Integration of GIS and Transportation Planning

Nick Meyers, GISP
Tyler Johnson, CFM

July 14, 2016

Agenda

- Purpose of a Prediction Tool
- Demo web GIS Application
- Python Programming

Purpose

1. Integration of GIS capabilities
2. Identify crash “hot spots” by intersection
3. Inform staff and clients of problem areas
4. Improve success with funding opportunities

How it will be used

- Tool will be used two times per year
  - Metro
  - Greater Minnesota

- Application contains two separate crash analysis datasets for reactive and proactive projects

Reactive dataset

- 3 years of crash data (2013-2015)
  - Determined the following by intersection:
    - 3 Year Crash Cost
    - Total Benefits Over Service Life

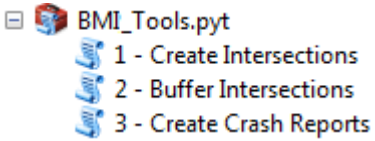
Proactive dataset

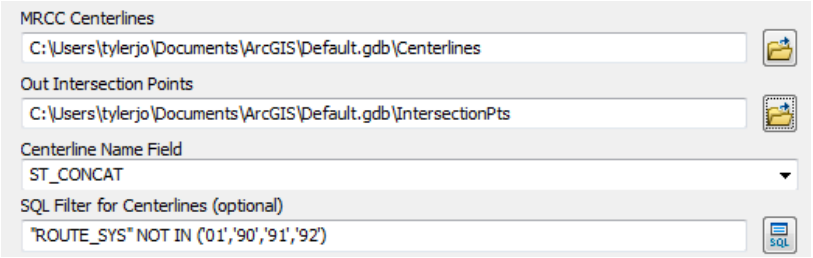
- 10 years of crash data (2006-2015)
  - To calculate Total Fatal & Incapacitating Crashes by intersection

Web GIS Demo

How the Script Tool Works

- Everything saved within a .PYT Python toolbox
  - Approximately 1,300 lines
  - Several standard lib and 3rd party modules
  - Includes toolbox and tool setup as well as helper functions and schema lists/dictionaries
- 3 Tools/Steps:
  - Create Intersection Points
  - Buffer Intersections
  - Create Crash Reports





Getting Intersection Points

- Input centerline dataset
- Parameters for street name field and SQL query
- Make filtered feature layer and run self-intersect
- Create attributed points through GP tools and cursors

```
# spatial join to get intersections and clean up field
spa_join = r'tmp_join'
arcpy.analysis.SpatialJoin(unique, unsplit, spa_join, field_mapping=fmappings)
with arcpy.da.UpdateCursor(spa_join, ['INTERSECTIONS']) as rows:
    for row in rows:
        if row[0] and not row[0] in ("","  ","0"):
            st_lst = [s.strip().upper() for s in row[0].split('&')]
            streets = ' & '.join(sorted(list(set(filter(None, st_lst)))))
        else:
            streets = "Unnamed Streets"
        rows.updateRow([streets])

# buffer features
tmp_buf = r'tmp_buffers'
arcpy.analysis.Buffer(spa_join, tmp_buf, '100 Feet',
                      dissolve_option='LIST', dissolve_field='Intersections')

# explode (needed so intersection points aren't created midway b/n multipart buffers
buffer_explode = 'tmp_buffer_explode'
arcpy.MultipartToSinglepart_management(tmp_buf, buffer_explode)

# create intersections
tmp_pts = "tmp_points"
arcpy.management.FeatureToPoint(buffer_explode, out_pts)
arcpy.AddMessage(out_pts)

# add intersection ID, clean up intersection field and add count
c = 1
arcpy.management.AddField(out_pts, 'Count', 'SHORT')
arcpy.management.AddField(out_pts, INT_ID, 'TEXT')
with arcpy.da.UpdateCursor(out_pts, [INT_ID, 'Intersections', 'Count']) as rows:
    for row in rows:
        row[0] = 'N-{}'.format(c)
        row[2] = row[1].count('&') + 1
        if row[2] < 2:
            rows.deleteRow()
        else:
            rows.updateRow(row)
            c += 1
```

