COMP473: Advanced Algorithms

Mark Dras

Department of Computing
Macquarie University

2007
Parameterized Complexity

Outline

1 Parameterized Complexity

2 Templates

3 Ant Colony System with Structure
Consider the following graph:
One observation

- note the ‘tail’: if we could solve the rest of the graph, we’d immediately know what to do with the tail
  - alternating vertices would go in the vertex cover
Another Observation

- similarly for triangles
  - two vertices from triangles need to go in the vertex cover
Parameterized Complexity

Triangles Removed
This idea of shrinking a problem is called kernelization.

Kernelization is a technique used in parameterized complexity for designing efficient exact algorithms.

- That is, the kernelization techniques reducing the graph preserve the solution.
- Kernelization rules are developed with accompanying formal proofs.

Kernelization algorithms typically involve a kernelization stage followed by some other methodology.
In shrinking the problem, Parameterized Complexity uses the notion of a parameter.

Here, we’ll be looking at the $k$-vertex cover problem: a decision problem asking whether there’s a vertex cover of size $k$ or not.

Parameterized complexity is an expansion upon regular complexity, providing a framework for dealing with computational intractability.

Parameterized complexity also provides tools for helping us deal with problems of complexity similar to the $k$-vertex cover problem.
Each natural problem may have many corresponding parameterized problems.
Parameters are naturally occurring within their natural problem.
Examples for vertex cover:
- Maximum size of solution.
- Maximum degree within graph.
- Minimum degree within graph.
- Maximum number of edits allowed per change.
Three Kernelization algorithms

- Buss’s Algorithm
- Balasubramanian, Fellows, and Raman
- Downey, Fellows, and Stege
Parameterized Complexity

Buss’s Algorithm

- Parameter $k$ is the size of the desired vertex cover
- Split algorithm into two parts:
  1. Reduce graph to a problem kernel of size $O(k^2)$
  2. Use brute force search to solve the reduced problem instance
Buss’s Algorithm

The following algorithm assumes that the graph representation is given as an adjacency list:

1. Find a set $H$ of all vertices of degree more than $k$ in $G$. Let $|H| = b$. If $b > k$ then answer no. Otherwise include $H$ in the vertex cover, remove it and the edges incident on them from $G$. Let $k' = k - b$. Remove any resulting isolated vertices.

2. If the resulting graph has more than $kk'$ edges, then answer no.

3. Find by brute force whether the resulting graph has a vertex cover of size $k'$. If so then answer yes. Otherwise answer no.
If $G$ has a vertex $v$ of degree greater than $k$, then replace $(G, k)$ with $(G - v, k - 1)$ and place $v$ in the vertex cover. Suppose that the parameter $k$ for this problem is 4.
Kernelization Rule 1

[Diagram of a graph with a central node labeled 'V' and several other nodes connected to it.]
Kernelization Rule 1
Buss’ Algorithm

- Kernelization rule reduces the problem to a problem kernel such that it has at most $kk'$ edges and at most $2kk'$ nodes.
- Runs in time $O(kn + 2^k)$. 
The kernelization stage involves placing in the vertex cover:

- all vertices whose degree is greater than $k$.
- all vertices which have at least one neighbour with a degree of 1.
- all vertices $y$ and $z$ which are connected to each other and another vertex $v$ that has a degree of 2.
If $G$ has a pendant edge $uv$ with $u$ having degree 1, then replace $(G, k)$ with $(G - \{u, v\}, k - 1)$ and place $v$ in the vertex cover.
Kernelization Rule 2
Kernelization Rule 2
Kernelization Rule 3

If $G$ has a vertex $u$ of degree 2, with neighbours $y$ and $z$, and $y$ and $z$ are adjacent, then replace $(G, k)$ with $(G - \{u, y, z\}, k - 2)$ and place $y$ and $z$ in the vertex cover.
Kernelization Rule 3
Followed by bounded search tree.

