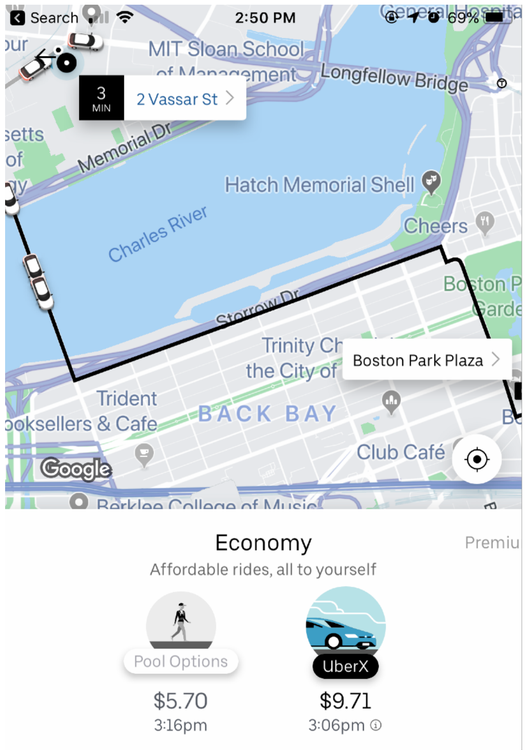
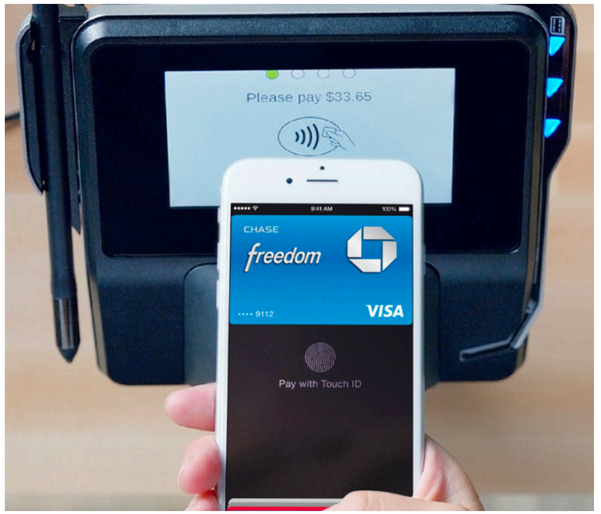
GraphIt: A DSL for High-Performance Graph Analytics

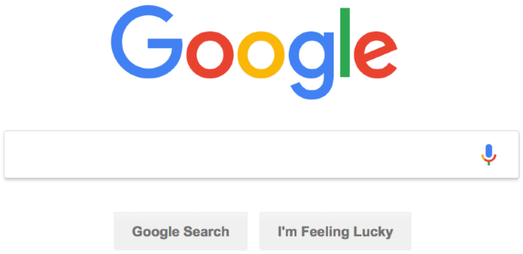
Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi,
Shoaib Kamil, Julian Shun, Saman Amarasinghe

Massachusetts
Institute of
Technology

Graphs Are Everywhere



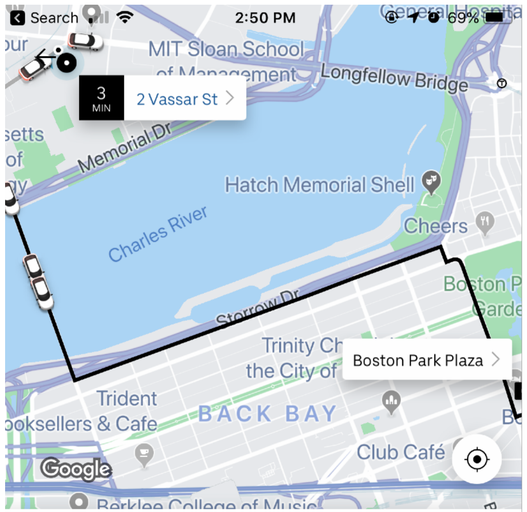


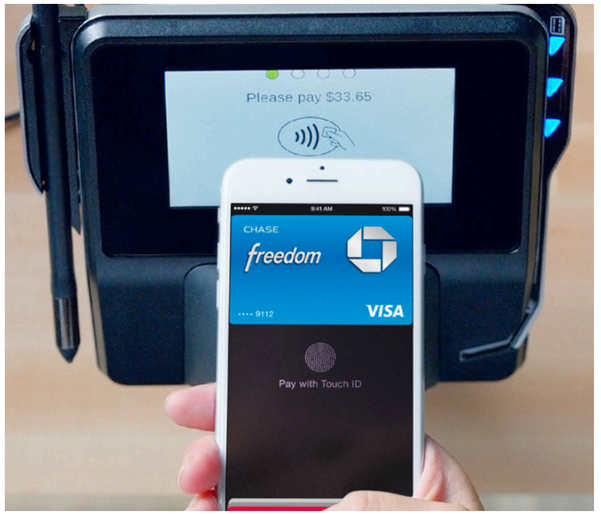


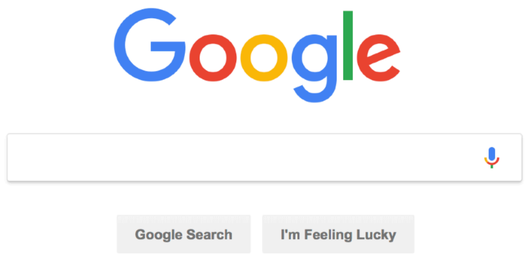
Recommendations for You, Yunming

http://google.com . https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png, https://www.bankinfosecurity.com/webinars/customer-awareness-what-works-in-fraud-detection-prevention-w-423 http://amazon.com https://makeawebsitehub.com/social-media-sites/

Performance is Important







Recommendations for You, Yunming

http://google.com . https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png, https://www.bankinfosecurity.com/webinars/customer-awareness-what-works-in-fraud-detection-prevention-w-423 http://amazon.com https://makeawebsitehub.com/social-media-sites/

PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
    for (i = 0; i < max_iter; i++) {
        for (src : graph.vertices()) {
            for (dst : graph.getOutgoingNeighbors(node)) {
                new_rank[dst] += old_rank[src]/out_degree[src]; } }
    for (node : graph.vertices()) {
        new_rank[node] = base_score + damping*new_rank[node]; }
    swap (old_rank, new_rank); }
}
```

PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
    for (i = 0; i < max_iter; i++) {
        for (src : graph.vertices()) {
            for (dst : graph.getOutgoingNeighbors(node)) {
                new_rank[dst] += old_rank[src]/out_degree[src]; } }
    for (node : graph.vertices()) {
        new_rank[node] = base_score + damping*new_rank[node]; }
    swap (old_rank, new_rank); }
}
```

PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
    for (i = 0; i < max_iter; i++) {
        for (src : graph.vertices()) {
            for (dst : graph.getOutgoingNeighbors(node)) {
                new_rank[dst] += old_rank[src]/out_degree[src]; } }
    for (node : graph.vertices()) {
        new_rank[node] = base_score + damping*new_rank[node]; }
    swap (old_rank, new_rank); }
}
```

Hand-Optimized C++

```
template<typename APPLY_FUNC>
void edgeset_apply_pull_parallel(Graph &g, APPLY_FUNC apply_func) {
    int64_t numVertices = g.num_nodes(), numEdges = g.num_edges();
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            local_new_rank[socketId][n] = new_rank[n]; } }
    int numPlaces = omp_get_num_places();
    int numSegments = g.getNumSegments("s1");
    int segmentsPerSocket = (numSegments + numPlaces - 1) / numPlaces;
        #pragma omp parallel num_threads(numPlaces) proc_bind(spread){
        int socketId = omp_get_place_num();
        for (int i = 0; i < segmentsPerSocket; i++) {
            int segmentId = socketId + i * numPlaces;
            if (segmentId >= numSegments) break;
            auto sg = g.getSegmentedGraph(std::string("s1"), segmentId);
            #pragma omp parallel num_threads(omp_get_place_num_procs(socketId)) proc_bind(close){
                #pragma omp for schedule(dynamic, 1024)
                for (NodeID localId = 0; localId < sg->numVertices; localId++) {
                    NodeID d = sg->graphId[localId];
                    for (int64_t ngh = sg->vertexArray[localId]; ngh < sg->vertexArray[localId + 1]; ngh++) {
                        NodeID s = sg->edgeArray[ngh];
                        local_new_rank[socketId][d] += contrib[s]; }}}}}
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            new_rank[n] += local_new_rank[socketId][n]; }}}
struct updateVertex {
    void operator() (NodeID v) {
        double old_score = old_rank[v];
        new_rank[v] = (beta_score + (damp * new_rank[v]));
        error[v] = fabs((new_rank[v] - old_rank[v])) ;
        old_rank[v] = new_rank[v];
        new_rank[v] = ((float) 0) ; }; };
void pagerank(Graph &g, double *new_rank, double *old_rank, int *out_degree, int max_iter) {
    for (int i = (0); i < (max_iter); i++) {
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            contrib[v] = (old_rank[v] / out_degree[v]);};
        edgeset_apply_pull_parallel(edges, updateEdge());
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            updateVertex()(v_iter); }; }
```

More than 23x faster

Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores.

Hand-Optimized C++

```
template<typename APPLY_FUNC>
void edgeset_apply_pull_parallel(Graph &g, APPLY_FUNC apply_func) {
    int64_t numVertices = g.num_nodes(), numEdges = g.num_edges();
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            local_new_rank[socketId][n] = new_rank[n]; } }
    int numPlaces = omp_get_num_places();
    int numSegments = g.getNumSegments("s1");
    int segmentsPerSocket = (numSegments + numPlaces - 1) / numPlaces;
        #pragma omp parallel num_threads(numPlaces) proc_bind(spread){
        int socketId = omp_get_place_num();
        for (int i = 0; i < segmentsPerSocket; i++) {
            int segmentId = socketId + i * numPlaces;
            if (segmentId >= numSegments) break;
            auto sg = g.getSegmentedGraph(std::string("s1"), segmentId);
            #pragma omp parallel num_threads(omp_get_place_num_procs(socketId)) proc_bind(close){
                #pragma omp for schedule(dynamic, 1024)
                for (NodeID localId = 0; localId < sg->numVertices; localId++) {
                    NodeID d = sg->graphId[localId];
                    for (int64_t ngh = sg->vertexArray[localId]; ngh < sg->vertexArray[localId + 1]; ngh++) {
                        NodeID s = sg->edgeArray[ngh];
                        local_new_rank[socketId][d] += contrib[s]; }}}}}
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            new_rank[n] += local_new_rank[socketId][n]; }}}
struct updateVertex {
    void operator() (NodeID v) {
        double old_score = old_rank[v];
        new_rank[v] = (beta_score + (damp * new_rank[v]));
        error[v] = fabs((new_rank[v] - old_rank[v])) ;
        old_rank[v] = new_rank[v];
        new_rank[v] = ((float) 0) ; }; };
void pagerank(Graph &g, double *new_rank, double *old_rank, int *out_degree, int max_iter) {
    for (int i = (0); i < (max_iter); i++) {
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            contrib[v] = (old_rank[v] / out_degree[v]);};
        edgeset_apply_pull_parallel(edges, updateEdge());
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            updateVertex()(v_iter); }; }
```

More than 23x faster

Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores.

Multi-Threaded

Load Balanced

NUMA Optimized

Cache Optimized

Hand-Optimized C++

