# A Polynomial Time Algorithm to Minimize Total Travel Time in k-Depot Storage/Retrieval System

Amir Gharehgozli, **Chao Xu**, Wenda Zhang

Aug 24, 2018

# A warehouse

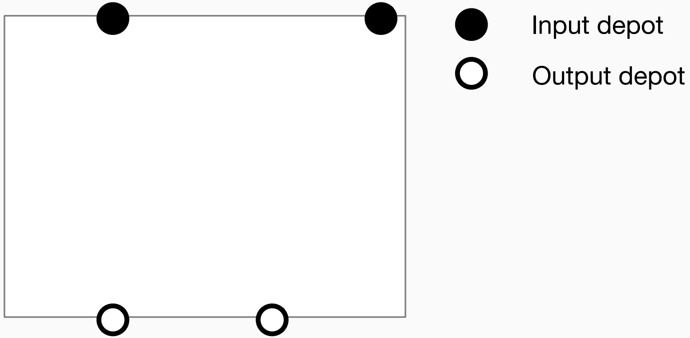
An automated warehouse with input depots and output depots. It has to complete input(storage) and output(retrieval) requests.

**Figure 1:** Demag V-type crane machine. Source: `demagcranes.com`

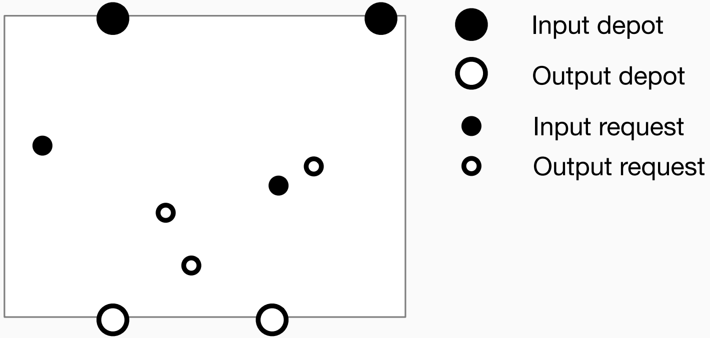
# The storage and retrial machine

- The machine start at some depot.
- The machine can hold at most one item.
- The machine can pick up an item from any input depot, and drop off the item at a input request location.
- The machine can pick up an item from a output request location, and drop off the item at any output depot.
- The machine must return to the original depot.

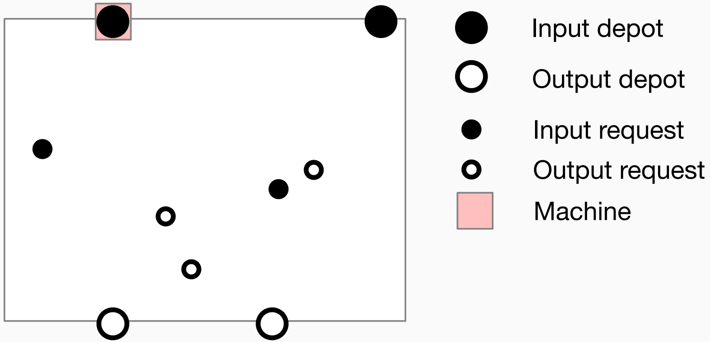
# Example



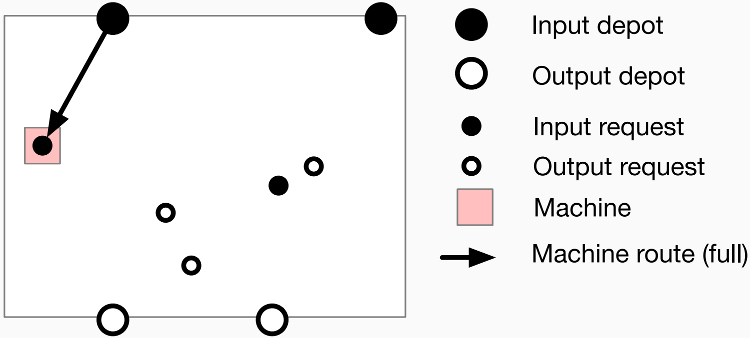
# Example



# Example

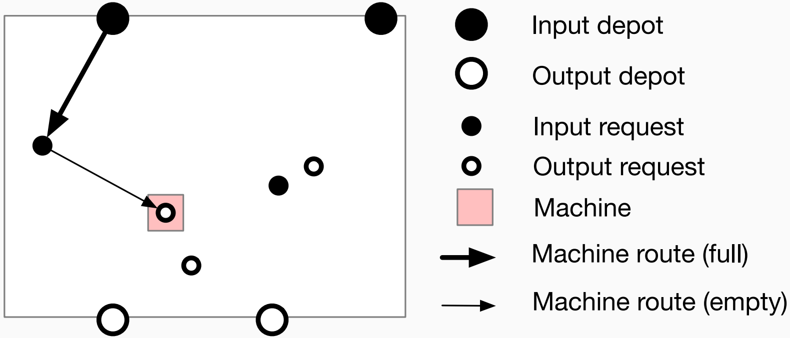

Input depot
Output depot
Input request
Output request
Machine

# Example

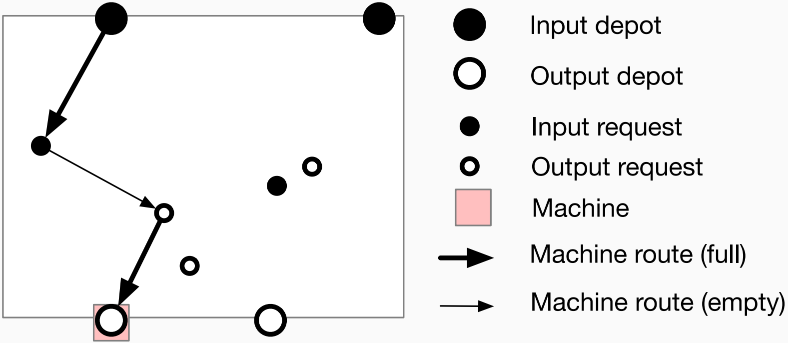


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)

