Vladimir Milenković

The Travelling Salesman problem - Comparison of different approaches

Diploma in Computer Science

Trinity College

23rd March 2020

Proforma

Name:	**Vladimir Milenković**
College:	**Trinity College**
Project Title:	**The Travelling Salesman problem - Comparison of different approaches**
Examination:	**Diploma in Computer Science, May 2020**
Word Count:	**TODO**[1]
Project Originator:	**Dr T. Sauerwald and V. Milenkovic**
Supervisor:	**Dr T. Sauerwald**

Original Aims of the Project

To code, compare and contrast different algorithms and approaches in solving one of the most famous problems ever to exist - the Travelling Salesman Problem. All the code done for this project is easy to use for anybody wishing to tackle this problem, and the overview of all the algorithms with the suggested usecase for each one is included as well.

Work Completed

All work that was done during this project will be mentioned in the dissertation, the code will be submitted as well.

Special Difficulties

Getting Lin-Kernighan algorithm to work significantly better than the other algorithms, reproducing the results mentioned on the original paper. Also, the

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Blossom algorithm for minimum weight perfect matching is hard to get working.

Declaration

I, Vladimir Milenkovic of Trinity College, being a candidate for Part II of the Computer Science Tripos , hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Vladimir Milenkovic

Date

Contents

List of Figures

Acknowledgements

Add some acknowledgements here. Please, do add some acknowledgements here.

Chapter 1

Introduction

1.1 Motivation

My project tries to tackle one of the most famous problems in the history of Computer science - the Travelling Salesman problem. In that problem, we are given a weighted graph $G = (V, E, w)$, where V denotes the set of vertices, E denotes a set of edges, and $w : E \to \mathbb{R}^+$ is a weight function, mapping edges to weight that's assigned to them. We are interesting in finding a minimum-weight cycle in this graph. As we can see, a problem similar to this can arise in many real-world situations (a mailman having to visit a list of houses to deliver the mail, for example), so it is expected that a lot of effort has been put into solving this problem as efficiently as possible. This problem is known to be $\mathcal{NP}$-hard, basically meaning that there is no polynomial time algorithm solving this problem. With that in mind, we are forced to release some requirements of our algorithm: not demanding that we get a fully correct solution, only sufficiently close to the solution, and/or not demanding to finish in polynomial time, but sufficient fast. This project will try to give a summary of some more or less successful attemps, and analyze what are the advangates and disadvantages of the algorithms involved.

1.2 Algorithms

In this section, I will list all the algorithms that I have taken into consideration for this project. I have made this choice after analyzing already existings benchhmarks about Travelling Salesman problem, trying to find the best tradeoff between complexity, running time, difficulty to reproduce and the result that the

algorithm is producing. After having that finished, I have decided to split the algorithms in several classes, listed below:

- **Exact algorithms**
- **Approximation algorithms**
- **Improvement algorithms**
- **Heuristic algorithms**
- `Optimization algorithms`

1.2.1 Exact algorithms

Exact algorithms are those algorithms which produce the correct solution every time they are run. Knowing the $\mathcal{NP}$-hardness of the TSP, we know that there is no algorithm of this kind running in polynomial time. I have considered 3 algorithms here:

- *Brute-force algorithm*
- *Held-Karp (dynamic programming) algorithm*
- *Branch-and-Bound algorithm*

1.2.2 Approximation algorithms

Approximation algorithms are ones that can guarantee some bound concerning the output of the algorithm. For example, the *2-approximation algorithm* guarantees that the cost of the cycle it returned is at most two times larger than the cost of the optimal cycle. They usually run in polynomial time, and all the algorithms I've included in the project do belong in that category.

- *2-approximation algorithm*
- *Christofides-greedy algorithm*
- *Christofides algorithm*

1.2.3 Improvement algorithms

Improvement algorithms are those algorithm which consecutively try to improve an existing solution. Starting with some solution (greedy, for example), they try to alter it in a way that a better solution is produced. Running this algorithm for some time will eventually make the algorithm converge, getting a solution which we take as a final one. In practice, combining improvement techniques with some good strategies for the initial solution produce the best results, and a lot of best solutions on well-known TSP instances are obtained using improvement methods.

- *2-opt algorithm*
- *3-opt algorithm*
- *Lin-Kernighan algorithm*

1.2.4 Heuristic algorithms

Heuristic algorithms are algorithms which usually have polynomial running time, usually do not have a guaranteed bound of error, but in general they produce results that are close to the optimal solution. There are a lot of different heuristic for tackling this problem, so I have selected a few that did produce some solid results. Also, these algorithms should not be too hard to code or to understand, in general, and it is surprising how good results can they achieve:

- *Random algorithm*
- *Nearest neighbour algorithm (greedy)*
- *Insertion heuristic - cheapest, farthest, nearest, random*
- *Convex-hull heuristic algorithm*

1.2.5 Optimization algorithms

In **optimization algorithms**, we are starting with random solutions, computing the results those solutions are producing and then adapting towards solutions which are more preferable (have less cost in this example). Running this iteratively, we converge to a solution which can be pretty close to the correct one, no guarantees shown. There are also quite some algorithms which perform the process outlined above, here are some:

- *Ant-colony optimization algorithm*
- *Simulated annealing algorithm*

Chapter 2

Preparation

In this chapter, I will describe, in full detail, all the algorithms and all the theoretical background needed to start the implementation part of the project, as well as some lemmas and proofs about working performance of the procedures described.

2.1 Complexity analysis

I have already mentioned the fact that the Travelling Salesman Problem can not be solved in polynomial time. Now, we would like to see in which class does this problem belong to. In order to do that, we do not need the fully detailed definitions of them because they are not the main point of this dissertation - so I will give a brief and informal definition of complexity classes we take interest in here:

- $\mathcal{P}$ - the set of all problems which can be correctly solved in polynomial time.
- $\mathcal{NP}$ - the set of all problems whose *solutions* can be *verified* in polynomial time.
- $\mathcal{NP}$**-hard** - all the problems which can't be solved in polynomial time. This definition is very informal, but it suffices here.
- $\mathcal{NP}$**-complete** - problems which are both in $\mathcal{NP}$ and $\mathcal{NP}$-hard.