```
template<typename APPLY_FUNC>
void edgeset_apply_pull_parallel(Graph &g, APPLY_FUNC apply_func) {
    int64_t numVertices = g.num_nodes(), numEdges = g.num_edges();
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            local_new_rank[socketId][n] = new_rank[n]; } }
    int numPlaces = omp_get_num_places();
    int numSegments = g.getNumSegments("s1");
    int segmentsPerSocket = (numSegments + numPlaces - 1) / numPlaces;
        #pragma omp parallel num_threads(numPlaces) proc_bind(spread){
        int socketId = omp_get_place_num();
        for (int i = 0; i < segmentsPerSocket; i++) {
            int segmentId = socketId + i * numPlaces;
            if (segmentId >= numSegments) break;
            auto sg = g.getSegmentedGraph(std::string("s1"), segmentId);
            #pragma omp parallel num_threads(omp_get_place_num_procs(socketId)) proc_bind(close){
                #pragma omp for schedule(dynamic, 1024)
                for (NodeID localId = 0; localId < sg->numVertices; localId++) {
                    NodeID d = sg->graphId[localId];
                    for (int64_t ngh = sg->vertexArray[localId]; ngh < sg->vertexArray[localId + 1]; ngh++) {
                        NodeID s = sg->edgeArray[ngh];
                        local_new_rank[socketId][d] += contrib[s]; }}}}}
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            new_rank[n] += local_new_rank[socketId][n]; }}}
struct updateVertex {
    void operator() (NodeID v) {
        double old_score = old_rank[v];
        new_rank[v] = (beta_score + (damp * new_rank[v]));
        error[v] = fabs((new_rank[v] - old_rank[v])) ;
        old_rank[v] = new_rank[v];
        new_rank[v] = ((float) 0) ; }; };
void pagerank(Graph &g, double *new_rank, double *old_rank, int *out_degree, int max_iter) {
    for (int i = (0); i < (max_iter); i++) {
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            contrib[v] = (old_rank[v] / out_degree[v]);};
        edgeset_apply_pull_parallel(edges, updateEdge());
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            updateVertex()(v_iter); }; }
```

More than 23x faster

Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores.

Multi-Threaded

Load Balanced

NUMA Optimized

Cache Optimized

(1) Hard to write correctly.

(2) Extremely difficult to experiment with different combinations of optimizations

GraphIt

A Domain-Specific Language for Graph Applications

- **Decouple algorithm from optimization for graph applications**
  - **Algorithm**: What to Compute
  - **Optimization (schedule)**: How to Compute

GraphIt

A Domain-Specific Language for Graph Applications

- **Decouple algorithm from optimization for graph applications**
  - **Algorithm**: What to Compute
  - **Optimization (schedule)**: How to Compute
- **Optimization (schedule) representation**
  - **Easy to use** for users to try different combinations
  - **Powerful** enough to beat hand-hand-optimized libraries by up to 4.8x

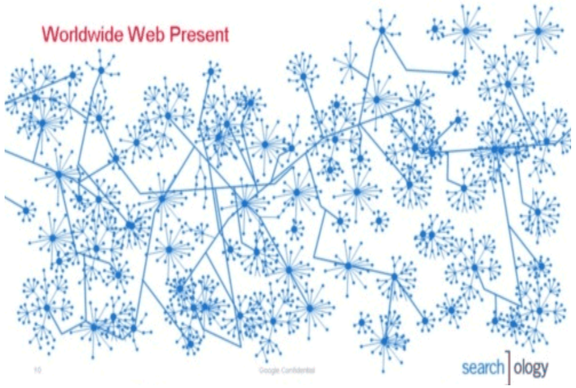
Outline

- Graph Applications Overview
- Optimization Tradeoff Space
- GraphIt DSL
- Evaluation

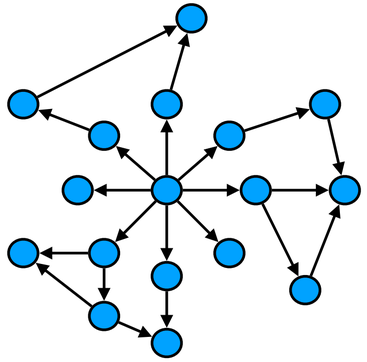
Outline

- Graph Applications Overview
- Optimization Tradeoff Space
- GraphIt DSL
- Evaluation

Power-Law Graphs

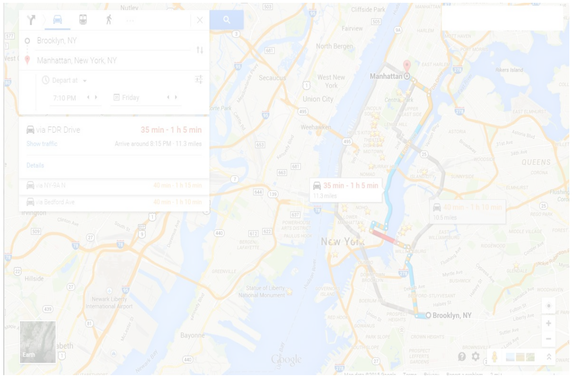


World Wide Web

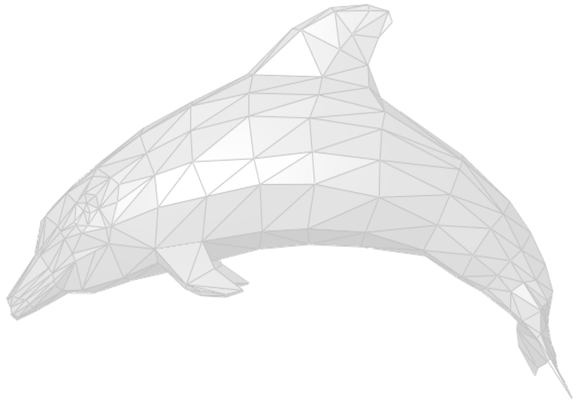


Power-Law Degree Distribution, Small Diameter, Poor Locality

Social Networks



Maps

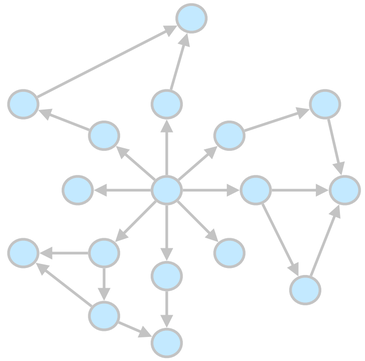


Engineering Meshes

1. http://googlesystem.blogspot.com/2007/05/world-wide-web-as-seen-by-google.html 2. http://www.facebookfever.com/introducing-facebook-new-graph-api-explorer-features/ 3. http://maps.google.com **4. https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png**

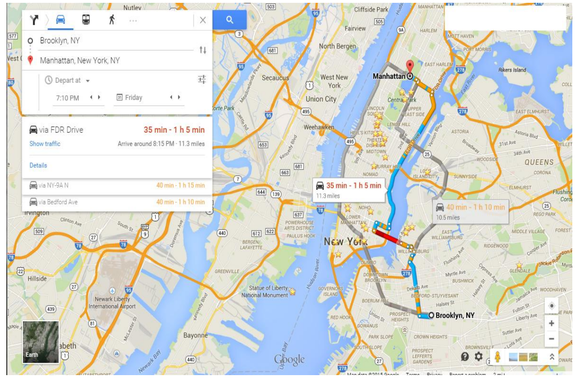
Bounded-Degree Graphs

World Wide Web

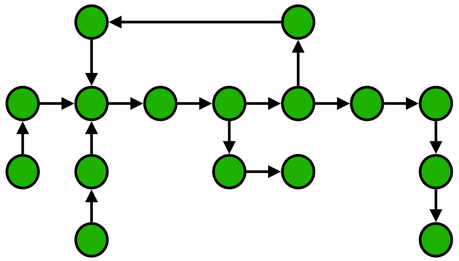


Power-Law Degree Distribution,
Small Diameter, Poor Locality

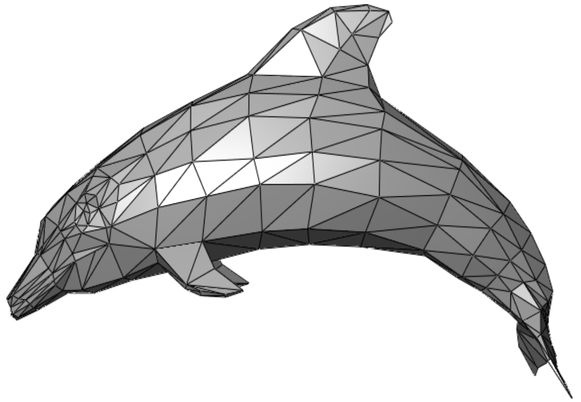
Social Networks



Maps



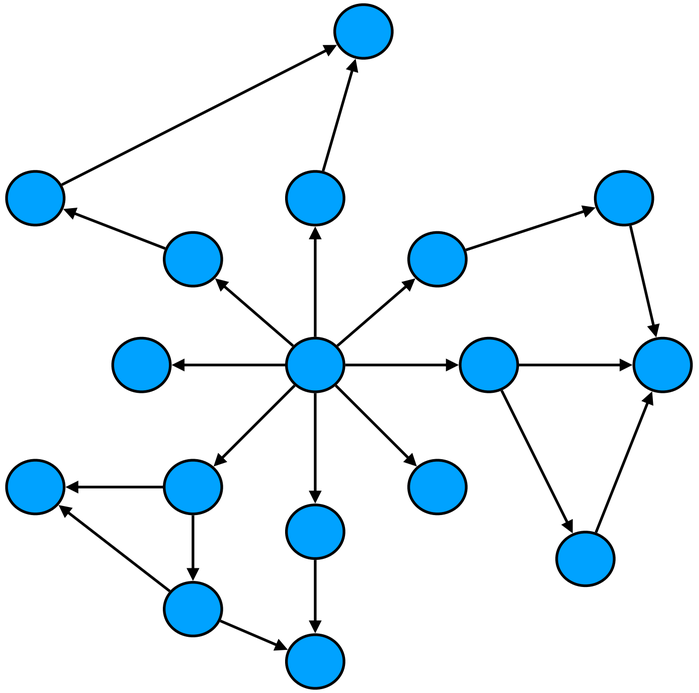
Bounded Degree Distribution
Large Diameter, Excellent Locality



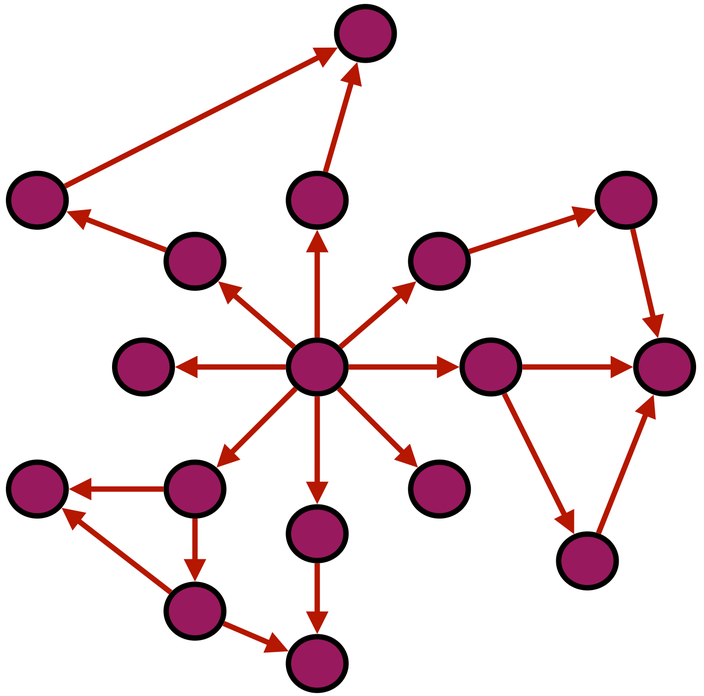
Engineering Meshes

1. http://googlesystem.blogspot.com/2007/05/world-wide-web-as-seen-by-google.html 2. http://www.facebookfever.com/introducing-facebook-new-graph-api-explorer-features/ 3. http://maps.google.com **4. https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png**

Graph Algorithms

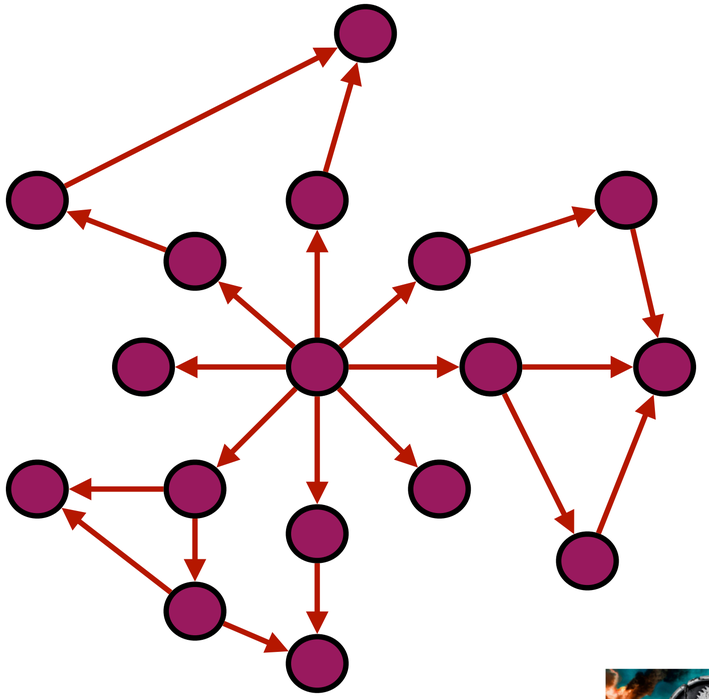


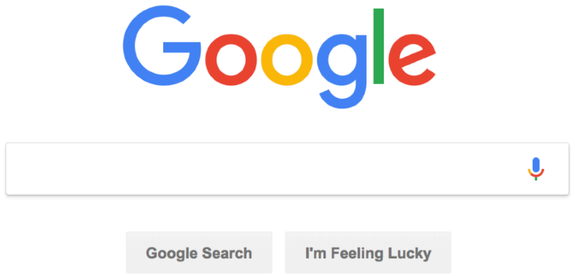
Topology-Driven Algorithms



Work on All Edges and Vertices

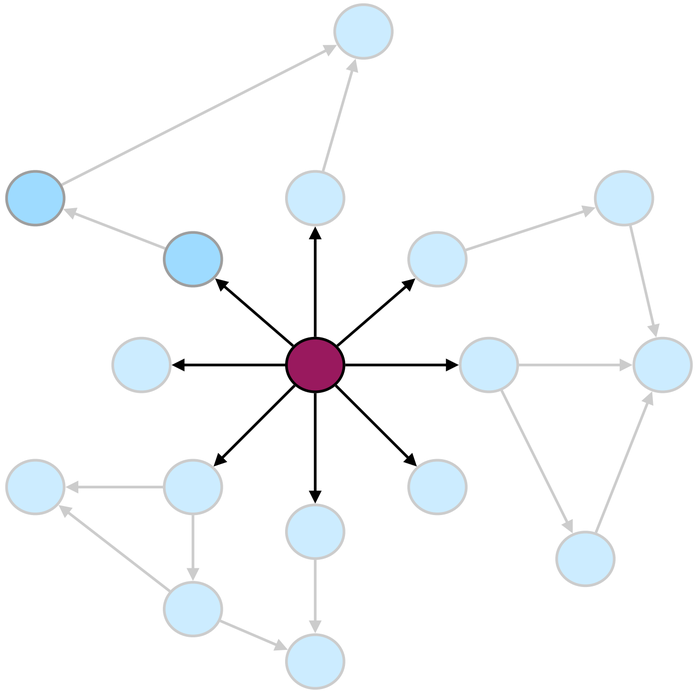
Topology-Driven Algorithms



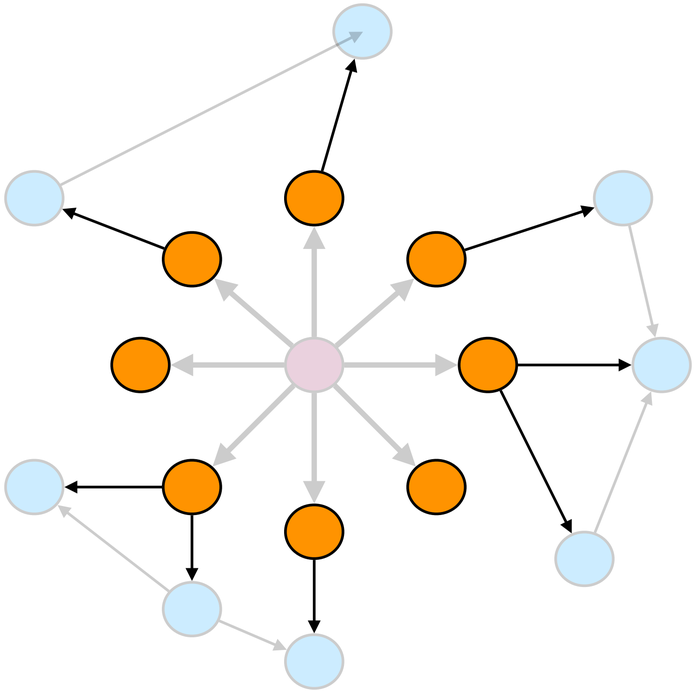


Recommendations for You, Yunming

