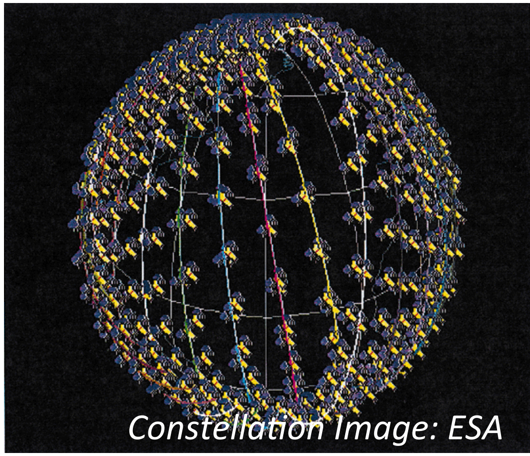
Ilities in Operations: Flexibility, Reconfigurability, Scalability, etc.

Better Design: Many conflicting variables and objectives thus better methods needed in Phase A+ - coupled models, machine learning, planning/scheduling methods, etc.

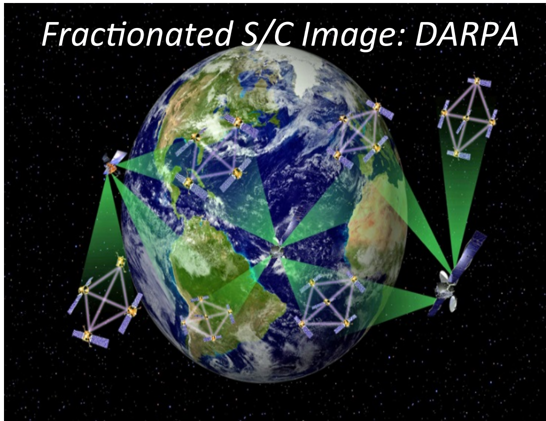
Tradespace exploration is required early in the design cycle NASA GSFC is building a software tool called Tradespace Analysis Tool for Constellations (TAT-C), to address some of the above questions.



Formation Image: CNES

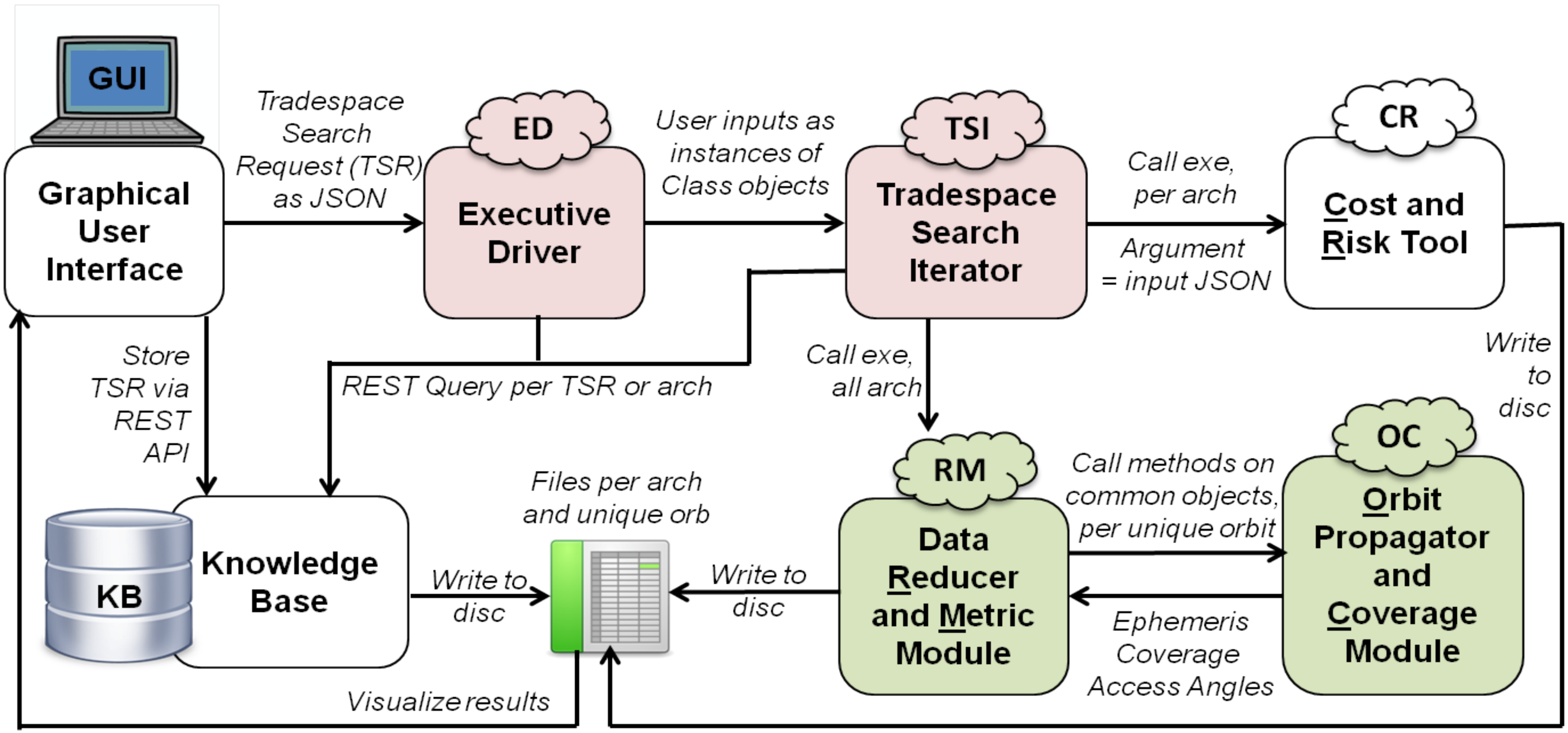


Constellation Image: ESA



Fractionated S/C Image: DARPA

Information Flow



- Pink: Python
- Green: C++
- White: Not covered in this paper

Previous: S. Nag, S.P. Hughes, J.J. Le Moigne, "Streamlining the Design Tradespace for Earth Imaging Constellations", AIAA Space Conference, Long Beach California, September 2016

Executive Driver

- Sets up global macros for the location of the user folder and exe folders
- Reads TSR json (and throws errors if incorrect)
- Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - Mission Specs – Overall concept parameters including launch and ground stations

```
class MissionConcepts:
    def __init__(self,startEpoch=time.mktime(datetime.utcn
        objectsOfInterest='Sun',groundStationOptions='NENa
                 launchOptions='Primary',organization='Aca
                     propulsion=0):
        self.startEpoch = startEpoch
        self.areaOfInterest = areaOfInterest
        self.objectsOfInterest = objectsOfInterest
        self.groundStationOptions = groundStationOptions
        self.launchOptions = launchOptions
        self.organization = organization
        self.performancePeriod = performancePeriod
        self.duration = duration
        self.propulsion = propulsion
```