Buffering Intersection Points

- Crash data is tabulated for each intersection point
- Crashes are clipped and spatially joined to intersection buffers
- Script uses dictionaries, nested lists, and search/update cursors to populate table

```
diag_dom = {1: 'REAR_END',
            2: 'SIDESWIPE',
            3: 'LEFT_TURN',
            98: 'NOT_APPLICABLE',
            99: 'UNKNOWN_TYPE',
            90: 'OTHER',
            -8: 'BIKE_CRASH',
            -9: 'PED_CRASH'}

crash_schema = [('TOT_CRASH', 'LONG', 'Total Crashes'),
                ('REAR_END', 'LONG', 'Rear End'),
                ('SIDESWIPE', 'LONG', 'Sideswipe'),
                ('LEFT_TURN', 'LONG', 'Left Turn'),
                ('RAN_OFF_LEFT', 'LONG', 'Ran off Road - Left Side'),
                ('RIGHT_ANGLE', 'LONG', 'Right Angle'),
                ('RIGHT_TURN', 'LONG', 'Right Turn'),
                ('RAN_OFF_RIGHT', 'LONG', 'Ran off Road - Right Side'),
```

```
    # buffer intersections
    int_buff = 'IntersectionBuffers_{}'.format(suffix)
    arcpy.analysis.Buffer(out_pts, int_buff, buff_dist)

    # add fields based on crash_schema nested list
    for f in crash_schema:
        arcpy.management.AddField(int_buff, f[0], f[1], field_alias=f[2])

    # make filtered crash feature layer and clip to buffers
    arcpy.AddMessage("\nCreating crash points layer from {}\n  Where clause: {}".format(crash_points, where_clause))
    crash_lyr = str(arcpy.management.MakeFeatureLayer(crash_points, 'crash_lyr', where_clause).getOutput(0))
    crash_clip = r'in_memory\crash_pts_clip'
    arcpy.analysis.Clip(crash_lyr, int_buff, crash_clip)

    # spatial join clipped crash point layer with intersection buffers
    tmp_crsh = r'Crashes_{}'.format(suffix)
    arcpy.analysis.SpatialJoin(crash_clip, int_buff, tmp_crsh, "JOIN_ONE_TO_MANY")


    # create dictionary to store crash data
    # dictionary layout-> {int_id: ['TOT_CRASH','REAR','RIGHT','FATAL','INCAPACITATING','NON_INCAPACITATING','POSSIBLE',
    cd = {}
    with arcpy.da.SearchCursor(int_buff, INT_ID) as dctrows:
        for k in dctrows:
            cd[k[0]] = collections.OrderedDict(zip([f[0] for f in crash_schema],[0]*len([f[0] for f in crash_schema])))

    # calculate crash cost
    diagram_dict = {k:[] for k in cd.keys()}
    crsh_fields = [INT_ID, 'DIAG', 'SEV', 'CRASH_COST', 'TYPE_']
    with arcpy.da.SearchCursor(tmp_crsh, crsh_fields) as rows:
        for row in rows:
            if row[4] == 7:
                # crash with pedestrian
                crash_type = -9
            elif row[4] == 6:
                # crash with bicycle
                crash_type = -8
            else:
                crash_type = row[1]
            crash_severity = row[2]
            crash_cost = sev_dom[crash_severity]
            diagram_dict[row[0]].append([crash_type, crash_severity, crash_cost])
```