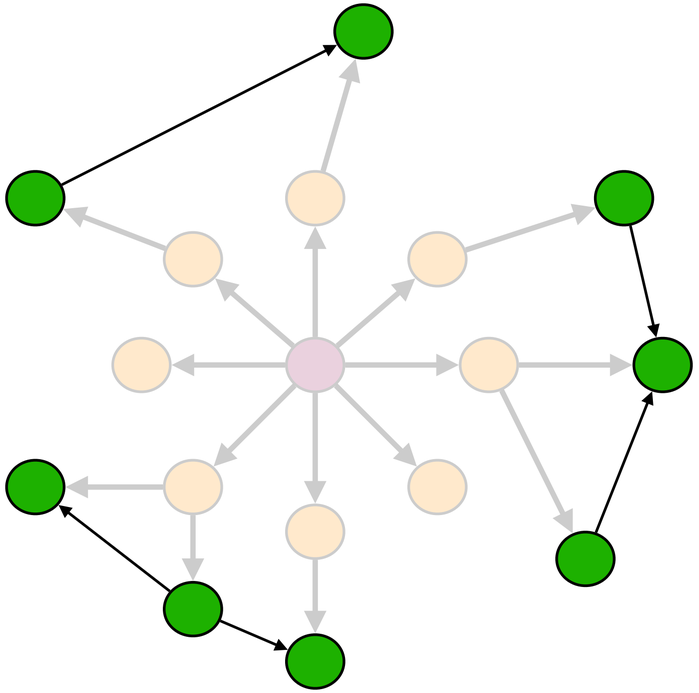
Data-Driven Algorithms



Data-Driven Algorithms

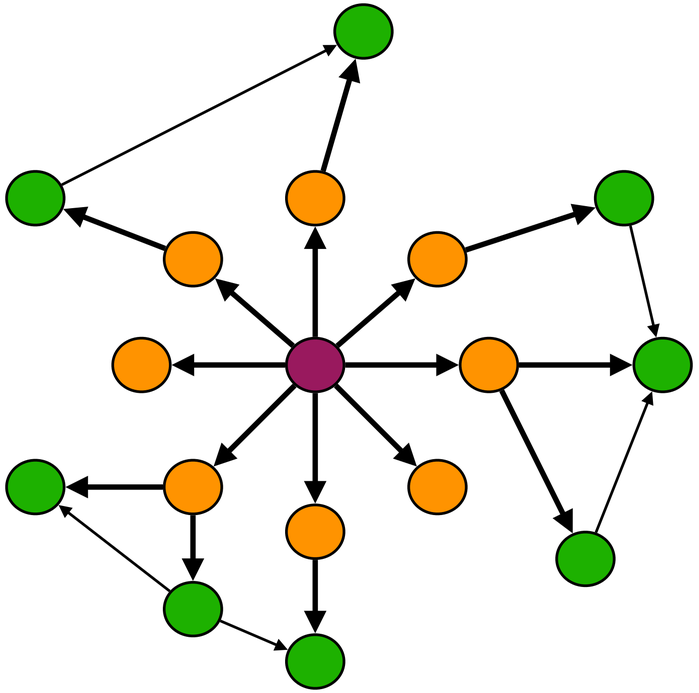


Data-Driven Algorithms



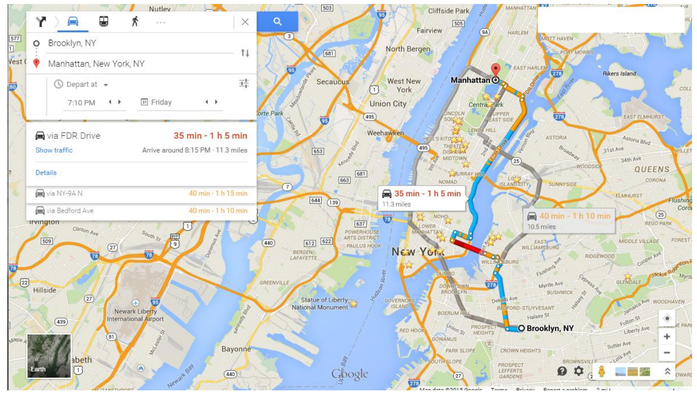
Work on a subset of vertices and edges
(Data-Driven)

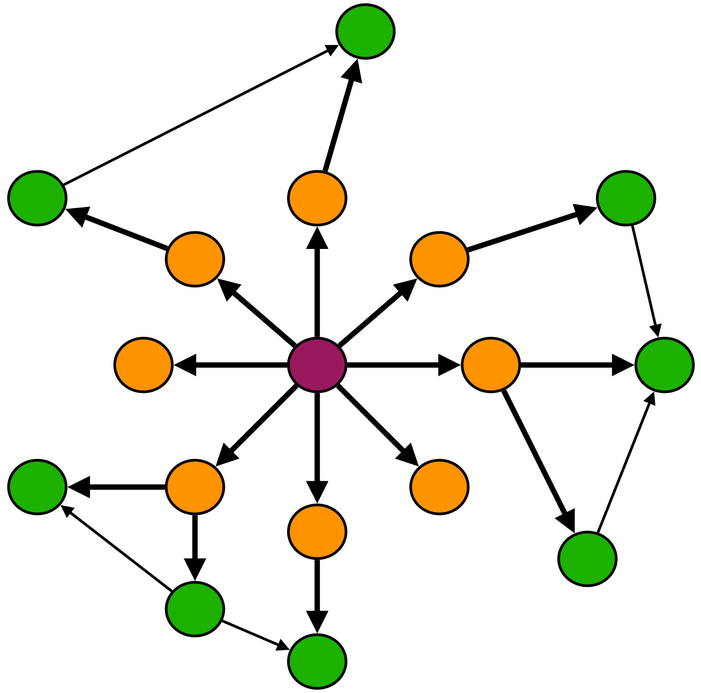
Data-Driven Algorithms

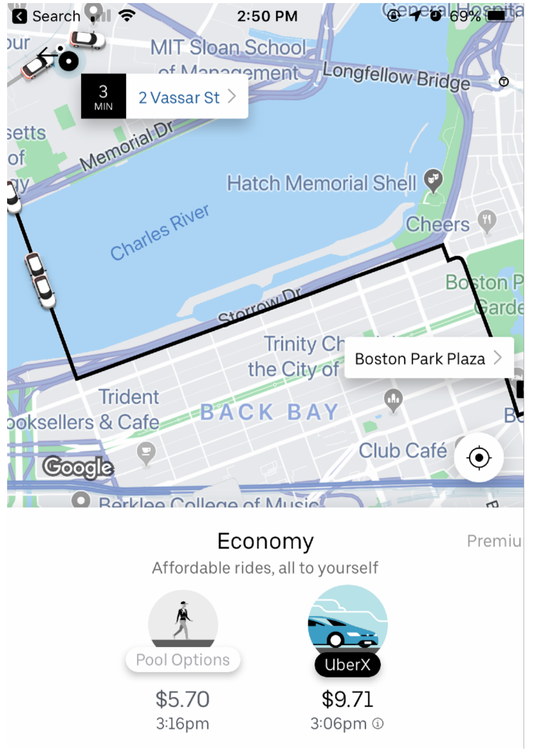


Work on a subset of vertices and edges
(Data-Driven)

Data-Driven Algorithms







Graph Execution Hardware

CPU

Xeon Phi

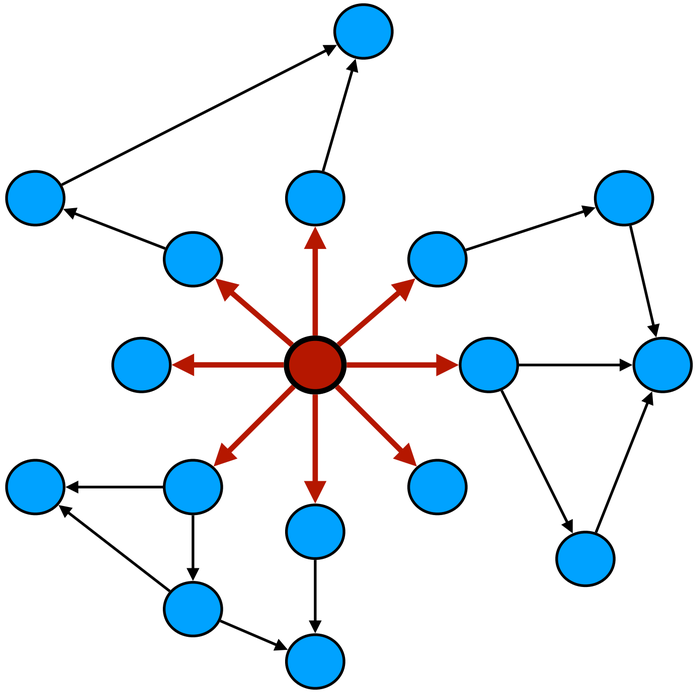
GPU

Distributed Cluster

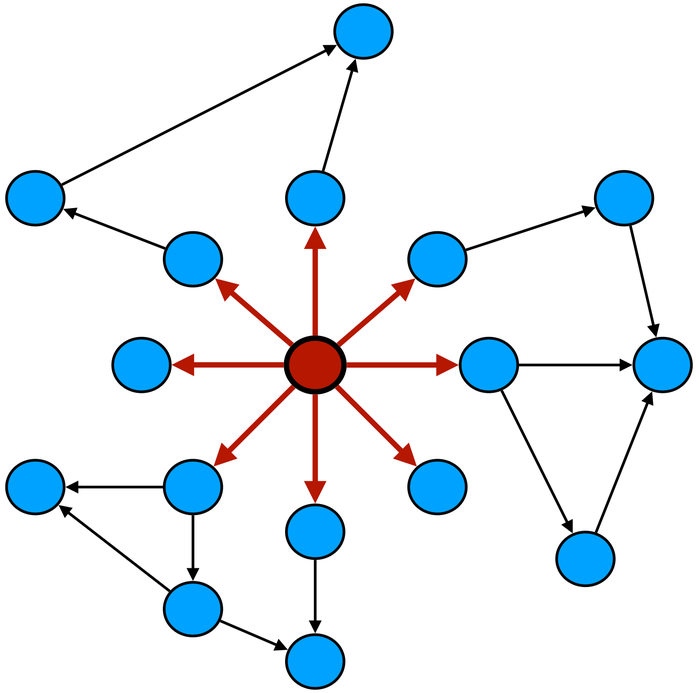
Outline

- Graph Applications Overview
- Optimization Tradeoff Space
- GraphIt DSL
- Evaluation

Push Traversal



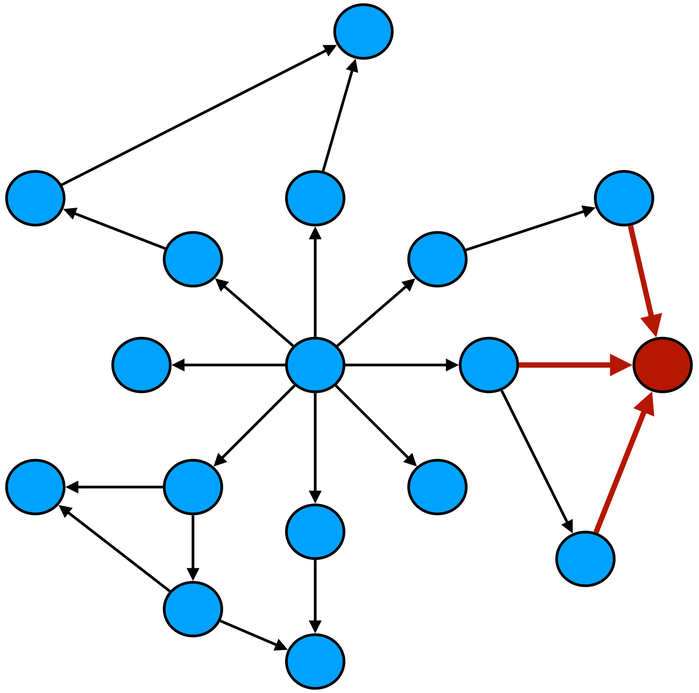
Push Traversal



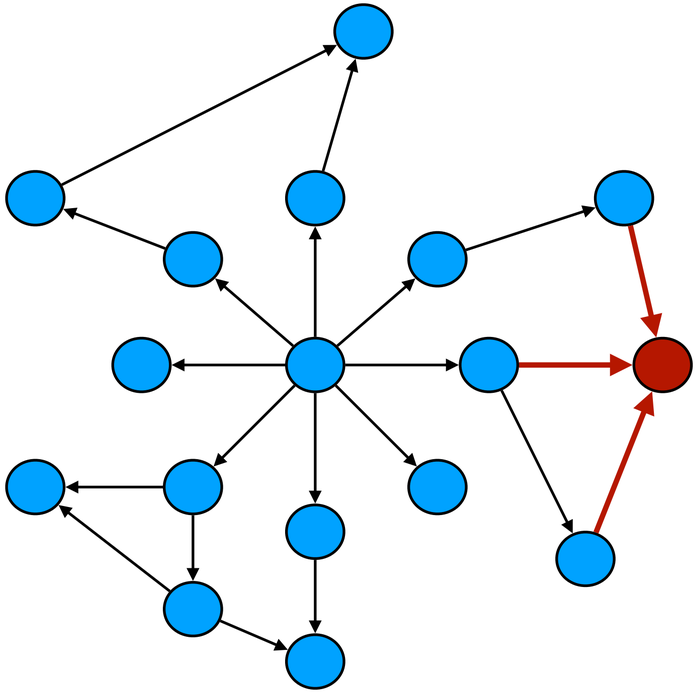
Incurs overhead with atomics

Traverses no extra edges

Pull Traversal



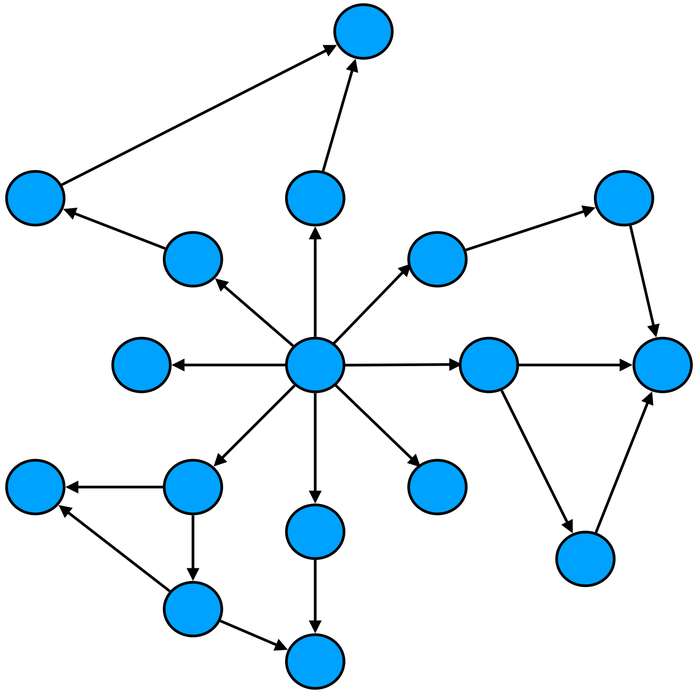
Pull Traversal



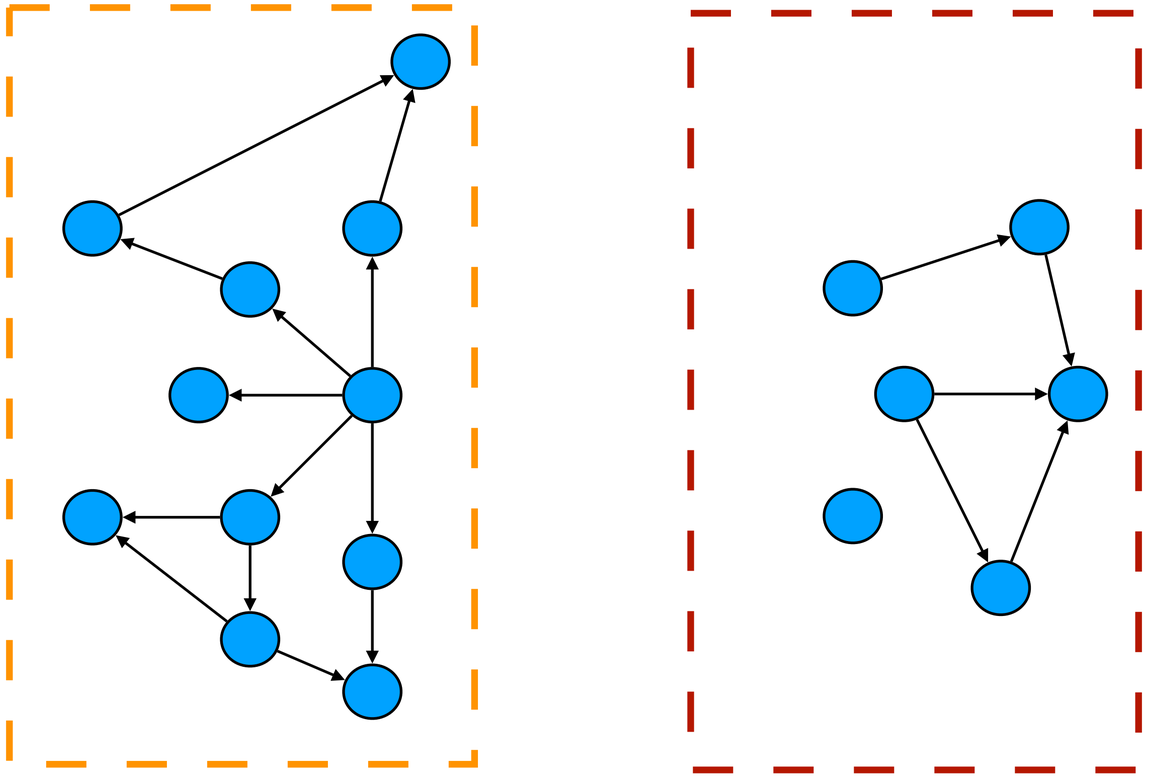
Incurs no overhead from atomics

Traverses extra edges

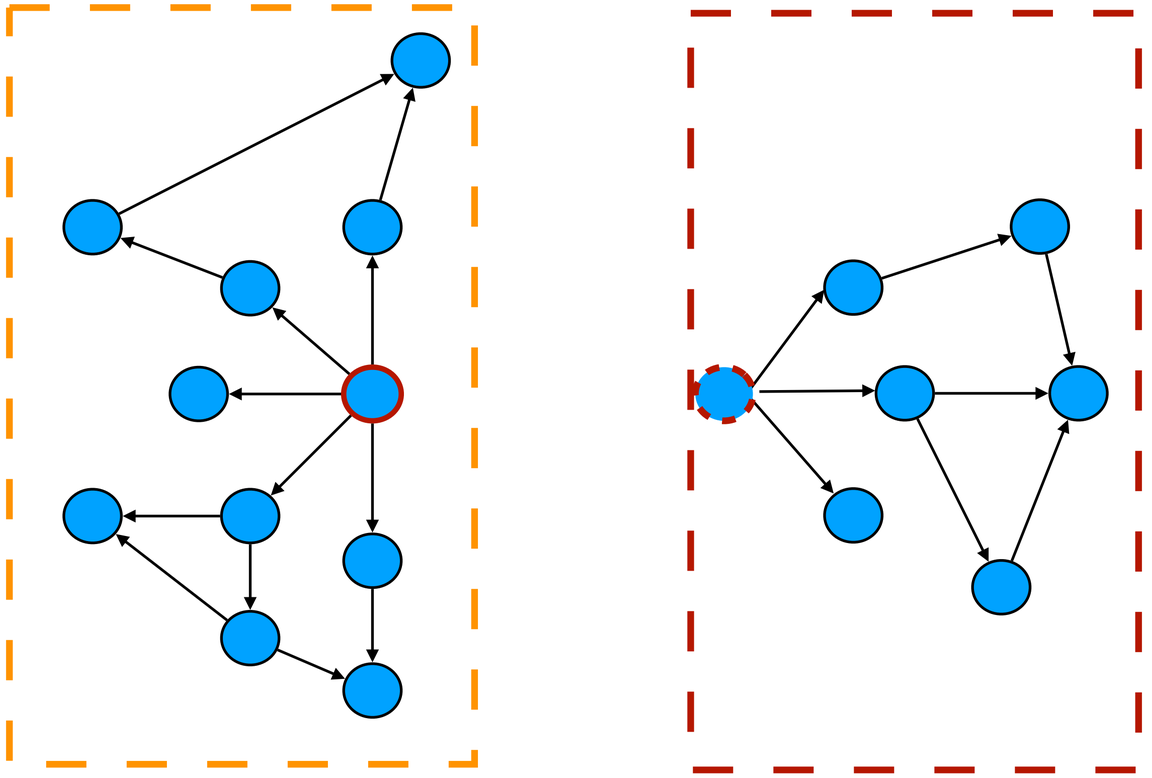
Partitioning



Partitioning



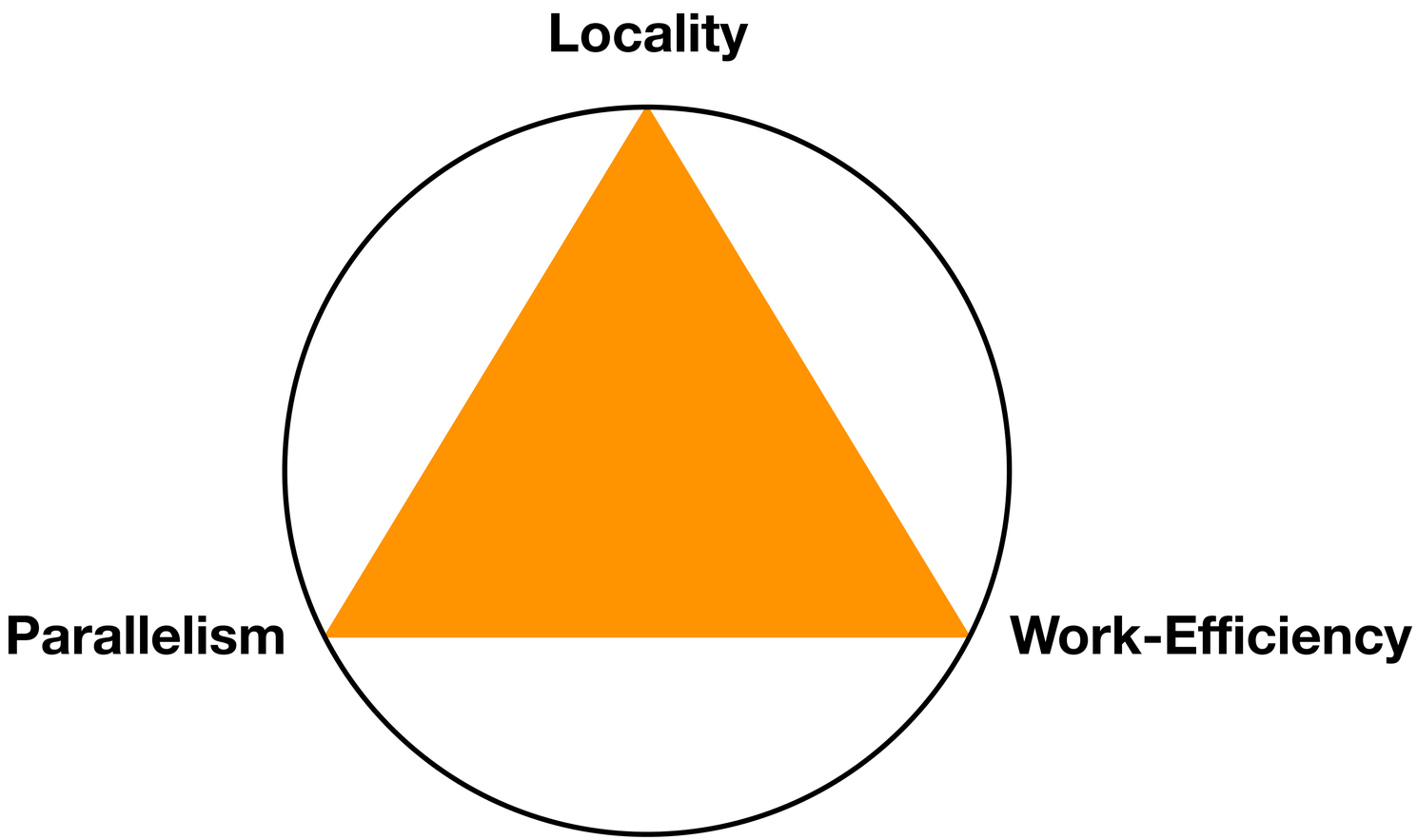
Partitioning



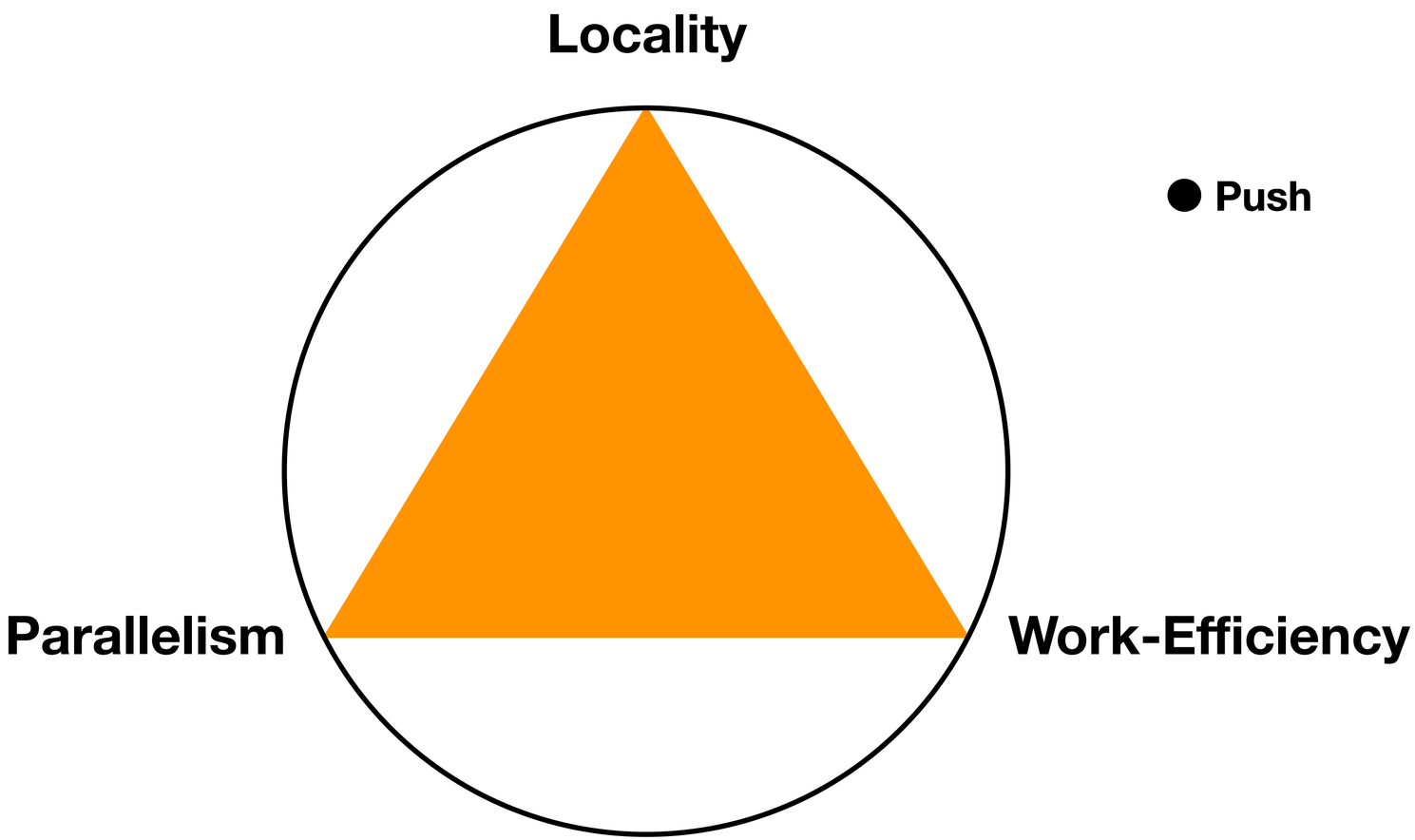
Improves locality

Needs extra instructions to traverse two graphs

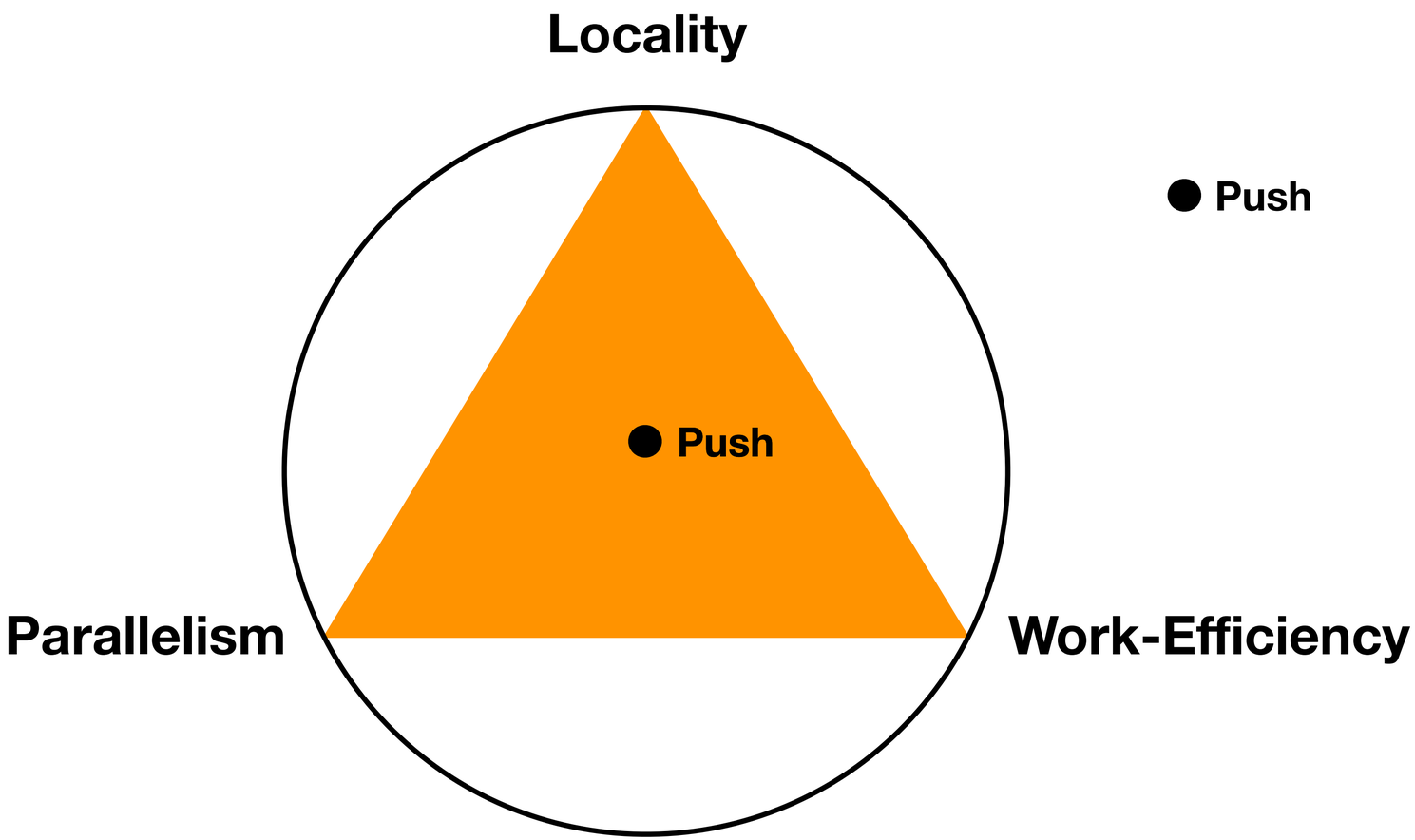
Optimization Tradeoff Space



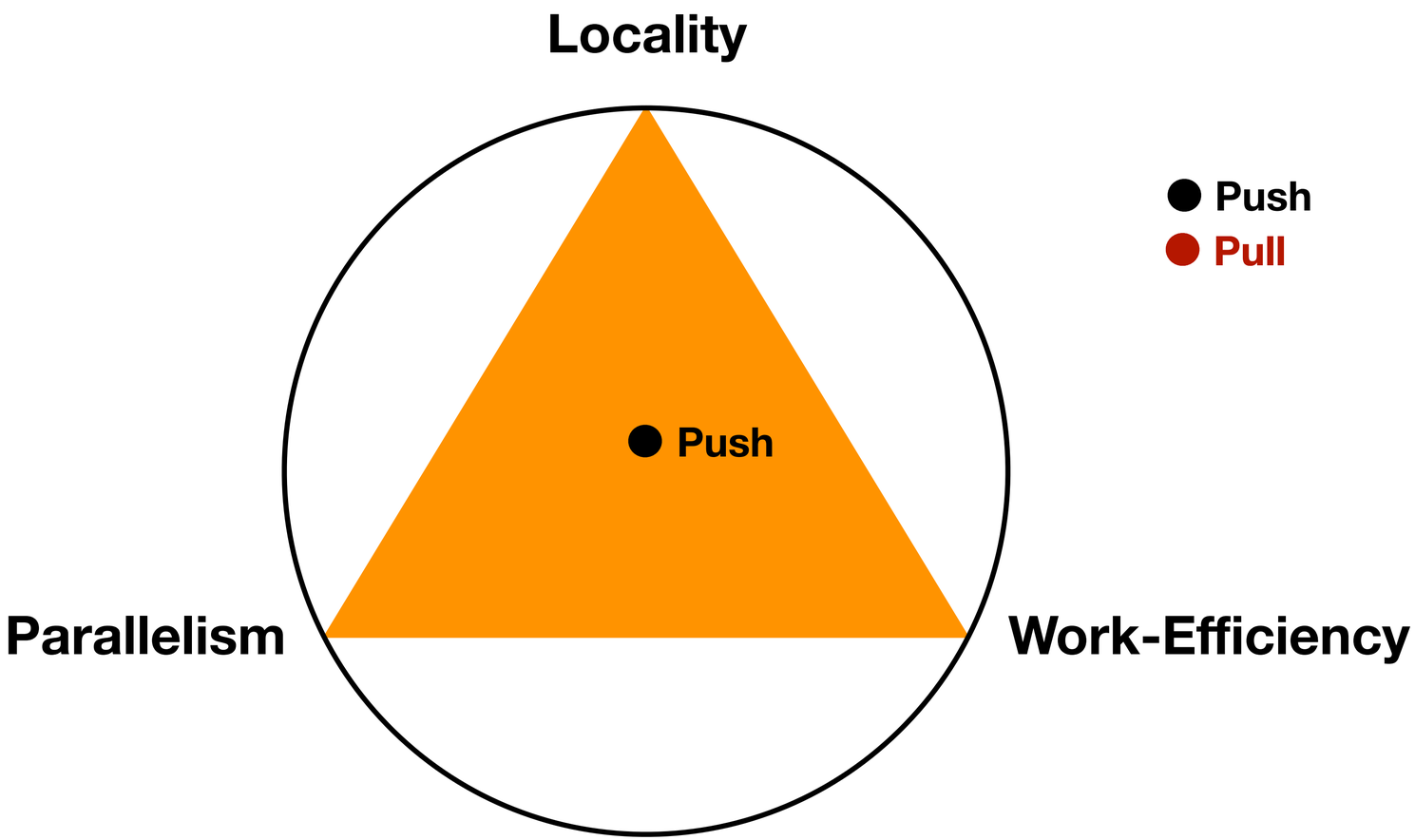
Optimization Tradeoff Space



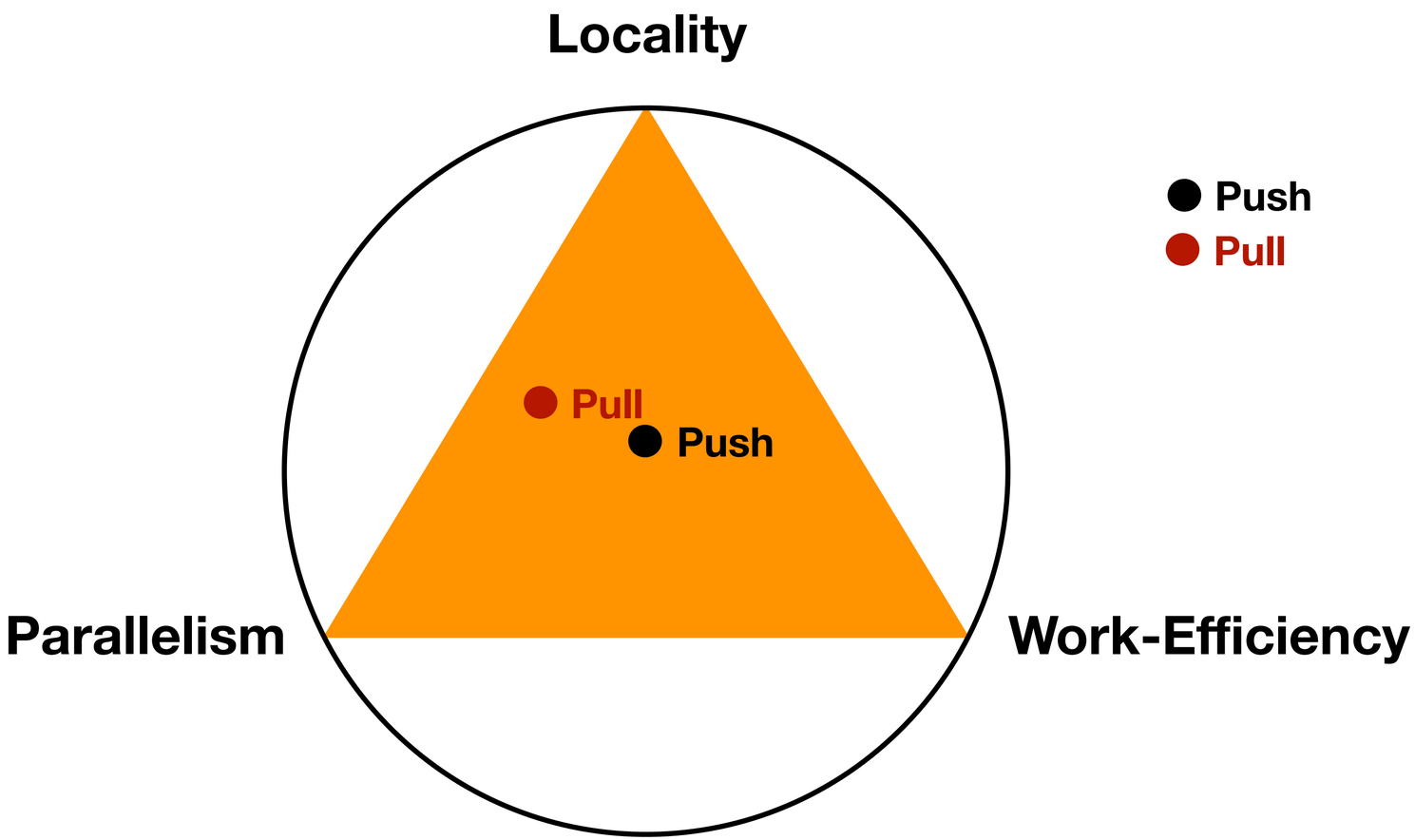
Optimization Tradeoff Space



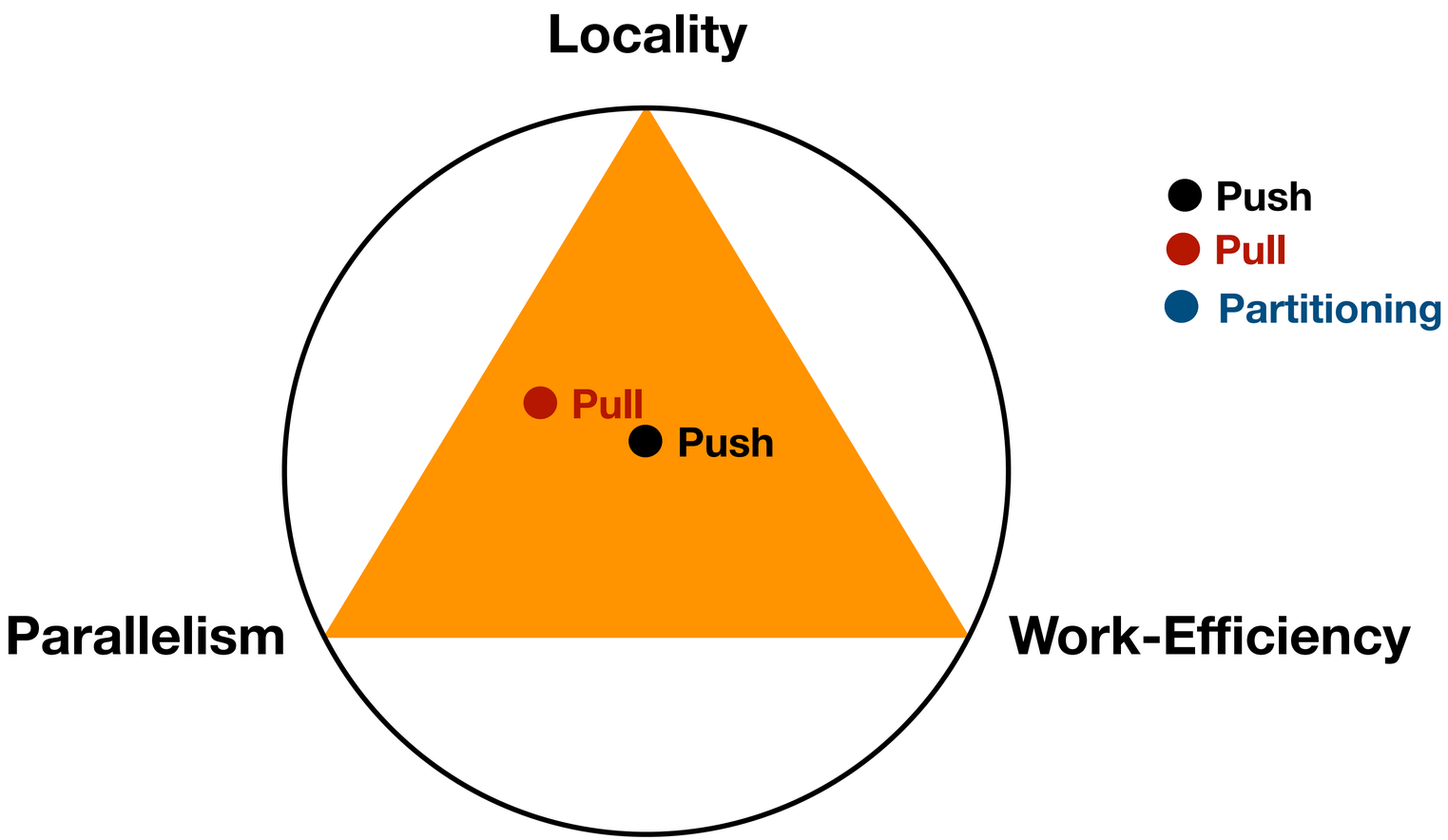
Optimization Tradeoff Space



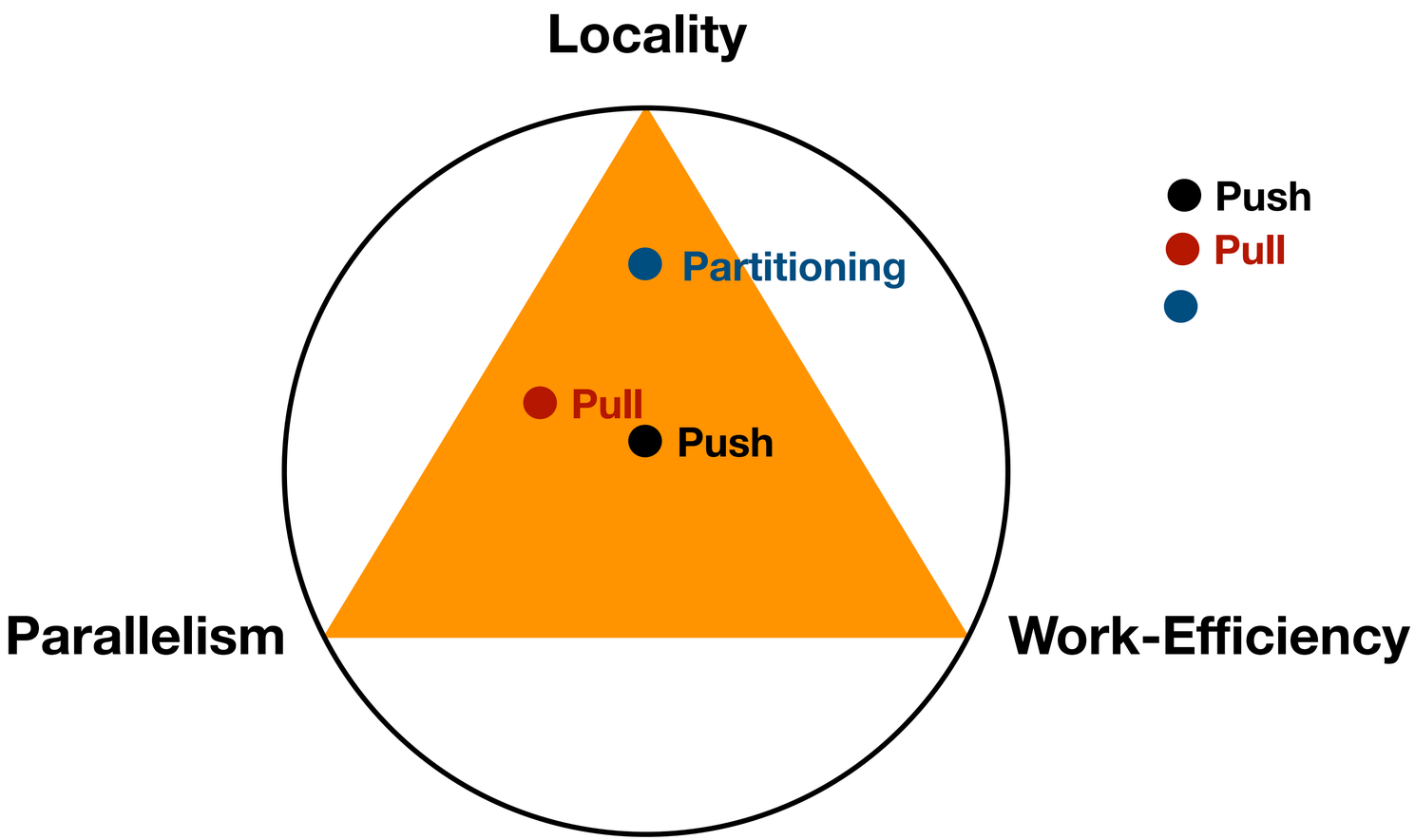
Optimization Tradeoff Space



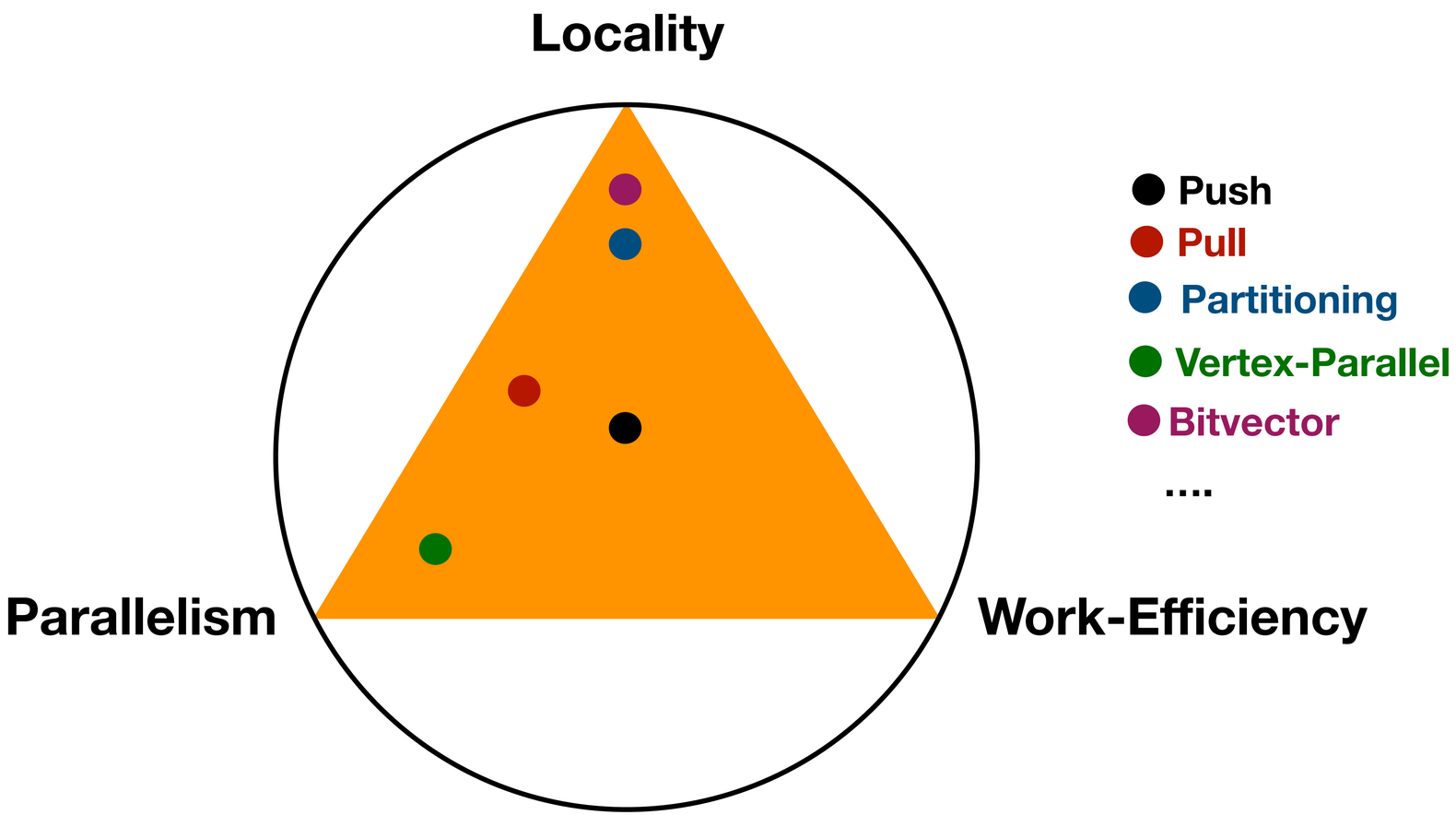
Optimization Tradeoff Space

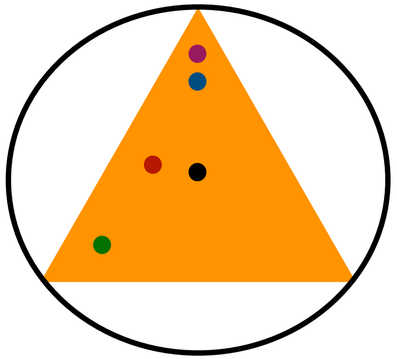


Optimization Tradeoff Space

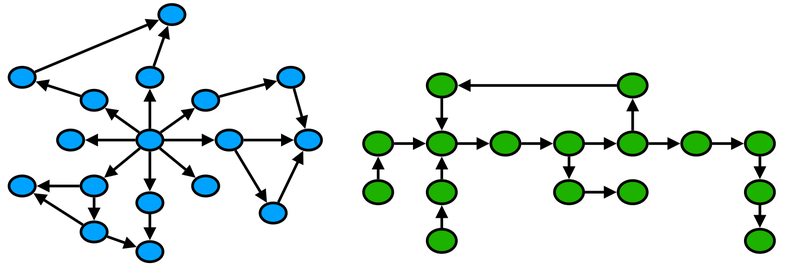


Optimization Tradeoff Space

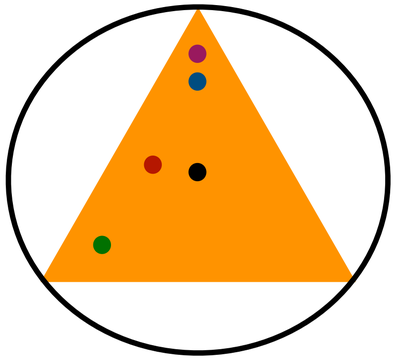




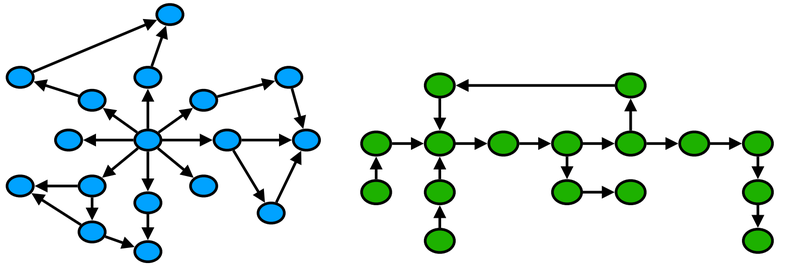
Optimizations



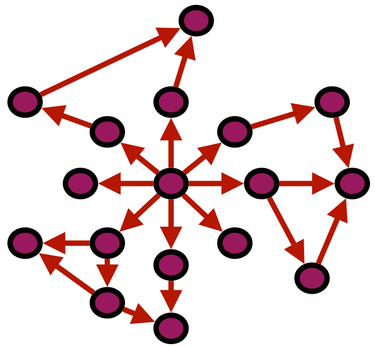
Graphs



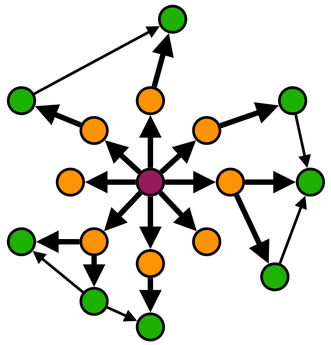
Optimizations

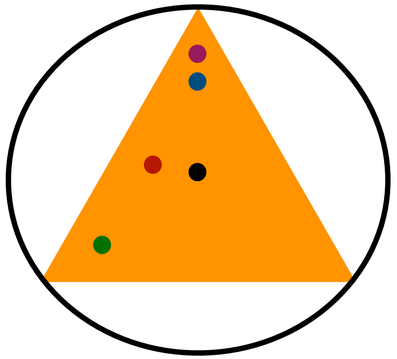


Graphs

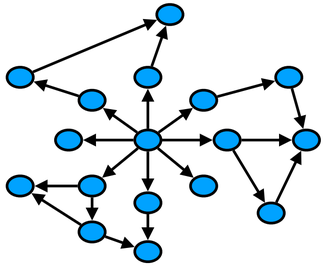


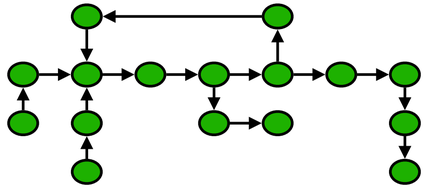
Algorithms



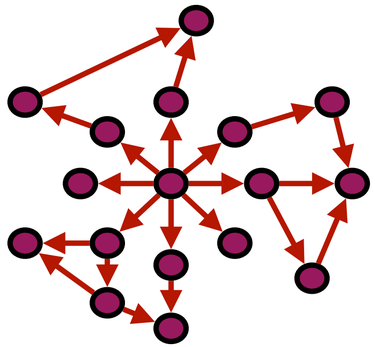


Optimizations

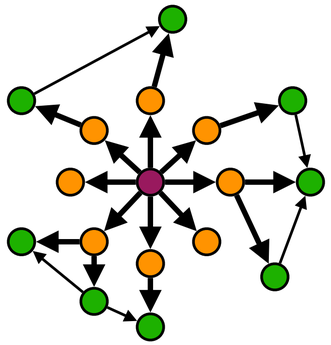


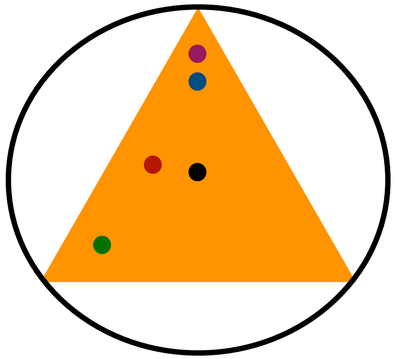


Graphs



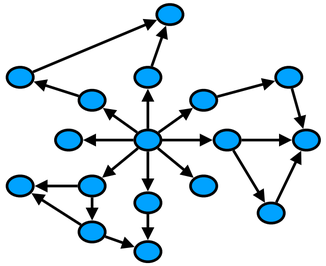
Algorithms



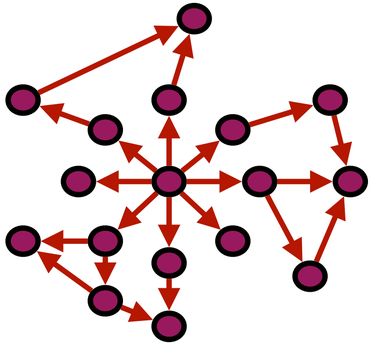


Optimizations

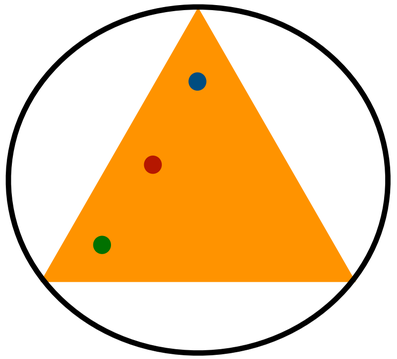
Hardware



Graphs



Algorithms

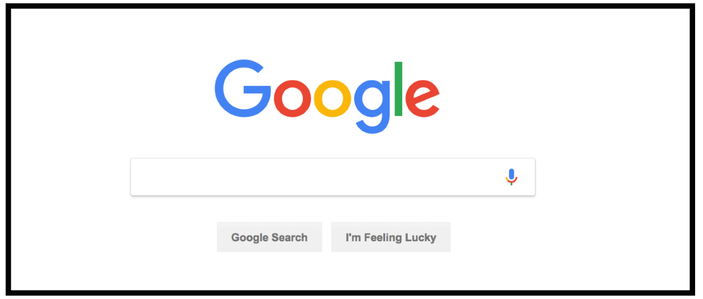


Optimizations

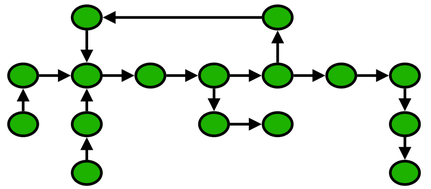
Pull

Partitioning

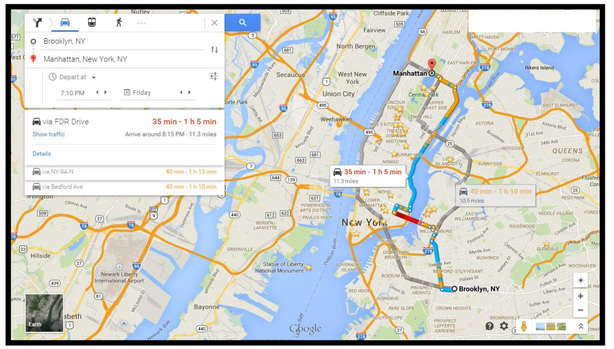
Vertex-Parallel



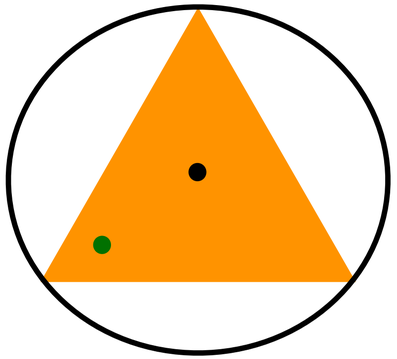
Hardware



Graphs

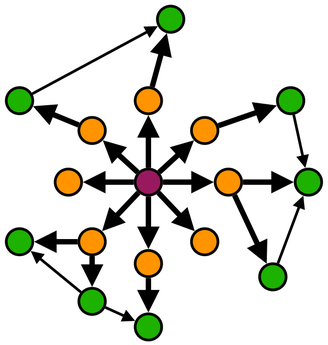


Algorithms

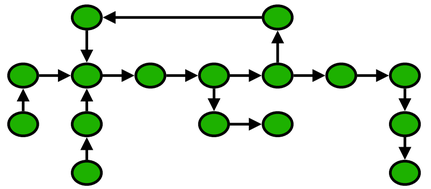


Optimizations

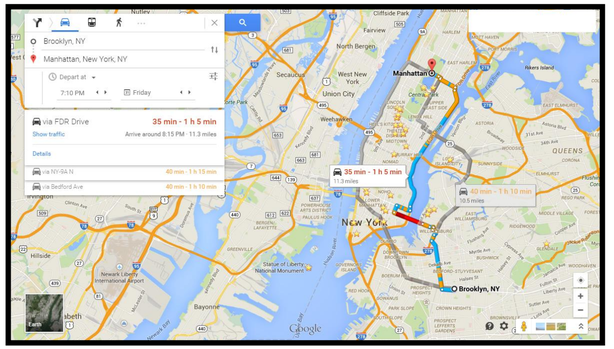
Hardware



Push
Vertex-Parallel

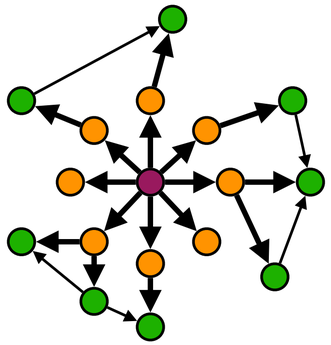


Graphs

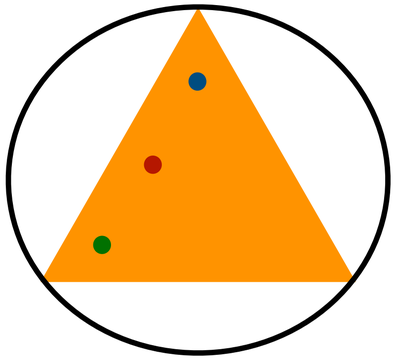


Hardware

Algorithms



Bad optimizations (schedules) can be > 100x slower



Optimizations

Pull

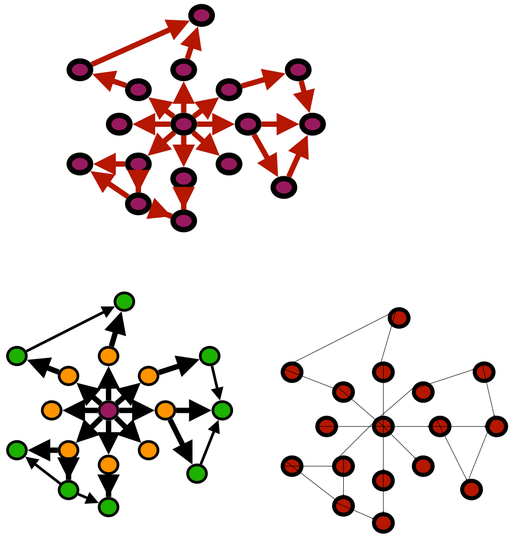
Partitioning

Vertex-Parallel

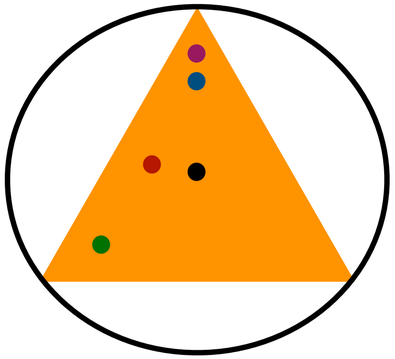
Outline

- Graph Applications Overview
- Optimization Tradeoff Space
- GraphIt DSL
- Evaluation

GraphIt DSL

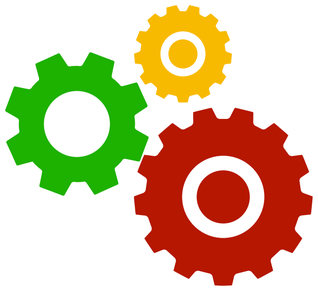


Algorithm Representation (Algorithm Language)



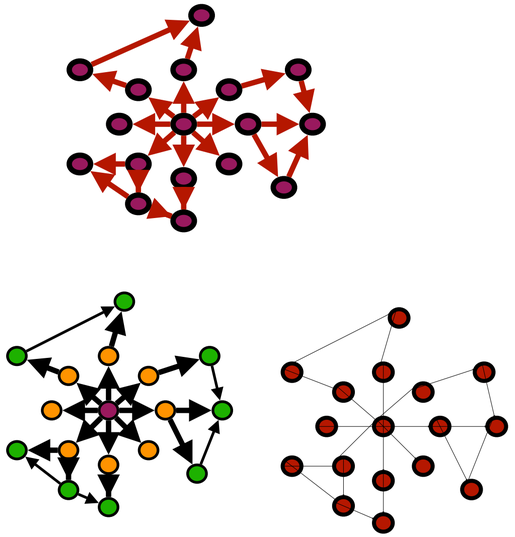
Optimization Representation

- **Scheduling Language**
- **Schedule Representation (e.g. Graph Iteration Space)**



Autotuner

GraphIt DSL



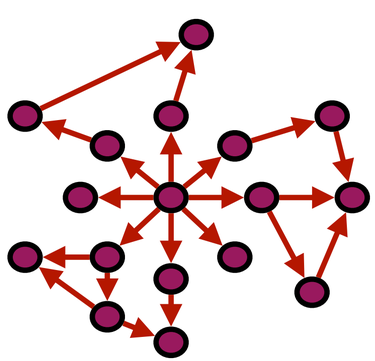
Algorithm Representation
(Algorithm Language)

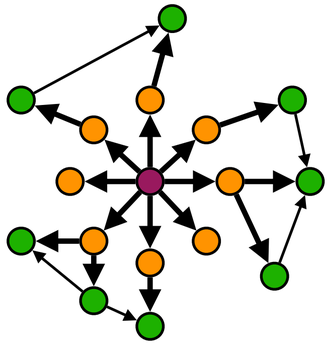
Optimization Representation

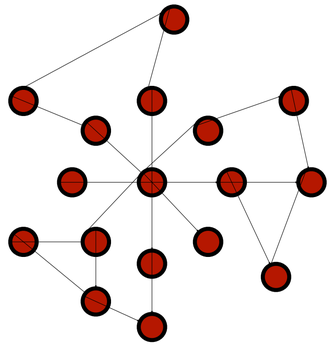
- **Scheduling Language**
- **Schedule Representation**
  (e.g. Graph Iteration Space)

Autotuner

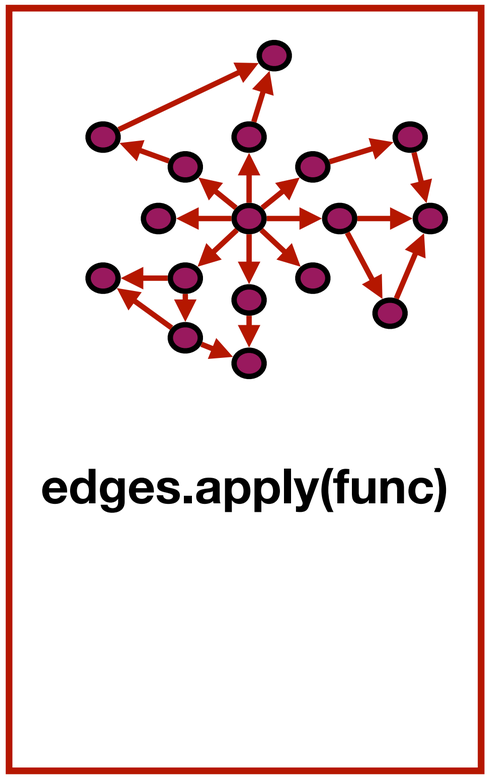
Algorithm Language

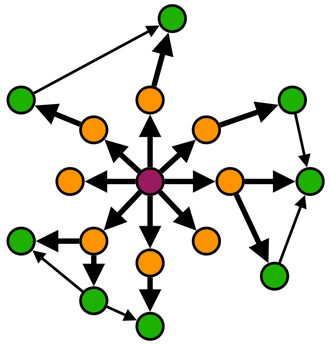


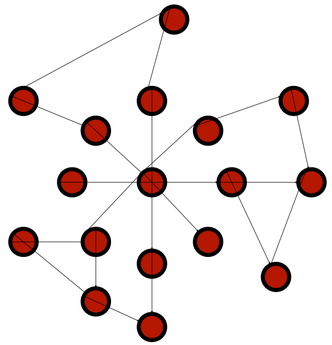




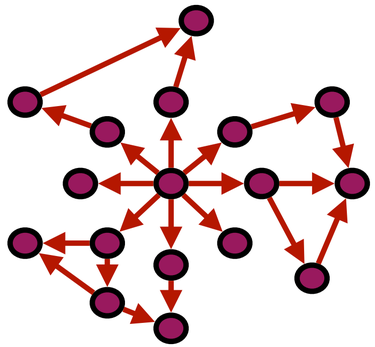
Algorithm Language



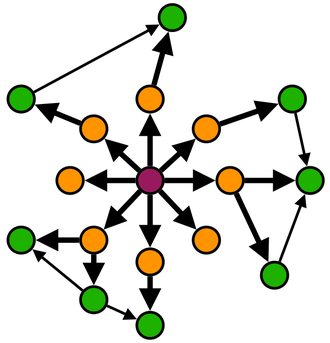




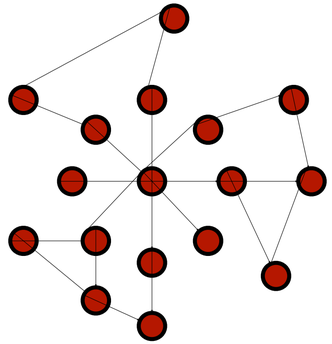
Algorithm Language



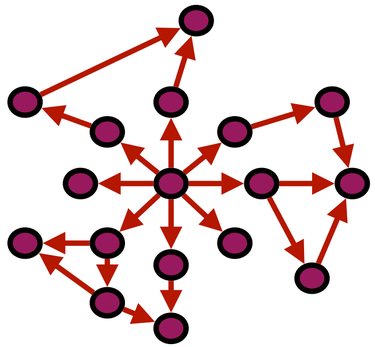
edges.apply(func)



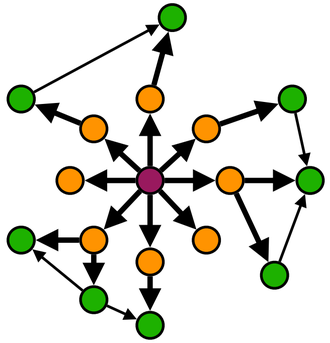
edges.from(vertexset)
.to(vertexset)
.srcFilter(func)
.dstFilter(func)
.apply(func)



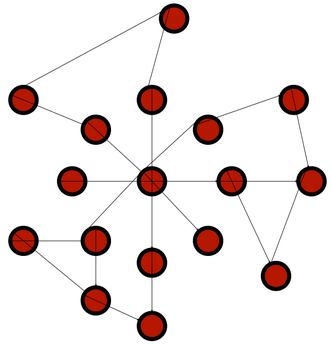
Algorithm Language



edges.apply(func)

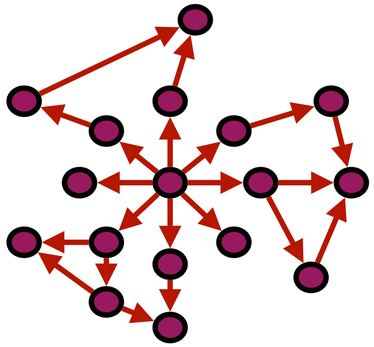


edges.from(vertexset)
.to(vertexset)
.srcFilter(func)
.dstFilter(func)
.apply(func)



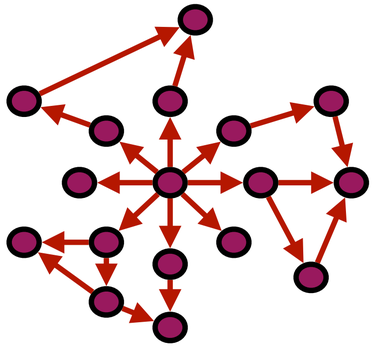
vertices.apply(func)

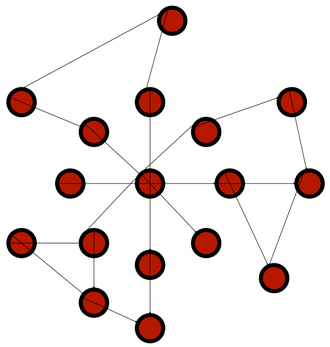
PageRank Example