  - Launch Vehicle – Primary/Secondary OR custom txt file w/ parameters as columns and LV as rows
  - Observatory Specs – Custom txt w/ parameters as columns (and unique satellites as rows)
  - Payload Specs – Custom txt w/ parameters as columns (and unique payload per sat as rows)
  - Satellite Orbits – Exact specs or ranges provided in the TSR
  - Output Bounds – Ranges provided in the TSR or not bounded
- Creates list for all available ground stations w/ lat, lon, rented and comm bands
- Gets Planet Labs ephemeris for ad-hoc, unmaintained constellation type

Executive Driver

- Sets up global macros for the location of the user folder and exe folders
- Reads TSR json (and throws errors if incorrect)
- Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - Launch Vehicle – Primary/Secondary OR custom txt file w/ parameters as columns and LV as rows

```
class LaunchVehicle:
    def __init__(self,name = "Pegasus",
        self.name = name
        self.dryMass = dryMass
        self.propMass = propMass
        self.Isp = Isp
        self.payMass = payMass
        self.maxRelight = maxRelight
        self.reliability = reliability
        self.cost = cost
        self.mtbl = mtbl # days
```

+ methods for reading TSR
+ methods for computing spread for precession constellations, provide fuel for maintenance, number of satellites per launch, etc.
+ parameters for cost

  - Observatory Specs – Custom txt w/ parameters as columns (and unique satellites as rows)
  - Payload Specs – Custom txt w/ parameters as columns (and unique payload per sat as rows)
  - Satellite Orbits – Exact specs or ranges provided in the TSR
  - Output Bounds – Ranges provided in the TSR or not bounded
- Creates list for all available ground stations w/ lat, lon, rented and comm bands
- Gets Planet Labs ephemeris for ad-hoc, unmaintained constellation type

Executive Driver

- Sets up global macros for the location of the user folder and exe folders
- Reads TSR json (and throws errors if incorrect)
- Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - Mission Specs – Overall concept parameters including launch and ground stations
  - Observatory Specs – Custom txt w/ parameters as columns (and unique satellites as rows)

```
class ObservatorySpecifications:
    def __init__(self,obsMass=200,obsPower=150,obsVolume=300,a
                 crossTrackSlewOfCenter=0,angularScanRate=0,an
        self.obsMass = obsMass
        self.obsPower = obsPower
        self.obsVolume = obsVolume
        self.alongTrackConeAngle = alongTrackConeAngle
        self.crossTrackConeAngle = crossTrackConeAngle
        self.alongTrackSlewOfCenter = alongTrackSlewOfCenter
        self.crossTrackSlewOfCenter = crossTrackSlewOfCenter
        self.angularScanRate = angularScanRate
        self.angularScanStartPhase = angularScanStartPhase
        self.communicationBand = communicationBand
```

+ methods for reading TSR
+ Mass for computing LV numbers needed
+ Communication bands for computing appropriate ground stations for downlink

  - Payload Specs – Custom txt w/ parameters as columns (and unique payload per sat as rows)
  - Satellite Orbits – Exact specs or ranges provided in the TSR
  - Output Bounds – Ranges provided in the TSR or not bounded
- Creates list for all available ground stations w/ lat, lon, rented and comm bands
- Gets Planet Labs ephemeris for ad-hoc, unmaintained constellation type

Executive Driver

- Sets up global macros for the location of the user folder and exe folders
- Reads TSR json (and throws errors if incorrect)
- Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - Mission Specs – Overall concept parameters including launch and ground stations
  - Observatory Specs – Custom txt w/ parameters as columns (and unique satellites as rows)
  - Payload Specs – Custom txt w/ parameters as columns (and unique payload per sat as rows)

```
class InstrumentSpecifications:
    def __init__(self,instruConops=0,instruConopsPartner=
                 specReso=[50,60,70],minRadiometricReso=1
                 overallDataRate=1.5,solarConditions='no_
                 isRect=0,fovAT=15):
        self.instruConops = instruConops
        self.instruConopsPartner = instruConopsPartner
        self.instruMass = instruMass
        self.instruPower = instruPower
        self.instruVolume = instruVolume
        self.specBands = specBands
        self.specReso = specReso
        self.minRadiometricReso = minRadiometricReso
        self.fovCT = fovCT
        self.instFieldOfView = instFieldOfView
        self.measurementTime = measurementTime
        self.overallDataRate = overallDataRate
        self.solarConditions = solarConditions
        self.sunglintPref = sunglintPref
        self.occultationAltitudes = occultationAltitudes
        self.isRect = isRect
        self.fovAT = fovAT
```

+ methods for reading TSR
+ bands for aperture size for cost
+ FOV or sensor shape for RMOC
+ solar conditions for RMOC angles
+ iFOV for RMOC

Executive Driver

- Sets up global macros for the location of the user folder and exe folders
- Reads TSR json (and throws errors if incorrect)
- Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - Mission Specs – Overall concept parameters including launch and ground stations
  - Observatory Specs – Custom txt w/ parameters as columns (and unique satellites as rows)
  - Payload Specs – Custom txt w/ parameters as columns (and unique payload per sat as rows)
  - Satellite Orbits – Exact specs or ranges provided in the TSR

```
class SatelliteOrbits:
    def __init__(self,existingSatelliteOptions='null',nu
                 specialOrbits='null',propagationFidelit
        self.existingSatelliteOptions = existingSatellit
        self.numberOfNewSats = numberOfNewSats
        self.altitudeRange = altitudeRange
        self.inclinationRange = inclinationRange
        self.specialOrbits = specialOrbits
        self.propagationFidelity = propagationFidelity
```

+ methods for reading TSR
+ Methods to create full range of orbits to pass to RMOC

  - Output Bounds – Ranges provided in the TSR or not bounded
- Creates list for all available ground stations w/ lat, lon, rented and comm bands
- Gets Planet Labs ephemeris for ad-hoc, unmaintained constellation type

Executive Driver

- ~~Creates classes, instantiates with TSR inputs or defaults to Landsat 8 values
  - ~~
  - Output Bounds – Ranges provided in the TSR or not bounded

```
class OutputBounds:
    def __init__(self,timeToCoverage='null',accessTim
                 alongOverlap='null',signalNoiseRatic
                 sunAzimuth='null',spatialResolution=
                 objAzimuth='null',objRange='null',nu
        self.timeToCoverage = timeToCoverage
        self.accessTime = accessTime
        self.downlinkLatency = downlinkLatency
        self.revisitTime = revisitTime
        self.crossOverlap = crossOverlap
        self.alongOverlap = alongOverlap
        self.signalNoiseRatio = signalNoiseRatio
        self.dlAccessPerDay = dlAccessPerDay
        self.obsZenith = obsZenith
        self.obsAzimuth = obsAzimuth
        self.sunZenith = sunZenith
        self.sunAzimuth = sunAzimuth
        self.spatialResolution = spatialResolution
        self.crossSwath = crossSwath
        self.alongSwath = alongSwath
        self.objZenith = objZenith
        self.objAzimuth = objAzimuth
        self.objRange = objRange
        self.numPassesPM = numPassesPM
```