Creating Crash Reports By City

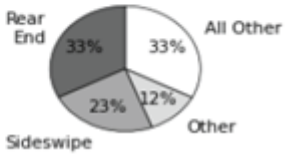
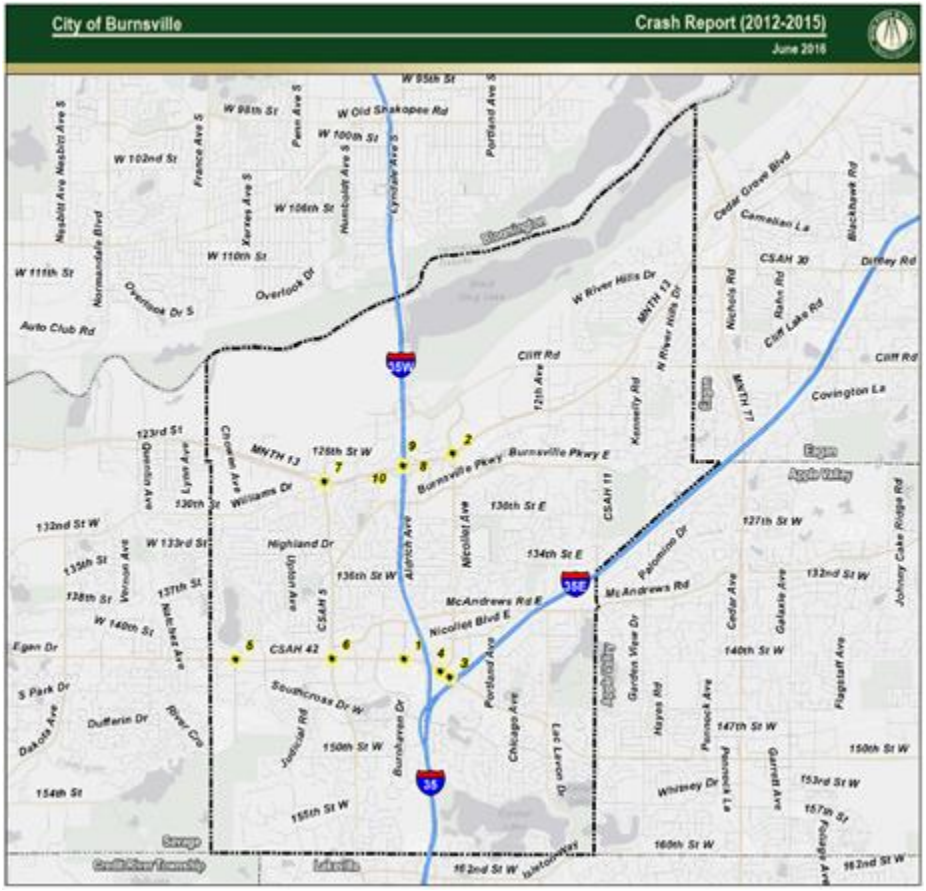
- Inputs are buffer centroids and city/township boundaries
- Utilizes *reportlab* and *matplotlib* modules, as well as template map documents to build a PDF report for each city

```
# list of strings we'll use to substitute into where_clause
strfmt = ctu_fields + ctu_str.split("-") + [sort_field]

# example clause: "Name" = 'Minneapolis' AND "CTU_Type" = 'CITY' AND "County_num" = '053' AND "CRASH_COST" > 0
where_clause = """{0}" = '{3}' AND "{1}" = '{4}' AND "{2}" = '{5}' AND "{6}" > 0""".format(*strfmt)

# use cursor to get a nested list of row values, then use heapq to get the 10 largest based on sort field
lst = [r for r in arcpy.da.SearchCursor(outjoin, table_fields, where_clause)]
top10rows = heapq.nlargest(10, lst, key=lambda x:x[sort_field_i])
top10IDs = [i[table_fields.index(INT_ID)] for i in top10rows]
```

Creating Reports continued…

```
doc = SimpleDocTemplate(out_pdf, pagesize=LETTER,
                        leftMargin=0.5*inch,
                        rightMargin=0.5*inch,
                        topMargin=0.2*inch,
                        bottomMargin=0.2*inch,)


t = Table(data, col_widths, row_heights)

img = Image(mappng)
img.drawHeight = 7*inch
img.drawWidth = 7.9*inch

mapdata = [[brandimg],[img]]
maptablewidths = [7.9*inch]*2
maptableheights = [0.6*inch, 7*inch]
maptable = Table(mapdata, colWidths=maptablewidths, rowHeights=maptableheights)
maptable.setStyle(TableStyle([('ALIGN',(0,0),(-1,-1),'CENTER'),
                        ('VALIGN', (0,0),(-1,-1),'MIDDLE'),
                        ('BOX', (0,1), (-1,-1), 0.75, colors.black)]))

elements = [t, PageBreak(), maptable]
doc.build(elements)
```

Crash Report (2012-2015)

June 2016

Crashes by Severity					Highest Frequency Crash Types
K	A	B	C	PDO	
0	0	3	6	34