```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end
```

PageRank Example

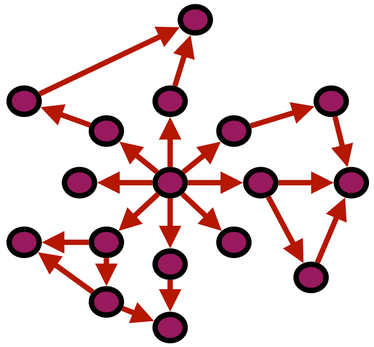


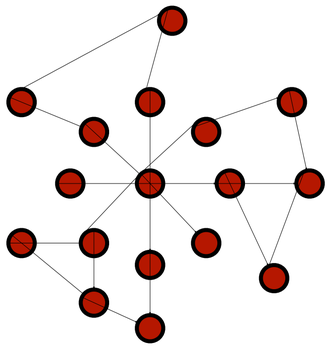


```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end
```

PageRank Example



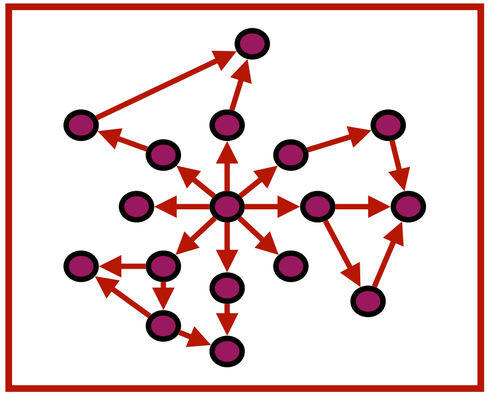


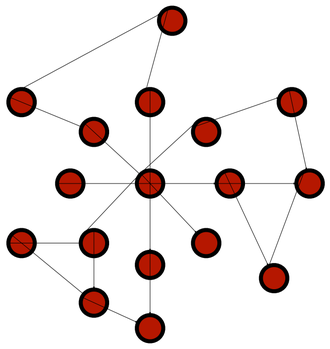
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```

PageRank Example



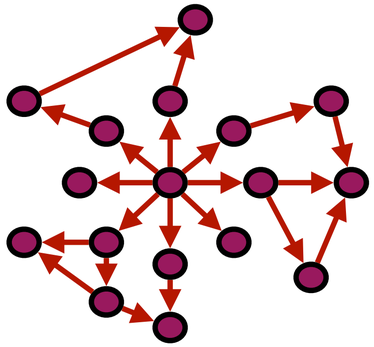


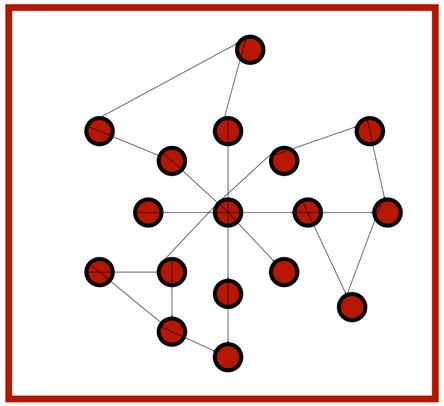
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```

PageRank Example





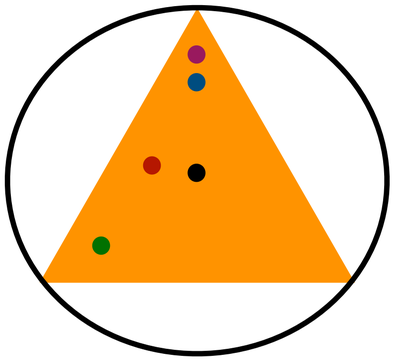
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end
```

```
func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end
```

```
func main()
    for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
end
```