Now, the Travelling Salesman Problem, defined in the introduction is not $\mathcal{NP}$-complete. This might look a bit strange at first glance, but then, we remember that all $\mathcal{NP}$-complete problems are so-called decision problems - problems deciding whether something is true or not.

It is not hard to see that the TSP does belong in $\mathcal{NP}$-hard, using my previous $\mathcal{NP}$ complexity class definition. Also, for the formal proof, Karp TODO showed in 1972. that the Hamiltonian cycle problem is $\mathcal{NP}$-complete, which implies the $\mathcal{NP}$-hardness of the TSP. But, surprisingly?, it does not belong to $\mathcal{NP}$. In the verification problem, we would be given a graph and one of cycles from that graph, and we are to judge whether that exact cycle is the shortest one. Note that we can not decide that without actually solving the TSP and knowing what the optimal cycle's cost is, so this can not be done in polynomial time knowing the fact we can not solve TSP in that time. So, this definition of the TSP is not in $\mathcal{NP}$, making it not in $\mathcal{NP}$-complete as well.

The thing that is a bit strange in the above conclusion is the fact that most of the famous 'hard' problems are $\mathcal{NP}$-complete as well. But, all those problems are decision problems, unlike our first TSP definition (and pretty much the only sensible definition of this problem). We can artificially make TSP a decision problem, by formulating it as: given a graph and a number, decide whether the cost of the shortest cycle is less than the number. This, rephrased TSP, is indeed in $\mathcal{NP}$-complete - the verification is straightforwardly done in polynomial time.

Conclusion to all this is that we definitely need and that it is actaully sensible to use some alternate problem solving algorithms than just the exact one - we need to apply some heuristics.

Chapter 3

Implementation

3.1 Verbatim text

Verbatim text can be included using `\begin{verbatim}` and `\end{verbatim}`. I normally use a slightly smaller font and often squeeze the lines a little closer together, as in:

```
GET "libhdr"

GLOBAL { count:200; all  }

LET try(ld, row, rd) BE TEST row=all
                        THEN count := count + 1
                        ELSE { LET poss = all & ~(ld | row | rd)
                               UNTIL poss=0 DO
                               { LET p = poss & -poss
                                 poss := poss - p
                                 try(ld+p << 1, row+p, rd+p >> 1)
                               }
                             }
LET start() = VALOF
{ all := 1
  FOR i = 1 TO 12 DO
  { count := 0
    try(0, 0, 0)
    writef("Number of solutions to %i2-queens is %i5*n", i, count)
    all := 2*all + 1
  }
  RESULTIS 0
}
```

3.2 Tables

Here is a simple example[1] of a table.

Left Justified	Centred	Right Justified
First	A	XXX
Second	AA	XX
Last	AAA	X

There is another example table in the proforma.

3.3 Simple diagrams

Simple diagrams can be written directly in LaTeX. For example, see figure 3.1 on page 9 and see figure 3.2 on page 9.

3.4 Adding more complicated graphics

The use of LaTeX format can be tedious and it is often better to use encapsulated postscript to represent complicated graphics. Figure 3.3 and 3.5 on page 11 are examples. The second figure was drawn using `xfig` and exported in `.eps` format. This is my recommended way of drawing all diagrams.

[1] A footnote

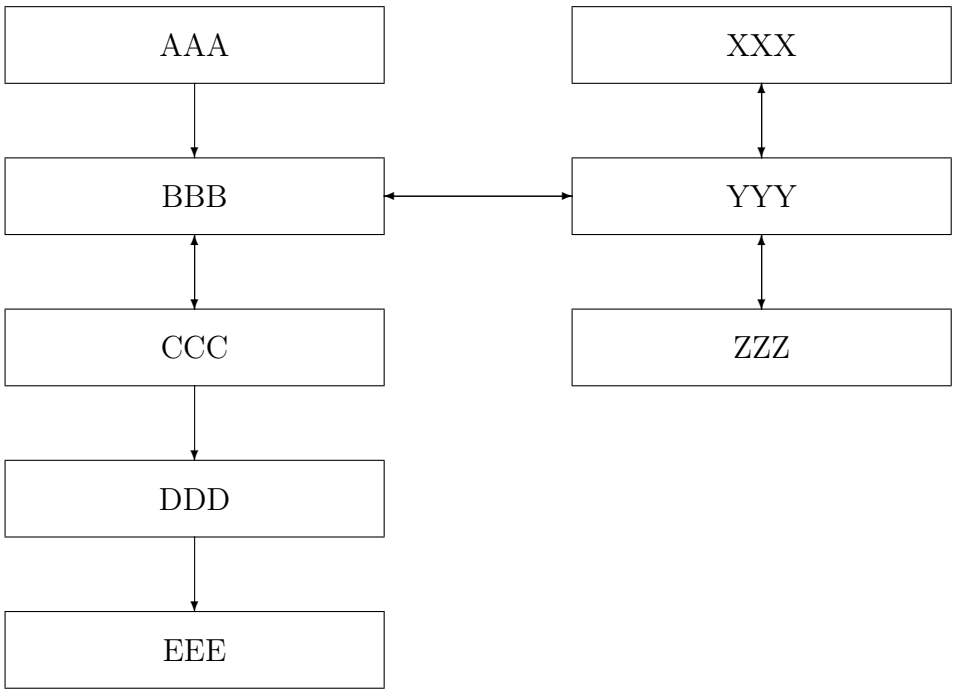

Figure 3.1: A picture composed of boxes and vectors.

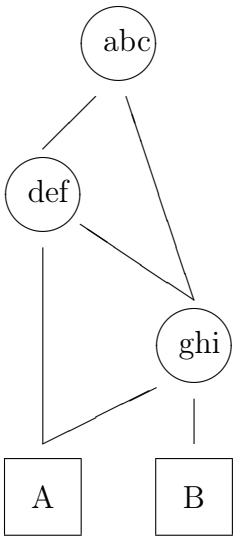

Figure 3.2: A diagram composed of circles, lines and boxes.

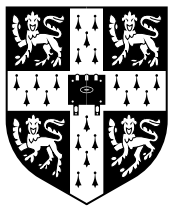

Figure 3.3: Example figure using encapsulated postscript

Figure 3.4: Example figure where a picture can be pasted in

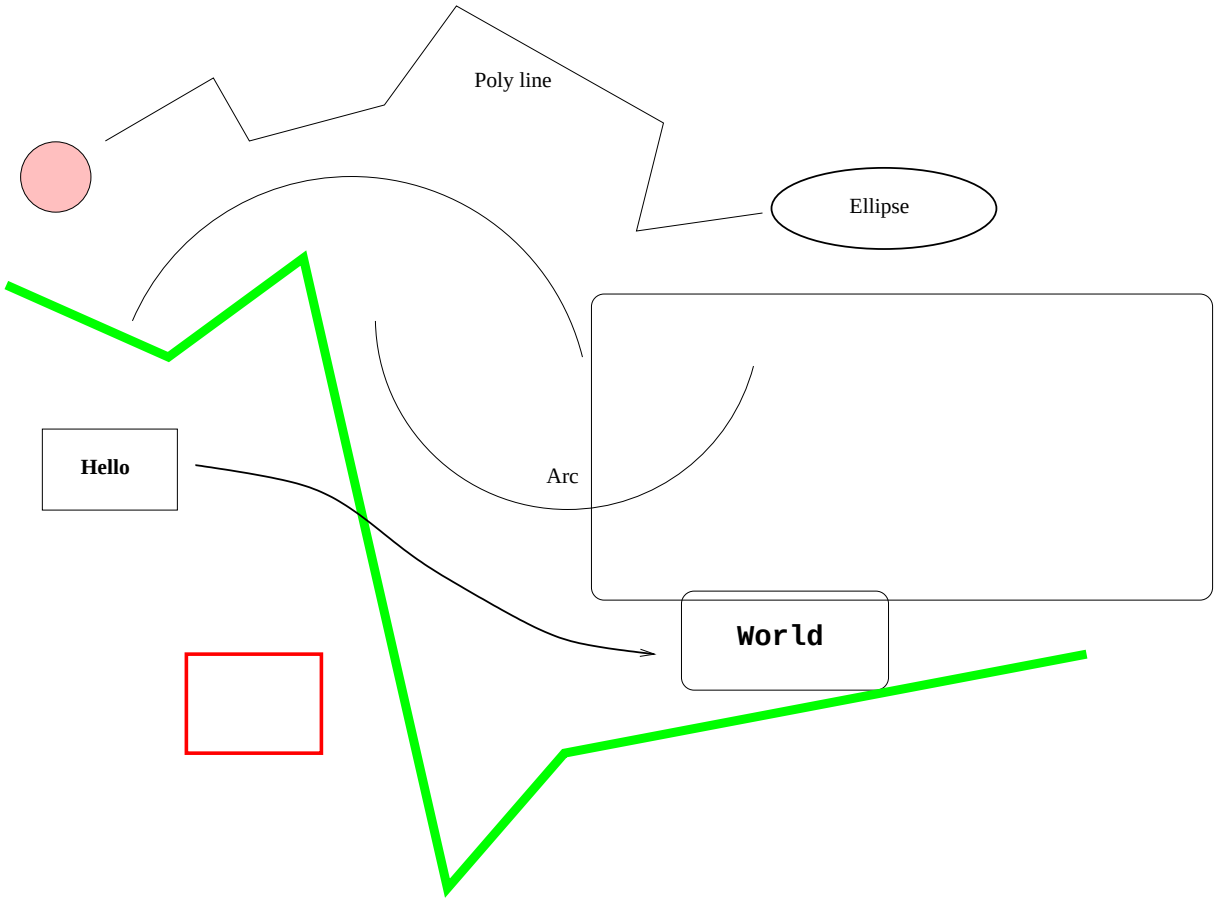

Figure 3.5: Example diagram drawn using `xfig`

Chapter 4

Evaluation

4.1 Printing and binding

If you have access to a laser printer that can print on two sides, you can use it to print two copies of your dissertation and then get them bound by the Computer Laboratory Bookshop. Otherwise, print your dissertation single sided and get the Bookshop to copy and bind it double sided.

Better printing quality can sometimes be obtained by giving the Bookshop an MSDOS 1.44 Mbyte 3.5" floppy disc containing the Postscript form of your dissertation. If the file is too large a compressed version with `zip` but not `gnuzip` nor `compress` is acceptable. However they prefer the uncompressed form if possible. From my experience I do not recommend this method.

4.1.1 Things to note

- Ensure that there are the correct number of blank pages inserted so that each double sided page has a front and a back. So, for example, the title page must be followed by an absolutely blank page (not even a page number).
- Submitted postscript introduces more potential problems. Therefore you must either allow two iterations of the binding process (once in a digital form, falling back to a second, paper, submission if necessary) or submit both paper and electronic versions.
- There may be unexpected problems with fonts.

4.2 Further information

See the Computer Lab's world wide web pages at URL:
`http://www.cl.cam.ac.uk/TeXdoc/TeXdocs.html`

Chapter 5

Conclusion

I hope that this rough guide to writing a dissertation is LaTeX has been helpful and saved you time.

Bibliography

[1] S.W. Moore. How to prepare a dissertation in latex, 1995.

Appendix A

Latex source

A.1 diss.tex

```
\documentclass[12pt,twoside,notitlepage]{report}

\usepackage{a4}
\usepackage{verbatim}
\usepackage[ampersand]{easylist}
\usepackage{algorithm}
\usepackage{hyperref}
\usepackage{algpseudocode}
\usepackage{amsbsy}
\usepackage{amsfonts}
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{amsthm}
\usepackage{booktabs}
\usepackage{caption}
\usepackage{color}
\usepackage{cleveref}
\usepackage{empheq}
\usepackage{enumitem}
\usepackage{etex}
\usepackage{fancyhdr}
\usepackage{float}
\usepackage{footnote}
\usepackage{framed}
\usepackage[T1]{fontenc}
\usepackage[UKenglish]{isodate}
\usepackage{latexsym}
\usepackage{leftidx}
\usepackage{listings}
\usepackage{mathrsfs}
\usepackage{mathtools}
\usepackage{multicol}
\usepackage{pgfplots}
\usepackage{soul}
\usepackage{subfig}
\usepackage{tikz}
\usepackage[nottoc]{tocbibind}
```

```
\input{epsf}                            % to allow postscript inclusions
% On thor and CUS read top of file:
%     /opt/TeX/lib/texmf/tex/dvips/epsf.sty
% On CL machines read:
%     /usr/lib/tex/macros/dvips/epsf.tex



\raggedbottom                           % try to avoid widows and orphans
\sloppy
\clubpenalty1000%
\widowpenalty1000%

\addtolength{\oddsidemargin}{6mm}       % adjust margins
\addtolength{\evensidemargin}{-8mm}

\renewcommand{\baselinestretch}{1.1}    % adjust line spacing to make
                                        % more readable

\begin{document}

\bibliographystyle{plain}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title


\pagestyle{empty}

\hfill{\LARGE \bf Vladimir Milenkovi\'c}

\vspace*{60mm}
\begin{center}
\Huge
{\bf The Travelling Salesman problem - Comparison of different approaches } \\
\vspace*{5mm}
Diploma in Computer Science \\
\vspace*{5mm}
Trinity College \\
\vspace*{5mm}
\today  % today's date
\end{center}

\cleardoublepage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Proforma, table of contents and list of figures

\setcounter{page}{1}
\pagenumbering{roman}
\pagestyle{plain}

\chapter*{Proforma}

{\large
\begin{tabular}{ll}
Name:               & \bf Vladimir Milenkovi\'c                  \\
```

```
College:            & \bf Trinity College                    \\
Project Title:      & \bf The Travelling Salesman problem - \\
& \bf Comparison of different approaches \\
Examination:        & \bf Diploma in Computer Science, May 2020       \\
Word Count:         & \bf TODO\footnotemark[1] \\
Project Originator: & \bf Dr T. Sauerwald and V. Milenkovic                   \\
Supervisor:         & \bf Dr T. Sauerwald                    \\
\end{tabular}
}
\footnotetext[1]{This word count was computed
by {\tt detex diss.tex | tr -cd '0-9A-Za-z $\tt\backslash$n' | wc -w}
}
\stepcounter{footnote}


\section*{Original Aims of the Project}

To code, compare and contrast different algorithms and approaches in solving one of the most famous problems ever to


\section*{Work Completed}

All work that was done during this project will be mentioned in the dissertation, the code will be submitted as well

\section*{Special Difficulties}

Getting Lin-Kernighan algorithm to work significantly better than the other algorithms, reproducing the results ment

\newpage
\section*{Declaration}

I, Vladimir Milenkovic of Trinity College, being a candidate for Part II of the Computer
Science Tripos , hereby declare
that this dissertation and the work described in it are my own work,
unaided except as may be specified below, and that the dissertation
does not contain material that has already been used to any substantial
extent for a comparable purpose.

\bigskip
\leftline{Signed Vladimir Milenkovic}

\medskip
\leftline{Date} % TODO

\cleardoublepage

\tableofcontents

\listoffigures

\newpage
\section*{Acknowledgements}

Add some acknowledgements here. Please, do add some acknowledgements here.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now for the chapters

\cleardoublepage        % just to make sure before the page numbering
```

```
                        % is changed

\setcounter{page}{1}
\pagenumbering{arabic}
\pagestyle{headings}

\chapter{Introduction}

\section{Motivation}

My project tries to tackle one of the most famous problems in the history of Computer science - the Travelling Salesman pr

\section{Algorithms}

In this section, I will list all the algorithms that I have taken into consideration for this project. I have made this ch

\begin{itemize}

\item {\bf Exact algorithms}
\item {\bf Approximation algorithms}
\item {\bf Improvement algorithms}
\item {\bf Heuristic algorithms}
\item {\tt Optimization algorithms}
\end{itemize}

\subsection{Exact algorithms}

{\bf Exact algorithms} are those algorithms which produce the correct solution every time they are run. Knowing the $\math

\begin{itemize}

\item {\it Brute-force algorithm}
\item {\it Held-Karp (dynamic programming) algorithm}
\item {\it Branch-and-Bound algorithm}

\end{itemize}

\subsection{Approximation algorithms}

{\bf Approximation algorithms} are ones that can guarantee some bound concerning the output of the algorithm. For example,

\begin{itemize}

\item {\it 2-approximation algorithm}
\item {\it Christofides-greedy algorithm}
\item {\it Christofides algorithm}

\end{itemize}

\subsection{Improvement algorithms}

{\bf Improvement algorithms} are those algorithm which consecutively try to improve an existing solution. Starting with so

\begin{itemize}

\item {\it 2-opt algorithm}
\item {\it 3-opt algorithm}
% TODO \item {\it 4-opt algorithm}
\item {\it Lin-Kernighan algorithm}
```

```
\end{itemize}

\subsection{Heuristic algorithms}

{\bf Heuristic algorithms} are algorithms which usually have polynomial running time, usually do not have a guarante

\begin{itemize}

\item {\it Random algorithm}
\item {\it Nearest neighbour algorithm (greedy)}
\item {\it Insertion heuristic - cheapest, farthest, nearest, random}
\item {\it Convex-hull heuristic algorithm}

\end{itemize}

\subsection{Optimization algorithms}

In {\bf optimization algorithms}, we are starting with random solutions, computing the results those solutions are p

\begin{itemize}

\item {\it Ant-colony optimization algorithm}
\item {\it Simulated annealing algorithm}

\end{itemize}

\cleardoublepage



\chapter{Preparation}

In this chapter, I will describe, in full detail, all the algorithms and all the theoretical background needed to st

\section{Complexity analysis}

I have already mentioned the fact that the Travelling Salesman Problem can not be solved in polynomial time. Now, we

\begin{itemize}

\item {\bf $\mathcal{P}$} - the set of all problems which can be correctly solved in polynomial time.
\item {\bf $\mathcal{NP}$} - the set of all problems whose {\it solutions} can be {\it verified} in polynomial time.
\item {\bf $\mathcal{NP}$-hard} - all the problems which can't be solved in polynomial time. This definition is very
\item {\bf $\mathcal{NP}$-complete} - problems which are both in $\mathcal{NP}$ and $\mathcal{NP}$-hard.

\end{itemize}

Now, the Travelling Salesman Problem, defined in the introduction is not $\mathcal{NP}$-complete. This might look a

\smallskip

It is not hard to see that the TSP does belong in $\mathcal{NP}$-hard, using my previous $\mathcal{NP}$ complexity c

\smallskip

The thing that is a bit strange in the above conclusion is the fact that most of the famous 'hard' problems are $\ma

\medskip
```

```
Conclusion to all this is that we definitely need and that it is actaully sensible to use some alternate problem solving a

\cleardoublepage
\chapter{Implementation}

\section{Verbatim text}

Verbatim text can be included using \verb|\begin{verbatim}| and
\verb|\end{verbatim}|. I normally use a slightly smaller font and
often squeeze the lines a little closer together, as in:

{\renewcommand{\baselinestretch}{0.8}\small\begin{verbatim}
GET "libhdr"

GLOBAL { count:200; all  }

LET try(ld, row, rd) BE TEST row=all
                        THEN count := count + 1
                        ELSE { LET poss = all & ~(ld | row | rd)
                               UNTIL poss=0 DO
                               { LET p = poss & -poss
                                 poss := poss - p
                                 try(ld+p << 1, row+p, rd+p >> 1)
                               }
                             }
LET start() = VALOF
{ all := 1
  FOR i = 1 TO 12 DO
  { count := 0
    try(0, 0, 0)
    writef("Number of solutions to %i2-queens is %i5*n", i, count)
    all := 2*all + 1
  }
  RESULTIS 0
}
\end{verbatim}
}

\section{Tables}

\begin{samepage}
Here is a simple example\footnote{A footnote} of a table.

\begin{center}
\begin{tabular}{l|c|r}
Left      & Centred & Right \\
Justified &         & Justified \\[3mm]
%\hline\\%[-2mm]
First     & A       & XXX \\
Second    & AA      & XX  \\
Last      & AAA     & X   \\
\end{tabular}
\end{center}

\noindent
There is another example table in the proforma.
\end{samepage}
```

```
\section{Simple diagrams}

Simple diagrams can be written directly in \LaTeX.  For example, see
figure~\ref{latexpic1} on page~\pageref{latexpic1} and see
figure~\ref{latexpic2} on page~\pageref{latexpic2}.

\begin{figure}
\setlength{\unitlength}{1mm}
\begin{center}
\begin{picture}(125,100)
\put(0,80){\framebox(50,10){AAA}}
\put(0,60){\framebox(50,10){BBB}}
\put(0,40){\framebox(50,10){CCC}}
\put(0,20){\framebox(50,10){DDD}}
\put(0,00){\framebox(50,10){EEE}}

\put(75,80){\framebox(50,10){XXX}}
\put(75,60){\framebox(50,10){YYY}}
\put(75,40){\framebox(50,10){ZZZ}}

\put(25,80){\vector(0,-1){10}}
\put(25,60){\vector(0,-1){10}}
\put(25,50){\vector(0,1){10}}
\put(25,40){\vector(0,-1){10}}
\put(25,20){\vector(0,-1){10}}

\put(100,80){\vector(0,-1){10}}
\put(100,70){\vector(0,1){10}}
\put(100,60){\vector(0,-1){10}}
\put(100,50){\vector(0,1){10}}

\put(50,65){\vector(1,0){25}}
\put(75,65){\vector(-1,0){25}}
\end{picture}
\end{center}
\caption{\label{latexpic1}A picture composed of boxes and vectors.}
\end{figure}

\begin{figure}
\setlength{\unitlength}{1mm}
\begin{center}

\begin{picture}(100,70)
\put(47,65){\circle{10}}
\put(45,64){abc}

\put(37,45){\circle{10}}
\put(37,51){\line(1,1){7}}
\put(35,44){def}

\put(57,25){\circle{10}}
\put(57,31){\line(-1,3){9}}
\put(57,31){\line(-3,2){15}}
\put(55,24){ghi}

\put(32,0){\framebox(10,10){A}}
\put(52,0){\framebox(10,10){B}}
\put(37,12){\line(0,1){26}}
\put(37,12){\line(2,1){15}}
```

```
\put(57,12){\line(0,2){6}}
\end{picture}

\end{center}
\caption{\label{latexpic2}A diagram composed of circles, lines and boxes.}
\end{figure}



\section{Adding more complicated graphics}

The use of \LaTeX\ format can be tedious and it is often better to use
encapsulated postscript to represent complicated graphics.
Figure~\ref{epsfig} and ~\ref{xfig} on page \pageref{xfig} are
examples. The second figure was drawn using {\tt xfig} and exported in
{\tt.eps} format. This is my recommended way of drawing all diagrams.


\begin{figure}[tbh]
\centerline{\epsfbox{figs/cuarms.eps}}
\caption{\label{epsfig}Example figure using encapsulated postscript}
\end{figure}

\begin{figure}[tbh]
\vspace{4in}
\caption{\label{pastedfig}Example figure where a picture can be pasted in}
\end{figure}


\begin{figure}[tbh]
\centerline{\epsfbox{figs/diagram.eps}}
\caption{\label{xfig}Example diagram drawn using {\tt xfig}}
\end{figure}




\cleardoublepage
\chapter{Evaluation}

\section{Printing and binding}

If you have access to a laser printer that can print on two sides, you
can use it to print two copies of your dissertation and then get them
bound by the Computer Laboratory Bookshop. Otherwise, print your
dissertation single sided and get the Bookshop to copy and bind it double
sided.


Better printing quality can sometimes be obtained by giving the
Bookshop an MSDOS 1.44~Mbyte 3.5" floppy disc containing the
Postscript form of your dissertation. If the file is too large a
compressed version with {\tt zip} but not {\tt gnuzip} nor {\tt
compress} is acceptable. However they prefer the uncompressed form if
possible. From my experience I do not recommend this method.

\subsection{Things to note}

\begin{itemize}
```

```
\item Ensure that there are the correct number of blank pages inserted
so that each double sided page has a front and a back.  So, for
example, the title page must be followed by an absolutely blank page
(not even a page number).

\item Submitted postscript introduces more potential problems.
Therefore you must either allow two iterations of the binding process
(once in a digital form, falling back to a second, paper, submission if
necessary) or submit both paper and electronic versions.

\item There may be unexpected problems with fonts.

\end{itemize}

\section{Further information}

See the Computer Lab's world wide web pages at URL:

{\tt http://www.cl.cam.ac.uk/TeXdoc/TeXdocs.html}


\cleardoublepage
\chapter{Conclusion}

I hope that this rough guide to writing a dissertation is \LaTeX\ has
been helpful and saved you time.




\cleardoublepage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the bibliography

\addcontentsline{toc}{chapter}{Bibliography}
\nocite*{\bibliography{refs}}
\cleardoublepage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the appendices
\appendix

\chapter{Latex source}

\section{diss.tex}
{\scriptsize\verbatiminput{diss.tex}}

\section{proposal.tex}
{\scriptsize\verbatiminput{proposal.tex}}

\section{propbody.tex}
{\scriptsize\verbatiminput{propbody.tex}}



\cleardoublepage

\chapter{Makefile}
```

```
\section{\label{makefile}Makefile}
{\scriptsize\verbatiminput{makefile.txt}}

\section{refs.bib}
{\scriptsize\verbatiminput{refs.bib}}


\cleardoublepage

\chapter{Project Proposal}

\input{temp_propbody}

\end{document}
```

A.2 proposal.tex

```
% This is a LaTeX driving document to produce a standalone copy
% of the project proposal held in propbody.tex.  Notice that
% propbody can be used in this context as well as being incorporated
% in the dissertation (see diss.tex).

\documentstyle[12pt,a4]{article}
\begin{document}

\include{propbody}

\end{document}
```

A.3 propbody.tex

```
% Draft #1 (final?)

\vfil

\centerline{\Large Diploma in Computer Science Project Proposal}
\vspace{0.4in}
\centerline{\Large How to write a dissertation in \LaTeX\ }
\vspace{0.4in}
\centerline{\large M. Richards, St John's College}
\vspace{0.3in}
\centerline{\large Originator: Dr M. Richards}
\vspace{0.3in}
\centerline{\large 21 November 2000}

\vfil

\subsection*{Special Resources Required}
File space on Thor -- 25Mbytes\\
Account on the DEC Workstations -- 15Mbytes\\
An account on Ouse\\
```

```
The use of my own IBM PC (1000GHz Pentium, 200Mb RAM and 40Gb Disk).
\vspace{0.2in}

\noindent
{\bf Project Supervisor:} Dr M. Richards
\vspace{0.2in}

\noindent
{\bf Director of Studies:} Dr M. Richards
\vspace{0.2in}
\noindent

\noindent
{\bf Project Overseers:} Dr~F.~H.~King  \& Dr~S.~W.~Moore

\vfil
\pagebreak

% Main document

\section*{Introduction}

Many students write their CST and Diploma dissertations in \LaTeX\ and
spend a fair amount of time learning just how to do that. The purpos of
this project is to write a demonsatration dissertation that explains in
detail how it done and how the result can be given to the Bookshop
on an MSDOS floppy disk for printing and binding.

\section*{Work that has to be done}

The project breaks down into the following main sections:-

\begin{enumerate}

\item The construction of a skeleton dissertation with the required
structure. This involves writing the Makefile and makeing dummy files
for the title page, the proforma, chapters 1 to 5, the appendices and
the proposal.

\item Filling in the details required in the cover page and proforma.

\item Writing the contents of chapters 1 to 5, including examples
of common \LaTeX\ constructs.

\item Adding a example of how to use floating figures and encapsulated
postscript diagrams.

\end{enumerate}

\section*{Difficulties to Overcome}

The following main learning tasks will have to be undertaken before
the project can be started:

\begin{itemize}

\item To learn \LaTeX\ and its use on Thor.

\item To discover how to incorporate encapsulated postscript into
```

```
a \LaTeX\ document, and to find a suitable drawing package on Thor
to recommend.

\item To discover what format the Bookshop would like for the finished
dissertation, and how to deal with postscript files that are too
large to fit on a single floppy disk.

\end{itemize}



\section*{Starting Point}

I have a reasonable working knowledge of \LaTeX\ and have convenient
access to Thor using an IBM PC in my office. Writing MSDOS disks is no
problem.

\section*{Resources}

This project requires little file space so 25Mbytes of disk space on Thor
should be sufficient. I plan to use my own IBM PC to write floppy disks,
but could use the PWF PCs if my own machine breaks down.

Backup will be on floppy disks.

\section*{Work Plan}

Planned starting date is 01/12/2000.

\subsection*{Michaelmas Term}

By the end of this term I intend to have completed the learning tasks
outlined in the relevant section.


\subsection*{Lent Term}

By the division of term the overall structure of the dissertation
will have been written and tested.

By the end of term, example figures using encapsulated postscript
will have been included.


\subsection*{Easter Term}

On completion of the exams I will incorporate final details into
the dissertation including a bibliography using bibtex and a table of contents.
The estimated completion date being 25/07/2001 to allow
plenty of time should any unforeseen problems arise.
```

Appendix B

Makefile

B.1 Makefile

```
# This is the Makefile for the demonstration dissertation
# written by Martin Richards
#
# Note that continuation lines require '\'
# and that TAB is used after ':' and before unix commands.

DISS = diss.tex refs.bib propbody.tex figs/diagram.eps makefile.txt

PROP = proposal.tex propbody.tex

help:
        @echo
        @echo "USAGE:"
        @echo
        @echo "make          display help information"
        @echo "make prop     make the proposal and view it using xdvi"
        @echo "make diss.ps  make a postscript version of the dissertation"
        @echo "make diss.pdf make a .pdf version of the dissertation"
        @echo "make gv       view the dissertation with ghostview"
        @echo "make gs       view the dissertation with ghostscript"
        @echo "make all      construct proposal.dvi and diss.ps"
        @echo "make count    display an estimated word count"
        @echo "make pub      put demodiss.tar on my homepage"
        @echo "make clean    remove all remakeable files"
        @echo "make pr       print the dissertation"
        @echo

prop:   proposal.dvi
        xdvi proposal.dvi

diss.ps:        $(DISS)
        latex diss
        bibtex diss
        latex diss
        bibtex diss
        latex diss
        bibtex diss
```

```
        dvips -Ppdf -G0 -t a4 -pp 0-200 -o diss.ps diss.dvi

diss.pdf:       diss.ps
        ps2pdf diss.ps

makefile.txt:   Makefile
        expand Makefile >makefile.txt
count:
        detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w

proposal.dvi: $(PROP)
        latex proposal

all:    proposal.dvi diss.ps

pub:    diss.pdf
        cp diss.pdf /homes/mr/public_html/demodiss.pdf
        make clean
        (cd ..; tar cfv /homes/mr/public_html/demodiss.tar demodiss)

clean:
        rm -f diss.ps *.dvi *.aux *.log *.err
        rm -f core *~ *.lof *.toc *.blg *.bbl
        rm -f makefile.txt

gv:     diss.ps
        ghostview diss.ps

gs:     diss.ps
        gs diss.ps

pr:     diss.ps
        lpr diss.ps
```

B.2 refs.bib

```
@REPORT{Moore95,
TITLE = "How to prepare a dissertation in LaTeX",
AUTHOR = "Moore, S.W.",
YEAR = "1995"}
```

Appendix C

Project Proposal

Vladimir Milenković
Trinity College
vm370@cam.ac.uk

Individual project

Computer Science Tripos, Part II

The Travelling Salesman problem - Comparison of different approaches

---

18 October 2019

Project Originators:
Dr Thomas Sauerwald
Vladimir Milenković

Project Supervisor:
Dr Thomas Sauerwald

Director of Studies:
Prof Frank Stajano
Dr Sean Holden

Project Overseers:
Dr Rafal Mantiuk
Prof Andrew Pitts

Introduction and Description of the Work

The Travelling Salesman problem is one of the most famous NP-hard problems. It has been the topic of many researches, and many famous mathematicians and computer scientists tried to approach it. This problem is occurring frequently even in real life - for example, we are willing to visit multiple cities and we know what the flight cost between each pair of cities is, can we make an cost-optimal route visiting all of them? Also, a good solution to this problem would have several applications: in planning, logistics, microchip manufacturing, etc..

The mathematical formulation of the problem follows: given an undirected weighted graph, find the Hamiltonian cycle with minimum weight, where we define the weight of a cycle as a sum of weights of all the edges involved.

As this problem has been and still is one of the most major 'unsolved' problems in Computer Science, there has been many different tries to find a solution which is good by some metric. Some of the approaches that were taken with more or less success are:

- Exact algorithms: the most straight forward brute-force approach can be done in $\mathcal{O}(n!)$ by simply trying all the possible permutations. There is a dynamic programming approach in $\mathcal{O}(2^n n^2)$, done by Held–Karp. There is no known algorithm which solves the TSP in $\mathcal{O}\big((2-\varepsilon)^n\big)$.
- Approximation algorithms: these are the algorithms that don't search for the optimal solution, but they find the solution that is guaranteed to be, for example, at most 2 times worse that the optimal one. Approximation algorithms nowadays can usually find the solution in 2-3% to the optimal solution within reasonable time. FPTAS (Fully Polynomial-Time Approximation Scheme) is one of the algorithms that I will implement.
- Heuristic algorithms: In this type of algorithms, there are no guarantees about the solution we're going to find whatsoever, we are trying to follow some 'sensible' heuristics in order to make our solution as good as possible.
- Genetic algorithms: which are the randomized algorithms in which we are simulating some processes observed in natural evolution.
- Neural networks: There are many papers about trying to approach the TSP using neural networks. There has been some success, mostly using

recurrent neural networks, but right now they are not outperforming other algorithms.

In this project, I will be implementing atleast 2 algorihms per each category stated above and comparing them to each other on different types of graphs. I will mainly focus on comparing their solution correctness, and comparing their running time. My language of choice of implementing the algorithm is C++, due to my already substantial experience with it. Also, one of the reasons is that C++ is performance efficient, thus my programs will be running a bit faster compared to some other options I could've taken. Also, in the neural network part of the project, I will probably be using Python, because of the existence of various machine learning libraries, such as TensorFlow/Keras.

Relevant courses

Concerning the courses I have already and will be taking at my Computer Science Tripos, here are the ones that my project will interfere with:

Algorithms Main part of this project is going to be actually coding the algorithms solving the problem. Both my previous algorithm experience and knowledges I've gotten during my Part IA Algorithms course will come in handy.

Artificial Intelligence I/II Since I'll be coding some heuristic algorithms, as well as doing some machine learning, things I've learned are going to be influential for the project.

Programming in C/C++ My main choice of language for this project is going to be C++.

Complexity Theory The problem this whole project is about is NP-hard, so a proof of that is going to be in the project.

Project Structure

This project intends to produce benchmarks and results about various algorithms and their behaviour on substantially different inputs.

The execution of the project can be split into 5 main objectives. In the project timetable given below, I will write down the times at which I'm planning to finish each of them. Objectives, in chronological order, can be found below:

- **Preparation** – this part of the project mainly consists of getting more familiar with the topic. My plan, in order to successfully finish this objective, is to read books and papers about algorithms solving the TSP and thinking about ways to implement them, trying to pick the most concise and the most effective implementation. To confirm that I've successfully finished this chapter, I'm planning to be able to prove the correctness, the complexity and to be completely familiar with every algorithm I'm going to implement in the project.

- **Implementation** – this part of the project consists of actually implementing the algorithms, as well as the generators of the data I'm going to test on and all the other necessary code in order to be able to efficiently evaluate the algorithms I've coded. By generators of the data, I mean both collecting the already existing data from the web (e.g. flight durations, road lengths, etc..), as well as generating my own random data with some tunable parameters (e.g. graph density).

- **Evaluation** – this objective consists of evaluating the algorithms on different examples, plotting the graphs of different characteristics I'm going to measure. For example: running time, space complexity, deviation from the optimal solution (if known), impact of varying algorithm parameters (if suitable). Here, I'm planning to use some of the well-known TSP datasets available online, in order to test my implementation versus already existing ones. One of the datasets I'm planning to use is the one available at University of Waterloo TSP page.

- **Extensions** – this objective consists of me trying to give my own optimizations of certain algorithms on different types of graphs, in order to get a better performance. One more thing I could add to this project is to try to solve it using linear programming, and to benchmark that approach versus already existing ones. I have several ideas about observing how do heuristic approaches behave when altering some parameters in the actual heuristic being used and plotting the outcomes. I'm planning to start doing Extensions and trying to achieve something new after completely finishing the core project.

- **Dissertation** – this part of the project consists of collecting all the code involved in the project, as well as writing the dissertation. In the dissertation, I would like to cleanly show how all of the above objectives have been accomplished in a chronological order.

Success Criteria

I will consider my project as a successfully completed one after achieving all of the above objectives apart from **Extensions** one. Several checkpoints I should accomplish at that point are:

- Well written and easily readable source code for all the algorithms should be provided at the end of the project, as well as the empirical validation that all the algorithms performed close to as one could predict based on the already existing knowledge.
- Scripts needed to compile and benchmark the project.
- Plots about algorithms performance, after being evaluated multiple times on various test samples, both generated and downloaded.
- The dissertation, in which my process of accomplishing all the objectives should be clearly explained, as well as the conclusions I've achieved after being able to evaluate different approaches and some directions for future work on this topic.

I hope that **Extensions** objective will also be achieved before the project submission deadline, however it is not the core part of the project and shouldn't be essential for its success.

Resources

I'm planning to use my own laptop (Dell XPS 9570, 16 GB RAM, Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, running GNU/Linux). I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure. My contingency plans are listed below:

- I will be using GitHub as the projects main backup tool, creating a private repository and storing all the data necessary for the project, including the dissertation, in that repository.

- I will also backup all the data at my PC on daily basis, on my own external HDD.

I'm pretty sure that no other resources are required in order to complete my project.

Project timetable

My high level milestones to be achieved are: getting prepared and studying for the project until the end of November, actually implementing the project until the end of January, finishing the evaluation part by the end of February, and writing the main part of the dissertation by the end of March. If everything works as expected, I'm planning to to my Extensions objective in April, trying further to improve the project. I expect the submission by the end of the April, so I don't get too close to the deadline, which is a situation in which nobody would like to be.

Concerning a bit more detailed timetable, I will divide the time from today until the end of the project in 15 two-weeks periods, and I will list the small objectives I will aim to achieve in each of them.

Part 1: *18 October – 31 November*

→ Writing and refining the project proposal.

- **Deadline:** Project proposal (25 October)

→ Creating the GitHub repository and setuping the software and hardware needed to achieve my contingency plans.

Part 2: *1 November – 14 November*

→ Getting more familiar with heuristic algorithms in general.

→ Reading the books and the papers about existing TSP algorithms and deeply studying the subject.

Part 3: *15 November – 28 November*

→ Efficient implementation and testing of exact TSP algorithm.

→ Coding approximation algorithms (Christofides' algorithm being one of them).

Part 4: *29 November – 12 December*

→ Testing the approximation algorithms

→ Starting to work on different heuristic approaches.

Part 5: *13 December – 26 December*

→ Further working on heuristic algorithm. I want to make sure that all the parameters of the heuristic are going to be easily tunable, in order for Extensions to be done without much problem.

Part 6: *27 December – 8 January*

→ Testing heuristic algorithms.

→ Reading more papers about machine learning approach to TSP, here focusing mainly on the implementation aspects.

Part 7: *9 January – 22 January*

→ Implementing some of the papers with the idea from above.

→ Testing and being able to produce the same results as in the paper.

Part 8: *23 January – 4 February*

- **DEADLINE:** Progress report submission (31 January)

→ Buffer slot to finish any unfinished work from before.

→ Writing the project progress report and getting ready for the progress report presentation.

Part 9: *5 February – 18 February*

→ Writing the generators for the data I'm going to test my various algorithms on.

→ Writing the code that's going to link my algorithms with the generators, allowing easy testing and evaluation part.

Part 10: *19 February – 4 March*

→ At this point, if everything above is done correctly, the process of evaluating and producing the plots shouldn't be too big of a deal. Generating suitable plots for the dissertation.

→ Finishing if any of previous objectives isn't already achieved.

Part 11: *5 March – 18 March*

→ Starting to work on the project dissertation, writing the introduction chapter.

Part 12: *19 March – 1 April*

→ First draft of the dissertation.

Part 13: *2 April – 15 April*

→ Further refining the dissertation, listening to the comments from both my supervisor and DOSes.

→ Start working on Extensions topics.

Part 14: *16 April – 29 April*

→ Finishing my dissertation and making it ready for submission.

→ Further working on Extensions, and incorporating it in the dissertation if successful.

Milestone: Final version of the dissertation ready for submission.

Part 15: *30 April – 8 May*

→ Dissertation submission.

- **DEADLINE:** Dissertation (8 May)