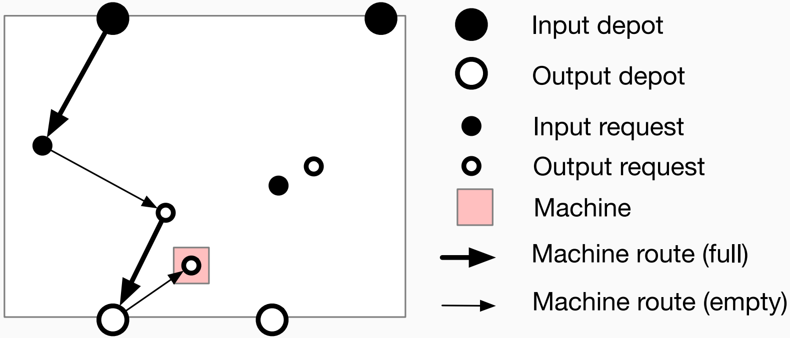
# Example


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

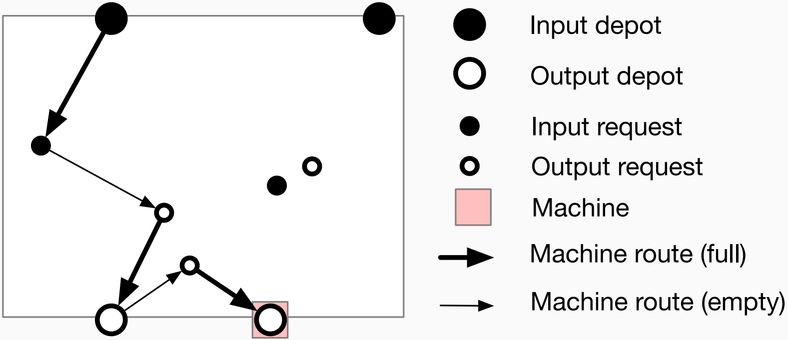
# Example


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example

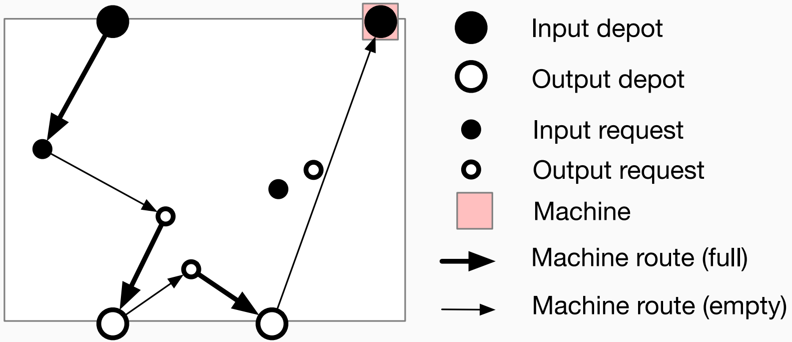

Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example

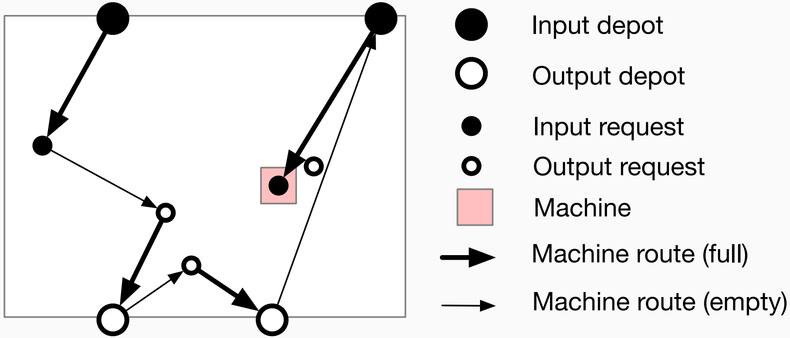


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

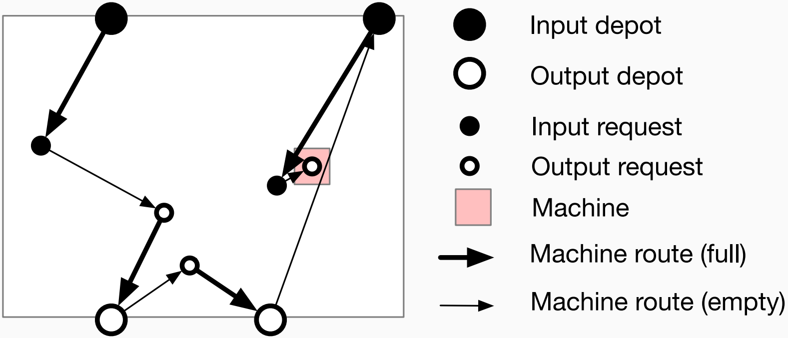
# Example


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example

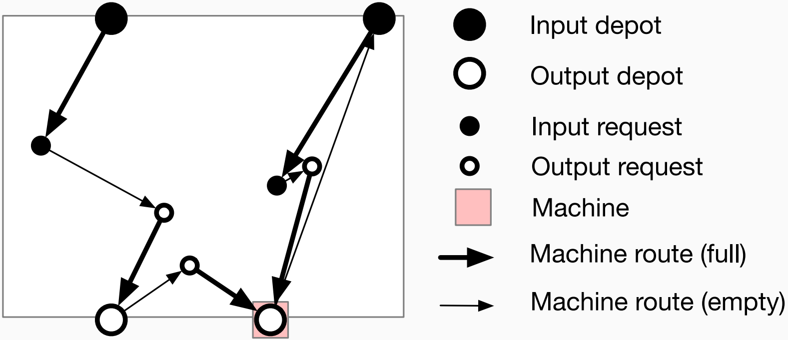

Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example

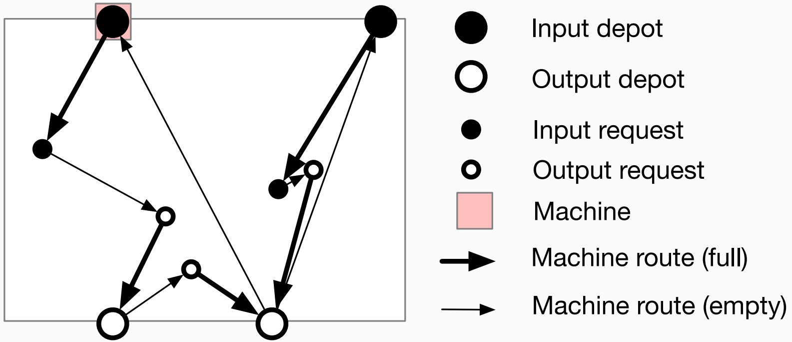


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Example


Input depot
Output depot
Input request
Output request
Machine
Machine route (full)
Machine route (empty)

# Observations

At most 2 request can be completed per depot to depot trip.