Runs in time $O(kn + 1.324718^k k^2)$. 
(0): If $G$ has a vertex $v$ of degree greater than $k$, then replace $(G, k)$ with $(G - v, k - 1)$.

(1): If $G$ has two nonadjacent vertices $u, v$ such that $|N(u) \cup N(v)| > k$, then replace $(G, k)$ with $(G + uv, k)$.

(2): If $G$ has adjacent vertices $u$ and $v$ such that $N(v) \subseteq N[u]$, then replace $(G, k)$ with $(G - u, k - 1)$.

(3): If $G$ has a pendant edge $uv$ with $u$ having degree 1, then replace $(G, k)$ with $(G - \{u, v\}, k - 1)$. 
(4): If $G$ has a vertex $x$ of degree 2, with neighbours $a$ and $b$, and none of the above cases applies (and thus $a$ and $b$ are not adjacent), then replace $(G, k)$ with $(G', k)$ where $G'$ is obtained from $G$ by:

- Deleting the vertex $x$
- Adding the edge $ab$
- Adding all possible edges between \{a, b\} and $N(a) \cup N(b)$.

(5): If $G$ has a vertex $x$ of degree 3, with neighbours $a$, $b$, $c$, and none of the above cases applies, then replace $(G, k)$ with $(G', k)$ according to one of the following cases depending on the number of edges between $a$, $b$ and $c$.

(5.1): There are no edges between the vertices $a$, $b$, $c$. In this case $G'$ is obtained from $G$ by:

- Deleting vertex $x$ from $G$.
- Adding edges from $c$ to all the vertices in $N(a)$.
- Adding edges from $a$ to all the vertices in $N(b)$.
- Adding edges from $b$ to all the vertices in $N(c)$.
- Adding edges $ab$ and $bc$.

(5.2): There is exactly one edge in $G'$ between the vertices $a$, $b$, $c$ which we assume to be the edge $ab$. In this case $G'$ is obtained from $G$ by:

- Deleting vertex $x$ from $G$.
- Adding edges from $c$ to all the vertices in $N(a) \cup N(b)$.
- Adding edges from $a$ to all the vertices in $N(c)$.
- Adding edges from $b$ to all the vertices in $N(c)$.
- Adding edge $bc$.
- Adding edge $ac$. 
Kernelization Rule 4

If $G$ has adjacent vertices $u$ and $v$ such that $N(v) \subseteq N[u]$, then replace $(G, k)$ with $(G - u, k - 1)$ and place $u$ in the vertex cover.
Kernelization Rule 4
Kernelization stage followed by a bounded search tree stage.
Kernelization also integrated into the bounded search tree stage.
Runs in time $O(1.31951^k k^2 + kn)$ and is feasible for values of $k \leq 60$. 
Parameterized Complexity

Two NP-complete Problems

- **k - Vertex Cover Problem**
  - Best algorithm runs in time $O(1.271^k k^2 + kn)$.
  - Useful for values of $k \leq 400$.

- **k - Dominating Set Problem**
  - Best algorithm runs in time $O(n^{k+1})$.
  - Still generally intractable.

### Table: The ratio $\frac{n^{k+1}}{2^k n}$ for various values of $n$ and $k$. 

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
<th>$k = 5$</th>
<th>$k = 10$</th>
<th>$k = 20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>625</td>
<td>15,625</td>
<td>390,625</td>
<td>$1.9 \times 10^{12}$</td>
<td>$1.8 \times 10^{26}$</td>
</tr>
<tr>
<td>100</td>
<td>2,500</td>
<td>125,000</td>
<td>6,250,000</td>
<td>$9.8 \times 10^{14}$</td>
<td>$9.5 \times 10^{31}$</td>
</tr>
<tr>
<td>150</td>
<td>5,625</td>
<td>421,875</td>
<td>31,640,625</td>
<td>$3.7 \times 10^{16}$</td>
<td>$2.1 \times 10^{35}$</td>
</tr>
</tbody>
</table>
Algorithms like those we have for the \( k \)-Vertex Cover Problem.