+ methods for reading TSR
+ (Methods for filtering the variables to the TSR with the ability to re-introduce without a full rerun)

- Creates list for all available ground stations w/ lat, lon, rented and comm bands
- Gets Planet Labs ephemeris for ad-hoc, unmaintained constellation type

Tradespace Search Iterator

- Variables – Constellation type, number of sats, altitude, inclination, number of planes (uniform sats per plane), field of view or regard, launch vehicle (affects dispatch batches and cost), ground station number (affects performance only)
- Number of satellites filtered by output bounds, LV and GS restricted to a provided max number. All else are free variables.
- Loops over all values over all variables and creates the entire file tree with JSON file inputs to CaR and RMOC
- Calls CaR exe per loop
- Calls RMOC exe after creating the entire file tree
- Files in green stored *after* the exe runs from RMOC or CaR

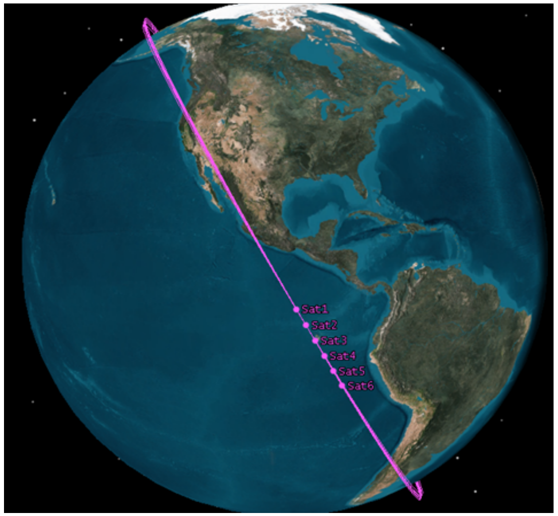
```
User Directory
├───TradespaceSearchRequest.json
├───InstrumentSpecifications.txt
├───ObservatorySpecifications.txt
├───Landsat_landImages.txt
├───Mono
│   ├───Orb0000
│   │   ├───Pnt000_Pay00
│   │   │   ├───ReductionMetrics.json
│   │   │   ├───obs.csv
│   │   │   └───angles.csv
│   │   ├───Pnt000_Pay01
│   │   ├───Pnt000_Pay02
│   │   └───....
│   ├───Orb0001
│   ├───Orb0002
│   └───....
├───Ground
│   ├───GS01
│   │   ├───GroundStations.json
│   ├───GS02
│   └───....
└───DSMs
    ├───Subspace00000
    │   ├───Manifest.json
    │   ├───CostRisk.json
    │   ├───gbl.csv
    │   ├───lcl.csv
    │   └───CostOutput.json
    ├───Subspace00001
    ├───Subspace00002
    └───.....
```

Types of Constellations

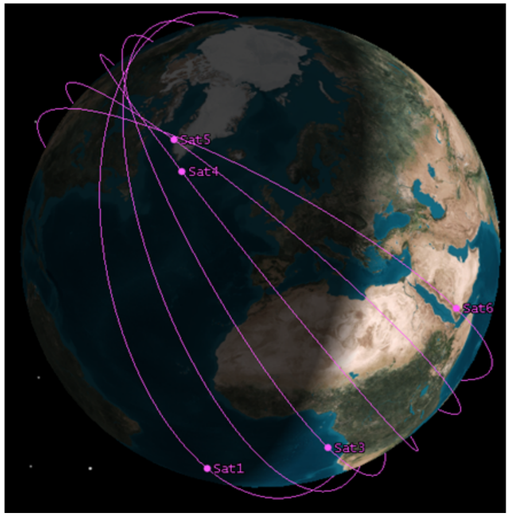
- Homo and Heterogeneous Walker Constellations i.e. a different altitude inclination combination for each plane with a different RAAN
- Added precessing type constellations which spread in RAAN over time due to being dropped off at slightly different altitudes and inclinations.
  - The time required is a function of the differentials and chief orbit
  - The differentials are a function of delta-V available, chief orbit and LV relights
  - The delta-V available is a function of the LV, LV and payload mass, etc.

Time ….

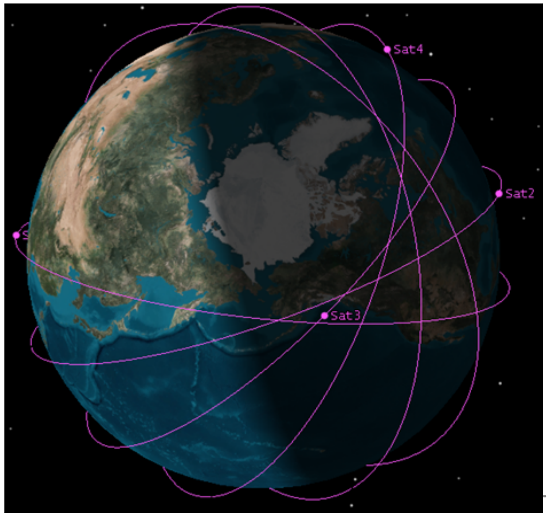
Deployment + 0 months



Deployment + 3 months

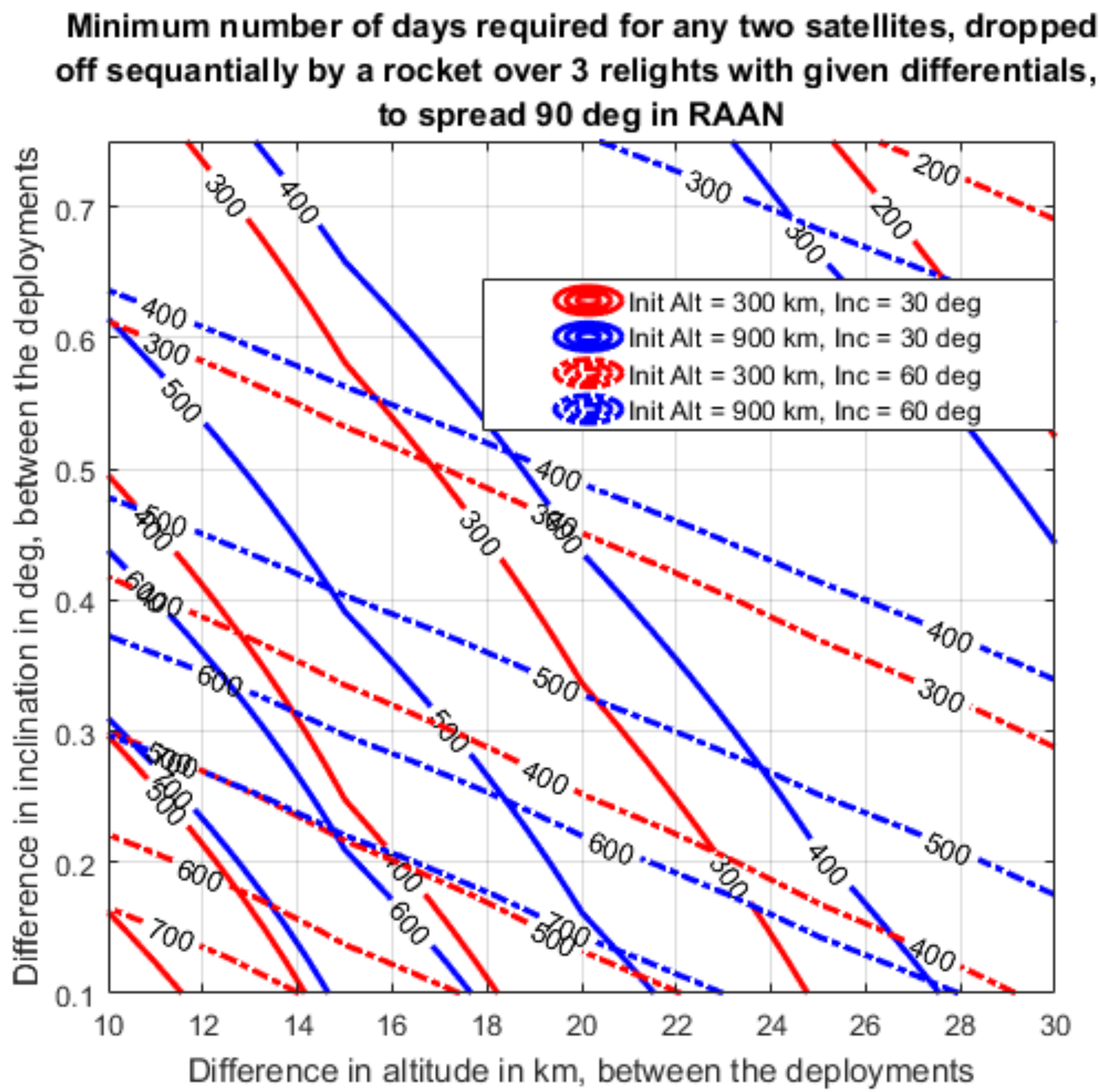


Deployment + 6 months



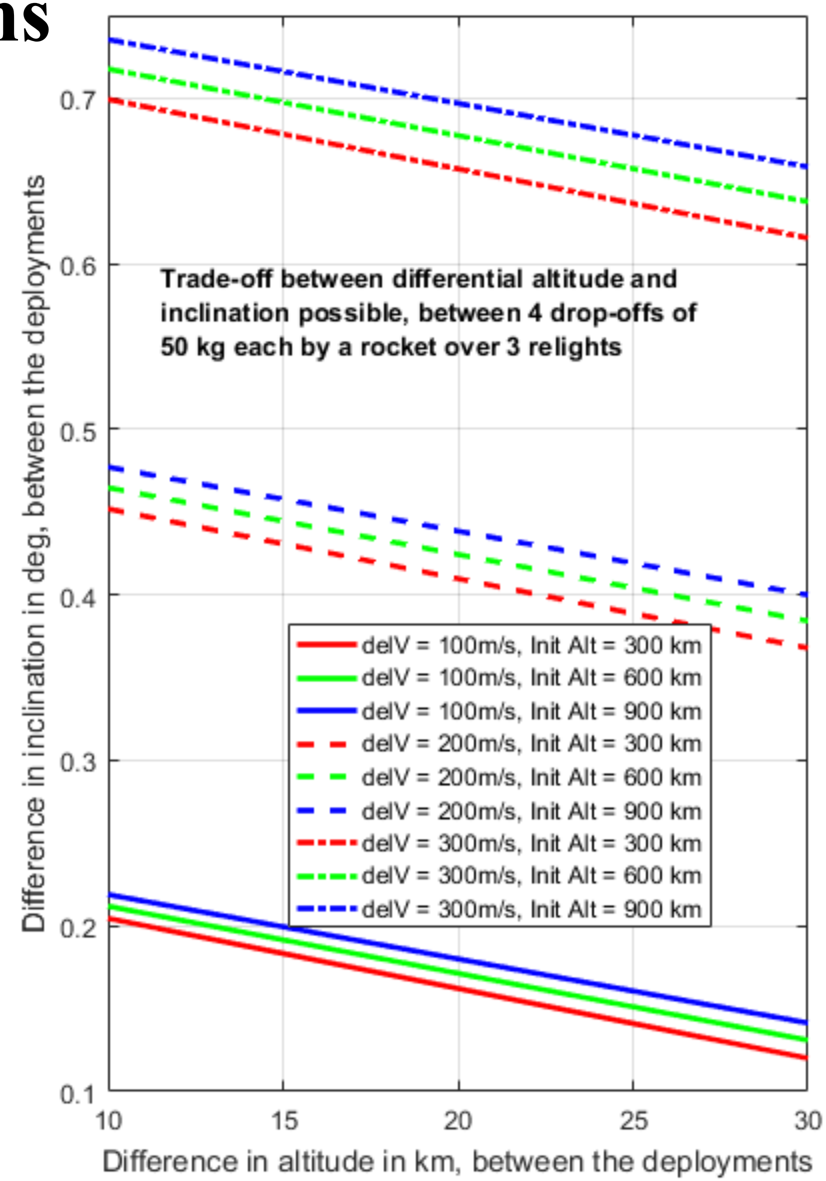
Precessing Constellations

- **The time required to spread out in RAAN is a function of the differentials and chief orbit**
- The differentials are a function of delta-V available, chief orbit and LV relights
- The delta-V available is a function of the LV, LV and payload mass, etc.



Precessing Constellations

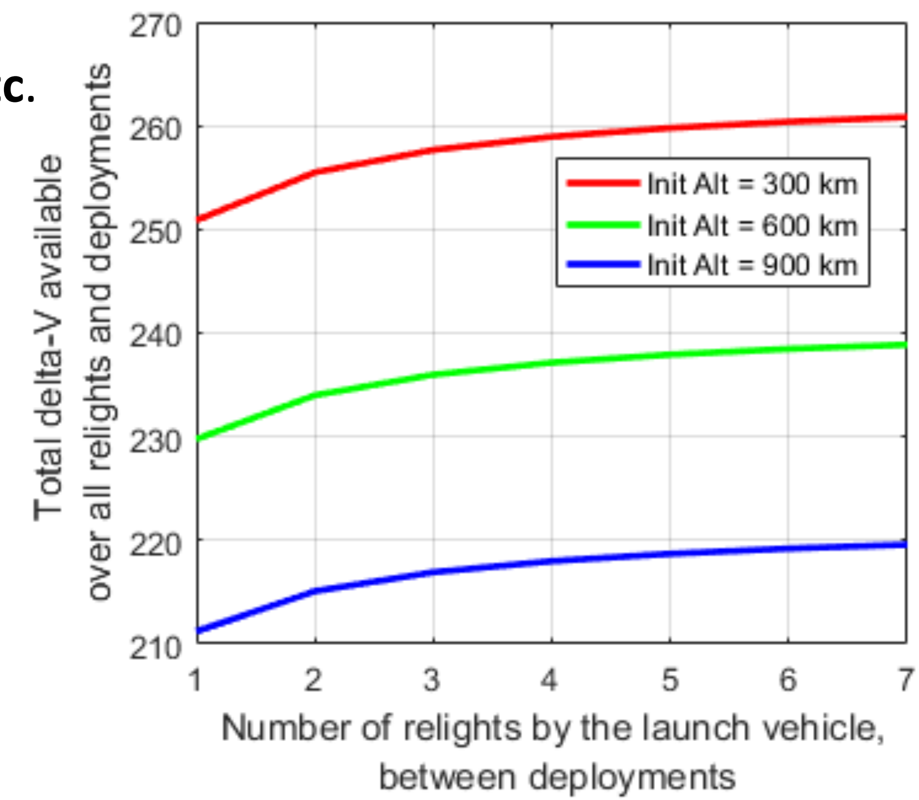
- The time required to spread out in RAAN is a function of the differentials and chief orbit
- **The differentials are a function of delta-V available, chief orbit and relights**
  - Example shown for 3 relights and 4 dropoffs using my equation below assuming equal differential-alt and inc at every relight
  - In a practical scenario, usually the fuel expended is considered equal at relight and corresponding differentials computed
  - Not a very large difference eitherway
- The delta-V available is a function of the LV, LV and payload mass, etc.



$$\Delta V_{total} = \sqrt{\mu/_r} + \sqrt{\mu/_{[r+n\Delta h]}} + 2\sin[\Delta i/_2] \sum_{m=0}^{n-1} \sqrt{\mu/_{[r+m\Delta h]}}$$

Precessing Constellations

- The time required to spread out in RAAN is a function of the differentials and chief orbit
- The differentials are a function of delta-V available, chief orbit and relights
- **The delta-V available is a function of the LV, LV and payload mass, etc.**

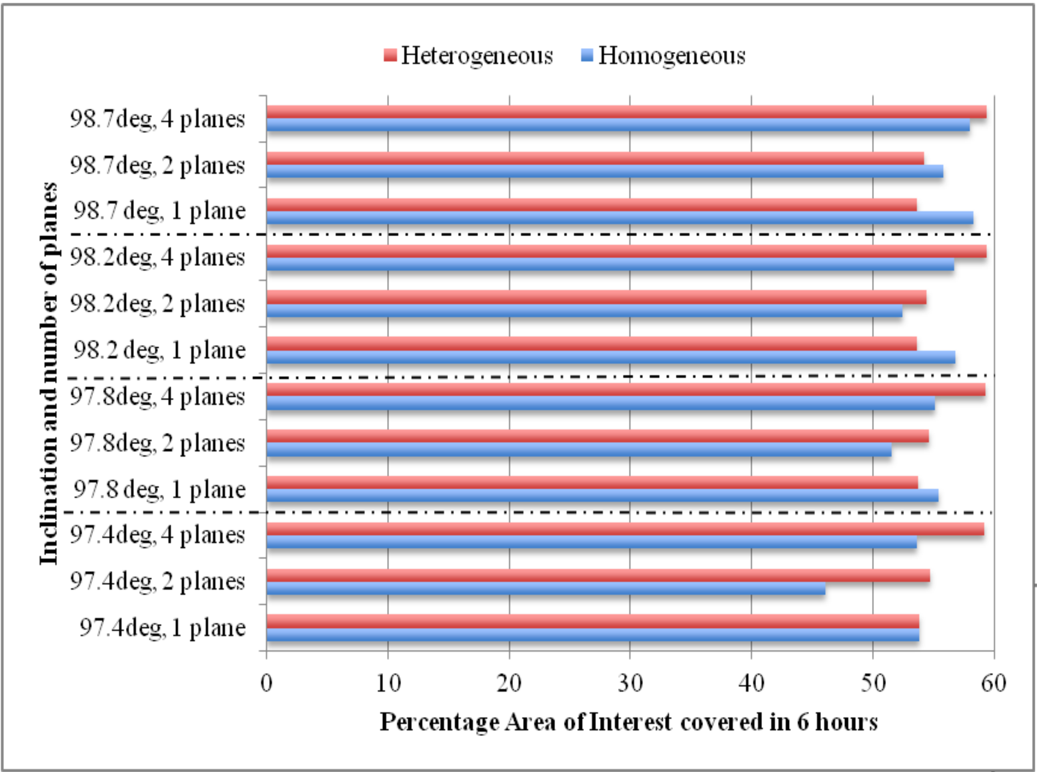


Example: Orbital ATK Pegasus Unlimited relights, 200 kg payload mass, 127 kg HAPS mass (for more precise initial orbit injection), 50 kg adapter mass, 57 kg total fuel mass (ED keeps 30% margin on fuel because of pre-Phase A uncertainty)

Delta-V available per drop and total, can be calculated from LV params.

One architecture per nsats, chief alt, chief inc and LV (easily scales with the DB) represented by differentials => computed per time required or maximum available fuel

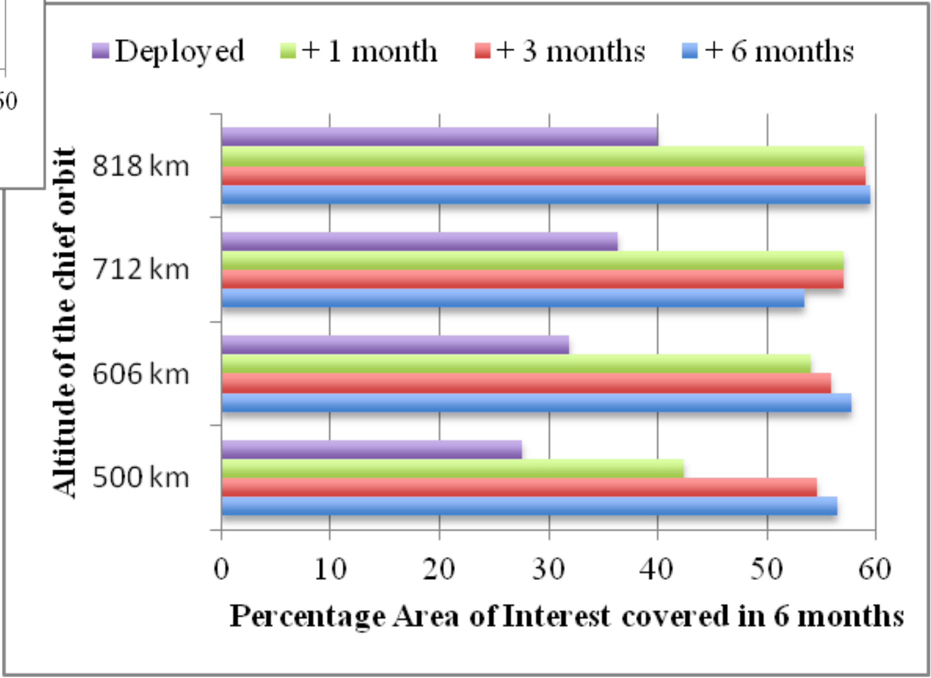
Results: Percentage Area Covered



Walker Constellations

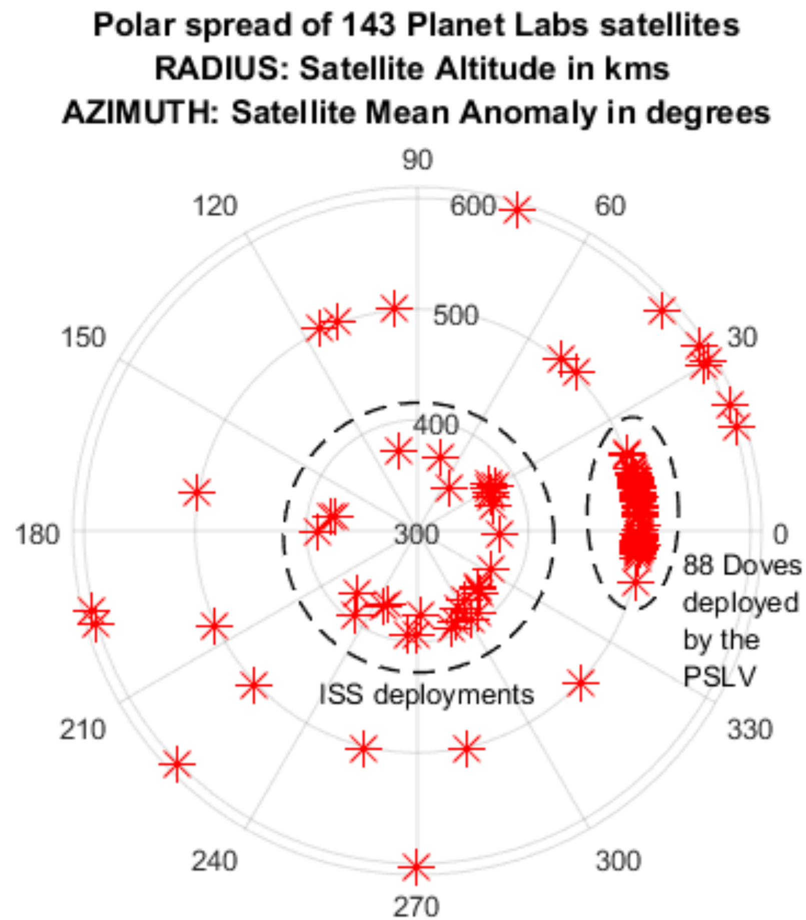
VS.

Precessing Constellations



Types of Constellations

- Ad-Hoc constellations (and even deployment using a single rocket) are well-captured by Planet Labs and their well-spread 143 satellites
- Currently, 100 sats @ 500+/-3 km SSO, 11 sats @ 600+/-3 km SSO, 32 sats @ ISS orbit < 400 km
- All TLE data is online
- 88 Doves deployed on Feb 15, 2017 and TLE analyzed on Feb 18, 2017
- Total TA spread = 27.5 deg
- Average between satellites = 0.3 deg (assumed to be constrained only by launch tracking window)
- All launched within 10 minutes

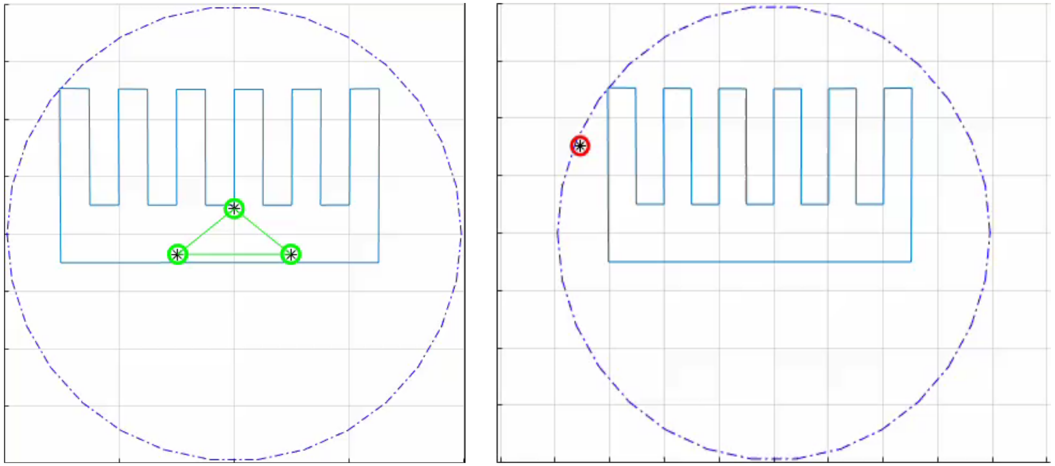


Data Reducer and Metric Computer

- Processes all Mono satellite files and all Ground satellite files in file tree
- Propagates all spacecraft and stores coverage in memory => OC call
- Can support atmospheric drag modeling (for missions without maintenance) and rectangular sensors in the last 6 months => OC call

$$cone = \cos^{-1}(\cos(alFOV/2)\cos(crFOV/2))$$

$$clock = \sin^{-1}\left(\sin(crFOV/2)\big/\sin(aFOV/2) * \sin(cone)\right)$$



- Processes all the DSM files in file tree as a permutation of Mono, and computes results per DSM and all ephemeris per mono

Data Reducer and Metric Computer

- 4 types of outputs: 2 per architecture and 2 per monolithic spacecraft
- Architecture: gbl.csv and lcl.csv
- Mono: eph.csv and angles.csv

gbl.csv

Time [s]		TimeToCoverage[s]			AccessTime [s]			RevisitTime [s]			Coverage	NumOfPOIpasses		
t0	t1	TCavg	TCmin	TCmax	ATavg	ATmin	ATmax	RTavg	RTmin	RTmax	% Grid Covere	PASavg	PASmin	PASmax
0	2592000	200098	27	906039	19.3808	0	54	242682	5832	906282	99.3647	12.2833	0	102

	Data Latency [s]			NumGSpasses	TotalDownlinkT	DownlinkTimePerPass [s]			CrossSwath[km]			AlongSwath [km]			SpatialResolution [m]	
PASmax	DLavg	DLmin	DLmax	PassesPerDay	DLTimePerDay	DLTavg	DLTmin	DLTmax	CSavg	CSmin	CSmax	ASavg	ASmin	ASmax	SRmin	SRmax
102	15500	5454	36882	5.4	2101.5	2.40226	54	486	185.852	185.852	185.852	185.852	185.852	185.852	29.5368	32.0538

Time [s]		POI	[deg]	[deg]	[km]	AccessTime [s]			RevisitTime [s]			TimeToCove	Number of Passes
t0	t1	POI	lat	lon	alt	ATavg	ATmin	ATmax	RvTavg	RvTmin	RvTmax	TCcov	numPass
0	2592000	128	82.8571	0	0	14.2105	0	27	46232.2	5859	83187	1485	57
0	2592000	129	82.8571	11.6129	0	18.4737	0	27	46227.9	5859	83160	1458	57
0	2592000	130	82.8571	23.2258	0	20.25	0	27	47066.9	5859	83160	1431	56
0	2592000	131	82.8571	34.8387	0	21	0	27	47273.5	5859	83160	78705	54
0	2592000	132	82.8571	46.4516	0	20.5	0	27	47385	5859	83160	72792	54
0	2592000	133	82.8571	58.0645	0	19.1455	0	27	46399.5	5859	83160	72765	55
0	2592000	134	82.8571	69.6774	0	16.5	0	27	47278	5859	83160	66852	54
0	2592000	135	82.8571	81.2903	0	11	0	27	47283.6	5859	83187	66825	54
0	2592000	136	82.8571	92.9032	0	5.89091	0	27	46304	5859	83187	66798	55
0	2592000	137	82.8571	104.516	0	2.41072	0	27	45572.6	5886	83187	60885	56

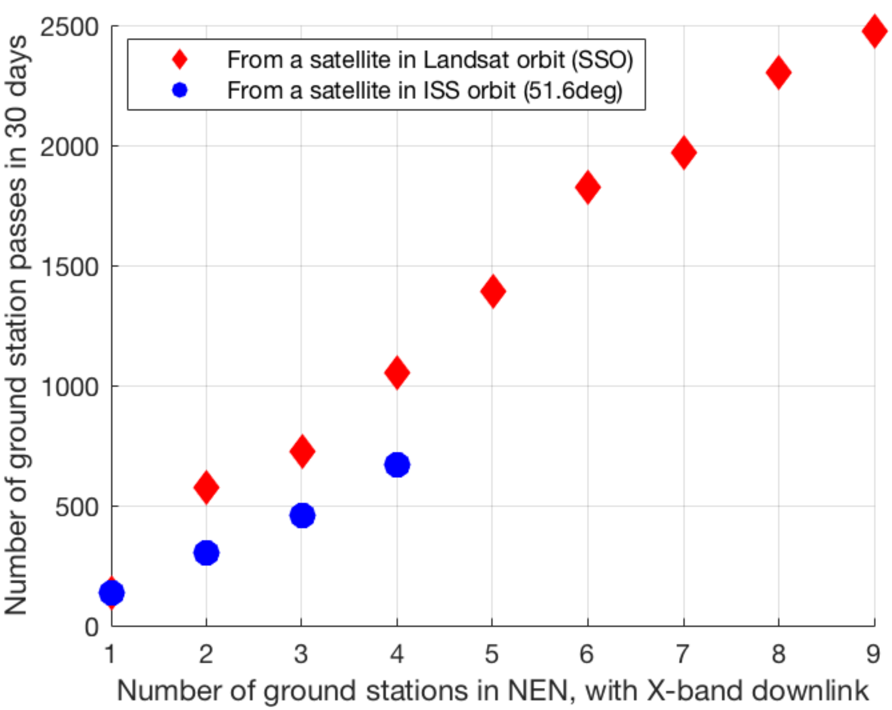
lcl.csv

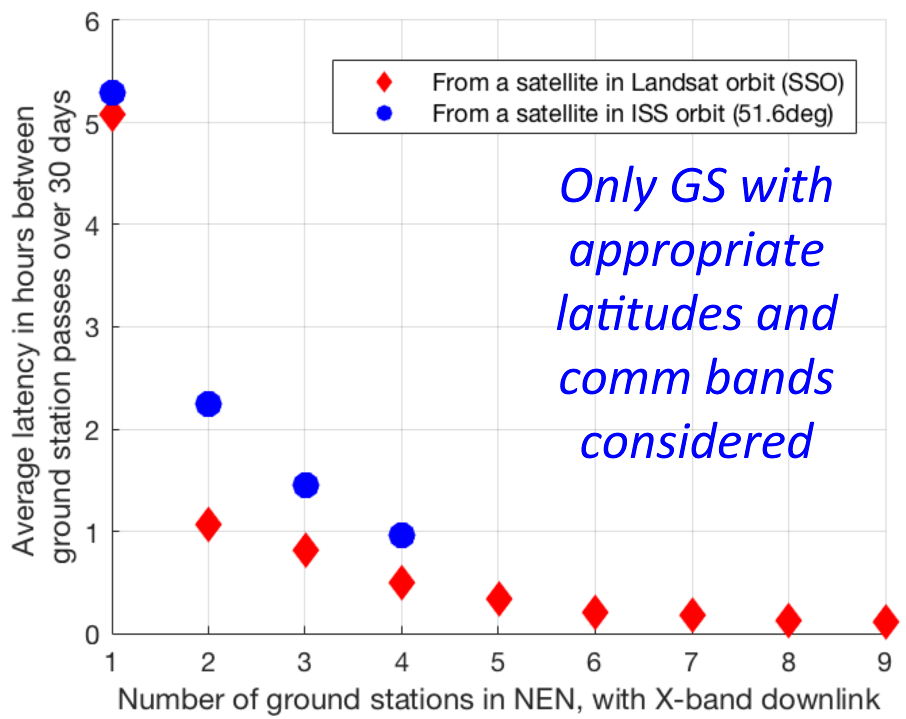
Time[s]	Ecc[deg]	Inc[deg]	SMA[km]	AOP[deg]	RAAN[deg]	MA[deg]	Lat[deg]	Lon[deg]	Alt[km]
43	3.82E-16	98.2081	7083.14	0	0.00049052	2.60613	2.59503	97.9013	705.047
86	2.87E-16	98.2081	7083.14	0	0.00098105	5.21226	5.18983	97.3483	705.177
172	3.33E-16	98.2081	7083.14	0	0.00196209	10.4245	10.3778	96.2315	705.692
215	6.10E-16	98.2081	7083.14	0	0.00245261	13.0306	12.9704	95.6645	706.073
258	4.30E-16	98.2081	7083.14	0	0.00294314	15.6368	15.5618	95.0894	706.532
344	5.62E-16	98.2081	7083.14	0	0.00392418	20.849	20.7401	93.9072	707.667
387	3.73E-16	98.2081	7083.14	0	0.00441471	23.4552	23.3264	93.2956	708.335
473	1.44E-16	98.2081	7083.14	0	0.00539575	28.6674	28.4919	92.0186	709.841