GraphIt DSL

Algorithm Representation
(Algorithm Language)

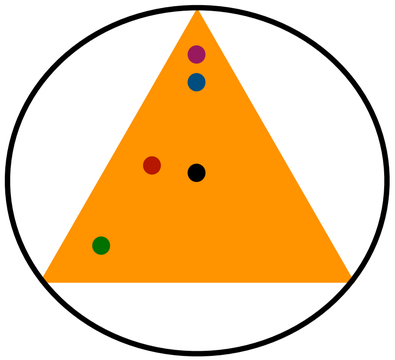


Optimization Representation

- **Scheduling Language**
- **Schedule Representation (e.g. Graph Iteration Space)**

Autotuner

GraphIt DSL



Algorithm Representation
(Algorithm Language)

Optimization Representation

- **Scheduling Language**
- **Schedule Representation (e.g. Graph Iteration Space)**

Autotuner

Scheduling Language

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```

Scheduling Language

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```

Scheduling Language

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

func main()
    for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     new_rank[dst]  +=  old_rank[src] / out_degree[src];
   }
}

….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
```

Schedule 2

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

func main()
    for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
      atomic_add (new_rank[dst],
                        old_rank[src] / out_degree[src] );
   }
}

….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```

Schedule 3

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID dst : vertices) {
  for(NodeID src : G.getInNgh(dst)){
     new_rank[dst]  += old_rank[src] / out_degree[src];
   }
}

….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```

Schedule 4

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Pseudo Generated Code

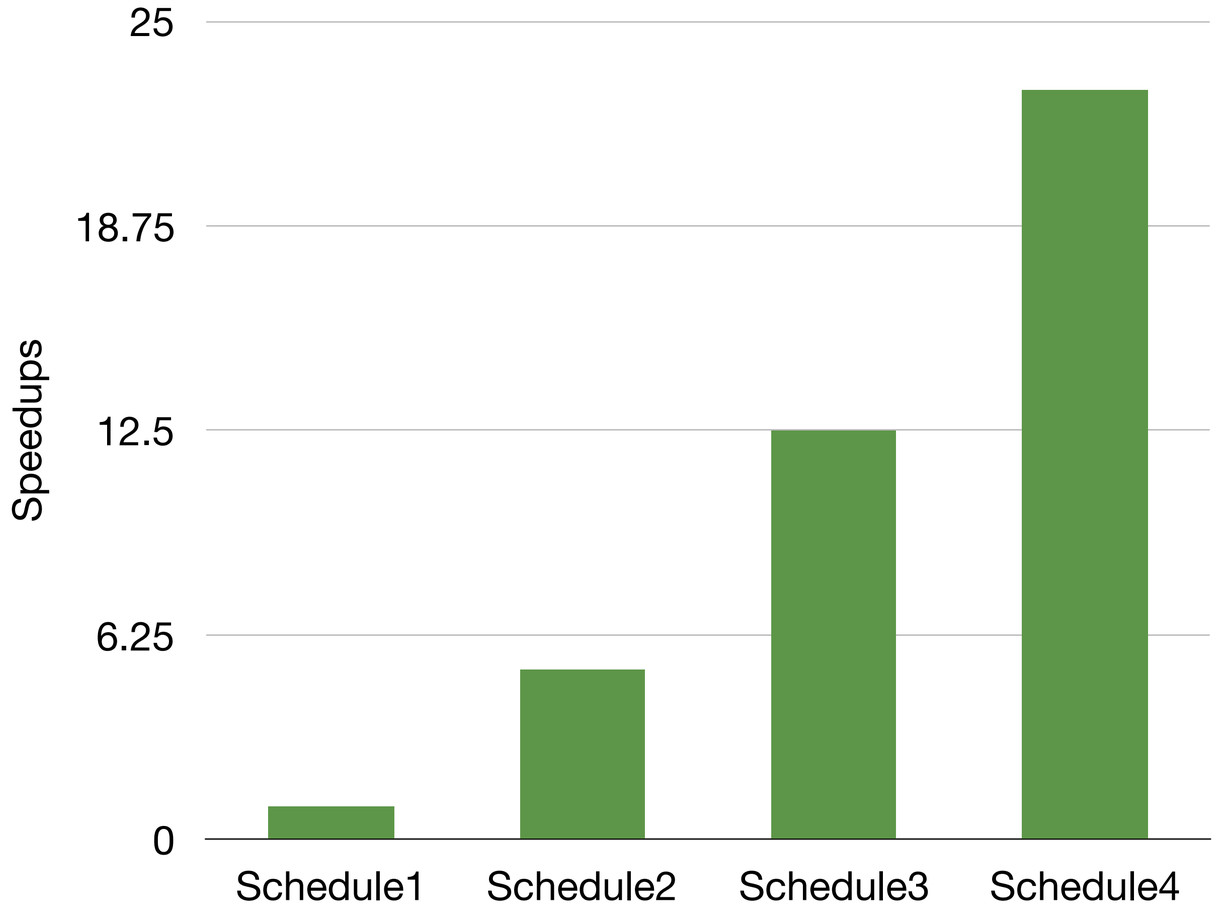
```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];


…
for (Subgraph sg : G.subgraphs) {
  parallel_for (NodeID dst : verticesa) {
    for(NodeID src : G.getInNgh(dst)){
     new_rank[dst]  += old_rank[src] / out_degree[src];
    }
  }
}
….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
    program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Speedups of Schedules



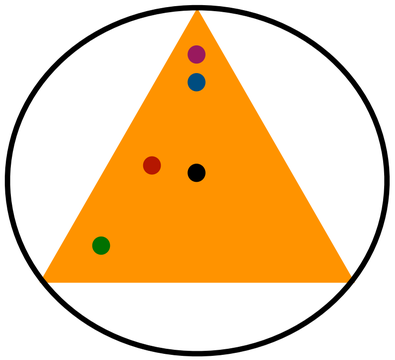
Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Many More Optimizations

- Direction optimizations (configApplyDirection),
  - SparsePush, DensePush, DensePull, DensePull-SparsePush, DensePush-SparsePush
- Parallelization strategies (configApplyParallelization)
  - serial, dynamic-vertex-parallel, static-vertexparallel, edge-aware-dynamic-vertex-parallel, edge-parallel
- Cache (configApplyNumSSG)
  - fixed-vertex-count, edge-aware-vertexcount
- NUMA (configApplyNUMA)
  - serial, static-parallel, dynamic-parallel
- AoS, SoA (fuseFields)
- Vertexset data layout (configApplyDenseVertexSet)
  - bitvector, boolean array

GraphIt DSL

Algorithm Representation
(Algorithm Language)



Optimization Representation

- **Scheduling Language**
- **Schedule Representation (e.g. Graph Iteration Space)**

Autotuner

Schedule Representation

Algorithm Specification

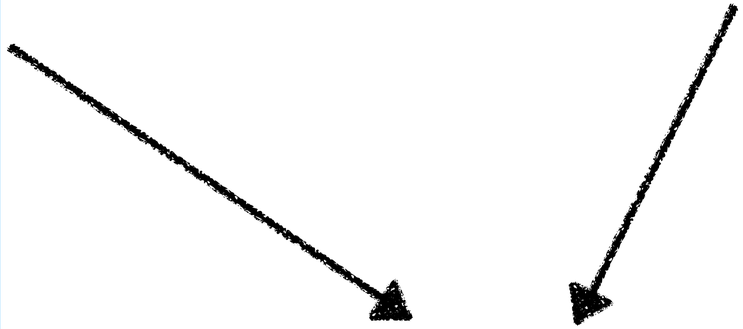
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```



GraphIt Compiler

Schedule Representation

Algorithm Specification

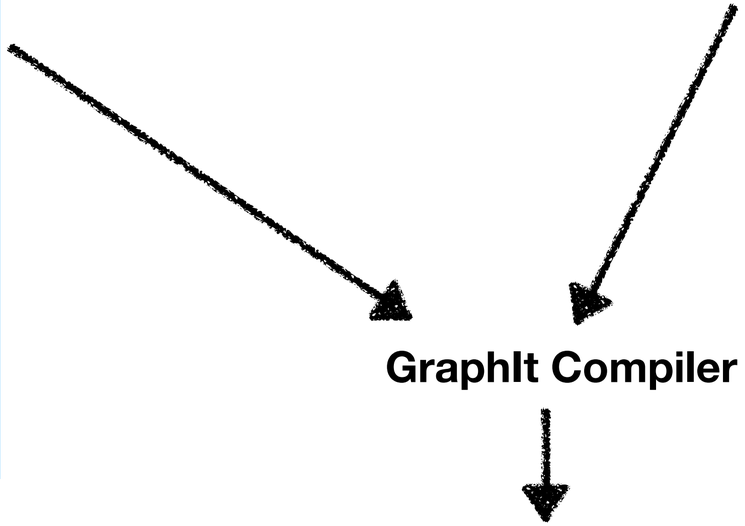
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```



Pseudo Generated Code

```
…
parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     new_rank[dst]  = atomic_add ( new_rank[dst] ,
           old_rank[src] / out_degree[src] );
   }
}
….
```

Schedule Representation

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```

Internal Schedule Representation

Pseudo Generated Code

```
…
parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     new_rank[dst]  = atomic_add ( new_rank[dst] ,
           old_rank[src] / out_degree[src] );
   }
}
….
```

Schedule Representation

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

 func main()
    for i in 1:11
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
    end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```

Internal Schedule Representation

Reason about Optimization Composition, Validity, and Code Generation

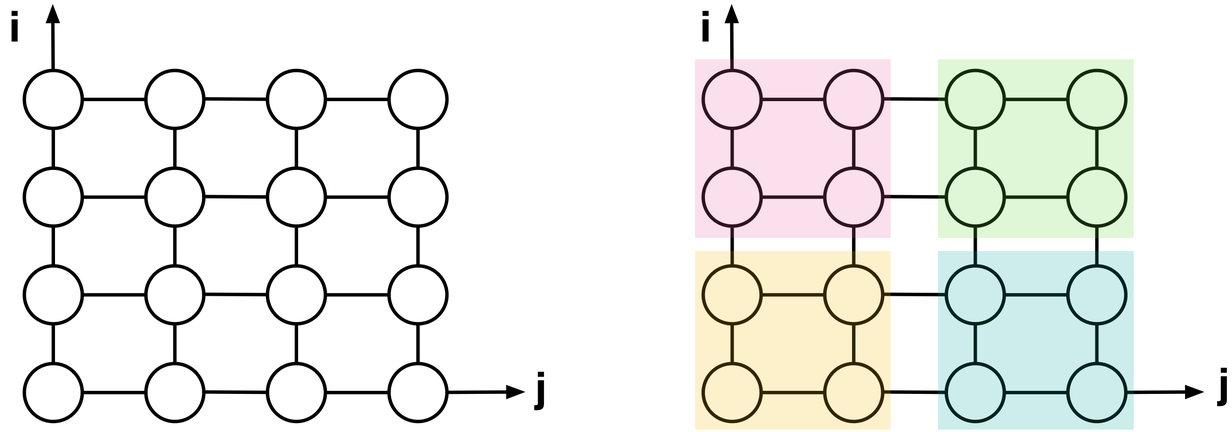
Pseudo Generated Code

```
...
parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     new_rank[dst]  = atomic_add ( new_rank[dst] ,
           old_rank[src] / out_degree[src] );
   }
}
....
```

Iteration Spaces

- Very versatile representation of dense loops and arrays
- Used for:
  - Program analyses
  - Composition of complex loop transformations
  - Framework for code generation

Iteration Space Representation



Iteration Space Vectors

<i, j>

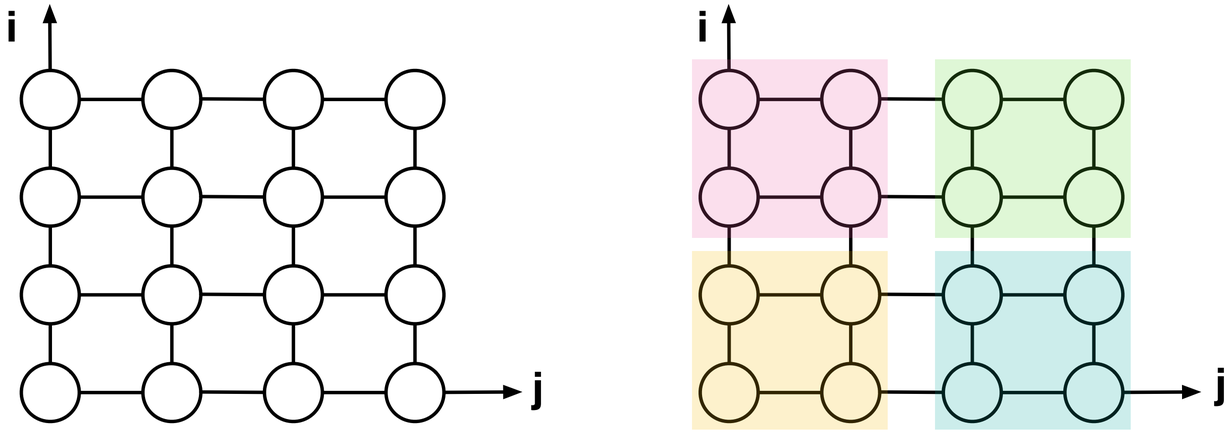
<i_1, j_1, i_2, j_2>

Iteration Spaces

- Very versatile representation of dense loops and arrays
- Used for:
  - Program analyses
  - Composition of complex loop transformations
  - Framework for code generation

We extend it to <u>sparse</u> loops

Iteration Space Representation



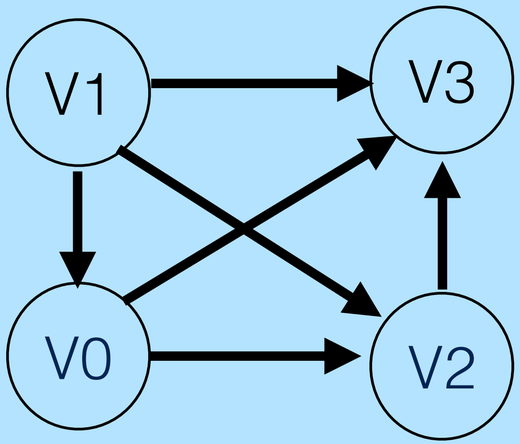
Iteration Space Vectors

<i, j>

<i_1, j_1, i_2, j_2>

Graph Iteration Space

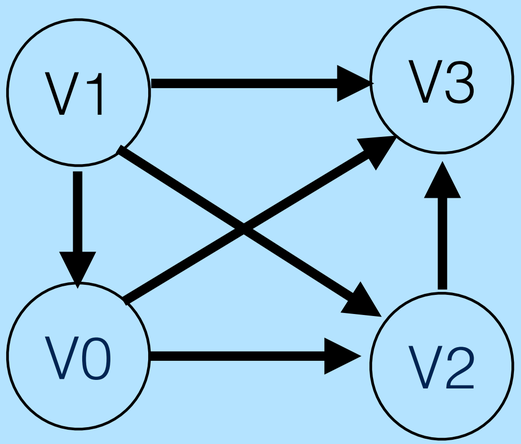
Edges.apply(F);



Input Graph

Graph Iteration Space

Edges.apply(F);



Input Graph

InnerIter [src]

OuterIter [dst]