**DEFINITION:** A parameterized language \( L \) is multiplicatively fixed-parameter tractable if it can be determined in time \( f(k)q(n) \) whether \((x, k) \in L\), where \(|x| = n\), \(q(n)\) is a polynomial in \(n\), and \(f\) is a function (unrestricted).

**DEFINITION:** A parameterized language \( L \) is additively fixed-parameter tractable if it can be determined in time \( f(k) + q(n) \) whether \((x, k) \in L\), where \(|x| = n\), \(q(n)\) is a polynomial in \(n\), and \(f\) is a function (unrestricted).
W[1] and XP

- W[1] complexity class is an example of computational intractability within parameterized complexity.
- W[1] contains all the problems where there is a parametric transformation from them to the Weighted 3 CNF SAT problem.
- XP contains all the problems that are solvable in polynomial time for fixed $k$ without making our central distinction about whether this “fixed $k$” is ending up in the exponent or not.
- The W-Hierarchy

$$FPT \subseteq W[1] \subseteq XP$$ (1)
Kernelization is a tool for designing FPT algorithms.

A parameterized language $L$ is fixed-parameter tractable if and only if it is kernelizable.

Kernelization is about reducing a problem to its problem kernel.

The problem contains enough information to find an optimal solution yet is reduced such that the complexity of the problem is bounded by the parameter $k$. 
<table>
<thead>
<tr>
<th>Bipartite Matching Cardinality</th>
<th>Minimum Disjunctive Normal Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossing Number For Max Degree 3 Graphs</td>
<td>Minimum Fill-In</td>
</tr>
<tr>
<td>Cutwidth</td>
<td>Minor Order Test</td>
</tr>
<tr>
<td>$\prod_{i,j,k}$ Graph Modification Problem</td>
<td>More Than Half Max-3SAT</td>
</tr>
<tr>
<td>Diameter Improvement For Planar Graphs</td>
<td>Multidimensional Matching</td>
</tr>
<tr>
<td>Dual Of Coloring</td>
<td>$k$—Perfect Matchings</td>
</tr>
<tr>
<td>Dual of Irredundant Set</td>
<td>Nearly A Partition</td>
</tr>
<tr>
<td>Planar Diameter Improvement</td>
<td>Planar Face Cover</td>
</tr>
<tr>
<td>Planar Dominating Improvement Number</td>
<td>Pure Implicational Satisfiability Of Fixed F-Depth</td>
</tr>
<tr>
<td>Disjoint Paths</td>
<td>Rectilinear Picture Compression</td>
</tr>
<tr>
<td>Feedback Vertex Set</td>
<td>Restricted Alternating Hitting Set</td>
</tr>
<tr>
<td>Gate Matrix Layout</td>
<td>Search Number</td>
</tr>
<tr>
<td>Graph Genus</td>
<td>Set Basis</td>
</tr>
<tr>
<td>Grouping By Swapping</td>
<td>Short 3 Dimensional Matching</td>
</tr>
<tr>
<td>Hitting Set For Size Three Sets</td>
<td>Shortest Common Supersequence II</td>
</tr>
<tr>
<td>Induced Minor Testing For Planar Graphs</td>
<td>Steiner Tree</td>
</tr>
<tr>
<td>Long Cycle</td>
<td>Steiner Problem in Hypercubes</td>
</tr>
<tr>
<td>$k$—Leaf Spanning Tree</td>
<td>Treewidth</td>
</tr>
<tr>
<td>Bounded Character State Perfect Phylogeny</td>
<td>Unique Hitting Set</td>
</tr>
<tr>
<td>Matrix Domination</td>
<td>Vertex Cover</td>
</tr>
</tbody>
</table>
1. Parameterized Complexity

2. Templates

3. Ant Colony System with Structure
Templates have been applied to e.g. the data analysis problem and the graph partitioning problem.

