A substantial effort was put into Graph Theory for Maple 2023, including improved ability to solve traveling salesman problems, support for multigraphs, new commands for graph computation, and advances in visualization.

> with(GraphTheory):

### Traveling Salesman

The `TravelingSalesman` command now makes use of Concorde, a well-known library implementing highly efficient heuristics for solving instances of the traveling salesman problem. This addition considerably increases the size of problems which TravelingSalesman is able to handle.

### Example: Using TravelingSalesman with vertexpositions

In the following example, we import a dataset of the 100 largest cities in the continent of Africa (including the island of Madagascar) and ask TravelingSalesman to find a minimal tour of them.

> AfricanCities := Import("example/AfricanCities100.tsv", base = datadir, output = Matrix, skiplines = 1)

```
AfricanCities :=
   
   1 "Cairo" "Egypt"  22183200 30.044444  31.235833
   2 "Kinshasa" "Democratic Republic of the Congo" 16315534  325  15.322222
   3 "Lagos" "Nigeria" 15387639 6.455027  3.384082
   4 "Giza" "Egypt" 9200000  29.987  31.2118
   5 "Luanda" "Angola" 8952496  8.38333  13.234444
   6 "Dar es Salaam" "Tanzania" 7404689  6.81611  39.280278
   7 "Khartoum" "Sudan" 6160327  15.500556  32.56
   8 "Johannesburg" "South Africa" 6065354  26.20444  28.045556
   9 "Abidjan" "Côte d'Ivoire" 5515790  5.316667  4.033333
   10 "Alexandria" "Egypt" 5483605  31.1975  29.8925
```

100 × 6 Matrix
> \[ G := \text{CompleteGraph}( \text{convert(AfricanCities[...2, list]), vertexpositions = AfricanCities[...[6, 5]])} \]

\[ G := \text{Graph 1: an undirected graph with 100 vertices and 4950 edge(s)} \]

With the new vertexpositions option, TravelingSalesman uses edge weights computed from the geometric distance between vertices in the graph layout specified previously when CompleteGraph was called. The new startvertex option allows a particular starting vertex to be specified.

> \[ W, T := \text{TravelingSalesman}( G, \text{vertexpositions, startvertex = "Cairo"}) \]


We will illustrate the tour by constructing a subgraph consisting only of the edges included in the optimal tour across \( G \).
\( TG := \text{Subgraph} \left( G, \text{Trail}(T) \right) ; \)

\( TG := \text{Graph 2: an undirected graph with 100 vertices and 100 edge(s)} \)

\( \text{DrawGraph}(TG) \)
To better visualize the tour we can combine this with a country map of Africa.

```prolog
plots:-display(Import("example/AfricaMap.kml", base = datadir), DrawGraph(TG, stylesheet = [vertexcolor = black]));
```

Using TravelingSalesman with an arbitrary weight matrix

The previous example demonstrates the use of TravelingSalesman with edge weights corresponding directly to geometric distances between vertices. In many instances of the traveling salesman problem, the weights do not correspond so directly to the geometry.

Fortunately, TravelingSalesman can also compute a tour when given an arbitrary weight matrix. To illustrate this, let us extend the previous example of a tour of 100 African cities to incorporate the idea that there might be some increased cost associated with traversing an international border.
First let us get the matrix of Euclidean distances from the previous example:

\[
\begin{align*}
\text{DM} & := \text{WeightMatrix}(\text{MakeWeighted}(G, \text{vertexpositions}, \text{metric} = \text{Euclidean})) \\
\end{align*}
\]

Now we can construct a new weight matrix \( W_M \) in which the weight is doubled whenever the cities connected are in distinct countries.

\[
\begin{align*}
\text{WM} & := \text{Matrix}(\text{upperbound}(\text{DM}), (i, j) \text{ ifelse} (\text{AfricanCities}[i, 3] = \text{AfricanCities}[j, 3], \text{DM}[i, j], 2 \cdot \text{DM}[i, j]), \text{datatype} = \text{double}) \\
\end{align*}
\]

We can now pass the weight matrix \( W_M \) directly to \( \\text{TravelingSalesman} \) to get a new tour.

\[
\begin{align*}
W_2, T_2 & := \text{TravelingSalesman}(G, W_M, \text{startvertex} = "Cairo") \\
\end{align*}
\]

We can now pass the matrix \( W_M \) directly to \( \\text{TravelingSalesman} \) to get a new tour.
This new tour is similar to the previous one in many respects, but notably does not visit Sudan or the Democratic Republic of the Congo twice as the first one did, illustrating the effect of our increased edge weights.

\[
\text{TG2} := \text{Subgraph}(G, \text{Trail}(T2))
\]

\[
\text{TG2} := \text{Graph 3: an undirected graph with 100 vertices and 100 edge(s)}
\]

> plots:-display(Import("example/AfricaMap.kml", base = datadir, title = none),
  DrawGraph(TG2, stylesheet = [vertexcolor = black]))
Finally we can draw the original tour (in green) and the new tour (in red) to see the effect of the altered weights.

```
plots:-display([DrawGraph(TG, stylesheet = [edgecolor = "darkgreen"]),DrawGraph(TG2, stylesheet = [edgecolor = red])])
```

![Graphs showing the original and new tours](image)

### Multigraph support

GraphTheory now supports multigraphs. That is, graphs in which there may be multiple edges between the same pair of vertices.

```
MG := Graph(4,

```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```


```
MG := Graph 3: an undirected multigraph with 4 vertices and 5 edge(s)
```

The new command `IsMultigraph` tests whether a graph is a multigraph.

```
IsMultigraph(MG)
```

```
true
```

Many other commands have been updated to support multigraphs.

The `Edges` command for this graph returns a list, not a set, and repeats the edge between vertices 3 and 4 twice.

```
Edges(MG)
```

```
[ {1, 2}, {1, 2}, {2, 3}, {3, 4}, {3, 4} ]
```
Graph products

A graph product is a binary operation on graphs which takes two graphs $G_1$ and $G_2$ and produces a graph $H$ with the following properties:

- The vertex set of $H$ is the Cartesian product $V(G_1) \times V(G_2)$ where $V(G_1)$ and $V(G_2)$ are the vertex sets of $G_1$ and $G_2$, respectively.

$$
\begin{align*}
V(G_1) \\
V(G_2)
\end{align*}
$$

- Two vertices $(u_j, v_1)$ and $(u_2, v_2)$ of $H$ are connected by an edge, iff some condition about $u_j, v_1 \in G_1$ and $u_2, v_2 \in G_2$ is fulfilled.
Many different graph products have been defined which differ in the condition imposed on the edges. To the existing `CartesianProduct` and `TensorProduct` commands, the following new graph products have been added:

<table>
<thead>
<tr>
<th>Name</th>
<th>Edge Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConormalProduct</td>
<td>$u_1$ and $v_1$ share an edge in $G_1$ or $u_2$ and $v_2$ share an edge in $G_2$</td>
</tr>
<tr>
<td>LexicographicProduct</td>
<td>$u_1$ and $v_1$ share an edge in $G_1$, or $u_1 = v_1$ in $G_1$ and $u_2$ and $v_2$ share an edge in $G_2$</td>
</tr>
<tr>
<td>ModularProduct</td>
<td>$u_1$ and $v_1$ share an edge in $G_1$ and $u_2$ and $v_2$ share an edge in $G_2$, or $u_1$ and $v_1$ do not share an edge in $G_1$ and $u_2$ and $v_2$ do not share an edge in $G_2$</td>
</tr>
<tr>
<td>StrongProduct</td>
<td>$u_1 = v_1$ in $G_1$ and $u_2$ and $v_2$ share an edge in $G_2$, or $u_1$ and $v_1$ share an edge in $G_1$ and $u_2 = v_2$ in $G_2$, or $u_1$ and $v_1$ share an edge in $G_1$ and $u_2$ and $v_2$ share an edge in $G_2$</td>
</tr>
</tbody>
</table>