	V0	V1	V2	V3
v0	0	1	0	0
v1	0	0	0	0
v2	1	1	0	0
v3	1	1	1	0

Adjacency Matrix

Dense Iteration Space

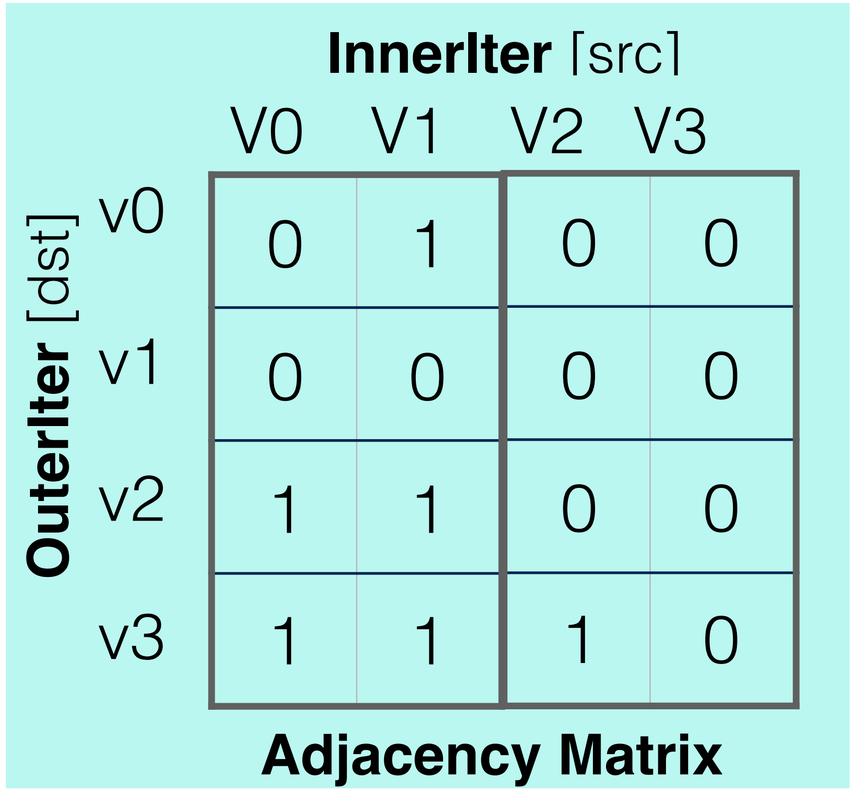
Edges.apply(F);

Loops

```
for OuterIter in [v0-v3] do
    for InnerIter in [v0-v3] do
        If edge (OuterIter, InnerIter) exists
            F(OuterIter, InnerIter)
    done
done
```

Iteration Space Vector

< OuterIter, InnerIter >



Dense Iteration Space

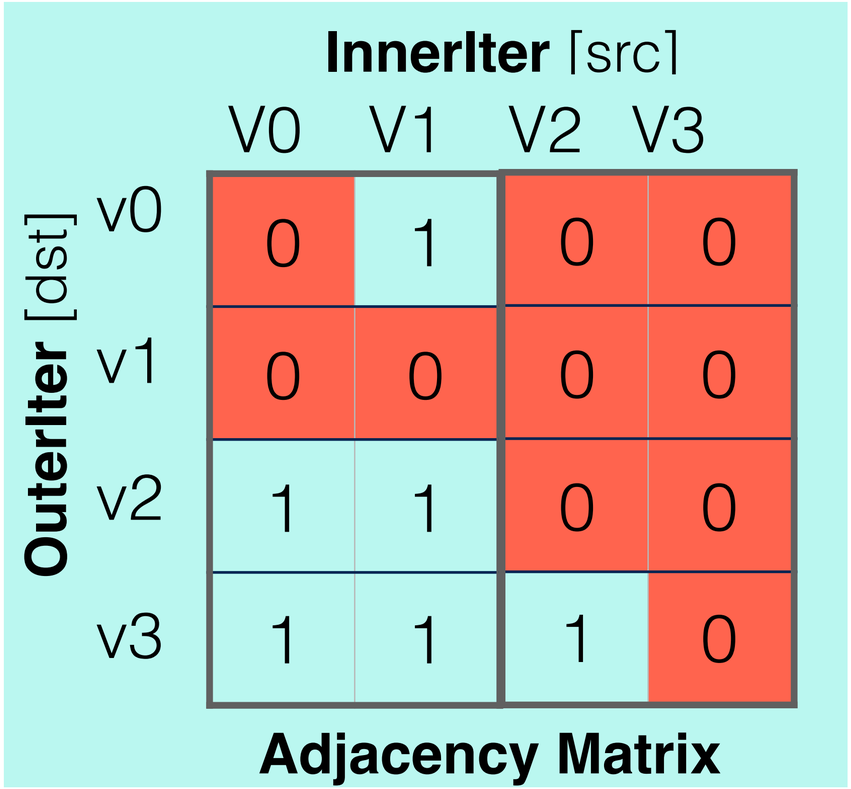
Edges.apply(F);

Loops

```
for OuterIter in [v0-v3] do
    for InnerIter in [v0-v3] do
        If edge (OuterIter, InnerIter) exists
            F(OuterIter, InnerIter)
    done
done
```

Iteration Space Vector

< OuterIter, InnerIter >



Graph Iteration Space

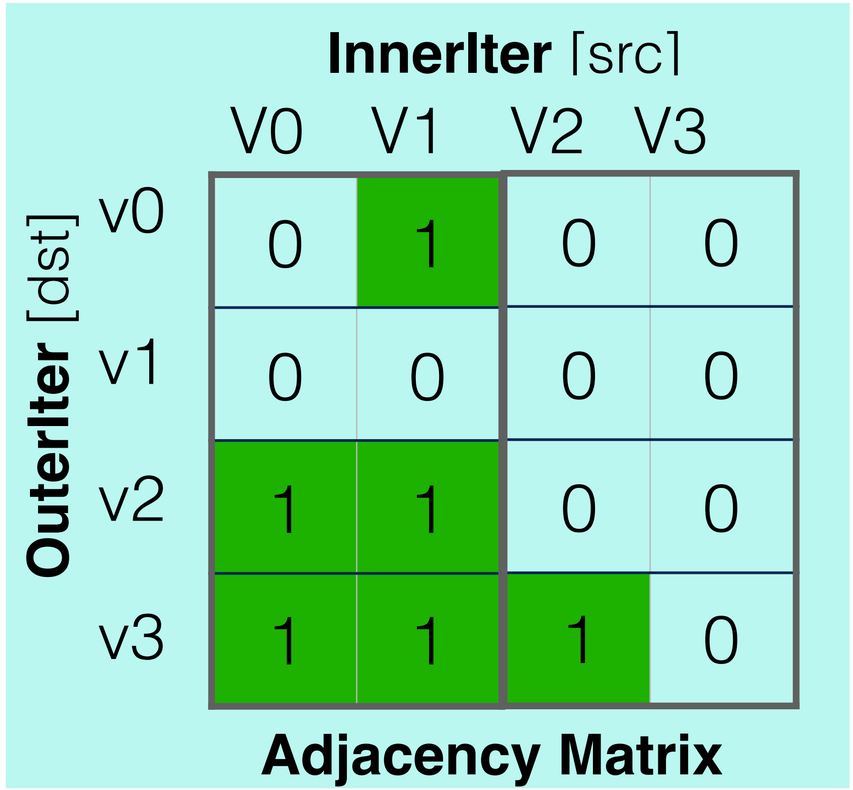
Edges.apply(F);

Loops

```
for OuterIter [dst] in vertices:
    for InnerIter [src] in in_ngh(OuterIter):
        F (OuterIter,InnerIter);
```

Iteration Space Vector

< OuterIter[dst], InnerIter[src] >

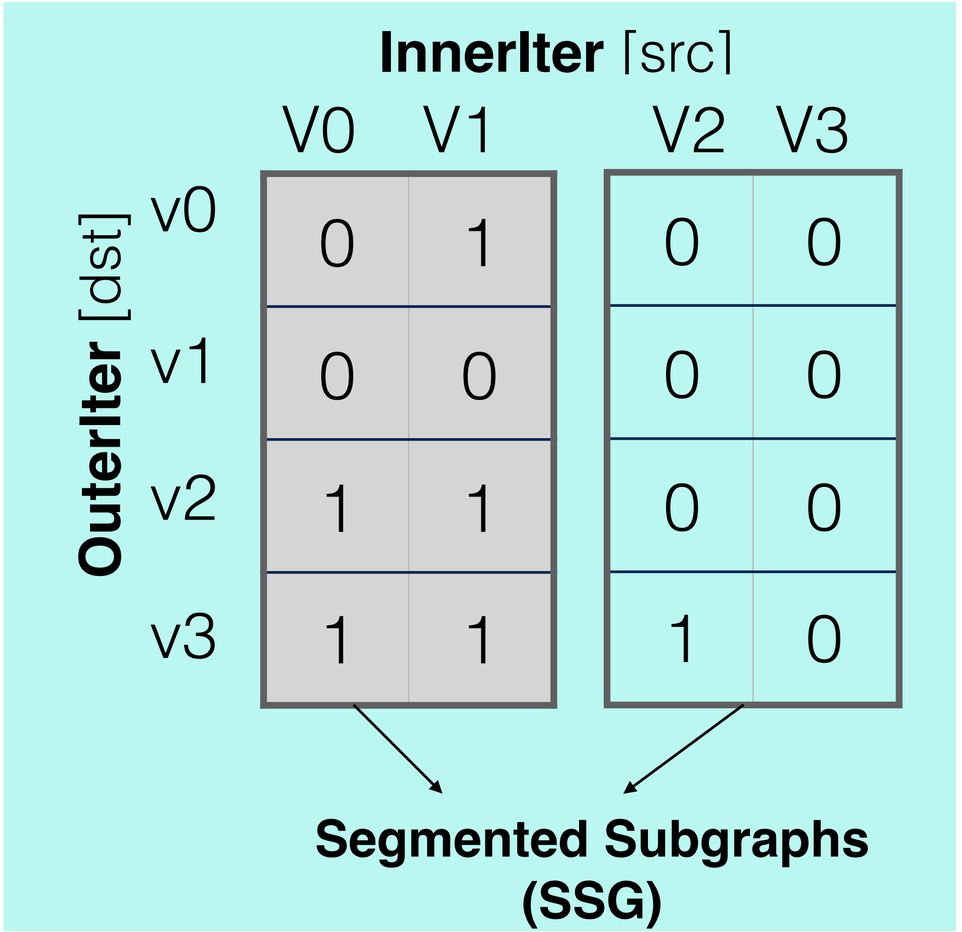


Graph Iteration Space

Edges.apply(F);

Loops

Iteration Space Vector

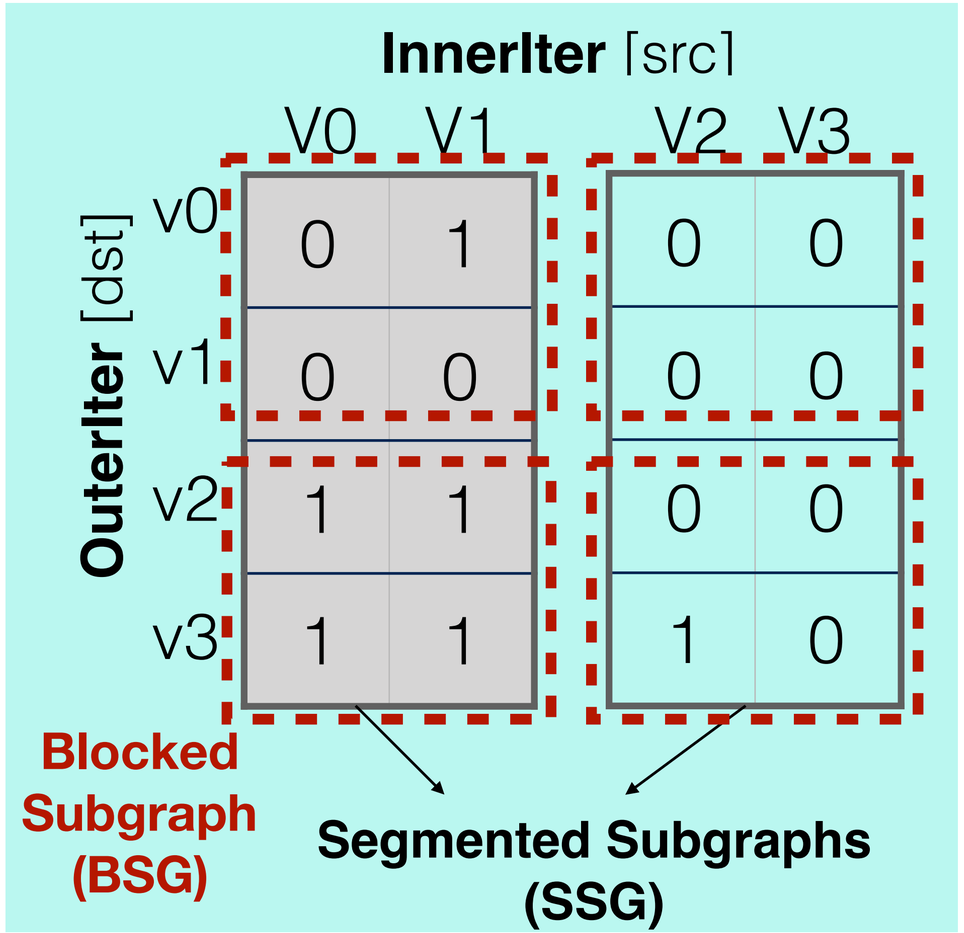


Graph Iteration Space

Edges.apply(F);

Loops

Iteration Space Vector



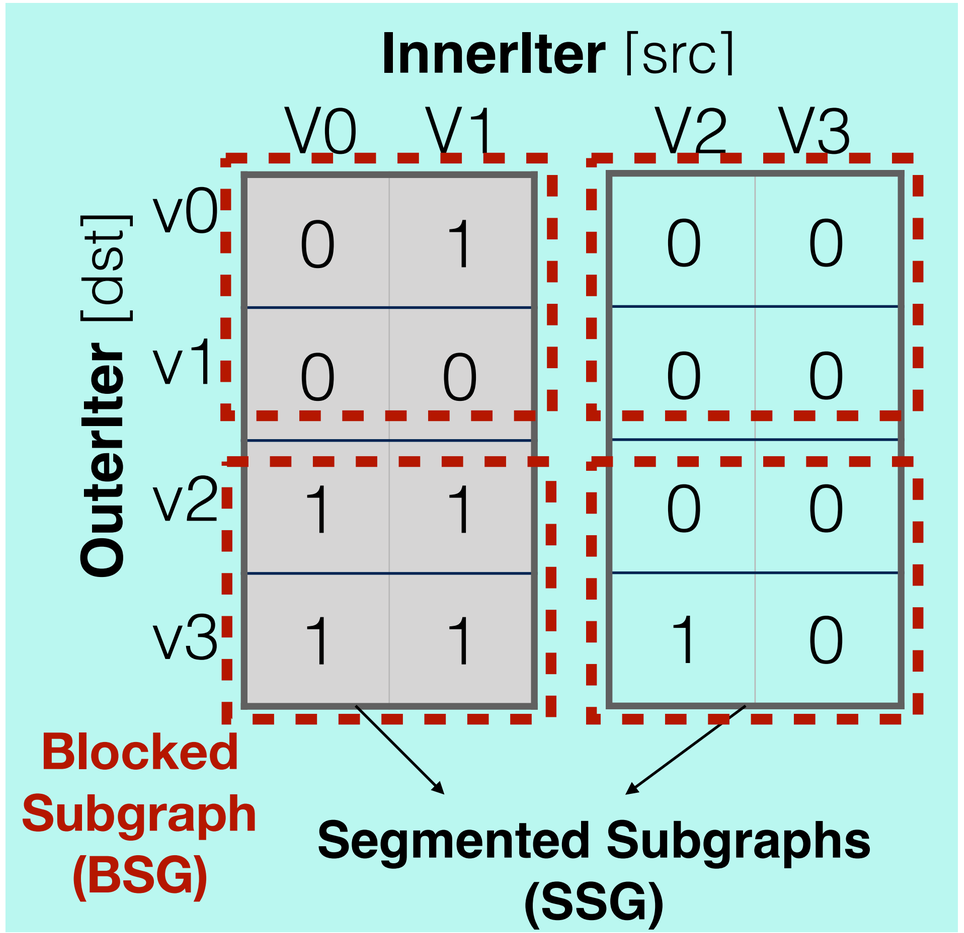
Graph Iteration Space

Edges.apply(F);

Loops

```
for SSG in partitioned SSGs:
    for BSG in SSG:
        for OuterIter [dst] in BSG:
            for InnerIter [src] in_ngh(OuterIter):
                F (OuterIter,InnerIter);
```

Iteration Space Vector



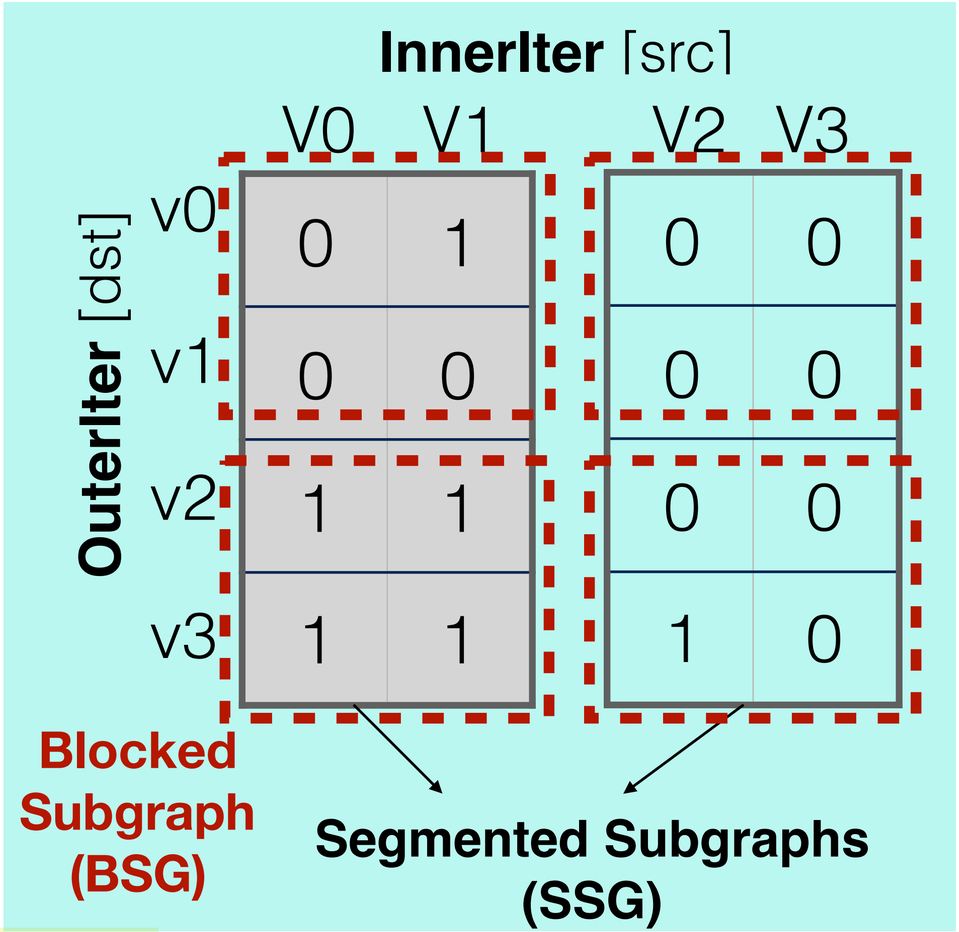
Graph Iteration Space

Edges.apply(F);

Loops

```
for SSG in partitioned SSGs:
    for BSG in SSG:
        for OuterIter [dst] in BSG:
            for InnerIter [src] in_ngh(OuterIter):
                F (OuterIter,InnerIter);
```

Iteration Space Vector



< **SSG**, **BSG**, OuterIter [dst], InnerIter [src] >

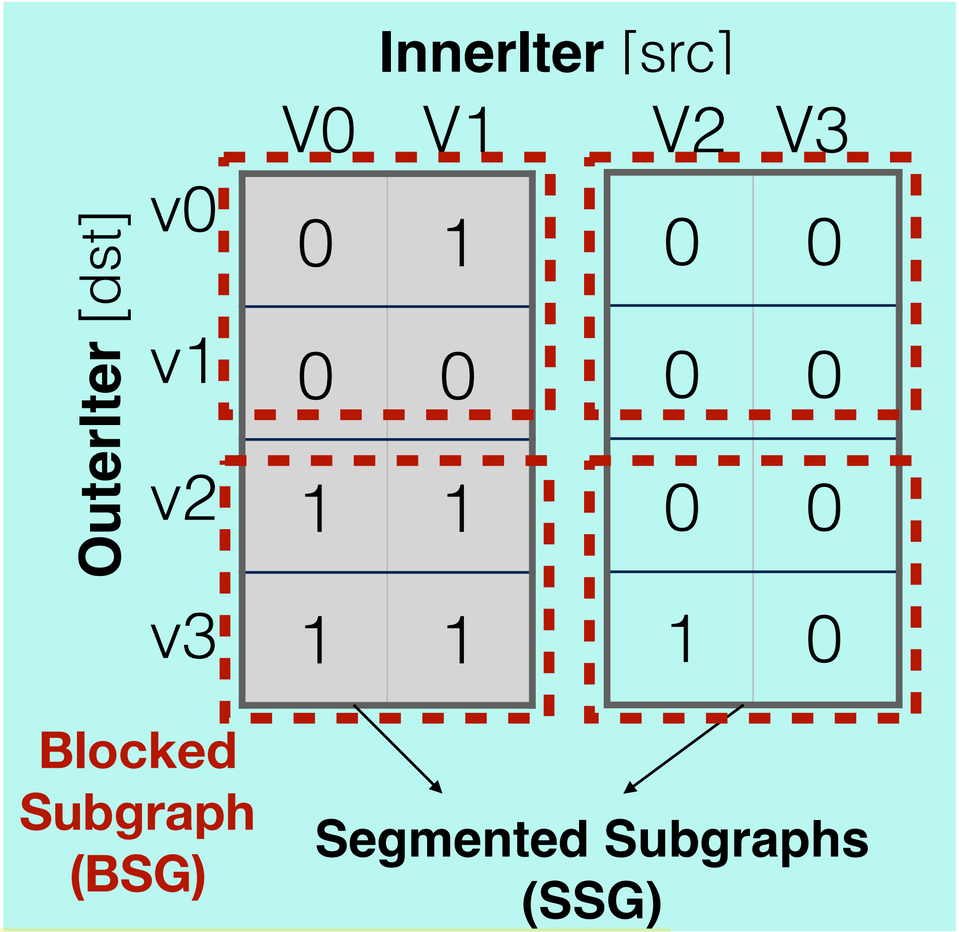
Graph Iteration Space

Edges.apply(F);

Loops

```
for SSG in partitioned SSGs:
    for BSG in SSG:
        for OuterIter [dst] in BSG:
            for InnerIter [src] in_ngh(OuterIter):
                F (OuterIter,InnerIter);
```

Iteration Space Vector



< SSG **[tags]**, BSG **[tags]**, OuterIter [dst, **tags**], InnerIter [src, **tags**] >

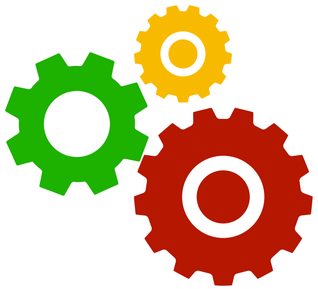
Augmented with Parallelization, Partitioning, Data Layout Tags

GraphIt DSL

Algorithm
Representation
(Algorithm Language)

Optimization Representation

- Scheduling Language
- Schedule Representation
(e.g. Graph Iteration Space)



Autotuner

Schedule 4

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
         #s1# edges.apply(updateEdge);
         vertices.apply(updateVertex);
     end
 end
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
    program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Schedule 4

