Permutation Diameter for Undirected Graphs

Definitions

Let Γ be a finite, connected, undirected graph with vertex set V, (conventionally $\{1, \cdots, n\}$) and edge set E. Let S_n denote the group of all permutations on V, and for $\sigma \in S_n$ we say that the set of i such that $i \neq \sigma(i)$ is **support** of σ, denoted by $supp(\sigma)$. Then $\tau \in S_n$ is said to be **Γ-compatible** provided $(i, \tau(i))$ belongs to E for all $i \in supp(\tau)$.

For $\sigma \in S_n$ we define the **Γ-height**, denoted $h_\Gamma(\sigma)$, to be the minimum k such that $\tau_1 \cdots \tau_k = \sigma$, for τ_i Γ-compatible. Such a sequence τ_i is called a **Γ-compatible expansion** of σ.

The **permutation diameter** of the graph Γ is the maximum Γ-height for all σ in S_n.

For $\sigma \in S_n$ we define its **Γ-weight**, denoted $w_\Gamma(\sigma)$, to be $\sum_i d(i, \sigma(i))$. Note that the weight is 0 iff $\sigma = I$.

For $i \neq j$ vertexes in Γ, consider the set of minimal paths from i to j, and denote by $SNSP_\Gamma(i,j)$ the set of second nodes (or first steps) in this set of minimal paths. Then we denote by $ssnsp_\Gamma(i,j)$ the element of $SNSP_\Gamma(i,j)$ with smallest index. (Note: This clearly depends on the indexing, and is not intrinsic to the graph.) Finally, for $\sigma \in S_n$, we denote by $Fl_\Gamma(\sigma)$ the directed subgraph of Γ whose edges go from i to $ssnsp_\Gamma(i, \sigma(i))$ for all $i \in supp(\sigma)$. We refer to this directed subgraph of Γ as the **flow graph** for σ.

Observations

The first thing to notice is that, since Γ in an undirected graph, τ is Γ-compatible iff τ^{-1} is. Next, if we consider cycles of τ, we see that transpositions correspond to edges of Γ, and the longer cycles of τ correspond to cycles of Γ. That is, the set of Γ-compatible permutations can be thought of as being formed from all products of disjoint edges and directed cycles of Γ.

Notice also that the permutation diameter of Γ is the diameter of another graph. If we take the Cayley graph on S_n with respect to the set of Γ-compatible elements, then the diameter of that graph is the permutation diameter of Γ.

One way to think of Γ-compatible permutations is as the set of permutations of fixed-size blocks of data that can occur in a single, synchronized, parallelized step in a network of processors with connection graph Γ. Therefore, the permutation diameter is an upper bound on how long it would take to do an arbitrary rearrangement of the data in this processor network.

It is easy to see that the permutation diameter for a connected graph is finite. If the diameter of Γ is Δ, then $h_\Gamma((ij))$ is at most $2\Delta - 1$. To see this, let $i = v_0 \cdots v_k = j$ be a path between i and j. Then (ij) is given by $(iv_1)(v_1v_2) \cdots (v_{k-1}j)(v_{k-1}v_{k-2}) \cdots (v_2v_1)(v_1i)$. Since every permutation in S_n can be written as a product of at most n transpositions, we see that the permutation diameter is at most $n(2\Delta - 1)$.

Constructing, for any $\sigma \in S_n$, a sequence τ_i of Γ-compatible permutations such that $\tau_k \cdots \tau_1 = \sigma$ is equivalent to producing a sequence of permutations such that σ, $\sigma\tau_1^{-1}$, $\sigma\tau_1^{-1}\tau_2^{-1}$, ..., $\sigma\tau_1^{-1}\tau_2^{-1} \cdots \tau_k^{-1} = I$. This means that a "good enough" algorithm for computing a Γ-compatible approximation of permutations can be readily converted into an algorithm for computing compatible expansions.

Finally, we make some observations about weights. If τ and σ belong to S_n, then $w(\sigma\tau^{-1}) = \sum_i dist(\tau(i), \sigma(i))$. (This is obvious since we are just reindexing the summation index from i to $\tau(i)$.) In particular, if τ is a

Γ-compatible permutation such that for every $i \in supp(\tau)$, $(i, \tau(i))$ is an edge in $Fl_\Gamma(\sigma)$, then $w(\sigma\tau^{-1})$ is $w(\sigma) - |supp(\tau)|$. Next, if τ is Γ-compatible transposition (ij) where (i, j) is an edge in $Fl_\Gamma(\sigma)$ and j is a fixed point of σ, then $w(\sigma\tau^{-1}) = w(\sigma)$. This is because $dist(\tau(i), \sigma(i))$ is $dist(i, \sigma(i)) - 1$, while $dist(\tau(j), \sigma(j)) = 1$, and no other terms in the sum are affected. (Obviously this equality is also true if τ is the product of disjoint transpositions of this type.) Finally, note that in this case that while $Fl_\Gamma(\sigma)$ has "an arrow" only at node i, $Fl_\Gamma(\sigma\tau^{-1})$ has arrows at both i and j (and at anyplace else that $Fl_\Gamma(\sigma)$ did). This is because the only affected nodes are i and j, and $\sigma\tau^{-1}(j) = \sigma(i) \neq j$ (because $i \neq j$ and j was chosen so that $\sigma(j) = j$). Also, $\sigma\tau^{-1}(i) = j \neq i$. The point of this that we can construct a new permutation with the same weight as the old one, but which has fewer fixed points.

Computing σ

The final observation above shows how to construct greedy algorithms for computing a Γ-compatible expansion of a given permutation. There are 3 basic operations that can be combined in various ways to do this:

Removing Cycles: If we compute the cycle decomposition of σ and find Γ-compatible cycles in it, then we can immediately remove those. This step reduces the weight by the sum of the cycle lengths found. Note: This is a treated as distinct step from removing virtual cycles, below, because it is probably much cheaper.

Removing Virtual Cycles: We can find all of the Γ-compatible cycles in $Fl_\Gamma(\sigma)$ and remove those. This step reduces the weight by the sum of the cycle lengths found, too.

Arrow Creation: We can find all of the arrows in $Fl_\Gamma(\sigma)$ that point to a fixed point of σ, and create arrows at those fixed points. This step doesn't change the weight.

A family of greedy algorithms can be constructed from this collection of operations. All that is required for such an algorithm to converge is to apply arrow creation at any point when there are no virtual cycles to remove. Since the only thing that can cause there to be no virtual cycles is that paths in $Fl_\Gamma(\sigma)$ must terminate in fixed points of σ, removing any such "sinks" makes it possible to reduce the degree. Presumably the choice of order for these operations and their precise implementation would affect the results of the algorithm, but I can no longer recall the details.