- $C := \text{CycleGraph}(3)$
  
  $C := \text{Graph 4: an undirected graph with 3 vertices and 3 edge(s)}$

- $H := \text{SpecialGraphs:-HouseGraph( )}$

  $H := \text{Graph 5: an undirected graph with 5 vertices and 6 edge(s)}$

- $LP := \text{LexicographicProduct}(C, H)$

  $LP := \text{Graph 6: an undirected graph with 15 vertices and 93 edge(s)}$
> plots:-display(DrawGraph~((C|H|L|P)))
Additions to SpecialGraphs
Maple 2023 provides support for 6 additional Special Graphs, bringing the total to 119.

> with(SpecialGraphs):

<table>
<thead>
<tr>
<th>Bishops's Graph</th>
<th>Bouquet Graph</th>
<th>Dipole Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG46 := BishopsGraph(4, 5)</td>
<td>BG := BouquetGraph(5)</td>
<td>DG := DipoleGraph(2)</td>
</tr>
<tr>
<td>( BG46 := )</td>
<td>( BG := )</td>
<td>( DG := )</td>
</tr>
<tr>
<td>( Graph 7: an )</td>
<td>( Graph 8: an )</td>
<td>( Graph 9: an )</td>
</tr>
<tr>
<td>( undirected )</td>
<td>( undirected )</td>
<td>( undirected )</td>
</tr>
<tr>
<td>( graph with 20 )</td>
<td>( multigraph with 1 )</td>
<td>( multigraph with 2 )</td>
</tr>
<tr>
<td>( vertices and )</td>
<td>( vertex, no )</td>
<td>( vertices )</td>
</tr>
<tr>
<td>( 40 edge(s) )</td>
<td>( edges, and 5 )</td>
<td>( and 2 edge(s) )</td>
</tr>
<tr>
<td>( )</td>
<td>( self-loops )</td>
<td>( )</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
1:5 & \quad 2:5 & \quad 3:5 & \quad 4:5 \\
1:4 & \quad 2:4 & \quad 3:4 & \quad 4:4 \\
1:3 & \quad 2:3 & \quad 3:3 & \quad 4:3 \\
1:2 & \quad 2:2 & \quad 3:2 & \quad 4:2 \\
1:1 & \quad 2:1 & \quad 3:1 & \quad 4:1 \\
\end{align*}
\]

\[
\begin{align*}
1 & \quad 2:5 & \quad 3:5 & \quad 4:5 \\
1 & \quad 2:4 & \quad 3:4 & \quad 4:4 \\
1 & \quad 2:3 & \quad 3:3 & \quad 4:3 \\
1 & \quad 2:2 & \quad 3:2 & \quad 4:2 \\
1 & \quad 2:1 & \quad 3:1 & \quad 4:1 \\
\end{align*}
\]

\[
\begin{align*}
5 & \quad 1 \\
1 & \quad 2 \\
2 & \quad 2
\end{align*}
\]
<table>
<thead>
<tr>
<th>Hamming Graph</th>
<th>House Graph</th>
<th>Windmill Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; HG33 := HammingGraph(3, 3)</td>
<td>&gt; HG := HouseGraph( ) HG := Graph 10: an undirected graph with 27 vertices and 81 edge(s)</td>
<td>&gt; WG := WindmillGraph(3, 7) WG := Graph 12: an undirected graph with 15 vertices and 21 edge(s)</td>
</tr>
<tr>
<td>Graph 10: an undirected graph with 27 vertices and 81 edge(s)</td>
<td>&gt; DrawGraph(HG, stylesheet = [vertexpadding = 12])</td>
<td>&gt; DrawGraph(WG, stylesheet = [vertexpadding = 10])</td>
</tr>
</tbody>
</table>

![Diagram of Hamming Graph HG33](image1)

![Diagram of House Graph HG](image2)

![Diagram of Windmill Graph WG](image3)