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

func main()
    for i in 1:11
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
end
```

Finding the best schedule can be hard for non-experts.

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
    program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Goal

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```

Ideally, the user only need to write the algorithm

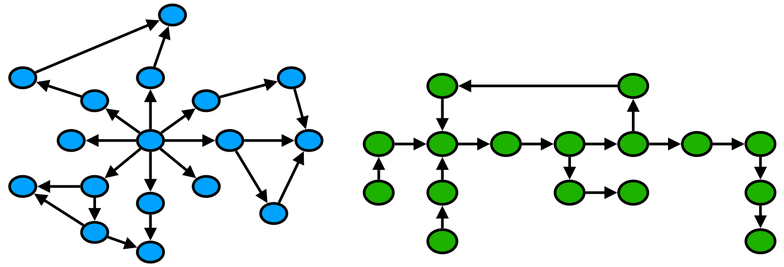
Autotuner

Algorithm Specification

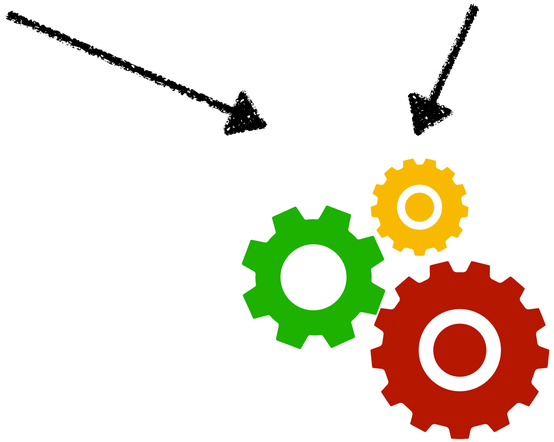
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
         #s1# edges.apply(updateEdge);
         vertices.apply(updateVertex);
     end
 end
```



Input Graphs



Autotuner

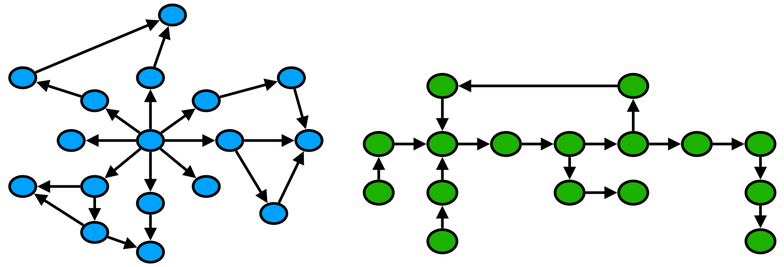
Autotuner

Algorithm Specification

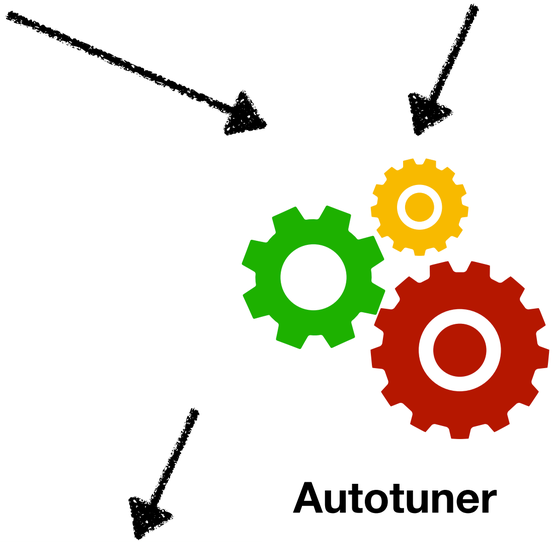
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:11
         #s1# edges.apply(updateEdge);
         vertices.apply(updateVertex);
     end
 end
```



Input Graphs

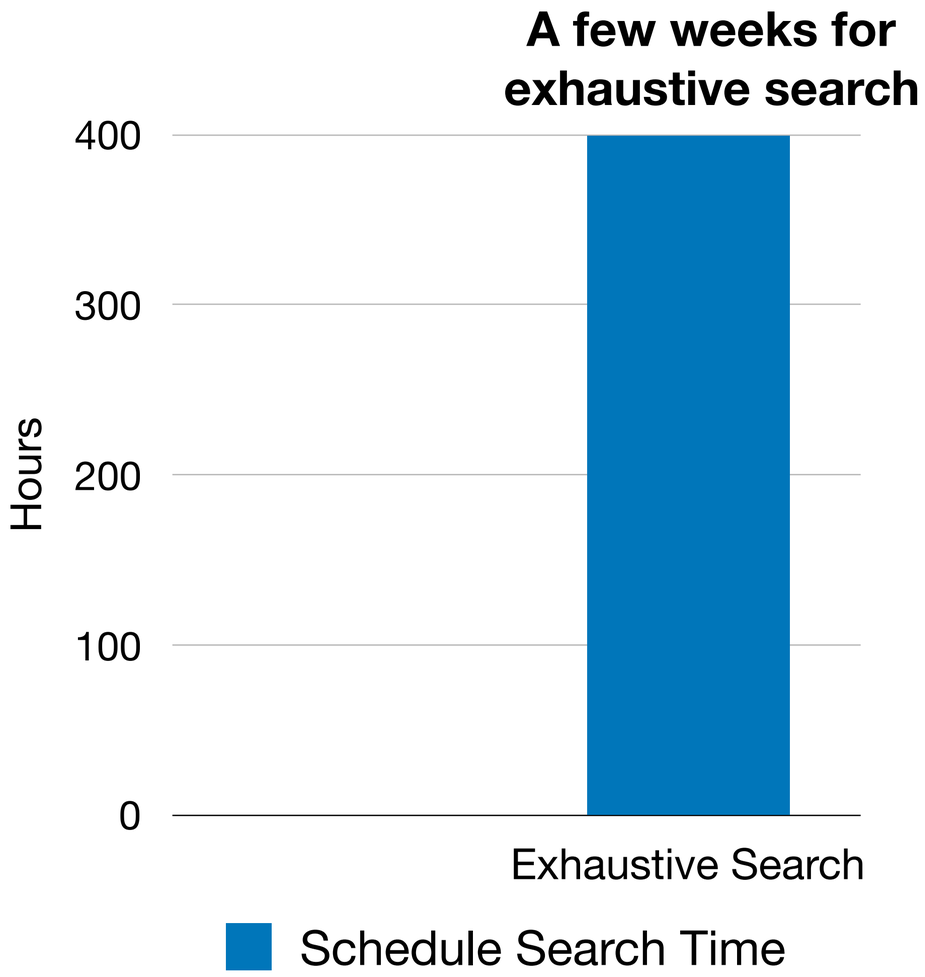


Autotuner

Scheduling Functions

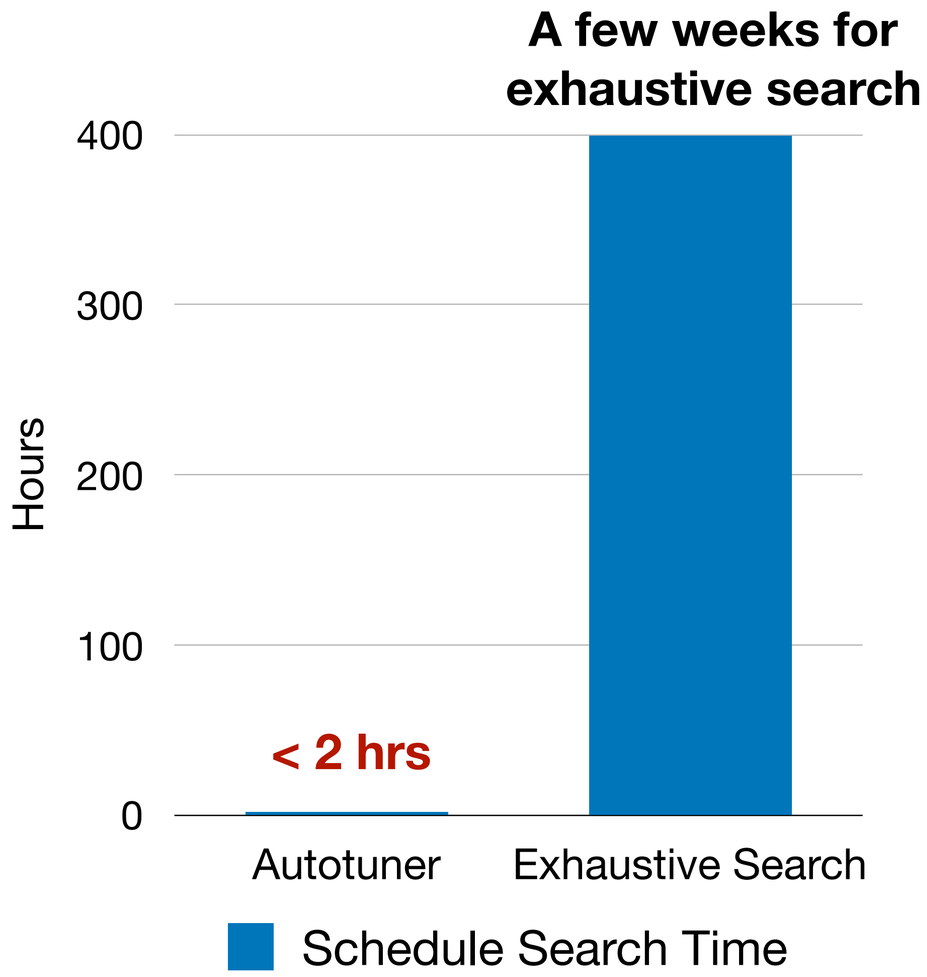
```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
    program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Autotuner


A few weeks for exhaustive search
400
300
200
100
0
Hours
Exhaustive Search
Schedule Search Time

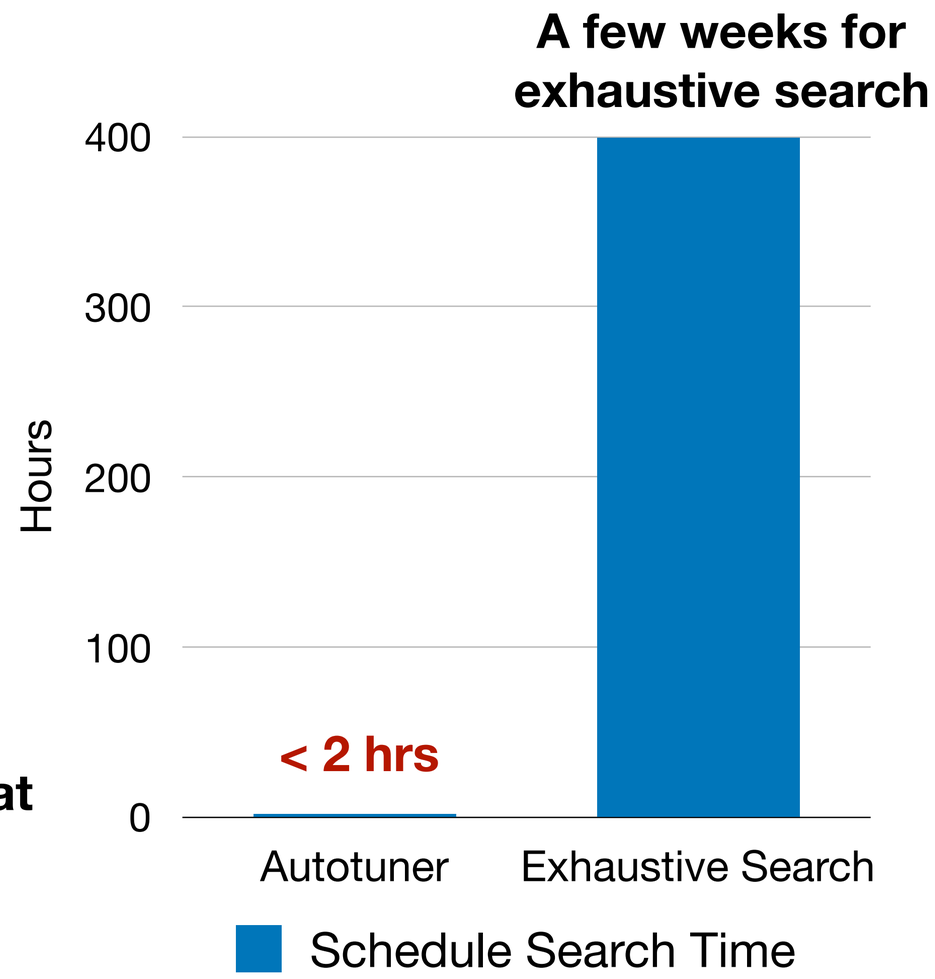
Autotuner

Uses an ensemble of search methods. Build on top of OpenTuner [PACT14]



Autotuner

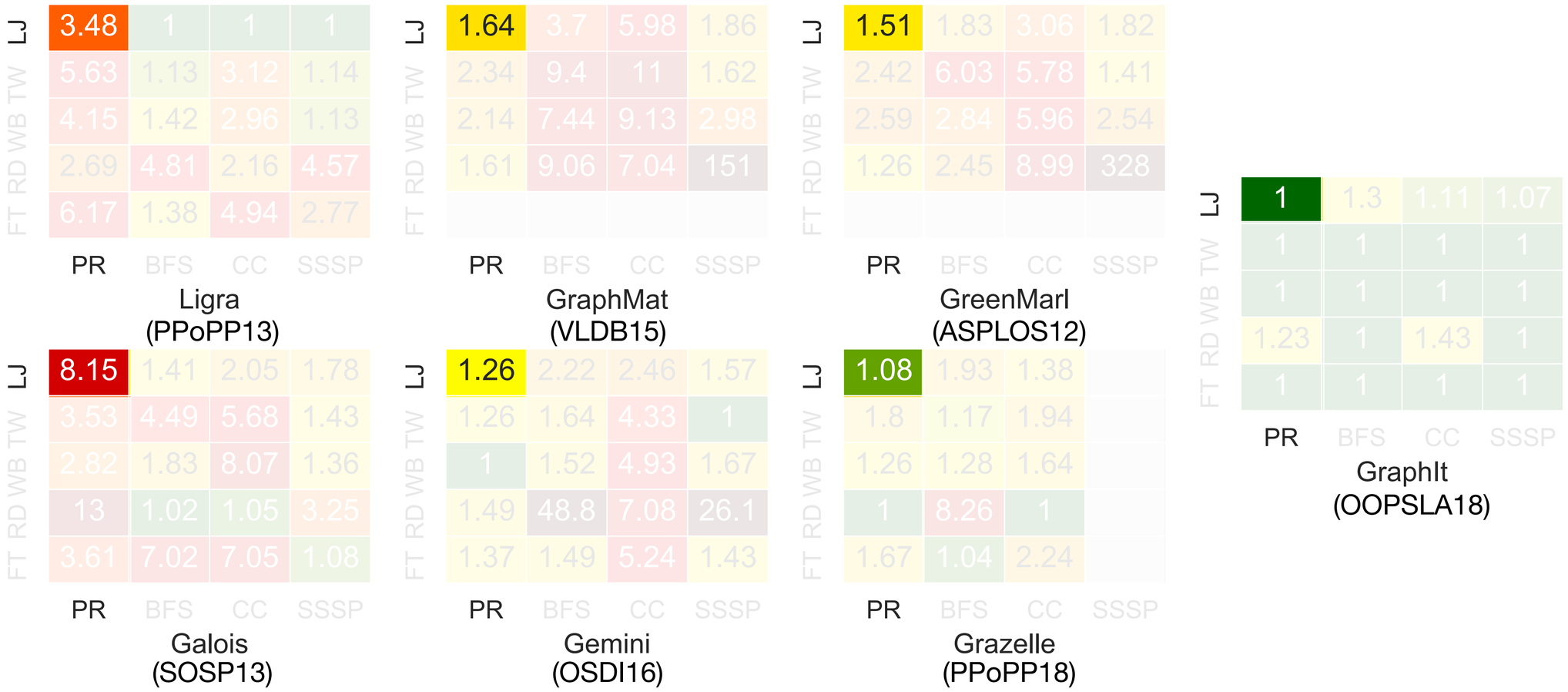
Finds a few schedules that outperform hand-tuned schedules



Outline

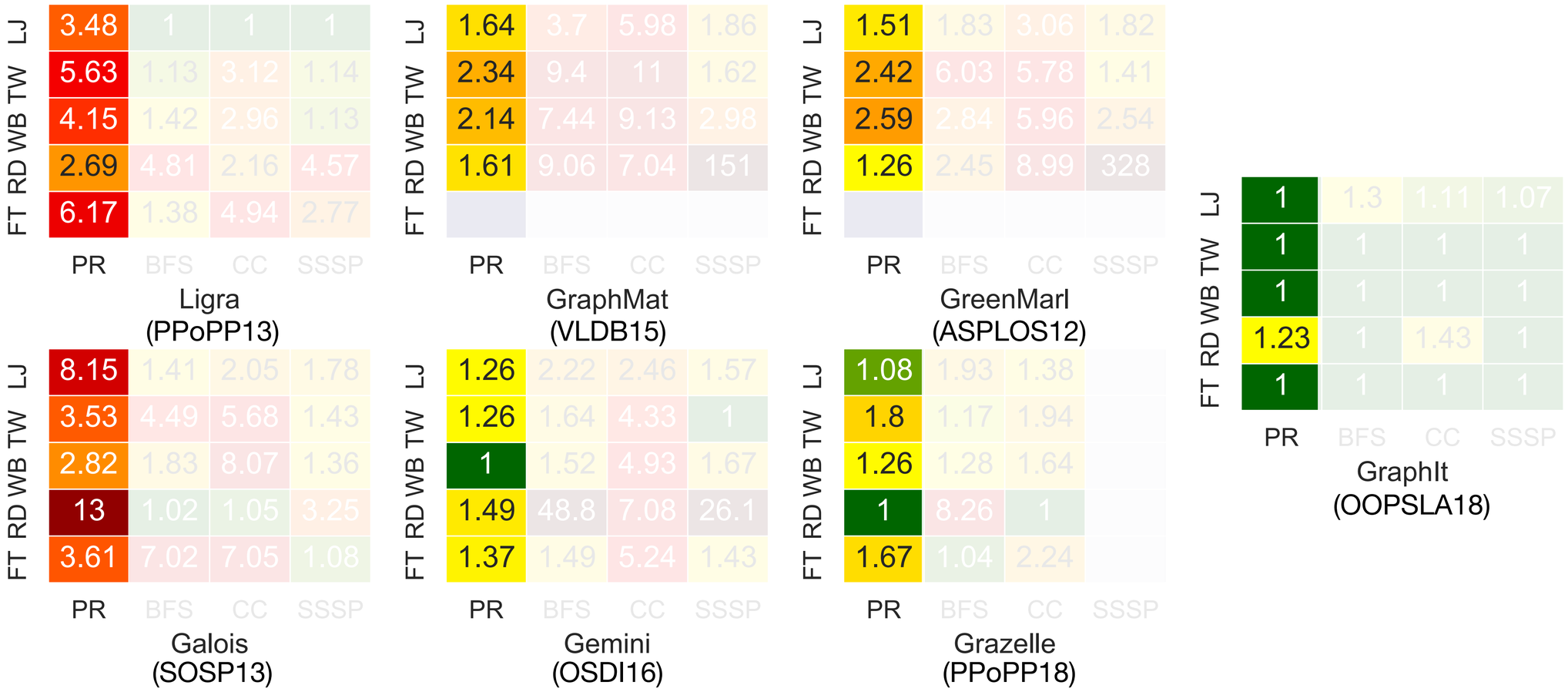
- Graph Applications Overview
- Optimization Tradeoff Space
- GraphIt DSL
- Evaluation

State of the Art and GraphIt



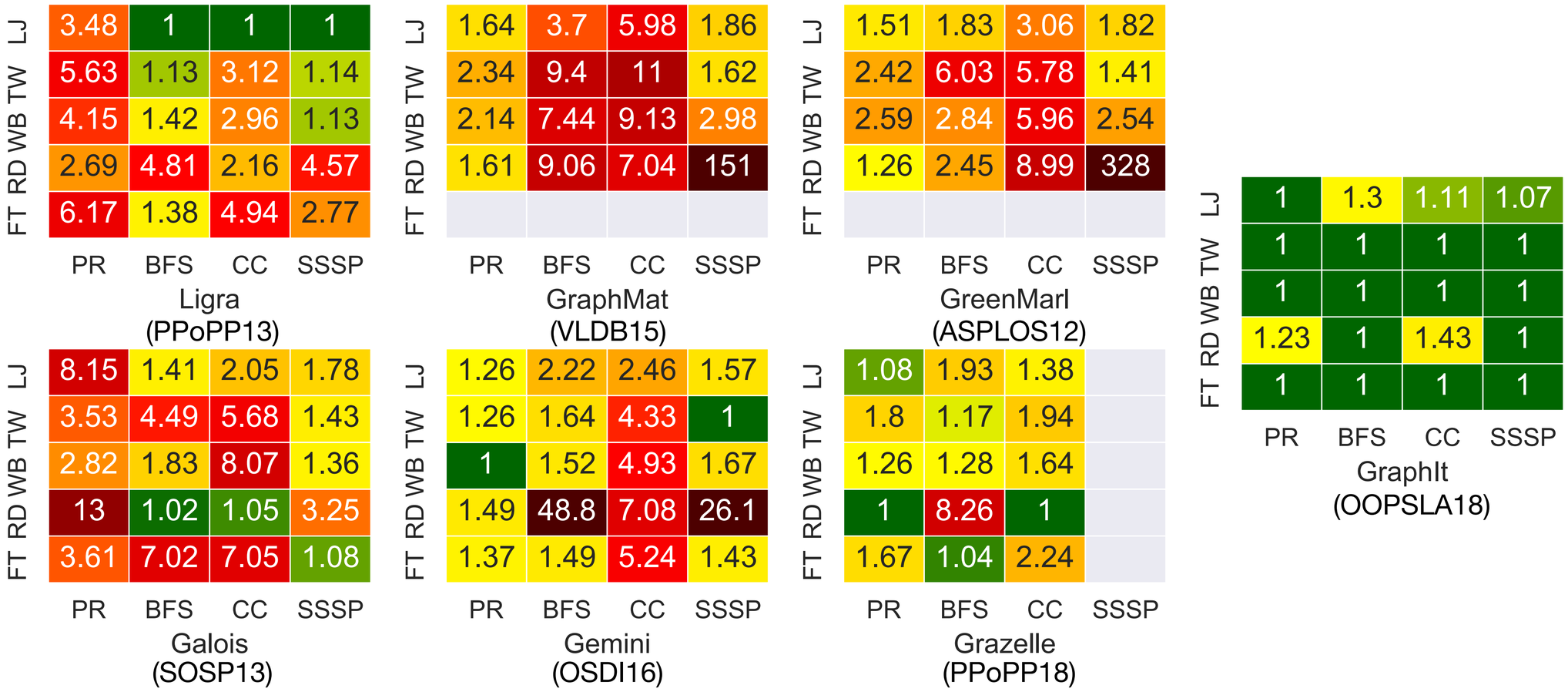
Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