- Data analysis and graph partitioning both involve ants picking up objects and dropping them elsewhere.
- Both algorithms have formulas determining the probability of picking up or dropping an object.
- These formulas have been modified by incorporating the template into the calculation of these formulas.
- The template is a probability map that covers the entire problem space specifying the desire for an object to be picked up or placed there.

The idea is that template mechanisms are combined with self-organization mechanisms such that the system self organizes along the template.
Kernelization rules reduce a problem to its problem kernel.

This knowledge of problem structure would provide a good map for directing self-organisation.

If we can find a way to combine self-organisation and kernelization, then kernelization rules provide a rich and readily available methodology for designing templates for combinatorial optimization problems.
Ant Colony System with Structure

Outline

1. Parameterized Complexity

2. Templates

3. Ant Colony System with Structure
Our Kernelization Rule

- Four rules in one
- Our single kernelization rule is:

\[
\text{If } G \text{ has adjacent vertices } u \text{ and } v \text{ such that } N(v) \subseteq N[u], \text{ then replace } (G, k) \text{ with } (G - u, k - 1) \text{ and place } u \text{ in the vertex cover.}
\]
Initial attempt was simply to kernelize a graph and then run Ant Colony System on the resulting kernel.

- We call this algorithm **Kernelized Ant Colony System**.

However, reducing a problem to its problem kernel is not always feasible, since it often involves removing nodes and edges.
Perhaps a better idea is to try placing pheromone on the nodes that should be included according to kernelization.

**Idea**: set pheromone on nodes indicated by kernelization rule (to 0.9).

- **PreKernelized Ant Colony System** — this kernelization is performed once in the initialisation phase.
- **CycleKernelized Ant Colony System** — this kernelization is performed every cycle as part of the global pheromone update rule.

However, these are just a variation on the idea of performing kernelization and ACO in sequence: kernelization as preprocessing

- Instead, use ants to kernelize during ACO
KernelAnts Ant Colony System

- This algorithm uses a further $m$ ants to kernelize the graph and place pheromone on the nodes included.
- This process happens in parallel with the original $m$ ants executing regular ACS.
- These kernelants set the pheromone on included nodes to 0.9.
TransKernelized Ant Colony System

A further advancement is to incorporate kernelization into the ants random proportional transition rule. For ant $k$ at city $i$, this rule chooses the city $j$ to visit next according to the rule:

$$
j = \begin{cases} 
\text{any node } u \text{ such that } u \in (J^k \cap \chi) & \text{if } q \leq q_0 \text{ and } |J^k \cap \chi| > 0; \\
\arg \max_{u \in J^k} \{[\tau_u(t)] \cdot [\eta_u]^\beta\} & \text{if } q \leq q_0 \text{ and } |J^k \cap \chi| = 0; \\
J & \text{if } q > q_0,
\end{cases}

(2)$$

where $\chi$ is the set of nodes to be included in the solution according to kernelization; $q$ is randomly selected from the distribution $[0, 1]$; $q_0$ is a tunable parameter such that $0 \leq q_0 \leq 1$; and $J \in J^k_i$ is a city that is randomly selected according to the probability:

$$p^k_{ij}(t) = \frac{[\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta}{\sum_{l \in J^k_i} [\tau_{il}(t)] \cdot [\eta_{il}]^\beta}

(3)$$
Neighbourhood TransKernelized Ant Colony System

The problem with TransKernelized Ant Colony System is that it requires a lot of kernelization on the fly.

An alternative is to pick a node using the normal random proportional transition rule and then try kernelizing all the neighbours of that node just to ensure that none of them make a better choice.