- input depot $\rightarrow$ input request$\rightarrow$ output request$\rightarrow$ output depot
- input depot$\rightarrow$ input request$\rightarrow$ arbitrary depot
- arbitrary depot$\rightarrow$ output request$\rightarrow$ output depot
- output depot $\rightarrow$ input depot

# Formalizing the problem: The input

- Input depots $D_I$, output depots $D_O$, $D = D_I \cup D_O$. $|D| = k$.
- Input request $R_I$, output request $R_O$, $R = R_I \cup R_O$. $|R| = n$.
- $V = D \cup R$, the set of vertices.
- $\text{dist} : V \times V \to \mathbb{R}_+$ a asymmetric metric.

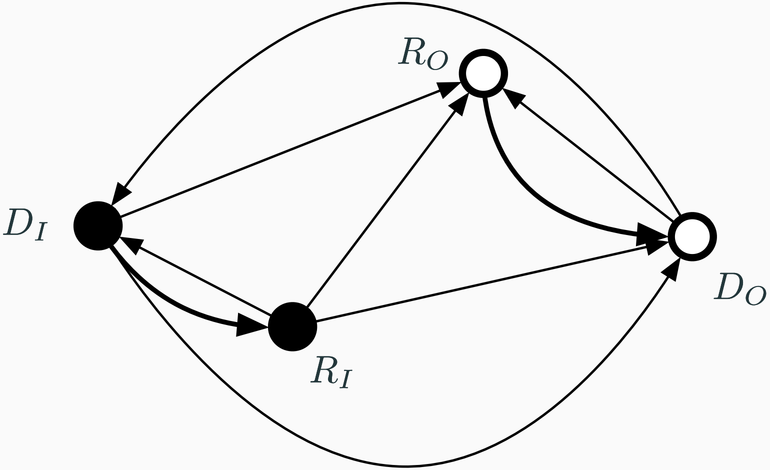
# Model as a walk on a graph

For input $D_I, D_O, R_I, R_O, c$, we construct the following weighted directed graph $G$ on vertices $V$.

- For $d, d' \in D$, there is an edge $(d, d')$.
- For $v \in V$, $u \in R_I$, there is an edge $(u, v)$.
- For $v \in V$, $u \in R_O$, there is an edge $(v, u)$.
- For $v \in R_I$, $d \in D_I$, there is an edge $(d, v)$.
- For $v \in R_O$, $d \in D_O$, there is an edge $(v, d)$.

The cost of an edge $c(u, v) = \text{dist}(u, v)$. Such graph $G$ is called a warehouse network.

# Warehouse network, high level view



# The abstract problem

**Problem: $k$-depot warehouse tour**

**Input:** A warehouse network with $k$ depot vertices.

**Output:** A minimum cost closed walk that goes through every vertex at least once.

# The abstract problem

**Problem: $k$-depot warehouse tour**
**Input:** A warehouse network with $k$ depot vertices.
**Output:** A minimum cost closed walk that goes through every vertex at least once.

Note: we assume in the optimal solution, all $k$ depot has to be visited.

# The abstract problem

**Problem: $k$-depot warehouse tour**
**Input:** A warehouse network with $k$ depot vertices.
**Output:** A minimum cost closed walk that goes through every vertex at least once.

Note: we assume in the optimal solution, all $k$ depot has to be visited.

Closely related to TSP.

# The abstract problem

**Problem: $k$-depot warehouse tour**
**Input:** A warehouse network with $k$ depot vertices.
**Output:** A minimum cost closed walk that goes through every vertex at least once.

Note: we assume in the optimal solution, all $k$ depot has to be visited.

Closely related to TSP.

Observation: there is an optimal solution that goes through each vertex in $R$ exactly once, because dist is a metric.

# Previous results

A regular depot is a pair of input depot $d$ and output depot $d'$ with $\text{dist}(d, d') = 0$.

**[Gharehgozli, Yu, Zhang, de Koster '17]** considered special cases of the problem.

- $k = 4$: 2 pairs of regular depots. Running time $O(n^6)$.
- $k = 2$: 2 depots, one input, one output. Running time $O(n^3)$.

# Our result

Let $MCF(n, m)$ be the running time to solve min-cost flow on a unit capacity graph with $m$ edges and $n$ vertices.
$MCF(n, m) = \tilde{O}(\sqrt{n}m)$, **[Lee-Sidford '13]**.

**Theorem**
*The k-depot warehouse tour can be solved in*

- $O(n^{k+1} + MCF(n, n^2))$ *time if all depots are input(output) depots.*
- $O(n^k + MCF(n, n^2))$ *time otherwise.*

# Our result

Let $MCF(n, m)$ be the running time to solve min-cost flow on a unit capacity graph with $m$ edges and $n$ vertices.
$MCF(n, m) = \tilde{O}(\sqrt{n}m)$, **[Lee-Sidford '13]**.

**Theorem**
*The k-depot warehouse tour can be solved in*

- $O(n^{k+1} + MCF(n, n^2))$ *time if all depots are input(output) depots.*
- $O(n^k + MCF(n, n^2))$ *time otherwise.*

Counterintuitive! Having depots of only one type is harder.

# A simple polynomial time algorithm

# What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

## What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

1. Circulation property: The in-degree and out-degree are the same for each vertex.

## What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

1. Circulation property: The in-degree and out-degree are the same for each vertex.
2. Covering property: Each vertex has in-degree at least 1.

## What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

1. Circulation property: The in-degree and out-degree are the same for each vertex.
2. Covering property: Each vertex has in-degree at least 1. Each vertex in $R$ has in-degree exactly 1.