eph.csv

Results: Ground Station Trades

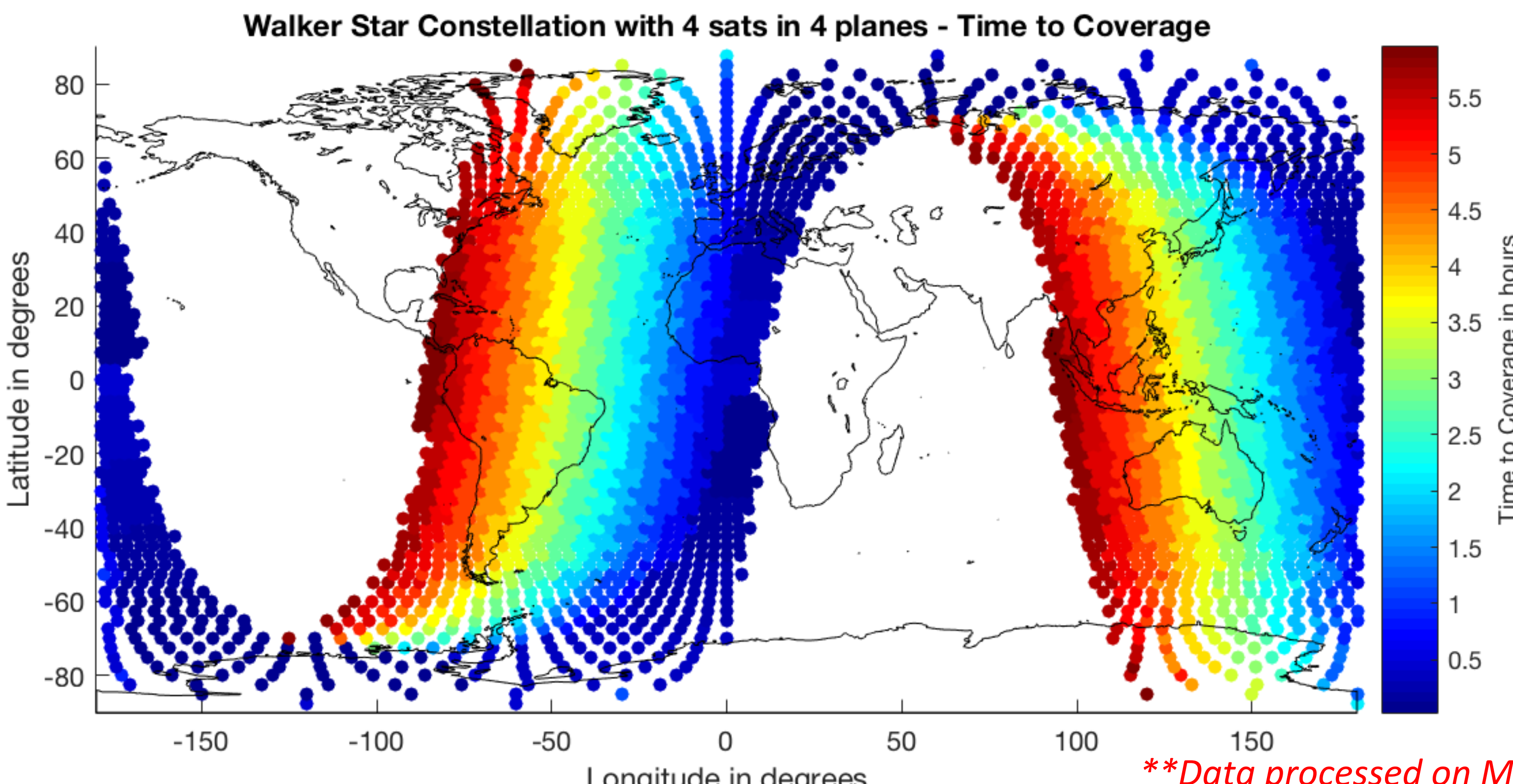
- The number and position of ground stations with respect to any satellite orbit affects the latency of data downlink, number of GS accesses or passes available, access time, etc.
- Example of 2 single-sats downlinking to the NEN
- The output bounds specified by user should determine the minimum number of ground stations to be included in the trade, for a given spacecraft and orbit.





Results: Inputs to a GMAO OSSE

- 4 satellite clusters in LEO between 500-820 km with a ~90 deg full FOV
- Each cluster is 120 deg apart in phase
- Find the best constellation to cover the globe in 6 hours
- % Coverage varies between 27.5% and 56.1% over all subspaces or arch
- -0.3% case shown... More non TAT-C optimization can improve till ~65%



***Data processed on MATLAB*

Summary / Future Work

- Software tools for the pre-phase A design of constellations for Earth Science are essential to understand trade-offs at the concept stage
- TAT-C will facilitate DSM Pre-Phase A investigations and by allowing the users to optimize DSM designs with respect to a-priori science goals [Full tool in a future publication]
- Executive Driver (ED), Orbit and Coverage (OC), Data Reduction and Metric Computation (RM) modules read user inputs and output constraints, generate architectures of constellations, propagate them and evaluate metrics
- Use Cases – Landsat, Wide Angle Radiometer. Results validated against AGI STK; New constellations, LV and GS trades introduced
- Future Work – Instrument module, higher fidelity LV module, addition of machine learning to tradespace search
- Parallel Work – Schedule optimization for agile steering constellations using ground or onboard autonomy

Acknowledgements

Funding: NASA Earth Science Technology Office (ESTO) Advanced Information Systems Technology Program 2014 (2015-2017)

Other Team Members: Paul Grogan, Matt Holland, Olivier de Weck/Afreen Siddiqi, Philip Dabney and Veronica Foreman

Thank you!

Questions?

Sreeja.Nag@nasa.gov

sreejanag@alum.mit.edu