Finding Large Induced Subgraphs

Hemanshu Kaul

Illinois Institute of Technology
www.math.iit.edu/~kaul
kaul@math.iit.edu

Joint work with
S. Kapoor, M. Pelsmajer (IIT)
Largest Induced Subgraph

Given a graph $G$, find the largest induced subgraph of $G$.

Ah!
Given a graph $G$, find the densest induced subgraph of $G$.

**Goldberg, 1984**: Polynomial-time algorithm based on network flows.
Densest $k$-Subgraph Problem

Given a graph $G$, find the largest induced subgraph of $G$ on $k$-vertices.
Densest $k$-Subgraph Problem

Given a graph $G$, find the largest induced subgraph of $G$ on $k$-vertices.

- NP-hard even on Chordal graphs (Corneil, Perl, 1984), on Planar graphs (Keil, Brecht, 1991)
- $n^{1/3}$-approximation algorithm (Feige, Kortsarz, Peleg, 2001)
- No PTAS in general under a complexity assumption (Khot, 2004)
An Extremal Problem

Given graph on $n$ vertices and $m$ edges, what can be said about the maximum number of edges in an induced subgraph on $k$ vertices within any such graph.

$$f(n, m, k) = \max l \ 	ext{such that every graph on } n \text{ vertices and } m \text{ edges has an induced subgraph on } k \text{ vertices with at least } l \text{ edges.}$$

Studied since 1970s, Chung-Erdős-Spencer (1985) gave a complete description of $f$ when $k = o(n)$, e.g.,

- $f(n, m, k) = \Theta(k^2 m/n^2)$ when $k > (n^2 \log n)/m$
- $f(n, m, k) = \Theta(k \log n/ \log(n^2/km))$
  when $k < (n^2/m) \log n \log \log n$
We are interested in a weighted version of the densest $k$-subgraph problem.

Given a graph $G$ with cost associated with each of its vertices, and benefit associated with each of its edges and vertices.

Find the induced subgraph whose cost doesn’t exceed a given budget while its total benefit is maximized.
Graph Knapsack Problem: Given an instance $GKP(G, b, w, W)$, where $G = (V, E)$ is an undirected graph with $n$ vertices, $w : V \rightarrow \mathbb{Z}^+$ is a weight function, $b : E \cup V \rightarrow \mathbb{Z}$ is a benefit function on vertices and edges, and $W$ is a weight bound.

$$\text{maximize } b(G[S])$$
$$\text{such that }$$
$$\text{weight}(S) \leq W$$
Graph Knapsack Problem

For a graph $G$ defined on $V$, the benefit of a subgraph $H = (V_H, E_H)$ is

$$b(H) = \sum_{v \in V_H} b(v) + \sum_{e \in E_H} b(e)$$

while its weight is

$$w(H) = \sum_{v \in V_H} w(v).$$
Graph Knapsack Problem

Given a subset of vertices $S$, we consider the subgraph induced by $S$, termed $G[S]$.

The Graph Knapsack Problem (GKP) asks for a subset of vertices, $S \subseteq V$ so as to maximize the benefit of the induced subgraph, $b(G[S])$ with the budget restriction that its weight $w(G[S])$ is less than $W$. 
Graph Knapsack Problem

Relationship to Large Subgraph Problems

GKP is related to the maximum clique problem. We can reduce the clique problem to the graph-knapsack problem.

Given a graph $G$, suppose we wish to determine if $G$ contains a clique of size $t$. We define an instance of GKP on $G$ with $W = t$, $w_i = 1$, $b_i = 0$, $b_e = 1$ for $e \in E(G)$. Graph $G$ has a $K_t$ iff GKP has benefit at least $\binom{t}{2}$.

We may note that, unless $P = NP$, achieving an approximation ratio better than $n^{1-\epsilon}$ is impossible for the clique problem.
Graph Knapsack Problem

Relationship to Large Subgraph Problems

GKP also generalizes the Densest $k$-Subgraph problem.

This corresponds to GKP with edges of benefit 1 while vertices have zero benefit, and the weight of each vertex is 1 with $W = k$. 
Graph Knapsack Problem

Relationship to Large Subgraph Problems

GKP also generalizes the Densest $k$-Subgraph problem.

This corresponds to GKP with edges of benefit 1 while vertices have zero benefit, and the weight of each vertex is 1 with $W = k$.