# What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

1. Circulation property: The in-degree and out-degree are the same for each vertex.
2. Covering property: Each vertex has in-degree at least 1. Each vertex in $R$ has in-degree exactly 1.
3. Connectivity property: $H$ is (weakly) connected.

# What does a solution look like?

The feasible solution is a closed walk $W$. The (disjoint) union of the edges in the solution is a multigraph $H$ with the following properties.

1. Circulation property: The in-degree and out-degree are the same for each vertex.
2. Covering property: Each vertex has in-degree at least 1. Each vertex in $R$ has in-degree exactly 1.
3. Connectivity property: $H$ is (weakly) connected.

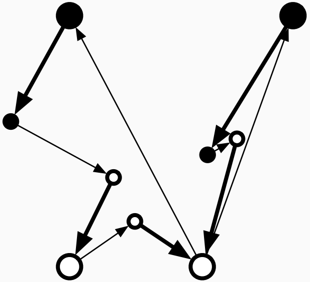
Every graph with the above properties induces a feasible solution: it is a Eulerian graph that contains all vertices.

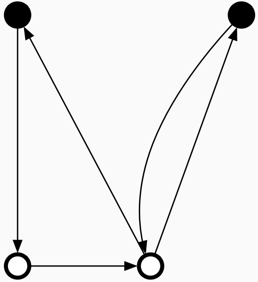
# A simpler connectivity condition

**Theorem**
*If $H$ a subgraph of $G$ has the circulation property and covering property, then it is connected if and only if $D$ is connected.*

# Structure graph of a solution

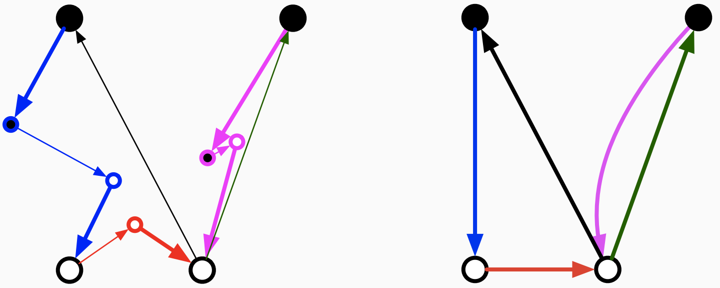
The structure graph of a solution is obtained by the following transformation. For each depot to depot path that does not contain any other depot $P$. Let $P'$ be the sequence of internal vertices, and $P$ is from $d$ to $d'$. We create an edge $e$ from $d$ to $d'$, and give it the label $P'$.





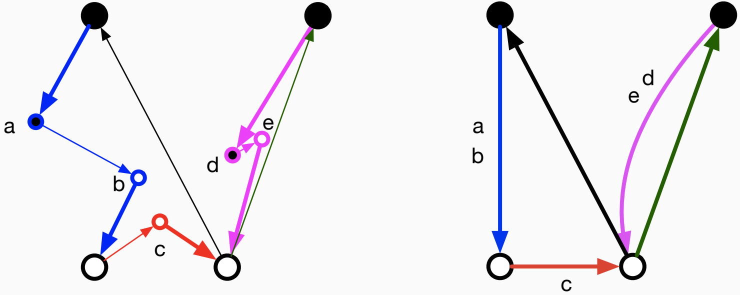
## Structure graph of a solution

The structure graph of a solution is obtained by the following transformation. For each depot to depot path that does not contain any other depot $P$. Let $P'$ be the sequence of internal vertices, and $P$ is from $d$ to $d'$. We create an edge $e$ from $d$ to $d'$, and give it the label $P'$.



# Structure graph of a solution

The structure graph of a solution is obtained by the following transformation. For each depot to depot path that does not contain any other depot $P$. Let $P'$ be the sequence of internal vertices, and $P$ is from $d$ to $d'$. We create an edge $e$ from $d$ to $d'$, and give it the label $P'$.



# Idea

If $F$ is a subset of edges of a structure graph, $\phi(F)$ are the edges in the solution corresponding to $F$.

# Idea

If $F$ is a subset of edges of a structure graph, $\phi(F)$ are the edges in the solution corresponding to $F$.

- The optimal structure graph contains some connected subgraph $F$.

# Idea

If $F$ is a subset of edges of a structure graph, $\phi(F)$ are the edges in the solution corresponding to $F$.

- The optimal structure graph contains some connected subgraph $F$.
- Find minimum cost valid subgraph of $G$ containing all the edges of $\phi(F)$ implies finding an optimal solution.

# The algorithm

$\mathcal{T}$ be a set of trees(in the undirected sense) such that every structure graph must contain at least one of the tree as a subgraph.

# The algorithm

$\mathcal{T}$ be a set of trees(in the undirected sense) such that every structure graph must contain at least one of the tree as a subgraph.

For each $T \in \mathcal{T}$

Find an minimum cost valid subgraph of $G$ that contains all edges in $\phi(T)$.

Return the minimum

## The algorithm

$\mathcal{T}$ be a set of trees(in the undirected sense) such that every structure graph must contain at least one of the tree as a subgraph.

For each $T \in \mathcal{T}$

Find an minimum cost valid subgraph of $G$ that contains all edges in $\phi(T)$.

Return the minimum

Running time

$O(|\mathcal{T}| \times$ time to find a minimum cost valid subgraph).

# The algorithm

$\mathcal{T}$ be a set of trees(in the undirected sense) such that every structure graph must contain at least one of the tree as a subgraph.
For each $T \in \mathcal{T}$

Find an minimum cost valid subgraph of $G$ that contains all edges in $\phi(T)$.

Return the minimum

Running time
$O(|\mathcal{T}| \times$ time to find a minimum cost valid subgraph$)$.

Time to find a minimum cost valid subgraph:

# The algorithm

$\mathcal{T}$ be a set of trees(in the undirected sense) such that every structure graph must contain at least one of the tree as a subgraph.
For each $T \in \mathcal{T}$
    Find an minimum cost valid subgraph of $G$ that contains all edges in $\phi(T)$.
Return the minimum

Running time
$O(|\mathcal{T}| \times$ time to find a minimum cost valid subgraph).

Time to find a minimum cost valid subgraph:reduces to a min-cost flow computation on a unit capacity graph of $O(n^2)$ edges.

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.

The weight of a tree is the number of labels on the edges.

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.

The weight of a tree is the number of labels on the edges.

Claim: $|\mathcal{T}| = O(n^{2(k-1)})$.

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.
The weight of a tree is the number of labels on the edges.
Claim: $|\mathcal{T}| = O(n^{2(k-1)})$.

- There are $k$ nodes, so there can be $f(k)$ trees (ignoring labels).

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.
The weight of a tree is the number of labels on the edges.
Claim: $|\mathcal{T}| = O(n^{2(k-1)})$.

- There are $k$ nodes, so there can be $f(k)$ trees (ignoring labels).
- Each tree has $k-1$ edges. Each edge can have at most 2 labels.

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.
The weight of a tree is the number of labels on the edges.
Claim: $|\mathcal{T}| = O(n^{2(k-1)})$.

- There are $k$ nodes, so there can be $f(k)$ trees (ignoring labels).
- Each tree has $k-1$ edges. Each edge can have at most 2 labels.
- Each tree has at most $2(k-1)$ labels (weight at most $2(k-1)$).

# A candidate set of trees

$\mathcal{T}$ is the set of spanning trees that can appear in a structure graph.
The weight of a tree is the number of labels on the edges.
Claim: $|\mathcal{T}| = O(n^{2(k-1)})$.

- There are $k$ nodes, so there can be $f(k)$ trees (ignoring labels).
- Each tree has $k-1$ edges. Each edge can have at most 2 labels.
- Each tree has at most $2(k-1)$ labels (weight at most $2(k-1)$).

There are

$$f(k)\binom{n}{2}\binom{n-2}{2}\dots\binom{n-2(k-1)}{2} = O(n^{2(k-1)})$$

trees.

# A slow polynomial time algorithm

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in $O(n^{2(k-1)} MCF(n, n^2))$ time.*

# A slow polynomial time algorithm

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in* $O(n^{2(k-1)} MCF(n, n^2))$ *time.*

A simple improvement: use dynamic min-cost flow. Update the valid subgraph in $O(n^2)$ time.

# A slow polynomial time algorithm

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in* $O(n^{2(k-1)} MCF(n, n^2))$ *time.*

A simple improvement: use dynamic min-cost flow. Update the valid subgraph in $O(n^2)$ time.

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in* $O(n^{2(k-1)} \cdot n^2 + MCF(n, n^2)) = O(n^{2k} + MCF(n, n^2))$ *time.*

# A slow polynomial time algorithm

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in* $O(n^{2(k-1)}MCF(n, n^2))$ *time.*

A simple improvement: use dynamic min-cost flow. Update the valid subgraph in $O(n^2)$ time.

**Theorem**
*There exists an algorithm that solves the k-depot warehouse tour problem in* $O(n^{2(k-1)} \cdot n^2 + MCF(n, n^2)) = O(n^{2k} + MCF(n, n^2))$ *time.*

Worse than the state of the art for $k \leq 4$.

# Faster algorithm: using a better set of trees.

Our analysis:

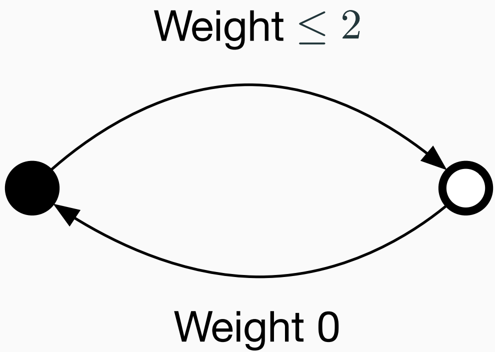
- $\mathcal{T}$ set of possible spanning trees in structure graphs.
- Bound $|\mathcal{T}|$ by $O(n^w)$, where $w$ is the maximum weight over all trees in $\mathcal{T}$.

Our analysis:

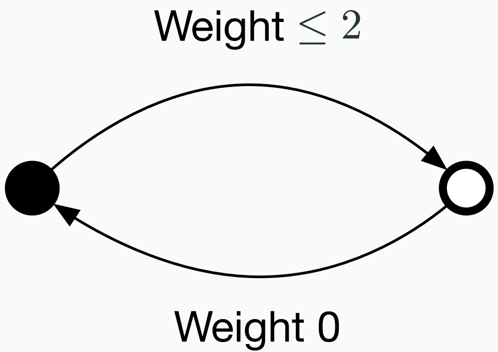
- $\mathcal{T}$ set of possible spanning trees in structure graphs.
- Bound $|\mathcal{T}|$ by $O(n^w)$, where $w$ is the maximum weight over all trees in $\mathcal{T}$.

Idea: Let $\mathcal{T}$ be the set of *minimum* spanning trees.

# Do we expect improvements?



# Do we expect improvements?




Yes!

# The punch line

$\text{mst}(H)$: the weight of the minimum spanning tree in $H$.

# The punch line

mst($H$): the weight of the minimum spanning tree in $H$.

**Theorem (MST theorem)**
*Let $H$ be a structure graph on $k$ vertices, and $|D_I|, |D_O| \geq 1$, then* $\mathrm{mst}(H) \leq k - 2$.

# The punch line

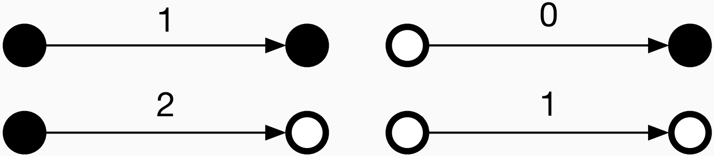
mst($H$): the weight of the minimum spanning tree in $H$.

**Theorem (MST theorem)**
*Let $H$ be a structure graph on $k$ vertices, and $|D_I|, |D_O| \geq 1$, then* mst$(H) \leq k - 2$.

**Corollary**
*There exists an algorithm for $k$-depot warehouse tour with running time $O(n^k + MCF(n, n^2))$, for the case when there is at least one input and output depot.*

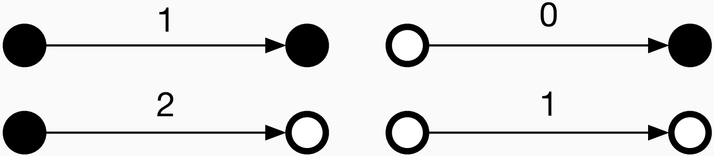
# Weights of the structure graph

Upper bound on the weights of edges, depending on depot type.



# Weights of the structure graph

Upper bound on the weights of edges, depending on depot type.

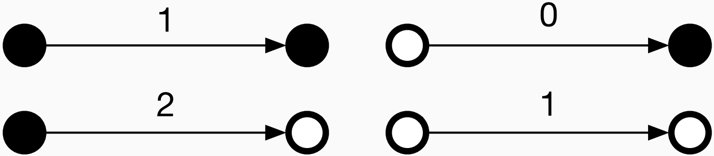



Summarized by having two kinds of vertex weights.

- $w_0(v) = 0$ if $v \in D_O$, $w_0(v) = 1$ if $v \in D_I$.
- $w_1(v) = 0$ if $v \in D_I$, $w_1(v) = 1$ if $v \in D_O$.
- $w'((u, v)) = w_0(u) + w_1(v)$.

# Weights of the structure graph

Upper bound on the weights of edges, depending on depot type.



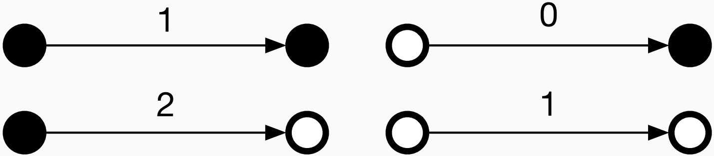

Summarized by having two kinds of vertex weights.

- $w_0(v) = 0$ if $v \in D_O$, $w_0(v) = 1$ if $v \in D_I$.
- $w_1(v) = 0$ if $v \in D_I$, $w_1(v) = 1$ if $v \in D_O$.
- $w'((u, v)) = w_0(u) + w_1(v)$.

$w'(e)$ is an upper bound to the edge weight of $e$.

# Weights of the structure graph

Upper bound on the weights of edges, depending on depot type.




Summarized by having two kinds of vertex weights.

- $w_0(v) = 0$ if $v \in D_O$, $w_0(v) = 1$ if $v \in D_I$.
- $w_1(v) = 0$ if $v \in D_I$, $w_1(v) = 1$ if $v \in D_O$.
- $w'((u, v)) = w_0(u) + w_1(v)$.

$w'(e)$ is an upper bound to the edge weight of $e$. We will abuse the notation and refer $w'(e)$ as the edge weight.

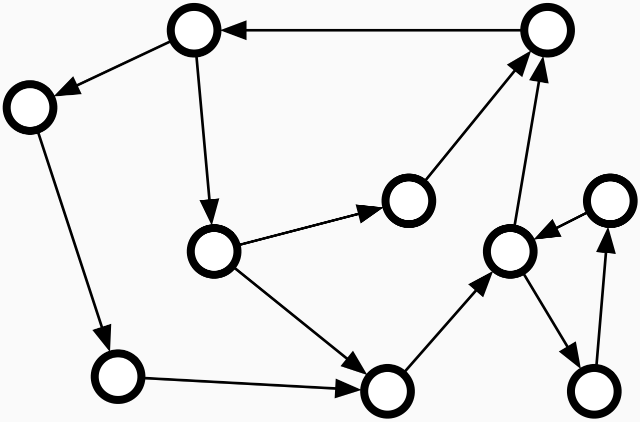
# Ear decomposition

Let $G = (V, E)$ be a directed graph. A sequence of set of edges $E_1, \ldots, E_k$ that partitions $E$ is a ear decomposition if:

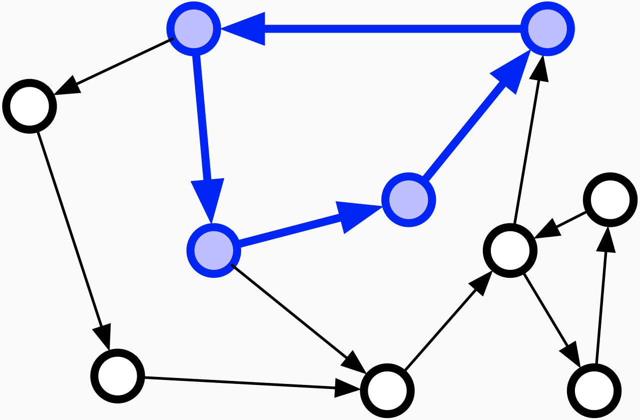
- $E_1$ is a cycle, each $E_2, \ldots, E_k$ is a path(including cycles).
- The start and end of the path $E_i$ are vertices in $V(E_1 \cup \ldots \cup E_{i-1})$. No other vertex in $V(E_i)$ is in $V(E_1 \cup \ldots \cup E_{i-1})$.

$E_1, \ldots, E_k$ are called ears.

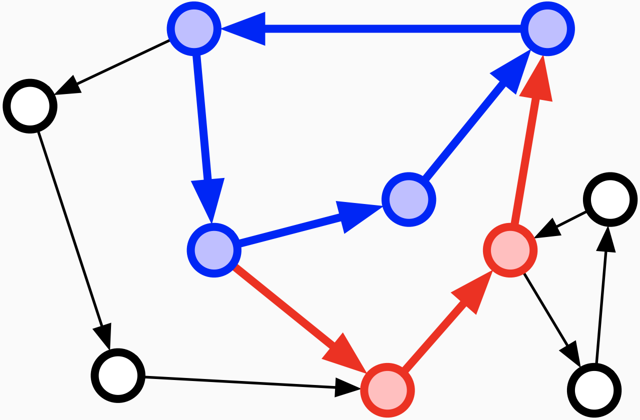
# Example of an ear decomposition



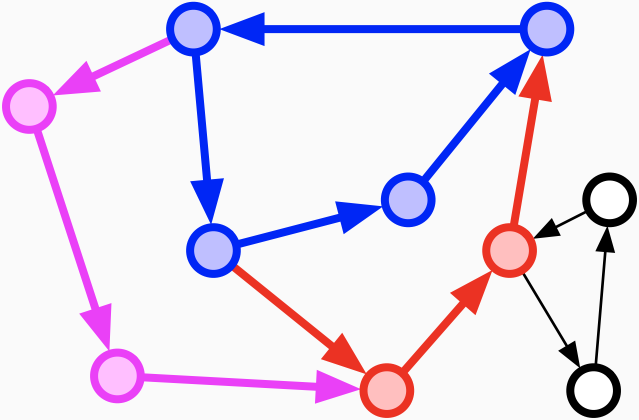
# Example of an ear decomposition



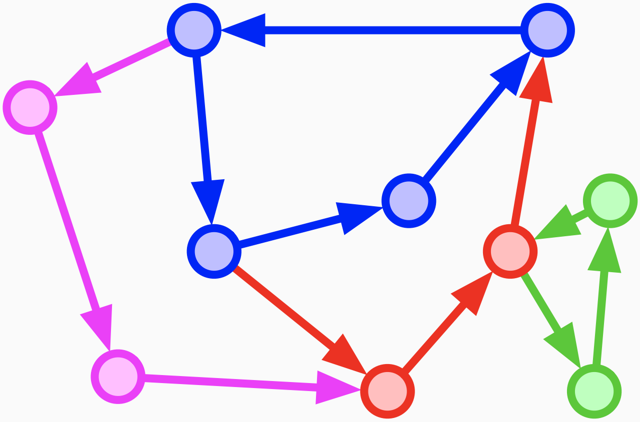
# Example of an ear decomposition



# Example of an ear decomposition



# Example of an ear decomposition



# Ear decomposition

**Theorem**
*Let $G$ be a strongly connected directed graph, and $C$ is a cycle in $G$. There exists a ear decomposition $E_1, \ldots, E_j$ where $E_1 = C$.*

## Proof of the MST theorem

Proof by induction on the number of ears in the ear decomposition.

Let $H$ have ear decomposition $E_1, \ldots, E_t$. We can chose $E_1$ to be a cycle with at least one input depot and one output depot.

# Base Case

**Theorem**
*Let $P = v_1, \ldots, v_n$ be a path and $P$ start with a input depot, and end with an output depot, then there exists an edge of weight $2$.*

# Base Case

**Theorem**
*Let* $P = v_1, \ldots, v_n$ *be a path and* $P$ *start with a input depot, and end with an output depot, then there exists an edge of weight* $2$*.*

**Proof.**
Since $v_1$ is an input depot, $v_n$ is an output depot. For some $i$, $v_i$ is an input depot and $v_{i+1}$ is an output depot. The edge $v_i v_{i+1}$ has weight 2. □

# Base Case

**Theorem**
*Let $P = v_1, \ldots, v_n$ be a path and $P$ start with a input depot, and end with an output depot, then there exists an edge of weight 2.*

**Proof.**
Since $v_1$ is an input depot, $v_n$ is an output depot. For some $i$, $v_i$ is an input depot and $v_{i+1}$ is an output depot. The edge $v_i v_{i+1}$ has weight 2. □

**Theorem**
*$C$ is a cycle of $k$ vertices with at least one input depot and one output depot, then $\mathrm{mst}(C) = k - 2$.*

# Base Case

**Theorem**
*Let $P = v_1, \ldots, v_n$ be a path and $P$ start with a input depot, and end with an output depot, then there exists an edge of weight 2.*

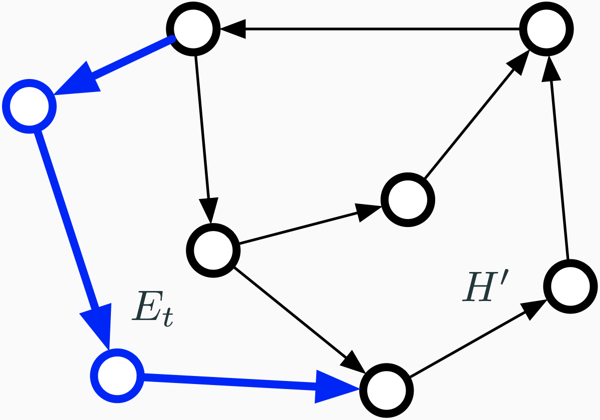
**Proof.**
Since $v_1$ is an input depot, $v_n$ is an output depot. For some $i$, $v_i$ is an input depot and $v_{i+1}$ is an output depot. The edge $v_i v_{i+1}$ has weight 2. □

**Theorem**
*$C$ is a cycle of $k$ vertices with at least one input depot and one output depot, then $\mathrm{mst}(C) = k - 2$.*

**Proof.**
The total edge weight is $\sum_{e \in C} w'(e) = \sum_{v \in C} w_0(v) + w_1(v) = k$. Take any path from an input depot to an output depot, and remove the weight 2 edge in the path. □

# Inductive step

Path case. Assume $E_t$ is a path and not a cycle.
$H' = (V(E_1 \cup \ldots \cup E_{t-1}), E_1 \cup \ldots \cup E_{t-1})$.
$\text{mst}(H) \leq \text{mst}(H') + w'(E_t) - \max_{e \in E_t} w'(e)$.

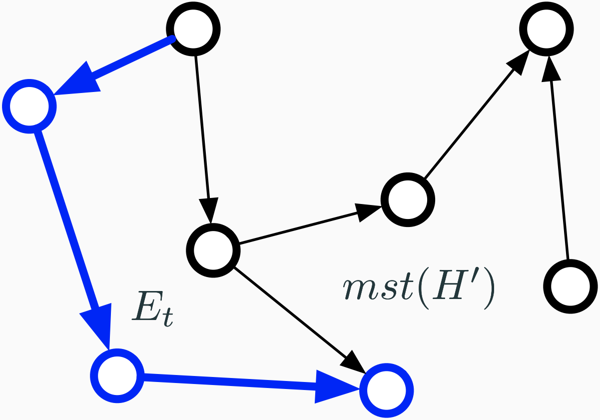


# Inductive step

Path case. Assume $E_t$ is a path and not a cycle.

$H' = (V(E_1 \cup \ldots \cup E_{t-1}), E_1 \cup \ldots \cup E_{t-1})$.

$\text{mst}(H) \leq \text{mst}(H') + w'(E_t) - \max_{e \in E_t} w'(e)$.

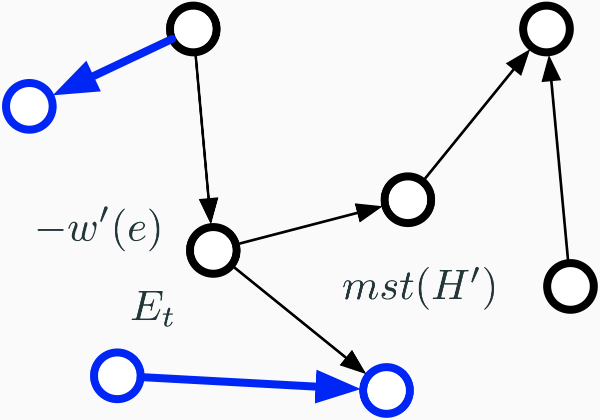


# Inductive step

Path case. Assume $E_t$ is a path and not a cycle.

$H' = (V(E_1 \cup \ldots \cup E_{t-1}), E_1 \cup \ldots \cup E_{t-1})$.

$\text{mst}(H) \leq \text{mst}(H') + w'(E_t) - \max_{e \in E_t} w'(e)$.



# Inductive step. cont.

$$\begin{aligned}
\mathrm{mst}(H) &\le \mathrm{mst}(H') + \sum_{e \in E_t} w'(e) - \max_{e \in E_t} w'(e) \\
&\le (|V(H')| - 2) + \sum_{e \in E_t} w'(e) - \max_{e \in E_t} w'(e) \\
&= (|V(H')| - 2) + (|V(E_t)| - w_1(u) - w_0(v)) - \max_{e \in E_t} w'(e) \\
&= (|V(H)| - 2) + 2 - w_1(u) - w_0(v) - \max_{e \in E_t} w'(e).
\end{aligned}$$

We have to show that $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \ge 2$.

# Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

## Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

- $w_1(u) = 1$, one of the edges containing $u$ has weight at least 1.

# Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

- $w_1(u) = 1$, one of the edges containing $u$ has weight at least 1.
- Similarly for $w_0(v) = 1$.

## Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

- $w_1(u) = 1$, one of the edges containing $u$ has weight at least 1.
- Similarly for $w_0(v) = 1$.
- $w_1(u) + w_0(v) = 0$, then $E_t$ is a path from a input depot to a output depot, hence there exists an edge of weight 2 in $E_t$.

## Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

- $w_1(u) = 1$, one of the edges containing $u$ has weight at least 1.
- Similarly for $w_0(v) = 1$.
- $w_1(u) + w_0(v) = 0$, then $E_t$ is a path from a input depot to a output depot, hence there exists an edge of weight 2 in $E_t$.

The case where $E_t$ is a cycle is similar.

## Inductive step. cont.

Prove that: $w_1(u) + w_0(v) + \max_{e \in E_t} w'(e) \geq 2$.

- $w_1(u) = 1$, one of the edges containing $u$ has weight at least 1.
- Similarly for $w_0(v) = 1$.
- $w_1(u) + w_0(v) = 0$, then $E_t$ is a path from a input depot to a output depot, hence there exists an edge of weight 2 in $E_t$.

The case where $E_t$ is a cycle is similar. This completes the proof.

# What about only input depots?

**Theorem**
*Let $H$ be a $k$-vertex structure graph with only input depots, then* $\mathrm{mst}(H) \leq k - 1$.

# What about only input depots?

**Theorem**
*Let $H$ be a $k$-vertex structure graph with only input depots, then* $\text{mst}(H) \leq k - 1$.

Same proof by induction on ear decomposition. The base case is a single cycle $C$, where $\text{mst}(C) = k - 1$, the rest of the proof follows.

# Variations

- What if the dist is not a metric?

# Variations

- What if the dist is not a metric? Use a shortest path metric $\text{dist}'$ instead.

## Variations

- What if the dist is not a metric? Use a shortest path metric $\text{dist}'$ instead.
- What if the machine can start only in locations $L_{start}$, and end in a set of locations $L_{end}$?

# Variations

- What if the dist is not a metric? Use a shortest path metric dist′ instead.
- What if the machine can start only in locations $L_{start}$, and end in a set of locations $L_{end}$? Simple transformation to the case where machine start and end at same position.

# Variations

- What if the dist is not a metric? Use a shortest path metric dist′ instead.
- What if the machine can start only in locations $L_{start}$, and end in a set of locations $L_{end}$? Simple transformation to the case where machine start and end at same position.
- What if each input request can only be completed by a particular input depot?

# Variations

- What if the dist is not a metric? Use a shortest path metric dist′ instead.
- What if the machine can start only in locations $L_{start}$, and end in a set of locations $L_{end}$? Simple transformation to the case where machine start and end at same position.
- What if each input request can only be completed by a particular input depot? Remove edges from depots to the request in the warehouse network, compute a new metric, and use the new metric to construct the warehouse network.

## Some ongoing work

- Output requests only, but a machine can hold two item at a time.

## Some ongoing work

- Output requests only, but a machine can hold two item at a time. Solvable in polynomial time if the metric is symmetric. [Xu, Yang, Zhang Unpublished]

# Some ongoing work

- Output requests only, but a machine can hold two item at a time. Solvable in polynomial time if the metric is symmetric. [Xu, Yang, Zhang Unpublished]
- Multiple machines.

## Some ongoing work

- Output requests only, but a machine can hold two item at a time. Solvable in polynomial time if the metric is symmetric. [Xu, Yang, Zhang Unpublished]
- Multiple machines. Solvable in polynomial time if number of empty depot is constant. [Unpublished]

## Some ongoing work

- Output requests only, but a machine can hold two item at a time. Solvable in polynomial time if the metric is symmetric. [Xu, Yang, Zhang Unpublished]
- Multiple machines. Solvable in polynomial time if number of empty depot is constant. [Unpublished]
- Each request is a set of locations.

## Some ongoing work

- Output requests only, but a machine can hold two item at a time. Solvable in polynomial time if the metric is symmetric. [Xu, Yang, Zhang Unpublished]
- Multiple machines. Solvable in polynomial time if number of empty depot is constant. [Unpublished]
- Each request is a set of locations. Unknown status, preliminary work with Madan and Shen.

Thank you!