Since either the chosen node or all of its neighbours are to be included, it is safe to include this chosen node should none of its neighbours be kernelizable.
Evaluation

- On two kinds of data
  - Benchmark graphs: Benchmarks with Hidden Optimum Solutions for Graph Problems (Xu, 2005), size 450–1534 nodes
  - Random graphs: generated according to Skiena, size 100–1000 nodes
- Ran for length of time equal to 200 iterations of basic ACS
- Compared using Mann-Whitney U-test
Data sample

<table>
<thead>
<tr>
<th>Problem</th>
<th>SYL</th>
<th>ACS</th>
<th>PKACS</th>
<th>CKACS</th>
<th>KAACS</th>
<th>KACS</th>
<th>TKACS</th>
<th>NTKACS</th>
<th>Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>frb59-26-1</td>
<td>1496</td>
<td>1494</td>
<td>1492</td>
<td>1491</td>
<td>1488</td>
<td>1487</td>
<td>1493</td>
<td>1488</td>
<td>1475</td>
</tr>
<tr>
<td>frb59-26-2</td>
<td>1493</td>
<td>1492</td>
<td>1490</td>
<td>1490</td>
<td>1488</td>
<td>1490</td>
<td>1492</td>
<td>1491</td>
<td>1475</td>
</tr>
<tr>
<td>frb59-26-3</td>
<td>1494</td>
<td>1492</td>
<td>1493</td>
<td>1490</td>
<td>1487</td>
<td>1492</td>
<td>1494</td>
<td>1490</td>
<td>1475</td>
</tr>
<tr>
<td>frb59-26-4</td>
<td>1494</td>
<td>1492</td>
<td>1489</td>
<td>1490</td>
<td>1487</td>
<td>1491</td>
<td>1492</td>
<td>1489</td>
<td>1475</td>
</tr>
<tr>
<td>frb59-26-5</td>
<td>1494</td>
<td>1493</td>
<td>1492</td>
<td>1486</td>
<td>1487</td>
<td>1487</td>
<td>1491</td>
<td>1490</td>
<td>1475</td>
</tr>
</tbody>
</table>

- sum: 392798 391811 391050 391104 390351 390391 391456 390955 38690
- average: 981.995 979.527 977.625 977.76 975.877 975.977 978.64 977.387 967.25

- All kernelization versions better than basic ACS
- KAACS and KACS best
Data sample

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Edges</th>
<th>SYL</th>
<th>ACS</th>
<th>PKACS</th>
<th>CKACS</th>
<th>KAACS</th>
<th>KACS</th>
<th>TKACS</th>
<th>NTKACS</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>800</td>
<td>227</td>
<td>218</td>
<td>218</td>
<td>218</td>
<td>218</td>
<td>217</td>
<td>216</td>
<td>217</td>
</tr>
<tr>
<td>400</td>
<td>800</td>
<td>230</td>
<td>219</td>
<td>217</td>
<td>216</td>
<td>218</td>
<td>216</td>
<td>215</td>
<td>215</td>
</tr>
<tr>
<td>400</td>
<td>800</td>
<td>223</td>
<td>212</td>
<td>213</td>
<td>211</td>
<td>212</td>
<td>212</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>279</td>
<td>266</td>
<td>265</td>
<td>268</td>
<td>266</td>
<td>266</td>
<td>264</td>
<td>264</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>280</td>
<td>265</td>
<td>265</td>
<td>265</td>
<td>262</td>
<td>263</td>
<td>262</td>
<td>262</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>277</td>
<td>264</td>
<td>265</td>
<td>266</td>
<td>264</td>
<td>264</td>
<td>263</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td>99269</td>
<td>95598</td>
<td>95446</td>
<td>95445</td>
<td>95374</td>
<td>95381</td>
<td>95067</td>
<td>95008</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>198.538</td>
<td>191.196</td>
<td>190.892</td>
<td>190.89</td>
<td>190.748</td>
<td>190.762</td>
<td>190.134</td>
<td>190.016</td>
</tr>
</tbody>
</table>

All kernelization versions better than basic ACS
NTKACS best, then TKACS next