- NP-hard even on Chordal graphs (Corneil, Perl, 1984), on Planar graphs (Keil, Brecht, 1991)
- $n^{1/3}$-approximation algorithm (Feige, Kortsarz, Peleg, 2001)
- No PTAS in general under a complexity assumption (Khot, 2004)
Selection of Projects: We want to pick a collection of projects such that their combined “benefit” is maximized while the total cost does not exceed the given budget.

Previous Research: Choose projects such that the chosen projects have largest sum of individual benefits while their total cost does not exceed the total budget, $W$. This is simply the classical 0-1 Knapsack problem.

$$\max \sum_{i=1}^{n} B(i)x_i$$

subject to

$$\sum_{i} w_i x_i \leq W$$

$$x_i \in \{0, 1\}$$
GKP: Resource Allocation under Dependencies

**Criticism:** We have to choose multiple projects for implementation simultaneously, which means that such projects cannot be considered independent of each other. It may happen that two projects which are individually beneficial, will together negate either of their benefits.

The overall benefits of a collection of projects may be greater than, equal to, or smaller than the sum of individual benefits.

Think of highway projects on a transportation network.
GKP: Resource Allocation under Dependencies

Given a set of items \( V = \{v_1, ..., v_n\} \) (projects) and a knapsack of limited capacity \( W \) (the budget).

To each item we associate a benefit \( b(v_i) \) (benefit of that project) and a positive weight \( w_j \) (cost of that project).

To each pair \( (r = 2) \) of items we associate a benefit \( b(v_i, v_j) \).
\[
b(e) = b(uv) = B(u, v) - (B(u) + B(v)),
\]
where \( B \) is the benefit function. The benefits on the edges could be positive or negative. \( b(e) = 0 \) corresponds to no interdependency between projects.
GKP: Resource Allocation under Dependencies

We can formulate the problem as a 0-1 Quadratic Program:

\[
\begin{align*}
\text{maximize } & \sum_i b(v_i)x_i + \sum_{v_iv_j \in E(G)} b(v_iv_j)x_ix_j \\
\text{such that } & \sum_j w(v_j)x_i \leq W \\
x_i & \in \{0, 1\}
\end{align*}
\]

Replacing the term \(x_ix_j\) by an integer variable \(x_{ij} \in \{0, 1\}\) and adding the constraints \(x_{ij} \leq \frac{x_i + x_j}{2}\) and \(x_{ij} \geq \frac{x_i + x_j - 1}{2}\) provides an integer linear program (ILP) for the problem.
When $r > 2$, the underlying structure considers $r$-wise dependencies, that is it forms a $r$-uniform hypergraph.

The definitions given above generalize in a straightforward manner to the Hypergraph Knapsack Problem (HKP).

Let $H = (V, E)$ be a hypergraph. For any subset $S$ of vertices in $H$, let $w(S) = \sum_{v \in S} w(v)$ and let $b(S) = \sum_{v \in S} b(v) + \sum_{e \in E : e \subseteq S} b(e)$.

As before HKP asks for a subset of vertices $S$ that maximizes the benefit with the restriction that its weight is less than $W$. 
The **Quadratic Knapsack Problem (QKP)** is the appropriate problem for comparison with GKP. They are essentially the same problem when benefits are non-negative.

\[
\text{maximize } \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} x_i x_j \\
\text{such that } \sum_{i=1}^{n} w_i x_i \leq W \\
x_i \in \{0, 1\}
\]
No approximation algorithms or FPTAS are known for the general QKP. The focus has been on IP-based exact methods.

**Rader and Woeginger (2002)** developed a FPTAS for the case when all benefits are non-negative and the underlying graph is a series-parallel graph. They also show that when QKP has both negative and non-negative benefits, it can not have a constant factor approximation unless $P = NP$.

Note that the Hypergraph Knapsack Problem (HKP) can **not** be reduced to the QKP or some version of it.
Generalized Knapsack Problems

Relationship to other Knapsack Problems

The idea of using discrete structures like graphs, digraphs, posets to generalize the classical knapsack problem by modeling some sort of dependency among the items is not a new one.

However all such generalizations of the Knapsack problem restrict the choice of subset of items that can be picked. While our model does not restrict the choices directly, instead it modifies the benefit function so that the benefit on the edge between a pair of items could act as a penalty (if its negative) or an inducement (if its positive) towards the choice of those two items.
Generalized Knapsack Problems

Relationship to other Knapsack Problems

The **Knapsack Problem with Conflict Graph** is a knapsack problem where each edge in the underlying conflict graph on the items introduces the constraint that at most one of those two items can be chosen. This can be modeled as the Graphical Knapsack problem by putting large negative benefit on the edges of the conflict graph and using that as the underlying graph for GKP.
Generalized Knapsack Problems

Relationship to other Knapsack Problems

The **Constrained Knapsack Problem** in which dependencies between items are given by a graph. In the first version, an item can be selected only if at least one of its neighbors is also selected. In the second version, an item can be selected only when all its neighbors are also selected. These can also be modeled as GKP.

Also, **Precedence-Constrained Knapsack Problem**, **Subset-Union Knapsack**, etc.
Greedy Algorithm

Fix an integer $t$. The greedy algorithm can be defined naturally as:

1. Initialize $S = \emptyset$
2. Pick a subset $T$ of $V(G) - S$ of cardinality at most $t$ such that its benefit (the sum of the benefits of the vertices and edges induced by $T$ in $S \cup T$) to weight ratio is highest
3. Update $S = S \cup T$ if weight of $S \cup T$ satisfies the budget constraint, and then go to step 2. Otherwise pick whichever of $S$ or $T$ has larger benefit as the final solution.

When $t = 1$, the worst case benefit ratio can be made arbitrarily bad.
Greedy Algorithm

The difficulty in analyzing this greedy algorithm:

- Handling two kinds of “weights”
- Each step depends on partial solution from previous steps in an involved manner due to edges that go across.

Main idea:

- An arbitrary instance of GKP with greedy solution $A$ and optimal solution $O$ defines a new instance of GKP which has disjoint greedy and optimal solutions with its greedy solution same as $A$ and benefit of its optimal solution no worse than $b(O)$.
- Apply averaging arguments on this new instance, and use the disjointness of the two solutions and their relation to original instance to get the bound on the ratio of original benefits.
The greedy algorithm is a \((16 \min(n, W)/t)\)-factor polynomial time \((O(2^{t+1}\binom{n+1}{t+1}))\)-running time) approximation algorithm for \(GKP(G, b, w, W)\) with \(n\) vertices, when \(b\) is a non-negative function.

This analysis is sharp.
We can construct a family of instances of GKP where ratio of the optimal solution to the greedy solution is \(\Omega\left(\frac{n}{t}\right)\).

No such results are known for Quadratic Knapsack Problem.
The greedy algorithm is a \((2 \min(n, W)W/t)\)-factor polynomial time \((O(2^{t+1}(n+1))\)-running time) approximation algorithm for \(GKP(G, b, w, W)\) with \(n\) vertices, when \(b\) can take both negative and non-negative values.

This analysis is sharp.

We can construct a family of instances of GKP where ratio of the optimal solution to the greedy solution is \(\Omega(\frac{n^2}{t})\) where \(W = \Theta(n)\).

Again, no such results are known for Quadratic Knapsack Problem.
Greedy Algorithm

Why the extra factor of $W$ when negative benefits are possible?

When benefits are non-negative, we can show that $w(v) \leq W/2$ for all $v$ which implies that $W/w(A) \leq 2$.

When benefits are negative, this ratio can be as bad as essentially $W$. 
Greedy Algorithm for Hypergraph Knapsack

The definition of the greedy algorithm works for Hypergraph Knapsack problem as well.

However, taking $t < r$ (where $r$ is the largest size of an edge in the underlying hypergraph) can make the worst case benefit ratio arbitrarily bad.
Greedy Algorithm for Hypergraph Knapsack

S. Kapoor, H. Kaul, and M. Pelsmajer, 2011

The greedy algorithm is a \( 16 \left( \frac{\min(n,W)}{t-r+1} \right)^{r-1} \)-factor polynomial time \( O(2^t \binom{n}{t}) \)-running time) approximation algorithm for \( HKP(H, b, w, W) \) with \( n \) vertices and \( r \)-uniform edges, when \( b \) is a non-negative function.

This analysis is essentially sharp. We can construct a family of instances of HKP where ratio of the optimal solution to the greedy solution is \( \Omega\left( \frac{(n-r+1)^{r-1}}{t^{r-1}} \right) \).
The greedy algorithm is a \(2W \left( \frac{\min(n, W)}{t-r+1} \right)^{r-1} \)-factor polynomial time \(O(2^t \binom{n}{t})\)-running time approximation algorithm for \(HKP(H, b, w, W)\) with \(n\) vertices and \(r\)-uniform edges, when \(b\) can take both negative and non-negative values. This analysis is essentially sharp. We can construct a family of instances of HKP where ratio of the optimal solution to the greedy solution is \(\Omega\left(\frac{(n-r+1)^{r-1}n}{t^{r-1}}\right)\).
FPTAS for bounded tree-width graphs

S. Kapoor, H. Kaul, and M. Pelsmajer, 2011

Let $G$ be a graph with tree-width at most $k$. Then $GKP(G, b, w, W)$ can be approximated to within a factor of $(1 + \epsilon)$ in time $O\left(\frac{2^k n^9 \log n}{\epsilon^2}\right)$.

This result extends to HKP with hypergraph of bounded tree-width.
Both based on a pseudo-polynomial dynamic programming algorithm with lots of book-keeping.

Previous result: [Rader and Woeginger, 2002] FPTAS for QKP when the underlying graph is series-parallel, which is a family of graphs with tree-width 2.
Randomized Approximation Algorithm for GKP

S. Kapoor and H. Kaul, 2011+

A polynomial-time randomized algorithm that approximates GKP to the factor $O(n^{2/3} \log^4 n)$ when $b$ is non-negative.

Main Tools:

- Greedy Algorithm (allows us to assume $W > n^{2/3}$)
- Hyperbolic relaxation of GKP
- Chernoff-Hoeffding tail bounds
- Kim-Vu polynomial concentration
Randomized Approximation Algorithm for GKP

Solve the relaxation of the hyperbolic program \((HP^*)\) to get optimal solution \(x_u^*\)

\[
\begin{align*}
\text{maximize} & \quad \sum_{uv \in E(G)} b(uv)x_{uv} \\
\text{such that} & \quad \sum_i w(u)x_u \leq W \\
& \quad x_ux_v \geq x_{uv}^2
\end{align*}
\]

Generate a random 0-1 solution \(Y\), \(Y_u = 1\) with probability \(\sqrt{x_u^*/\lambda}\)
Choose a scaling factor \( \lambda \) so that \( E[w(Y)] \leq \lambda W \)

Use Chernoff-Hoeffding to show the concentration of the weight around its mean, so the budget can be satisfied w.h.p.
Randomized Approximation Algorithm for GKP

Define
\[ \varepsilon_0 = \mathbb{E}[b(Y)], \quad \text{i.e.,} \quad \frac{\text{OPT}(HP^*)}{\lambda}, \quad \text{a measure of global solution.} \]
\[ \varepsilon_1 = \max_v (\sum_{u \in N(v)} P[Y_u = 1]), \quad \text{a measure of dense local neighborhood solution.} \]
\[ \varepsilon_2 = \max_{uv \in E(G)} b(uv), \quad \text{a measure of most beneficial edge.} \]

J-H. Kim, Van Vu, 2001

\[
P[\mathbb{E}[Y] - Y > t^2] < 2e^2 \exp \left( -\frac{t}{32(2\varepsilon\varepsilon')^{1/4}} + \log n \right)
\]

where \( \varepsilon = \max\{\varepsilon_0, \varepsilon_1, \varepsilon_2\} \), and \( \varepsilon' = \max\{\varepsilon_1, \varepsilon_2\} \).
Randomized Approximation Algorithm for GKP

- When $\varepsilon_2 > \varepsilon_1$ and $\varepsilon_0 < \varepsilon_2 \log^4 n$, $\max b_{uv}$ works as a good solution.

- When $\varepsilon_2 > \varepsilon_1$ and $\varepsilon_0 > \varepsilon_2 \log^4 n$, Kim-Vu applies to $Y$, the randomized solution.

- When $\varepsilon_1 > \varepsilon_2$ and $\varepsilon_0 > \varepsilon_1 \log^4 n$, Kim-Vu applies to $Y$, the randomized solution.

- When $\varepsilon_1 > \varepsilon_2$ and $\varepsilon_0 < \varepsilon_1 \log^4 n$, a different randomized solution based on the dense local neighborhood solution: center of the neighborhood is picked with probability 1, neighbors are picked with same probability as before and nothing else is picked, which is shown to be concentrated via Chernoff-Hoeffding.