State of the Art and GraphIt



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

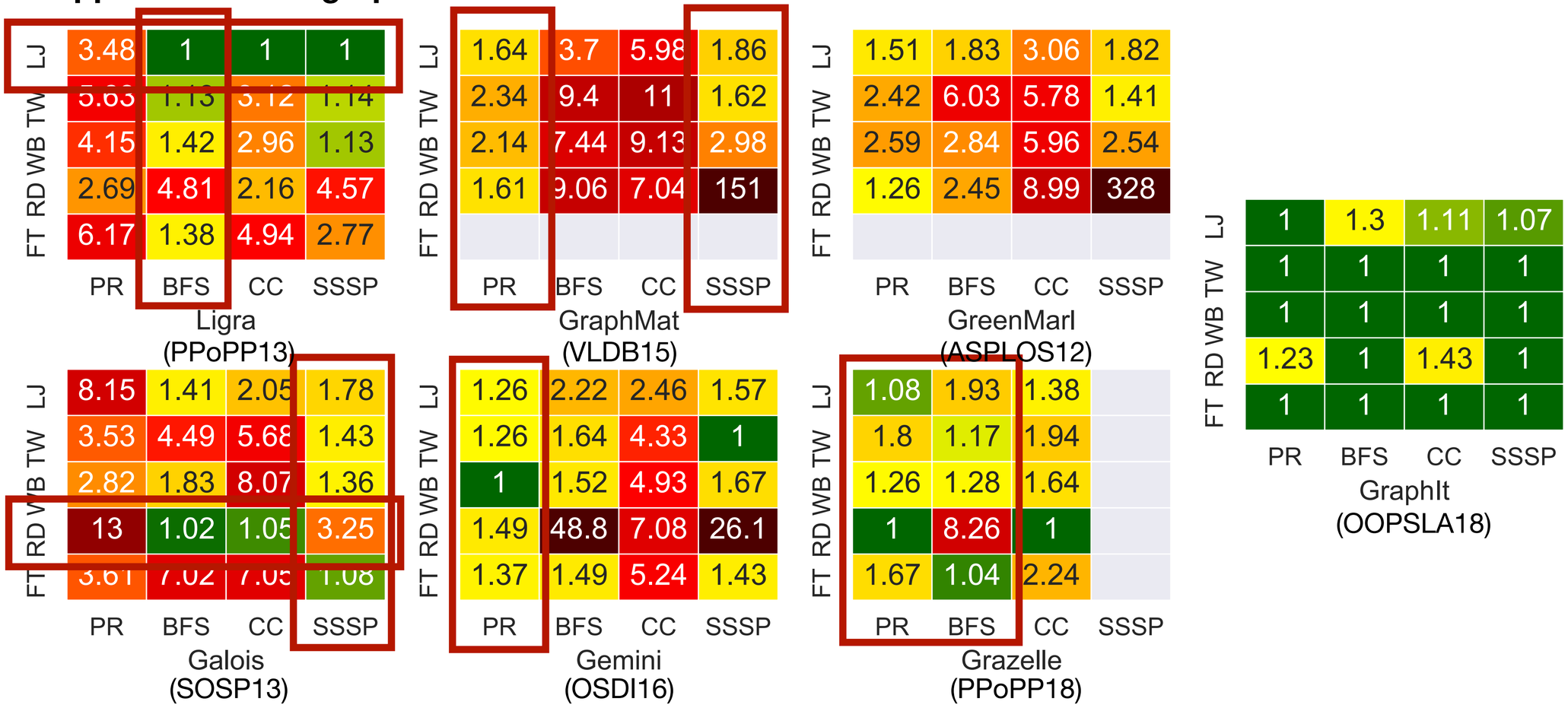
State of the Art and GraphIt



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

State of the Art and GraphIt

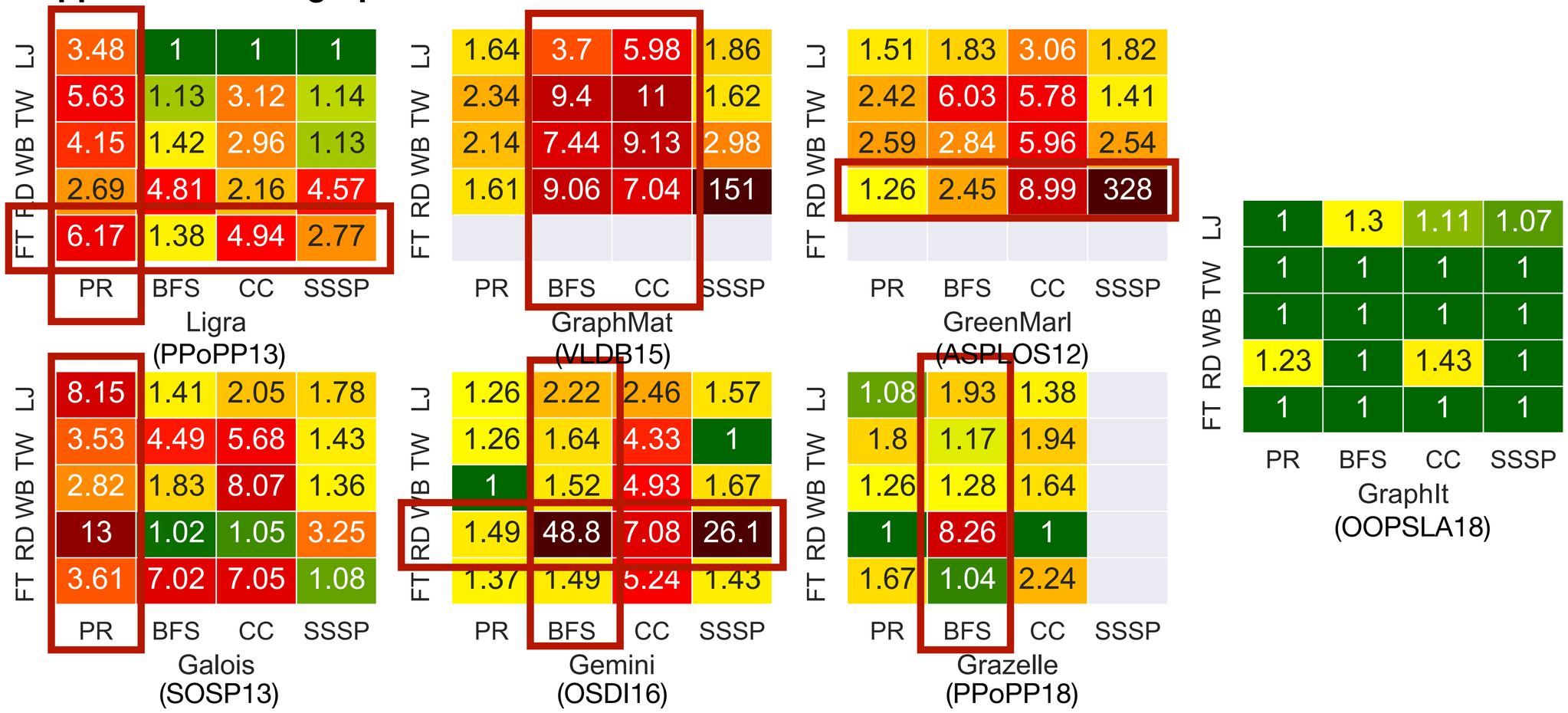
Most frameworks are good at certain applications and graphs



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

State of the Art and GraphIt

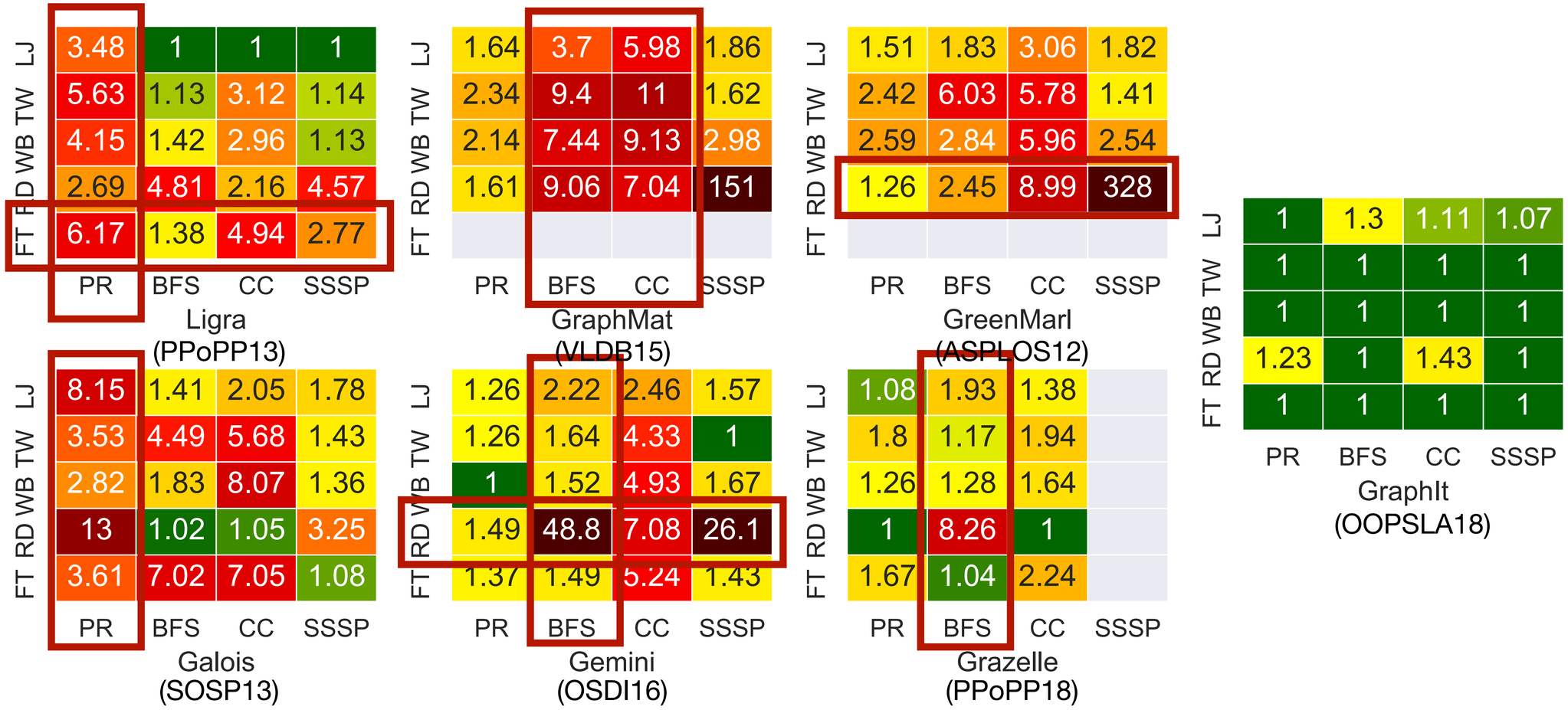
Most frameworks are bad at certain applications and graphs



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

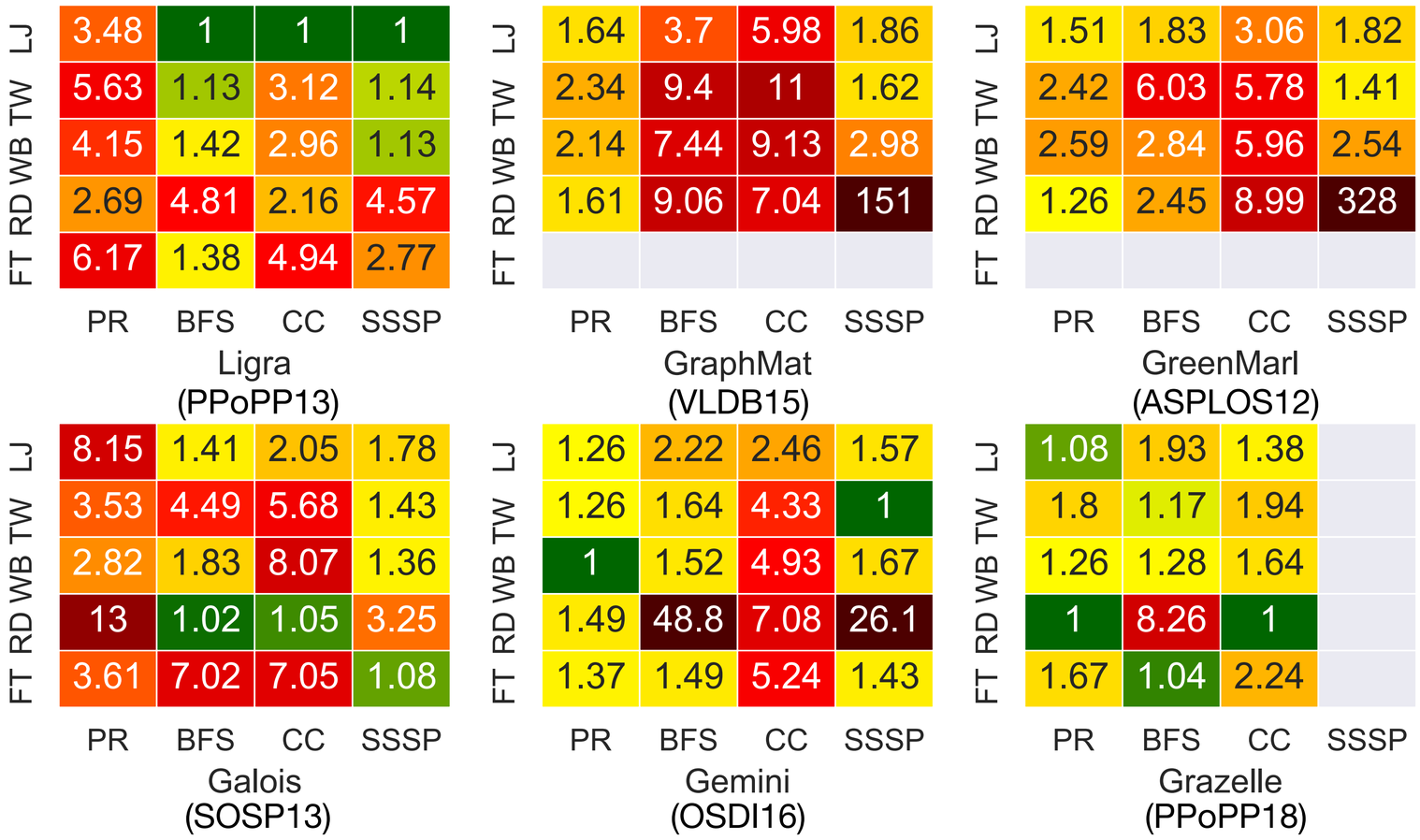
State of the Art and GraphIt

Previous work support a subset of optimizations

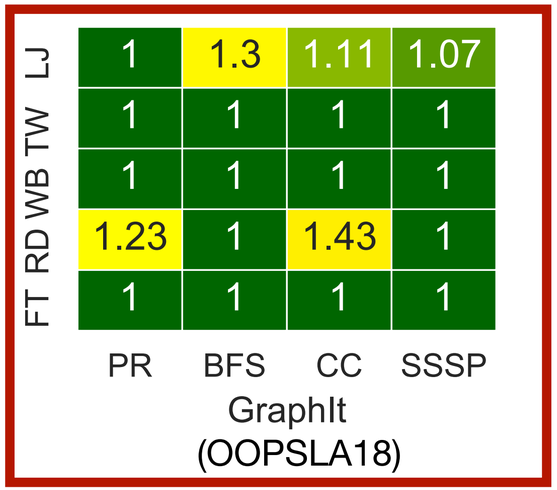


Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Consistent High-Performance

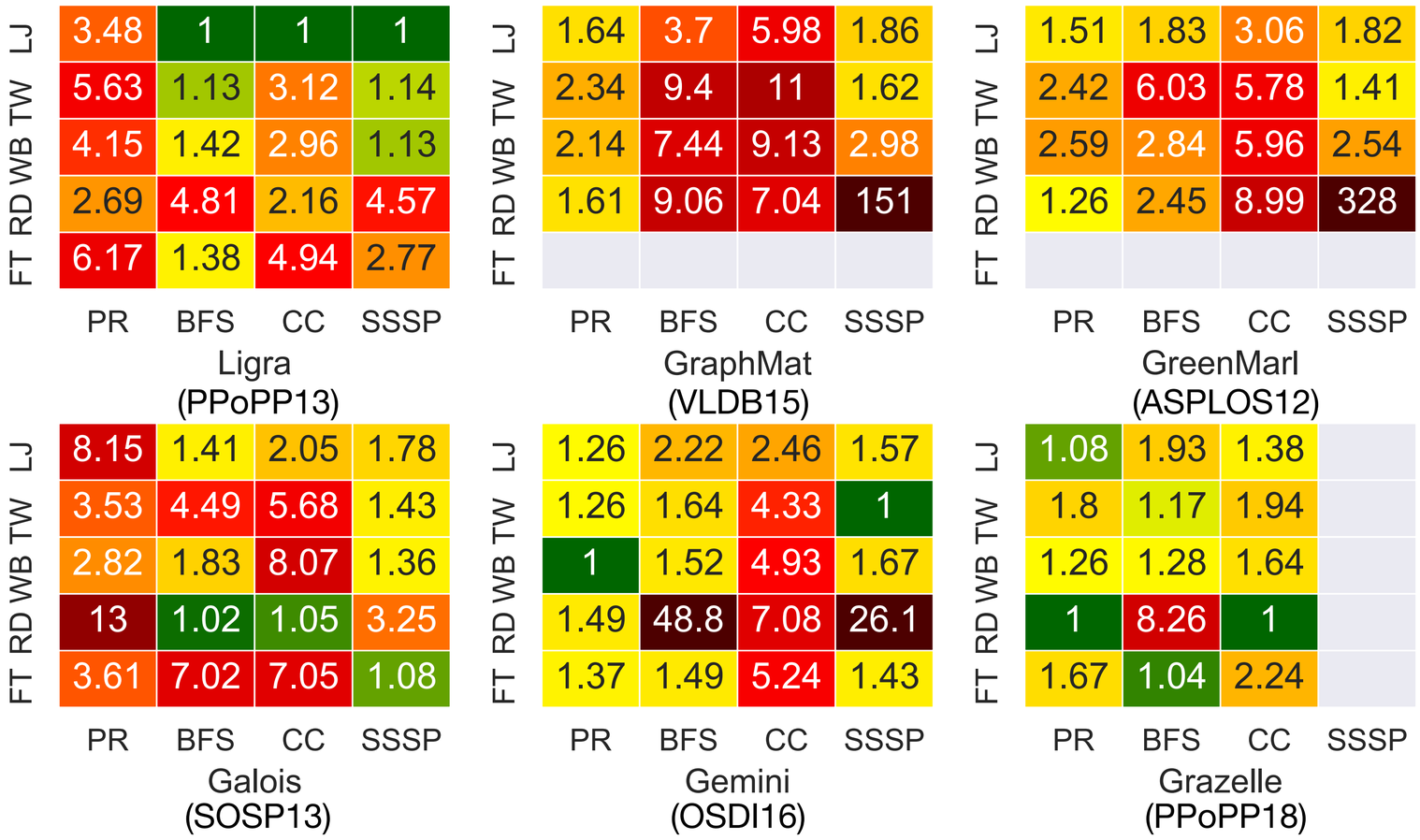


Good across different applications and graphs

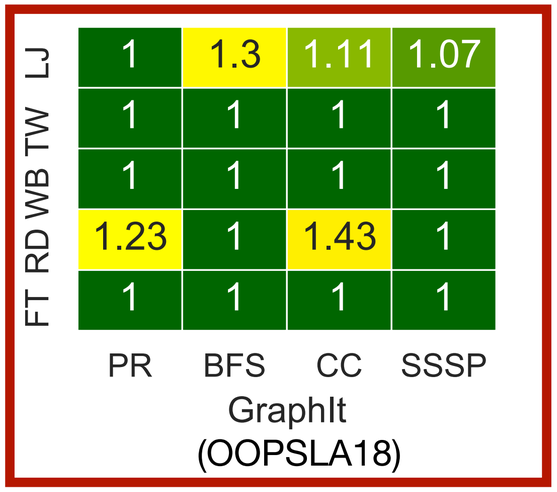


Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Speedup over State of the Art

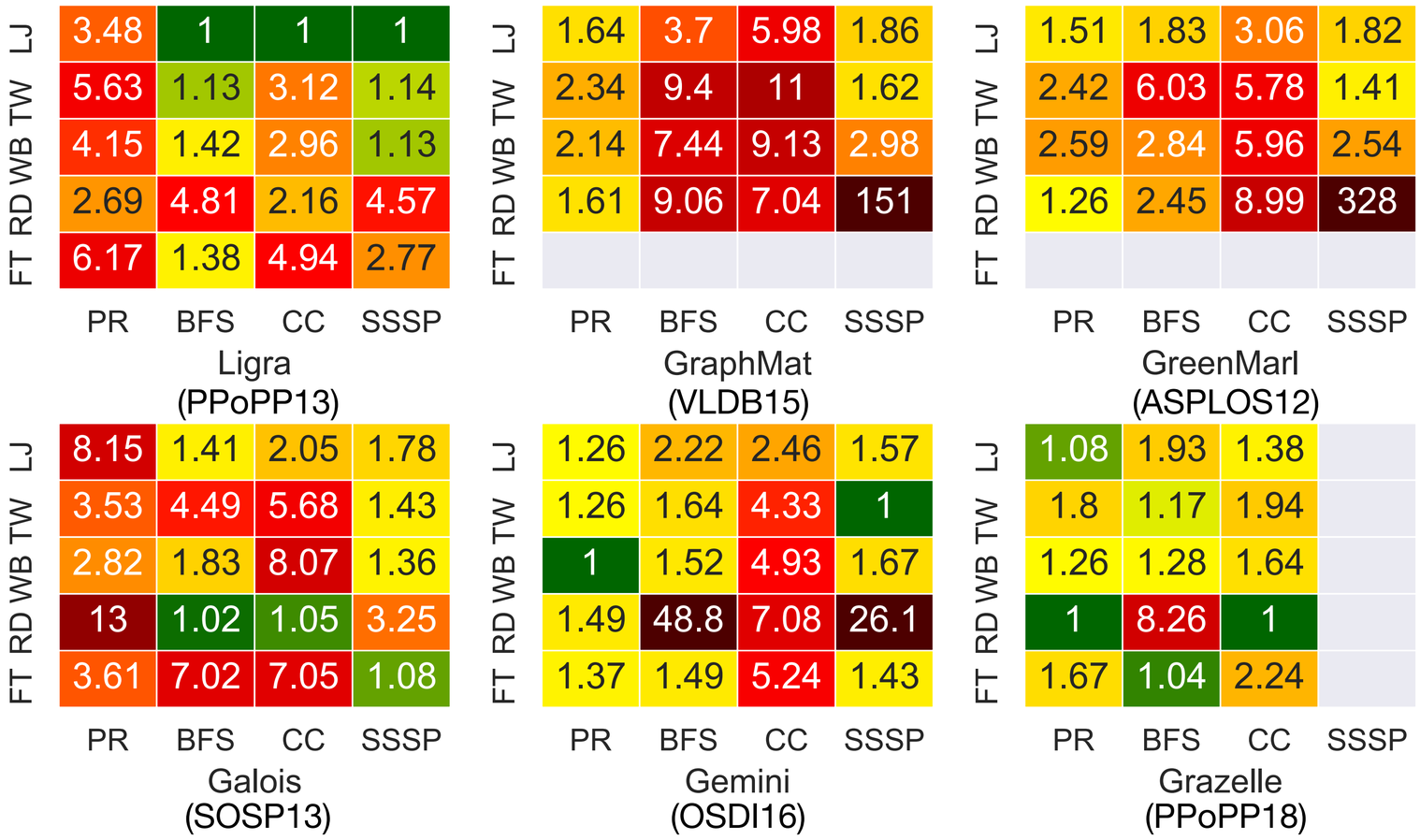


Finds previously unexplored combinations of optimizations

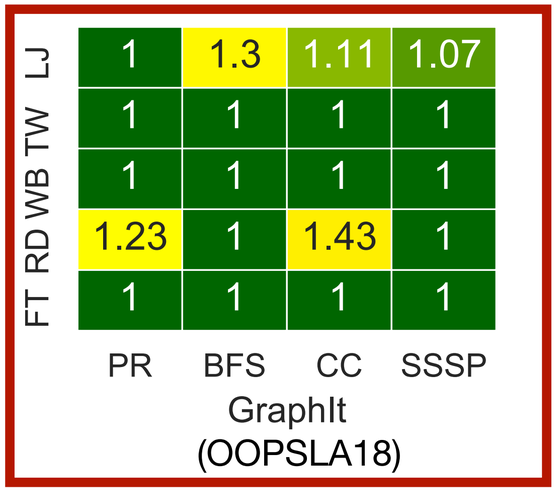


Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Ease-of-Use



Reduces the lines of code by an order of magnitude compare to the next fastest framework



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Related Work

Graph optimizations:

Ligra [PPoPP13], Galois [SOSP13], Propagation Blocking [IPDPS17], CSR Segmentation [BigData 17], Grazelle [PPoPP18] …

- We focus on composing the optimizations

Graph DSLs:

GreenMarl [ASPLOS12], EmptyHeaded [SIGMOD16], Elixir [OOPSLA12], Gluon [PLDI18], Abelian [EuroPar18]…

- We expose extensive performance tuning capabilities

Summary

- Graph application's performance bottleneck depends on data, algorithm, and hardware.
- Decoupling algorithm from optimization achieves consistent high-performance and ease-of-use
- Open source (**graphit-lang.org**).

This Work Supported By: